Notes on Machine Learning

Tong Jia

cecilio.jia@gmail.com

October 26, 2021

Individual Research Organization, Beijing

"This is a clever quote or perhaps a dedication."

Personal Research Internship, Beijing
github.com/Cecilio-Jia

Content

Part I

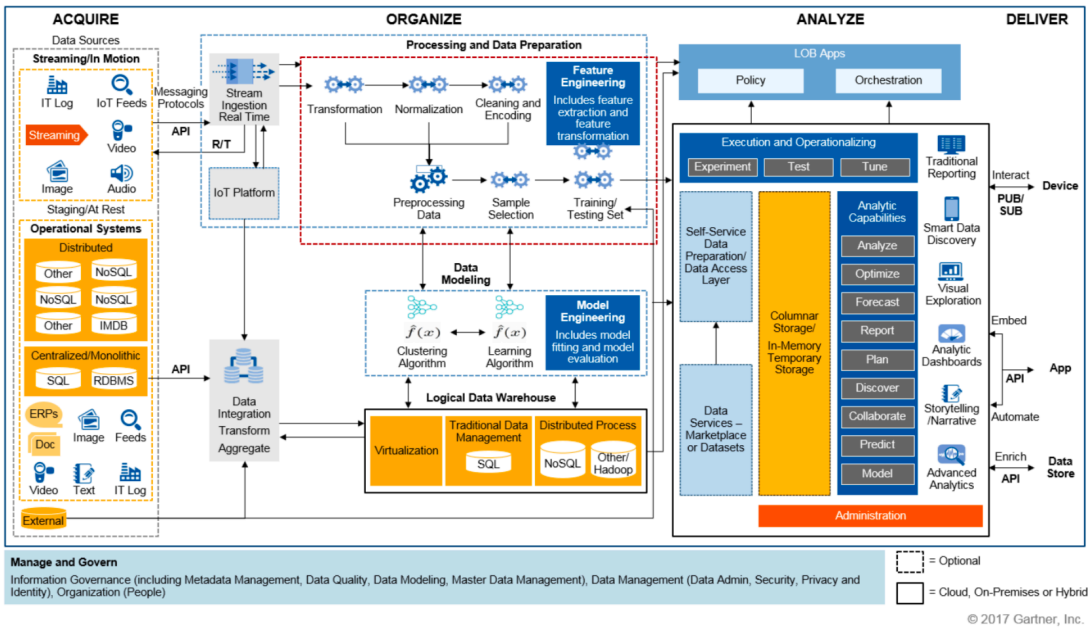
Machine Learning Basic

1 Introduction

1.1 Machine Learning System

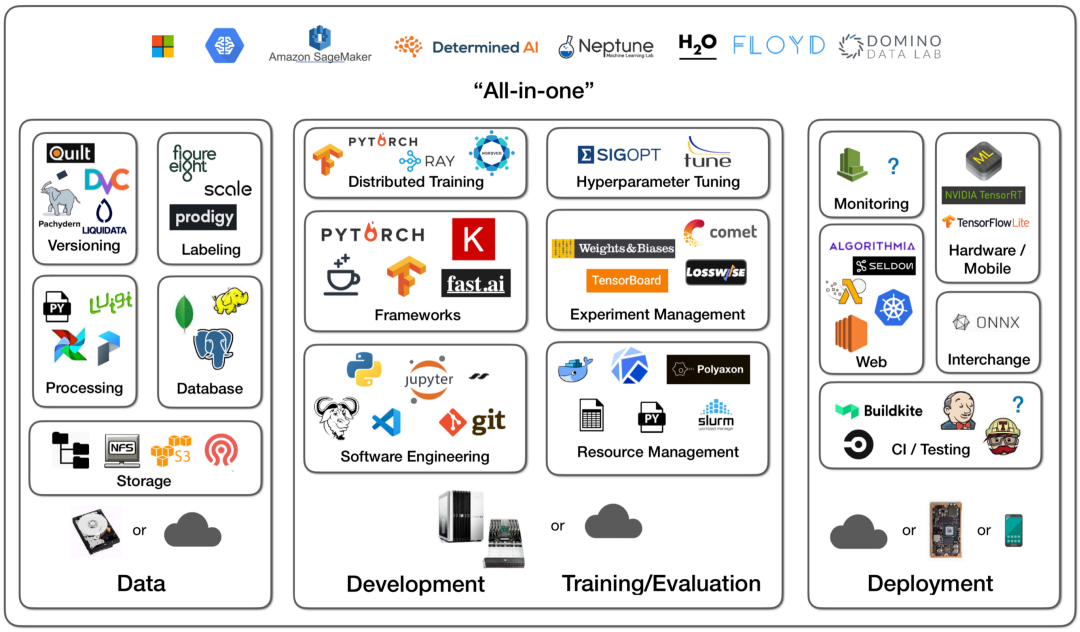
1.1.1 Architecture of End-to-End Machine learning System

完整的端到端机器学习系统平台架构如下：



- Acquisition Module:
  - API Sub-Module:
    - ERP(Enterprise Resource Planning) Databases API
    - IoT(Internet of things) Devices API
    - App(Mobile application) Databases API
  - Data Ingestion Sub-Module
- Organization Module
- Analysis Module
- Delivery Module

1.1.2 Ecosystem of Machine Learning Production



Sergey Karayev at Full Stack Deep Learning Bootcamp 2019, https://fullstackdeeplearning.com/

图 1: Full stack of deep learning. Image Source: Production Level Deep Learning

Reference

- A Guide to Production Level Deep Learning

1.2 Framework of Machine Learning

- 完整的机器学习项目：

  1. 特征框架
  2. 训练框架
  3. 服务框架
  4. 评估框架
  5. 监控框架

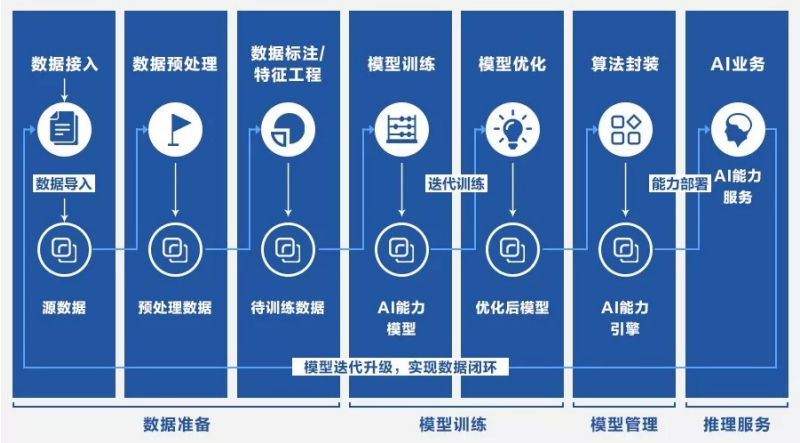


图 2: Megvii Brain++ AI Algorithm Platform.

- 机器学习训练”五部曲”：

  1. **Dataset**: $\mathcal{D} = \{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$
  2. **Formulation**: $f(\boldsymbol{x}; \boldsymbol{\theta})$
  3. **Objective**: $Obj(\boldsymbol{\theta}) = \underbrace{\mathcal{L}(\boldsymbol{\theta})}_{\text{Training loss}} + \underbrace{\Omega(\boldsymbol{\theta})}_{\text{Regularization}}$
  4. **Optimization**: SGD, Adam...
  5. **Evaluation**: 算法指标，商业指标，用户体验...

1.3 Framework of Model in Practice

1. 数据采集
2. 数据处理
3. 特征选择
4. 样本划分
5. 模型设计
6. 模型离线训练
7. 模型评估
8. 参数调优
9. 模型对比
10. 模型上线
11. 线上评估 (A/B Test)

1.4 Large-Scale Machine Learning Training System

1.4.1 Reference

- Carnegie Mellon University: Scheduling For Efficient Large-Scale Machine Learning Training

2 Dataset

2.1 Dataset Representation

本书中对数据符号（notations）表示作如下统一：

- N: 训练数据集的样本个数
- d: 任意样本的特征维度
- K: 离散型概率分布的类别个数
- $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$: 第 i 个训练样本的特征向量
- $x_j^{(i)} \in \mathbb{R}$: 第 i 个训练样本的第 j 个维度
- $y^{(i)} \in \mathbb{R}$: 第 i 个训练样本的标签
- $\widehat{y}^{(i)} \in \mathbb{R}$: 第 i 个训练样本的估计
- $\boldsymbol{y}^{(i)} \in (0,1)^K$: 第 i 个训练样本的标签概率分布
- $\widehat{\boldsymbol{y}}^{(i)} \in (0,1)^K$: 第 i 个训练样本的估计概率分布
- $y_k^{(i)} \in \mathbb{R}$: 第 i 个训练样本第 k 类概率标签
- $\widehat{y}_k^{(i)} \in \mathbb{R}$: 第 i 个训练样本第 k 类概率估计
- $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^N$: 训练数据集

对数据操作符号作如下统一：

- $\cdot$：向量内积（结果为一个标量）
- $\odot$：向量对应位相乘（结果为一个向量）
- $\oplus$: 向量的拼接（concatenation）

2.2 Independent Variable

2.2.1 Category features

One-hot encoding

- 类别型属性本质是字符串
- 只有数值才能被计算机（算法）识别
- One-hot 编码方式将每一个 field 的字符串分类 ⇒ 离散型 field 值概率分布

原始数据：

性别	国籍
男	中国
女	美国
男	新加坡

独热编码之后的数据：

性别 = 男	性别 = 女	国籍 = 中国	国籍 = 美国	国籍 = 新加坡
1	0	1	0	0
0	1	0	1	0
1	0	0	0	1

2.2.2 Numerical features

Discretization

问题：Why discretizing a numerical feature into categorical features ? 在工业界，很少直接将连续型特征 (numerical features) 直接作为 Logistic Regression 模型的输入特征，而是将某一连续的特征离散化为一系列 0/1 特征再交给 Logistic Regression 模型，优势如下：

- **对异常数据有强鲁棒性**。例如：age=300
- **模型更加稳定**。例如：对用户的年龄进行离散化，20-30 作为一个区间，不会因为一个用户的年龄长了一岁就变成一个完全不同的人。当然处于区间相邻处的样本刚好相反，所以如何划分区间是门学问，一般有几种：
  - 专家经验
  - 数据分位点
  - **信息增益最大化**
- **引入非线性变换**。逻辑回归属于广义的线性模型，因此表达能力受到了限制，某一连续型特征离散化为 N 个 0-1 特征后，每个 0-1 特征有单独的权重，相当于为模型引入了非线性，能够提升模型的表达能力，加大拟合程度。
- **便于之后进行特征交叉**。离散化后可以进行特征交叉 (feature interactions)，进一步引入非线性，提升模型表达能力。
- **降低了模型过拟合的风险**。
- **离散特征的增加和减少都很容易，易于模型的快速迭代**。
- **稀疏向量内积乘法速度快，计算结果方便存储，容易扩展**。

Discretization algorithms 参考综述性论文A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning[?]

2.3 Dependent Variable

2.3.1 Imbalanced dataset in classification task

1. 正样本**重采样（Oversampling）**
2. 负样本**下采样（Undersampling）**
3. 正样本上的 **SMOTE**[?] 算法

Oversampling 以 Logistic Regression 为例，说明 oversample 对模型的影响：

$$
\begin{aligned}
\because z &= \sum_{j=1}^{d} w_j x_j + b \\
\phi(z) &= \frac{1}{1+\exp(-z)} \\
\therefore \phi(\boldsymbol{x}\,;\boldsymbol{w},b) &= \frac{1}{1+\exp(-\sum\limits_{i=1}^{d} w_j x_j - b)}
\end{aligned}
$$

loss function 如下：

$$
J(\boldsymbol{w},b) = \underbrace{-\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}\log\widehat{y}^{(i)} + (1-y^{(i)})\log(1-\widehat{y}^{(i)})\right)}_{\text{training loss}} + \underbrace{\frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}}_{\text{l2 regularization}}
$$

因此模型系数（slope）和截距（intercept）的负梯度如下：

$$
\begin{aligned}
-\frac{\partial J(\boldsymbol{w},b\,;\boldsymbol{x}^{(i)},y^{(i)})}{\partial w_j} &= \frac{\partial}{\partial w_j}\left(y^{(i)}\log\widehat{y}^{(i)} + (1-y^{(i)})\log(1-\widehat{y}^{(i)})\right) - \frac{\partial}{\partial w_j}\left(\frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}\right) \\
&= \frac{y^{(i)}}{\widehat{y}^{(i)}}\frac{\partial\widehat{y}^{(i)}}{\partial w_j} + \frac{1-y^{(i)}}{1-\widehat{y}^{(i)}}(-1)\frac{\partial\widehat{y}^{(i)}}{\partial w_j} - \lambda w_j \\
&= \frac{\widehat{y}^{(i)}(y^{(i)}-1) + y^{(i)}(1-\widehat{y}^{(i)})}{\widehat{y}^{(i)}(1-\widehat{y}^{(i)})}\frac{\partial\widehat{y}^{(i)}}{\partial w_j} - \lambda w_j \\
\because \frac{\partial\widehat{y}^{(i)}}{\partial w_j} &= \frac{\partial\phi(\boldsymbol{x}^{(i)}\,;\boldsymbol{w},b)}{\partial w_j} = \frac{\partial\phi(z)}{\partial z}\frac{\partial z}{\partial w_j} \\
&= \big(1-\phi(z)\big)\,\phi(z)\,x_j^{(i)} = \big(1-\widehat{y}^{(i)}\big)\widehat{y}^{(i)}\,x_j^{(i)} \\
\therefore -\frac{\partial J(\boldsymbol{w},b\,;\boldsymbol{x}^{(i)},y^{(i)})}{\partial w_j} &= \frac{\widehat{y}^{(i)}y^{(i)} - \widehat{y}^{(i)} + y^{(i)} - \widehat{y}^{(i)}y^{(i)}}{\widehat{y}^{(i)}(1-\widehat{y}^{(i)})}\big(1-\widehat{y}^{(i)}\big)\,\widehat{y}^{(i)}\,x_j^{(i)} - \lambda w_j \\
&= (y^{(i)} - \widehat{y}^{(i)})x_j^{(i)} - \lambda w_j
\end{aligned}
$$

$$
\begin{aligned}
-\frac{\partial J(\boldsymbol{w},b\,;\boldsymbol{x}^{(i)},y^{(i)})}{\partial b} &= \frac{\partial}{\partial b}\left(y^{(i)}\log\widehat{y}^{(i)} + (1-y^{(i)})\log(1-\widehat{y}^{(i)})\right) - \underbrace{\frac{\partial}{\partial b}\left(\frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}\right)}_{=0} \\
&= \frac{y^{(i)}}{\widehat{y}^{(i)}}\frac{\partial\widehat{y}^{(i)}}{\partial b} + \frac{1-y^{(i)}}{1-\widehat{y}^{(i)}}(-1)\frac{\partial\widehat{y}^{(i)}}{\partial b} \\
&= \frac{\widehat{y}^{(i)}(y^{(i)}-1) + y^{(i)}(1-\widehat{y}^{(i)})}{\widehat{y}^{(i)}(1-\widehat{y}^{(i)})}\frac{\partial\widehat{y}^{(i)}}{\partial b}
\end{aligned}
$$

$$
\begin{aligned}
\because \frac{\partial \widehat{y}^{(i)}}{\partial b} &= \frac{\partial \phi(\boldsymbol{x}^{(i)};\boldsymbol{w},b)}{\partial b} = \frac{\partial \phi(z)}{\partial z}\frac{\partial z}{\partial b} \\
&= \big(1-\phi(z)\big)\,\phi(z)\cdot 1 = \big(1-\widehat{y}^{(i)}\big)\widehat{y}^{(i)} \\
\therefore -\frac{\partial J(\boldsymbol{w},b;\boldsymbol{x}^{(i)},y^{(i)})}{\partial b} &= \frac{\widehat{y}^{(i)}y^{(i)} - \widehat{y}^{(i)} + y^{(i)} - \widehat{y}^{(i)}y^{(i)}}{\widehat{y}^{(i)}(1-\widehat{y}^{(i)})}\big(1-\widehat{y}^{(i)}\big)\,\widehat{y}^{(i)} \\
&= y^{(i)} - \widehat{y}^{(i)}
\end{aligned}
$$

因此（以 SGD 为例）参数的更新公式为：

$$
\begin{aligned}
w_j &\leftarrow w_j + \eta\Big[(y^{(i)} - \widehat{y}^{(i)})x_j^{(i)} - \lambda w_j\Big] \\
b &\leftarrow b + \eta(y^{(i)} - \widehat{y}^{(i)})
\end{aligned}
$$

oversampling 对模型的影响总结如下：

- **Oversampling 不影响模型的系数（slope），但是会放大模型的截距（intercept）**（因为当 oversample 正样本时，正样本数量增多，参数 b 的更新会使得其数值越来越大 $\because \hat{y}^{(i)} < 1$）
- 因为截距放大了，**预测事件的概率也都变大了**
- 评价准则中，Sensitivity $= \frac{\mathrm{TP}}{\mathrm{TP+FN}}$（真实为正，预测为正）和 Specificity $= \frac{\mathrm{TN}}{\mathrm{TN+FP}}$（真实为负，预测为负）不受影响，但是 False Positive Rate FPR $= \frac{\mathrm{FP}}{\mathrm{FP+TN}}$（真实为负，预测为正）和 Fales Negative Rate FNR $= \frac{\mathrm{FN}}{\mathrm{FN+TP}}$（真实为正，预测为负）会受影响
- ROC curve 不受影响

oversampling 之后如何修正？

- 假设：P_0 为对某个样本预测为 0（负类）的概率（在 oversample 后的训练样本上学习后），$P_1 = 1 - P_0$ 为预测为 1（正类）的概率
- 令：$A = \frac{P_1}{\frac{\text{Oversample后target 为 1 的训练样本比例}}{\text{Oversample前target 为 1 的训练样本比例}}}$，$B = \frac{P_0}{\frac{\text{Oversample后target 为 0 的训练样本比例}}{\text{Oversample前target 为 0 的训练样本比例}}}$
- 修正后的概率为：$P_1 = \frac{\mathrm{A}}{\mathrm{A+B}}$，$P_0 = \frac{\mathrm{B}}{\mathrm{A+B}}$

SMOTE 算法具体做法如下：

1. 找到一个正样本 $\boldsymbol{x}_1$（少数类别的样本）
2. 根据 KNN 计算找出 $\boldsymbol{x}_i$ 最近邻的 k 个**正样本**（该例中 k=5，找到的近邻正样本为 $\boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6$）
3. 在 $\boldsymbol{x}_1$ 和选出的近邻正样本的线段上生成新的正样本（凸组合），作为新的正样本，对原始数据集进行扩充

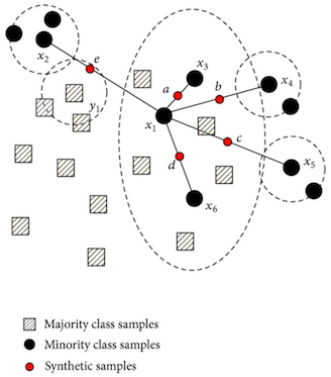


图 3: Visualization of SMOTE

优劣对比

- 正样本上的 Oversampling 从**很小的正样本池**中引入重复样本，因此容易导致模型过拟合；
- 负样本上的 Undersampling 可能会漏洞在两个类中提供关键差异信息的重要样本，因此容易导致模型效果不佳；
- SMOTE 利用正样本的信息合成了新的正样本，而不是重复已有的正样本，因此在**一定程度**上缓解了过拟合，但是这依旧不能缓解所有的过拟合问题，因为它的生成还是来自于已有的正样本。

2.4 Dataset Preprocessing

2.4.1 Image preprocessing

2.4.2 Text preprocessing

NLP 模型任务（以英文任务为例）基本工作流程如下：

1. 文本获取（Raw text data）
2. 分词（Segmentation）
3. 清洗（Cleaning）
   - 去除标点符号
   - 英文大写转换为小写
   - 数字归一化
   - 停用词库 & 低频词库
   - ...
4. 标准化（Normalization）
   - 词形还原（Lemmatization）（**转变** e.g. driving ⇒ drive, drove ⇒ drive）
   - 词干提取（Stemming）（**缩减** e.g. cats ⇒ cat, effective ⇒ effect）
5. 特征提取（Feature extraction）
   - TF-IDF
   - Word2Vec
   - CountVectorizer
6. 建模（Modeling）

2.5 Analysis of Dataset（未完成）

2.5.1 Significance Analysis

2.5.2 Correlation Analysis

2.5.3 Principal Component Analysis (PCA)

主成分分析（Principal Component Analysis, PCA）是一种常用的**无监督数据降维**方法，使得原始高维空间 $\mathcal{X}$ 内的**高维特征向量集** $\mathcal{D}_{\mathcal{X}}=\left\{\boldsymbol{x}^{(i)}\in\mathbb{R}^{\mathcal{X}}\right\}_{i=1}^{N}$ 在转换 $\mathcal{X}\to\mathcal{Z},\ |\mathcal{X}|>|\mathcal{Z}|$ 后的低维空间 $\mathcal{Z}$ 中**低维特征向量集** $\mathcal{D}_{\mathcal{Z}}=\left\{\boldsymbol{z}^{(i)}\in\mathbb{R}^{\mathcal{Z}}\right\}_{i=1}^{N}$ 的方差最大。

Methodology 原始高维空间 $\mathcal{X}$ 内样本集的均值（中心点）为：

$$\bar{\boldsymbol{x}}=\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}^{(i)}$$

投影空间 $\mathcal{Z}$ 内所有样本的方差为：

$$\begin{aligned}\sigma(\mathcal{D}_{\mathcal{Z}})&=\frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{z}^{(i)}-\bar{\boldsymbol{z}}\right)^2\\&=\frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{w}\cdot\boldsymbol{x}^{(i)}-\boldsymbol{w}\cdot\bar{\boldsymbol{x}}\right)^2\end{aligned}\quad\text{（考虑特例：投影空间 }\mathcal{Z}\text{ 为一维空间，}z=\boldsymbol{w}\cdot\boldsymbol{x}\text{）}$$

2.5.4 Linear Discriminant Analysis (LDA)

2.6 Summary of Dataset

- 模型对数据的使用，其实是两种模式的权衡：
  - **海量离散特征** + **简单模型** (Logistic Regression)
  - **少量连续特征** + **复杂模型** (XGBoost)

  也就是离散化的特征加线性模型，或者数值型特征加深度学习。二者的选择就是看喜欢折腾特征还是折腾模型了。通常而言，前者容易，而且可以 n 个人一起并行做，有成功经验，后者目前看着很赞，但是能走多远还拭目以待。
- **数据决定了结果的上限 (天花板)，而模型只决定距离这个"天花板"还有多远**。

3 Objective Function

3.1 Expected risk & Empirical risk

3.1.1 Expected risk minimization

期望风险（Expected risk）是对输入（input: $x \in \mathbb{R}^{\mathcal{X}}$）和输出（output: $y \in \mathbb{R}^{\mathcal{Y}}$）联合分布空间 $\mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}}$ 上对目标函数的度量，记联合概率分布（joint probability distribution）为：

$$\mathcal{P} : \mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}} \to \mathbb{P}$$

那么期望风险目标函数 $R(h) : \mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}} \to \mathbb{R}$ 为：

$$\begin{aligned} R(h_\theta) &= \mathbb{E}\left[\mathcal{L}\left(h_\theta(x), y\right)\right] \\ &= \int_{\mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}}} \mathcal{L}\left(h_\theta(x), y\right) d\mathcal{P}(x, y) \end{aligned}$$

其中 θ 为 hypothesis function: $h : \mathcal{X} \to \mathcal{Y}$ 的参数，而最终得到的最优 hypothesis function h^* 为全部假设空间 $\mathcal{H}$ 内，使得期望风险 $R(h)$ 最小的那个：

$$h^* = \arg\min_{h \in \mathcal{H}} R(h)$$

3.1.2 Empirical risk minimization

通常，由于分布 $\mathcal{P}(x, y)$ 未知，所以导致期望风险 $R(h)$ 不能够被直接计算，进而导致最优假设函数 h^* 不可得。然而我们可以通过对训练集 $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ 上的损失函数取平均值来计算近似值，称为经验风险（Empirical risk）$R_{\text{emp}}(h) : \mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}} \to \mathbb{R}$：

$$R_{\text{emp}}(h_\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(h_\theta(x^{(i)}), y^{(i)})$$

其中训练集 $\mathcal{D}$ 中的所有样本来自于：**独立同分布假设下，在样本联合分布空间 $\mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{Y}}$ 内，基于联合概率分布 $\mathcal{P}(x, y)$ 的 n 次有放回采样**。则最优的假设函数 h^* 为：

$$h^* = \arg\min_{h \in \mathcal{H}} R_{\text{emp}}(h)$$

3.2 Loss Function for Regression

- input feature vector: $x \in \mathbb{R}^{\mathcal{X}}$
- label of output scalar: $y \in \mathbb{R}$
- hypothesis function(output a predicted real-value scalar): $h_\theta : \mathbb{R}^{\mathcal{X}} \to \mathbb{R}$

3.2.1 Mean squared error

$$\min_\theta \ J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - h_\theta(x^{(i)}) \right)^2$$

3.2.2 Mean absolute error

$$\min_\theta \ J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left| y^{(i)} - h_\theta(x^{(i)}) \right|$$

3.2.3 Huber loss

Definition of Huber loss is as follow:

$$\mathcal{L}_\delta(y, h_\theta(x)) = \begin{cases} \frac{1}{2}(y - h_\theta(x))^2 & \text{if } |y - h_\theta(x)| \leqslant \delta, \\ \delta \left| y - h_\theta(x) \right| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

and the empirical risk with huber loss is as follow:

$$\min_\theta \ J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_\delta(y^{(i)}, h_\theta(x^{(i)}))$$

Comparison among Squared loss, Absolute loss and Huber loss:

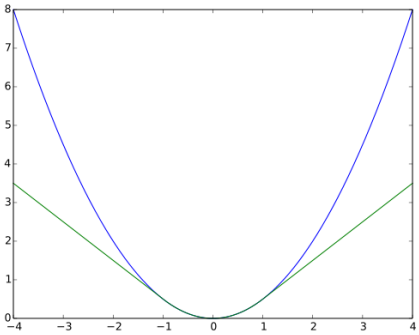


图 4: Huber loss (green $\delta = 1$) and squared error loss (blue) as a function of $y - h_\theta(x)$.

- Squared loss:

  - 优点：最常用的损失函数，对非异常点情况下，存在较大误差的点有较好的关注。
  - 缺点：**会对异常点（outlier）**(x, y) **施以较大的惩罚**，即当异常点很多时，目标函数过于关注对异常点的惩罚，从而导致模型 h_θ 会出现比较大偏差，因此**不够鲁棒（robust）**。

- Absolute loss:
  - 优点：当存在较多异常点（outlier）时，MAE 的表现较好。
  - 缺点：在 $y - h_\theta(x) = 0$ 处时不可导（虽然一般优化任务中不存在训练残差为 0 的情况）。

- Huber loss:
  - 优点：比较**Robust（对异常值不敏感）**的损失函数，同时在 $y - h_\theta(x) = 0$ 处可导。

3.3 Loss Function for Classification

3.3.1 Hinge loss

Definition of Hinge loss is as follow:

$$\mathcal{L}(y, h_\theta(x)) = \max\Big(0, 1 - y \cdot h_\theta(x)\Big)$$

so:

$$\min_\theta \ J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \max\Big(0, 1 - y^{(i)} \cdot h_\theta(x^{(i)})\Big)$$

where：

- $y \in \{-1, +1\}$ is label output
- $h_\theta(x) \in (-1, +1)$ is predicted output

含义：

- 正确分类时，y 与 $h_\theta(x)$ 同号，此时的损失值域 $\mathcal{L} \in [0, 1)$
- 错误分类时，y 与 $h_\theta(x)$ 异号，此时的损失值域 $\mathcal{L} \in [0, +\infty)$
- 考虑预测类别和真实类别相同（预测正确）的情况:
  - $y = +1$ 且 $h_\theta(x) = +100000$ 时，预测类别和真实类别相同
  - 且此时 $1 - y \cdot h_\theta(x) = 1 - 1 \times 100000 = -99999$，即理论上应该得到的奖励为 $-(1 - y \cdot h_\theta(x)) = 99999$
  - 但是此时 loss 为 $\mathcal{L} = \max\Big(0, 1 - y \cdot h_\theta(x)\Big) = 0$，即实际上得到的奖励为 $-\mathcal{L} = 0$，即此时不会得到任何奖励。

  因此，**Hinge loss 不鼓励模型 h_θ 过分自信**，即预测输出 $h_\theta(x) \in (-1, +1)$ 就可以了，不用 $|h_\theta(x)| > 1$.
- 考虑预测类别和真实类别不同（预测错误）的情况:
  - $y = +1$ 且 $h_\theta(x) = -100000$ 时，预测类别和真实类别相反
  - 且此时 $1 - y \cdot h_\theta(x) = 1 - 1 \times (-100000) = 100001$，即理论上应该得到的奖励为 $-(1 - y \cdot h_\theta(x)) = -100001$
  - 此时 loss 为 $\mathcal{L} = \max\Big(0, 1 - y \cdot h_\theta(x)\Big) = 100001$，即实际上得到的奖励为 $-\mathcal{L} = -100001$，惩罚很大，因此需要着重针对该样本进行参数学习，努力将其分类正确。

  因此，Hinge loss 更加关注分类错误样本，而不太关注分类正确样本的学习，所以可能导致的效果是**maximum-margin**，即：
  - 正样本和负样本之间的 $h_\theta(x)$ 差异非常明显
  - 正样本和正样本之间的 $h_\theta(x)$ 差异不明显
  - 负样本和负样本之间的 $h_\theta(x)$ 差异不明显

3.3.2 Log-likelihood

- input feature vector: $x \in \mathbb{R}^{\mathcal{X}}$
- label of output: $y \in \mathcal{Y} = \{1, ..., K\}$, K is total number of classes.
- label of output probability distribution: $p \in \mathbb{P}^{\mathcal{Y}}$
- hypothesis function(output a predicted probability distribution over all classes): $h_\theta : \mathbb{R}^{\mathcal{X}} \to \mathbb{P}^{\mathcal{Y}}$

基于样本独立同分布的假设，最大化所有用于训练的观测样本 $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ 发生的概率，即可通过训练得到样本输入特征向量 x 与输出标签概率分布 y 之间的真实（近似真实）关系：

$$
\begin{aligned}
& \max \ P(\mathcal{D}) \\
\Rightarrow \ & \max \ \prod_{i=1}^{N} p(x^{(i)}, y^{(i)}) \quad (\because \text{i.i.d}) \\
\Rightarrow \ & \max \ \prod_{i=1}^{N} p(y^{(i)}|x^{(i)}) p(x^{(i)}) \quad (\because \ p(x,y) = p(y\,|x) p(x)) \\
\Rightarrow \ & \max \ \frac{1}{N} \prod_{i=1}^{N} p(y^{(i)}|x^{(i)}) \quad (\because \ p(x^{(i)}) \approx \frac{1}{N}) \\
\Rightarrow \ & \max \ \frac{1}{N} \log \left(\prod_{i=1}^{N} p(y^{(i)}|x^{(i)}) \right) \\
\Rightarrow \ & \max \ \frac{1}{N} \sum_{i=1}^{N} \log p(y^{(i)}|x^{(i)})
\end{aligned}
$$

3.3.3 Kullback–Leibler divergence

Convert maximizing Log-likelihood into minimizing KL-divergence:

$$
\begin{aligned}
& \max \ \frac{1}{N} \sum_{i=1}^{N} \log p(y^{(i)}|x^{(i)}) \\
\Rightarrow \ & \min \ \frac{1}{N} \sum_{i=1}^{N} D_{\mathrm{KL}}(p^{(i)} \| h_\theta(x^{(i)})) \quad (i.e. \ y^{(i)} \sim p^{(i)} \in \mathbb{P}^{|K|}) \\
\Rightarrow \ & \min \ \frac{1}{N} \sum_{i=1}^{N} D_{\mathrm{KL}}(p^{(i)} \| q^{(i)}) \quad (i.e. \ q^{(i)} = h_\theta(x^{(i)}) \in \mathbb{P}^{\mathcal{Y}}) \\
\Rightarrow \ & \min \ \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log \frac{p_k^{(i)}}{q_k^{(i)}} \quad \text{(definition of Kullback–Leibler divergence)} \\
\Rightarrow \ & \min \ \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \left(p_k^{(i)} \log p_k^{(i)} - p_k^{(i)} \log q_k^{(i)} \right) \\
\Rightarrow \ & \min \ \underbrace{\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log p_k^{(i)}}_{\text{entropy}} \underbrace{- \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log q_k^{(i)}}_{\text{cross entropy}}
\end{aligned}
$$

$$\Rightarrow \min_{\theta} J(\theta) = \underbrace{\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_k^{(i)} \log p_k^{(i)}}_{\text{constant}} - \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_k^{(i)} \log \Big(h_\theta(x^{(i)})\Big)_k$$

Step-by-step understanding of KL-divergence:

- 信息量（Information）：

$$I = -\log p,\ p \in \mathbb{P}$$

表示一个概率事件的“震惊”程度。概率越小，信息量越大（e.g. 国足世界杯夺冠）。

图 5: 国足世界杯夺冠的概率 p 很小，因此信息量 $I = -\log p$ 很大

- 熵（Entropy）：

$$\mathcal{H}(p) = \sum_{k=1}^{K} p_k(-\log p_k) = -\sum_{k=1}^{K} p_k \log p_k$$

表示一个概率分布 p 的期望信息量。

- KL 散度（KL-divergence）：

$$\begin{aligned} D_{\mathrm{KL}}(p \,\|\, q) &= \sum_{k=1}^{K} p_k \log \frac{p_k}{q_k} \\ &= \sum_{k=1}^{K} (p_k \log p_k - p_k \log q_k) \\ &= \sum_{k=1}^{K} \Big(p_k(-\log q_k) - p_k(-\log p_k)\Big) \end{aligned}$$

表示两个概率分布之间的差异。

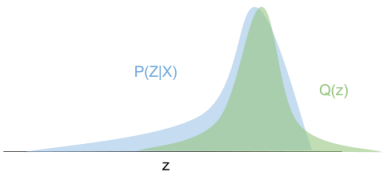


图 6: KL-Divergence 用于衡量两个概率分布之间的差异

KL-Divergence 中**含有训练参数 θ 的部分得在 q，常数部分在 p**，因为 KL 散度没有距离定义上的对称性，所以反过来无法优化计算。

3.3.4 Cross entropy

Definition of Cross entropy between the distributions $\boldsymbol{p}$ and $\boldsymbol{q}$ is as follow:

$$\mathcal{H}(\boldsymbol{p}, \boldsymbol{q}) = \mathbb{E}_p\left[-\log q\right] = -\sum_{k=1}^{K} p_k \log q_k = -\left\langle \boldsymbol{p}, \log \boldsymbol{q} \right\rangle$$

where the **inner product $< \cdot, \cdot >$ computes a similarity measure between the network prediction $\boldsymbol{q}$ and the corresponding data label $\boldsymbol{p}$**, and:

- $\boldsymbol{p} \in \mathbb{P}^{|K|}$: target probability distribution(**always 0-1 probability distribution**) over K classes
- $\boldsymbol{q} = h_\theta(x) \in \mathbb{P}^{|K|}$: predicted probability distribution over K classes:

$$h_\theta(x) = \left[\mathbb{P}(\text{class}(x) = 1), \quad \mathbb{P}(\text{class}(x) = 2), \quad ..., \quad \mathbb{P}(\text{class}(x) = K)\right].$$

Directly drop entropy term in KL-divergence, we can get empirical risk with cross entropy loss as follow:

$$\min \underbrace{\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_k^{(i)} \log p_k^{(i)}}_{\text{entropy}} \underbrace{-\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_k^{(i)} \log q_k^{(i)}}_{\text{cross entropy}}$$

$$\Rightarrow \min_\theta \; J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_k^{(i)} \log q_k^{(i)}$$

$$\Rightarrow \min_\theta \; J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_k^{(i)} \log \left[h_\theta(x^{(i)})\right]_k$$

$$\Rightarrow \min_\theta \; J(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \left\langle p^{(i)}, \log h_\theta(x^{(i)}) \right\rangle \quad \text{(inner product operation between vectors)}$$

3.3.5 Focal loss

Cross Entropy We introduce the focal loss[?] starting from the cross entropy (CE) loss for binary classification:

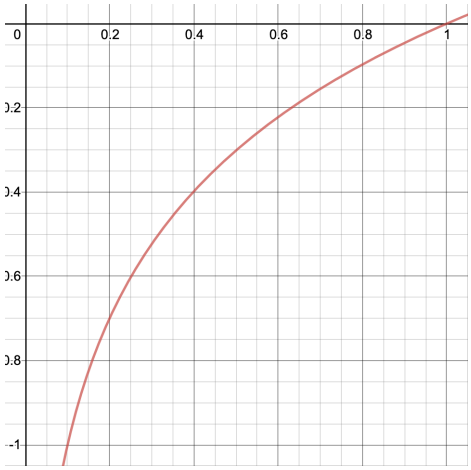
$$\begin{aligned}\mathrm{CE}(p,y) &= -\Big(y\log(p)+(1-y)\log(1-p)\Big)\\ &= \begin{cases} -\log(p) & \text{if } y=1,\\ -\log(1-p) & \text{otherwise.}\end{cases}\end{aligned}$$

where $y \in \{0,1\}$ specifies the ground-truth class and $p \in [0,1]$ is the model ' s estimated probability for the class with label $y=1$. For notational convenience, define the **probability of classification correctly**:

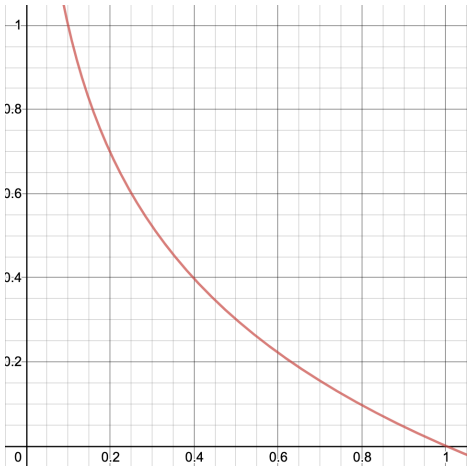
$$p_{\mathrm{t}} = \begin{cases} p & \text{if } y=1\\ 1-p & \text{otherwise.}\end{cases}$$

we can rewrite cross entropy(even for multi classification task) as follow:

$$\mathrm{CE}(p,y) = \mathrm{CE}(p_{\mathrm{t}}) = -\log(p_{\mathrm{t}})$$



(a) $\log(p_{\mathrm{t}})$ as a reward of correct classification.



(b) $-\log(p_{\mathrm{t}})$ as a loss of correct classification.

CE loss 存在的一个关键问题是：**样本集合几乎为易分类样本** $(p_{\mathrm{t}} \gg 0.5)$ **且非平衡样本** $(n_{\mathbf{pos}} \ll n_{\mathbf{neg}})$ **时，训练阶段对正样本的学习效果很差，进而导致预测阶段对正样本的预测准确率很低**。

- 假定所有的样本:

  $$\mathcal{D} = \left\{(x^{(i)}, y^{(i)}) | x^{(i)} \in \mathbb{R}^{\mathcal{X}}, y^{(i)} \in \{0,1\}\right\}_{i=1}^{N}$$

  都是易分类样本：

  $$p_{\mathrm{t}} \gg 0.5$$

- 那么当 p_{ture} 来自于一个相对较高的置信区间时 (e.g. $p_{\mathrm{t}} \in (0.7, 0.98)$)，样本集合 $\mathcal{D}$ 上的总 CE loss 均来自于易分类样本带来的不大也不小的损失 $\mathrm{CE}(p_{\mathrm{t}}) = -\log(p_{\mathrm{t}})$

- 但是对于易分类样本，我们并不知道其样本是正样本 $y^{(i)}=1$ 还是负样本 $y^{(i)}=0$，因此在**不平衡样本**情况下，样本集合 $\mathcal{D}$ 上的总 CE loss 几乎来自于负样本：

$$\begin{aligned}\sum_{i=1}^{N}-\log(p_{\mathrm{t}}^{(i)})&=\sum_{i=1}^{n_{\mathrm{neg}}}-\log(p_{\mathrm{t}}^{(i)})+\sum_{i=1}^{n_{\mathrm{pos}}}-\log(p_{\mathrm{t}}^{(i)})\quad(n_{\mathrm{neg}}+n_{\mathrm{pos}}=n)\\&\approx\sum_{i=1}^{n_{\mathrm{neg}}}-\log(p_{\mathrm{t}}^{(i)})\quad(\because\ n_{\mathrm{pos}}\ll n_{\mathrm{neg}})\end{aligned}$$

  因此，这就造成了将正样本 (少数类别样本) 的 CE loss 淹没在大海中，进而导致：

  1. 训练阶段：对负样本的学习严重冗余，而对**正样本的学习严重匮乏**
  2. 测试阶段：对负样本的预测准确率很高，而对**正样本的预测准确率很低**

Balanced Cross Entropy 因此为了平衡正负样本的 loss，引入了一个权重因子 (weight factor)：$\alpha\in[0,1]$，即：

$$\sum_{i=1}^{N}-\log(p_{\mathrm{t}}^{(i)})=\sum_{i=1}^{N_{\mathrm{neg}}}-(1-\alpha)\log(p_{\mathrm{t}}^{(i)})+\sum_{i=1}^{N_{\mathrm{pos}}}-\alpha\log(p_{\mathrm{t}}^{(i)})$$

为了简化表示，可以记为：

$$\alpha_{\mathrm{t}}=\begin{cases}\alpha\gg0.5 & \text{if } y=1\,(\text{positive sample}),\\ 1-\alpha\ll0.5 & \text{otherwise (negative sample).}\end{cases}$$

因此之前单一样本上的 CE loss:

$$\mathrm{CE}(p_{\mathrm{t}})=-\log(p_{\mathrm{t}})$$

经过**考虑正负样本的加权**后变为：

$$\mathrm{CE}(p_{\mathrm{t}})=-\alpha_{\mathrm{t}}\log(p_{\mathrm{t}})$$

Note: How to set α_{t}?

- inverse class frequency

$$\alpha_{\mathrm{t}}=\begin{cases}\frac{1}{N_{\mathrm{pos}}} & \text{if } y=1\,(\text{positive sample}),\\ \frac{1}{N_{\mathrm{neg}}} & \text{otherwise (negative sample).}\end{cases}$$

- reciprocal

$$\alpha_{\mathrm{t}}=\begin{cases}\frac{N_{\mathrm{neg}}}{N} & \text{if } y=1\,(\text{positive sample}),\\ \frac{N_{\mathrm{pos}}}{N} & \text{otherwise (negative sample).}\end{cases}$$

- hyperparameter to set by cross validation

Focal Loss Definition 很多任务（e.g. Dense Object Detection）的数据存在以下特点：

- 容易判别分类 ($p_{\mathrm{t}}\gg0.5$) 的负样本 ($y=0$) 占据了总 loss 的大部分，并主导了梯度
- 虽然 α_{t} 平衡了正负样本的重要性（balances the importance of positive/negative examples），但是其并没有让损失函数可以有效辨别划分难易样本（does not differentiate between easy/hard examples），进而更加关注难分的样本（hard example）

因此针对难分的样本，设置损失函数如下：

$$\mathrm{FL}(p_\mathrm{t}) = -(1-p_\mathrm{t})^\gamma \log(p_\mathrm{t})$$

其中 $\gamma \geq 0$ 为可调的超参数。该公式解释如下：

- 当样本很难分时:
  - $p_\mathrm{t} \to 0$ 很小
  - 该样本本身的损失为 $\mathrm{CE}(p_\mathrm{t}) = -\log(p_\mathrm{t})$ 大
  - 而因为难分，我们希望更加关注它，即给该样本设置更大的难易重要度（importance），因此 $(1-p_\mathrm{t})^\gamma \to 1$ 很大
- 当样本容易分时:
  - $p_\mathrm{t} \to 1$ 很大
  - 该样本本身的损失为 $\mathrm{CE}(p_\mathrm{t}) = -\log(p_\mathrm{t})$ 小
  - 而因为易分，我们希望相对更忽略它，即给该样本设置更小的难易重要度（importance），因此 $(1-p_\mathrm{t})^\gamma \to 0$ 很小

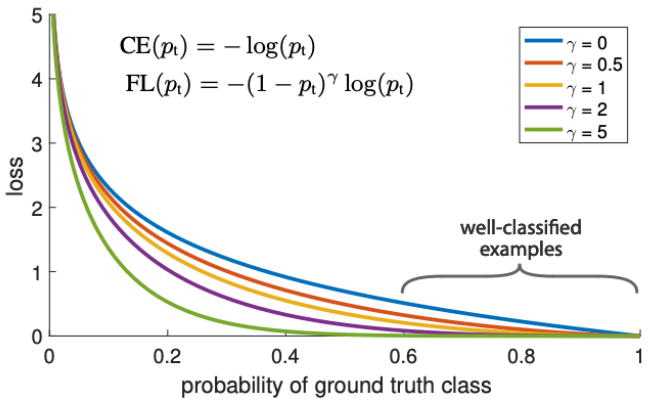


图 7: We propose a novel loss we term the Focal Loss that adds a factor $(1-p_\mathrm{t})^\gamma$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_\mathrm{t} > 0.5$), putting more focus on hard, misclassified examples.

Focal loss 相对于 CE loss 的优点是：在样本普遍易分类 ($p_\mathrm{t} > 0.5$) 的情况下，更加关注那些相对模棱两可 (e.g. $p_\mathrm{t} = 0.55$) 的样本，赋予其指数级别相对高的重要性权重。

考虑 Balanced Cross Entropy，因此最终 Focal Loss 为：

$$\mathrm{FL}(p_\mathrm{t}) = -\alpha_\mathrm{t}(1-p_\mathrm{t})^\gamma \log(p_\mathrm{t})$$

训练样本上的 Focal Loss 为：

$$\min_\theta \ J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\alpha_\mathrm{t}(1-p_\mathrm{t}^{(i)})^\gamma \log(p_\mathrm{t}^{(i)})$$

因此 Focal Loss 的适用场景是：**训练样本几乎均为易分类样本，且正负样本不平衡，即：普遍为相对易分类 ($p_t > 0.5$) 的负样本 ($y = 0$)。 Focal loss 相对于 CE loss 的缺点是**：极度不稳定，因此不建议直接适用 Focal Loss，最好是经过几个 epoch 之后再将 Cross Entropy 改为 Focal Loss，或者只是将 Focal Loss 作为 Fine-tune。

3.3.6 Gradient Harmonized Mechanism (GHM)

内容来自论文Gradient Harmonized Single-stage Detector[?]，该方法改进了Focal Loss[?] 中超参数 (α, γ) 需要耗费大量工作进行调参的难题。

Problem Description Consider the basic binary cross entropy loss：

$$\mathcal{L}_{\mathrm{CE}}(p^*, p_\theta(x)) = \begin{cases} -\log\big(p_\theta(x)\big) & \text{if } p^* = 1 \\ -\log\big(1 - p_\theta(x)\big) & \text{if } p^* = 0 \end{cases}$$

where:

- $p_\theta(x) \in [0, 1]$: probability predicted by the model
- $p^* \in \{0, 1\}$: ground-truth label

Note $z_\theta(x) \in \mathbb{R}$ as the logit otuput of model, therefore：

$$p_\theta(x) = \sigma(z_\theta(x)) \quad (\sigma(\cdot) \text{ means the sigmoid function.})$$

gradient of binary cross entropy loss with regard to $z_\theta(x)$:

$$\begin{aligned} \frac{\partial \mathcal{L}_{\mathrm{CE}}(p^*, p_\theta(x))}{\partial z_\theta(x)} &= \frac{\partial \mathcal{L}_{\mathrm{CE}}(p^*, p_\theta(x))}{\partial p_\theta(x)} \frac{\partial p_\theta(x)}{\partial z_\theta(x)} \\ \because \frac{\partial p_\theta(x)}{\partial z_\theta(x)} &= \frac{\partial}{\partial z_\theta(x)} \left(\frac{1}{1 + e^{-z_\theta(x)}} \right) \\ &= \big(1 - \sigma(z_\theta(x))\big)\sigma(z_\theta(x)) \\ &= (1 - p_\theta(x)) \cdot p_\theta(x) \\ \because \frac{\partial \mathcal{L}_{\mathrm{CE}}(p^*, p_\theta(x))}{\partial p_\theta(x)} &= \begin{cases} -\frac{1}{p_\theta(x)} & \text{if } p^* = 1 \\ \frac{1}{1 - p_\theta(x)} & \text{if } p^* = 0 \end{cases} \\ \therefore \frac{\partial \mathcal{L}_{\mathrm{CE}}(p^*, p_\theta(x))}{\partial z_\theta(x)} &= \begin{cases} p_\theta(x) - 1 & \text{if } p^* = 1 \\ p_\theta(x) & \text{if } p^* = 0 \end{cases} \\ &= p_\theta(x) - p^* \end{aligned}$$

Now we define gradient norm $g(x) \in [0, 1]$ as follow, which means the norm of gradient w.r.t $z_\theta(x)$:

$$g(x) = |p_\theta(x) - p^*| = \begin{cases} p^* - p_\theta(x) & \text{if } p^* = 1 \\ p_\theta(x) & \text{if } p^* = 0 \end{cases}$$

The value of $g(x)$ represents **attribute (e.g. easy or hard) of an example** and implies the example ' s impact on the global gradient.
很多现实分类任务(e.g. face detection.)存在和遇到的问题：**disharmony of gradient norm distribution**

- Easy examples:
  - the number of very easy examples(i.e. samples with very small $g(x)$) is extremely large, which have a great impact on the global gradient.
- Hard examples:

- a converged model $p_\theta(x)$ still can't handle some very hard examples(samples with large $g(x)$), whose number is even larger than the examples with medium difficulty(samples with medium $g(x)$).
- These very hard examples(samples with large $g(x)$) can be regarded as **outliers** since their gradient directions($p_\theta(x) - p^*$) tends to vary largely from the gradient directions of the large amount of other examples. That is, if the converged model is forced to learn to classify these outliers better, the classification of the large number of other examples tends to be less accurate.

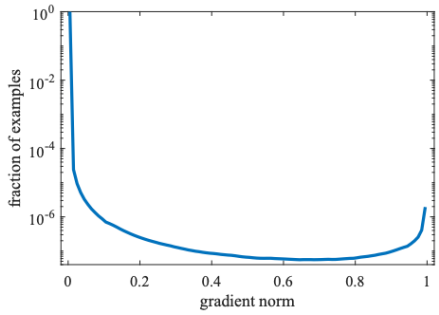


图 8: The distribution of the gradient norm $g(x)$ from a converged one-stage detection model. Note that the y-axis uses log scale since **easy negative samples have a dominant number**, and the number of examples with different gradient norm can differ by orders of magnitude. Cite from Gradient Harmonized Single-stage Detector[?]

Gradient Density

Definition 3.1 (**gradient density function**). 训练样本集上关于某一样本 (x, y) 的 gradient norm g 的梯度密度函数 (gradient density function) 可以表示为如下形式：

$$\mathrm{GD}(g) = \frac{1}{l_\epsilon(g)} \sum_{i=1}^{N} \delta_\epsilon(g(x^{(i)}), g)$$

其中，$g(x^{(i)})$ 为训练集中第 i 个样本的 gradient norm，并且：

$$\delta_\epsilon(x, y) = \begin{cases} 1 & \text{if } y - \frac{\epsilon}{2} \le x < y + \frac{\epsilon}{2} \\ 0 & \text{otherwise.} \end{cases}$$

$$l_\epsilon(g) = \min(g + \frac{\epsilon}{2}, 1) - \max(g - \frac{\epsilon}{2}, 0)$$

对于 g 的梯度密度表示：

- $\sum_{i=1}^{N} \delta_\epsilon(g(x^{(i)}), g)$: 训练集中 gradient norm 落在以 g 为中心，以 ϵ 为最大区间直径的区间中的样本个数
- $\mathrm{GD}(g) = \frac{1}{l_\epsilon(g)} \sum_{i=1}^{N} \delta_\epsilon(g(x^{(i)}), g)$: 将上述区间内样本个数 $\sum_{i=1}^{N} \delta_\epsilon(g(x^{(i)}), g)$ 除以区间长度 $l_\epsilon(g)$，得到训练集中 gradient norm 在以 g 为中心，以 ϵ 为最大区间直径的区间的 uniform probability distribution.

Definition 3.2 (**gradient density harmonizing parameter**).

$$\beta^{(i)} = \frac{N}{\mathrm{GD}(g(x^{(i)}))}$$

其中 N 为训练集中训练样本的个数。为了更好地理解 gradient density harmonizing parameter $\beta^{(i)}$，可以将其重写为：

$$\beta^{(i)} = \frac{1}{\frac{\mathrm{GD}(g(x^{(i)}))}{N}}$$

其中的分母 $\frac{\mathrm{GD}(g(x^{(i)}))}{N}$ 表示所有训练样本中，gradient norm 大小在第 i 个样本的 gradient norm $g(x^{(i)})$ 附近的**训练样本比例**，因此可以发现，随着梯度密度 $\mathrm{GD}(g(x^{(i)}))$ 的增加，$\frac{\mathrm{GD}(g(x^{(i)}))}{N}$ 增加，相应的损失函数的权重 $\beta^{(i)}$ 就对应地减小。

因此 GHM 在分类和回归问题的目标函数为：

- GHM-C Loss:

$$\begin{aligned}
\mathcal{L}_{\text{GHM-C}} &= \frac{1}{N}\sum_{i=1}^{N}\beta^{(i)}\mathcal{L}_{\mathrm{CE}}(y^{(i)}, h_\theta(x^{(i)})) \\
&= \frac{1}{N}\sum_{i=1}^{N}\frac{N}{\mathrm{GD}(g(x^{(i)}))}\mathcal{L}_{\mathrm{CE}}(y^{(i)}, h_\theta(x^{(i)})) \\
&= \sum_{i=1}^{N}\frac{1}{\mathrm{GD}(g(x^{(i)}))}\mathcal{L}_{\mathrm{CE}}(y^{(i)}, h_\theta(x^{(i)}))
\end{aligned}$$

因此：

- easy samples: largely down-weighted, address imbalance problem
- outliers(very hard samples): slightly down-weighted, address outliers problem
- medium samples: up-weighted, more focus!

3.3.7 Softmax + Cross entropy loss

Cross entropy for multi-class classification task is as follow:

$$\min_{\theta} J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(x^{(i)}, y^{(i)}; \theta)$$

$$\Rightarrow \min_{\theta} J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\mathrm{CE}(p^{(i)}, h_\theta(x^{(i)})) \quad (y^{(i)} \sim p^{(i)} \in \mathbb{P}^{|K|})$$

$$\Rightarrow \min_{\theta} J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\left\langle p^{(i)}, \log h_\theta(x^{(i)})\right\rangle \quad (h_\theta(x^{(i)}) \in \mathbb{P}^{|K|})$$

$$\Rightarrow \min_{\theta} J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_k^{(i)} \log\left[h_\theta(x^{(i)})\right]_k$$

$$\Rightarrow \min_{\theta} J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_k^{(i)} \log \frac{e^{[z_\theta(x^{(i)})]_k}}{\sum_{j=1}^{K} e^{[z_\theta(x^{(i)})]_j}} \quad \text{(i.e. softmax function: } \left[h_\theta(x^{(i)})\right]_k = \frac{e^{[z_\theta(x^{(i)})]_k}}{\sum_{j=1}^{K} e^{[z_\theta(x^{(i)})]_j}} \in \mathbb{P})$$

Where $[z_\theta(x^{(i)})]_k \in \mathbb{R}$ represents the **logit output(score)** of k-th class.

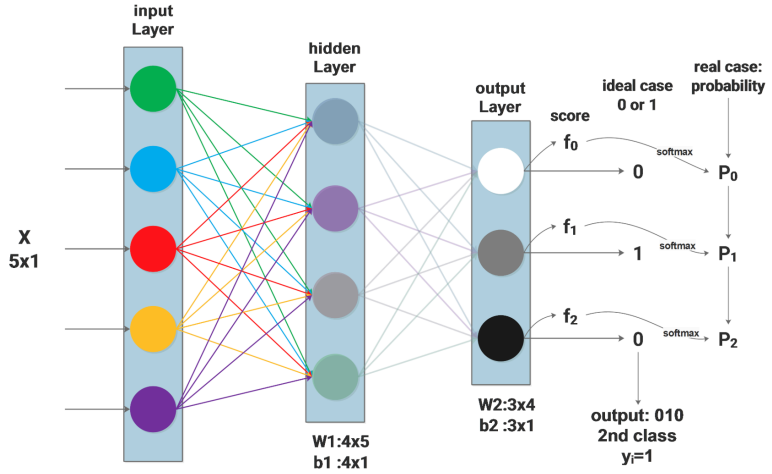


图 9: Softmax loss function. Cite from Wangxin's Blog: NN Softmax loss function

If the label probability distribution is **0-1 probability distribution**, we can rewrite softmax loss as follow:

$$\min_{\theta} J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_k^{(i)} \log \frac{e^{[z_\theta(x^{(i)})]_k}}{\sum_{j=1}^{K} e^{[z_\theta(x^{(i)})]_j}}$$

$$\Rightarrow \min_{\theta} J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} \mathrm{I}(k = y^{(i)}) \log \frac{e^{[z_\theta(x^{(i)})]_k}}{\sum_{j=1}^{K} e^{[z_\theta(x^{(i)})]_j}} \quad \text{(i.e. 0-1 label probability distribution)}$$

Where the definition of indicator function is as follow:

$$\mathrm{I}(k = y^{(i)}) = \begin{cases} 1 & \text{if } k = y^{(i)} \in \{1, ..., K\}, \\ 0 & \text{otherwise.} \end{cases}$$

And we can rewrite softmax loss as follow:

$$\min_{\theta} J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} \mathrm{I}(k = y^{(i)}) \log \frac{e^{[z_\theta(x^{(i)})]_k}}{\sum_{j=1}^{K} e^{[z_\theta(x^{(i)})]_j}}$$

$$\Rightarrow \min_{\theta} J(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{[z_\theta(x^{(i)})]_{y^{(i)}}}}{\sum_{j=1}^{K} e^{[z_\theta(x^{(i)})]_j}}$$

$$\Rightarrow \min_{\theta} J(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{W_{y^{(i)}}^{\mathrm{T}} \cdot f_\phi(x^{(i)})}}{\sum_{j=1}^{K} e^{W_j^{\mathrm{T}} \cdot f_\phi(x^{(i)})}} \quad \text{(seperate logit output layer and previous hidden layers)}$$

Where: $\theta = \{\phi, W\}$

- ϕ: parameters of all hidden layers.
- $W \in \mathbb{R}^{d\times K}$: parameters of last layer, mapping from **latent feature space** to **logit distribution space**.
- $f_\phi(x^{(i)}) \in \mathbb{R}^d$: latent feature vector of final hidden layer.
- $W_j \in \mathbb{R}^d$ means the j-th column of parameter matrix W, also:

$$[z_\theta(x^{(i)})]_j = W_j^{\mathrm{T}} \cdot f_\phi(x^{(i)}) = \left\langle W_j^{\mathrm{T}}, f_\phi(x^{(i)}) \right\rangle$$

Note: we omit the bias b_j in $[z_\theta(x^{(i)})]_j$ here to simplify analysis. Because $[z_\theta(x^{(i)})]_j$ is the inner product between W_j^{T} and $f_\phi(x^{(i)})$, therefore it can be also formulated as follow:

$$\min_{\phi,W} J(\phi, W) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{W_{y^{(i)}}^{\mathrm{T}} \cdot f_\phi(x^{(i)})}}{\sum_{j=1}^{K} e^{W_j^{\mathrm{T}} \cdot f_\phi(x^{(i)})}}$$

$$\Rightarrow \min_{\phi,W} J(\phi, W) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \cos(\theta_{y^{(i)},i})}}{\sum_{j=1}^{K} e^{\|W_j^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}} \quad (\because W_j^{\mathrm{T}} \cdot f_\phi(x^{(i)}) = \|W_j^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \cos(\theta_{j,i}))$$

Where $\theta_{j,i} \in [0, \pi]$ ($j \in \{1, ..., K\}$) is the angle between the parameter vector W_j^{T} and feature vector $f_\phi(x^{(i)})$.

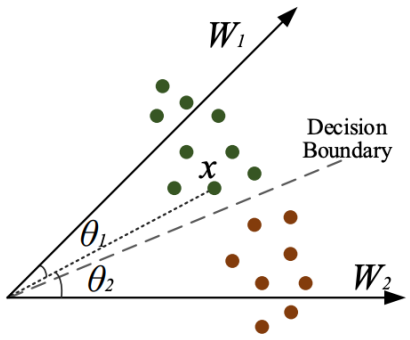
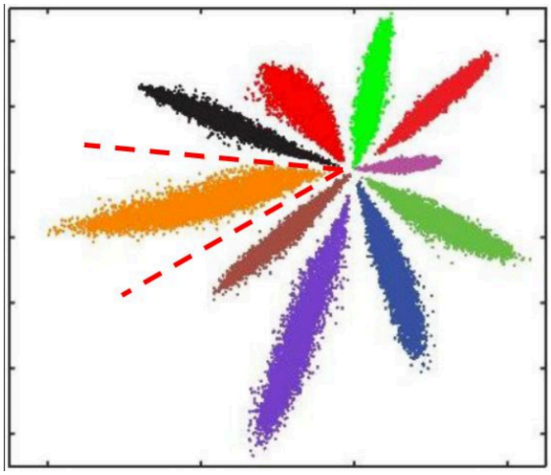


图 10: $\theta_{j,i} \in [0, \pi]$ ($j \in \{1, ..., K\}$) is the angle between the parameter vector W_j^{T} and feature vector $f_\phi(x^{(i)})$.

How to modify Softmax loss? 以 MNIST 作为 dataset，CNN 作为 model inference, Softmax Loss 作为 loss function 为例：MNIST 上 10 分类的 2 维特征映射可视化如下：

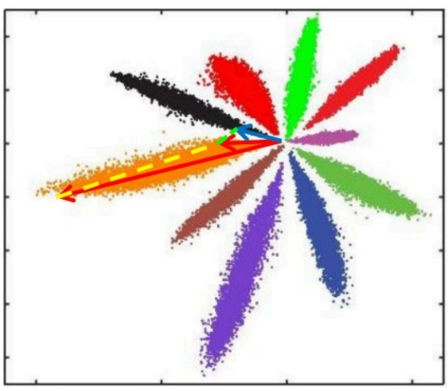


不同类别之间的抽取特征向量 $f_\phi(\boldsymbol{x})$ 明显分隔开了，但这种情况并不满足很多应用场景（e.g. 人脸识别）中特征向量对比差异的需求。因为特征向量相似度的计算常用：

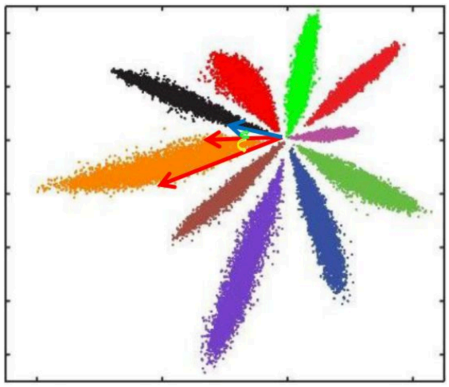
- 欧式距离（L2 distance）
- 余弦距离（cosine distance）

所以在此分别讨论两种情况下遇到的问题。

- 欧式距离（L2 distance）：L2 距离越小，向量相似度越高。但是可能同类的特征向量距离（黄色）比不同类的特征向量距离（绿色）更大。
- 余弦距离（cosine distance）：夹角越小，cos 距离越大，向量相似度越高。但是可能同类的特征向量夹角（黄色）比不同类的特征向量夹角（绿色）更大



(a) l2 distance



(b) cos distance

因此总结来讲：

- Softmax Loss 训练的深度特征，会把整个超空间或者超球，按照分类个数进行划分，保证类别是可分的，这一点对一般多分类任务如 MNIST 和 ImageNet 非常合适，因为测试类别必定在训练类别中。
- Softmax Loss 存在的问题是：**不要求类内紧凑和类间分离**，这一点非常不适合复杂的多分类任务 (e.g. 人脸识别)。因为训练集的 10000 人数（10000 类），相对测试集整个世界 70 亿人类（70 亿类）来说，非常微不足道，而我们不可能拿到所有人的训练样本，更过分的是，一般我们还要求训练集和测试集不重叠。
- 因此需要改造 Softmax Loss，指导思想是：**除保证可分性外，还要做到特征向量 $f_\phi(\boldsymbol{x})$ 类内尽可能紧凑，类间尽可能分离。**
- 对 Softmax loss 的改造有如下几类：

  1. Margin-Loss: insert a geodesic distance margin between the feature sample $f_\phi(\boldsymbol{x})$ and the corresponding centre $\boldsymbol{W}_y$.
     - Large-Margin Softmax Loss[**?**]
     - Angular-Softmax Loss[**?**]
     - ArcFace Loss(Additive Angular-Margin Loss)[**?**]
  2. Intra-Loss: decrease the geodesic distance between the feature sample $f_\phi(\boldsymbol{x})$ and the corresponding centre $\boldsymbol{W}_y$.
  3. Inter-Loss: increase the geodesic distance between different centres(i.e. $\boldsymbol{W}_{j_1}, \boldsymbol{W}_{j_2}, \forall j_1, j_2 \in \{1, ..., K\}, j_1 \neq j_2$).
  4. Triplet-Loss: insert a geodesic distance margin between triplet samples.

因此改造 Softmax Loss 的**模版形式**如下：

$$\mathcal{L}(x^{(i)}, y^{(i)}, \phi, W) = \underbrace{-\log\left(\frac{e^{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \cos(m_1 \cdot \theta_{y^{(i)},i} + m_2)}}{e^{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \cos(m_1 \cdot \theta_{y^{(i)},i} + m_2)} + \sum_{j=1, j\neq y^{(i)}}^{K} e^{\|W_j^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right)}_{\text{Margin-Loss}}$$

$$\underbrace{+\|f_\phi(x^{(i)}) - W_{y^{(i)}}\|}_{\text{Intra-Loss}}$$

$$\underbrace{-\frac{1}{K-1}\sum_{j=1, j\neq y^{(i)}}^{K} \|W_{y^{(i)}}^{\mathrm{T}} - W_j\|}_{\text{Inter-Loss}}$$

3.3.8 Large-Margin Softmax + Cross entropy loss (L-Softmax)

论文出处：Large-Margin Softmax Loss for Convolutional Neural Networks[?]. 该目标函数**强制让样本特征向量 $f_\phi(x) \in \mathbb{R}^d$ 和对应类别的参数向量 W_y 之间的夹角增加到原来 Softmax loss 的 m 倍。**

Intuition softmax loss 如下：

$$\min_{\phi,W} J(\phi, W) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \cos(\theta_{y^{(i)},i})}}{\sum_{j=1}^{K} e^{\|W_j^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}$$

Where $\theta_{j,i} \in [0, \pi]$ ($j \in \{1, ..., K\}$) is the angle between the parameter vector W_j^{T} and feature vector $f_\phi(x^{(i)})$.

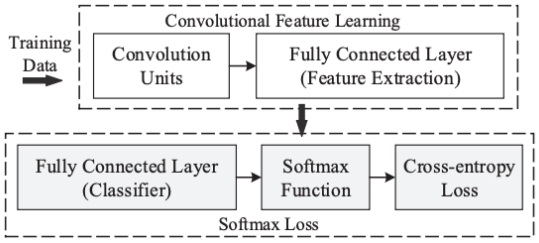


图 11: Standard CNNs can be viewed as convolutional feature learning machines that are supervised by the softmax loss.

Consider the binary classification:

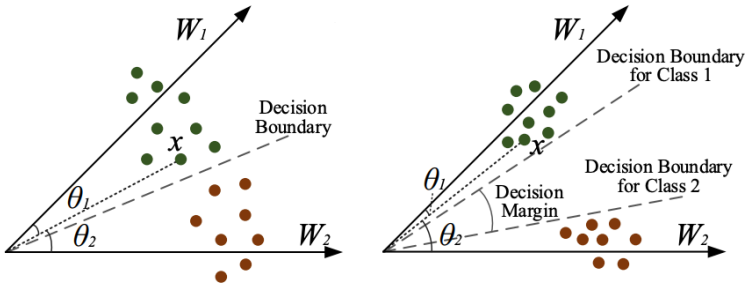


图 12: Original Softmax Loss and L-Softmax Loss

- Suppose we have a sample $f_\phi(\boldsymbol{x})$ from 1-th class.
- The original softmax is to force:

$$\boldsymbol{W}_1^T \boldsymbol{x} > \boldsymbol{W}_2^T \boldsymbol{x}$$

namely:

$$\|\boldsymbol{W}_1^T\| \|\boldsymbol{x}\| \cos(\theta_1) > \|\boldsymbol{W}_2^T\| \|\boldsymbol{x}\| \cos(\theta_2)$$

in order to classify $\boldsymbol{x}$ correctly.

- However, we want to make the classification more rigorous in order to produce a decision margin.
- So we instead require:

$$\|\boldsymbol{W}_1^T\|\|\boldsymbol{x}\|\cos(m\theta_1) > \|\boldsymbol{W}_2^T\|\|\boldsymbol{x}\|\cos(\theta_2) \quad (0 \le \theta_1 \le \frac{\pi}{m})$$

  where m is a **positive integer**.

Theorem 1 (**Inequality of Large-Margin Softmax Loss**)**.** Because the following inequality holds:

$$\begin{aligned}\|\boldsymbol{W}_1^T\|\|\boldsymbol{x}\|\cos(\theta_1) &> \|\boldsymbol{W}_2^T\|\|\boldsymbol{x}\|\cos(m\theta_1) \\ &> \|\boldsymbol{W}_2^T\|\|\boldsymbol{x}\|\cos(\theta_2)\end{aligned}$$

Therefore, $\|\boldsymbol{W}_1^T\|\|\boldsymbol{x}\|\cos(\theta_1) > \|\boldsymbol{W}_2^T\|\|\boldsymbol{x}\|\cos(\theta_2)$ has to hold. So the new classification criteria is a stronger requirement to correctly classify $\boldsymbol{x}$, producing a more rigorous decision boundary for 1-th class.

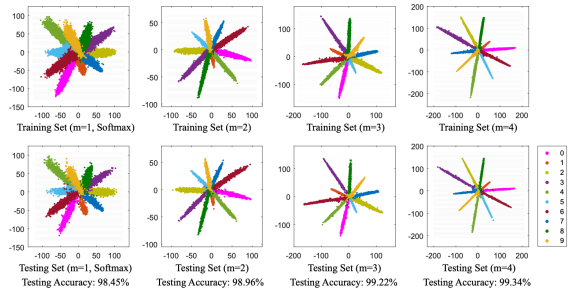


图 13: CNN-leanrned features visualization (Softmax Loss ($m = 1$) vs. L-Softmax loss ($m = 2, 3, 4$)) in MNIST dataset. Specifically, we set the feature (input of the L-Softmax loss) dimension as 2, and then plot them by class. We omit the constant term in the fully connected layer, since it just complicates our analysis and nearly does not affect the performance.

Definition

Definition 3.3 (**L-Softmax loss**)**.** the L-Softmax loss is defined as:

$$\mathcal{L}(x^{(i)}, y^{(i)}; W, \phi) = -\log\left(\frac{e^{\|W_{y^{(i)}}^{\mathrm{T}}\|\|f_\phi(x^{(i)})\|\psi(\theta_{y^{(i)},i})}}{\underbrace{e^{\|W_{y^{(i)}}^{\mathrm{T}}\|\|f_\phi(x^{(i)})\|\psi(\theta_{y^{(i)},i})}}_{\text{positive class}} + \underbrace{\sum_{j=1,j\neq y^{(i)}}^{K} e^{\|W_j^{\mathrm{T}}\|\|f_\phi(x^{(i)})\|\cos(\theta_{j,i})}}_{\text{negative classes}}}\right)$$

in which we generally require:

$$\psi(\theta) = \begin{cases} \cos(m\theta), & 0 \le \theta \le \frac{\pi}{m} \\ \mathcal{D}(\theta), & \frac{\pi}{m} \le \theta \le \pi \end{cases}$$

where:

- m is a positive integer that is closely related to the classification margin.
- with larger m, the classification margin becomes larger and the learning objective also becomes harder.
- $\mathcal{D}(\theta)$(e.g. $\cos(\theta)$) is required to be a monotonically decreasing function and $\mathcal{D}(\frac{\pi}{m})$ should equal $\cos(\frac{\pi}{m})$.
- To simplify the forward and backward propagation, the author construct a specific $\psi(\theta)$ in the paper:

$$\psi(\theta) = (-1)^k \cos(m\theta) - 2k, \quad \theta \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]$$

where $k \in [0, m-1]$ and k is an integer.

但是 L-Softmax loss 存在缺陷：

1. **没有对参数向量 W_j, $\forall j \in \{1, ..., K\}$ 进行归一化**（normalization），导致的结果是：
   - 由于 $\|W_{y^{(i)}}\|$ 很大，所以即使 $W_{y^{(i)}}$ 和 $f_\phi(x^{(i)})$ 之间的夹角 $\theta_{y^{(i)},i}$ 不用很小，也可以使得向量内积 $W_{y^{(i)}} \cdot f_\phi(x^{(i)})$ 的值很大，从而影响类别的判别
   - 而我们真正希望优化参数 $\{\phi, W\}$ 得到的结果是向量 $W_{y^{(i)}}$ 和 $f_\phi(x^{(i)})$ 之间的夹角 $\theta_{y^{(i)},i}$ 非常小！
2. **没有对隐特征向量 $f_\phi(x)$ 进行归一化**（normalization），导致的结果同上，没有完全专注于训练向量的方向，而有可能是向量长度。因此根据论文L2-constrained Softmax Loss for Discriminative Face Verification[?]，对 $f_\phi(x)$ 进行归一化操作.

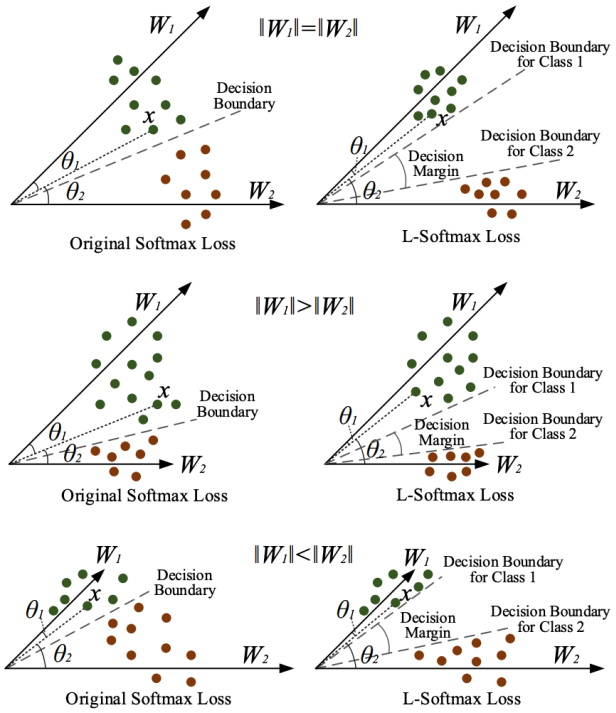


图 14: Examples of Geometric Interpretation

Optimization（未完成）

- $\nabla_{W_{y^{(i)}}} \mathcal{L}(x^{(i)}, y^{(i)}; W, \phi)$
- $\nabla_{W_j} \mathcal{L}(x^{(i)}, y^{(i)}; W, \phi) \quad (j \neq y^{(i)})$

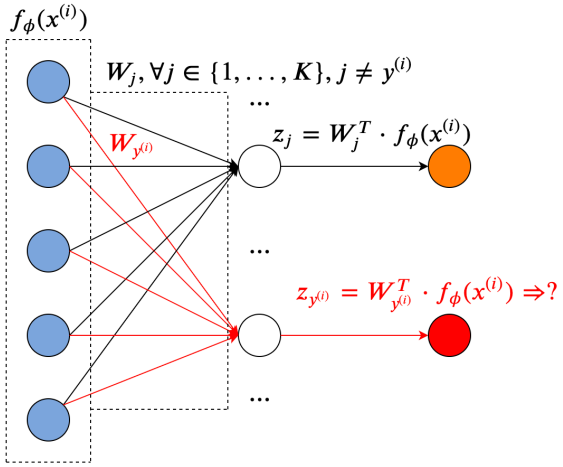


图 15: Optimization for Large-margin softmax + cross-entropy loss.

Proof. 正样本上的逻辑输出（logit output）正向推导表示如下：

$$
\begin{aligned}
z_{y^{(i)}} &= W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)}) \\
&= \|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \cos(\theta_{y^{(i)},i}) \\
&\Rightarrow \|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \psi(\theta_{y^{(i)},i}) \quad \text{（修正原始 softmax 函数中正样本对应的夹角）} \\
&= \|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \Big((-1)^k \cos(m\theta_{y^{(i)},i}) - 2k \Big) \quad (\because\ \psi(\theta) = (-1)^k \cos(m\theta) - 2k, \quad \theta \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]) \\
&= (-1)^k \cdot \|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \cos(m\theta_{y^{(i)},i}) - 2k \cdot \|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \\
\because\ \cos(m\theta) &= C_m^0 \cos^m(\theta) - C_m^2 \cos^{m-2}(\theta) \sin^2(\theta) + C_m^4 \cos^{m-4}(\theta) \sin^4(\theta) + \cdots + \underbrace{(-1)^n C_m^{2n} \cos^{m-2n}(\theta) \sin^{2n}(\theta)}_{\text{标准项}} + \cdots \\
&= C_m^0 \cos^m(\theta) - C_m^2 \cos^{m-2}(\theta)(1 - \cos^2(\theta)) + C_m^4 \cos^{m-4}(\theta)(1 - \cos^2(\theta))^2 + \cdots \\
&+ \underbrace{(-1)^n C_m^{2n} \cos^{m-2n}(\theta)(1 - \cos^2(\theta))^n}_{\text{标准项}} + \cdots \quad (2n \leq m \text{ 并且 } n \text{ 为整数.}) \\
\because\ \cos(\theta_{y^{(i)},i}) &= \frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \quad (\theta_{y^{(i)},i} \text{ 为正样本嵌入向量 } f_\phi(x^{(i)}) \text{ 和对应参数 } W_{y^{(i)}} \text{ 之间的夹角}) \\
\therefore\ z_{y^{(i)}} &= (-1)^k \cdot \|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \Big(C_m^0 \cos^m(\theta_{y^{(i)},i}) - C_m^2 \cos^{m-2}(\theta_{y^{(i)},i}) \sin^2(\theta_{y^{(i)},i}) \\
&+ C_m^4 \cos^{m-4}(\theta_{y^{(i)},i}) \sin^4(\theta_{y^{(i)},i}) + \cdots + (-1)^n C_m^{2n} \cos^{m-2n}(\theta_{y^{(i)},i}) \sin^{2n}(\theta_{y^{(i)},i}) + \cdots \Big) \\
&- 2k \cdot \|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|
\end{aligned}
$$

$$
\begin{aligned}
\therefore\ z_{y^{(i)}} = (-1)^k \cdot \|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \Bigg(& C_m^0 \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^m \\
& - C_m^2 \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^{m-2} \left(1 - \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^2 \right) \\
& + C_m^4 \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^{m-4} \left(1 - \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^2 + \cdots \\
& + (-1)^n C_m^{2n} \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^{m-2n} \left(1 - \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^n + \cdots \Bigg) - 2k \cdot \|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|
\end{aligned}
$$

Consider the gradient of one specific term and note as:

$$
T_n(W_{y^{(i)}}, f_\phi(x^{(i)})) = (-1)^n C_m^{2n} \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^{m-2n} \left(1 - \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^n
$$

$$
\because\ \frac{\partial}{\partial f_\phi(x^{(i)})} \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^{m-2n} = \frac{(m-2n) \big(W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)}) \big)^{m-2n-1} W_{y^{(i)}}}{\Big(\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \Big)^{m-2n-1}}
$$

$$
\because\ \frac{\partial}{\partial f_\phi(x^{(i)})} \left(1 - \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^n = (-2n) \cdot \left(1 - \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^{n-1} \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right) W_{y^{(i)}}
$$

$$
\begin{aligned}
\therefore\ \frac{\partial T_n(W_{y^{(i)}}, f_\phi(x^{(i)}))}{\partial f_\phi(x^{(i)})} = (-1)^n C_m^{2n} \Bigg(& \frac{(m-2n) \big(W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)}) \big)^{m-2n-1} W_{y^{(i)}}}{\Big(\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\| \Big)^{m-2n-1}} \left(1 - \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^n \\
& + \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^{m-2n} (-2n) \cdot \left(1 - \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right)^2 \right)^{n-1} \left(\frac{W_{y^{(i)}}^{\mathrm{T}} f_\phi(x^{(i)})}{\|W_{y^{(i)}}^{\mathrm{T}}\| \|f_\phi(x^{(i)})\|} \right) W_{y^{(i)}} \Bigg)
\end{aligned}
$$

Reference

- github: Large-Margin Softmax Loss
- Billy's Blog: <Tensorflow> How to implement the larget margin softmax loss in tensorflow
- 知乎：人脸识别的 LOSS（上）
- 知乎：人脸识别的 LOSS（下）

3.3.9 Additive Margin Softmax + Cross entropy loss (AM-Softmax)

论文出处：Additive Margin Softmax for Face Verification[?]

3.3.10 Angular-Softmax + Cross entropy loss (A-Softmax)

论文出处：SphereFace: Deep Hypersphere Embedding for Face Recognition[?]. 该目标函数相对于L-Softmax loss[?]**归一化了权重 W，让训练更加集中在优化深度特征映射 $f_\phi(x)$ 和特征向量 W_j 的角度上，降低样本数量不均衡问题。**

Definition

$$
\begin{aligned}
\mathcal{L}(x^{(i)}, y^{(i)}; W, \phi) &= \mathrm{CE}(p^{(i)}, h_{\phi,W}(x^{(i)})) \quad (y^{(i)} \sim p^{(i)} \in \mathbb{P}^{|K|}) \\
&= -\left\langle p^{(i)}, \log\left(h_{\phi,W}(x^{(i)})\right)\right\rangle \quad (h_{\phi,W}(x^{(i)}) \in \mathbb{P}^{|K|}) \\
&= -\sum_{k=1}^{K} p_k^{(i)} \log\left(h_{\phi,W_k}(x^{(i)})\right) \quad (W_k \text{ is the } k\text{-th column of } W,\ h_{\phi,W_k}(x^{(i)}) \in \mathbb{P}) \\
&= -\sum_{k=1}^{K} p_k^{(i)} \log\left(\frac{e^{z_{\phi,W_k}(x^{(i)})}}{\sum_{j=1}^{K} e^{z_{\phi,W_j}(x^{(i)})}}\right) \quad (\text{softmax function, } z_{\phi,W_j}(x) \in \mathbb{R} \text{ outputs logit score}) \\
&= -\sum_{k=1}^{K} p_k^{(i)} \log\left(\frac{e^{W_k^T \cdot f_\phi(x^{(i)}) + b_k}}{\sum_{j=1}^{K} e^{W_j^T \cdot f_\phi(x^{(i)}) + b_j}}\right) \quad (\text{unpack classifier fully connected layer}) \\
&= -\sum_{k=1}^{K} p_k^{(i)} \log\left(\frac{e^{\|W_k\| \|f_\phi(x^{(i)})\| \cos(\theta_{k,i}) + b_k}}{\sum_{j=1}^{K} e^{\|W_j\| \|f_\phi(x^{(i)})\| \cos(\theta_{j,i}) + b_j}}\right) \\
&= -\sum_{k=1}^{K} p_k^{(i)} \log\left(\frac{e^{\|f_\phi(x^{(i)})\| \cos(\theta_{k,i}) + b_k}}{\sum_{j=1}^{K} e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i}) + b_j}}\right) \quad (\textbf{constraint: normalize } \|W_j\| = 1,\ \forall j \in \{1, ..., K\}) \\
&= -\sum_{k=1}^{K} p_k^{(i)} \log\left(\frac{e^{\|f_\phi(x^{(i)})\| \cos(\theta_{k,i})}}{\sum_{j=1}^{K} e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right) \quad (\text{zero the bias}) \\
&= -\sum_{k=1}^{K} \mathrm{I}(k = y^{(i)}) \log\left(\frac{e^{\|f_\phi(x^{(i)})\| \cos(\theta_{k,i})}}{\sum_{j=1}^{K} e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right) \quad (\text{using 0-1 label probability distribution}) \\
&= -\log\left(\frac{e^{\|f_\phi(x^{(i)})\| \cos(\theta_{y^{(i)},i})}}{\sum_{j=1}^{K} e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right)
\end{aligned}
$$

在此 normalize $\boldsymbol{W}_j, \forall j$ 并且 zero $b_j \forall j$ 基础上，再利用 L-Softmax Loss 的方法，可得：

$$
\mathcal{L}_{\text{ang}}(x^{(i)}, y^{(i)}; W, \phi) = -\log\left(\frac{e^{\|f_\phi(x^{(i)})\| \cos(m\theta_{y^{(i)},i})}}{e^{\|f_\phi(x^{(i)})\| \cos(m\theta_{y^{(i)},i})} + \sum_{j=1, j\neq y^{(i)}}^{K} e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right)
$$

其中超参数 m 为大于 0 的正整数，并且向量 $W_{y^{(i)}}$ 和 $f_\phi(x^{(i)})$ 之间的夹角 $\theta_{y^{(i)},i}$ 必须满足:

$$
\theta_{y^{(i)},i} \in \left[0, \frac{\pi}{m}\right]
$$

将上述公式中的 $\cos(m\theta_{y^{(i)},i})$ 泛化为函数 $\psi(\theta_{y^{(i)},i})$，并且该函数满足：

- $\psi(\theta_{y^{(i)},i})$ 随着因变量 $\theta_{y^{(i)},i}$ 单调递减
- 在 $\left[0, \frac{\pi}{m}\right]$ 区间，$\psi(\theta_{y^{(i)},i}) = \cos(m\theta_{y^{(i)},i})$

因此泛化版的 A-Softmax loss 可以表示如下：

$$
\mathcal{L}_{\text{ang}}(x^{(i)}, y^{(i)}; W, \phi) = -\log\left(\frac{e^{\|f_\phi(x^{(i)})\| \psi(\theta_{y^{(i)},i})}}{e^{\|f_\phi(x^{(i)})\| \psi(\theta_{y^{(i)},i})} + \sum_{j=1, j\neq y^{(i)}}^{K} e^{\|f_\phi(x^{(i)})\| \cos(\theta_{j,i})}}\right)
$$

in which we define

$$\psi(\theta_{y^{(i)},i}) = (-1)^k \cos(m\theta_{y^{(i)},i}) - 2k,\ \theta_{y^{(i)},i} \in \left[\frac{k\pi}{m}, \frac{(k+1)\pi}{m}\right]$$

for all $k \in [0, m-1], (m \geq 1)$. 最终的结果如图所示：

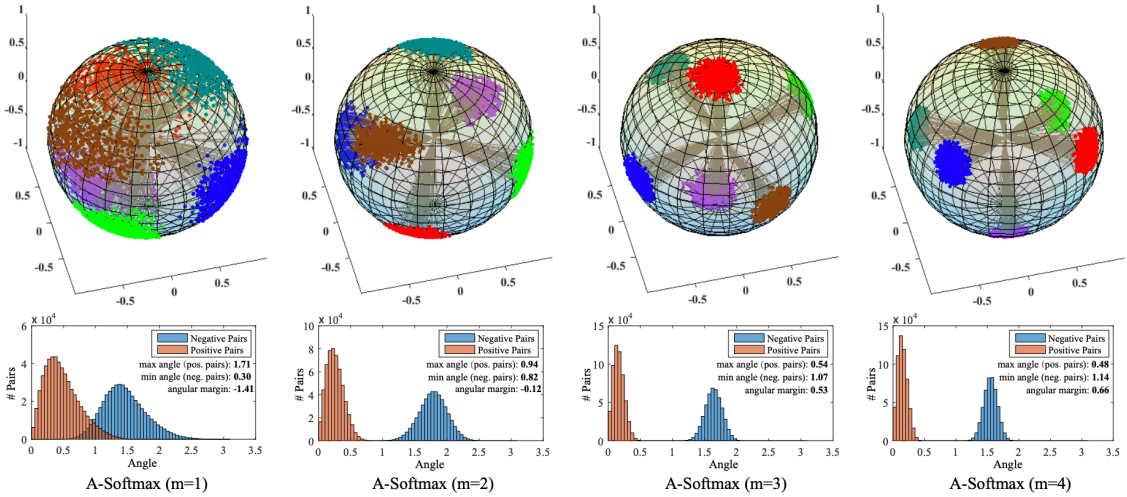


图 16: Visualization of features learned with different m. The first row shows the 3D features projected on the unit sphere. The projected points are the intersection points of the feature vectors and the unit sphere. The second row shows the angle distribution of both positive pairs and negative pairs (we choose class 1 and class 2 from the subset to construct positive and negative pairs). Orange area indicates positive pairs while blue indicates negative pairs.

注意：A-Softmax loss[?] 和L-Softmax loss[?] 都使用了乘性 margin 使不同类别的特征 $f_\phi(\boldsymbol{x}) \in \mathbb{R}^d$ 更加分离，并且特征相似度的计算都采用了 cos 距离。但需要注意的是，这两个 loss 如果**直接训练很难收敛**，因此实际中都采用了**退火优化策略**（annealing optimization strategy），logit output 的表示如下：

$$z_{y^{(i)}} = \frac{\lambda\|W_{y^{(i)}}\|\|f_\phi(x)\|\cos(\theta_{y^{(i)},i}) + \|W_{y^{(i)}}\|\|f_\phi(x)\|\psi(\theta_{y^{(i)},i})}{\lambda + 1}$$

即逐步由相对容易训练的 Softmax Loss 退火到相对困难训练的**注意：**A-Softmax loss[?] 或L-Softmax loss[?]。

Reference

- A Performance Comparison of Loss Functions for Deep Face Recognition[?]
- github: Angular-Softmax loss

3.3.11 ArcFace + Cross entropy loss

论文出处：ArcFace: Additive Angular Margin Loss for Deep Face Recognition[?]

Definition

$$\mathcal{L}_{\text{arc}}(x^{(i)}, y^{(i)}; W, \phi) = -\log\left(\frac{e^{\alpha\|W_{y^{(i)}}^{\mathrm{T}}\|\|f_\phi(x^{(i)})\|\cos(\theta_{y^{(i)},i}+m)}}{e^{\alpha\|W_{y^{(i)}}^{\mathrm{T}}\|\|f_\phi(x^{(i)})\|\cos(\theta_{y^{(i)},i}+m)} + \sum_{j=1,j\neq y^{(i)}}^{K} e^{\alpha\|W_j^{\mathrm{T}}\|\|f_\phi(x^{(i)})\|\cos(\theta_{j,i})}}\right)$$

为了保障 W_j^{T} 和 $f_\phi(x)$ 的学习都是在方向上而不是在无用的长度上，因此在计算前先对 W_j^{T} 和 $f_\phi(x)$ 进行归一化操作：

$$\|W_{y^{(i)}}^{\mathrm{T}}\| = 1$$
$$\|f_\phi(x^{(i)})\| = 1$$

而后为了保障训练速度和效果，添加了一个**re-scale factor:** $\alpha \in (0, +\infty)$

$$s(y^{(i)}, i) \in \mathbb{R} = \alpha \cdot \|W_{y^{(i)}}^{\mathrm{T}}\|\|f_\phi(x^{(i)})\| = \alpha$$

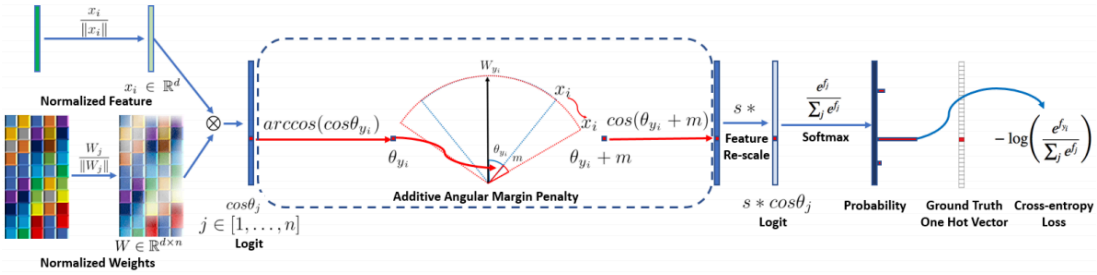


图 17: Training a DCNN for face recognition supervised by the ArcFace loss. Based on the feature $x^{(i)}$ and weight W normalization, we get the $\cos(\theta_{j,i})$ (logit) for each class as $W_j^{\mathrm{T}}x^{(i)}$. We calculate the $\arccos(\theta_{y^{(i)},i})$ and get the angle between the feature $x^{(i)}$ and the ground truth weight $W_{y^{(i)}}$. In fact, W_j provides a kind of center for each class. Then, we add an angular margin penalty m on the target (ground truth) angle $\theta_{y^{(i)},i}$. After that, we calculate $\cos(\theta_{y^{(i)},i}+m)$ and multiply all logits by the feature scale s. The logits then go through the softmax function and contribute to the cross entropy loss.

因此真正的目标函数为：

$$\min_{\phi,W} J(\phi, W) = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{e^{s(y^{(i)},i)\cos(\theta_{y^{(i)},i}+m)}}{e^{s(y^{(i)},i)\cos(\theta_{y^{(i)},i}+m)} + \sum_{j=1,j\neq y^{(i)}}^{K} e^{s(j,i)\cos(\theta_{j,i})}}\right)$$

Idea

- Margin-Loss 的模版：综合对正样本夹角 $\theta_{y^{(i)},i}$ 的加和乘：$\cos(\theta_{y^{(i)},i}+m) \Rightarrow \cos(m_1\cdot\theta_{y^{(i)},i}+m_2)$
- 不同样本的 $s(y^{(i)}, i) = \alpha \cdot \|W_{y^{(i)}}^{\mathrm{T}}\|\|f_\phi(x^{(i)})\|\,(\alpha > 0)$ 中对应的 α 尺度应该不同
- 融合 online hard sample mining(e.g. focal loss)

3.3.12 Triplet loss

论文出处：FaceNet: A Unified Embedding for Face Recognition and Clustering[?]。该类目标函数用于**细粒度的个体识别**应用场景。并且 Triplet loss 用于**embedding** $f_\phi(x) \in \mathbb{R}^D$ 的学习，而之后没有 softmax 层，也就没有分类参数 $W \in \mathbb{R}^{D\times K}$ 的学习。

Introduction Triplet loss的引入需要从传统分类任务的损失函数 softmax + cross-entropy 说起，而传统 softmax + cross-entropy 分类训练结构 + 损失函数的框架存在如下问题：

1. 假设每一个目标（identity）的 embedding 维度为 D，即：$f_\phi(x) \in \mathbb{R}^D$；在分类**类别数量固定**的情况下，总分类个数为 K，那么仅 softmax 层的参数数量共计 $D \times K$ 个，如果类别数量非常庞大（e.g. 全中国人的人脸识别，13 亿个类），则算法无论从参数设置存储角度，还是参数的训练学习角度看都非常不现实。

2. 在一些识别任务中，**类别个数 K 为变量而非固定值**，譬如在人脸识别任务中，有可能这次识别目标规模小，需要在 $K = 100$ 个类别中进行识别比较；而下一次识别规模较大，需要在包含上次所有类别的 $K = 10000$ 个类别中进行比较，那么这一次有 $10000 - 100 = 9900$ 个类没有被训练过，不靠谱。

3. 很多时候特别在**细粒度识别任务**中，识别比较更多地是**“相对优势”ranking 的思想而非“非黑即白”的 classification 的思想**，譬如 NLP 领域的近似文本匹配任务，并不是一个非黑即白的二分类任务，更多地是模型 rank 出一个结果，看谁相对于其他更加匹配。

Methodology $\left(x^{(A)} \in \mathbb{R}^{\mathcal{X}}, x^{(P)} \in \mathbb{R}^{\mathcal{X}}, x^{(N)} \in \mathbb{R}^{\mathcal{X}}\right)$ 的三元组为一个 sample，并且从样本空间到 embedding 空间的映射函数为 $f_\phi(\cdot): \mathbb{R}^{\mathcal{X}} \to \mathbb{R}^D$。Triplet loss 的目的如下：

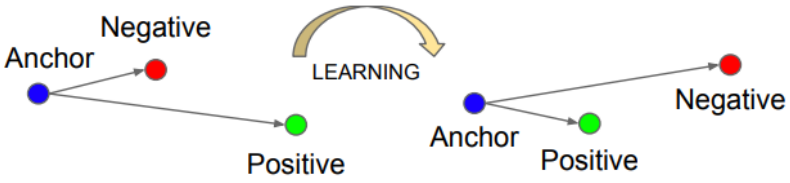


图 18: The Triplet Loss minimizes the distance between an anchor and a positive, both of which have the same identity, and maximizes the distance between the anchor and a negative of a different identity.

- 最小化 Anchor (baseline) 和 Positive 的**绝对**距离：

$$\min_\phi \; d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right)$$

- 最大化 Anchor (baseline) 和 Negative 的**绝对**距离：

$$\begin{aligned} & \max_\phi \; d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right) \\ \Rightarrow \quad & \min_\phi \; -d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right) \end{aligned}$$

- (**难理解点**) 保障 Anchor (baseline) 和 Positive 之间的距离**相对小于**Anchor (baseline) 和 Negative 之间的距离：

$$
\begin{aligned}
& d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right) < d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right) \\
\Rightarrow\quad & d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right) - d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right) < 0 \\
\Rightarrow\quad & d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right) - d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right) < 0 - \alpha \quad (\alpha > 0 \text{ 为 margin 超参数，更加严格}) \\
\Rightarrow\quad & d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right) - d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right) + \alpha < 0
\end{aligned}
$$

  α 的存在是为了尽可能限制 $d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right) > d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right)$。又因为 $d(\cdot,\cdot) \geq 0$ 恒成立，因此我们希望上述距离之差的值尽可能小（实际上最终越负数越好），即：

$$
\min_{\phi}\ d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right) - d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right) + \alpha
$$

  但是从**训练阶段优化的角度**看，分两类情况：

  1. 当 $d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right) - d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right) + \alpha < 0$ 时：即 Anchor 到 Positive 之间的距离已经比 Anchor 到 Negative 之间的距离小 α 了，已经很好地满足了要求了，所以不必考虑这个三元组样本了，直接将这个样本对应的损失 mask 掉（损失函数值设置为 0）。
  2. 当 $d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right) - d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right) + \alpha \geq 0$ 时：可视为 Anchor 到 Positive 之间的距离还比 Anchor 到 Negative 大，即不满足条件，仍然需要优化学习。

  综上所述，目标函数最终设置为：

$$
\mathcal{L}\left(\phi; \left(x^{(A)}, x^{(P)}, x^{(N)}\right)\right) = \max\left(d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right) - d\left(f_\phi(x^{(A)}), f_\phi(x^{(N)})\right) + \alpha, 0\right)
$$

注意上述距离 $d(\cdot,\cdot)$ $(e.g.\ d\left(f_\phi(x^{(A)}), f_\phi(x^{(P)})\right) = \|f_\phi(x^{(A)}) - f_\phi(x^{(P)})\|^2)$ 是在 embedding 空间内的距离，而不是原始样本空间内的距离。

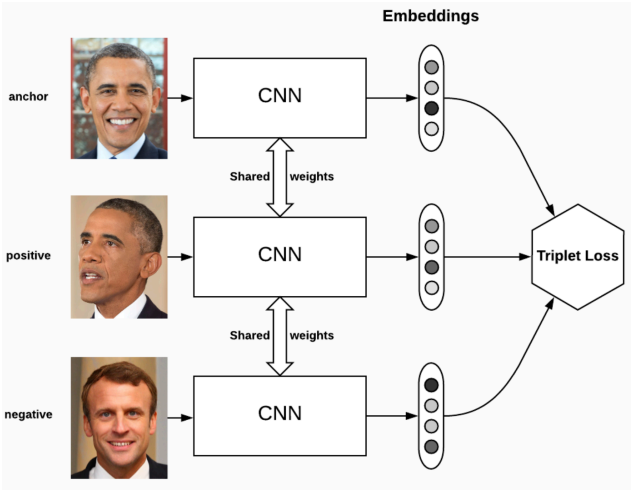


图 19: Triplet loss on two positive faces (Obama) and one negative face (Macron).

因此针对由三元组样本构成的数据集 $\mathcal{D}=\left\{\left(x^{(A)},x^{(P)},x^{(N)}\right)\right\}$ 的训练目标函数为：

$$\min_{\phi}\ \mathcal{L}(\phi;\mathcal{D})=\frac{1}{|\mathcal{D}|}\sum_{\left(x^{(A)},x^{(P)},x^{(N)}\right)\in\mathcal{D}}\max\left(d\left(f_\phi(x^{(A)}),f_\phi(x^{(P)})\right)-d\left(f_\phi(x^{(A)}),f_\phi(x^{(N)})\right)+\alpha,0\right)$$

注意：一定先对 embedding 进行归一化再学习，如此是针对 embedding 的方向学习而不是大小：

$$f_\phi(x^{(A)})\leftarrow\frac{f_\phi(x^{(A)})}{\|f_\phi(x^{(A)})\|},\quad f_\phi(x^{(P)})\leftarrow\frac{f_\phi(x^{(P)})}{\|f_\phi(x^{(P)})\|},\quad f_\phi(x^{(N)})\leftarrow\frac{f_\phi(x^{(N)})}{\|f_\phi(x^{(N)})\|}$$

Triplet mining 对于一个三元组样本中的 Negative，一般可将一个三元组样本 $\left(x^{(A)},x^{(P)},x^{(N)}\right)$ 分为三类：

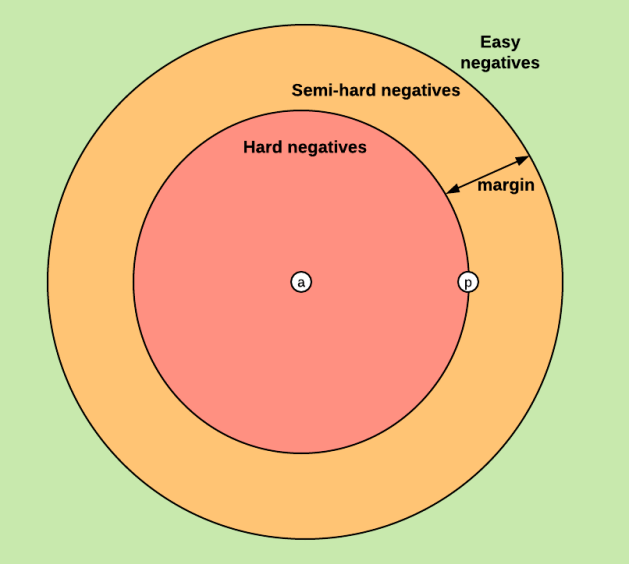


图 20: The three types of negatives, given an anchor and a positive. Image source: Olivier Moindrot blog: Triplet Loss and Online Triplet Mining in TensorFlow.

1. **easy triplets (easy negatives)**: triplets with loss of 0, because

$$d\left(f_\phi(x^{(A)}),f_\phi(x^{(P)})\right)+\alpha<d\left(f_\phi(x^{(A)}),f_\phi(x^{(N)})\right)$$

2. **semi-hard triplets (semi-hard negatives)**: triplets where the negative is not closer to the anchor than the positive, but which still have positive loss:

$$d\left(f_\phi(x^{(A)}),f_\phi(x^{(P)})\right)<d\left(f_\phi(x^{(A)}),f_\phi(x^{(N)})\right)<d\left(f_\phi(x^{(A)}),f_\phi(x^{(P)})\right)+\alpha$$

3. **hard triplets (hard negatives)**: triplets where the negative is closer to the anchor than the positive, i.e.

$$\begin{aligned}&d\left(f_\phi(x^{(A)}),f_\phi(x^{(N)})\right)<d\left(f_\phi(x^{(A)}),f_\phi(x^{(P)})\right)\\\Rightarrow\ &d\left(f_\phi(x^{(A)}),f_\phi(x^{(P)})\right)-d\left(f_\phi(x^{(A)}),f_\phi(x^{(N)})\right)>0\end{aligned}$$

原始FaceNet[?] 论文中通过为每对 $\left(x^{(A)}, x^{(P)}\right)$ 随机选择**semi-hard**的 $x^{(N)}$，构造出所有的三元组样本 $\left(x^{(A)}, x^{(P)}, x^{(N)}\right)$，然后在这些三元组样本的集合 $\mathcal{D} = \left\{\left(x^{(A)}, x^{(P)}, x^{(N)}\right)\right\}$ 上进行学习训练。

Offline and online triplet mining（未完成）

Offline triplet mining

Online triplet mining

Idea. 在深度模型中考虑使用层次化的多损失函数（hierarchical multi loss functions），譬如：

- 在 embedding（kernel）层的学习中使用 triplet loss，更好地学习每个样本 x 对应的抽取特征 $f_\phi(x) \in \mathbb{R}^{D_{\text{kernel}}}$
- 在最后的 softmax 分类层的学习中使用 cross entropy loss，从而更好地学习分类系数 $W \in \mathbb{R}^{D_{\text{kernel}} \times K}$

而事实上已经有算法，特别是人脸识别领域的算法使用了这样 idea，用于同时加强 $f_\phi(x)$ 和 softmax 层参数 W 的多损失函数。譬如Deep Learning Face Representation by Joint Identification-Verification (DeepID2)[?]

- Identification Loss:

$$\text{Ident}(f_\phi(x), t; W) = -\sum_{k=1}^{K} p_k \log \hat{y}_k = -\log \hat{p}_t \quad (\text{where } \hat{p}_t = \frac{e^{W_t^T f_\phi(x) + b_t}}{\sum_{k=1}^{K} e^{W_k^T f_\phi(x) + b_k}})$$

- Verification Loss:

$$\text{Verif}(f_\phi(x^{(i)}), f_\phi(x^{(j)}), y_{ij}, \phi) = \begin{cases} \frac{1}{2}\|f_\phi(x^{(i)}) - f_\phi(x^{(j)})\|_2^2 & \text{if: } y_{ij} = 1 \quad \text{(same person)} \\ \frac{1}{2}\max\left(0, m - \|f_\phi(x^{(i)}) - f_\phi(x^{(j)})\|_2\right)^2 & \text{if: } y_{ij} = -1 \quad \text{(different person)} \end{cases}$$

Reference

- Triplet Loss and Online Triplet Mining in TensorFlow
- In Defense of the Triplet Loss for Person Re-Identification[?]
- Brandon Amos: OpenFace 0.2.0: Higher accuracy and halved execution time
- Deep Learning Face Representations with Different Loss Functions for Face Recognition

3.3.13 COCO loss

论文出处：Learning Deep Features via Congenerous Cosine Loss for Person Recognition[?] 以及Rethinking Feature Discrimination and Polymerization for Large-scale Recognition[?]

Reference

- Slides: Learning Deep Features via Congenerous Cosine Loss for Person Recognition(COCO loss)

3.3.14 Center loss

论文出处：A Discriminative Feature Learning Approach for Deep Face Recognition[?]

3.3.15 Center Invariant loss

论文出处：Deep Face Recognition with Center Invariant Loss

3.3.16 Wasserstein distance and the Kantorovich-Rubinstein duality

Earth Mover's Distance For discrete probability distributions, the Wasserstein distance is also descriptively called the earth mover ' s distance (EMD). If we imagine the distributions as different heaps of a certain amount of earth, then the EMD is the minimal total amount of work it takes to transform one heap into the other. Work is defined as the amount of earth in a chunk times the distance it was moved. Let ' s call our discrete distributions P_r and P_θ, each with l possible states(chucks) x or y respectively, and take two arbitrary distributions as an example.

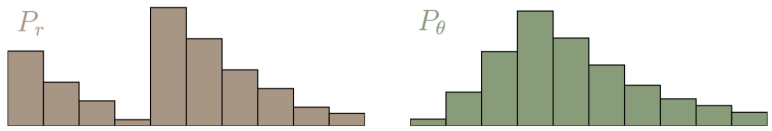


图 21: Probability distribution P_r and P_θ, each with ten states(chucks).

Calculating the EMD is in itself an optimization problem: There are infinitely many ways to move the earth around, and we need to find the optimal one. We call the transport plan that we are trying to find $\gamma(x, y) \in \mathbb{R}^{l\times l}$. It simply states how we distribute the amount of earth from one place y(one chunk in P_r) over the domain of x(one chunk in P_θ), or vice versa.

To be a valid transport plan, the transport plan $\gamma(x, y)$ must satisfy the following constraints:

$$\begin{cases} \sum_x \gamma(x,y) = P_r(y) \quad \forall y \in \{1,...,l\} \\ \\ \sum_y \gamma(x,y) = P_\theta(x) \quad \forall x \in \{1,...,l\} \end{cases}$$

This ensures that following this plan yields the correct distributions.

Equivalently, we can call γ a **joined probability distribution** and require that:

$$\gamma \in \Pi(P_r, P_\theta)$$

where $\Pi(P_r, P_\theta)$ is the set of all distributions whose marginal distributions are P_r and P_θ respectively. To get the EMD, we have to multiply every value of $\gamma \in \mathbb{R}^{l\times l}$ with the Euclidian distance between x and y. With that, the definition of the Earth mover ' s distance is:

$$\begin{aligned} \text{EMD}(P_r, P_\theta) &= \inf_{\gamma\in\Pi} \sum_{x,y} \|x-y\|\gamma(x,y) \\ &= \inf_{\gamma\in\Pi} \mathbb{E}_{(x,y)\sim\gamma}\|x-y\| \end{aligned}$$

We can also set $\boldsymbol{\Gamma} = \gamma(x, y)$ and $\boldsymbol{D} = \|x - y\|$, with $\boldsymbol{\Gamma}, \boldsymbol{D} \in \mathbb{R}^{l\times l}$, Now we can write:

$$\text{EMD}(P_r, P_\theta) = \inf_{\gamma\in\Pi} \langle \boldsymbol{\Gamma}, \boldsymbol{D} \rangle_{\text{F}}$$

where $< \cdot, \cdot >_{\text{F}}$ is the Frobenius inner product (sum of all the element-wise products).

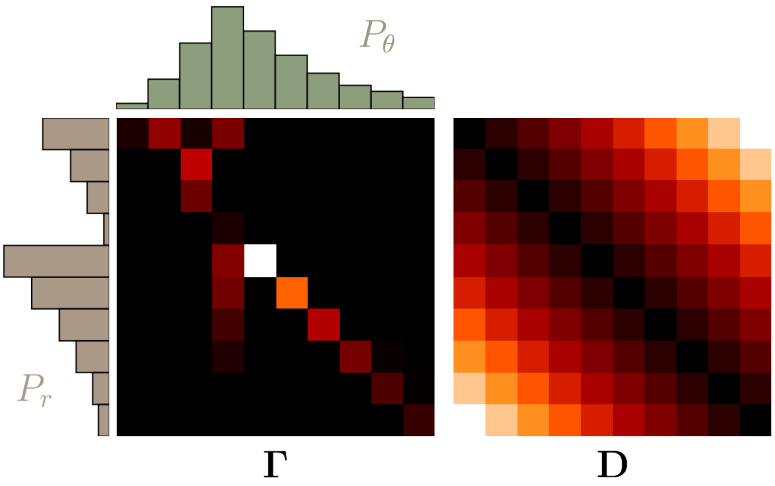


图 22: Transport plan $\mathbf{\Gamma}$ with marginal distributions P_r and P_θ, and distances $\boldsymbol{D}$.

Linear Programming In the picture above, we can see the optimal transport plan $\mathbf{\Gamma}$, It can be calculated using the generic method of Linear Programming(LP). With LP, we can solve problems of a certain canonical form:

- variable: Find a vector $\boldsymbol{x} \in \mathbb{R}^n$
- objective: that minimizes the cost $z = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x},\ \boldsymbol{c} \in \mathbb{R}^n$
- constraint: $\boldsymbol{x}$ is constrainted by the equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{x} \geq \mathbf{0}$

To cast our problem of finding the EMD into this form, we have to flatten $\mathbf{\Gamma}$ and $\boldsymbol{D}$ as follow:

$$\boldsymbol{x} = \mathrm{vec}(\mathbf{\Gamma}) \in \mathbb{R}^{l^2}$$
$$\boldsymbol{c} = \mathrm{vec}(\boldsymbol{D}) \in \mathbb{R}^{l^2}$$

and make RHS of constraint as follow:

$$\boldsymbol{b} = \begin{bmatrix} P_r \\ P_\theta \end{bmatrix} \in \mathbb{R}^{2l}$$

LHS of constraint as follow:

Reference

- vincentherrmann.github.io: Wasserstein GAN and the Kantorovich-Rubinstein Duality
- Read-through: Wasserstein GAN
- Wasserstein of Wasserstein Loss for Learning Generative Models[**?**]

3.4 Regularization

3.4.1 L1 regularization (Lasso regularization)

3.4.2 L2 regularization (Ridge regularization)

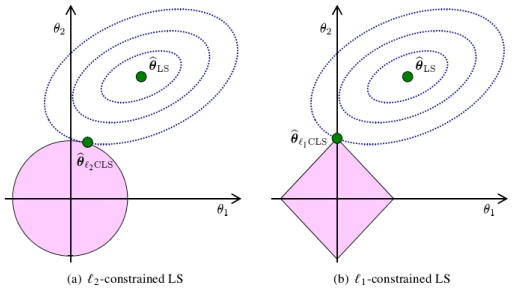


图 23: L1 (Lasso) regularization and L2 (Ridge) regularization.

3.4.3 Mini-batch aware regularization

在稀疏学习任务（e.g. CTR 预估）中，为了防止过拟合，需要引入正则化项，但是又希望对参数 $\boldsymbol{\theta}$ 的学习在每一个 mini-batch 内都能充分进行，因此考虑对正则化项做一些 mini-batch 自适应的设计。
根据论文DIN, Zhou et al., 2018[?]，因此在一批训练数据 $\mathcal{B}=\left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{M}$ 内参数 w_j 的学习如下：

- 判断当前批训练数据 $\mathcal{B}$ 内所有 sample 的第 j 维数据是否存在非 0 值：

$$I_j=\begin{cases}1, & \exists\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\in\mathcal{B},\ \text{s.t. } x_j^{(i)}\neq 0\\ 0, & \text{otherwise.}\end{cases}$$

- 设计属于当前批训练数据 $\mathcal{B}$ 的正则化项：

$$\Omega_{\mathcal{B}}(w_j)=\lambda\frac{1}{n_j}w_jI_j,\quad (n_j \text{ 表示在所有训练样本 } \mathcal{D}=\left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N} \text{ 中第 } j \text{ 维数据非 0 的训练样本个数})$$

- 计算当前 mini-batch 数据 $\mathcal{B}$ 作用下参数 w_j 的更新：

$$w_j\leftarrow w_j-\eta\left(\frac{1}{|\mathcal{B}|}\sum_{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\in\mathcal{B}}\frac{\partial\mathcal{L}\left(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)}\right)}{\partial w_j}+\lambda\frac{1}{n_j}w_jI_j\right)$$

  注意：原始论文中正则化距离采用了 l1-norm，也可以尝试 l2-norm 或者其他，具体效果依照实验超参数调节结果而定。

因此，自适应 (aware) 主要来自于两层次方面：

- 全部训练样本集 $\mathcal{D}=\left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$ 层次：维度 j 上非的样本出现次数越少，惩罚越重，有效防止过拟合（原因：维度 j 上非 0 的样本出现次数多了，参数的有效学习几率更高，学习更加充分，因此几乎不存在因为有效训练样本太少而导致的过拟合问题，所以惩罚反而更轻）。
- 批训练样本集 $\mathcal{B}=\left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{M}$ 层次：批内数据中维度 j 上出现了非 0 值，则作用惩罚项，否则不作用惩罚项。

3.4.4 Summary of regularization

- 一视同仁的 regularization 策略（e.g. L1 regularization, L2 regularization）
- 针对单个特征（考虑有效特征训练样本个数）的 regularization 策略（e.g. Mini-batch aware regularization）
- 针对数值型和类别型特征，分别设置不同的 regularization 策略（未来可展开工作）
- 针对组（group）、域（field）、特征（feature）的重要程度，设置不同的 regularization 策略（未来可展开工作）

4 Optimization Algorithms

4.1 Overview

- 机器学习，深度学习以及强化学习的本质为解一个最优化问题
- 优化问题又可分为无约束优化问题（unconstrainted optimization）和约束优化（constrainted optimization）问题
- 求解优化问题就需要各种各样的优化算法，通常是对高度非凸优化问题进行优化求解

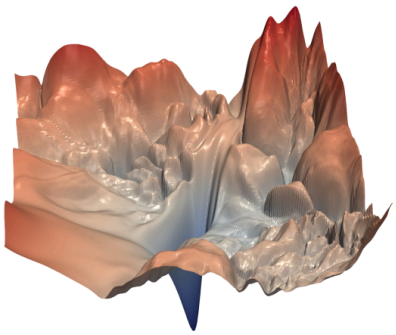


图 24: Visualizing the Loss Landscape of Neural Nets[?]

4.2 Gradient Descent Variants

每一次迭代更新参数时，我们会考虑如下内容：

- 参数迭代更新的准确性
- 参数迭代更新的速度

然后需要在这两者之间进行权衡。而批数据量 m 是决定这两者的因素，因此根据 m 大小不同，可以做如下分类：

- **Batch gradient descent**: 每次迭代使用全部训练数据更新参数

$$w \leftarrow w - \eta \cdot \frac{\partial J(\boldsymbol{w}; \boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)}; y^{(1)}, ..., y^{(n)})}{\partial w}$$

- **Stochastic gradient descent**: 每次迭代使用随机选择的一条训练数据 $(\boldsymbol{x^{(i)}}, y^{(i)})$ 更新参数

$$w \leftarrow w - \eta \cdot \frac{\partial J(\boldsymbol{w}; \boldsymbol{x^{(i)}}; y^{(i)})}{\partial w}$$

- **Mini-batch gradient descent**: 每次迭代使用随机选择的 $m\,(m \ll n)$ 条训练数据更新参数

$$w \leftarrow w - \eta \cdot \frac{\partial J(\boldsymbol{w}; \boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}; y^{(1)}, ..., y^{(m)})}{\partial w}$$

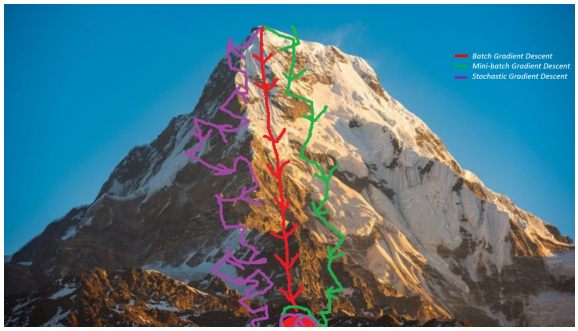


图 25: Comparsion of gradient descent with different training samples

这三种方式各有特点：

- **Batch gradient descent**:
  - 优点：更新最平稳
  - 缺点：更新速度最慢
- **Stochastic gradient descent**:
  - 优点：波动性会使参数跳跃到新的，潜在更好的局部最优点；更新速度最快
  - 缺点：更新最不稳定，波动性太强；没有充分利用计算机的矩阵加速性能
- **Mini-batch gradient descent**: 二者的折中

– 优点 1：减少了参数更新时 SGD 的波动，做到更平稳的收敛

– 优点 2: 充分利用深度学习库里高效的矩阵计算性能

关于 GD, SGD, MBGD 三者的形象对比：

- 多个“ 长者”，多个“ 人生经验”，少走“ 弯路”，走的稳
- “ 人生经验”太多记不下（空间开销），自行消化太费时（时间消耗）
- 随机选一部分“ 长者”的“ 人生经验”作为指导依据

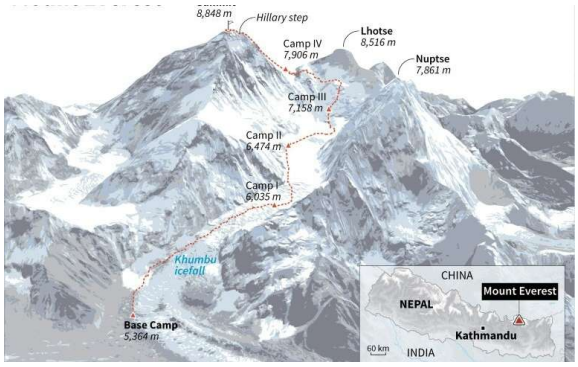


图 26: Optimization task just like get down the mountain

4.3 Gradient Descent Optimization Algorithms

而基于 Mini-batch gradient descent，对学习率 η 和梯度方向 $\frac{\partial J(\boldsymbol{w})}{\partial w}$ 进行适当的变化，可以得到如下一些梯度下降的变种优化算法 [?]。

4.3.1 SGD

Algorithm 1: Stochastic gradient descent (SGD) update[?]

Input: Learning rate η_0;
Initial parameter $\boldsymbol{\theta}$.

Initialize global step counter $t \leftarrow 0$
while *stopping criterion not met* **do**
  Update global step counter $t \leftarrow t+1$
  Sample a minibatch of m samples from the training dataset with corresponding targets: $\left\{\boldsymbol{x^{(i)}}, y^{(i)}\right\}_{i=1}^{m}$.
  Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\boldsymbol{x^{(i)}}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_t \boldsymbol{g}$
end

算法解释

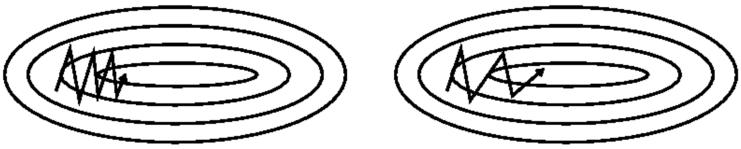
- SGD 算法一般还会对学习率 η 采用**衰减策略**，从而防止训练后期出现梯度爆炸的情况。

4.3.2 Momentum

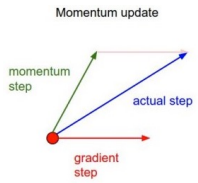
Algorithm 2: Stochastic gradient descent with momentum[**?**]

Input: Learning rate η;
Momentum parameter α;
Initial parameter $\boldsymbol{\theta}$;
Initial velocity $\boldsymbol{v}$.

while *stopping criterion not met* **do**
  Sample a minibatch of m samples from the training dataset with corresponding targets: $\left\{\boldsymbol{x^{(i)}}, y^{(i)}\right\}_{i=1}^{m}$.
  Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\boldsymbol{x^{(i)}}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta \boldsymbol{g}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end



(a) left—SGD without momentum, right—SGD with momentum



(b) Momentum update

图 27: Momentum algorithm and update

算法解释

- Momentum 优化算法相对于 SGD 考虑了先前迭代 step 的梯度：
  - 当前步更新方向 $-\boldsymbol{g}$ 和先前累计更新方向 $\boldsymbol{v}$ 相似时，累计负梯度起到了加速并减小震荡的作用
  - 当前步更新方向 $-\boldsymbol{g}$ 和先前累计更新方向 $\boldsymbol{v}$ 相反时，累计负梯度起到了减速并减小震荡的作用
- Momentum 相当于之前 $\frac{1}{1-\alpha}$ 个时刻的梯度向量和的平均值，因此最大可以实现 $\frac{1}{1-\alpha}$ 倍的加速
- α 一般取 0.9

4.3.3 Nesterov momentum

Algorithm 3: SGD with Nesterov momentum[**?**]

Input: Learning rate η;
Momentum parameter α;
Initial parameter $\boldsymbol{\theta}$;
Initial velocity $\boldsymbol{v}$.

while *stopping criterion not met* **do**
  Sample a minibatch of m samples from the training dataset with corresponding targets: $\left\{\boldsymbol{x^{(i)}}, y^{(i)}\right\}_{i=1}^{m}$.
  Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
  Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\tilde{\boldsymbol{\theta}}} \mathcal{L}\left(f\left(\boldsymbol{x^{(i)}}; \tilde{\boldsymbol{\theta}}\right), y^{(i)}\right)$
  Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta \boldsymbol{g}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end

算法解释

- Nesterov momentum 优化算法最核心的思想就是**预判前方走势**
- 因为步骤 $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$ 通过在累计方向 $\boldsymbol{v}$ 上前进一小步，将优化初始点带到了新的出发点 $\tilde{\boldsymbol{\theta}}$
- 从当前迭代的原始出发点 $\boldsymbol{\theta}$ 到新出发点 $\tilde{\boldsymbol{\theta}}$ 的方向和累计更新方向 $\boldsymbol{v}$ 是相同的，即**“顺应大势所趋，在潮流方向上先偷偷挪了一小步”**
- 所以相对于 Momentum 算法，更可以实现加速的目的。

SGD, Momentum, Nesterov 优化算法存在的问题

- 在每一个 step 的更新中，找到一个合适的学习率 η 是十分困难的。而学习率 η 太小导致收敛速度太慢，学习率 η 太大又会导致在损失函数最小值点附近波动甚至发散。

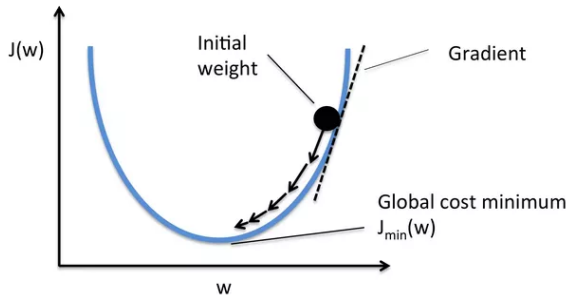


图 28: Loss function & gradient based optimization

- 训练过程中，学习率的衰减策略 η 是人为制定的，不能够自动匹配数据的特征
- 所有参数的更新都使用相同的学习率 η。如果数据非常稀疏，并且特征的频率大为不同，我们不应该对所有的特征使用相同的学习率 η，而应该对出现频率低的特征使用相对大的学习率，而对出现频率高的特征使用相对小的学习率
- 很难避免次优点（鞍点 saddle points）

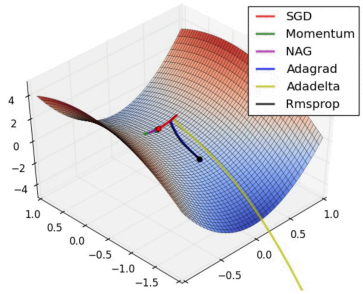


图 29: Hard for SGD, Momentum, Nesterov optimizers to avoid saddle point

解决办法：使用**自适应学习率优化算法**（Adaptive learning rate）

4.3.4 AdaGrad

Algorithm 4: The AdaGrad algorithm[?]

Input: Global learning rate η;
Initial parameter $\boldsymbol{\theta}$;
Small constant ϵ, perhaps 10^{-7} for numerical stability.

Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
while *stopping criterion not met* **do**
  Sample a minibatch of m samples from the training dataset with corresponding targets: $\left\{\boldsymbol{x^{(i)}}, y^{(i)}\right\}_{i=1}^{m}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \mathcal{L}\big(f\left(\boldsymbol{x^{(i)}}; \boldsymbol{\theta}\right), y^{(i)}\big)$
  Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
  Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\eta}{\sqrt{\boldsymbol{r}}+\epsilon} \odot \boldsymbol{g}$ (Division and square root applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end

算法解释

- **算法优点：**特别适合**稀疏数据**（sparse data），因为每一步迭代更新的过程中，调整了学习率，使得不频繁的参数（infrequent parameter）得到更大的更新，频繁的参数（frequent parameter）得到更小的更新。
  - **公式解释** 以 Logistic Regression 为例，迭代更新公式 (为了方便不考虑正则化项) 如下：

$$w_j \leftarrow w_j + \eta\left(\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \widehat{y}^{(i)})x_j^{(i)}\right)$$

    当数据集的第 j 个维度总为 0（稀疏）时，$g_j = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \widehat{y}^{(i)})x_j^{(i)}$ 的取值非常小，所以为了让参数 w_j 得到足够充分的更新，需要学习率 η 非常大，但此时别的特征维度上数据稠密的话，$g_j = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \widehat{y}^{(i)})x_j^{(i)}$ 的取值就非常大，因此会发散！
  - **解决办法** Adagrad 的解决办法是**独立地累积每一个维度上梯度的平方**，在第 t 个 step 任意维度 j 上的梯度平方累积为：

$$r_j = \sum_{t'=1}^{t} g_{j,t'}^2$$

    * 维度 j 稀疏：

$$g_{j,t'}^2\text{小} \quad \Rightarrow \quad r_j = \sum_{t'=1}^{t} g_{j,t'}^2\text{小} \quad \Rightarrow \quad \sqrt{r_j}+\epsilon\text{小} \quad \Rightarrow \quad \underbrace{\frac{\eta}{\sqrt{r_j}+\epsilon}}_{\text{实际学习率}}\text{大}$$

    * 维度 j 稠密：

$$g_{j,t'}^2\text{大} \quad \Rightarrow \quad r_j = \sum_{t'=1}^{t} g_{j,t'}^2\text{大} \quad \Rightarrow \quad \sqrt{r_j}+\epsilon\text{大} \quad \Rightarrow \quad \underbrace{\frac{\eta}{\sqrt{r_j}+\epsilon}}_{\text{实际学习率}}\text{小}$$

所以实现了学习率的自适应调节。

- **算法缺点：**训练中后期学习率为 0，训不动了，或者直接使得训练提前结束。因此针对这个问题，后期又有了更好的优化算法（AdaDelta, RMSProp, Adam$\cdots$）。

4.3.5 AdaDelta

Algorithm 5: The Adadelta algorithm[**?**]

Input: Global learning rate η,;
Decay rate ρ;
Initial parameter $\boldsymbol{\theta}_0$;
Small constant ϵ, usually 10^{-8} used to stabilize division by small numbers.

Initialize global step counter $t \leftarrow 0$
Initialize squared gradient accumulation variable $\boldsymbol{r}_t = \mathbf{0}$
Initialize squared update of parameter accumulation variable $\boldsymbol{s}_t = \mathbf{0}$
while *stopping criterion not met* **do**
  Update global step counter $t \leftarrow t + 1$
  Sample a minibatch of m samples from the training dataset with corresponding targets: $\left\{\boldsymbol{x^{(i)}}, y^{(i)}\right\}_{i=1}^{m}$.
  Compute gradient: $\boldsymbol{g}_t \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\boldsymbol{x^{(i)}}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  Accumulate squared gradient: $\boldsymbol{r}_t \leftarrow \rho \boldsymbol{r}_{t-1} + (1-\rho)\boldsymbol{g}_t \odot \boldsymbol{g}_t$
  Compute update: $\Delta\boldsymbol{\theta}_t \leftarrow -\eta \frac{\sqrt{\boldsymbol{s}_{t-1}+\epsilon}}{\sqrt{\boldsymbol{r}_t+\epsilon}} \odot \boldsymbol{g}_t$ (Division and square root applied element-wise)
  Accumulate update: $\boldsymbol{s}_t \leftarrow \rho \boldsymbol{s}_{t-1} + (1-\rho)\boldsymbol{\theta}_t \odot \boldsymbol{\theta}_t$
  Apply update: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_t$
end

算法解释

- Adadelta 算法中 Global learning rate η 实质为一个 smoothing factor.
- 当先前累计的参数更新量大时，当前 step 的参数更新值就大。
- 当先前累计的梯度大时（说明可学习样本充分），当前 step 的参数更新值就小。

4.3.6 RMSProp

Algorithm 6: The RMSProp algorithm[**?**]

Input: Global learning rate η, decay rate ρ;
Initial parameter $\boldsymbol{\theta}$;
Small constant ϵ, usually 10^{-10} used to stabilize division by small numbers.

Initialize squared gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
while *stopping criterion not met* **do**
  Sample a minibatch of m samples from the training dataset with corresponding targets: $\left\{\boldsymbol{x^{(i)}}, y^{(i)}\right\}_{i=1}^{m}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\boldsymbol{x^{(i)}}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$
  Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\eta}{\sqrt{\boldsymbol{r}}+\epsilon} \odot \boldsymbol{g}$ (Division and square root applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end

算法解释

- 只累计之前最新的 $\frac{1}{1-\rho}$ 个梯度的平方，因此 r_j 不会一直增大，同时同一维度在时间 step 上实现学习率的自适应调节。
- RMSProp 算法相对于 AdaGrad 算法，其将二阶梯度累积量由：

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$$

  替换为：

$$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$$

4.3.7 Adam

Algorithm 7: The Adam algorithm[?]

Input: Global learning rate η (Suggested default: 0.001);
Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$ (Suggested defaults: 0.9 and 0.999 respectively);
Small constant ϵ used for numcerical stabilization (Suggested default: 10^{-8}); Initial parameter $\boldsymbol{\theta}$.

Initialize 1st and 2nd moment variables: $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
Initialize time step $t = 0$
while *stopping criterion not met* **do**
  Sample a minibatch of m samples with corresponding targets: $\left\{\boldsymbol{x^{(i)}}, y^{(i)}\right\}_{i=1}^{m}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\boldsymbol{x^{(i)}}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  $t \leftarrow t + 1$
  Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
  Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
  Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
  Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
  Compute update: $\Delta\boldsymbol{\theta} = -\eta \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\epsilon}$ (operations applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end

算法解释

- 引入**一阶动量**是为了调整优化方向。
- 引入**二阶动量**是为了调整优化步长。
- **把一阶动量和二阶动量都用起来**，同时做了修正，那这就是 Adam(adaptive + momentum) 算法了。

TensorFlow Implement

```
# Copyright (C) 2020 Tong Jia. All rights reserved.
from tensorflow.python.eager import context
from tensorflow.python.framework import ops
from tensorflow.python.ops import control_flow_ops
from tensorflow.python.ops import math_ops
from tensorflow.python.ops import resource_variable_ops
from tensorflow.python.ops import state_ops
from tensorflow.python.training import optimizer
from tensorflow.python.training import training_ops


class AdamOptimizer(optimizer.Optimizer):
    """Optimizer that implements the Adam algorithm.
```

```
  References:
    Adam - A Method for Stochastic Optimization:
      [Kingma et al., 2015](https://arxiv.org/abs/1412.6980)
      ([pdf](https://arxiv.org/pdf/1412.6980.pdf))
  """

  def __init__(self,
               learning_rate=0.001,
               beta1=0.9,
               beta2=0.999,
               epsilon=1e-8,
               use_locking=False,
               name="Adam"):
    r"""Construct a new Adam optimizer.
    Initialization:
    $$m_0 := 0 \text{(Initialize initial 1st moment vector)}$$
    $$v_0 := 0 \text{(Initialize initial 2nd moment vector)}$$
    $$t := 0 \text{(Initialize timestep)}$$
    The update rule for `variable` with gradient `g` uses an optimization
    described at the end of section 2 of the paper:
    $$t := t + 1$$
    $$\text{lr}_t := \mathrm{learning_rate} *
      \sqrt{1 - \beta_2^t} / (1 - \beta_1^t)$$
    $$m_t := \beta_1 * m_{t-1} + (1 - \beta_1) * g$$
    $$v_t := \beta_2 * v_{t-1} + (1 - \beta_2) * g * g$$
    $$\text{variable} := \text{variable} -
      \text{lr}_t * m_t / (\sqrt{v_t} + \epsilon)$$
    The default value of 1e-8 for epsilon might not be a good default in
    general. For example, when training an Inception network on ImageNet a
    current good choice is 1.0 or 0.1. Note that since AdamOptimizer uses the
    formulation just before Section 2.1 of the Kingma and Ba paper rather than
    the formulation in Algorithm 1, the "epsilon" referred to here is "epsilon
    hat" in the paper.
    The sparse implementation of this algorithm (used when the gradient is an
    IndexedSlices object, typically because of `tf.gather` or an embedding
    lookup in the forward pass) does apply momentum to variable slices even if
    they were not used in the forward pass (meaning they have a gradient equal
    to zero). Momentum decay (beta1) is also applied to the entire momentum
    accumulator. This means that the sparse behavior is equivalent to the dense
    behavior (in contrast to some momentum implementations which ignore momentum
    unless a variable slice was actually used).
    Args:
      learning_rate: A Tensor or a floating point value. The learning rate.
      beta1: A float value or a constant float tensor. The exponential decay
        rate for the 1st moment estimates.
      beta2: A float value or a constant float tensor. The exponential decay
        rate for the 2nd moment estimates.
      epsilon: A small constant for numerical stability. This epsilon is
        "epsilon hat" in the Kingma and Ba paper (in the formula just before
        Section 2.1), not the epsilon in Algorithm 1 of the paper.
      use_locking: If True use locks for update operations.
      name: Optional name for the operations created when applying gradients.
```

```
        Defaults to "Adam". @compatibility(eager) When eager execution is
        enabled, `learning_rate`, `beta1`, `beta2`, and `epsilon` can each be a
        callable that takes no arguments and returns the actual value to use.
        This can be useful for changing these values across different
        invocations of optimizer functions. @end_compatibility
    """
    super(AdamOptimizer, self).__init__(use_locking, name)
    self._lr =learning_rate
    self._beta1 =beta1
    self._beta2 =beta2
    self._epsilon =epsilon

    # Tensor versions of the constructor arguments, created in _prepare().
    self._lr_t =None
    self._beta1_t =None
    self._beta2_t =None
    self._epsilon_t =None

def _get_beta_accumulators(self):
    with ops.init_scope():
        if context.executing_eagerly():
            graph =None
        else:
            graph =ops.get_default_graph()
        return (self._get_non_slot_variable("beta1_power", graph=graph),
                self._get_non_slot_variable("beta2_power", graph=graph))

def _create_slots(self, var_list):
    # Create the beta1 and beta2 accumulators on the same device as the first
    # variable. Sort the var_list to make sure this device is consistent across
    # workers (these need to go on the same PS, otherwise some updates are
    # silently ignored).
    first_var =min(var_list, key=lambda x: x.name)
    self._create_non_slot_variable(
        initial_value=self._beta1, name="beta1_power", colocate_with=first_var)
    self._create_non_slot_variable(
        initial_value=self._beta2, name="beta2_power", colocate_with=first_var)

    # Create slots for the first and second moments.
    for v in var_list:
        self._zeros_slot(v, "m", self._name)
        self._zeros_slot(v, "v", self._name)

def _prepare(self):
    lr =self._call_if_callable(self._lr)
    beta1 =self._call_if_callable(self._beta1)
    beta2 =self._call_if_callable(self._beta2)
    epsilon =self._call_if_callable(self._epsilon)

    self._lr_t =ops.convert_to_tensor(lr, name="learning_rate")
    self._beta1_t =ops.convert_to_tensor(beta1, name="beta1")
    self._beta2_t =ops.convert_to_tensor(beta2, name="beta2")
```

```
    self._epsilon_t =ops.convert_to_tensor(epsilon, name="epsilon")

def _apply_dense(self, grad, var):
    m =self.get_slot(var, "m")
    v =self.get_slot(var, "v")
    beta1_power, beta2_power =self._get_beta_accumulators()
    return training_ops.apply_adam(
        var,
        m,
        v,
        math_ops.cast(beta1_power, var.dtype.base_dtype),
        math_ops.cast(beta2_power, var.dtype.base_dtype),
        math_ops.cast(self._lr_t, var.dtype.base_dtype),
        math_ops.cast(self._beta1_t, var.dtype.base_dtype),
        math_ops.cast(self._beta2_t, var.dtype.base_dtype),
        math_ops.cast(self._epsilon_t, var.dtype.base_dtype),
        grad,
        use_locking=self._use_locking).op

def _resource_apply_dense(self, grad, var):
    m =self.get_slot(var, "m")
    v =self.get_slot(var, "v")
    beta1_power, beta2_power =self._get_beta_accumulators()
    return training_ops.resource_apply_adam(
        var.handle,
        m.handle,
        v.handle,
        math_ops.cast(beta1_power, grad.dtype.base_dtype),
        math_ops.cast(beta2_power, grad.dtype.base_dtype),
        math_ops.cast(self._lr_t, grad.dtype.base_dtype),
        math_ops.cast(self._beta1_t, grad.dtype.base_dtype),
        math_ops.cast(self._beta2_t, grad.dtype.base_dtype),
        math_ops.cast(self._epsilon_t, grad.dtype.base_dtype),
        grad,
        use_locking=self._use_locking)

def _apply_sparse_shared(self, grad, var, indices, scatter_add):
    beta1_power, beta2_power =self._get_beta_accumulators()
    beta1_power =math_ops.cast(beta1_power, var.dtype.base_dtype)
    beta2_power =math_ops.cast(beta2_power, var.dtype.base_dtype)
    lr_t =math_ops.cast(self._lr_t, var.dtype.base_dtype)
    beta1_t =math_ops.cast(self._beta1_t, var.dtype.base_dtype)
    beta2_t =math_ops.cast(self._beta2_t, var.dtype.base_dtype)
    epsilon_t =math_ops.cast(self._epsilon_t, var.dtype.base_dtype)
    lr =(lr_t *math_ops.sqrt(1 -beta2_power) /(1 -beta1_power))
    # m_t = beta1 * m + (1 - beta1) * g_t
    m =self.get_slot(var, "m")
    m_scaled_g_values =grad *(1 -beta1_t)
    m_t =state_ops.assign(m, m *beta1_t, use_locking=self._use_locking)
    with ops.control_dependencies([m_t]):
        m_t =scatter_add(m, indices, m_scaled_g_values)
    # v_t = beta2 * v + (1 - beta2) * (g_t * g_t)
```

```
        v = self.get_slot(var, "v")
        v_scaled_g_values =(grad *grad) *(1 -beta2_t)
        v_t =state_ops.assign(v, v *beta2_t, use_locking=self._use_locking)
        with ops.control_dependencies([v_t]):
            v_t =scatter_add(v, indices, v_scaled_g_values)
        v_sqrt =math_ops.sqrt(v_t)
        var_update =state_ops.assign_sub(
            var, lr *m_t /(v_sqrt +epsilon_t), use_locking=self._use_locking)
        return control_flow_ops.group(*[var_update, m_t, v_t])

    def _apply_sparse(self, grad, var):
        return self._apply_sparse_shared(
            grad.values,
            var,
            grad.indices,
            lambda x, i, v: state_ops.scatter_add( # pylint: disable=g-long-lambda
                x,
                i,
                v,
                use_locking=self._use_locking))

    def _resource_scatter_add(self, x, i, v):
        with ops.control_dependencies(
                [resource_variable_ops.resource_scatter_add(x.handle, i, v)]):
            return x.value()

    def _resource_apply_sparse(self, grad, var, indices):
        return self._apply_sparse_shared(grad, var, indices,
                                         self._resource_scatter_add)

    def _finish(self, update_ops, name_scope):
        # Update the power accumulators.
        with ops.control_dependencies(update_ops):
            beta1_power, beta2_power =self._get_beta_accumulators()
            with ops.colocate_with(beta1_power):
                update_beta1 =beta1_power.assign(
                    beta1_power *self._beta1_t, use_locking=self._use_locking)
                update_beta2 =beta2_power.assign(
                    beta2_power *self._beta2_t, use_locking=self._use_locking)
        return control_flow_ops.group(
            *update_ops +[update_beta1, update_beta2], name=name_scope)
```

Torch Implement

```
# Copyright (C) 2020 Tong Jia. All rights reserved.
import math
import torch
from torch.optim.optimizer import Optimizer
```

```
class Adam(Optimizer):
    r"""Implements Adam algorithm.
    It has been proposed in `Adam: A Method for Stochastic Optimization`_.
    Arguments:
        params (iterable): iterable of parameters to optimize or dicts defining
            parameter groups
        lr (float, optional): learning rate (default: 1e-3)
        betas (Tuple[float, float], optional): coefficients used for computing
            running averages of gradient and its square (default: (0.9, 0.999))
        eps (float, optional): term added to the denominator to improve
            numerical stability (default: 1e-8)
        weight_decay (float, optional): weight decay (L2 penalty) (default: 0)
        amsgrad (boolean, optional): whether to use the AMSGrad variant of this
            algorithm from the paper `On the Convergence of Adam and Beyond`_
            (default: False)
    .. _Adam\: A Method for Stochastic Optimization:
        https://arxiv.org/abs/1412.6980
    .. _On the Convergence of Adam and Beyond:
        https://openreview.net/forum?id=ryQu7f-RZ
    """

    def __init__(self, params, lr=1e-3, betas=(0.9, 0.999), eps=1e-8,
                 weight_decay=0, amsgrad=False):
        if not 0.0 <=lr:
            raise ValueError("Invalid learning rate: {}".format(lr))
        if not 0.0 <=eps:
            raise ValueError("Invalid epsilon value: {}".format(eps))
        if not 0.0 <=betas[0] <1.0:
            raise ValueError("Invalid beta parameter at index 0: {}".format(betas[0]))
        if not 0.0 <=betas[1] <1.0:
            raise ValueError("Invalid beta parameter at index 1: {}".format(betas[1]))
        defaults =dict(lr=lr, betas=betas, eps=eps,
                       weight_decay=weight_decay, amsgrad=amsgrad)
        super(Adam, self).__init__(params, defaults)

    def __setstate__(self, state):
        super(Adam, self).__setstate__(state)
        for group in self.param_groups:
            group.setdefault('amsgrad', False)

    def step(self, closure=None):
        """Performs a single optimization step.
        Arguments:
            closure (callable, optional): A closure that reevaluates the model
                and returns the loss.
        """
        loss =None
        if closure is not None:
            loss =closure()

        for group in self.param_groups:
            for p in group['params']:
```

```
            if p.grad is None:
                continue
            grad =p.grad.data
            if grad.is_sparse:
                raise RuntimeError('Adam does not support sparse gradients, please consider SparseAdam
                                                                    instead')
            amsgrad =group['amsgrad']

            state =self.state[p]

            # State initialization
            if len(state) ==0:
                state['step'] =0
                # Exponential moving average of gradient values
                state['exp_avg'] =torch.zeros_like(p.data, memory_format=torch.preserve_format)
                # Exponential moving average of squared gradient values
                state['exp_avg_sq'] =torch.zeros_like(p.data, memory_format=torch.preserve_format)
                if amsgrad:
                    # Maintains max of all exp. moving avg. of sq. grad. values
                    state['max_exp_avg_sq'] =torch.zeros_like(p.data, memory_format=torch.preserve_format)

            exp_avg, exp_avg_sq =state['exp_avg'], state['exp_avg_sq']
            if amsgrad:
                max_exp_avg_sq =state['max_exp_avg_sq']
            beta1, beta2 =group['betas']

            state['step'] +=1
            bias_correction1 =1 -beta1 **state['step']
            bias_correction2 =1 -beta2 **state['step']

            if group['weight_decay'] !=0:
                grad =grad.add(group['weight_decay'], p.data)

            # Decay the first and second moment running average coefficient
            exp_avg.mul_(beta1).add_(1 -beta1, grad)
            exp_avg_sq.mul_(beta2).addcmul_(1 -beta2, grad, grad)
            if amsgrad:
                # Maintains the maximum of all 2nd moment running avg. till now
                torch.max(max_exp_avg_sq, exp_avg_sq, out=max_exp_avg_sq)
                # Use the max. for normalizing running avg. of gradient
                denom =(max_exp_avg_sq.sqrt() /math.sqrt(bias_correction2)).add_(group['eps'])
            else:
                denom =(exp_avg_sq.sqrt() /math.sqrt(bias_correction2)).add_(group['eps'])

            step_size =group['lr'] /bias_correction1

            p.data.addcdiv_(-step_size, exp_avg, denom)

    return loss
```

4.3.8 AdaBound

Algorithm 8: The AdaBound algorithm[**?**]

Input: Global learning rate α (Suggested default: 0.001);
Lower bound function for learning rate clipping $\eta_l(t)$ (t is the time step);
Upper bound function for learning rate clipping $\eta_u(t)$;
Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$ (Suggested defaults: 0.9 and 0.999 respectively);
Small constant ϵ used for numcerical stabilization (Suggested default: 10^{-8}); Initial parameter $\boldsymbol{\theta}$.

Initialize 1st and 2nd moment variables: $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
Initialize time step $t = 0$
while *stopping criterion not met* **do**
  Sample a minibatch of m samples with corresponding targets: $\left\{\boldsymbol{x^{(i)}}, y^{(i)}\right\}_{i=1}^{m}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \mathrm{L}\left(f\left(\boldsymbol{x^{(i)}}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  $t \leftarrow t + 1$
  Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
  Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
  Clip learning rate $\frac{\alpha}{\sqrt{\boldsymbol{r}}+\epsilon}$ by $\eta_l(t)$ and $\eta_u(t)$ and correct it:

$$\hat{\boldsymbol{\eta}} \leftarrow \mathrm{Clip}\left(\frac{\alpha}{\sqrt{\boldsymbol{r}} + \epsilon}, \eta_l(t), \eta_u(t)\right), \quad \boldsymbol{\eta} \leftarrow \frac{\hat{\boldsymbol{\eta}}}{\sqrt{t}}$$

  Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{\eta} \odot \boldsymbol{s}$ (operations applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end

算法解释

- AdaBound 的核心在于**学习率剪裁 (learning rate clipping)**
- Adam 优化算法虽然比 SGD 更快，但存在两个重大缺陷：
  - **结果不收敛**
  - **找不到全局最优点**

  而造成这两大缺陷的原因为：
  - 不稳定的学习率（e.g. 某一维度上一个 time step 为 1000，下一个 time step 为 0.1）
  - 极端学习率（e.g. 某一个 time step 时，一个维度为 1000，另一个维度为 0.1）

  因此 AdaBound 解决这个问题的方式是将任意时刻 t 的学习率限制在 $\eta_l(t)$ 和 $\eta_u(t)$ 之间
- 因此希望优化算法在训练前期像 Adam 一样快速，后期像 SGD 一样效果好，而：
  - 当 $\eta_l(t) = \alpha$ 且 $\eta_u(t) = \alpha$ 时：优化算法即为 SGD

  - 当 $\eta_l(t)=0$ 且 $\eta_u(t)=\infty$ 时：优化算法即为 Adam

  因此为了实现算法从 Adam 到 SGD 的平滑过度，则将 $\eta_l(t)$ 和 $\eta_u(t)$ 变为随时间变化的函数：

  - lower bound $\eta_l(t)$：从 0 到 α 的递增
  - upper bound $\eta_u(t)$：从 ∞ 到 α 的递减

  在这种情况下，AdaBound 开始时像 Adam 一样训练速度很快，随着学习率边界越来越受到限制，它又逐渐转变为 SGD

- $\eta_l(t)$ 从 0 到 α 的具体递增策略，以及 $\eta_u(t)$ 从 ∞ 到 α 的具体递减策略可以人为设置
- AdaBound 的优点：对超参数（e.g. global learning rate α）不是很敏感，省去了大量调参的时间

4.3.9 RAdam（未完成）

Algorithm 9: Rectified Adam. All operations are element-wise[**?**].

Input: Learning rate η;
Momentum parameter α;
Initial parameter $\boldsymbol{\theta}$;
Initial velocity $\boldsymbol{v}$.

while *stopping criterion not met* **do**

end

4.3.10 SWATS: Improving Generalization Performance by Switching from Adam to SGD

Algorithm 10: SWATS[?]

Input: Objective function L;
Initial parameter $\boldsymbol{\theta}_0$;
Learning rate $\eta = 10^{-3}$, accumulator coefficients $(\beta_1, \beta_2) = (0.9, 0.999)$;
Small constant $\epsilon = 10^{-9}$ used to stabilize division by small numbers, phase = Adam.

Initialize global step counter $t \leftarrow 0$, velocity for SGDM $\boldsymbol{v}_t \leftarrow \mathbf{0}$, 1st and 2nd variables for Adam $\boldsymbol{m}_t \leftarrow \mathbf{0},\ \boldsymbol{a}_t \leftarrow \mathbf{0}$, exponential average value $\lambda_t \leftarrow 0$.
while *stopping criterion not met* **do**
  Update global step counter: $t = t + 1$
  Sample a minibatch of m samples from the training dataset with corresponding targets: $\left\{\boldsymbol{x^{(i)}}, y^{(i)}\right\}_{i=1}^{m}$.
  Compute gradients: $\boldsymbol{g}_t \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} \mathrm{L}\big(f\left(\boldsymbol{x^{(i)}}; \boldsymbol{\theta}_{t-1}\right), y^{(i)}\big)$
  if *phase = SGD* **then**
    Compute velocity update: $\boldsymbol{v}_t = \beta_1 \boldsymbol{v}_{t-1} - \boldsymbol{g}_t$ (Note there doesn't exist η in front of $\boldsymbol{g}_t$)
    Update parameter: $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + (1-\beta_1)\Lambda \boldsymbol{v}_t$
    continue
  end
  Update 1st variables for Adam: $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$
  Update 2nd variables for Adam: $\boldsymbol{a}_t = \beta_2 \boldsymbol{a}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$
  Correct bias of 1st and 2nd variables for Adam: $\widehat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-\beta_1^t}$
  Correct bias of 2nd variables for Adam: $\widehat{\boldsymbol{a}}_t = \frac{\boldsymbol{a}_t}{1-\beta_2^t}$
  Compute update of Adam: $\boldsymbol{p}_t = -\eta \frac{\widehat{\boldsymbol{m}}_t}{\sqrt{\widehat{\boldsymbol{a}}_t} + \epsilon}$ (element-wise op)
  if $\boldsymbol{p}_t^T \boldsymbol{g}_t \neq 0$ **then**
    $\gamma_t = \frac{\boldsymbol{p}_t^{\mathrm{T}} \boldsymbol{p}_t}{-\boldsymbol{p}_t^{\mathrm{T}} \boldsymbol{g}_t} \in \mathbb{R}, \quad \lambda_t = \beta_2 \lambda_{t-1} + (1-\beta_2)\gamma_t$
    if $t > 1$ *and* $|\frac{\lambda_t}{(1-\beta_2^t)} - \gamma_t| < \epsilon$ **then**
      phase = SGD
      $\boldsymbol{v}_t = \mathbf{0}$, $\Lambda = \frac{\lambda_t}{1-\beta_2^t} \in \mathbb{R}$
    end
  else
    $\lambda_t = \lambda_{t-1}$
  end
end

算法解释 论文表述了存在的两个问题：

- **Problem-1:**
  何时切换优化算法 Adam ⇒ SGD？疑难点在于：切换太晚，Adam 可能已经跑到自己的盆地去了，SGD 再怎么好也跑不出来了；切换太早，影响收敛速度和效果。
- **Problem-2:**

Adam ⇒ SGD 切换算法后使用什么样的学习率 η^{SGD}？疑难点在于：Adam 使用自适应学习率，依赖的是二阶动量的积累，SGD 接着训练的话，很难找到一个很好的衔接学习率。

解答如下：

- **Solution-1:**
- **Solution-2:**

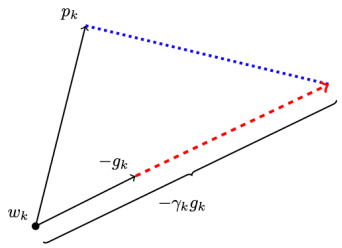


图 30: Illustrating the learning rate for SGD (γ_t) estimated by our proposed projection given an iterate $\boldsymbol{\theta_t}$, a stochastic gradient $\boldsymbol{g}_t$ and the Adam step $\boldsymbol{p}_t$

- Adam 的更新方向：$\boldsymbol{\theta}_t^{\text{Adam}} = -\frac{\eta\widehat{\boldsymbol{m}}_t}{\sqrt{\widehat{\boldsymbol{a}}_t}+\epsilon}$
- SGD 的更新方向：$\boldsymbol{\theta}_t^{\text{SGD}} = -\eta^{\text{SGD}}\boldsymbol{g}_t$

如果 SGD 要接过 Adam 的大旗，那么需要先后做两件事情：

- 首先沿着 Adam 的更新方向 $\boldsymbol{\theta}_t^{\text{Adam}}$ 走一步老路
- 而后沿着 $\boldsymbol{\theta}_t^{\text{Adam}}$ 的正交方向走一步属于自己的新路

那么用数学语言表示，即：

- 将 $\boldsymbol{\theta}_t^{\text{SGD}}$ 分解为 $\boldsymbol{\theta}_t^{\text{Adam}}$ 方向和其正交方向两个方向
- $\boldsymbol{\theta}_t^{\text{SGD}}$ 在 $\boldsymbol{\theta}_t^{\text{Adam}}$ 方向上的投影，应该正好等于 $\boldsymbol{\theta}_t^{\text{Adam}}$

方程描述如下：

$$
\begin{aligned}
& \operatorname{proj}\boldsymbol{\theta}_t^{\text{SGD}} = \boldsymbol{\theta}_t^{\text{Adam}} \\
\Rightarrow \quad & \frac{(\boldsymbol{\theta}_t^{\text{SGD}})^{\text{T}} \odot \boldsymbol{\theta}_t^{\text{Adam}}}{|\boldsymbol{\theta}_t^{\text{Adam}}|} = \boldsymbol{\theta}_t^{\text{Adam}} \\
\Rightarrow \quad & \frac{\left((\boldsymbol{\theta}_t^{\text{SGD}})^{\text{T}} \odot \boldsymbol{\theta}_t^{\text{Adam}}\right)^{\text{T}}\left((\boldsymbol{\theta}_t^{\text{SGD}})^{\text{T}} \odot \boldsymbol{\theta}_t^{\text{Adam}}\right)}{(\boldsymbol{\theta}_t^{\text{Adam}})^{\text{T}}\boldsymbol{\theta}_t^{\text{Adam}}}
\end{aligned}
$$

4.4 Online Learning Optimization Algorithms

4.4.1 FTRL（未完成）

4.5 Constrained Optimization Algorithms（未完成）

4.5.1 Proxy-Lagrangian Optimization

本节选自论文Two-Player Games for Efficient Non-Convex Constrained Optimization[?] 以及Training Well-Generalizing Classifiers for Fairness Metrics and Other Data-Dependent Constraints[?]

4.6 Summary of Gradient Based Optimizers

1. **算法孰优孰劣尚无定论** 刚入门优先考虑两种方式：
   - SGD + Nesterov Momentum
   - Adam
2. **选择熟练的算法** 这样可以更熟练地利用经验进行调参。
3. **前期选择较大的 batch-size，后期选择较小的 batch-size**
   - 前期选择较大的 batch-size：为了梯度稳定，少走弯路
   - 后期选择较小的 batch-size：为了利用梯度的震荡性，更有可能跳出鞍点
4. **充分了解数据** 如果数据是非常稀疏的，那么**优先考虑自适应学习率**的算法如：
   - AdaGrad
   - AdaDelta
   - RMSprop
   - Adam
5. **根据需求来选择**
   - 模型设计实验阶段：因为需要快速验证新模型的效果，所以可先用 Adam 算法进行快速实验
   - 模型上线或待发布阶段：因为需要追求极致的效果，所以可用精调的 SGD 进行模型的极致优化
6. **先用小数据集进行实验** 有论文研究指出，优化算法的收敛速度和数据集的大小关系不大。因此可以先用一个具有代表性的小数据集进行实验，测试一下最好的优化算法，并通过参数搜索寻找最优的优化超参数。
7. **考虑不同算法的组合** 先用 Adam 进行快速下降，而后切换到 SGD 进行充分的调优（有 paper 说明[?]）。
8. **数据集一定要充分打散** 这样在使用自适应算法时候，可以避免学习过度，学习不足，使得下降方向出现偏差的问题。
9. **训练过程中需要持续监控训练集和验证集** 监控目标函数值，RMSE，AUC 等评价准则。其中：
   - 监控训练集：为了保障模型进行充分的训练，即下降方向正确，学习率足够高
   - 监控验证集：为了避免模型训练过拟合
10. **制定一个合适的学习率衰减策略** 可以使用**定期衰减策略**，比如：
    - 每过 N 个 epoch 就衰减一次
    - 利用精度或 AUC 等性能指标监控，当验证集上的指标不变或者下跌时，就降低学习率
11. **基于动量的优化算法不适用于梯度不稳定的情况，因此此时 RMSprop 为最佳算法** 譬如在 GAN、WGAN 等各种生成对抗网络模型的训练中，梯度十分不稳定，因此不要使用 Momentum, Nesterov 和 Adam 这样基于动量的优化算法，此时 RMSprop 拥有最佳效果

4.7 References

- What's up with Deep Learning optimizers since Adam?
- CSE599s: Online Learning, Spring 2014.

5 Evaluation Metrics

- 评价模型好坏并没有统一的准则
- 评价准则一般针对的是测试数据 (out-samples)，而不是训练数据 (in-samples)
- 具体应用场景中，牵扯多方利益:
  - 用户
  - 供应商/内容生产方
  - 产品运营 (平台)
- 总体而言，模型的评价准则包括三个层次：
  - **算法层面**
  - 企业层面
  - 用户层面

5.1 Regression Task

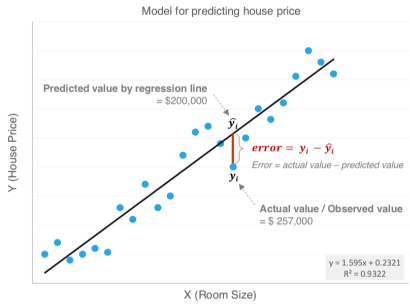


图 31: Error of regression task

5.1.1 Mean Absolute Error (MAE)

$$\mathrm{MAE}(f;\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\left|y^{(i)} - \hat{y}^{(i)}\right|$$

5.1.2 Mean Squared Error (MSE)

$$\mathrm{MSE}(f;\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

5.1.3 Root Mean Squared Error (RMSE)

$$\mathrm{RMSE}(f;\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\left(y^{(i)}-\hat{y}^{(i)}\right)^2}$$

5.1.4 Mean Absolute Percentage Error (MAPE)

$$\mathrm{MAPE}(f;\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|y^{(i)}-\hat{y}^{(i)}\right|}{\left|y^{(i)}\right|}$$

5.1.5 Coefficient of determination (R^2)

$$\begin{aligned}\mathrm{R}^2 &= \frac{\sum_{i=1}^{N}(y^{(i)}-\overline{y})^2}{\sum_{i=1}^{N}(y^{(i)}-\overline{y})^2+\sum_{i=1}^{N}(y^{(i)}-\hat{y}^{(i)})^2}\\ &= 1-\frac{\sum_{i=1}^{N}(y^{(i)}-\hat{y}^{(i)})^2}{\sum_{i=1}^{N}(y^{(i)}-\overline{y})^2+\sum_{i=1}^{N}(y^{(i)}-\hat{y}^{(i)})^2}\end{aligned}$$

5.2 Classification Task

5.2.1 Log Loss

$$-\sum_{i=1}^{N}\left(y^{(i)}\log\hat{y}^{(i)}+(1-y^{(i)})\log\left(1-\hat{y}^{(i)}\right)\right)$$

5.2.2 Confusion Matrix

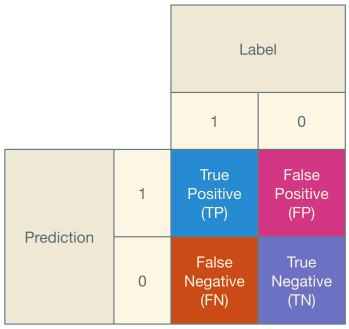


图 32: Confusion matrix for binary classification

- TP: 预测为正类（Positive），结果预测对了（True），即真实也为正类
- FP: 预测为正类（Positive），结果预测错了（False），即真实为负类
- FN: 预测为负类（Negative），结果预测错了（False），即真实为正类
- TN: 预测为负类（Negative），结果预测对了（True），即真实也为负类

5.2.3 Accuracy

$$\frac{\mathrm{TP}+\mathrm{TN}}{\#}$$

5.2.4 Precision

所有**预测**为正的样本中，有多少**真实**也是正样本

$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$$

5.2.5 Recall (Sensitivity)

所有**真实**为正的样本中，有多少**预测**也是正样本

$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$$

5.2.6 Specificity

所有**真实**为负的样本中，有多少**预测**也是负样本

$$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$$

5.2.7 F1-score

$$\frac{2}{\frac{1}{\mathrm{Precision}}+\frac{1}{\mathrm{Recall}}}$$

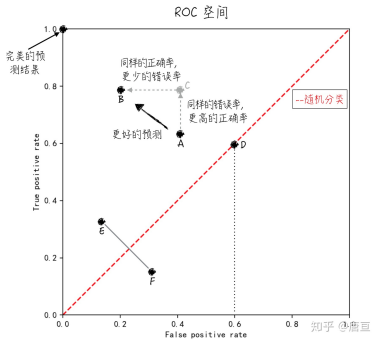
问题：何时使用何种评价准则？

- Accuracy
  - 正负样本数量相对均衡时 (猫狗识别样本包涵 55% 的猫和 45% 的狗)
  - 正负样本意义权重相当 (猫狗分类正样本猫和副样本狗，没有谁更被看中；而癌症的分类的就更看重正样本，不推荐 Accuracy)
- Precision, Recall 强调正样本
- Specificity 强调负样本
- F1-score 均衡 Precision 和 Recall

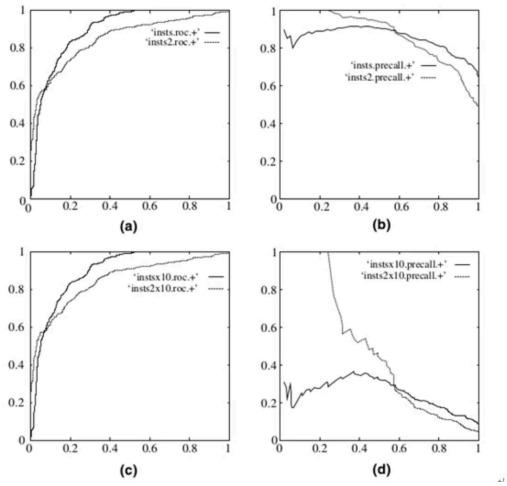
5.2.8 AUC & ROC

- ROC: 是一种显示分类模型在所有分类阈值（如 Logistic Regression 的阀值，不只是 0.5）下效果的曲线。该曲线的横坐标和纵坐标是两个内容：
  - TPR(True Positive Rate): $\mathrm{TPR}=\frac{\mathrm{TP}}{\mathrm{TP+FN}}$, 即召回率，值越大，在正样本上的准确率越高
  - FPR(False Positive Rate): $\mathrm{FPR}=\frac{\mathrm{FP}}{\mathrm{FP+TN}}$, 值越小，在负样本上的错误率越小，正确率越大

  TPR 越大越好，同时 FPR 越小越好，那么设置横轴为 FPR，纵轴为 TPR，那么曲线越靠近左上角点越好。
- AUC: 即 ROC 曲线下方的面积



问题：既然已经存在很多准则，为何还要使用 ROC 和 AUC？ 因为 ROC 曲线有个很好的特性：当测试集中的正负样本的分布变换的时候，ROC 曲线能够保持不变。在实际的数据集中经常会出现样本类不平衡，即正负样本比例差距较大，而且测试数据中的正负样本也可能随着时间变化。下图是 ROC 曲线和 Presision-Recall 曲线的对比：在图中:



- (a) 和 (c) 为 Roc 曲线
- (b) 和 (d) 为 Precision-Recall 曲线
- (a) 和 (b) 展示的是分类其在原始测试集 (正负样本分布平衡) 的结果
- (c) 和 (d) 是将测试集中负样本的数量增加到原来的 10 倍后，分类器的结果

可以明显的看出，ROC 曲线基本保持原貌，而 Precision-Recall 曲线变化较大。

AUC 的计算

5.3 Generative Adversarial Task（未完成）

参考自论文An empirical study on evaluation metrics of generative adversarial networks[?]

5.3.1 Inception Score (IS)（未完成）

Definition 5.1. Inception Score(IS) 的公式定义如下：

$$\mathrm{IS}(G) = \exp\left(\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_g}\left[D_{\mathrm{KL}}(p_{\mathcal{M}}(y|\boldsymbol{x})\|p_{\mathcal{M}}(y))\right]\right)$$

其中：

- $p_{\mathcal{M}}(y|\boldsymbol{x}) \in \mathbb{P}^K$: 条件标签概率分布（conditional class distribution）。即将样本 $\boldsymbol{x}$ 输入某分类判别模型 $\mathcal{M}$(e.g. softmax model) 得到的标签概率分布
- $p_{\mathcal{M}}(y)$: 边缘标签概率分布（marginal class distribution）。

$$p_{\mathcal{M}}(y) = \int_{\boldsymbol{x}} p(y|\boldsymbol{x}) d\mathbb{P}_g(\boldsymbol{x})$$

Reference

- A Note on the Inception Score[?]
- CSDN: 全面解析 Inception Score 原理及其局限性

5.3.2 Fréchet Inception Distance (FID)

Definition 5.2. Fréchet Inception Distance(FID) 的公式定义如下：

Reference

- GANs trained by a two time-scale update rule converge to a local Nash equilibrium[?]

5.3.3 Mode Score (MS)

5.3.4 Kernel Maximum Mean Discrepancy (Kernel MMD)

5.3.5 Wasserstein Distance (WD)

6 Model, Feature & Hyper-parameters Selections

6.1 Model Selection

6.1.1 Overfitting & Underfitting

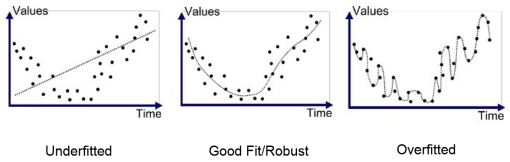


图 33: Underfitting and overfitting

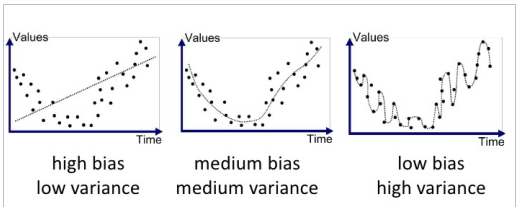


图 34: Underfitting and overfitting correspond to bias and variance

实际中遇到的例子：销量预测，单品的时间序列预测往往不准，因为 high-variance；按类别的销量预测结果会好，因为 low-variance.

6.1.2 Bias-Variance Decomposition

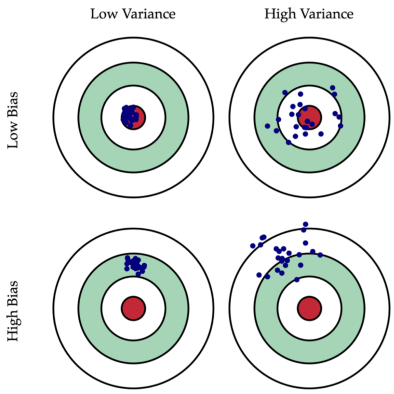


图 35: Bias vs Variance

Introduction Bias 和 Variance 是理解 underfitting 和 overfitting 的一种方式。模型在**训练样本**上的误差可以分解为如下几个部分：

- Bias: The part of the models can't fit the data
- Variance: The part of the models could fit the data, but doesn't because it's hard to fit
- Noise: Randomly error can't be controlled

Proof 假定一个**真正的预测函数在我们预测前是已经存在的**（上帝创造出来放在那里，等着我们去发现的函数），也就是我们希望可以训练出的最完美的函数，记为：

$$y = f(\boldsymbol{x})$$

我们希望通过拟合现有训练数据集学到这个上帝函数 $f(\boldsymbol{x})$，训练数据集记作：

$$\mathcal{D} = \{(\boldsymbol{x}^{(i)}, t^{(i)})\}_{i=1}^{N}$$

注意其中 $t^{(i)}$ 是夹杂了随机噪声的标签：

$$t^{(i)} = y^{(i)} + \epsilon,\ \mathbb{E}(\epsilon) = 0 \quad \forall i \in \{1, ..., n\}$$

那么根据给定的训练数据集，我们希望训练出一个模型去近似拟合真实的上帝函数 $f(\boldsymbol{x})$（梦想还是要有的，万一实现了呢），记根据训练数据 $\mathcal{D}$ 训练出的模型函数为:

$$\phi = \phi(\boldsymbol{\theta}; \mathcal{D})$$

我们使用 **MSE 期望**表示模型在训练样本上的误差，即：

$$\begin{aligned}\text{error} &= \mathbb{E}\Big\{\frac{1}{N}\sum_{i=1}^{N}(t^{(i)} - \phi^{(i)})^2\Big\} \quad \text{（注意从训练数据中观测到的标签是 } t^{(i)}\text{，而不是 } y^{(i)}\text{）} \\ &= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\big\{(t^{(i)} - \phi^{(i)})^2\big\}\end{aligned}$$

问题：为什么是 MSE 的期望而不是 MSE？ 因为譬如模型为神经网络时，改变训练数据的顺序，都可能导致模型的预测函数变化，因此用于之后测试阶段时，对同样的测试样本会有不同的输出，所以一次的预测结果没有足够的说服力，只能多测几次，求期望。

那么对于任何一个**训练样本** $(\boldsymbol{x}^{(i)}, t^{(i)})\ \forall i \in \{1, ..., n\}$，为了符号表示的便捷性，不写索引记为 $(\boldsymbol{x}, t)$，其**期望平方误差**可以做如下拆解：

$$
\begin{aligned}
\mathbb{E}\{(t-\phi)^2\} &= \mathbb{E}\{(t-y+y-\phi)^2\} \quad (\textbf{trick1: 添加上帝函数值 y}) \\
&= \mathbb{E}\{(t-y)^2 + (y-\phi)^2 + 2(t-y)(y-\phi)\} \\
&= \mathbb{E}\{\underbrace{(t-y)}_{\epsilon}{}^2\} + \mathbb{E}\{(y-\phi)^2\} + 2\mathbb{E}\{(t-y)(y-\phi)\} \\
&= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y-\phi)^2\} + 2\mathbb{E}\{ty - y^2 - t\phi + y\phi\} \\
&= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y-\phi)^2\} + 2\Big(\mathbb{E}\{ty\} - \mathbb{E}\{y^2\} - \mathbb{E}\{t\phi\} + \mathbb{E}\{y\phi\}\Big) \\
\because \quad \mathbb{E}\{ty\} &= y\mathbb{E}\{t\} \quad (\because y \text{ 为上帝 label，是已知的常数}) \\
&= y\mathbb{E}\{(y+\epsilon)\} \quad (\because t = y + \epsilon) \\
&= y\mathbb{E}\{y\} + y\mathbb{E}\{\epsilon\} = y^2 \quad (\because \text{常数的期望为常数}) \\
\because \quad \mathbb{E}\{y^2\} &= y\mathbb{E}\{y\} \quad (\text{先提一个常数 y，之后再提另一个}) \\
&= y^2 \\
\because \quad \mathbb{E}\{t\phi\} &= \mathbb{E}\{(y+\epsilon)\phi\} \quad (\because t = y + \epsilon) \\
&= \mathbb{E}\{y\phi\} + \mathbb{E}\{\epsilon\phi\} \\
&= \mathbb{E}\{y\phi\} \quad (\because \epsilon \text{ 和 } \phi \text{ 相互独立}, \therefore \mathbb{E}\{\epsilon\phi\} = 0) \\
\therefore \quad \mathbb{E}\{(t-\phi)^2\} &= \mathbb{E}\{\epsilon^2\} + E\{(y-\phi)^2\} \quad (\textbf{trick2: 添加预测函数 } \phi \textbf{ 的期望 } E\{\phi\}) \\
&= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \mathbb{E}\{\phi\} + E\{\phi\} - \phi)^2\} \\
&= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \mathbb{E}\{\phi\})^2 + (\mathbb{E}\{\phi\} - \phi)^2 + 2(y - \mathbb{E}\{\phi\})(\mathbb{E}\{\phi\} - \phi)\} \\
&= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \mathbb{E}\{\phi\})^2\} + \mathbb{E}\{(\mathbb{E}\{\phi\} - \phi)^2\} + 2\mathbb{E}\{(y - \mathbb{E}\{\phi\})(\mathbb{E}\{\phi\} - \phi)\} \\
&= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \mathbb{E}\{\phi\})^2\} + \mathbb{E}\{(\mathbb{E}\{\phi\} - \phi)^2\} \\
&\quad + 2\mathbb{E}\{y\mathbb{E}\{\phi\} - (\mathbb{E}\{\phi\})^2 - y\phi + \mathbb{E}\{\phi\}\phi\} \\
&= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(y - \mathbb{E}\{\phi\})^2\} + \mathbb{E}\{(\mathbb{E}\{\phi\} - \phi)^2\} \\
&\quad + 2\Big(\underbrace{\mathbb{E}\{y\mathbb{E}\{\phi\}\} - \mathbb{E}\{(\mathbb{E}\{\phi\})^2\} - \mathbb{E}\{y\phi\} + \mathbb{E}\{\mathbb{E}\{\phi\}\phi\}}_{\text{该项为 } 0}\Big) \\
\because \quad \mathbb{E}\{y\mathbb{E}\{\phi\}\} &= y\mathbb{E}\{\mathbb{E}\{\phi\}\} \quad (\because y \text{ 为常数，可提出}) \\
&= y\mathbb{E}\{\phi\} \quad (\because \textbf{期望 } \mathbb{E}\{\phi\} \textbf{ 是常数}. \therefore \mathbb{E}\{\mathbb{E}\{\phi\}\} = \mathbb{E}\{\phi\}) \\
\mathbb{E}\{y\phi\} &= y\mathbb{E}\{\phi\} \\
\mathbb{E}\{(\mathbb{E}\{\phi\})^2\} &= \mathbb{E}\{\phi\}\mathbb{E}\{\mathbb{E}\{\phi\}\} \quad (\because \textbf{期望 } \mathbb{E}\{\phi\} \textbf{ 是常数}) \\
&= (\mathbb{E}\{\phi\})^2 \\
\mathbb{E}\{\mathbb{E}\{\phi\}\phi\} &= \mathbb{E}\{\phi\}\mathbb{E}\{\phi\} = (\mathbb{E}\{\phi\})^2 \\
\therefore \quad \mathbb{E}\{(t-\phi)^2\} &= \underbrace{\mathbb{E}\{\epsilon^2\}}_{\text{noise}} + \underbrace{\mathbb{E}\{(y - \mathbb{E}\{\phi\})^2\}}_{\text{Bias}} + \underbrace{\mathbb{E}\{(\mathbb{E}\{\phi\} - \phi)^2\}}_{\text{Variance}}
\end{aligned}
$$

所以最终模型在训练数据集上的误差被拆分成了偏差，方差和噪声三项内容。

Bias-variance decomposition note

- 第一次拆解：引入上帝 label: y
- 第二次拆解：引入预测函数的期望 $\mathbb{E}\{\phi\}$
- Bias-Variance Decomposition 是在**训练集上的错误分解，而不是测试集上的错误分解**

6.2 Feature Selection

特征选择依据思路不同，分为以下三类：

- **Filter**: 自变量和目标变量之间的关系
- **Wrapper**: 通过目标函数（譬如：mse，logloss）或评价准则（譬如：AUC）决定是否加入一个变量
- **Embedded**: 学习器自身自动选择特征

6.2.1 Filter methods

6.2.2 Wrapper methods

Note of wrapper methods 实际操作中具体的流程是（group ⇒ field）层次模式的：

1. group 级别测试（如：所有的天气 fields（最高温，最低温，PM2.5，天气类型）为一个天气 group）
   - 选定一个基础的 group G_{base}，测试模型在 G_{base} 上的模型效果
   - 添加一个候选 group $G_{\text{candidate}}$，测试模型在 $\{G_{\text{base}}, G_{\text{candidate}}\}$ 上的模型效果
2. field 级别测试
   - 对**同一**确定候选 group $G_{\text{candidate}}$ 内所有的**单个** field $f_{\text{candidate},i}$ 进行测试
   - 对**同一**确定候选 group $G_{\text{candidate}}$ 内所有的**组合** fields $\{f_{\text{candidate},i_1}, ..., f_{\text{candidate},i_m}\}$ 进行测试
   - 对**不同**候选 group 组合 $\{G_{\text{candidate}_1}, ..., G_{\text{candidate}_n}\}$ 内的所有**组合** fields $\{f_{\text{candidate}_1,i_1}, ..., f_{\text{candidate}_n,i_m}\}$ 进行测试

6.2.3 Embedded methods

6.3 Hyper-parameters Selection

6.3.1 Model validation strategies

- Hold-out validation
- K-fold cross validation
- Leave-one-out cross validation

Part II

Machine Learning Model and Theory

7 Logistic Regression

7.1 Logistic Regression

7.1.1 Model prediction

Logistic Regression 是一个用于**二分类问题**的线性模型，预测判别函数可表示为：

$$\phi(\boldsymbol{x};\boldsymbol{w}) = \frac{1}{1+\exp(-\sum_{j=1}^{d} w_j x_j)}$$

其中判别函数包括两部分：

1. 内函数 (linear function): $z = \boldsymbol{w}^T\boldsymbol{x} = \sum_{j=1}^{d} w_j x_j$
2. 外函数 (sigmoid function): $\phi(z) = \frac{1}{1+\exp(-z)}$
3. 合并表示：$\phi(\boldsymbol{x};\boldsymbol{w}) = \frac{1}{1+\exp(-\sum_{j=1}^{d} w_j x_j)}$

判别类别条件为：

$$h(\boldsymbol{x}) = \begin{cases} 0, & \text{if } \phi(\boldsymbol{x};\boldsymbol{w}) \le \alpha \\ 1, & \text{if } \phi(\boldsymbol{x};\boldsymbol{w}) > \alpha \end{cases}$$

其中 α 为判别罚值，一般有 $\alpha = 0.5$。

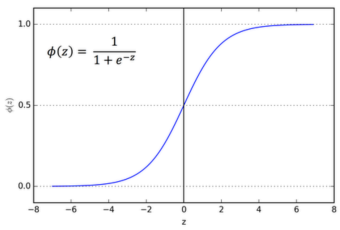


图 36: Sigmoid function

待学习参数 Logistic Regression 的待学习参数为：

$$\boldsymbol{w} \in \mathbb{R}^d$$

其中 w_j, $j \in \{1, ..., d\}$ 表示原始数据第 j 个维度上的权重。

7.1.2 Model training

Dataset 训练数据的表达如下：

- 训练数据集：$\mathcal{D} = \{(\boldsymbol{x^{(i)}}, y^{(i)})\}_{i=1}^{N}$
- 样本标签：$y^{(i)} \in \{0, 1\}$

Modeling details Logistic Regression 的判别函数包括两部分：

1. 内函数 (linear function): $z = \boldsymbol{w}^T\boldsymbol{x} = \sum_{j=1}^{d} w_j x_j$
2. 外函数 (sigmoid function): $\phi(z) = \frac{1}{1+\exp(-z)}$
3. 合并表示：$\phi(\boldsymbol{x};\boldsymbol{w}) = \frac{1}{1+\exp(-\sum_{j=1}^{d} w_j x_j)}$

Loss function 以下从两个角度推导 Logistic Regression 的**目标函数**（求梯度之前的所有建模步骤）：

- Maximum Likelihood Estimation
- Kullback–Leibler-Divergence

Maximum Likelihood Estimation 为了数学公式表达的简便，我们做如下符号记录：

- $p(\cdot)$: 事件 · 发生的概率
- 样本事件 $\boldsymbol{x^{(i)}}$ 发生（$y^{(i)} = 1$）的**估计概率**：

$$q^{(i)} = p(y^{(i)} = 1 \,|\, \boldsymbol{x^{(i)}})$$

- 样本事件 $\boldsymbol{x^{(i)}}$ 不发生（$y^{(i)} = 0$）的**估计概率**：

$$\begin{aligned} 1 - q^{(i)} &= p(y^{(i)} = 0 \,|\, \boldsymbol{x^{(i)}}) \\ &= 1 - p(y^{(i)} = 1 \,|\, \boldsymbol{x^{(i)}}) \end{aligned}$$

整合二者，利用 0/1 表示二分类标签的便利性，可得训练样本（带标签的样本事件）$(\boldsymbol{x^{(i)}}, y^{(i)})$ 出现的估计概率为：

$$(q^{(i)})^{y^{(i)}}(1 - q^{(i)})^{1-y^{(i)}}$$

假设所有的训练样本都是**独立同分布**（independently and identically distributed）的，那么所有训练样本都出现的概率为：

$$\prod_{i=1}^{N} \left((q^{(i)})^{y^{(i)}}(1 - q^{(i)})^{1-y^{(i)}}\right)$$

那么模型的优化目标就是最大化所有样本都出现的概率，即：

$$\begin{aligned} & \max \quad \prod_{i=1}^{N} \left((q^{(i)})^{y^{(i)}}(1 - q^{(i)})^{1-y^{(i)}}\right) \\ \Longrightarrow \quad & \max \quad \underbrace{\log\left(\prod_{i=1}^{N} \left((q^{(i)})^{y^{(i)}}(1 - q^{(i)})^{1-y^{(i)}}\right)\right)}_{\text{对目标函数取对数}} \\ \Longrightarrow \quad & \max \quad \sum_{i=1}^{N} \log\left((q^{(i)})^{y^{(i)}}(1 - q^{(i)})^{1-y^{(i)}}\right) \end{aligned}$$

$$
\begin{aligned}
\Longrightarrow \quad & \max \quad \sum_{i=1}^{N} \big(\log(q^{(i)})^{y^{(i)}} + \log(1-q^{(i)})^{1-y^{(i)}} \big) \\
\Longrightarrow \quad & \max \quad \sum_{i=1}^{N} \big(y^{(i)} \log(q^{(i)}) + (1-y^{(i)}) \log(1-q^{(i)}) \big) \\
\Longrightarrow \quad & \min \quad -\frac{1}{N} \sum_{i=1}^{N} \big(y^{(i)} \log(q^{(i)}) + (1-y^{(i)}) \log(1-q^{(i)}) \big) \\
\Longrightarrow \quad & \min \quad -\frac{1}{N} \sum_{i=1}^{N} \big(y^{(i)} \log(\widehat{y}^{(i)}) + (1-y^{(i)}) \log(1-\widehat{y}^{(i)}) \big) \quad (\because \ \widehat{y}^{(i)} = q^{(i)})
\end{aligned}
$$

因此最终的训练损失函数为：

$$
\min \quad -\frac{1}{N} \sum_{i=1}^{N} \big(y^{(i)} \log(\widehat{y}^{(i)}) + (1-y^{(i)}) \log(1-\widehat{y}^{(i)}) \big)
$$

Kullback–Leibler-Divergence 为了数学符号表示的便捷性，我们做如下符号表示：

- $\boldsymbol{p}$: 样本类别的真实概率分布
- $\boldsymbol{q}$: 样本类别的估计概率分布

根据 KL-Divergence 进行目标函数的推导，实际上是希望最小化所有训练样本类别的真实概率分布和估计概率分布的差异，即：

$$
\begin{aligned}
& \min \quad \frac{1}{N} \sum_{i=1}^{N} D_{\mathrm{KL}}(\boldsymbol{p^{(i)}} \,||\, \boldsymbol{q^{(i)}}) \\
\Longrightarrow \quad & \min \quad \frac{1}{N} \sum_{i=1}^{N} \big(\boldsymbol{p^{(i)}} \cdot \log(\frac{\boldsymbol{p^{(i)}}}{\boldsymbol{q^{(i)}}}) \big) \\
\Longrightarrow \quad & \min \quad \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log(\frac{p_k^{(i)}}{q_k^{(i)}}) \\
\Longrightarrow \quad & \min \quad \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \big(\log(p_k^{(i)}) - \log(q_k^{(i)}) \big) \\
\Longrightarrow \quad & \min \quad \underbrace{\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log(p_k^{(i)})}_{\text{常数，可忽略}} - \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log(q_k^{(i)}) \\
\Longrightarrow \quad & \min \quad -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log(q_k^{(i)}) \\
\Longrightarrow \quad & \min \quad -\frac{1}{N} \sum_{i=1}^{N} \big(y^{(i)} \log(q^{(i)}) + (1-y^{(i)}) \log(1-q^{(i)}) \big) \\
\Longrightarrow \quad & \min \quad -\frac{1}{N} \sum_{i=1}^{N} \big(y^{(i)} \log(\widehat{y}^{(i)}) + (1-y^{(i)}) \log(1-\widehat{y}^{(i)}) \big) \quad (\because \ \widehat{y}^{(i)} = q^{(i)})
\end{aligned}
$$

因此最终的训练损失函数为：

$$
\min \quad -\frac{1}{N} \sum_{i=1}^{N} \big(y^{(i)} \log(\widehat{y}^{(i)}) + (1-y^{(i)}) \log(1-\widehat{y}^{(i)}) \big)
$$

Optimization 引入 l2 正则化项，Logistic Regression 模型的目标函数为：

$$J(\boldsymbol{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}\log(\widehat{y}^{(i)}) + (1-y^{(i)})\log(1-\widehat{y}^{(i)})\right) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$

为了之后计算的便捷性，先对 $\phi(z)$ 求梯度：

$$\begin{aligned}\frac{\partial\phi(z)}{\partial z} &= -\big(1+e^{-z}\big)^{-2}e^{-z}(-1)\\ &= \frac{e^{-z}}{(1+e^{-z})^2}\\ &= \frac{e^{-z}}{1+e^{-z}}\cdot\frac{1}{1+e^{-z}}\\ &= \big(1-\phi(z)\big)\phi(z)\end{aligned}$$

对第 j 个维度的参数求梯度：

$$\begin{aligned}-\frac{\partial J(\boldsymbol{w};\boldsymbol{x^{(i)}};y^{(i)})}{\partial w_j} &= \frac{\partial}{\partial w_j}\left(y^{(i)}\log\widehat{y}^{(i)} + (1-y^{(i)})\log(1-\widehat{y}^{(i)})\right) - \frac{\partial}{\partial w_j}\left(\frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}\right)\\ &= \frac{y^{(i)}}{\widehat{y}^{(i)}}\frac{\partial\widehat{y}^{(i)}}{\partial w_j} + \frac{1-y^{(i)}}{1-\widehat{y}^{(i)}}(-1)\frac{\partial\widehat{y}^{(i)}}{\partial w_j} - \lambda w_j\\ &= \frac{\widehat{y}^{(i)}(y^{(i)}-1)+y^{(i)}(1-\widehat{y}^{(i)})}{\widehat{y}^{(i)}(1-\widehat{y}^{(i)})}\frac{\partial\widehat{y}^{(i)}}{\partial w_j} - \lambda w_j\end{aligned}$$

根据之前对 $\phi(z)$ 所求梯度，运用梯度链式法则有：

$$\frac{\partial\widehat{y}^{(i)}}{\partial w_j} = \frac{\partial\phi(\boldsymbol{x^{(i)}};\boldsymbol{w})}{\partial w_j} = \frac{\partial\phi(z^{(i)})}{\partial z^{(i)}}\frac{\partial z^{(i)}}{\partial w_j}$$

因此代入上式中可继续化简得：

$$\begin{aligned}-\frac{\partial J(\boldsymbol{w};\boldsymbol{x^{(i)}};y^{(i)})}{\partial w_j} &= \frac{\widehat{y}^{(i)}(y^{(i)}-1)+y^{(i)}(1-\widehat{y}^{(i)})}{\widehat{y}^{(i)}(1-\widehat{y}^{(i)})}\frac{\partial\widehat{y}^{(i)}}{\partial w_j} - \lambda w_j\\ &= \frac{\widehat{y}^{(i)}(y^{(i)}-1)+y^{(i)}(1-\widehat{y}^{(i)})}{\widehat{y}^{(i)}(1-\widehat{y}^{(i)})}\frac{\partial\phi(z)}{\partial z}\frac{\partial z}{\partial w_j} - \lambda w_j\\ &= \frac{\widehat{y}^{(i)}(y^{(i)}-1)+y^{(i)}(1-\widehat{y}^{(i)})}{\widehat{y}^{(i)}(1-\widehat{y}^{(i)})}\big(1-\phi(z^{(i)})\big)\phi(z^{(i)})x_j^{(i)} - \lambda w_j\\ &= \frac{\widehat{y}^{(i)}y^{(i)}-\widehat{y}^{(i)}+y^{(i)}-\widehat{y}^{(i)}y^{(i)}}{\widehat{y}^{(i)}(1-\widehat{y}^{(i)})}\big(1-\widehat{y}^{(i)}\big)\widehat{y}^{(i)}x_j^{(i)} - \lambda w_j \quad (\because\ \phi(z^{(i)}) = \widehat{y}^{(i)})\\ &= (y^{(i)}-\widehat{y}^{(i)})x_j^{(i)} - \lambda w_j\end{aligned}$$

因此使用 Mini-Batch-SGD 优化算法时迭代更新公式为：

$$\begin{aligned}w_j &\leftarrow w_j - \eta\left(\frac{1}{m}\sum_{i=1}^{m}\frac{\partial J(\boldsymbol{w};\boldsymbol{x^{(i)}};y^{(i)})}{\partial w_j}\right)\\ w_j &\leftarrow w_j + \eta\left(\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}-\widehat{y}^{(i)})x_j^{(i)} - \lambda w_j\right)\end{aligned}$$

7.1.3 Model implement

```
# -*- coding: utf-8 -*-
"""
Created on 2018/10/31 05:03
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
    An Implement of Logistic Regression Based on TensorFlow.
"""
import tensorflow as tf
from tensorflow.contrib.layers import l2_regularizer

def input_fn(
        filenames,
        delimiter,
        batch_size,
        epochs,
        identifier=":",
        perform_shuffle=True,
        dtype_indices=tf.int32,
        dtype_values=tf.float32,
        name_feature_indices="feat_inds",
        name_feature_values="feat_vals",
        num_parallel_calls=10,
        buffer_size_prefetch=500000,
        buffer_size_shuffle=2048):
    """
    Description
    -----------
    The input function for loading training dataset when using tensorflow high level API Estimator. The columns
                                                                are
    delimiter(e.g. tab) separeted with the following schema:
    <label> <index 1>:<value 1> ... <index j>:<value j> ... <index d>:<value d> (d is the field size)
    where:
        delimiter is "\t" in normal txt file, "," in normal csv file;
        identifier always equals to ":".

    Parameters
    ----------
    :param filenames: A list containing one or more filenames.
    :param delimiter: A str(e.g. "\t") to separate different <feature index j>:<feature value j> pairs in
                                                                dataset files.
    :param batch_size: An integer scalar, representing the number of consecutive elements of this dataset to
                                                                combine in a single batch.
    :param epochs: An integer scalar, representing the number of times the dataset should be repeated.
    :param identifier: A str(e.g. defaults to ":") to seprate feature index and feature value in one specific
                                                                pair.
    :param perform_shuffle: A bool variable(defaults to True) to instruct whether randomly shuffles the
                                                                elements of this dataset.
    :param dtype_indices: The numerical type of indices. Note it must be equal to dtype_ind in model_fn.
    :param dtype_values: The numerical type of values. Note it must be equal to dtype_val in model_fn.
    :param name_feature_indices: A str, representing the name of feature indices in return.
```

```
    :param name_feature_values: A str, representing the name of feature values in return.
    :param num_parallel_calls: A integer scalar, representing the number elements to process in parallel.
    :param buffer_size_prefetch: A integer scalar, representing the maximum number of elements that will be
                                                        buffered when prefetching.
    :param buffer_size_shuffle: A integer scalar, representing the number of elements from this dataset from
                                                        which the new dataset will sample.

    Returns
    -------
    :return: A dict of two Tensors, representing feature indices and feature values in a single batch.
    {
        <name_feature_indices>: feature indices Tensor in shape of (batch_size, field_size),
        <name_feature_values>: feature values Tensor in shape of (batch_size, field_size)
    }
    :return: A Tensor in shape of (batch_size, ), representing labels in a single batch.
    """
    def map_func(line):
        columns =tf.string_split(source=[line], delimiter=delimiter, skip_empty=False)
        label =tf.string_to_number(string_tensor=columns.values[0], out_type=dtype_values)
        splits =tf.string_split(source=columns.values[1: ], delimiter=identifier, skip_empty=False)
        features =tf.reshape(tensor=splits.values, shape=splits.dense_shape) # A tensor in shape of
                                                            (field_size, 2), the first column are feature
                                                            indices, and the second column are feature values
        feat_inds_str, feat_vals_str =tf.split(value=features, num_or_size_splits=2, axis=1) # Two tensors in
                                                            shape of (field_size, 1)
        feat_inds =tf.string_to_number(string_tensor=feat_inds_str, out_type=dtype_indices) # A tensor in shape
                                                            of (field_size, 1)
        feat_vals =tf.string_to_number(string_tensor=feat_vals_str, out_type=dtype_values) # A tensor in shape
                                                            of (field_size, 1)
        feat_inds =tf.reshape(tensor=feat_inds, shape=(-1, )) # A tensor in shape of (field_size, )
        feat_vals =tf.reshape(tensor=feat_vals, shape=(-1, )) # A tensor in shape of (field_size, )
        return {name_feature_indices: feat_inds, name_feature_values: feat_vals}, label

    dataset =tf.data.TextLineDataset(filenames=filenames). \
        map(map_func=map_func, num_parallel_calls=num_parallel_calls). \
        prefetch(buffer_size=buffer_size_prefetch)
    if perform_shuffle ==True:
        dataset =dataset.shuffle(buffer_size=buffer_size_shuffle)
    dataset =dataset. \
        repeat(count=epochs). \
        batch(batch_size=batch_size, drop_remainder=False)
    iterator =dataset.make_one_shot_iterator()
    features, labels =iterator.get_next()
    return features, labels


def model_fn(features, labels, mode, params):
    """Model function of Logistic Regression model for sparse predictive analytics."""
    # ----------hyper-parameters----------
    feature_size =params["feature_size"]
    use_global_bias =params["use_global_bias"]
    l2_reg =params["l2_reg"]
```

```
optimizer =params["optimizer"]
learning_rate =params["learning_rate"]
name_feature_indices =params["name_feature_indices"]
name_feature_values =params["name_feature_values"]
dtype_indices =params["dtype_indices"]
dtype_values =params["dtype_values"]

# ----------features----------
# with tf.variable_scope(name_or_scope="features"):
feat_inds =features[name_feature_indices] # A tensor in shape of (None, field_size)
feat_vals =features[name_feature_values] # A tensor in shape of (None, field_size)

# ----------Build f(x)----------
with tf.variable_scope(name_or_scope="parameters"):
    # ----------parameters----------
    w0 =tf.get_variable(
        name="w0",
        shape=[1, ],
        dtype=dtype_values,
        initializer=tf.zeros_initializer(dtype=dtype_values)
    ) # The variable of global bias
    W =tf.get_variable(
        name="W",
        shape=[feature_size, ],
        dtype=dtype_values,
        initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.01, dtype=dtype_values),
        regularizer=l2_regularizer(scale=l2_reg)
    ) # The variables of weights (first order)

with tf.variable_scope(name_or_scope="embedding-lookup"):
    # embedding lookup operations for weights
    w =tf.nn.embedding_lookup(params=W, ids=feat_inds) # A tensor in shape of (None, field_size)

with tf.variable_scope(name_or_scope="linear-part"):
    y_lr =tf.reduce_sum(
        input_tensor=tf.multiply(x=w, y=feat_vals), # A tensor in shape of (None, field_size),
        axis=1,
        keepdims=False
    ) # A tensor in shape of (None, )

with tf.variable_scope(name_or_scope="output"):
    if use_global_bias ==True:
        logits =w0 +y_lr
    else:
        logits =y_lr
    # A tensor in shape of (None, )

    predictions ={
        # Generate predictions (for PREDICT and EVAL mode)
        "probability": tf.sigmoid(x=logits),
        "class": tf.cast(x=tf.round(x=tf.sigmoid(x=logits)), dtype=dtype_indices)
    }
```

```
    export_outputs ={
        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
                                                                tf.estimator.export.PredictOutput(predictions)
    }

# ----------Provide an estimator spec for `ModeKeys.PREDICTION` mode----------
if mode ==tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions, export_outputs=export_outputs)

# ----------Build loss function (for both TRAIN and EVAL modes)----------
with tf.variable_scope(name_or_scope="loss"):
    loss =tf.reduce_mean(
        input_tensor=tf.nn.sigmoid_cross_entropy_with_logits(labels=labels, logits=logits), # A tensor in
                                                                shape of (None, )
        axis=0,
        keepdims=False
    ) # A scaler representing the residual loss
    reg_loss =tf.reduce_sum(
        input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
        axis=0,
        keepdims=False
    ) # A scaler representing the regularization loss
    loss +=reg_loss

# ----------Build optimizer----------
with tf.variable_scope(name_or_scope="optimization"):
    if (optimizer.lower() =="sgd"):
        opt =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() =="momentum"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=False)
    elif (optimizer.lower() =="nesterov"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=True)
    elif (optimizer.lower() =="adagrad"):
        opt =tf.train.AdagradOptimizer(learning_rate=learning_rate, initial_accumulator_value=0.1)
    elif (optimizer.lower() =="adadelta"):
        opt =tf.train.AdadeltaOptimizer(learning_rate=learning_rate, rho=0.95, epsilon=1e-8)
    elif (optimizer.lower() =="rmsprop"):
        opt =tf.train.RMSPropOptimizer(learning_rate=learning_rate, decay=0.9, epsilon=1e-10)
    elif (optimizer.lower() =="ftrl"):
        opt =tf.train.FtrlOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() =="adam"):
        opt =tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8)
    else:
        raise ValueError("Argument <optimizer> of model_fn is invalid")

    train_op =opt.minimize(loss=loss, global_step=tf.train.get_global_step())

# ----------Provide an estimator spec for `ModeKeys.TRAIN` mode----------
if (mode ==tf.estimator.ModeKeys.TRAIN):
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

# ----------Build evaluation metrics----------
```

```
with tf.variable_scope(name_or_scope="evaluation"):
    eval_metric_ops ={
        "accuracy": tf.metrics.accuracy(labels=labels, predictions=predictions["class"]),
        "precision": tf.metrics.precision(labels=labels, predictions=predictions["class"]),
        "recall": tf.metrics.recall(labels=labels, predictions=predictions["class"]),
        "auc": tf.metrics.auc(labels=labels, predictions=predictions["class"])
    }

# ----------Provide an estimator spec for `ModeKeys.EVAL` mode----------
if mode ==tf.estimator.ModeKeys.EVAL:
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
                                                            eval_metric_ops=eval_metric_ops)
```

7.2 Softmax Regression

7.2.1 Model prediction

Softmax Regression，又称 Multinomial Logistic Regression，是一个用于**多分类**的线性模型，其本质是将多分类问题转换为多分类类别的**离散型概率分布**问题。判别函数包括两部分：

- 内函数：每类的 logit 函数：

$$z_k = \sum_{j=1}^{d} w_{k,j} x_j, \quad k \in \{1, ..., K\}$$

- 外函数：Softmax 函数，即每类的**概率估计**（**注意指数函数中没有负号**）：

$$q_k = \frac{\exp(z_k)}{\sum_{l=1}^{K} \exp(z_l)}, \quad k \in \{1, ..., K\}$$

- 合并表示：第 k 类的概率估计：

$$q_k = \frac{\exp(\sum_{j=1}^{d} w_{k,j} x_j)}{\sum_{l=1}^{K} \exp(\sum_{j=1}^{d} w_{l,j} x_j)} = p(\, y = k \,|\, \boldsymbol{x} \,)$$

其中 $w_{k,j}$ 表示，特征向量 $\boldsymbol{x}$ 的第 j 维到输出第 k 类之间的参数。计算完每类的概率，概率最大的类别就是

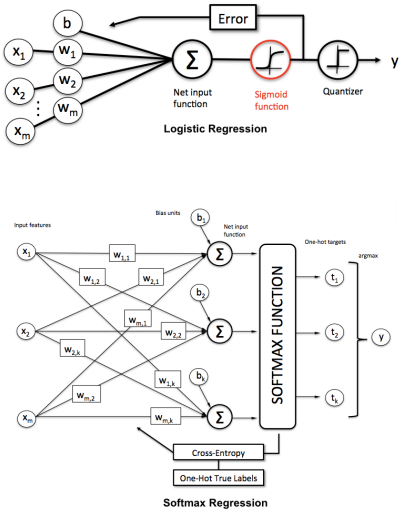


图 37: Softmax regression & logistic regression

最终的预测类别。

待学习参数 Softmax Regression 的待学习参数为矩阵：

$$w \in \mathbb{R}^{K \times d}$$

其中 $w_{k,j}$, $k \in \{1, ..., K\}$, $j \in \{1, ..., d\}$ 表示链接特征向量 $\boldsymbol{x}$ 的第 j 个维度和最终分类第 k 类的参数。

7.2.2 Model training

Dataset 训练数据的表达如下：

- 训练数据集：$\mathcal{D} = \{(\boldsymbol{x^{(i)}}, y^{(i)})\}_{i=1}^{N}$
- 样本标签：$y^{(i)} \in \mathbb{R}^K$

Modeling details Softmax Regression 的判别函数包括两部分：

- 内函数：每类的 logit 函数：

$$z_k = \sum_{j=1}^{d} w_{k,j} x_j, \quad k \in \{1, ..., K\}$$

- 外函数：Softmax 函数，即每类的**概率估计**：

$$q_k = \frac{\exp(z_k)}{\sum\limits_{l=1}^{K} \exp(z_l)}, \quad k \in \{1, ..., K\}$$

- 合并表示：第 k 类的概率估计：

$$q_k = \frac{\exp(\sum\limits_{j=1}^{d} w_{k,j} x_j)}{\sum\limits_{l=1}^{K} \exp(\sum\limits_{j=1}^{d} w_{l,j} x_j)}$$

Loss function 根据 KL-Divergence 进行表示可得：

$$\begin{aligned}
& \min \quad \frac{1}{N} \sum_{i=1}^{N} D_{\mathrm{KL}}(\boldsymbol{p^{(i)}} \,||\, \boldsymbol{q^{(i)}}) \\
\Longrightarrow \quad & \min \quad \frac{1}{N} \sum_{i=1}^{N} \left(\boldsymbol{p^{(i)}} \cdot \log(\frac{\boldsymbol{p^{(i)}}}{\boldsymbol{q^{(i)}}})\right) \\
\Longrightarrow \quad & \min \quad \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log(\frac{p_k^{(i)}}{q_k^{(i)}}) \\
\Longrightarrow \quad & \min \quad \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \big(\log(p_k^{(i)}) - \log(q_k^{(i)})\big) \\
\Longrightarrow \quad & \min \quad \underbrace{\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log(p_k^{(i)})}_{\text{熵，常数}} + \underbrace{\left(-\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log(q_k^{(i)})\right)}_{\text{交叉熵}} \\
\Longrightarrow \quad & \min \quad -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^{(i)} \log(q_k^{(i)})
\end{aligned}$$

注意：

- 之后推导分类问题的损失函数时，只写交叉熵 (Cross Entropy) 即可，不必再从 KL-Divergence 起步
- **Maximum Log Likelihood ⇔ Minimize KL-Divergence**

Optimization 引入 l2 正则化项，Softmax Regression 模型的目标函数为：

$$\begin{aligned} J(\boldsymbol{w}) &= -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_k^{(i)} \log(q_k^{(i)}) + \frac{\lambda}{2}\sum_{k=1}^{K}\sum_{j=1}^{d} w_{k,j}^2 \\ &= -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} \underbrace{I(y^{(i)}=k)}_{0/1 \text{ 独热离散型分布}} \log(q_k^{(i)}) + \frac{\lambda}{2}\sum_{k=1}^{K}\sum_{j=1}^{d} w_{k,j}^2 \end{aligned}$$

对联系输入第 j 个维度和输出第 k 类的参数 $w_{k,j}$ 求梯度，可得：

$$\frac{\partial J(\boldsymbol{w};\boldsymbol{x^{(i)}};y^{(i)})}{\partial w_{k,j}} = -I(y^{(i)}=k)\frac{1}{q_k^{(i)}}\frac{\partial q_k^{(i)}}{\partial w_{k,j}} + \lambda w_{k,j}$$

根据链式求导法则可得：

$$\frac{\partial q_k^{(i)}}{\partial w_{k,j}} = \frac{\partial q_k^{(i)}}{\partial z_k^{(i)}}\frac{\partial z_k^{(i)}}{\partial w_{k,j}}$$

为了简化表示暂时不考虑样本索引 i，因为：

$$\begin{aligned} \frac{\partial q_k}{\partial z_k} &= \frac{\partial}{\partial z_k}\frac{\exp(z_k)}{\sum_{l=1}^{K}\exp(z_l)} \quad (\because q_k = \frac{\exp(z_k)}{\sum_{l=1}^{K}\exp(z_l)}) \\ &= \frac{\exp(z_k)\sum_{l=1}^{K}\exp(z_l) - \exp(z_k)\exp(z_k)}{\Big(\sum_{l=1}^{K}\exp(z_l)\Big)^2} \\ &= \frac{\exp(z_k)\Big(\sum_{l=1}^{K}\exp(z_l) - \exp(z_k)\Big)}{\Big(\sum_{l=1}^{K}\exp(z_l)\Big)^2} \\ &= \frac{\exp(z_k)}{\sum_{l=1}^{K}\exp(z_l)} \times \frac{\sum_{l=1}^{K}\exp(z_l) - \exp(z_k)}{\sum_{l=1}^{K}\exp(z_l)} \\ &= \frac{\exp(z_k)}{\sum_{l=1}^{K}\exp(z_l)} \times \Big(1 - \frac{\exp(z_k)}{\sum_{l=1}^{K}\exp(z_l)}\Big) \\ &= q_k(1-q_k) \\ \frac{\partial z_k^{(i)}}{\partial w_{k,j}} &= \frac{\partial}{\partial w_{k,j}}\sum_{j=1}^{d} w_{k,j}x_j \quad (\because z_k = \sum_{j=1}^{d} w_{k,j}x_j) \\ &= x_j \end{aligned}$$

所以代入上式可得：

$$\begin{aligned} \frac{\partial q_k^{(i)}}{\partial w_{k,j}} &= \frac{\partial q_k^{(i)}}{\partial z_k^{(i)}}\frac{\partial z_k^{(i)}}{\partial w_{k,j}} \\ &= q_k^{(i)}(1-q_k^{(i)})x_j^{(i)} \end{aligned}$$

$$
\begin{aligned}
\frac{\partial J(\boldsymbol{w};\boldsymbol{x^{(i)}};y^{(i)})}{\partial w_{k,j}} &= -I(y^{(i)}=k)\frac{1}{q_k^{(i)}}\frac{\partial q_k^{(i)}}{\partial w_{k,j}} + \lambda w_{k,j} \\
&= -I(y^{(i)}=k)\frac{1}{q_k^{(i)}}q_k^{(i)}(1-q_k^{(i)})x_j^{(i)} + \lambda w_{k,j} \\
&= -I(y^{(i)}=k)(1-q_k^{(i)})x_j^{(i)} + \lambda w_{k,j} \\
&= -I(y^{(i)}=k)(1-\widehat{y}_k^{(i)})x_j^{(i)} + \lambda w_{k,j} \quad (\because \widehat{y}_k^{(i)} = q_k^{(i)}) \\
&= \begin{cases} -(1-\widehat{y}_k^{(i)})x_j^{(i)} + \lambda w_{k,j}, & \text{if } y^{(i)}=k \\ \lambda w_{k,j}, & \text{if } y^{(i)}\neq k \end{cases}
\end{aligned}
$$

因此使用 Mini-Batch-SGD 优化算法时迭代更新公式为：

$$
w_{k,j} \leftarrow w_{k,j} - \eta\left(\frac{1}{m}\sum_{i=1}^{m}\frac{\partial J(\boldsymbol{w};\boldsymbol{x^{(i)}};y^{(i)})}{\partial w_{k,j}}\right)
$$

$$
w_{k,j} \leftarrow w_{k,j} - \eta\left(\frac{1}{m}\sum_{i=1}^{m}\underbrace{I(y^{(i)}=k)}_{p_k^{(i)}}(1-\widehat{y}_k^{(i)})x_j^{(i)} - \lambda w_{k,j}\right)
$$

7.2.3 Summary of softmax regression

- **Softmax Regression 中各个类之间不是相互独立的，而是相互影响的**。因为在 logit 转换概率的时候，使用的是 Softmax function，即：

$$
\widehat{y}_k = q_k = \frac{\exp(z_k)}{\sum_{l=1}^{K}\exp(z_l)} = \frac{\exp(\sum_{j=1}^{d} w_{k,j}x_j)}{\sum_{l=1}^{K}\exp(\sum_{j=1}^{d} w_{l,j}x_j)}
$$

  可以看到第 k 类的概率估计里不仅有第 k 类的参数，还有其他所有类的参数（分母），所以其不是类别独立的。

- **指数函数的作用是起到全部整合为正数，同时增强差异对比度**
  - $z_k < 0$ 时，$e^{z_k} \in (0,1)$，有效地将负数转换为了小的正数
  - $z_k > 0$ 时，$e^{z_k} \in (1,+\infty)$，且有效地放大了不同 z_k 之间对比程度
- **为了简化多分类问题的计算开销，可将 Softmax 函数转换为 K 个独立的 Logistic 函数**

7.2.4 Reference

- DeepNotes: Classification and Loss Evaluation - Softmax and Cross Entropy Loss

8 Naive Bayes

8.1 Naive Bayes Classification

8.1.1 Model prediction

Naive Bayes 是一个用于**多分类**的非线性模型预测时，我们需要**计算样本属于每一类的概率，然后选择概率最大的那一类作为预测类别**。所以样本 $\boldsymbol{x}$ 属于某一类 k 的概率为：

$$\begin{aligned} \mathrm{P}(y=k\,|\,\boldsymbol{x}) &= \frac{\mathrm{P}(\boldsymbol{x}, y=k)}{\mathrm{P}(\boldsymbol{x})} && \text{(条件概率公式)} \\ &= \frac{\mathrm{P}(\boldsymbol{x}\,|\,y=k)\mathrm{P}(y=k)}{\mathrm{P}(\boldsymbol{x})} && \text{(贝叶斯公式)} \end{aligned}$$

因为分母 $\mathrm{P}(\boldsymbol{x})$ 为归一化项，是一个常数，在比较不同类别概率的时候不起作用，因此计算的时候不必考虑，可以约去，所以实际计算的时候只是:

$$\widehat{y} = \arg\max_{k\in K}\ \mathrm{P}(\boldsymbol{x}\,|\,y=k)\mathrm{P}(y=k)$$

注意 Naive Bayes Classifier 最关键的一点：**条件独立假设**，即**假设特征之间都是相互独立的，无耦合，互不干扰**（但是实际中这种情况很少很少）。于是有：

$$\mathrm{P}(\boldsymbol{x}\,|\,y=k) = \prod_{j=1}^{d} \mathrm{P}(x_j\,|\,y=k)$$

所以：**朴素贝叶斯分类器 = 贝叶斯公式 + 条件独立假设**

$$\widehat{y} = \arg\max_{k\in K}\ \left[\prod_{j=1}^{d} \mathrm{P}(x_j\,|\,y=k)\right] \mathrm{P}(y=k)$$

8.1.2 Model training

Dataset 如下为一个二分类问题的数据集，其中任意 $\boldsymbol{x}^{(i)} \in \mathbb{R}^3$，我们将使用这个例子进行训练过程的解释说明

Example No.	Color	Type	Origin	Stolen?
1	Red	Sports	Domestic	Yes
2	Red	Sports	Domestic	No
3	Red	Sports	Domestic	Yes
4	Yellow	Sports	Domestic	No
5	Yellow	Sports	Imported	Yes
6	Yellow	SUV	Imported	No
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	No
10	Red	Sports	Imported	Yes

Model details 因此在训练中，我们需要计算的内容包括：

- 所有类的 $\mathrm{P}(y = k),\ k \in K$
- 每类每个特征上**每个特征取值**的 $\mathrm{P}(x_j \,|\, y = k)$，比如：Color 特征里 Red 为一个特征取值（$x_1 = \mathrm{Red}$）

我们希望估计样本 $\boldsymbol{x} = (\mathrm{Red}, \mathrm{SUV}, \mathrm{Domestic})$ 是否被 Stolen（这条样本没有出现在训练样本中），具体计算过程如下：

- $\mathrm{P}(y = \mathrm{Yes}) = 0.5$
- $\mathrm{P}(y = \mathrm{No}) = 0.5$
- 计算第一个特征（Color）：
  - $\mathrm{P}(x_1 = \mathrm{Red} \,|\, y = \mathrm{Yes}) = \frac{3+3\times0.5}{5+3} = 0.56$ (注：特征取之的条件概率计算使用了 **m-estimates**，下同)
  - $\mathrm{P}(x_1 = \mathrm{Red} \,|\, y = \mathrm{No}) = \frac{2+3\times0.5}{5+3} = 0.43$
- 计算第二个特征（Type）：
  - $\mathrm{P}(x_2 = \mathrm{SUV} \,|\, y = \mathrm{Yes}) = \frac{1+3\times0.5}{5+3} = 0.31$
  - $\mathrm{P}(x_2 = \mathrm{SUV} \,|\, y = \mathrm{No}) = \frac{3+3\times0.5}{5+3} = 0.56$
- 计算第三个特征（Origin）：
  - $\mathrm{P}(x_3 = \mathrm{Domestic} \,|\, y = \mathrm{Yes}) = \frac{2+3\times0.5}{5+3} = 0.43$
  - $\mathrm{P}(x_3 = \mathrm{Domestic} \,|\, y = \mathrm{No}) = \frac{3+3\times0.5}{5+3} = 0.56$
- 计算最终概率：
  - $\mathrm{P}(y = \mathrm{Yes}) \cdot \mathrm{P}(x_1 = \mathrm{Red} \,|\, y = \mathrm{Yes}) \cdot \mathrm{P}(x_2 = \mathrm{SUV} \,|\, y = \mathrm{Yes}) \cdot \mathrm{P}(x_3 = \mathrm{Domestic} \,|\, y = \mathrm{Yes}) = 0.5 \times 0.56 \times 0.31 \times 0.43 = 0.037$
  - $\mathrm{P}(y = \mathrm{No}) \cdot \mathrm{P}(x_1 = \mathrm{Red} \,|\, y = \mathrm{No}) \cdot \mathrm{P}(x_2 = \mathrm{SUV} \,|\, y = \mathrm{No}) \cdot \mathrm{P}(x_3 = \mathrm{Domestic} \,|\, y = \mathrm{No}) = 0.5 \times 0.43 \times 0.56 \times 0.56 = 0.069$
- 得出结果：Since $0.037 < 0.069$, our test sample gets classified as "No".

8.1.3 Naive bayes note

- 该模型为非参数模型，因此没有之后目标函数和优化算法的步骤
- 训练和预测同时进行

8.2 Logistic Regression VS Naive Bayes

- Logistic Regression 是**判别模型**，Naive Bayes 是**生成模型**
  - Naive Bayes 在计算 $\mathrm{P}(y \mid \boldsymbol{x})$ 之间，需要先从训练数据中计算出概率 $\mathrm{P}(\boldsymbol{x} \mid y)$ 以及 $\mathrm{P}(y)$
  - Logistic Regression 通过在训练数据上学习直接得到判别函数 $\mathrm{P}(y \mid \boldsymbol{x})$，不需要知道 $\mathrm{P}(\boldsymbol{x} \mid y)$ 以及 $\mathrm{P}(y)$
- Naive Bayes **必须建立在特征条件独立假设的基础上**，而 Logistic Regression 则不需要。当然，如果特征之间满足条件独立假设的条件，那么 Logistic Regression 能够取得非常好的效果，如果没有，LR 仍然能够通过调整参数让模型符合数据的分布，从而训练得到在现有数据集下的一个最优模型
- 数据集较小，使用 Naive Bayes；数据集较大，使用 Logistic Regression

9 Graphical Models

9.1 Hidden Markov Models (HMM)

9.1.1 Introduction（未完成）

隐马尔可夫模型（Hidden Markov Model, HMM）用于处理**时间序列数据**。HMM 包含两部分内容：隐藏变量（hidden variables）和观测变量（observation variables）。而 HMM 的能力在于：根据给定已知的观测变量序列，估计对应的隐藏变量序列，并对未来的观测变量序列做预测。

9.1.2 Model formulation

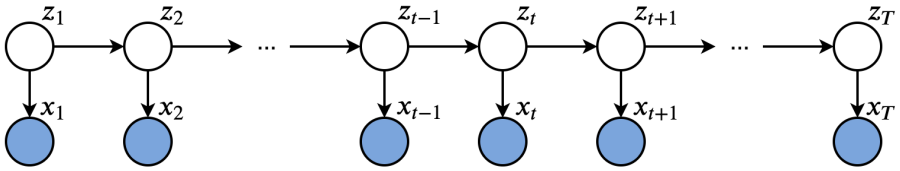


图 38: A Bayesian network specifying conditional independence relations for a hidden markov model.

Component 图中 z_t 为在时刻 t 的隐藏变量，x_t 为对应的观测变量。该模型的关键是隐藏变量之间满足如下条件独立性：给定 z_t 时，z_{t-1} 和 z_{t+1} 条件独立：

$$z_{t-1} \perp z_{t+1} \,|\, z_t$$

因此该模型的联合概率分布 $p(z_1, \cdots, z_T, x_1, \cdots, x_T)$ 可以表示如下：

$$p(z_1, \cdots, z_T, x_1, \cdots, x_T) = p(z_1) \left[\prod_{t=2}^{T} p(z_t|z_{t-1}) \right] \left[\prod_{t=1}^{T} p(x_t|z_t) \right]$$

因此该模型被拆解为了三部分：

1. 初始概率模型：$p(z_1)$
2. 转移概率模型：$p(z_t|z_{t-1})$
3. 发射概率模型：$p(x_t|z_t)$

Representation

Learning

9.1.3 Reference

- An Introduction to Hidden Markov Models and Bayesian Networks[?]
- CMU Slides: Hidden Markov Models and Conditional Random Fields

9.2 Conditional Random Fields (CRF)

9.2.1 Model formulation

9.2.2 Reference

- Hanna M.Wallach: Conditional Random Fields
- Edwin Chen: Introduction to Conditional Random Fields
- Eric P.Xing-CMU: Probabilistic Graphical Models

10 Time Series Models

10.1 ARIMA（未完成）

10.1.1 Autoregressive models (AR)

10.1.2 Moving average models (MA)

10.1.3 Autoregressive moving average models (ARMA)

10.2 Online ARIMA Algorithms for Time Series Prediction

10.2.1 Introduction

选自论文Online ARIMA Algorithms for Time Series Prediction[?]

10.3 Prophet（未完成）

10.4 Modeling Long and Short-Term Temporal Patterns with Deep Neural Networks（未完成）

11 Support Vector Machine

11.1 Hard-margin SVM Classification

考虑**所有训练样本线性可分**的情况，即如图所示，正负样本被完全地分割在了分割超平面的两侧：

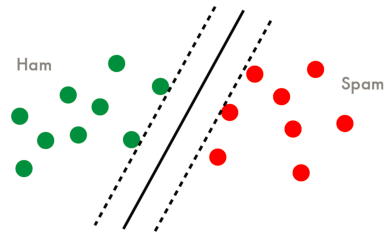


图 39: Linear separability case of SVM

11.1.1 Model prediction

预测阶段之前，我们已经学到了模型的参数 $\boldsymbol{w}$ 和 b，那么根据预测函数公式：

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$$

我们可以对测试样本 $\boldsymbol{x}$ 作出如下预测：

- $f(\boldsymbol{x}) > 0 \ \rightarrow \ \widehat{y} = 1$
- $f(\boldsymbol{x}) < 0 \ \rightarrow \ \widehat{y} = -1$

11.1.2 Model training

Dataset 训练数据为：

$$\begin{aligned}
&\mathcal{D}^{n \times d} = \{(\boldsymbol{x^{(i)}}, y^{(i)})\}_{i=1}^{N} \\
&\boldsymbol{x^{(i)}} \in \mathbb{R}^d \quad \forall i \in \{1, ..., n\} \\
&y^{(i)} \in \{-1, +1\} \quad \forall i \in \{1, ..., n\}
\end{aligned}$$

Model details[?] 我们要做的就是训练出参数 $\boldsymbol{w}$ 和 b。一般意义上，我们需要:

$$f(\boldsymbol{x^{(i)}}) = \boldsymbol{w} \cdot \boldsymbol{x^{(i)}} + b = \begin{cases} > 0 & , \quad \text{if } y^{(i)} = 1 \\ < 0 & , \quad \text{if } y^{(i)} = -1 \end{cases}$$

但是为了模型的鲁棒性，我们设置约束条件（最大间隔假设）为：

$$f(\boldsymbol{x^{(i)}}) = \boldsymbol{w} \cdot \boldsymbol{x^{(i)}} + b = \begin{cases} \geq 1 & , \quad \text{if } y^{(i)} = 1 \\ \leq -1 & , \quad \text{if } y^{(i)} = -1 \end{cases}$$

合并起来，即：

$$y^{(i)}(\boldsymbol{w} \cdot \boldsymbol{x^{(i)}} + b) \geq 1, \ \forall (\boldsymbol{x^{(i)}}, y^{(i)}) \in \mathcal{D}$$

而这就是 SVM 训练模型，即一个约束优化模型的**约束条件**。

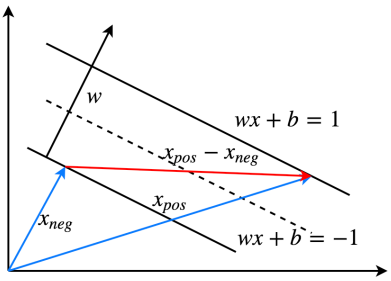


图 40: Width of Streets in SVM

问题：如何计算即“街宽”?

- 在负样本区域“街边”任意取一点，记作 $\boldsymbol{x}_-$
- 在正样本区域“街边”任意取一点，记作 $\boldsymbol{x}_+$
- 连线 $\boldsymbol{x}_-$ 和 $\boldsymbol{x}_+$，得到两条街之间任意的一条向量 $(\boldsymbol{x}_+ - \boldsymbol{x}_-)$（暂且叫做“街线”）
- 两条街之间的距离，实际上就是街线在法向量 $\boldsymbol{w}$ 方向上的投影，即：

$$\begin{aligned}\text{width} &= \|(\boldsymbol{x}_+ - \boldsymbol{x}_-)\| \cdot \cos\theta \\ &= \|(\boldsymbol{x}_+ - \boldsymbol{x}_-)\| \cdot \frac{(\boldsymbol{x}_+ - \boldsymbol{x}_-) \cdot \boldsymbol{w}}{\|(\boldsymbol{x}_+ - \boldsymbol{x}_-)\| \cdot \|\boldsymbol{w}\|} \\ &= \frac{(\boldsymbol{x}_+ - \boldsymbol{x}_-) \cdot \boldsymbol{w}}{\|\boldsymbol{w}\|} \\ &= \frac{\boldsymbol{w} \cdot \boldsymbol{x}_+}{\|\boldsymbol{w}\|} - \frac{\boldsymbol{w} \cdot \boldsymbol{x}_-}{\|\boldsymbol{w}\|}\end{aligned}$$

- 街边的点满足条件：$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) = 1$，因此：
  - 对 $\boldsymbol{x}_+$: 因为 $y_i = 1$，所以 $\boldsymbol{w} \cdot \boldsymbol{x}_+ = 1 - b$
  - 对 $\boldsymbol{x}_-$: 因为 $y_i = -1$，所以 $\boldsymbol{w} \cdot \boldsymbol{x}_- = -1 - b$
- 故而街道的距离可化简为：

$$\begin{aligned}\text{width} &= \frac{1-b}{\|\boldsymbol{w}\|} - \frac{-(1+b)}{\|\boldsymbol{w}\|} \\ &= \frac{1-b+1+b}{\|\boldsymbol{w}\|} \\ &= \frac{2}{\|\boldsymbol{w}\|}\end{aligned}$$

至此，优化目标和约束条件全有了，那么我们的训练模型可以建模为如下：

$$\begin{aligned}&\max \frac{2}{\|\boldsymbol{w}\|} \\ &\text{s.t. } y^{(i)} \cdot (\boldsymbol{w} \cdot \boldsymbol{x}^{(\boldsymbol{i})} + b) \geq 1 \quad \forall (\boldsymbol{x}^{(\boldsymbol{i})}, y^{(i)}) \in \mathcal{D}\end{aligned}$$

将目标函数进行等价变换，得到如下**最终原始优化模型 (primal problem)**：

$$\begin{aligned}&\min \frac{1}{2}\|\boldsymbol{w}\|^2\\&\text{s.t. } y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}}+b)\geq 1 \quad \forall(\boldsymbol{x^{(i)}},y^{(i)})\in\mathcal{D}\end{aligned}$$

这个原始模型有以下特点：

- 目标函数是关于 $\boldsymbol{w}$ 的凸函数
- 约束条件是关于 $\boldsymbol{w}$ 的线性函数

那么其对偶问题的最优解一定等于原始问题的最优解，如下介绍 SVM 原始模型的拉格朗日对偶模型。

SVM 对偶模型 SVM 原始模型的拉格朗日函数为：

$$J(\boldsymbol{w},b,\boldsymbol{\alpha})=\frac{1}{2}\|\boldsymbol{w}\|^2+\sum_{i=1}^{N}\alpha^{(i)}\big(1-y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}}+b)\big)$$

注意：原始模型的约束条件为 $\geq$，那么拉格朗日函数里为 $-$，即正好相反。根据 KKT 条件，即：

$$\begin{cases}\nabla_{\boldsymbol{w}}=0,\\ \nabla_b=0,\\ 1-y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}}+b)\leq 0, \quad \forall(\boldsymbol{x^{(i)}},y^{(i)})\in\mathcal{D}\\ \alpha^{(i)}\geq 0, \quad \forall i\in\{1,...,n\}\\ \alpha^{(i)}\big(1-y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}}+b)\big)=0, \quad \forall i\in\{1,...,n\}\end{cases}$$

那么根据第一个约束有：

$$\begin{aligned}\nabla_{\boldsymbol{w}}&=\boldsymbol{w}+\sum_{i=1}^{N}\alpha^{(i)}(-y^{(i)}\boldsymbol{x^{(i)}})\\&=\boldsymbol{w}-\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}=0\\ \boldsymbol{w}&=\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\end{aligned}$$

根据第二个约束有：

$$\begin{aligned}\nabla_b&=\sum_{i=1}^{N}\alpha^{(i)}(-y^{(i)})=0\\ \sum_{i=1}^{N}\alpha^{(i)}y^{(i)}&=0\end{aligned}$$

将 $\boldsymbol{w}=\sum\limits_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}$ 和 $\sum\limits_{i=1}^{N}\alpha^{(i)}y^{(i)}=0$ **代入拉格朗日函数**（还是一个最小化问题）中，可得：

$$J(\boldsymbol{w},b,\boldsymbol{\alpha})=\frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w}+\sum_{i=1}^{N}\alpha^{(i)}-\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}})-b\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}$$

$$
\begin{aligned}
&= \frac{1}{2}\big(\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\big)^2 + \sum_{i=1}^{N}\alpha^{(i)} - \sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\Big(\big(\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\big)\cdot\boldsymbol{x^{(i)}}\Big)\\
&= \underbrace{\frac{1}{2}\big(\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\big)\big(\sum_{j=1}^{n}\alpha^{(j)}y^{(j)}\boldsymbol{x^{(i)}}\big)}_{\text{用两个索引表示，之后可以交叉}} + \sum_{i=1}^{N}\alpha^{(i)} - \sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\Big(\big(\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\big)\cdot\boldsymbol{x^{(i)}}\Big)\\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} + \sum_{i=1}^{N}\alpha^{(i)} - \underbrace{\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\Big(\big(\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\big)\cdot\boldsymbol{x^{(i)}}\Big)}_{\text{对此项进行变换}}\\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} + \sum_{i=1}^{N}\alpha^{(i)} - \underbrace{\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\big(\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\big)\cdot\boldsymbol{x^{(i)}}}_{\text{都是在一起的乘法，把括号抹掉}}\\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} + \sum_{i=1}^{N}\alpha^{(i)} - \underbrace{\sum_{i=1}^{N}\alpha_i y_i \boldsymbol{x}_i\big(\sum_{j=1}^{n}\alpha_j y_j \boldsymbol{x}_j\big)}_{\text{把 } \boldsymbol{x}_i \text{ 提到前面}}\\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} + \sum_{i=1}^{N}\alpha^{(i)} - \sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}}\\
&= \sum_{i=1}^{N}\alpha^{(i)} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} = Q(\boldsymbol{\alpha})
\end{aligned}
$$

问题：既然 $\alpha^{(i)}\big(1-y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}}+b)\big)=0,\quad \forall i\in\{1,...,n\}$，为何不直接在拉格朗日函数中约掉第二项，还拼死拼活地展开？ 因为其中**含有拉格朗日乘子**，我们要将拉格朗日函数表示为乘子的函数，因此不能舍去。

Loss function 因此 SVM 对偶模型为：

$$
\begin{aligned}
&\max Q(\boldsymbol{\alpha}) = \sum_{i=1}^{N}\alpha^{(i)} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}}\\
&\text{s.t. } \sum_{i=1}^{N}\alpha^{(i)}y^{(i)} = 0,\\
&\qquad \alpha^{(i)} \geq 0,\ \forall i\in\{1,...,n\}
\end{aligned}
$$

注意这里的约束条件是 KKT 条件里第二个和第四个，至于为什么没有第一个，第三个和第五个，因为这三个其中都有 $\boldsymbol{w}$，而对偶里面没有 $\boldsymbol{w}$。

Optimization 这样解出了最优的 $\boldsymbol{\alpha}^*$ 之后，反带回去即可得到参数 $\boldsymbol{w}$ 和 b 的最优解，即：

$$
\begin{aligned}
&\boldsymbol{w}^* = \sum_{i=1}^{N}(\alpha^{(i)})^* y^{(i)}\boldsymbol{x^{(i)}}\\
&b^* = 1 - \boldsymbol{w}^*\cdot\boldsymbol{x^{(i)}},\quad (\boldsymbol{x^{(i)}}\text{为某一个正的支持向量})
\end{aligned}
$$

关于 $\boldsymbol{w}^*$ 以及 b^* 求解的解释说明 根据 KKT 条件的第五个条件，即:

$$\alpha^{(i)}\big(1-y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}}+b)\big)=0,\quad \forall i\in\{1,...,n\}$$

那么我们可以得知，对于所有的训练样本 $(\boldsymbol{x^{(i)}},y^{(i)})$:

- 要么 $\alpha^{(i)}=0$，且 $1-y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}}+b)\neq 0$（非支持向量）
- 要么 $y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}}+b)=1$，且 $\alpha^{(i)}>0$（**支持向量**）

再看看参数 $\boldsymbol{w}$ 的最优解公式：

$$\boldsymbol{w}^*=\sum_{i=1}^{N}(\alpha^{(i)})^*y^{(i)}\boldsymbol{x^{(i)}}$$

也就是说：那么非支持向量点，并不影响 $\boldsymbol{w}^*$（因为对应 $\alpha^{(i)}=0$），参数 $\boldsymbol{w}$ 的最优解**仅仅取决于所有的支持向量点**。在计算 b^* 时，我们就**仅拿一个正支持向量进行计算**就好，即:

$$\boldsymbol{w}^*\cdot\boldsymbol{x}^{\text{support+}}+b=1$$
$$b^*=1-\boldsymbol{w}^*\cdot\boldsymbol{x}^{\text{support+}}$$

所以得到的模型判别函数为：

$$f(\boldsymbol{x})=\big(\sum_{i=1}^{N}(\alpha^{(i)})^*y^{(i)}\boldsymbol{x^{(i)}}\big)\cdot\boldsymbol{x}+b^*$$

11.1.3 Summary of hard-margin svm

- **SVM 本质是线性分类器**，实际用的判别函数为线性函数 $f(\boldsymbol{x})=\boldsymbol{w}\cdot\boldsymbol{x}+b$，只不过**训练过程建模为一个约束优化模型**。
- **SVM 的样本类别标签表示不同于一般模型**。Logistic Regression 模型里，样本的类别标签 $y\in\{0,1\}$，而 SVM 里样本类别标签 $y\in\{-1,1\}$。

11.2 Soft-margin SVM Classification

之前 Hard-margin SVM Classification 的推导都是基于所有训练样本线性可分的假设下进行的，但实际中这种情况几乎不存在，因此我们想在**容忍一些误分类样本的情况下最大化间隔**。具体的方法是：为每个训练样本 $(\boldsymbol{x}_i, y_i)$ 引入一个松弛变量 ξ_i，用于**度量错误分类的程度**。

11.2.1 Model prediction

预测函数同上：

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$$

$$\widehat{y} = \begin{cases} -1, & \text{if } f(\boldsymbol{x}) > 0 \\ +1, & \text{if; } f(\boldsymbol{x}) < 0 \end{cases}$$

其中 $\boldsymbol{w}$ 和 b 为待学习参数

11.2.2 Model training of dual formulation

Dataset 训练数据为：

$$\mathcal{D}^{n\times d} = \{(\boldsymbol{x^{(i)}}, y^{(i)})\}_{i=1}^{N}$$
$$\boldsymbol{x^{(i)}} \in \mathbb{R}^d, \quad \forall i \in \{1, ..., n\}$$
$$y^{(i)} \in \{-1, +1\} \quad \forall i \in \{1, ..., n\}$$

Model details

原始模型

$$\begin{aligned} \min\ & Q_1(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi^{(i)} \\ \text{s.t. } & y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}} + b) \geq 1 - \xi^{(i)}, \quad \forall(\boldsymbol{x^{(i)}}, y^{(i)}) \in \mathcal{D} \\ & \xi^{(i)} \geq 0, \quad \forall i \in \{1, ..., n\}. \end{aligned}$$

对偶模型求解过程如下： 原始模型的拉格朗日函数为：

$$J(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi^{(i)} + \sum_{i=1}^{N}\alpha^{(i)}\left(1 - \xi^{(i)} - y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}} + b)\right) - \sum_{i=1}^{N}\beta^{(i)}\xi^{(i)}$$

根据 KKT 条件，即：

$$\begin{cases} \nabla_{\boldsymbol{w}} = 0 \\ \nabla_b = 0 \\ \nabla_{\boldsymbol{\xi}} = 0 \\ 1 - \xi^{(i)} - y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}} + b) \leq 0, \quad \forall(\boldsymbol{x^{(i)}}, y^{(i)}) \in \mathcal{D} \\ \xi^{(i)} \geq 0, \quad \forall i \in \{1, ..., n\} \\ \alpha^{(i)} \geq 0, \quad \forall i \in \{1, ..., n\} \\ \beta^{(i)} \geq 0, \quad \forall i \in \{1, ..., n\} \\ \alpha^{(i)}\left(1 - \xi^{(i)} - y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}} + b)\right) = 0, \quad \forall(\boldsymbol{x^{(i)}}, y^{(i)}) \in \mathcal{D} \\ \beta^{(i)}\xi^{(i)} = 0 \quad \forall i \in \{1, ..., n\} \end{cases}$$

解第一个约束，可得：

$$\nabla_{\boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \boldsymbol{x^{(i)}} = 0$$

解第二个约束，可得：

$$\nabla_b = -\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$$

解第三个约束，可得：

$$\nabla_{\xi^{(i)}} = C - \alpha^{(i)} - \beta^{(i)} = C - (\alpha^{(i)} + \beta^{(i)}) = 0, \quad \forall i \in \{1, ..., n\}$$

将解代入拉格朗日函数中，可得：

$$\begin{aligned}
J(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N}\xi^{(i)} + \sum_{i=1}^{N}\alpha^{(i)}\Big(1 - \xi^{(i)} - y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}} + b)\Big) - \sum_{i=1}^{N}\beta^{(i)}\xi^{(i)} \\
&= \frac{1}{2}\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\sum_{j=1}^{n}\alpha^{(j)}y^{(j)}\boldsymbol{x^{(j)}} + C\sum_{i=1}^{N}\xi^{(i)} + \sum_{i=1}^{N}\alpha^{(i)} - \sum_{i=1}^{N}\alpha^{(i)}\xi^{(i)} \\
&- \sum_{i=1}^{N}\alpha^{(i)}y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}}) - b\underbrace{\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}}_{\text{为 0，约掉}} - \sum_{i=1}^{N}\beta^{(i)}\xi^{(i)} \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} + C\sum_{i=1}^{N}\xi^{(i)} + \sum_{i=1}^{N}\alpha^{(i)} - \sum_{i=1}^{N}\big(\alpha^{(i)} + \beta^{(i)}\big)\xi^{(i)} \\
&- \sum_{i=1}^{N}\alpha^{(i)}y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}}) \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} + \sum_{i=1}^{N}\alpha^{(i)} + \sum_{i=1}^{N}\underbrace{\Big(C - \big(\alpha^{(i)} + \beta^{(i)}\big)\Big)}_{\text{为 0，约掉}}\xi^{(i)} \\
&- \sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\Big((\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}})\boldsymbol{x^{(i)}}\Big) \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} + \sum_{i=1}^{N}\alpha^{(i)} - \sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\big(\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\big) \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} + \sum_{i=1}^{N}\alpha^{(i)} - \sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\boldsymbol{x^{(i)}}\sum_{j=1}^{n}\alpha^{(j)}y^{(j)}\boldsymbol{x^{(j)}} \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} + \sum_{i=1}^{N}\alpha^{(i)} - \sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} \\
&= \sum_{i=1}^{N}\alpha^{(i)} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}} = Q_2(\boldsymbol{\alpha})
\end{aligned}$$

对偶模型

$$\max Q_2(\boldsymbol{\alpha}) = \sum_{i=1}^{N}\alpha^{(i)} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x^{(i)}}\boldsymbol{x^{(j)}}$$

$$\text{s.t.} \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0,$$
$$0 \le \alpha^{(i)} \le C, \quad \forall i \in \{1, ..., n\}$$

该模型可以解得 $\boldsymbol{\alpha}^*$，然后同上代入得到 $\boldsymbol{w}^*$ 以及 b^*.

注意

- 注意在 Soft-margin 的对偶问题中，松弛变量 $\xi^{(i)}$ 和其对应的拉格朗日乘子 $\beta^{(i)}$ 都不见了，该 Soft-margin 对偶模型和 Hard-margin 对偶模型唯一的区别就是乘子 $\alpha^{(i)}$ 的约束条件在 Soft-margin 中变得更严苛.

11.2.3 Model training of hinge loss

Hinge loss $l(z) = \max(0, 1 - z)$ 是一种损失函数，又称作 Max-margin objective，最著名的应用就是作为 SVM 的目标函数。

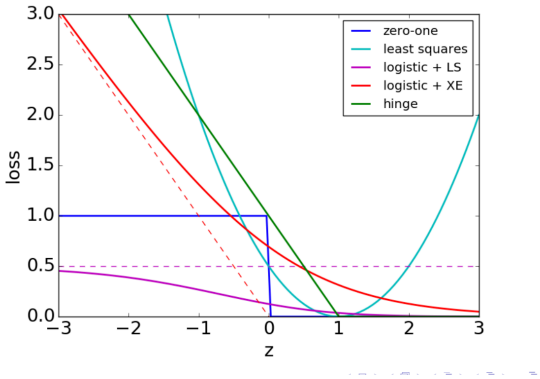


图 41: The hinge loss and other losses

- Binary classification:

$$l(\hat{y}) = \max(0,\ 1 - y \cdot \hat{y})$$

其中：

  - $y \in \{-1, +1\}$ 为真实值
  - $\hat{y} \in (-1, +1)$ 为预测值

含义：

  - 正确分类的损失 $l \in [0, 1)$
  - 错误分类的损失 $l \in (1, +\infty)$
  - $\hat{y} \in [-1, +1]$ 满足就可以了，不鼓励 $|\hat{y}| > 1$，即不鼓励模型**过分自信**。譬如：$y = +1$，而预测 $\hat{y} = +1000$ 时，虽然 $1 - y \cdot \hat{y} = 1 - 1000 = -999$，但是 $l = \max(0, -999) = 0$，因此不会得到任何奖励，即 reward $= -l = -0 = 0$。

Model details 两种求 Soft-margin SVM 模型的方式：

- 原始模型 ⇒ 对偶模型
- 原始模型 ⇒Hinge loss

原始模型如下：

$$\begin{aligned}\min\ & Q_1(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi^{(i)} \\ \text{s.t. } & y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}} + b) \geq 1 - \xi^{(i)}, \quad \forall(\boldsymbol{x^{(i)}}, y^{(i)}) \in \mathcal{D} \\ & \xi^{(i)} \geq 0, \quad \forall i \in \{1, ..., n\}.\end{aligned}$$

将约束进行变换，得：

$$\begin{aligned}\xi^{(i)} &\geq 1 - y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}} + b), \quad \forall i \in \{1, ..., n\}. \\ \xi^{(i)} &\geq 0, \quad \forall i \in \{1, ..., n\}. \\ \therefore \xi^{(i)} &\geq \max\left(0, 1 - y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}} + b)\right)\end{aligned}$$

Loss function 因此优化损失函数可以表示为：

$$\begin{aligned}& \min\ J(\boldsymbol{w}; b) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\max\left(0, 1 - y^{(i)}(\boldsymbol{w}\cdot\boldsymbol{x^{(i)}} + b)\right) \\ \Rightarrow\quad & \min\ J(\boldsymbol{w}; b) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\max\left(0, 1 - z^{(i)}(\boldsymbol{w}; b)\right) \\ \Rightarrow\quad & \min\ J(\boldsymbol{w}; b) = \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2}_{\text{l2-regularization}} + C\underbrace{\sum_{i=1}^{N}\mathcal{L}_{\text{hinge}}(z^{(i)})}_{\text{sum of hinge loss}}\end{aligned}$$

好处

- 转换为了明确的 loss function，便于使用机器学习框架化的程式进行学习训练
- 可以在线学习

Optimization 求解 $\frac{\partial J}{\partial w_j}$ 和 $\frac{\partial J}{\partial b}$ 即可：

- $\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial l}\frac{\partial l}{\partial z}\frac{\partial z}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_j}$
- $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial l}\frac{\partial l}{\partial z}\frac{\partial z}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial b}$

其中：

$$\frac{\partial l(z)}{\partial z} = \begin{cases} 0, & \text{if } z \geq 1; \\ -1, & \text{if } z < 1. \end{cases}$$

11.3 Kernel Trick for Nonlinear Classification

Introduction SVM 本质是个线性模型（超平面），但是如果训练数据不是线性可分的，那么就没有一个直接的超平面可以分离正负样本。因此为了学习一个非线性函数（nonlinear function），线性 SVM（Linear SVM）需要扩展到非线性 SVM（Nonlinear SVM），用以分类那些线性不可分的数据。

确定一个非线性 SVM 的分类函数（判定函数）包含两个步骤：

- 所有训练样本的特征向量需要转换为可以被线性分离的高维特征向量
- 在高维空间中找到一个可以将训练数据线性分离的分割超平面

这样一来，训练出的分割超平面在转换特征空间（transformed feature space）内是线性的，但是在原始特征空间（original feature space）内是非线性的。

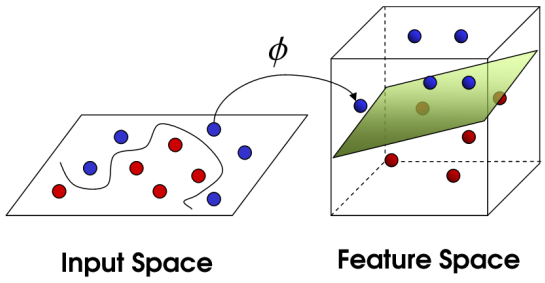


图 42: Feature Space Transformation using Kernel Method

我们用 $\phi(\cdot)$ 表示从原始特征空间到高维特征空间的**非线性映射**，那么分离边界可以表示为：

$$\boldsymbol{w} \cdot \phi(\boldsymbol{x}) + b = 0$$

对偶优化模型变为：

$$\begin{aligned} \max Q(\boldsymbol{\alpha}) &= \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{n} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \phi(\boldsymbol{x^{(i)}}) \cdot \phi(\boldsymbol{x^{(j)}}) \\ \text{s.t.} \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} &= 0, \\ 0 \leq \alpha^{(i)} &\leq C, \quad \forall i \in \{1, ..., n\} \end{aligned}$$

存在问题

- 一个有效的非线性映射 $\phi(\cdot)$ 非常难找到
- 直接计算 $\phi(\boldsymbol{x^{(i)}}) \cdot \phi(\boldsymbol{x^{(j)}})$ 非常复杂，受制于维度的问题。

解决方法 通过核技巧（kernel trick）解决，即：

$$K(\boldsymbol{u}, \boldsymbol{v}) = \phi(\boldsymbol{u}) \cdot \phi(\boldsymbol{v})$$

因此核函数是**作用在原始特征空间，用以替换在高维特征空间的成对儿特征向量的内积**。因此上述对偶模型转换为：

$$\begin{aligned}
&\max Q(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{n} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} K(\boldsymbol{x^{(i)}}, \boldsymbol{x^{(j)}}) \\
&\text{s.t.} \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0, \\
&\quad 0 \leq \alpha^{(i)} \leq C, \quad \forall i \in \{1, ..., n\}
\end{aligned}$$

所以最终的解为：

$$\begin{aligned}
f(\boldsymbol{x}) &= \big(\sum_{i=1}^{N} (\alpha^{(i)})^* y^{(i)} \phi(\boldsymbol{x^{(i)}}) \big) \cdot \phi(\boldsymbol{x}) + b^* \\
&= \sum_{i=1}^{N} (\alpha^{(i)})^* y^{(i)} \phi(\boldsymbol{x^{(i)}}) \cdot \phi(\boldsymbol{x}) + b^* \\
&= \sum_{i=1}^{N} (\alpha^{(i)})^* y^{(i)} K(\boldsymbol{x^{(i)}}, \boldsymbol{x}) + b^*
\end{aligned}$$

12 Decision Tree

12.1 Classification Tree

决策树（分类树）依据划分准则不同，可分为三类：

- ID3：信息增益

$$\text{Gain}(\mathcal{D}, a) = \text{Ent}(\mathcal{D}) - \sum_{v \in V_a} \frac{|\mathcal{D}_{a=v}|}{|\mathcal{D}|} \text{Ent}(\mathcal{D}_{a=v})$$

$$\text{Ent}(\mathcal{D}) = -\sum_{k=1}^{K} p_k \log p_k \quad (|K| \text{ is the total number of classes of labels})$$

- C4.5：信息增益率

$$\text{Gain-ratio}(\mathcal{D}, a) = \frac{\text{Gain}(\mathcal{D}, a)}{\text{IV}(a)}$$

$$\text{IV}(a) = -\sum_{v \in V_a} \frac{|\mathcal{D}_{a=v}|}{|\mathcal{D}|} \log \frac{|\mathcal{D}_{a=v}|}{|\mathcal{D}|}$$

- Classification Tree：基尼指数

$$\text{Gini-index}(\mathcal{D}, a) = \sum_{v \in V_a} \frac{|\mathcal{D}_{a=v}|}{|\mathcal{D}|} \text{Gini}(\mathcal{D}_{a=v})$$

$$\text{Gini}(\mathcal{D}) = 1 - \sum_{k=1}^{K} p_k^2 \quad (|K| \text{ is the total number of classes of labels})$$

12.1.1 Model prediction

如图所示，$\boldsymbol{x}$ 根据树的路径找到对应的叶子结点，返回叶子结点对应的预测输出。

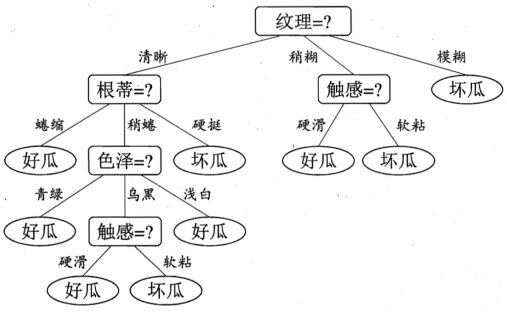


图 43: Decision Tree ID3

12.1.2 Model training

Dataset 考虑如下数据集 $\mathcal{D}$：

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

我们观测到所有特征上的所有可能取值如下：

- 色泽：{青绿，乌黑，浅白}
- 根蒂：{蜷缩，稍蜷，硬挺}
- 敲声：{浊响，沉闷，清脆}
- 纹理：{清晰，稍糊，模糊}
- 脐部：{凹陷，稍凹，平坦}
- 触感：{硬滑，软粘}

现在需要计算数据集在每个（属性，候选值）的划分准则取值。

Inference (e.g. ID3)

- 计算原始数据集 $\mathcal{D}$ 上的信息熵：

$$\begin{aligned}\mathrm{Ent}(\mathcal{D}) &= -\sum_{k=1}^{K} p_k \log p_k \\ &= -\sum_{k=1}^{2} p_k \log p_k \\ &= -\big(\frac{8}{17}\log\frac{8}{17} + \frac{9}{17}\log\frac{9}{17}\big) = 0.998\end{aligned}$$

- 计算以属性“色泽“作为分割点的信息增益 Gain($\mathcal{D}$, 色泽)：
  - 1. 计算 $\mathcal{D}_{色泽=青绿}$ 上的信息熵：

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	是
10	青绿	硬挺	清脆	清晰	平坦	软粘	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

$$\mathrm{Ent}(\mathcal{D}_{色泽=青绿}) = -(\frac{3}{6}\log\frac{3}{6} + \frac{3}{6}\log\frac{3}{6}) = 1.0$$

  - 2. 计算 $\mathcal{D}_{色泽=乌黑}$ 上的信息熵：

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否

$$\mathrm{Ent}(\mathcal{D}_{色泽=乌黑}) = -(\frac{4}{6}\log\frac{4}{6} + \frac{2}{6}\log\frac{2}{6}) = 0.918$$

  - 3. 计算 $\mathcal{D}_{色泽=浅白}$ 上的信息熵：

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
11	浅白	硬挺	清脆	模糊	平坦	硬滑	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	否

$$\mathrm{Ent}(\mathcal{D}_{色泽=浅白}) = -(\frac{1}{5}\log\frac{1}{5} + \frac{4}{5}\log\frac{4}{5}) = 0.722$$

  - 4. 计算信息增益：

$$\begin{aligned}\mathrm{Gain}(\mathcal{D}, 色泽) &= \mathrm{Ent}(\mathcal{D}) - \sum_{v=1}^{V}\frac{|\mathcal{D}_{色泽=v}|}{|\mathcal{D}|}\mathrm{Ent}(\mathcal{D}_{色泽=v}) \\ &= 0.998 - (\frac{6}{17}\times 1.000 + \frac{6}{17}\times 0.918 + \frac{5}{17}\times 0.722) \\ &= 0.109\end{aligned}$$

- 计算以属性“根蒂“作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{根蒂})$：

$$\text{Gain}(\mathcal{D}, \text{根蒂}) = 0.143$$

- 计算以属性“敲声“作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{敲声})$：

$$\text{Gain}(\mathcal{D}, \text{敲声}) = 0.141$$

- 计算以属性“纹理“作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{纹理})$：

$$\text{Gain}(\mathcal{D}, \text{纹理}) = 0.381$$

- 计算以属性“脐部“作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{脐部})$：

$$\text{Gain}(\mathcal{D}, \text{脐部}) = 0.289$$

- 计算以属性“触感“作为分割点的信息增益 $\text{Gain}(\mathcal{D}, \text{触感})$：

$$\text{Gain}(\mathcal{D}, \text{触感}) = 0.006$$

- 选择最有划分属性：纹理，并分割数据集：

  - $\mathcal{D}_{\text{纹理 = 清晰}}$：

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	是
10	青绿	硬挺	清脆	清晰	平坦	软粘	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否

  - $\mathcal{D}_{\text{纹理 = 稍糊}}$：

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

  - $\mathcal{D}_{\text{纹理 = 模糊}}$：

Example No.	色泽	根蒂	敲声	纹理	脐部	触感	好瓜？
11	浅白	硬挺	清脆	模糊	平坦	硬滑	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	否

- 依次对 $\mathcal{D}_{\text{纹理 = 清晰}}$，$\mathcal{D}_{\text{纹理 = 稍糊}}$ 和 $\mathcal{D}_{\text{纹理 = 模糊}}$ 重复上述对 $\mathcal{D}$ 的操作...

注意 ：信息熵描述的是一个概率分布的期望信息量，即**一个概率分布的不确定性：数值越大，不确定性越高**。决策树中希望通过属性对原始数据进行划分从而降低信息熵，因此一般划分前的信息熵大于划分后的信息熵，因此信息增益是正数。

ID3 $\Rightarrow$ C4.5 使用 Information Gain 作为划分准则的时候，**倾向于选择值域 $|V_a|$ 多的属性** a，然而很多时候，这样的划分方式会使得模型陷入**过拟合，不具有泛化能力，使得模型无法对新样本作出有效预测。**因此引入了信息增益率。

注意：C4.5 不是直接使用 Gain-ratio 作为选择划分属性的依据 Gain-ratio 倾向选择值域 $|V_a|$ 较少的属性 a，因此 C4.5 算法不是直接使用 Gain-ratio 作为选择划分属性的依据，而是使用**启发式方法**，譬如：先从候选划分属性中找出 Gain 大于平均水平的属性，然后再从中选择 Gain-ratio 最高的。

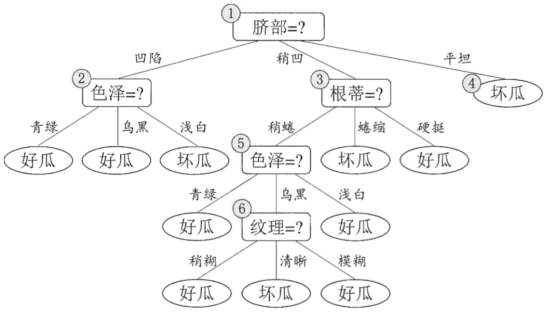


图 44: Decision tree with pre-pruning

问题：如何处理连续型特征（属性）？ 直观的想法是：将连续型属性转化为离散型属性。假设数据集 $\mathcal{D}$ 中连续属性 a 包含了 n 个不同的取值，则：

- 从小到大排序 n 个取值：$\{v_1, v_2, ..., v_n\}$
- 构造属性 a 的候选划分点集合：$\mathrm{T}_a = \left\{ \frac{v_i + v_{i+1}}{2} | 1 \leq i \leq n-1 \right\}$
- 根据候选划分点集合中的每个元素 $t \in \mathrm{T}_a$ 对样本进行二分：$\mathcal{D}_{a \leq t}$ 和 $\mathcal{D}_{a>t}$
- 计算 Gain（或者 Gain-ratio...）：

$$\begin{aligned} \mathrm{Gain}(\mathcal{D}, a) &= \max_{t \in \mathrm{T}_a} \mathrm{Gain}(\mathcal{D}, a, t) \\ &= \max_{t \in \mathrm{T}_a} \mathrm{Ent}(\mathcal{D}) - \sum_{\lambda \in \{+,-\}} \frac{|\mathcal{D}_t^\lambda|}{|\mathcal{D}|} \mathrm{Ent}(\mathcal{D}_t^\lambda) \end{aligned}$$

譬如：全部训练样本中，某连续型属性 a 的所有可能取值为：$\{3, 7, 1, 9\}$，那么排序后为 $\{1, 3, 7, 9\}$，然后构造候选划分点集合为 $\{2, 5, 8\}$...

12.1.3 Model pruning

剪枝（pruning）分为两种：

- Pre-pruning: 在决策树生成过程中，对每个节点在划分前进行估计，若当前节点的划分不能带来决策树泛化性能提升（e.g. 验证集上计算 Accuracy），则停止划分并将当前节点作为叶节点。
- Post-pruning: 先利用训练集完整地生成一棵决策树，然后自底向上地对非叶子结点 v 进行考察，若将以该结点 v 为根结点的子树替换为叶结点，并能带来决策树泛化性能（e.g. 验证集上计算 Accuracy）的提升，则将该子树替换为叶子结点。

(a) pre-pruning (b) post-pruning

图 45: Decision tree without pruning

注意：决策树泛化性能的评价一定得在验证集上进行。 优缺点对比：

- Pre-pruning:
  - 优点：时间开销相对于 Post-pruning 更小（因为是在训练树的过程中剪枝）
  - 缺点：容易欠拟合
- Post-pruning:
  - 优点：相对于 Pre-pruning 的决策树保留了更多的分支，因此欠拟合风险更小，泛化性能往往优于 Pre-pruning 决策树
  - 缺点：时间开销更大

12.2 Regression Tree

12.2.1 Model training

Algorithm 11: Regression Tree Algorithm

Input: Training dataset $\{(\boldsymbol{x^{(i)}}, y^{(i)})\}_{i=1}^{N}$, where $\boldsymbol{x^{(i)}} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$.

Output: The hypothesis $h(\boldsymbol{x})$.

for *j = 1, ..., d* **do**
  for *each candidate split value of feature j:* $v_{j,l} \in \mathbb{R}$ **do**
    Split the training dataset into **two parts**: $\mathrm{I}_< = \{i : x_j^{(i)} < v_{j,l}\}$ and $\mathrm{I}_> = \{i : x_j^{(i)} \geq v_{j,l}\}$
    Estimate the prediction values of two parts: $\hat{y}_< = \frac{\sum_{i \in \mathrm{I}_<} y^{(i)}}{|\mathrm{I}_<|}$ and $\hat{y}_> = \frac{\sum_{i \in \mathrm{I}_>} y^{(i)}}{|\mathrm{I}_>|}$
    Quality of split $v_{j,l}$ is measured by the **squared loss**: $\sum_{i \in \mathrm{I}_<} (y^{(i)} - \hat{y}_<)^2 + \sum_{i \in \mathrm{I}_>} (y^{(i)} - \hat{y}_>)^2$
  end
end
Choose the split point with minimal loss
Recurse on both children, with $\{(\boldsymbol{x^{(i)}}, y^{(i)})\}_{i \in \mathrm{I}_<}$ and $\{(\boldsymbol{x^{(i)}}, y^{(i)})\}_{i \in \mathrm{I}_>}$.

13 Ensemble Learning

13.1 Random Forest

13.1.1 Model formulation

Algorithm 12: Random Forest Regression Algorithm[?]

Input: Training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$;
Column sample rate $\alpha \in (0, 1]$;
Number of base tree model (CART): T.

Output: The final hypothesis $G(x) = \frac{1}{T}\sum_{t=1}^{T} h_t(x)$.

Initialize the set of weak regressors as $H = \{\}$
for *t = 1, ..., T* **do**
  Sampling with replacement n training samples from raw training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, generate new training dataset: $\mathcal{X}_t \in \mathbb{R}^{n\times d}, \mathcal{Y}_t \in \mathbb{R}^n$
  Randomly feature sample with $(d' = \alpha d)$ specific features for all samples in $\mathcal{X}_t$, generate new training samples $\mathcal{X}'_t \in \mathbb{R}^{n\times d'}$
  Train the weak tree regressor $h_t(x)$, providing it with the training dataset $(\mathcal{X}'_t, \mathcal{Y}_t)$
  Append weak regressor $h_t(x)$ to H.
end

行采样 采用**bootstrap**的方法，即采样次数等同于样本个数 n，但是每次采样都是有放回的采样。类似“盲人摸象”。

列采样 指定一个常数 $d' <= d$，在一次树模型的训练过程中，随机地从 d 维特征中选择 d' 维，切片数据集形成维度缩减的数据集用于训练。

不剪枝 一般训练随机森林的树模型不需要剪纸操作，因为对于单一的树模型，剪纸操作是为了防止过拟合，但是随机森林里，因为最后要平均，减小了 Variance，所以无需剪枝减小 Variance。

13.1.2 Random forest classification

预测值为“多数投票”

13.1.3 Random forest regression

预测值为“多值均值”

13.2 Adaboost

13.2.1 Adaboost for binary classification

Algorithm 13: Adaboost for Binary Classification Task

Input: Training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \{-1, +1\}$;
Weak hypothesis $h(x)$;
Number of weak hypothesis T.

Output: The final hypothesis $G(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$.

Initialize samples weight distribution: $\omega_1^{(i)} = \frac{1}{n}$, for all $i \in \{1, ..., n\}$

for $t = 1, ..., T$ **do**

Train the weak learner $h_t(x)$, providing it with the distribution $\boldsymbol{\omega}_t$

Get weak hypothesis $h_t(\boldsymbol{x}) : \mathbb{R}^d \to \{-1, +1\}$ with error:

$$\epsilon_t = \Pr_{\sim \boldsymbol{\omega}_t}(h_t(x^{(i)}) \neq y^{(i)}) = \sum_{i=1}^{N} I(h_t(x^{(i)}) \neq y^{(i)}) \cdot \omega_t^{(i)}$$

Choose $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$

Update sample weight distribution over all training samples:

$$\omega_{t+1}^{(i)} = \begin{cases} \frac{\omega_t^{(i)}}{Z_t} \times e^{-\alpha_t} & \text{if } h_t(x^{(i)}) = y^{(i)} \\ \frac{\omega_t^{(i)}}{Z_t} \times e^{\alpha_t} & \text{if } h_t(x^{(i)}) \neq y^{(i)} \end{cases}$$

where Z_t is a normalization factor, chosen so that $\boldsymbol{\omega}_{t+1}$ will be a distribution, namely:
$Z_t = \sum_{i=1}^{N} \omega_t^{(i)} \exp\big(-\alpha_t y^{(i)} h_t(x^{(i)})\big)$

end

可见在 Adaboost 的学习过程中，我们先初始化样本权重分布，然后开始迭代，在每一步迭代 t 中，我们需要做四件事情：

1. 在当前样本权重分布 $\boldsymbol{\omega}_t$ 的情况下训练一个弱学习器 $h_t(x)$
2. 计算 $h_t(x)$ 在分布 $\boldsymbol{\omega}_t$ 下的训练集 $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ 上的误差率 ϵ_t
3. 计算系数 α_t
4. 更新样本权重分布 $\boldsymbol{\omega}$

问题：怎样使用样本权重分布 ω？

- Resampling (e.g. Roulette wheel, random number of uniform distribution)
- Add sample weight in loss function (min $\frac{1}{N} \sum_{i=1}^{N} \omega_t^{(i)} \mathcal{L}(y^{(i)}, h_t(\boldsymbol{x^{(i)}}))$)

问题：为何弱分类器的系数 $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$？ 这是通过一个优化过程的得到的，解释推导如下：

Theorem 2 (The Training Error Bounds of Final Classifier)**.**

$$\frac{1}{N} \sum_{i=1}^{N} I(G(x^{(i)}) \neq y^{(i)}) \leq \frac{1}{N} \sum_{i=1}^{N} \exp(-y^{(i)} f(x^{(i)})) = \prod_{t=1}^{T} z_t$$

- n: Number of training samples;
- T: Number of weak hypothesises;
- $f(x)$: The final hypothesis, $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$;
- $G(x)$: The final classifier $G(x) = sign(f(x)) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$.

Proof. 先后证明上述连续不等式左右两个部分，具体证明过程如下：

1. For left part:
   - if $G(x^{(i)}) \neq y^{(i)}$:

$$\begin{aligned} I(G(x^{(i)}) \neq y^{(i)}) = 1 \ &\Rightarrow \ y^{(i)} f(x^{(i)}) < 0 \\ &\Rightarrow \ -y^{(i)} f(x^{(i)}) > 0 \\ &\Rightarrow \ \exp\left(y^{(i)} f(x^{(i)})\right) > 1 \end{aligned}$$

   - if $G(x^{(i)}) = y^{(i)}$:

$$\begin{aligned} y^{(i)} f(x^{(i)}) > 0 \ &\Rightarrow \ -y^{(i)} f(x^{(i)}) < 0 \\ &\Rightarrow \ \exp\left(y^{(i)} f(x^{(i)})\right) \in (0, 1) \end{aligned}$$

   - Therefore:

$$\frac{1}{N} \sum_{i=1}^{N} I(G(x^{(i)}) \neq y^{(i)}) \leq \frac{1}{N} \sum_{i=1}^{N} \exp(-y^{(i)} f(x^{(i)}))$$

2. For right part:

$$\begin{aligned} & \frac{1}{N} \sum_{i=1}^{N} \exp(-y^{(i)} f(x^{(i)})) \\ = & \frac{1}{N} \sum_{i=1}^{N} \exp(-y^{(i)} \sum_{t=1}^{T} \alpha_t h_t(x^{(i)})) \\ = & \sum_{i=1}^{N} \omega_1^{(i)} \exp(\sum_{t=1}^{T} -\alpha_t y^{(i)} h_t(x^{(i)})) \quad (\omega_1^{(i)} = \frac{1}{n}) \\ = & \sum_{i=1}^{N} \omega_1^{(i)} \prod_{t=1}^{T} \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) \\ = & Z_1 \sum_{i=1}^{N} \frac{\omega_1^{(i)} \exp(-\alpha_1 y^{(i)} h_1(x^{(i)}))}{Z_1} \prod_{t=2}^{T} \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) \\ = & Z_1 \sum_{i=1}^{N} \omega_2^{(i)} \prod_{t=2}^{T} \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) \\ = & Z_1 Z_2 \sum_{i=1}^{N} \omega_3^{(i)} \prod_{t=3}^{T} \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) \quad (\because \omega_{t+1}^{(i)} = \frac{\omega_t^{(i)} \exp(-\alpha_t y^{(i)} h_t(x^{(i)}))}{Z_t}) \\ = & Z_1 Z_2 ... Z_{T-1} \sum_{i=1}^{N} \omega_T^{(i)} \exp(-\alpha_T y^{(i)} h_T(x^{(i)})) = \prod_{t=1}^{T} Z_t \end{aligned}$$

因此有：

$$\frac{1}{N}\sum_{i=1}^{N} I(G(x^{(i)}) \neq y^{(i)}) \leq \frac{1}{N}\sum_{i=1}^{N} \exp(-y^{(i)} f(x^{(i)})) = \prod_{t=1}^{T} z_t$$

而后优化上述不等式的上界：

$$\begin{aligned} & \min \prod_{t=1}^{T} Z_t \\ \Rightarrow \quad & \min_{\alpha_t} Z_t \quad \text{(转换为独立优化任务)} \\ \Rightarrow \quad & \min_{\alpha_t} \sum_{i=1}^{N} \omega_t^{(i)} \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) \end{aligned}$$

因为是凸函数，所以最优解在梯度为 0 处，即：

$$\begin{aligned} & \frac{\partial \sum_{i=1}^{N} \omega_t^{(i)} \exp(-\alpha_t y^{(i)} h_t(x^{(i)}))}{\partial \alpha_t} = 0 \\ \Rightarrow \quad & \sum_{i=1}^{N} \omega_t^{(i)} \big(- y^{(i)} h_t(x^{(i)}) \big) \exp(-\alpha_t y^{(i)} h_t(x^{(i)})) = 0 \\ \Rightarrow \quad & \alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t}) \end{aligned}$$

13.2.2 Adaboost for multi classification

Algorithm 14: Adaboost.M1 for Multiclass Classification[**?**]

Input: Training dataset $\{\mathcal{X}, \mathcal{Y}\}$, where $\mathcal{X} \in \mathbb{R}^{n\times d}$, $\mathcal{Y} \in \mathbb{R}^n$, and $x^{(i)} \in \mathcal{X}$, $y^{(i)} \in \mathcal{Y} = \{1, 2, ..., K\}$;
Weak hypothesis $h(x)$;
Number of weak hypothesis T.

Output: The final hypothesis $G(x) = \arg\max_{y\in\mathcal{Y}} \sum_{t:h_t(x)=y} \log\frac{1}{\beta_t}$.

Initialize samples weight distribution: $\omega_i^1 = \frac{1}{n}$, for all $i \in n$

for t = *1, ..., T* **do**

Train the weak classifier $h_t(x)$, providing it with the distribution $\boldsymbol{\omega}^t$

Get weak hypothesis $h_t(x) : \mathcal{X} \to \mathcal{Y}$ with error:

$$\epsilon_t = \Pr_{\sim\boldsymbol{\omega}^t}(h_t(x^{(i)}) \neq y^{(i)}) = \sum_{i:h_t(x^{(i)})\neq y^{(i)}} \omega_i^t$$

If $\epsilon_t > \frac{1}{2}$, then set $T = t - 1$ and abort loop

Choose $\beta_t = \frac{\epsilon_t}{1-\epsilon_t} \in (0, 1)$

Update distribution $\boldsymbol{\omega}^t$:

$$\omega_{t+1}^{(i)} = \begin{cases} \frac{\omega_i^t}{Z_t} \times \beta_t & \text{if } h_t(x^{(i)}) = y^{(i)} \\ \frac{\omega_i^t}{Z_t} \times 1 & \text{if } h_t(x^{(i)}) \neq y^{(i)} \end{cases}$$

end

Adaboost.M1 的解释说明

- Adaboost.M1 算法中，要求每一个基分类器在多分类上的 0-1 损失都小于 $\frac{1}{2}$ (实际中这个条件有些不太好)；
- 当 $0 < \epsilon_t < \frac{1}{2}$ 时，ϵ_t 越小，$\beta_t \in (0, 1)$ 越小，$\frac{1}{\beta_t} \in (1, +\text{inf})$ 越大，那么 $\log\frac{1}{\beta_t} \in (1, +\text{inf})$ 越大，那么在最终分类器 $G(x)$ 中，该分类正确的弱分类器“话语权”更大；
- 因为 $\beta_t < 1$，所以正确分类的样本权重相对错误分类的样本而减小。

13.2.3 Adaboost for regression

Algorithm 15: Adaboost.R[**?**]

Input: Training dataset $\{\mathcal{X}, \mathcal{Y}\}$, where $\mathcal{X} \in \mathbb{R}^{n \times d}$, $\mathcal{Y} \in \mathbb{R}^n$, and $x^{(i)} \in \mathcal{X}$, $y^{(i)} \in \mathcal{Y}$;
Weak hypothesis $h(x)$;
Number of weak hypothesis T.

Output: The final hypothesis $G(x) = \sum_{t=1}^{T} \ln(\frac{1}{\alpha_t}) h_t(x)$.

Initialize samples weight distribution: $\omega_i^1 = \frac{1}{n}$, for all $i \in n$

for *t = 1, ..., T* **do**

Train the weak regressor $h_t(x)$, providing it with the distribution $\boldsymbol{\omega}^t$

Calculate the max error of all training samples:

$$E_t = \max |y^{(i)} - h_t(x^{(i)})|,\ i \in \{1, ..., n\}$$

Calculate the relative errors $e_t(i)$ over all training samples: $e_t(i) = \frac{(y^{(i)} - h_t(x^{(i)}))^2}{E_t^2}$(for square error), or $e_t(i) = \frac{|y^{(i)} - h_t(x^{(i)})|}{E_t}$(for absolute error)

Calculate the error rate of regressor $h_t(x)$: $\epsilon_t = \sum_{i=1}^{N} \omega_i^t e_t(i)$

Calculate the weight of weak hypothesis $h_t(x)$: $\alpha_t = \frac{\epsilon_t}{1 - \epsilon_t}$

Update distribution $\boldsymbol{\omega}^t$: $\omega_i^{t+1} = \frac{\omega_i^t \alpha_t^{1 - e_t(i)}}{Z_t}$, $Z_t = \sum_{i=1}^{m} \omega_i^t \alpha_t^{1 - e_t(i)}$

end

13.3 Gradient Boosting Machine

13.3.1 Model formulation

Algorithm 16: Gradient Boosting Machine Algorithm[?]

Input: Training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$;
Number of iterations M;
A differentiable loss function $\mathcal{L}(y, F(\boldsymbol{x}))$.

Output: The final hypothesis $F_M(x)$.

Initialize model with a constant value:

$$F_0(\boldsymbol{x}) = \arg\min_{\alpha} \sum_{i=1}^{N} \mathcal{L}(y^{(i)}, \alpha)$$

for *m = 1, ..., M* **do**

Compute so-called pseudo-residuals:

$$r_m^{(i)} = -\Big[\frac{\partial \mathcal{L}(y^{(i)}, F(x^{(i)}))}{\partial F(x^{(i)})}\Big]_{F(x)=F_{m-1}(x)}, i = 1, ..., n$$

Fit a base learner (e.g. tree, lr) $h_m(x)$ to pseudo-residuals, namely train it using the training set:

$$\mathcal{D} = \{(x^{(i)}, r_m^{(i)})\}_{i=1}^{N}$$

Compute multiplier α_m by solving the following one-dimensional optimization problem:

$$\alpha_m = \arg\min_{\alpha} \sum_{i=1}^{N} \mathcal{L}(y^{(i)}, F_{m-1}(x^{(i)}) + \alpha h_m(x^{(i)}))$$

Update the model:

$$F_m(\boldsymbol{x}) := F_{m-1}(\boldsymbol{x}) + \alpha_m \cdot h_m(x)$$

end

可见在 GBM 的学习中，我们先初始化模型为一个常数，然后进入迭代过程。在每一步迭代中，我们需要做四件事情 [?]：

1. 计算出当前每个训练样本的残差 $r_m^{(i)}$；
2. 运用残差作为 label 训练一个弱学习器 $h_m(\boldsymbol{x})$；
3. 计算一个一维优化问题（线搜索）得到当前弱学习器的乘子系数 α_m；
4. 更新模型 $F_m(\boldsymbol{x})$.

13.3.2 GBM regression

任意训练样本 $(\boldsymbol{x^{(i)}}, y^{(i)})$ 的 so-called pseudo-residuals：

$$\because \frac{\partial \mathcal{L}(y^{(i)}, F(\boldsymbol{x^{(i)}}))}{\partial F(\boldsymbol{x^{(i)}})} = \frac{\partial}{\partial F(\boldsymbol{x^{(i)}})} \frac{1}{2}\Big(y^{(i)} - F(\boldsymbol{x^{(i)}})\Big)^2$$

$$= -\Big(y^{(i)} - F(\boldsymbol{x^{(i)}})\Big)$$

$$\begin{aligned}\therefore r_m^{(i)} &= -\Big[\frac{\partial \mathcal{L}(y^{(i)}, F(\boldsymbol{x^{(i)}}))}{\partial F(\boldsymbol{x^{(i)}})}\Big]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})} \\ &= y^{(i)} - F_{m-1}(\boldsymbol{x^{(i)}}) \quad \text{(数值残差)}\end{aligned}$$

13.3.3 GBM classification

因为分类问题的损失函数（针对单一样本 $(\boldsymbol{x}, \boldsymbol{y})$）为 Cross Entropy，即：

$$\mathcal{L}(\boldsymbol{y}, F(\boldsymbol{x})) = -\boldsymbol{y} \cdot \log F(\boldsymbol{x}) = -\sum_{k=1}^{K} y_k \log F(x_k)$$

因此任意训练样本 $(\boldsymbol{x^{(i)}}, y^{(i)})$ 的 so-called pseudo-residuals：

$$r_m^{(i)} = -\Big[\frac{\partial \mathcal{L}(y^{(i)}, F(\boldsymbol{x^{(i)}}))}{\partial F(\boldsymbol{x^{(i)}})}\Big]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})} = y^{(i)} - F_{m-1}(\boldsymbol{x^{(i)}}) \quad \text{(概率残差)}$$

13.4 XGBoost

13.4.1 Model prediction

XGBoost[?] 模型的预测函数为：

$$F(\boldsymbol{x}) = \sum_{t=1}^{T} f_t(\boldsymbol{x})$$

13.4.2 Model training

Dataset 训练数据的表达如下：

- 训练数据集：$\mathcal{D} = \{(\boldsymbol{x^{(i)}}, y^{(i)})\}_{i=1}^{N}$
- 特征向量：$\boldsymbol{x^{(i)}} \in \mathbb{R}^d$
- 样本标签：$y^{(i)}$

Model details 我们做如下符号表示：

- $f_t(\boldsymbol{x})$: 第 t 轮迭代训练所得的弱学习器
- $\widehat{y}_t$: 第 t 轮**训练结束后**所得的强学习器，即：

$$\widehat{y}_t = \sum_{h=1}^{t} f_h(\boldsymbol{x})$$

第 t 轮结束后需要学习到的强学习器（预测函数）可表示为：

$$\begin{aligned} \widehat{y}_t &= \widehat{y}_{t-1} + f_t(\boldsymbol{x}) \\ &= \sum_{h=1}^{t-1} f_h(\boldsymbol{x}) + f_t(\boldsymbol{x}) \end{aligned}$$

Loss function 一般机器学习模型目标函数的表示结构如下：

$$Obj(\boldsymbol{\Theta}) = \mathcal{L}(\boldsymbol{\Theta}) + \Omega(\boldsymbol{\Theta})$$

其中：

- $l(\boldsymbol{\Theta})$: 损失函数，衡量模型拟合数据的能力
- $\Omega(\boldsymbol{\Theta})$: 正则化项 (Regularization)，防止模型过拟合 (overfitting)

采取此结构，Xgboost 在第 t 轮迭代的学习目标为：

$$\begin{aligned} & \min \sum_{i=1}^{N} \mathcal{L}\big(y^{(i)}, \widehat{y}_t^{(i)}\big) + \Omega(\widehat{y}_t) \\ \Rightarrow \quad & \min \sum_{i=1}^{N} \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)} + f_t(\boldsymbol{x^{(i)}})\Big) + \Omega\Big(\sum_{h=1}^{t} f_h(\boldsymbol{x})\Big) \end{aligned}$$

$$
\begin{aligned}
\Rightarrow \quad & \min \sum_{i=1}^{N} \mathcal{L}\Big(y^{(i)},\ \underbrace{\widehat{y}_{t-1}^{(i)}}_{\text{已知常数}} + f_t(\boldsymbol{x^{(i)}})\Big) + \Omega\Big(\underbrace{\sum_{h=1}^{t-1} f_h(\boldsymbol{x})}_{\text{已知常数}} + f_t(\boldsymbol{x})\Big) \\
\Rightarrow \quad & \min \sum_{i=1}^{N} \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)} + f_t(\boldsymbol{x^{(i)}})\Big) + \Omega\big(f_t(\boldsymbol{x})\big) + \text{constant} \\
\Rightarrow \quad & \min \sum_{i=1}^{N} \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)} + f_t(\boldsymbol{x^{(i)}})\Big) + \Omega\big(f_t(\boldsymbol{x})\big)
\end{aligned}
$$

根据泰勒展开公式（**泰勒二阶近似展开**）：

$$
\begin{aligned}
& \because \quad f(x+\Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2 \\
& \therefore \quad \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)} + f_t(\boldsymbol{x^{(i)}})\Big) \simeq \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)}\Big) + \frac{\partial \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)}\Big)}{\partial\, \widehat{y}_{t-1}^{(i)}} f_t(\boldsymbol{x^{(i)}}) + \frac{1}{2} \frac{\partial \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)}\Big)}{\partial^2\, \widehat{y}_{t-1}^{(i)}} f_t^2(\boldsymbol{x^{(i)}})
\end{aligned}
$$

注意：因为 $\mathcal{L}\Big(y^{(i)}, \underbrace{\widehat{y}_{t-1}^{(i)}}_{\text{对应}x} + \underbrace{f_t(\boldsymbol{x^{(i)}})}_{\text{对应}\Delta x}\Big)$，所以 $f'(x)$ 对应 $\frac{\partial \mathcal{L}\left(y^{(i)}, \widehat{y}_{t-1}^{(i)}\right)}{\partial\, \widehat{y}_{t-1}^{(i)}}$，而不是 $\frac{\partial \mathcal{L}\left(y^{(i)}, \widehat{y}_{t-1}^{(i)}\right)}{\partial\, f_t(\boldsymbol{x^{(i)}})}$

如果不好理解，考虑回归问题的平方误差损失函数，那么：

$$
\begin{aligned}
\frac{\partial \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)}\Big)}{\partial\, \widehat{y}_{t-1}^{(i)}} &= \frac{\partial}{\partial\, \widehat{y}_{t-1}^{(i)}} \frac{1}{2}\Big(y^{(i)} - \widehat{y}_{t-1}^{(i)}\Big)^2 \\
&= -\Big(y^{(i)} - \widehat{y}_{t-1}^{(i)}\Big) \\
\frac{\partial \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)}\Big)}{\partial^2\, \widehat{y}_{t-1}^{(i)}} &= -1
\end{aligned}
$$

代入目标函数继续化简：

$$
\begin{aligned}
& \min \sum_{i=1}^{N} \mathcal{L}\big(y^{(i)}, \widehat{y}_t^{(i)}\big) + \Omega(\widehat{y}_t) \\
\Rightarrow \quad & \min \sum_{i=1}^{N} \underbrace{\mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)} + f_t(\boldsymbol{x^{(i)}})\Big)}_{\text{利用泰勒展开等价替换}} + \Omega(f_t) \\
\Rightarrow \quad & \min \sum_{i=1}^{N} \left(\mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)}\Big) + \underbrace{\frac{\partial \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)}\Big)}{\partial\, \widehat{y}_{t-1}^{(i)}}}_{\text{记作 } g_{t-1}^{(i)}} f_t(\boldsymbol{x^{(i)}}) + \frac{1}{2} \underbrace{\frac{\partial \mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)}\Big)}{\partial^2\, \widehat{y}_{t-1}^{(i)}}}_{\text{记作 } h_{t-1}^{(i)}} f_t^2(\boldsymbol{x^{(i)}})\right) + \Omega(f_t) \\
\Rightarrow \quad & \min \sum_{i=1}^{N} \left(\underbrace{\mathcal{L}\Big(y^{(i)}, \widehat{y}_{t-1}^{(i)}\Big)}_{\text{常数}} + g_{t-1}^{(i)} f_t(\boldsymbol{x^{(i)}}) + \frac{1}{2} h_{t-1}^{(i)} f_t^2(\boldsymbol{x^{(i)}})\right) + \Omega(f_t) \\
\Rightarrow \quad & \min \sum_{i=1}^{N} \left(g_{t-1}^{(i)} f_t(\boldsymbol{x^{(i)}}) + \frac{1}{2} h_{t-1}^{(i)} f_t^2(\boldsymbol{x^{(i)}})\right) + \Omega(f_t)
\end{aligned}
$$

问题：为啥要这样费劲地用泰勒展开式写出来？

- **理论角度**：把 $\mathcal{L}\left(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t(\boldsymbol{x}^{(i)})\right)$ 用泰勒展开写出来我们就明明白白地知道自己要学什么了，怎么收敛。
- **工程角度**：目标函数的优化(学习)只取决于 $g^{(i)}$ 和 $h^{(i)}$ 所在的项，如果现在要求损失函数不是 squared loss 了，而是 logistic loss，那么这样把 $g^{(i)}$ 和 $h^{(i)}$ 显示地写出来就很有利于模型的便捷实现。

注意：XGBoost 使用 Taylor 二阶展开，一般的 GBM 只使用一阶展开

然而新的问题又来了

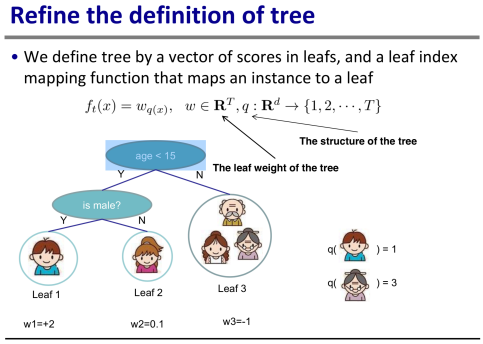


图 47: Refine the defination of tree

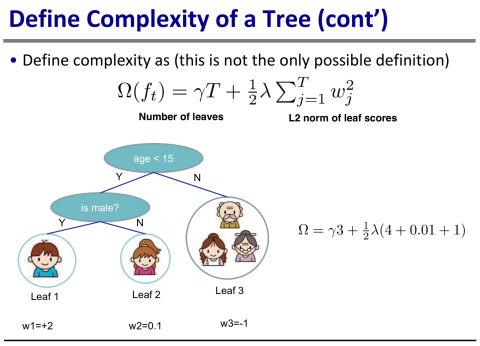


图 48: Define complexity of tree

1. **如何表示 $f_t(\boldsymbol{x})$？**假定**树的结构已确定**，那么有

$$f_t(\boldsymbol{x}) = w_{q(\boldsymbol{x})}$$

符号解释如下：

- T：叶子结点个数
- $\boldsymbol{w} \in \mathbb{R}^T$
- q： $\mathbb{R}^d \to \{1, ..., T\}$

步骤解释如下：

- $\boldsymbol{x}$ 根据函数 $q(\cdot)$ 找到对应的叶子节点 $q(\boldsymbol{x})$
- 函数 f_t 返回叶子结点 $q(\boldsymbol{x})$ 上的估计值

2. **如何定义 $\Omega(f_t)$?**

$$\Omega\big(f_t(\boldsymbol{x})\big) = \gamma T + \frac{\lambda}{2}\sum_{j=1}^{T} w_j^2$$

其中：

- γT: 控制叶子结点个数
- $\frac{\lambda}{2}\sum_{j=1}^{T} w_j^2$: 控制叶子结点权重的大小

该定义陈天奇给出，所以并不是唯一可行的定义。

代入目标函数继续化简：

$$\begin{aligned}
& \min \sum_{i=1}^{N} \mathcal{L}\big(y^{(i)}, \widehat{y}_t^{(i)}\big) + \Omega(\widehat{y}_t) \\
\Rightarrow \quad & \min \sum_{i=1}^{N} \left(g_{t-1}^{(i)}\, f_t(\boldsymbol{x^{(i)}}) + \frac{1}{2} h_{t-1}^{(i)}\, f_t^2(\boldsymbol{x^{(i)}}) \right) + \Omega(f_t) \\
\Rightarrow \quad & \min \sum_{i=1}^{N} \Bigg(\underbrace{g_{t-1}^{(i)}\, w_{q(\boldsymbol{x^{(i)}})} + \frac{1}{2} h_{t-1}^{(i)}\, w_{q(\boldsymbol{x^{(i)}})}^2}_{\text{又不知道怎么化简处理了}} \Bigg) + \gamma T + \frac{\lambda}{2} \sum_{j=1}^{T} w_j^2
\end{aligned}$$

因为 n 个训练样本都要落在叶子结点上，那么可以将上述损失函数表示为**各个叶子结点损失函数之和**，即第 j 个叶子结点的训练样本集合为：

$$I_j = \{ i \,|\, q(\boldsymbol{x^{(i)}}) = j \}$$

代入上式中可得：

$$\begin{aligned}
& \min \sum_{i=1}^{N} \mathcal{L}\big(y^{(i)}, \widehat{y}_t^{(i)}\big) + \Omega(\widehat{y}_t) \\
\Rightarrow \quad & \min \sum_{i=1}^{N} \left(g_{t-1}^{(i)}\, w_{q(\boldsymbol{x^{(i)}})} + \frac{1}{2} h_{t-1}^{(i)}\, w_{q(\boldsymbol{x^{(i)}})}^2 \right) + \gamma T + \frac{\lambda}{2} \sum_{j=1}^{T} w_j^2 \\
\Rightarrow \quad & \min \sum_{j=1}^{T} \sum_{i \in I_j} \Big(g_{t-1}^{(i)} w_j + \frac{1}{2} h_{t-1}^{(i)} w_j^2 \Big) + \gamma T + \frac{\lambda}{2} \sum_{j=1}^{T} w_j^2 \quad \text{（最关键步骤）} \\
\Rightarrow \quad & \min \sum_{j=1}^{T} \left(\sum_{i \in I_j} g_{t-1}^{(i)} w_j + \frac{1}{2} \sum_{i \in I_j} h_{t-1}^{(i)} w_j^2 \right) + \gamma T + \frac{\lambda}{2} \sum_{j=1}^{T} w_j^2 \\
\Rightarrow \quad & \min \sum_{j=1}^{T} \left(w_j \sum_{i \in I_j} g_{t-1}^{(i)} + \frac{1}{2} w_j^2 \Big(\sum_{i \in I_j} h_{t-1}^{(i)} + \lambda \Big) \right) + \gamma T
\end{aligned}$$

那么之后的任务就是独立地优化 T 个二次函数。记：

$$\begin{aligned}
G_{t-1}^{(j)} &= \sum_{i \in I_j} g_{t-1}^{(i)} \\
H_{t-1}^{(j)} &= \sum_{i \in I_j} h_{t-1}^{(i)}
\end{aligned}$$

所以上述目标函数又可以化简为：

$$\begin{aligned}
& \min \sum_{i=1}^{N} \mathcal{L}\big(y^{(i)}, \widehat{y}_t^{(i)}\big) + \Omega(\widehat{y}_t) \\
\Rightarrow \quad & \min \sum_{j=1}^{T} \left(w_j \sum_{i \in I_j} g_{t-1}^{(i)} + \frac{1}{2} w_j^2 \Big(\sum_{i \in I_j} h_{t-1}^{(i)} + \lambda \Big) \right) + \gamma T \\
\Rightarrow \quad & \min \sum_{j=1}^{T} \left(w_j G_{t-1}^{(j)} + \frac{1}{2} w_j^2 \Big(H_{t-1}^{(j)} + \lambda \Big) \right) + \gamma T
\end{aligned}$$

上述式子即为 Xgboost 模型最终的目标函数（二次凸函数）。

Optimization 因为二次凸函数有:

$$\arg\min_x \ Gx + \frac{1}{2}Hx^2 = -\frac{G}{H}$$
$$\min_x \ Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H}$$

所以可得目标函数的最优解及最小值分别为：

$$w_j^* = -\frac{G_{t-1}^{(j)}}{H_{t-1}^{(j)} + \lambda} = -\frac{\sum\limits_{i \in I_j} g_{t-1}^{(i)}}{\sum\limits_{i \in I_j} h_{t-1}^{(i)} + \lambda}$$

$$obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{(G_{t-1}^{(j)})^2}{H_{t-1}^{(j)} + \lambda} + \gamma T$$

注意如上的推导是基于树的结构（树的深度，叶子结点个数）确定的情况

问题：如何找到最好的划分结构？不可能遍历无穷种树结构，因此采取贪心选择，即：

1. 从深度为 0 的**二叉树**开始（一个根节点）
2. 对每个叶子节点尝试添加一个分割（split），将原来的数据分为**左右两部分**然后计算这样添加后目标函数的变化

问题：分割点划分的依据是什么？

$$\begin{aligned}
\text{Gain} &= obj_{\text{before-split}} - obj_{\text{after-split}} \\
&= \left(-\frac{1}{2}\frac{(G_\text{L}+G_\text{R})^2}{H_\text{L}+H_\text{R}+\lambda}+\gamma\right) - \left(-\frac{1}{2}\frac{G_\text{L}^2}{H_\text{L}+\lambda}+\gamma\right) - \left(-\frac{1}{2}\frac{G_\text{R}^2}{H_\text{R}+\lambda}+\gamma\right) \\
&= -\frac{1}{2}\frac{(G_\text{L}+G_\text{R})^2}{H_\text{L}+H_\text{R}+\lambda}+\gamma+\frac{1}{2}\frac{G_\text{L}^2}{H_\text{L}+\lambda}-\gamma+\frac{1}{2}\frac{G_\text{R}^2}{H_\text{R}+\lambda}-\gamma \\
&= \frac{1}{2}\left(\frac{G_\text{L}^2}{H_\text{L}+\lambda}+\frac{G_\text{R}^2}{H_\text{R}+\lambda}-\frac{(G_\text{L}+G_\text{R})^2}{H_\text{L}+H_\text{R}+\lambda}\right)-\gamma
\end{aligned}$$

实际上在找一个最优分割点时，我们全部要做的，就是计算每一边儿样本的 g_i 和 h_i 之和，然后计算一下 Gain。

问题：如何找到最优分割点？遍历所有特征

- numerical feature：排序所有特征值，然后以任意大小相邻的两个特征值的中点作为分割点，计算 Gain.
- categorical feature：(one-hot 后) 按照当前 one-hot 离散列是否为 1 分成两部分，计算 Gain.

精确找寻切分点算法如下：

Algorithm 17: Basic Exact Greedy Algorithm for Split Finding

Input: Instances set of current node: $\mathcal{I}^{m\times d}$.

Output: Split with max score.

$score \leftarrow -\inf$

$G \leftarrow \sum_{i\in\mathcal{I}} g^{(i)},\ H \leftarrow \sum_{i\in\mathcal{I}} h^{(i)}$

for *j = 1, ..., d* **do**

  $G_L \leftarrow 0,\ H_L \leftarrow 0$

  for *i in sorted($\mathcal{I}$, by $x_j^{(i)}$)* **do**

    $G_L \leftarrow G_L + g^{(i)},\ H_L \leftarrow H_L + h^{(i)}$

    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$

    $score \leftarrow \max(\, score\,,\ \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\,)$

  end

end

近似找寻切分点算法如下（先对特征进行**百分比分箱**）：

Algorithm 18: Approximate Algorithm for Split Finding

for *j = 1, ..., d* **do**

  Propose $S_j = \{s_{j1}, s_{j2}, ..., s_{jl}\}$ by percentiles on feature j

  Proposal can be done per tree (global), or per split (local).

end

for *j = 1, ..., d* **do**

  $G_{jv} \leftarrow \sum_{j\in\{j\,|\,s_{j,v}\geq x_j^{(i)}\geq s_{j,v-1}\}} g^{(i)}$

  $H_{jv} \leftarrow \sum_{j\in\{j\,|\,s_{j,v}\geq x_j^{(i)}\geq s_{j,v-1}\}} h^{(i)}$

end

Follow same step as in previous section to find max score only among proposed splits.

13.4.3 Model pruning

全都围绕**最优分割是否是 negative gain**

- Pre-stopping: Stop split if the best split have negative gain, but maybe a split can benefit future splits..
- Post-Prunning: Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain.

13.5 LightGBM

13.5.1 Introduction

论文出处：LightGBM: A Highly Efficient Gradient Boosting Decision Tree[?]。

GBDT 性能问题 Gradient Boosting Decision Tree (GBDT) 是一个很流行的机器学习算法理论框架，也有很多优秀的开源实现 (e.g. XGBoost，pGBDT)。然而虽然有很多工程上的优化方法已经被添加进了 GBDT 的工程实现中，但是当**数据量很大**，**数据维度很高**时，算法框架的有效性和可扩展性依然得不到满足。

原因分析 训练阶段，在每一棵 CART，每一个最优分割点的训练寻找过程中，需要计算**所有候选分割值（特征 + 特征值）**的损失，然后找到最小损失值对应的分割点作为最优分割点。因此时间的花费取决于：

- 训练样本的个数 N
- 训练样本的维度 d

太费时间了，还是一棵树就如此！

解决办法

- 减少训练数据的样本个数 N (number of instances)
- 减少训练数据的特征个数 d (number of features)

但是该办法不可行。因为**不知道如何进行采样**。因为这是 Gradient Boosting Machine，不是 Adaboost，这里面**没有样本权重** (sample weight)。因此引出了 LightGBM 模型，该模型实际上包含两个算法：

- **Gradient-based One-Side Sampling (GOSS)**: 减少训练样本个数 N
- **Exclusive Feature Bundling (EFB)**: 减少训练特征维度 d

13.5.2 Model training

Algorithm 19: Gradient-based One-Side Sampling Algorithm[**?**]

Input: Training dataset $\{(\boldsymbol{x^{(i)}}, y^{(i)})\}_{i=1}^{N}$, where $\boldsymbol{x^{(i)}} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$;
Number of base tree model (CART): T;
Sampling ratio of large gradient data: α;
Sampling ratio of small gradient data: β;
Loss function: $loss$;
Weak learner: $f(\boldsymbol{x})$.

Output: The final hypothesis $F(\boldsymbol{x}) = \sum_{t=1}^{T} f_t(\boldsymbol{x})$.

Models $\leftarrow \{\}$, fact $\leftarrow \frac{1-\alpha}{\beta}$
Compute the number of samples to keep (with large gradients): topN $= \alpha \times n$
Compute the number of samples to be random sampling (with small gradients): randN $= \beta \times n$
for $t = 1$ ***to*** T **do**
  $\boldsymbol{preds} \in \mathbb{R}^n \leftarrow \underbrace{\text{models.predict}(\mathcal{X}^{N\times d})}_{\text{先前的模型进行预测}}$
  $\underbrace{\text{Compute the gradients of current Models over all samples:}}_{\text{注意是计算所有原始样本的残差}}$ $\boldsymbol{g} \leftarrow loss(\boldsymbol{y}, \boldsymbol{preds})$
  $\underbrace{\text{Initialize relative weights of all samples:}}_{\text{注意是计算所有原始样本}}$ $\boldsymbol{w} = \underbrace{\{1, 1, ..., 1\}}_{\text{length: N}}$
  $\underbrace{\text{Sort all samples with indices based on calculated abs of gradients:}}_{\text{根据梯度的绝对值排序所有样本，排序记录样本索引（省空间）}}$
  $\boldsymbol{sorted} \leftarrow \text{GetSortedIndices}\big(abs(\boldsymbol{g})\big)$
  $\underbrace{\boldsymbol{topSet} \leftarrow \boldsymbol{sorted}[\mathbf{1:}\text{topN}]}_{\text{前 topN 个大梯度（残差）的样本全部保留}}$
  $\underbrace{\boldsymbol{randSet} \leftarrow \text{RandomPick}(\boldsymbol{sorted}[\text{topN}\mathbf{:}\boldsymbol{N}], \text{randN})}_{\text{从除去梯度最大的 topN 个样本的样本中随机抽取 randN 个样本，保留下来}}$
  $\boldsymbol{usedSet} \leftarrow \boldsymbol{topSet} + \boldsymbol{randSet}$
  Assign weight fact to the small gradient data: $\boldsymbol{w}[\boldsymbol{randSet}] \times = fact$
  Train a weak learner $f_t(\boldsymbol{x})$, using dataset $\{\mathcal{X}[\boldsymbol{usedSet}], \underbrace{-\boldsymbol{g}[\boldsymbol{usedSet}]}_{\text{Models 预测后得到的残差}}\}$ with sample weight distribution $\boldsymbol{w}[\boldsymbol{randSet}]$.
  Models.append($f_t(\boldsymbol{x})$)
end

13.5.3 Reference

- GPU-acceleration for Large-scale Tree Boosting[**?**]
- A Communication-Efficient Parallel Algorithm for Decision Tree[**?**]

13.6 CatBoost（未完成）

13.6.1 Introduction

论文出处：CatBoost: unbiased boosting with categorical features[?] 和CatBoost: gradient boosting with categorical features support[?]

13.6.2 Model formulation

Categorical feature

13.6.3 Reference

- Yandex CatBoost: a machine learning method based on gradient boosting over decision trees
- Datacruiser's Blog: CatBoost 算法疏理

13.7 NGBoost（未完成）

13.7.1 Introduction

根据论文NGBoost: Natural Gradient Boosting for Probabilistic Prediction[?]

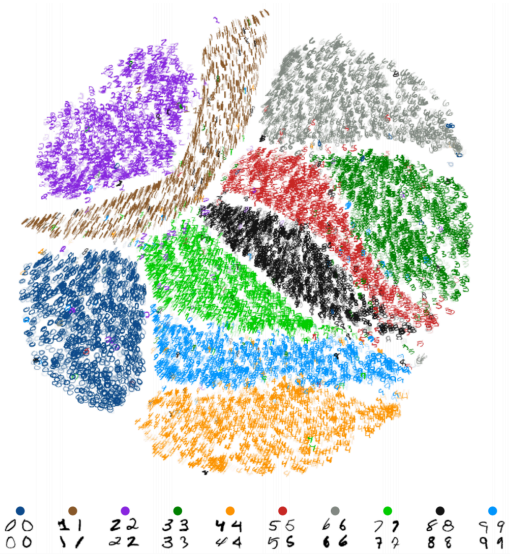
14 Dimensionality Reduction

14.1 t-Distributed Stochastic Neighbor Embedding (t-SNE)

14.1.1 Introduction

论文出处：Visualizing Data using t-SNE[?] 以及Accelerating t-SNE using Tree-Based Algorithms[?]。t-SNE 本质是一个需要通过优化得到最优参数的学习算法。一些应用如下所示：

- tSNE embedding of the MNIST dataset. Cite image from thesis: Dimensionality-Reduction Algorithms for Progressive Visual Analytics.



- Gridded t-SNE of images of animals from Caltech101.

14.1.2 Model formulation

Methodology t-SNE 算法的目的是：将样本集中的高维样本特征 $\mathcal{D}_{\mathcal{X}}=\{x^{(i)}\in\mathbb{R}^{\mathcal{X}}\}_{i=1}^{N}$ 表示为对应的第维样本特征 $\mathcal{D}_{\mathcal{Y}}=\{y^{(i)}\in\mathbb{R}^{\mathcal{Y}}\}_{i=1}^{N}$ ($\mathbb{R}^{\mathcal{Y}}\ll\mathbb{R}^{\mathcal{X}}$)，其算法过程包括三个部分：

- **高维空间距离转概率**。将高维空间内，样本集合 $\mathcal{D}_{\mathcal{X}}$ 中任意样本对 $x^{(i)},x^{(j)}\in\mathbb{R}^{\mathcal{X}}$ ($i,j\in\{1,...,N\},i\neq j$) 之间的**距离转换为条件概率** (i.g. affinities)：

$$p(x^{(j)}|x^{(i)})=\frac{\exp\left(-\frac{\|x^{(j)}-x^{(i)}\|^2}{2\sigma_i^2}\right)}{\sum_{k=1,k\neq i}^{N}\exp\left(-\frac{\|x^{(k)}-x^{(i)}\|^2}{2\sigma_i^2}\right)}$$

  其中 σ_i 为以样本点 $x^{(i)}$ 为中心的高斯分布 $\left(x^{(k)}-x^{(i)}|\,k\neq i\right)$ 的方差 (variance)。然后通过条件概率公式得到样本点 $x^{(i)}$ 和 $x^{(j)}$ 的**联合概率**$p(x^{(i)},x^{(j)})$:

$$\begin{aligned}p(x^{(i)},x^{(j)})&=p(x^{(i)})\cdot p(x^{(j)}|x^{(i)})\\&=p(x^{(j)})\cdot p(x^{(i)}|x^{(j)})\end{aligned}$$

  因此通过**加权**因子 $\alpha\in[0,1]$ 综合上述两种先验概率的概率计算方式，通过样本集内的点计算得到点 $x^{(i)}$ 和 $x^{(j)}$ 的联合概率 $p(x^{(i)},x^{(j)})$ 为：

$$\begin{aligned}p(x^{(i)},x^{(j)})&=\alpha\cdot p(x^{(i)})\cdot p(x^{(j)}|x^{(i)})+(1-\alpha)\cdot p(x^{(j)})\cdot p(x^{(i)}|x^{(j)})\quad(\alpha\in[0,1])\\&=\frac{\alpha}{N}\cdot p(x^{(j)}|x^{(i)})+\frac{1-\alpha}{N}\cdot p(x^{(i)}|x^{(j)})\quad(\because i.i.d,\therefore p^{(i)}=\frac{1}{N})\\&=\frac{\alpha}{N}\cdot\underbrace{\frac{\exp\left(-\frac{\|x^{(j)}-x^{(i)}\|^2}{2\sigma_i^2}\right)}{\sum_{k=1,k\neq i}^{N}\exp\left(-\frac{\|x^{(k)}-x^{(i)}\|^2}{2\sigma_i^2}\right)}}_{p(x^{(j)}|x^{(i)})}+\frac{1-\alpha}{N}\cdot\underbrace{\frac{\exp\left(-\frac{\|x^{(i)}-x^{(j)}\|^2}{2\sigma_j^2}\right)}{\sum_{k=1,k\neq j}^{N}\exp\left(-\frac{\|x^{(k)}-x^{(j)}\|^2}{2\sigma_j^2}\right)}}_{p(x^{(i)}|x^{(j)})}\end{aligned}$$

  若令 $\alpha=\frac{1}{2}$，则上述联合概率为：

$$p(x^{(i)},x^{(j)})=\frac{1}{2N}\cdot\left(\frac{\exp\left(-\frac{\|x^{(j)}-x^{(i)}\|^2}{2\sigma_i^2}\right)}{\sum_{k=1,k\neq i}^{N}\exp\left(-\frac{\|x^{(k)}-x^{(i)}\|^2}{2\sigma_i^2}\right)}+\frac{\exp\left(-\frac{\|x^{(i)}-x^{(j)}\|^2}{2\sigma_j^2}\right)}{\sum_{k=1,k\neq j}^{N}\exp\left(-\frac{\|x^{(k)}-x^{(j)}\|^2}{2\sigma_j^2}\right)}\right)$$

  Idea.可以考虑将 α 作为一个**attention weight**进行学习，做到：当 $p(x^{(j)}|x^{(i)})$ 相对于 $p(x^{(i)}|x^{(j)})$ 在 $p(x^{(i)},x^{(j)})$ 需要提供更多的信息时，$\frac{1}{2}\ll\alpha\leq 1$。
  因此最终得到原始高维空间 $\mathcal{X}$ 内样本之间的联合概率分布（如果将概率分布视为样本，则只是一条样本）：

$$\mathcal{P}=\left\{p(x^{(1)},x^{(2)}),\,p(x^{(1)},x^{(3)}),\,...,\,p(x^{(i)},x^{(j)}),\,...,\,p(x^{(N)},x^{(N-1)})\right\}\quad(i\neq j)$$

- **低维空间距离转概率**。其中低维空间 $\mathcal{Y}$ 中任意向量 $y^{(i)}\in\mathbb{R}^{\mathcal{Y}}$ 是待学习优化的参数，即：

$$\theta=\left(y^{(1)},...,y^{(i)},...,y^{(N)}\right)\in\mathbb{R}^{N\times|\mathcal{Y}|}$$

  其中 $y^{(i)}\in\mathbb{R}^{\mathcal{Y}}$ 为参数矩阵 $\theta\in\mathbb{R}^{N\times|\mathcal{Y}|}$ 中的第 i 行：

$$y^{(i)}=\theta_{((i),\cdot)}$$

而 $q(x^{(i)},x^{(j)})$ 的计算和 $p(x^{(i)},x^{(j)})$ 稍有不同，其直接根据距离计算联合概率分布而不是先由距离计算条件概率分布，再由条件概率分布计算联合概率分布。

$$q(x^{(i)},x^{(j)})=\frac{\frac{1}{1+\|y^{(j)}-y^{(i)}\|^2}}{\sum\limits_{i'=1}^{N}\sum\limits_{j'=1,j'\neq i'}^{N}\frac{1}{1+\|y^{(j')}-y^{(i')}\|^2}}$$

- **最小化联合概率分布差异**。最小化高维空间 $\mathcal{X}$ 内样本联合概率分布 $\mathcal{P}$ 和低维空间 $\mathcal{Y}$ 内样本联合概率分布 $\mathcal{Q}$ 之间的差异：

$$\begin{aligned}\min_{\theta}\ D_{\mathrm{KL}}(\mathcal{P}\|\mathcal{Q})&=\sum_{(i,j)\in\{1,\dots,N\},i\neq j}p(x^{(i)},x^{(j)})\log\frac{p(x^{(i)},x^{(j)})}{q(x^{(i)},x^{(j)};\theta)}\\ \Rightarrow\ \min_{\theta}\ D_{\mathrm{KL}}(\mathcal{P}\|\mathcal{Q})&=\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}p(x^{(i)},x^{(j)})\log\frac{p(x^{(i)},x^{(j)})}{q(x^{(i)},x^{(j)};\theta)}\end{aligned}$$

Objective

$$\min_{\theta}\ D_{\mathrm{KL}}(\mathcal{P}\|\mathcal{Q})=\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}p(x^{(i)},x^{(j)})\log\frac{p(x^{(i)},x^{(j)})}{q(x^{(i)},x^{(j)};\theta)}$$

其中：

- ground truth probability distribution 为：

$$\mathcal{P}=\Big(p(x^{(i)},x^{(j)})\Big)_{i,j\in\{1,\dots,N\},i\neq j}$$

- predicted probability distribution 为：

$$\mathcal{Q}=\Big(q(x^{(i)},x^{(j)};\theta)\Big)_{i,j\in\{1,\dots,N\},i\neq j}$$

Perplexity in t-SNE

- 算法中的困惑度（perplexity）$\mathrm{Perp}(\cdot)$ 是什么？

$$\begin{aligned}\mathrm{Perp}(\mathcal{P}(x^{(i)}))&=2^{\mathcal{H}(\mathcal{P}(x^{(i)}))}\\&=2^{-\sum_{j=1,j\neq i}^{N}p(x^{(j)}|x^{(i)})\cdot\log p(x^{(j)}|x^{(i)})}\end{aligned}$$

而后求解方程：

$$\mathrm{Perp}(\mathcal{P}(x^{(i)}))=Perp$$

即可得到 σ_i。

- 为什么引入困惑度（perplexity）？
- 如何选择？Automatic Selection of t-SNE Perplexity[?]
  一般取值设置在 5～50 之间。

Training procedure 训练算法的伪代码如下：

Algorithm 20: Simple version of t-Distributed Stochastic Neighbor Embedding[**?**]

Input: Dataset $\mathcal{D}_{\mathcal{X}} = \{x^{(i)} \in \mathbb{R}^{\mathcal{X}}\}_{i=1}^{N}$;
perplexity Attention weight of affinity α(Suggested default: $\frac{1}{2}$).

Output: The final parameters $\theta \in \mathbb{R}^{N \times |\mathcal{Y}|}$.

Compute original high-dimensional pairwise affinities $p(x^{(j)}|x^{(i)})$ with perplexity *Prep* using equation:

$$p(x^{(j)}|x^{(i)}) = \frac{\exp\left(-\frac{\|x^{(j)}-x^{(i)}\|^2}{2\sigma_i^2}\right)}{\sum_{k=1,k\neq i}^{N}\exp\left(-\frac{\|x^{(k)}-x^{(i)}\|^2}{2\sigma_i^2}\right)}$$

Set original high-dimensional joint probability distribution $p(x^{(i)}, x^{(j)})$:

$$p(x^{(i)}, x^{(j)}) = \frac{\alpha}{N} \cdot p(x^{(j)}|x^{(i)}) + \frac{1-\alpha}{N} \cdot p(x^{(i)}|x^{(j)})$$

Initialize embeddings of each sample in lower dimension space $\mathcal{Y}$ from $\mathcal{N}(0, 10^{-4}\mathcal{I})$:

$$\mathcal{D}_{\mathcal{Y}} = \left\{y^{(i)} = \theta_{((i),\cdot)} \in \mathbb{R}^{\mathcal{Y}}\right\}_{i=1}^{N}$$

while *stopping criterion not met* **do**

  Compute low-dimensional pairwise affinities $q(x^{(i)}, x^{(j)})$ using equation:

$$q(x^{(i)}, x^{(j)}) = \frac{\frac{1}{1+\|y^{(j)}-y^{(i)}\|^2}}{\sum_{i'=1}^{N}\sum_{j'=1,j'\neq i'}^{N}\frac{1}{1+\|y^{(j')}-y^{(i')}\|^2}}$$

  Update parameters θ by one step of gradient descent using:

$$\theta \leftarrow \theta - \eta\nabla_\theta \sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} p(x^{(i)}, x^{(j)}) \log\frac{p(x^{(i)}, x^{(j)})}{q(x^{(i)}, x^{(j)}; \theta)}$$

end

14.1.3 Reference

- Distill: How to Use t-SNE Effectively
- Dimensionality-Reduction Algorithms for Progressive Visual Analytics

Part III

Deep Learning Model and Theory

15 Dense Neural Networks (DNNs)

15.1 Fully Connection (FC)

15.1.1 Overview

Deep Learning[?] 的本质：

- 通过结构 cover 住数据特征之间的非线性
- 实现自动的特征抽取
- 达到端到端（end-to-end）的效果

15.1.2 Derivation of forward-propagation and back-propagation in fully-connected layer（未完成）

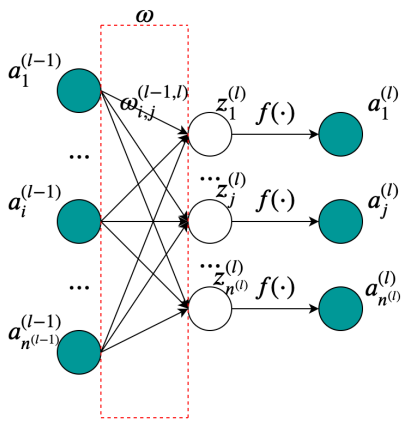


图 49: 全链接层前向推导过程中神经元的关系

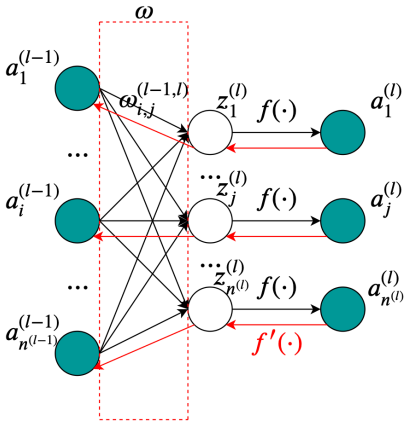


图 50: 全链接层反向优化过程中神经元的关系

forward-propagation 分别从标量运算（scalar operation）和张量运算（tensor operation）的角度分析正向推导过程。

scalar operation

$$z_j^{(l)} = \sum_{i=1}^{n^{(l-1)}} a_i^{(l-1)} \times w_{i,j}^{(l-1,l)}$$
$$a_j^{(l)} = f(z_j^{(l)})$$

其中：

- $a_i^{(l-1)}, w_{i,j}^{(l-1,l)}, z_j^{(l)}, a_j^{(l)} \in \mathbb{R}$
- $f(\cdot) : \mathbb{R} \to \mathbb{R}$

tensor operation

$$\boldsymbol{z}^{(l)} = \boldsymbol{a}^{(l-1)} \cdot \boldsymbol{w}^{(l-1,l)}$$
$$\boldsymbol{a}^{(l)} = f(\boldsymbol{z}^{(l)}) \quad \text{(element -wise operation)}$$

其中：

- $\boldsymbol{a}^{(l-1)} \in \mathbb{R}^{N \times D^{(l-1)}}$
- $\boldsymbol{w}^{(l-1,l)} \in \mathbb{R}^{D^{(l-1)} \times D^{(l)}}$
- $\boldsymbol{z}^{(l)}, \boldsymbol{a}^{(l)} \in \mathbb{R}^{N \times D^{(l)}}$

back-propagation 分别从标量运算（scalar operation）和张量运算（tensor operation）的角度分析反向传播过程。注意：**激活函数作用前后的导数为全导数，而没有偏导数**。

scalar operation

$$\frac{\partial \mathcal{L}}{\partial z_j^{(l)}} = \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} \cdot \frac{\mathrm{d}a_j^{(l)}}{\mathrm{d}z_j^{(l)}}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{i,j}^{(l-1,l)}} &= \frac{\partial \mathcal{L}}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l-1,l)}} \\ &= \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} \cdot \frac{\mathrm{d}a_j^{(l)}}{\mathrm{d}z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l-1,l)}}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial a_i^{(l-1)}} &= \sum_{j=1}^{n^{(l)}} \left(\frac{\partial \mathcal{L}}{\partial z_j^{(l)}} \cdot \frac{\mathrm{d}z_j^{(l)}}{\mathrm{d}a_i^{(l-1)}} \right) \\ &= \sum_{j=1}^{n^{(l)}} \left(\frac{\partial \mathcal{L}}{\partial a_j^{(l)}} \cdot \frac{\mathrm{d}a_j^{(l)}}{\mathrm{d}z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial a_i^{(l-1)}} \right)\end{aligned}$$

tensor operation

$$\nabla_{\boldsymbol{z}^{(l)}} \mathcal{L} = \nabla_{\boldsymbol{a}^{(l)}} \mathcal{L} \cdot \mathrm{diag}\left(\frac{\mathrm{d}\boldsymbol{a}^{(l)}}{\mathrm{d}\boldsymbol{z}^{(l)}} \right)$$

$$\nabla_{\boldsymbol{w}^{(l-1,l)}} \mathcal{L} = \nabla_{\boldsymbol{z}^{(l)}} \mathcal{L} \cdot \nabla_{\boldsymbol{w}^{(l-1,l)}} \boldsymbol{z}^{(l)}$$

其中将张量运算展开来看：

$$\nabla_{\boldsymbol{z}^{(l)}} \mathcal{L} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_1^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_j^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_{D^{(l)}}^{(1;l)}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial \mathcal{L}}{\partial z_1^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_j^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_{D^{(l)}}^{(n;l)}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial z_1^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_j^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial z_{D^{(l)}}^{(N;l)}} \end{bmatrix}_{\mathbb{R}^{N \times D^{(l)}}}$$

（上角标 $(n;l)$ 表示在神经网络第 l 层针对第 n 个样本）

$$= \underbrace{\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial a_1^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_{D^{(l)}}^{(1;l)}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial \mathcal{L}}{\partial a_1^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_{D^{(l)}}^{(n;l)}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial a_1^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_{D^{(l)}}^{(N;l)}} \end{bmatrix}}_{\nabla_{\boldsymbol{a}^{(l)}} \mathcal{L} \in \mathbb{R}^{N \times D^{(l)}}} \cdot \underbrace{\begin{bmatrix} \frac{\mathrm{d}a_1^{(1;l)}}{\mathrm{d}z_1^{(1;l)}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{\mathrm{d}a_j^{(n;l)}}{\mathrm{d}z_j^{(n;l)}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \frac{\mathrm{d}a_{D^{(l)}}^{(N;l)}}{\mathrm{d}z_{D^{(l)}}^{(N;l)}} \end{bmatrix}}_{\mathrm{diag}\left(\frac{\mathrm{d}\boldsymbol{a}^{(l)}}{\mathrm{d}\boldsymbol{z}^{(l)}}\right) \in \mathbb{R}^{D^{(l)} \times D^{(l)}}}$$

$$= \underbrace{\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial a_1^{(1;l)}} \frac{\mathrm{d}a_1^{(1;l)}}{\mathrm{d}z_1^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(1;l)}} \frac{\mathrm{d}a_j^{(1;l)}}{\mathrm{d}z_j^{(1;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_{D^{(l)}}^{(1;l)}} \frac{\mathrm{d}a_{D^{(l)}}^{(1;l)}}{\mathrm{d}z_{D^{(l)}}^{(1;l)}} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial a_1^{(n;l)}} \frac{\mathrm{d}a_1^{(n;l)}}{\mathrm{d}z_1^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(n;l)}} \frac{\mathrm{d}a_j^{(n;l)}}{\mathrm{d}z_j^{(n;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_{D^{(l)}}^{(n;l)}} \frac{\mathrm{d}a_{D^{(l)}}^{(n;l)}}{\mathrm{d}z_{D^{(l)}}^{(n;l)}} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial a_1^{(N;l)}} \frac{\mathrm{d}a_1^{(N;l)}}{\mathrm{d}z_1^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_j^{(N;l)}} \frac{\mathrm{d}a_j^{(N;l)}}{\mathrm{d}z_j^{(N;l)}} & \cdots & \frac{\partial \mathcal{L}}{\partial a_{D^{(l)}}^{(N;l)}} \frac{\mathrm{d}a_{D^{(l)}}^{(N;l)}}{\mathrm{d}z_{D^{(l)}}^{(N;l)}} \end{bmatrix}}_{\mathbb{R}^{N \times D^{(l)}}}$$

$$\nabla_{\boldsymbol{w}^{(l-1,l)}} \boldsymbol{z}^{(l)} = \left[\frac{\partial z^{(1;l)}}{\partial w_1^{(l-1,l)}} \right]$$

16 Convolutional Neural Networks (CNNs)

16.1 Convolution（未完成）

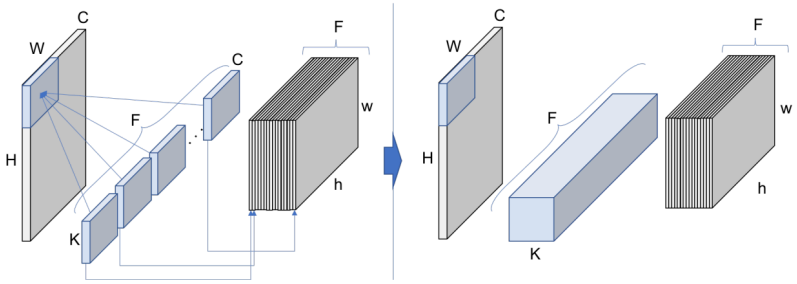


图 50: Different representation for convolutional kernels. Source: pabloruizruiz10: Convolution operation

16.1.1 Convolution as matrix multiplication

16.1.2 Derivation of forward-propagation and back-propagation in CNN（未完成，重要）

Convolution layer

- Input feature map: $\boldsymbol{X} \in \mathbb{R}^{W \times H \times C}$ or $\mathbb{R}^{C \times H \times W}$

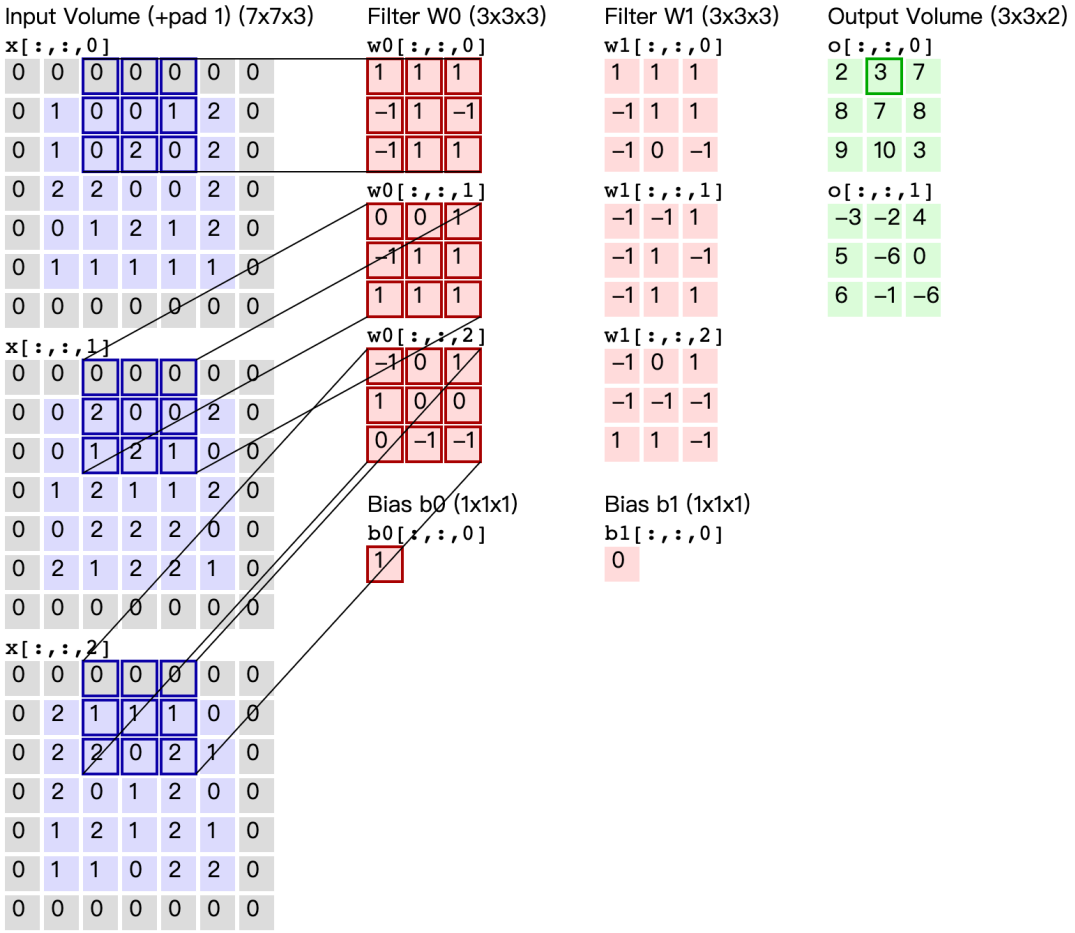


图 51: Convolution operation demo with $W_{\text{input}} = 5$, $H_{\text{input}} = 5$, $C_{\text{input}} = 3$, $K = 2$, $F = 3$, $S = 2$, $P = 1$.

16.1.3 Reference

- Derivation of Backpropagation in Convolutional Neural Network (CNN) [**?**]
- Deep learning for pedestrians: backpropagation in CNNs [**?**]
- Jake Bouvrie-MIT: Notes on Convolutional Neural Networks
- jefkine: Backpropagation In Convolutional Neural Networks
- CS231n: Convolutional Neural Networks for Visual Recognition
- denizyuret.github.io: Convolutional Neural Networks

16.1.4 Summary

- 卷机神经网络（CNN）最重要的作用就是抽取**局部特征**。
- 卷机神经网络架构的发展:
  1. LeNet5[?]: 1998
  2. AlexNet[?]: 2012
  3. VGG[?]: 2014
  4. GoogLeNet[?]: 2014 (2015.02 update to V3)
  5. ResNet[?]: 2015.12
  6. DenseNet[?]: 2016.08
  7. SqueezeNet[?]: 2016.11
  8. Xception[?]: 2016.07 (2017.04 update to V3)

16.2 Transposed Convolution（未完成）

16.2.1 Derivation of forward-propagation and back-propagation in Transposed Convolution（未完成，重要）

forward-propagation

back-propagation

16.2.2 Reference

- A guide to convolution arithmetic for deep learning[?]
- Distill: Deconvolution and Checkerboard Artifacts[?]