— “笔记”

April 30, 2017; rev. Wednesday 25th October, 2017

Qian Wang

第一章　预处理以及一些注意事项

Introduction

本章节内容主要介绍一些机器学习方法中常用的预处理方法以及一些注意事项

1.1　预处理

1.1.1　特征提取方法

常用的特征提取方法如下：

- 基于互信息（也叫做皮尔逊相关系数，信息增益）提取特征：$I(X,Y) = E(Y) - E(Y|X)$
- 基于决策树进行目标特征的选择（GBDT）
- L1，利用 L1 离散化特征，再使用 L2 进行交叉验证
- RF 和 LR 对模型特征进行打分
- 基于深度学习，进行自动选择特征
- 特征是否发散：方差接近于 0，不采用

1.1.2　特征标准化

常用的特征标准化方法如下：

- 对于逻辑回归或者神经网络来说，需要将数据归一化到 [-1,1]
- 对贝叶斯来说需要将数据归一化到 (0,1), 表达是或者不是
- 对于连续的特征来说，归一化可使用如下的公式：$x^* = \frac{x-\mu}{s}$, 其中如果归一化到（0,1）的话，s 为方差；如果归一化到 [-1,1]，$s = x_max - x_min$
- 对于非连续化的特征来说，采用 one-hot 编码实现

第二章　Logistic Regression

Introduction

本章内容主要来自于个人 YY

2.1　Sigmoid 函数

sigmoid 函数的主要公式如下所示：

$$p(x) = \frac{1}{1 + e^{-x}} \tag{1.1}$$

该公式主要用在二分类的问题中作为最后的预测结果分类，或者作为激活函数在神经网络中使用。

为什么要使用 sigmoid?

由于 sigmoid 的良好性质，求导容易，反向传播速度快，输出全部在 0-1 之间，永远为正，严格递增

缺点：在中心点变化快，比较敏感

2.2　Logistic Regression

逻辑回归主要用在二分类问题中，最后给出该样本属于正类或者负类的概率。决策函数就是 sigmoid 函数，就是说 sigmoid 的输出就是 LR 的输出，简化的公式如下：

$$h(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{2.2}$$

对于一个二分类问题来说，分类为 1 的概率就是 h(x), 而分类为 0 的概率为 1-h(x)，因此综合起来可以得到 LR 的决策函数为 (y=0 时前半部分不起作用，y=1 时后半部分不起作用，因此其实就是把分类为 0 和 1 的概率合起来写了)：

$$p(y|x;\theta) = h(x)^y[1 - h(x)]^{1-y} \tag{2.3}$$

对参数 w 做最大似然估计，我们可以得到似然函数：

$$\prod_{i=1}^{m} \boldsymbol{P}(\boldsymbol{y}_i|\boldsymbol{x}_i;\boldsymbol{\theta}) = \prod_{i=1}^{m} \boldsymbol{h}(\boldsymbol{x}_i)^{y_i}(\boldsymbol{1}-\boldsymbol{h}(\boldsymbol{x}_i))^{1-y_i} \tag{2.4}$$

对上式取对数得到

$$\boldsymbol{L}(\boldsymbol{\theta}) = \sum_{i=1}^{m} \left[\boldsymbol{y}^{(i)}\boldsymbol{log}\boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + (\boldsymbol{1}-\boldsymbol{y}^{(i)})\boldsymbol{log}(\boldsymbol{1}-\boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\right] \tag{2.5}$$

当我们使用最大似然估计的时候，目的就是最大化似然函数，但是当我们在机器学习的算法中，我们往往需要最小化一个函数来方便我们的计算，因此我们只需要对上式取负号即可得到 LR 的损失函数，乘以 $\frac{1}{m}$ 是为了方便计算。

$$\boldsymbol{J}(\boldsymbol{\theta}) = -\frac{\boldsymbol{1}}{\boldsymbol{m}} \sum_{i=1}^{m} \left[\boldsymbol{y}^{(i)}\boldsymbol{log}\boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + (\boldsymbol{1}-\boldsymbol{y}^{(i)})\boldsymbol{log}(\boldsymbol{1}-\boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\right] \tag{2.6}$$

通俗的来说，LR 就是用来将两堆数据分来。对于 LR 的输出，如果输出的值越大表明结果属于这一类的概率会越大，但是输出的结果并不是代表着概率。

LR 优点:

- 预测结果是介于 0 和 1 之间的概率
- 容易使用和解释
- 预测速度较快，计算量小

缺点：

- 当特征空间很大时，LR
- 容易欠拟合的性能不是很好
- 对于线性不可分的数据，需要定义非线性函数对特征进行映射
- 预测结果呈 S 型，中间的概率变化很大，很敏感

2.3 一些其他的问题

1. LR 是基于概率的一个模型，所有的点都会参与模型参数的更新。SVM 是基于最大化间隔的模型，由少量的支持向量决定。

2. SVM 对于异常点不敏感，而 LR 敏感。SVM 更加健壮，决策面不受非支持向量的影响。

第三章　正则化方法

Introduction

本章节内容主要介绍机器学习、深度学习总的正则化方法

一般来说，所有的监督学习都可以最小化下面的函数来表示：

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \sum_{i} \boldsymbol{L}(\boldsymbol{y}_i, \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{w})) + \boldsymbol{\lambda}\boldsymbol{\Omega}(\boldsymbol{w}) \tag{0.1}$$

其中，第一项一般为模型预测的结果与真实的结果之间的差距，可以用各种各样不同的函数来表示，第二项一般为正则化项，主要目的是使我们的模型更加简单，防止过拟合。

3.1　L1 L2

3.1.1　ill-condition

我们都知道优化问题有两大难题。一个是局部最小值的问题：我们要找的是全局最小值，如果局部最小值太多，那我们的优化算法就很容易陷入局部最小而不能自拔。另外一个就是 ill-condition 的问题。加入我们有个方程组 $\boldsymbol{Ax} = \boldsymbol{b}$，我们要做的是求解 $\boldsymbol{x}$, 如果 $\boldsymbol{A}$ 或者 $\boldsymbol{b}$ 稍微的改变，会使得 $\boldsymbol{x}$ 发生很大的变化，那么这个方程组系统就是 ill-condition 的，反之就是 well-condition 的。

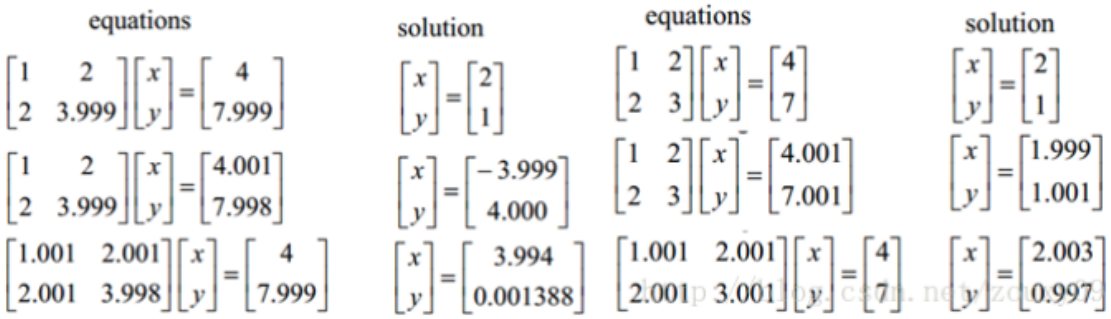


Figure 3.1: ill-condition

第一行我们假设 $\boldsymbol{Ax} = \boldsymbol{b}$，第二行我们稍微改变下 $\boldsymbol{A}$, 结果的变化就非常大，第三行我们

稍微改变下 $\boldsymbol{b}$，结果的改变同样是非常大的，因此我们可以认为我们的模型对错误的容忍力太低了，即对误差太敏感了。那么对于数据中难免存在的误差来说，模型的效果就非常差。

因此我们需要一个指标去衡量 ill-condition 问题中的变化问题。

condition number 衡量的就是在输入发生微小改变的时候，输出会发生多大的变化。也就是对系统微小变化的敏感度。condition number 比较小（在 1 附近）的就是 well-condition 的，比较大的（远大于 1 的）就是 ill-condition 的。

另外如果使用迭代优化的算法，当 condition number 太大的时候，会拖慢迭代的收敛速度。

3.1.2 L1

L1 norm 就是绝对值的和，公式如下

$\|\boldsymbol{x}\|_p = |\boldsymbol{x_1}| + |\boldsymbol{x_2}| + ... + |\boldsymbol{x_n}|$

L1 可以用来产生稀疏解，即参数具有 0 的最优值

3.1.3 L2

L2 norm 就是我们经常说的欧几里得范数，公式如下

$$\|\boldsymbol{x}\|_{\boldsymbol{2}} = (\sum_{\boldsymbol{i=1:n}} \boldsymbol{x_i^2})^{\frac{1}{2}} \tag{1.2}$$

使用了 L2 norm 后一个模型具有以下总的目标函数

$$\boldsymbol{\hat{J}(w; X, y) = \frac{\lambda}{2}\omega^T\omega + J(w; X, y)} \tag{1.3}$$

但是需要注意的是，由于 $\boldsymbol{\omega^T\omega}$ 与 b 无关，因此加入正则化项后，对于 b 的更新是没有任何影响的

caffe 中 weight decay 这个参数代表 L2 范数前的系数。

L2 的优点:

- 可以防止过拟合，提升模型的泛化能力
- L2 范数有助于处理 condition number 不好的情况下逆矩阵求逆很困难的情况（待定）。

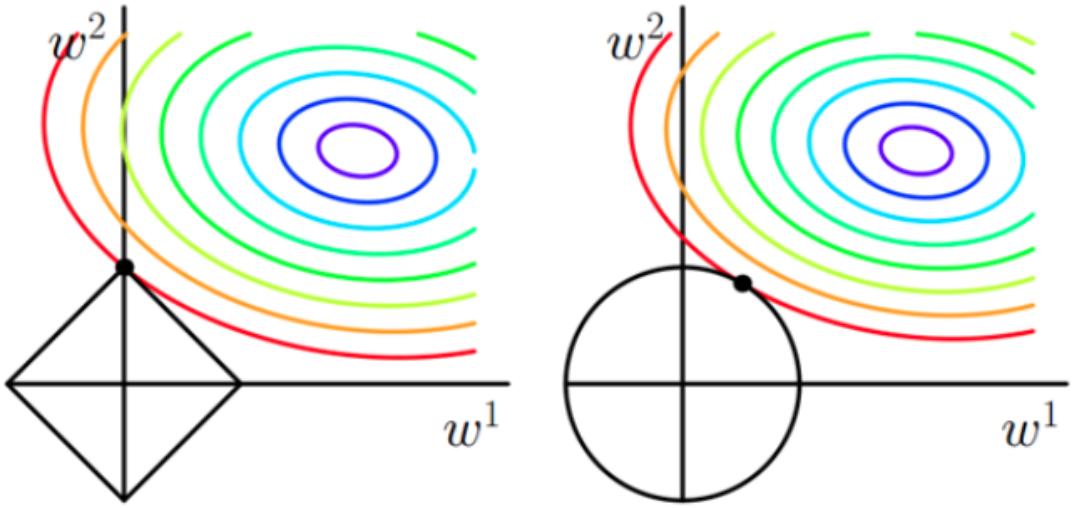
3.1.4 L1 与 L2 的不同

对于 L1 和 L2 规则化的代价函数来说，我们可以写成如下的形式

$$\boldsymbol{Lasso}\ \min_{\boldsymbol{w}} \boldsymbol{\frac{1}{n}\|y - Xw\|^2, s.t. \|w\|_1 <= C} \tag{1.4}$$

$$\boldsymbol{Ridge}\ \min_{\boldsymbol{w}} \frac{\mathbf{1}}{\boldsymbol{n}}\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}\|^{\mathbf{2}}, \boldsymbol{s.t.}\|\boldsymbol{w}\|_{\mathbf{2}} <= \boldsymbol{C} \tag{1.5}$$

为了便于可视化，我们考虑两维的情况，在 $(\boldsymbol{w_1}, \boldsymbol{w_2})$ 平面上画出目标函数的等高线，而约束条件则成为平面上半径为 C 的一个 norm ball。等高线与 norm ball 相交的地方就是最优解：



(a) ℓ_1-ball meets quadratic function. ℓ_1-ball has corners. It's very likely that the meet-point is at one of the corners.

(b) ℓ_2-ball meets quadratic function. ℓ_2-ball has no corner. It is very unlikely that the meet-point is on any of axes."

Figure 3.2: 图

可以看到，在相交的地方，L1 在和每个坐标轴相交的地方都有出现解，即更容易出现 $\boldsymbol{w_1}$ 或者 $\boldsymbol{w_2}$ 为 0 的解，可以用来提取特征，更加容易产生稀疏性。

L2 的话更容易出现 $\boldsymbol{w_1}$ 和 $\boldsymbol{w_2}$ 都不是 0 的情况，虽然有时候会出现很小的值。在更高维的情况下也是这样的，即基本上只是用来正则化而已。

3.2 数据集增强

需要注意的问题

- 不能应用改变正确类别的转换，如：光学字符识别任务需要认识到'b' 和'd' 的区别，因此对于这种任务来说，水平反转和旋转 180 度并不是适当的数据集增强方式。
- 在神经网络的输入层加入噪声，也可以看成是数据增强的一种形式。然而，神经网络被证明对噪声不是非常健壮。改善神经网络健壮性的方法之一是简单的将随机噪声施加到输入再进行训练。像隐藏单元事假噪声也是可行的，这被看成是在多个抽象层上进行数据集增强。实践证明，噪声被细心调整后，该方法是非常有效的。Dropout 可以被看做是通过乘性噪声构建新输入的过程。
- 在进行数据集增强的时候，需要进行对照试验。

3.3 提前终止

有时候会出现，虽然训练集的误差随着时间的推移逐渐降低，但是训练集的误差再次上升的情况。这种情况下我们需要提前终止模型的训练过程。

提前终止可以单独使用或与其它的正则化策略相结合。

提前终止需要验证集，即这一部分数据相当于在我们的训练过程中是没有参与模型的训练的。我们有两种策略来更好的使用所有的数据。第一种是，第一次训练我们使用训练数据确定最佳的训练步数，然后重新初始化网络，使用所有的数据进行训练。第二种是我们保持从第一轮训练获得的参数，然后使用所有的数据进行训练，但是这样的话，已经没有验证集可以指导我们需要在多少步后进行终止。

3.4 参数共享

参数共享也是一种正则化方法，是指强迫某些集合中的参数相等。

优点：可以显著减少模型所占用的内存

比如：卷积神经网络

3.5 bagging

Bagging：是通过结合几个模型降低泛化误差的技术。即分别训练几个不同的模型，让所有模型表决测试样例的输出。

Bagging 奏效的原因：不同的模型通常不会在测试集上产生完全相同的错误。

假设有 k 个不同的模型。在不同模型的误差完全相关的情况下，bagging 对于降低泛化误差没有任何帮助。但是在模型误差完全不相关的情况下，bagging 后的误差仅为之前误差的 $\frac{1}{k}$（期望）。这表明，模型的个数越多，则集成模型的误差降低的越小，呈线性关系，即集成后的模型最起码表现的比它的任何一个成员都要好。如果成员之间的误差是独立的，则集成将显著地比其成员表现得更好。

Bagging 需要构造 k 个不同的数据集，每个数据集与原始数据集具有相同数量的样例，但从原始样例中替换采样构成。即新的数据集中包含若干重复的实例，还会缺少很多实例。每个数据集包含样本的差异将导致训练模型之间的差异。

3.6 Dropout

Dropout 可以看成是集成非常多的大神经网络的 Bagging 方法。但是在 Dropout 中不同的模型之间是共享参数的，也就是说不同的模型之间具有相同的神经元的数量，但是在每一层中的神经元的个数可能是不同的，这一层中使用哪一个神经元也是不同的。

在使用 Dropout 时，每次训练的过程都只是一个子网络在训练，而不是训练整个网络。这也意味着，在一些小型的神经网路中使用 Dropout 并不是一个非常好的选择，而应该在一个大型的网络中使用 Dropout。也就是说，使用 Dropout 的代价就是提高了模型的计算代价。需要注意的是，如果你有非常大的训练数据集，使用 Dropout 进行正则化带来的好处可能小于所带来的计算代价的提升。

同样，在只有极少（如小于 5000 个）的训练样本可用时，Dropout 不会很有效。

Dropout 强大的大部分是由于施加到隐藏单元的乘性掩码噪声。BN 是在训练时向隐藏单元引入加性和乘性造成，同时这个噪声具有正则化的效果，有时候 Dropout 变得没有必要。

需要注意的是，在有些深度学习框架中，dropout ratio 表示的是保留的神经元的比例，有些是丢弃的神经元的比例。但是在大多数的神经网络的框架中都是表示保留的神经元的比例。

3.7 Spatial Dropout

如图所示，对于一个特征来说，传统的 Dropout 在随机丢弃点的时候，会丢弃这个特征中的一些点。而 Spatial Dropout 会直接丢弃整个特征。在实际的应用中，如在做物体检测，或者使用 Unet 的过程中 Spatial Dropout 的表现是更好的。

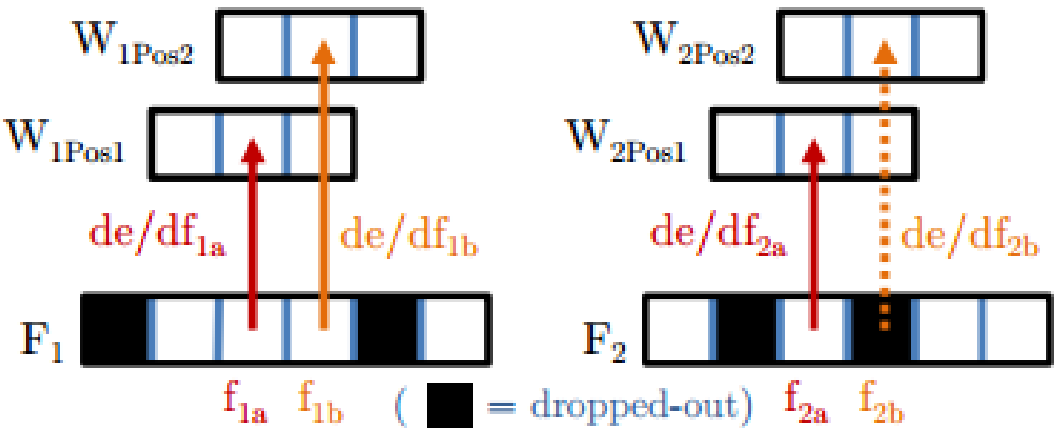


Figure 3.3: 原始 Dropout

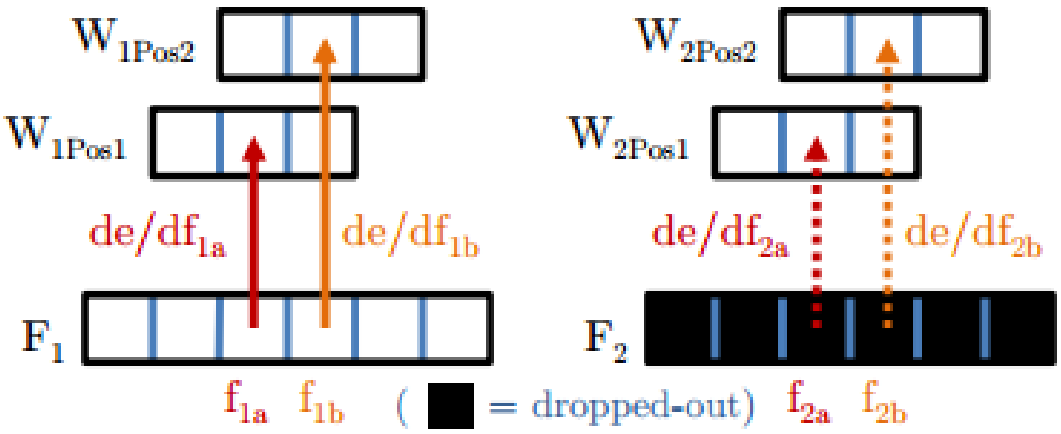


Figure 3.4: Spatial Dropout

3.8 对抗训练

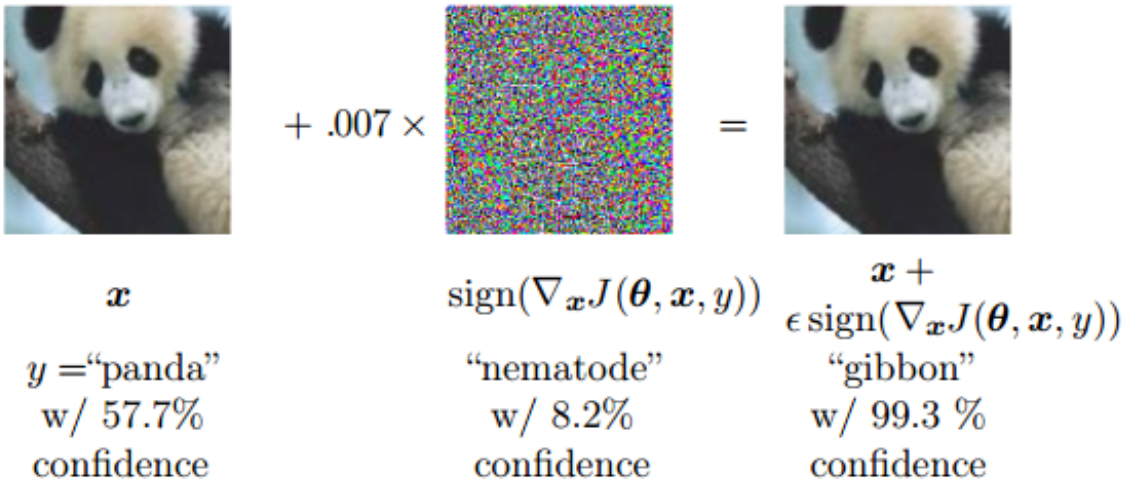


Figure 3.5: 图

通过添加一个不可察觉的小向量，我们可以改变 GoogleNet 对此图像的分类结果。如上图所示，我们人类可能不能观察出这种区别，但是网络的预测结果可能会千差万别。

对于上述这种情况我们可能需要使用对抗训练来进行正则化。

但是具体对抗训练是什么样的，暂时不清楚。---待补充

第四章　训练优化方法

Introduction

本章节内容主要介绍深度学习中遇到的优化方法

第五章　机器学习

Introduction

本章节主要介绍一些常用的机器学习方法，部分内容参考《统计学习方法》李航

5.1　决策树

5.1.1　熵

熵，表示一个信号源发出的信号的不确定程度。在信源发出的信号中，某信号出现的概率越大，熵越小。熵只依赖于分布，而与具体的取值无关。如果用随机变量来表示的话，熵越大，表示随机变量的不确定性越大。

香农使用 log 函数来定义样本 i 的信息熵：

$$\boldsymbol{f(p(i))} = \log \frac{\mathbf{1}}{\boldsymbol{p(i)}} = -\log \boldsymbol{p(i)} \tag{1.1}$$

因此在有 n 个样本的时候，我们用如下的公式来表示所有样本集合的信息熵：

$$\boldsymbol{E(P)} = \sum_{i}^{n} \log \boldsymbol{p(i)} \frac{\mathbf{1}}{\boldsymbol{p(i)}} = -\sum_{i}^{n} \boldsymbol{p(i)} \log \boldsymbol{p(i)} \tag{1.2}$$

然而，在机器学习中，我们通常使用模型分布 q(i) 来逼近真实分布 p(i)，因此交叉熵的公式为：

$$\boldsymbol{E(P\ Q)} = -\sum_{i}^{n} \boldsymbol{p(i)} \log \boldsymbol{q(i)} \tag{1.3}$$

在 tensorflow 或者其他的深度学习框架中，我们使用的也是如上的公式来表示交叉熵。y 表示网络的输出，y_ 表示真是的 label，公式如下：

$$\boldsymbol{cross\_entropy} = -\sum_{i}^{n} \boldsymbol{y\_} \log \boldsymbol{y} \tag{1.4}$$

5.1.2 信息增益

我们使用 g(D,A) 来表示特征 A 对数据集 D 的信息增益，E(D) 表示数据集 D 的熵，E(D|A) 表示在给定特征 A 的条件下 D 的经验熵。那么，我们可以使用如下的公式来表示信息增益：

$$\boldsymbol{g(D,A) = E(D) - E(D|A)} \tag{1.5}$$

E(D) 表示对数据集 D 进行分类的不确定性，E(D|A) 表示在给定特征 A 下对 D 分类的不确定性，那么它们的差就表示在使用了特征 A 后数据集信息不确定性的减少的程度。

信息增益的大小是相对于数据集而言的，并没有绝对意义，因此信息增益比可以解决这个问题。信息增益比：

$$\boldsymbol{g_R(D,A) = \frac{g(D,A)}{E(D)}} \tag{1.6}$$

在具体的算法中如果需要计算信息增益或者信息增益比，只需要分别计算 E(D),E(D|A) 即可。计算 $\boldsymbol{p_i}$ 时只需要计算 $\boldsymbol{C_i}$ 这一类在整个数据集中出现的次数即可，其他相关的参数也是类似计算。

5.1.3 决策树生成

ID3

对于训练数据 D，特征集 A，阈值 $\boldsymbol{\epsilon}$

- 如果 D 中的所有实例都属于同一类，则为一棵单节点树，算法结束。
- 如果特征集 A 为空，则 T 为单节点数，选择实例中出现次数最多的类作为该节点的类标记，算法结束
- 如果上述两条不满足，计算所有特征的信息增益，并选择信息增益最大的特征 $\boldsymbol{A_k}$，如果 $\boldsymbol{A_k}$ 小于阈值，则为单节点树，选择实例中出现次数最多的类作为该节点的类标记，算法结束。否则，对于 $\boldsymbol{A_k}$ 的每一个可能值 $\boldsymbol{a_i}$, 对数据集 D 进行划分，这个划分就是这棵树的多个子树。
- 重复上述步骤即可得到一颗决策树。

C4.5

C4.5 算法与 ID3 的不同之处是使用信息增益比来进行特征的选择。

5.1.4 防止过拟合

决策树的生成过程只考虑局部最优，这样就会造成全局不是最优的结果。因此，剪枝的过程中需要考虑全局最优。

决策树的剪枝通过最小化决策树的整体损失函数来实现。设决策树 T 有 |T| 个节点，t 为树 T 的叶节点，该叶节点有 N_t 个样本点, 其中属于 k 类的样本点有 N_{tk} 个，$E_t(T)$ 为叶节点的熵，α 为正则化参数，则决策树的损失函数定义为：

$$L = \sum_{t=1}^{|T|} N_t E_t(T) + \alpha |T| \tag{1.7}$$

决策树的损失函数表示的意思为，在保证整个数据集中样本点的熵的和尽量小的前提下，使得决策树中节点的数量更小。也就是说对于一些叶节点中样本的数量较少的节点，我们直接向上收缩，将这一个叶节点直接减掉，将其归到前一个分类条件中去。

5.1.5 CART 算法

回归树

分类树

5.2 随机森林

随机森林在属性（列）和数据（行）的使用上进行随机化，生成许多分类树，再汇总分类树的结果。随机森林在运算量没有显著提高的基础上提高了精度。

每棵决策树都只对部分数据进行决策，而随机采样可以保证有重复的样本被不同的决策树所分类，这样就可以对决策树的分类能力做出评价。

随机森林与 Bagging 有些类似。以决策树为基本模型的 bagging 在每次 bootstrap 放回抽样后，产生一次决策树，抽多少样本就产生多少棵树，在生成这些树的时候没有进行过多的干预。随机森林也是基于 bootstrap 进行随机抽样，但是每个节点的变量都是在随机产生的变量中产生。因此，不但样本是随机的，连每个节点变量的产生都是随机的。

随机森林的 loss function：熵

5.2.1 算法过程

- 首先是两个随机采样。对于每一棵决策树来说，都需要对行进行随机选择（有放回，可能会产生重复样本），然后进行列采样，即从所有的特征中抽取部分特征。这样每棵树所使用的数据都不是所有的数据。这两个随机性的加入使得随机森林不至于过拟合。（首先进行行采样，然后在每个节点分裂成子树的时候随机选择列，再计算信息增益等信息

进行特征选择）

- 然后对对于每一棵决策树来说，都采用完全分裂的方式建立。建立的过程同上一节的内容
- 由于步骤一的两个随机性的加入，使得随机森利即使不进行剪枝也不会出现过拟合的现象。
- 将生成的多棵决策树组合成随机森林，用随机森林分类器对新的数据进行判别与分类。分类结果按照多数表决决定。

为什么要对特征进行随机采样？如果有一个特征和标签特别强相关，那么在选择划分特征的时候，如果不随机的从所用的特征中选取一些特征的话，那么每一次这个强相关的特征都会被选取，那么每棵树都会是一样的。

5.2.2 算法优缺点

优点

- 两个随机性的引入，使得随机森林不容易过拟合，同时抗噪声能力更强
- 在处理高维数据的时候，不需要进行特征选择。顺便还可以确定重要的变量。
- 对数据的适应能力强，既能处理离散数据又能处理连续数据，数据集不需要进行规范化，不需要预处理，对缺失数据，非平衡数据稳健。
- 训练速度快，可以得到变量重要性的排序。可以进行特征提取
- 简单调参即可，跑一组试验画个图即可决定树的棵树。

缺点

- 模型规模大，评估速度慢
- 难以解释清楚

5.3 GBDT

GBDT(Gradient Boosting Decision Tree)，是一种迭代决策树算法。该算大由多棵决策树组成，所有树的结论累加起来作为最终答案。

5.3.1 回归树

随机森林中的树都是分类树，主要用于处理分类的情况。GBDT 中的树都是回归树，搞清楚这一点是非常重要的。

5.3.2 一些注意事项

- 自动发现有效的特征，特征组合，用来作为 LR 中的特征
- 每个节点得到一个预测值，最小化平方误差
- 以 CART 作为基分类器，基尼系数越大，不确定性越高 $\boldsymbol{Gini(D,A)=\frac{|D_1|}{|D|}Gini(D_1)+\frac{|D_2|}{|D|}Gini(D_2)}$

5.4 支持向量机

Introduction

SVM 的决策函数为 $\boldsymbol{sign(wx+b)}$

5.4.1 预备知识

函数间隔

在 SVM 中，分类的决策边界为一个超平面 $\boldsymbol{wx+b=0}$，一般来说，一个点距离这个超平面的远近可以表示分类的确信程度，在超平面确定的情况下，$|\boldsymbol{wx+b}|$ 可以表示点到超平面的距离。而 $\boldsymbol{wx+b}$ 的符号可以表示分类正确与否，即是否与 y 一致。因此，可以使用 $\boldsymbol{y(wx+b)}$ 用来表示分类的正确性与确定度，这就是函数间隔：

$$\hat{\gamma}_i = \boldsymbol{y_i(wx_i+b)} \tag{4.8}$$

但是函数间隔存在的一个致命的问题就是，当我们等比例的改变 w 和 b 的时候，函数间隔也在等比例的改变，但是此时我们用来分类的超平面却没有改变。这样当我们最大化间隔的时候将没有任何意义，因此我们需要对超平买你的法向量做一些约束，比如对 w 进行规范化，$\boldsymbol{\|w\|=1}$, 使得对于一个超平面来说，间隔是确定的，这样就成了几何间隔。

几何间隔

如上所述，我们可以得到几何间隔的定义。对于给定的数据集 T，给定的超平面 $\boldsymbol{wx+b=0}$，样本点 $\boldsymbol{(x_i,y_i)}$ 到超平面的几何间隔为：

$$\gamma_i = \boldsymbol{y_i}(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|})\boldsymbol{x_i} + \frac{\boldsymbol{b}}{\|\boldsymbol{w}\|} \tag{4.9}$$

由于我们在使用 SVM 的时候最有是要取最大的间隔，以上为每个点的几何间隔，因此我们需要得到所有点的几个间隔的最小值 ($\gamma = \min \gamma_i$)，我们只要使得这个最小值最大化，就可以获得我们的最终结果。γ 为整个数据集的几何间隔，γ_i 为第 i 个数据的几何间隔。

从函数间隔和几何间隔的定义我们可以得到它们之间的关系，如下式所示，当 $\|\boldsymbol{w}\| = \mathbf{1}$ 的时候，函数间隔和几何间隔是相等的，但是如果等比例的改变 w 和 b，函数间隔等比例的改变，但是几何间隔不变。

$$\gamma = \frac{\hat{\gamma}}{\|\boldsymbol{w}\|} \tag{4.10}$$

5.4.2 硬间隔最大化

我们的主要目的是获得一个分类平面，并且使得所有数据到这个平面的距离尽可能的远。即最大化几何间隔，使得数据集中的每个样本点到超平面的几何间隔最小是 γ

$$\max_{\boldsymbol{w},\boldsymbol{b}} \ \gamma \tag{4.11}$$

$$\boldsymbol{s.t.}\ \boldsymbol{y_i}(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\boldsymbol{x_i} + \frac{\boldsymbol{b}}{\|\boldsymbol{w}\|}) \geq \gamma \tag{4.12}$$

根据函数间隔与几何间隔的关系我们可以将上述公式转化为下面的公式：

$$\max_{\boldsymbol{w},\boldsymbol{b}} \ \frac{\hat{\gamma}}{\|\boldsymbol{w}\|} \tag{4.13}$$

$$\boldsymbol{s.t.}\ \boldsymbol{y_i}(\boldsymbol{w}\boldsymbol{x_i} + \boldsymbol{b}) \geq \hat{\gamma} \tag{4.14}$$

对于上面的公式来说，即使等比例的改变 w 和 b，也不会对目标函数的优化造成影响，因为等比例改变 w 和 b，虽然目标函数等比例的改变了，约束条件也会因此改变。也就是说，上式中函数间隔其实对结果没有任何作用。因此，我们可以使函数间隔为一个固定的值，在这里我们使 $\hat{\gamma} = \mathbf{1}$, 这样上述公式就简化了，同时我们将最大化问题改成最小化问题，就得到了下面的最优化问题。硬间隔最大化的 SVM 公式如下：

$$\min_{\boldsymbol{w},\boldsymbol{b}} \ \frac{\mathbf{1}}{\mathbf{2}}\|\boldsymbol{w}\|^{\mathbf{2}} \tag{4.15}$$

$$\boldsymbol{s.t.}\ \boldsymbol{y_i}(\boldsymbol{w}\boldsymbol{x_i} + \boldsymbol{b}) - \mathbf{1} \geq \mathbf{0} \tag{4.16}$$

求解上述公式可以得到 $\boldsymbol{w^*}, \boldsymbol{b^*}$,这样进一步我们就可以得到 SVM 分类的决策函数 $\boldsymbol{f(x)} = \boldsymbol{sign(w^*x + b^*)}$

5.4.3 硬间隔最大化的对偶问题

通过求解对偶问题可以得到原始问题的解，（最优化中的对偶问题，本章不做详细的描述），这样做有两个有点，一方面是对偶问题往往更容易求解，一方面是自然的引入了核函数。

首先我们需要构造拉格朗日函数，引入拉格朗日乘子 $\boldsymbol{\alpha_i \geq 0}$, 定义拉格朗日函数为：

$$\boldsymbol{L(w,b,\alpha) = \frac{1}{2}\|w\|^2 + \sum_{i=1}^{N} \alpha_i(1 - y_i(wx_i + b))} \tag{4.17}$$

现在问题等价于求解 $\min_{w,b}\ \max_{\alpha}\ \boldsymbol{L(w,b,\alpha)}$, 等价于求解 $\max_{\alpha}\ \min_{w,b}\ \boldsymbol{L(w,b,\alpha)}$, 那么我们首先需要求解 $\min_{w,b}\ \boldsymbol{L(w,b,\alpha)}$, 求解这个式子需要令其对于 w 和 b 的偏导数等与 0，即：

$$\boldsymbol{\frac{\partial_L}{\partial_w} = \|w\| - \sum \alpha_i x_i y_i = 0} \tag{4.18}$$

$$\boldsymbol{\frac{\partial_L}{\partial_b} = \sum \alpha_i y_i = 0} \tag{4.19}$$

将上面的导数带回原式得到：

$$\boldsymbol{L(w,b,\alpha) = -\frac{1}{2}\|w\|^2 + \sum_{i=0}^{N} \alpha_i} \tag{4.20}$$

接下来只需要求 $\max_{\alpha}\ \boldsymbol{L(w,b,\alpha)}$, 取负号即可直接转换成最小化问题，得到新的函数为：

$$\begin{aligned} &\min_{\alpha} \frac{1}{2}\|w\|^2 - \sum_{i=0}^{N} \alpha_i \\ &= \min_{\alpha} \frac{1}{2} \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j x_i x_j - \sum_{i=0}^{N} \alpha_i \end{aligned} \tag{4.21}$$

$$\boldsymbol{s.t.} \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{4.22}$$

$$\boldsymbol{\alpha_i \geq 0,\ i = 1, 2, ...N} \tag{4.23}$$

可以根据上述三个公式求得 $\boldsymbol{\alpha^*}$, 然后进一步求得 $\boldsymbol{w^*, b^*}$, 最终求得原始问题的解。下面还有一个问题就是，我们求得的解究竟是不是原始问题的解，如果是的话需要满足如下的 KKT 条件：

$$\boldsymbol{\frac{\partial_{L(w^*,b^*,\alpha^*)}}{\partial_w} = 0} \tag{4.24}$$

$$\frac{\partial_{L(w^*,b^*,\alpha^*)}}{\partial_b} = 0 \tag{4.25}$$

$$\alpha_i^*(y_i(w^*x_i + b^*) - 1) = 0, i = 1, 2, ..., N \tag{4.26}$$

$$y_i(w^*x_i + b^*) - 1 \geq 0, i = 1, 2, ..., N \tag{4.27}$$

$$\alpha_i^* \geq 0, i = 1, 2, ..., N \tag{4.28}$$

如果满足如上的 KKT 条件，我们就可以根据对偶问题来求原始问题的解了。有对偶问题，由公式 4.21-23 我们可以求得 α^*，进一步我们可以求得 w^*, b^*，这样我们就获得了整个原始问题中的参数。完成整个问题的求解

5.4.4 软间隔最大化

以上的问题仅仅是对于线性可分的情况下的训练数据，但是对于线性不可分的问题，以上的方法是不能很好的奏效的。软间隔最大化将以上方法扩展到了线性不可分的情况下了。并且这样做，可以避免 SVM 过拟合。

线性不可分意味着有时候某些样本点是不能满足函数间隔大于 1 的要求的，因此为了解决这个问题，对于每一个样本点 (x_i, y_i)，我们引入了一个松弛变量 $\xi_i \geq 0$，使得函数间隔加上松弛变量大于等于 1，，即我们的约束条件变成了如下的公式：

$$y_y(wx_i + b) \geq 1 - \xi_i \tag{4.29}$$

这样原始的线性不可分的线性支持向量机的问题，变成了如下的凸二次规划的问题（原始问题）：

$$\max_{w,b\,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \tag{4.30}$$

$$s.t.\ y_y(wx_i + b) \geq 1 - \xi_i, i = 1, ...N \tag{4.31}$$

$$\xi_i \geq 0, i = 1, ..., N \tag{4.32}$$

下面我们给出以上问题的对偶问题，首先需要构造拉格朗日函数：

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i + \sum_{i=1}^{N}\alpha_i(1 - y_i(wx_i + b) - \xi_i) - \sum_{i=1}^{N}\mu_i\xi_i \tag{4.33}$$

对偶问题是拉格朗日函数的极大极小问题，由上面我们知道，首先要求函数的极小，因此我们要对 $\boldsymbol{w}, \boldsymbol{b}, \boldsymbol{\xi}_i$, 来分别求偏导：

$$\frac{\boldsymbol{\partial}_{\boldsymbol{L}}}{\boldsymbol{\partial}_{\boldsymbol{w}}} = \|\boldsymbol{w}\| - \sum \boldsymbol{\alpha}_i \boldsymbol{x}_i \boldsymbol{y}_i = \mathbf{0} \tag{4.34}$$

$$\frac{\boldsymbol{\partial}_{\boldsymbol{L}}}{\boldsymbol{\partial}_{\boldsymbol{b}}} = \sum \boldsymbol{\alpha}_i \boldsymbol{y}_i = \mathbf{0} \tag{4.35}$$

$$\frac{\boldsymbol{\partial}_{\boldsymbol{L}}}{\boldsymbol{\partial}_{\boldsymbol{\xi}_i}} = \boldsymbol{C} - \boldsymbol{\alpha}_i - \boldsymbol{\mu}_i = \mathbf{0} \tag{4.36}$$

将上述三个公式带入原问题，我们可以得到原始问题的对偶问题：

$$\min_{\boldsymbol{\alpha}} \frac{\mathbf{1}}{\mathbf{2}} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}} \sum_{\boldsymbol{j}=\mathbf{1}}^{\boldsymbol{N}} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j \boldsymbol{y}_i \boldsymbol{y}_j \boldsymbol{x}_i \boldsymbol{x}_j - \sum_{\boldsymbol{i}=\mathbf{0}}^{\boldsymbol{N}} \boldsymbol{\alpha}_i \tag{4.37}$$

$$\boldsymbol{s.t.} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}} \boldsymbol{\alpha}_i \boldsymbol{y}_i = \mathbf{0} \tag{4.38}$$

$$\mathbf{0} \leq \boldsymbol{\alpha}_i \leq \boldsymbol{C},\ \boldsymbol{i} = \mathbf{1}, \mathbf{2}, \boldsymbol{...N} \tag{4.39}$$

当对偶问题满足如下的 KKT 条件时，我们就可以只求出如上对偶问题的解，来作为原始问题的解：

$$\frac{\boldsymbol{\partial}_{\boldsymbol{L}(\boldsymbol{w}^*, \boldsymbol{b}^*, \boldsymbol{\xi}^*, \boldsymbol{\alpha}^*, \boldsymbol{\mu}^*)}}{\boldsymbol{\partial}_{\boldsymbol{w}}} = \mathbf{0} \tag{4.40}$$

$$\frac{\boldsymbol{\partial}_{\boldsymbol{L}(\boldsymbol{w}^*, \boldsymbol{b}^*, \boldsymbol{\xi}^*, \boldsymbol{\alpha}^*, \boldsymbol{\mu}^*)}}{\boldsymbol{\partial}_{\boldsymbol{b}}} = \mathbf{0} \tag{4.41}$$

$$\frac{\boldsymbol{\partial}_{\boldsymbol{L}(\boldsymbol{w}^*, \boldsymbol{b}^*, \boldsymbol{\xi}^*, \boldsymbol{\alpha}^*, \boldsymbol{\mu}^*)}}{\boldsymbol{\partial}_{\boldsymbol{\xi}}} = \mathbf{0} \tag{4.42}$$

$$\boldsymbol{\alpha}_i^* (\boldsymbol{y}_i (\boldsymbol{w}^* \boldsymbol{x}_i + \boldsymbol{b}^*) - \mathbf{1} + \boldsymbol{\xi}_i^*) = \mathbf{0}, \boldsymbol{i} = \mathbf{1}, \mathbf{2}, \boldsymbol{..., N} \tag{4.43}$$

$$\boldsymbol{\mu}_i^* \boldsymbol{\xi}_i^* = \mathbf{0} \tag{4.44}$$

$$\boldsymbol{y}_i (\boldsymbol{w}^* \boldsymbol{x}_i + \boldsymbol{b}^*) - \mathbf{1} + \boldsymbol{\xi}_i^* \geq \mathbf{0}, \boldsymbol{i} = \mathbf{1}, \mathbf{2}, \boldsymbol{..., N} \tag{4.45}$$

$$\boldsymbol{\alpha}_i^* \geq \mathbf{0}, \boldsymbol{i} = \mathbf{1}, \mathbf{2}, \boldsymbol{..., N} \tag{4.46}$$

$$\xi_i^* \geq 0, i = 1, 2, ..., N \tag{4.47}$$

$$\mu_i^* \geq 0, i = 1, 2, ..., N \tag{4.48}$$

当对偶问题满足如上条件后，我们可以根据对偶问题求得一个最优解 $\alpha^* = (\alpha_i^*, ...\alpha_N^*)$，然后可以进一步计算出 w^*, b^*，这样我们就可以获得软间隔最大化的线性支持向量机的决策函数了。

5.4.5 非线性支持向量机

以上所有的内容，均为线性支持向量机，即使对于线性不可分的数据来说。

非线性分类问题是指通过利用非线性模型才能很好的进行分类的问题，非线性支持向量机主要解决非线性的分类问题。

核函数

核函数的定义为:$K(x, z) = \phi(x) \cdot \phi(z)$

核函数的想法是，在学习和预测时只使用核函数 $K(x, z)$，而不显式的定义映射函数 ϕ。通常来说，直接计算核函数 $K(x, z)$ 会更容易，但是通过 $\phi(x) \cdot \phi(z)$ 来计算核函数缺并不容易。在软间隔最大化的时候我们可以使用核函数来避免使用，如:

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=0}^{N} \alpha_i \tag{4.49}$$

正定核

这一部分的内容，有需要再进行补充

5.4.6 一些注意事项

- SVM 如何防止过拟合？松弛变量的加入，即软间隔最大化
- 为什么要转化为对偶问题？因为原问题是凸二次规划问题，转化为对偶问题更加高效。

5.5 AdaBoost

本章内容来自于网络以及张志华老师的就《机器学习导论》课程

本章基本内容就是给定一堆弱分类器，然后通过各种组合组成一个人强的分类器

5.5.1 离散的 AdaBoost

离散的 AdaBoost 算法步骤：

$\boldsymbol{w}$ 表示给数据的权值，$\boldsymbol{\alpha}$ 表示给分类器的权值

1. start with weights $\boldsymbol{w_i} = \frac{1}{N} \quad \boldsymbol{i = 1...N}$
2. repeat for m=1 to M
   - 使用输入数据训练一个分类器 $\boldsymbol{G_m(x)} \in \boldsymbol{(-1, 1)}$
   - 计算误差 err:
   $$\boldsymbol{E_w[I_(y_i \neq G_m(x_i))] = \frac{\sum_{i=1}^{N} w_i I_(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}}$$
   - 输出 $\boldsymbol{\alpha_m = \frac{1}{2} log(\frac{1-err}{err})}$, 从这里可以看出分类器的误差越大，权值越小
   - set $\boldsymbol{w_i = \frac{w_i}{Z_m}} \exp(\boldsymbol{\alpha_m I_(y \neq G_m(x))})$，其中 $\boldsymbol{Z_m}$ 是规范化因子，$\boldsymbol{Z_m = \sum_{i=1}^{N} w_i} \exp(\boldsymbol{-\alpha_m y_i G_m(x_i)})$ 如果一个数据点分类错了，那么给这个点的权值大一点。
3. 这样我们就得到了 $\boldsymbol{G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]}$

5.5.2 Forward Stagest Additive Modeling

考虑加法模型 (additive model)

$$\boldsymbol{f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)} \tag{5.50}$$

其中,$\boldsymbol{b(x; \gamma_m)}$ 为基函数，$\boldsymbol{\gamma_m}$ 为基函数的参数，$\boldsymbol{\beta_m}$ 为基函数的系数。

在给定训练数据以及 Loss Function 的情况下，相当于一个加法模型 $\boldsymbol{f(x)}$ 相当于一个 minimize Loss Function 的问题：

$$\boldsymbol{min_{\beta_m, \gamma_m} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m b(x; \gamma_m))} \tag{5.51}$$

从前向后，每一步值学习一个基函数及其系数，即每一步只需优化如下的损失函数：

$$\boldsymbol{min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta b(x; \gamma))} \tag{5.52}$$

前向分布算法：

输入：训练数据 $\boldsymbol{T = (x_1, y_1), ...(x_N, y_N)}$,Loss Function$\boldsymbol{L(x, f(x))}$, 基函数 $\boldsymbol{b(x; \gamma)}$

输出：加法模型 $\boldsymbol{f(x)}$

1. 初始化 $\boldsymbol{f_0(x) = 0}$
2. repeat for m=1 to M
   - 计算参数 $\boldsymbol{\beta_m, \gamma_m}$

$$(\boldsymbol{\beta_m, \gamma_m}) = \arg\min_{\boldsymbol{\beta,\gamma}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{L(y_i, f_{m-1}(x_i)\beta b(x;\gamma))} \tag{5.53}$$

   - 更新 $\boldsymbol{f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)}$
3. 得到加法模型

$$\boldsymbol{f(x) = \sum_{m=1}^{M} \beta_m b(x;\gamma_m)} \tag{5.54}$$

第六章　自然语言处理

Introduction

本章节主要介绍一些文本方面相关的知识。

6.1　如何表示词

Introduction

本节主要介绍如何使用词向量的形式去表达一个模型，本章节大部分内容来自知乎专栏以及 cs224d 课程。

准备重写这一部分的内容，整个符号体系沿用 cs224n 课程的符号体系，说明如下：

- V 表示中心词矩阵，其中的每一列 $\boldsymbol{v}_i$ 表示第 i 个中心词的词向量。
- U 表示输出词矩阵，其中的每一列 $\boldsymbol{u}_i$ 表示第 i 个上下文词的词向量，为 V 的转置。
- 我们使用 $\boldsymbol{w}_t$，来表示第 t 个词，这里不却分中心词和上下问的词。
- 用 T 来表示词表，|T| 来表示此表中词的个数

6.1.1　one-hot

首先，可以将每个词表示成 one-hot vector, 就是只有在对应位置的向量的值为 1，其余位置为 0，如下图所示：

$$w^{aardvark} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{a} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{at} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots w^{zebra} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Figure 6.1: one-hot

但是这种表示方法有几种缺点，一种是如果词表太长的话，我们可能没有那么多的内存去存储这么多的词向量，另外一种是，每个 word 之间都是独立的，我们没有办法从中找到他们之间的关系。

$$(w^{hotel})^T(w^{motel}) = (w^{hotel})^T(w^{cat}) = 0 \tag{1.1}$$

虽然两个向量之间的点乘可以表示 word 之间的关系，但是由于在 one-hot 中，任意两个向量都是正交的，因此他们之间的点乘结果都是 0，所以不能通过这种方式来表达向量之间的相似程度。因此我们需要通过某种方式，将这个高维空间压缩到一个低维的空间中去。

比如，我们可以通过一个词周围的词来表达这个词的信息。比如如下的例子。可以用 banking 周围的词来代表 banking 这个词，这样在大规模的预料中，这样的统计就变得比较重要了。

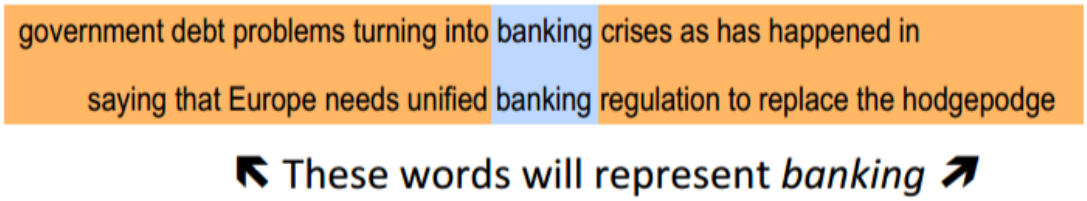


Figure 6.2: 词的表示形式

这样，我们就可以将之前的 one-hot 形式的稀疏矩阵变成一个稠密的矩阵了，但是这里的一个词的词向量是从计算机的角度来理解的，没有什么感知上的意义。这样，我们的问题就比较明确了，即找到一个 model 来刻画一个词和它的上下文之间的概率（即将 one-hot 变成稠密的矩阵），概率越大说明学习的越好。如果取负值，那么这就是我们的 loss function 了。

6.1.2 skip-gram

skip-gram 是 word2vec 的其中一种方法，主要的目的是找到一个词和它的上下文之间的关联。word2vec 主要有两种模型，两种训练的方法。

skip-gram 是一种根据当前的 word 预测 contex 的算法。

我们使用如下所示的图来表示 skip-gram，加入中心词 banking 的位置为 t，那么我们分别根据第 t 个词来预测第 t-m,...t-1,t+1,...,t+m 个词的概率。其中除了第 t 个词的其他的词叫做 output context words。每次计算的时候，取一个词作为中心词

这样我们就可以获得 skip-gram 的计算公式了，即分别计算已知第 t 个词，预测前后 m 个词的概率，然后最大化概率的成绩即可，对于第二个连乘来说计算的是这一个中心词的所有上下文词的概率，第一个连乘表示的是每次取一个词作为中心词，遍历所有的词。加入负号之后我们即可以直接作为 loss function 来使用了。公式如下：

$$L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m} p(w_{t+j}|w_t;\theta) \tag{1.2}$$

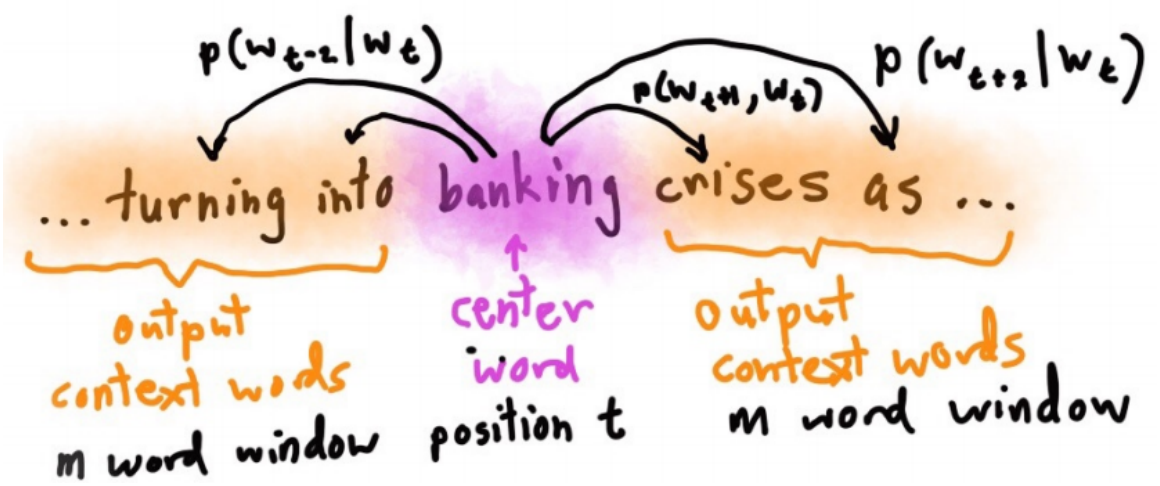


Figure 6.3: skip-gram

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m}\log p(w_{t+j}|w_t) \tag{1.3}$$

通常我们是使用交叉熵来作为损失函数，但是这里的向量使用的是 one-hot 的形式，计算结果时只有 trueclass 的位置有值，这样显得不太妥当。

这样我们已经获得了损失函数，下面我们需要做的就仅仅是求出这样一个概率就可以了。概率可以使用如下的公式来求解。同样首先我们需要定义一些表达式：c 表示中心词的下标；v_c 表示第 c 个中心词的词向量，u_o 表示第 o 个 outside 词的词向量。这样我们就可以表示已知第 c 个词求第 o 个词的概率：（这里我们用的是点积，即每个向量对应位置的元素相乘。点乘的意义是两个向量越是接近，点乘的结果越大）

这个公式其实就是算出中心词和每一个上下文词的相似度，相似度就是两个词的词向量的点积，$u_w v_c$，其中 u_o 就是第 w 个上下文词，v_c 代表当前的中心词，这里的 $p(o|c)$ 表达的是第 o 个上下文词与第 v 个中心词中间的关系，但是我们最后需要求出所有的上下文词与中心词之间的关系，因此根据下文中的推倒过程我们可以观察到，我们需要对 o 进行遍历 (从 1 到 |T|)，来获得我们想要的结果。

$$p(o|c) = \frac{exp(u_0^T v_c)}{\sum_{w=1}^{v} exp(u_w^T v_c)} \tag{1.4}$$

需要注意的是，我们通常初始化的时候 U 和 V（即输出词矩阵和中心词矩阵）都是随机初始化的，我们是在梯度下降的时候对这两个矩阵进行更新，最后网络的输出结果就是这两个矩阵，我们最终也仅仅是需要知道这两个矩阵即可，其他的像是损失函数以及网络的输出结果都是我们不关心的，之所以需要有这两个东西，就是为了获得输出词矩阵和中心矩阵。下面的几页 ppt 讲述了梯度下降的过程中是如何求解 v_c 的，即第 c 个中心词的词向量。输出词矩阵也是按照如下的方式方向传播，只不过符号不同而已。

下面，我们介绍如何通过反向传播算法求解 skip-gram。

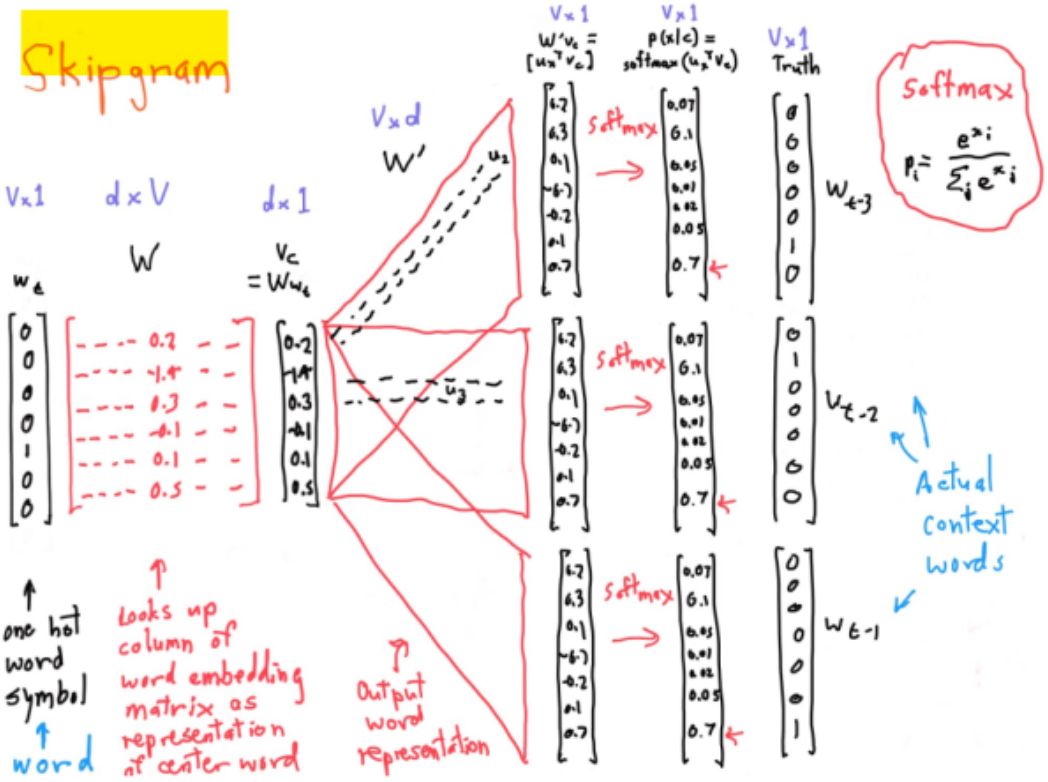


Figure 6.4: skip-gram

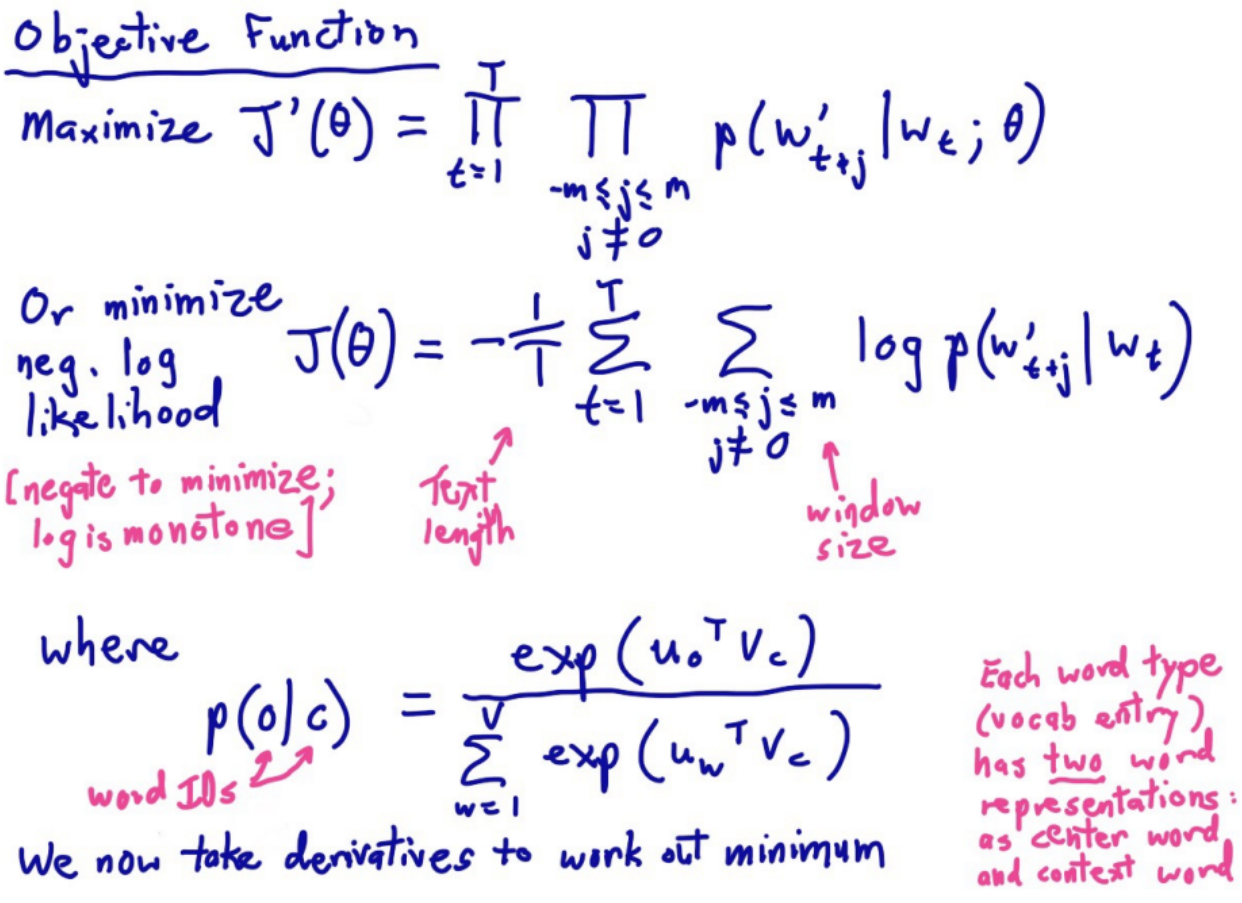


Figure 6.5: 目标

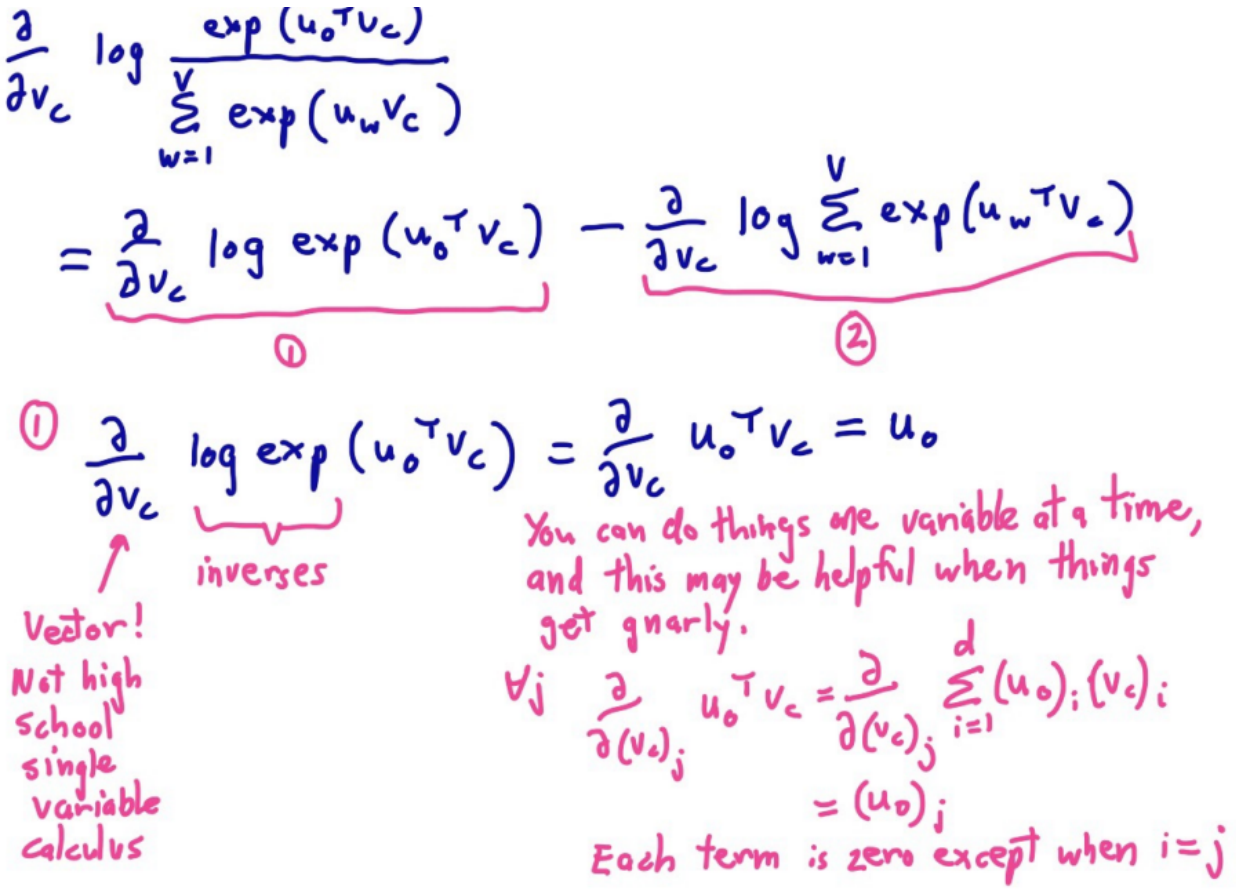


Figure 6.6: 反向传播

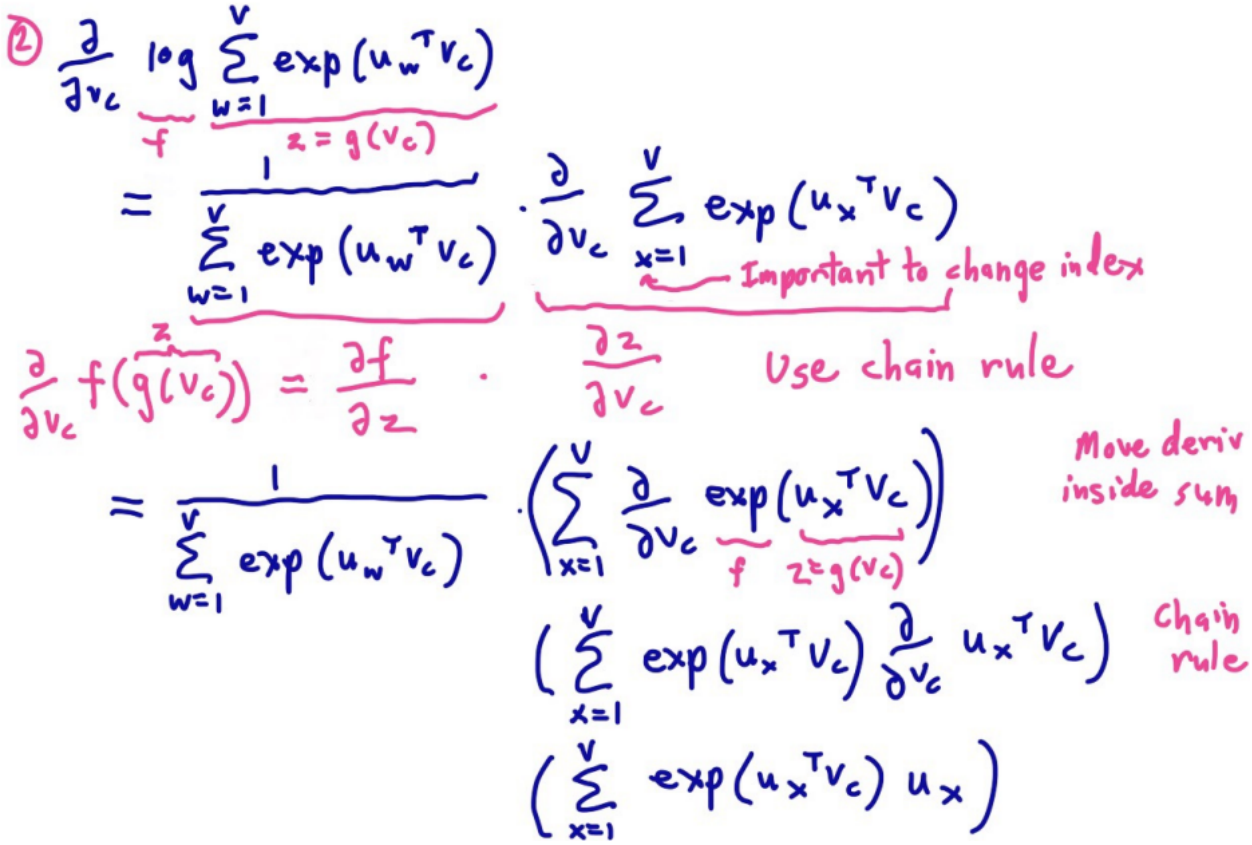


Figure 6.7: 反向传播

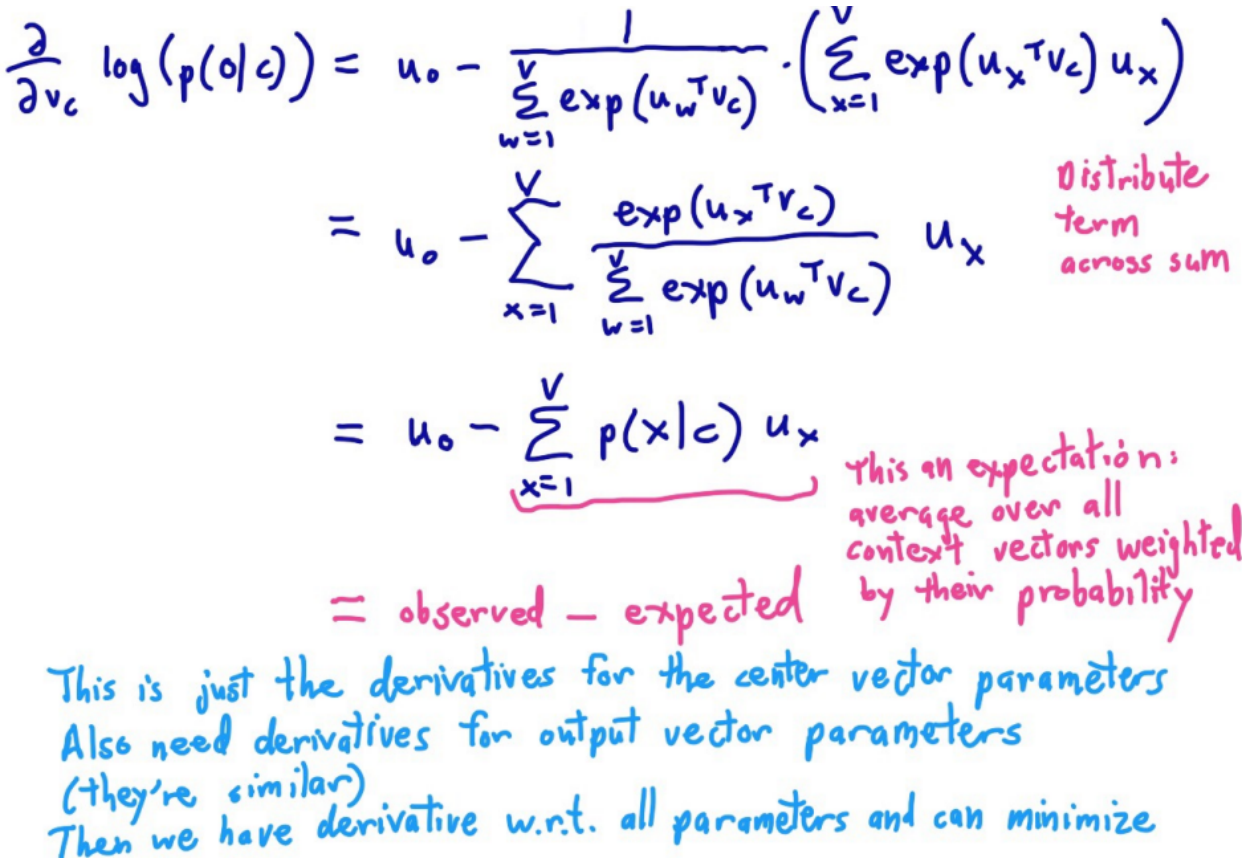


Figure 6.8: 反向传播

其中 m 表示我们的窗口的大小需要注意的是：在每一个窗口，我们更新这个窗口所遇到的所有的参数。如假设我们的窗口为 1，要处理的文本为 we like leaing a lot , 第一个窗口我们需要计算 internal vector $\boldsymbol{v}_{like}$ 和 external vector $\boldsymbol{u}_{we}, \boldsymbol{u}_{learing}$，第二个窗口的计算也是这样的。

6.1.3 Negatibe Sample

假如我们有这样一句话：I love deep learing and NLP

在使用 skip-gram 的时候，假设第一个中心词为 love，窗口大小 m=1，那么我们首先需要从 center word matrix V 中取出一列 $\boldsymbol{v}_{love}$，然后从 output word matrix U 中取出一列 $\boldsymbol{u}_I$ 来计算概率 $\boldsymbol{p}(\boldsymbol{I}|\boldsymbol{love}) = \frac{exp(u_I^T v_{love})}{\sum_{w=1}^{V} u_w^T v_{love}}$，这个公式中的 V 代表词表中的所有的词，然后从 center word matrix V 中取出一列 $\boldsymbol{v}_{love}$，然后从 output word matrix U 中取出一列 $\boldsymbol{u}_{deep}$ 来计算概率 $\boldsymbol{p}(\boldsymbol{deep}|\boldsymbol{love}) = \frac{exp(u_I^T v_{love})}{\sum_{w=1}^{V} u_w^T v_{love}}$, 这样当 love 作为中心词的时候我们需要计算的概率就已经计算完了。我们可以通过这种方式继续计算当 deep 为中心词，以及后面的词为中心词的所有情况所需要的概率。

那么这样做有什么缺点呢？在计算概率的时候，分母中我们需要计算每一个词表中的词与当前中心词的相似度，当词表变得很大的时候，这样的计算量明显是很吃内存的，并且大量的词与中心词是没有任何关系的，因此我们需要简化这样的操作。

在 cs224n 中只介绍了这么多，所以暂时只写这么多.

在上一节的内容中我们推导出了 word2vec 的损失函数是：

$$\boldsymbol{J}(\boldsymbol{\theta}) = -\frac{\mathbf{1}}{\boldsymbol{T}}\sum_{\boldsymbol{t}=\mathbf{1}}^{\boldsymbol{T}}\boldsymbol{J}_{\boldsymbol{t}}(\boldsymbol{\theta}) \tag{1.5}$$

在这里我们用 $\boldsymbol{\theta}$ 表示损失函数中所有的参数，这里是 U 和 V，其中 $\boldsymbol{J}_{\boldsymbol{t}}(\boldsymbol{\theta})$ 为：

$$\boldsymbol{J}_{\boldsymbol{t}}(\boldsymbol{\theta}) = \log \boldsymbol{\sigma}(\boldsymbol{u}_{\boldsymbol{o}}^{\boldsymbol{T}}\boldsymbol{v}_{\boldsymbol{c}}) + \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{k}}\boldsymbol{E}_{\boldsymbol{j}=\boldsymbol{P}(\boldsymbol{w})}[\log \boldsymbol{\sigma}(\boldsymbol{u}_{\boldsymbol{j}}^{\boldsymbol{T}}\boldsymbol{v}_{\boldsymbol{c}})] \tag{1.6}$$

k 表示我们使用的要进行负采样的个数。这里我们是使用的 $\boldsymbol{J}_{\boldsymbol{t}}(\boldsymbol{\theta})$ 来代替了之前所使用的 log 概率。之所以这样做是因为我们要最大化真正的 outside word 词在 $\boldsymbol{v}_{\boldsymbol{c}}$ 附近出现的概率，而最小化我们随机选择的词在 $\boldsymbol{v}_{\boldsymbol{c}}$ 附近出现的概率。

还有一个需要注意的是,在那篇原始论文中也出现过一次,就是这里的 $\boldsymbol{P}(\boldsymbol{w}) = \boldsymbol{U}(\boldsymbol{w})^{(}\mathbf{3}/\mathbf{4})/\boldsymbol{Z}$, 暂时不清楚是什么意思。$\boldsymbol{p}(\boldsymbol{w})$ 表示的是 noise distribution，这里的 $\boldsymbol{U}(\boldsymbol{w})$ 表示的是 unigram distribution。待补充

6.1.4 CBOW

CBOW 根据 surrounding context 来预测 center word，对于每一个词，需要学习两个 vector，v(input vector):when the word is in the context;u(output):when the word is in the center.

以上其实就基本是是 word2vec 的所有内容了。word2vec 最终可以获得很多相邻的词，如果我们可以将结果可视化出来的话，对于一个词来说，我们可以获得经常与这个词一起出现的词，比如对于 google 通常会和 yahoo 一起出现一样。

word2vec 的本质就是预测每一个词的周围的词，那么这也有很大的缺点，比如我们每次只计算一个词周围的词，造成了时间上的浪费。

那么有没有其他的解决方案呢？比如一次预测多个词，当然有!!!

6.1.5 基于 SVD 的方法

window based co-occurrence matrix

对于如下三句话来说，我们的窗口大小是 1（通常来说是 5-10），是对称的（即词出现在中心词的左边和右边都是没有关系的），但是在这一类算法中，并不是总是对称的

I like deep learing

I like NLP

I enjoy flying

对于这种算法，我们只要统计出如下的表格就可以了

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Figure 6.9: 统计表格

但是这样做也有着明显的缺点，比如我们的存储会随着此表的大小而逐渐的变大，模型不够鲁棒等。因此我们很需要对数据进行降维。

low dimensional vector

在这里我们将所有的词表中出现的词存储到一个固定的低维的向量中去，通常为 25-1000 维，降维方法使用 PCA。

改进

- 有一些经常出现的词，比如 the，he 等？这种情况我们可以让最大的 count=100，再大就不再增加了，或者直接忽略掉这些词
- 使用皮尔逊系数来代替 counts，然后把负样本的值标记为 0
- 根据与中央词的距离衰减词频权重

使用 SVD 表示词向量的方法虽然比较简单，但是效果也还不错。这样的方法以及效果，在这篇论文中有详细的介绍 An Improved Model of SemanRc Similarity Based on Lexical Co-Occurrence Rohde et al. 2005

但是使用 SVD 表示词向量有着各种各样的问题，比如计算复杂度比较高，不方便处理新词或者新的文档，与其他的 DL 模型的训练套路不同。

因此提出了一种结合了 word2vec 和 SVD 两种方法的新方法提出来了，GloVe 方法。

6.1.6 GloVe

Glove 的目标函数是：

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2 \tag{1.7}$$

其中，P_{ij} 表示第 i 和第 j 个词共同出现的频率，f 是一个 max 函数，图像如下图所示

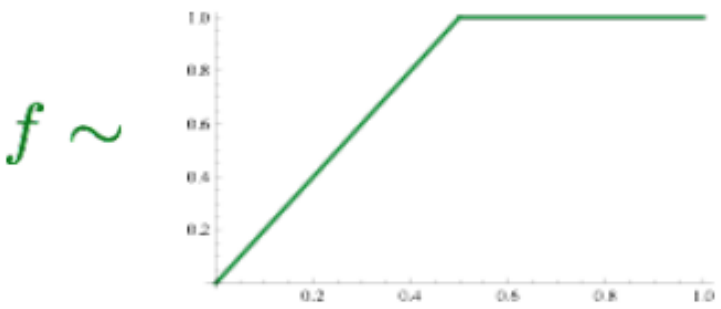


Figure 6.10: f 函数

P 是在一开始就计算好的一个矩阵，$f(P_{ij})$ 是控制该对词的权重，贡献次数越大，权重也越大，但是当大于一个阈值后，权重为 1，不再变化。后面的 square loss ，目的是希望频次过高时，两者的相似度也大，这样使得整体 loss 变小。

Glove 的优点是训练快，可以扩展到大规模语料，也适用于小的语料和向量

另外在损失函数中，我们发现 u 和 v 都捕捉到了共现信息，那么我们可以将他们直接加起来作为我们的信息 $X_{final} = U + V$, 相对于 word2vec 来说，只关注一个窗口内相近的词出现的频率，但是 Glove 更关注全局上两个词出现的频率。

6.1.7 评价方法

Intrinsic：专门设计单独的试验，由人工标注词语或句子相似度，与模型结果对比。好处是是计算速度快，但不知道对实际应用有无帮助。有人花了几年时间提高了在某个数据集上的分数，当将其词向量用于真实任务时并没有多少提高效果，想想真悲哀。

通过 a 对于 b 来讲，相当于 c 对于哪个词？主要通过如下的公式来获得

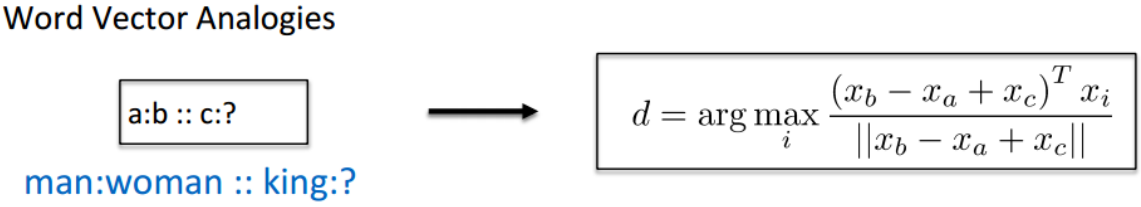


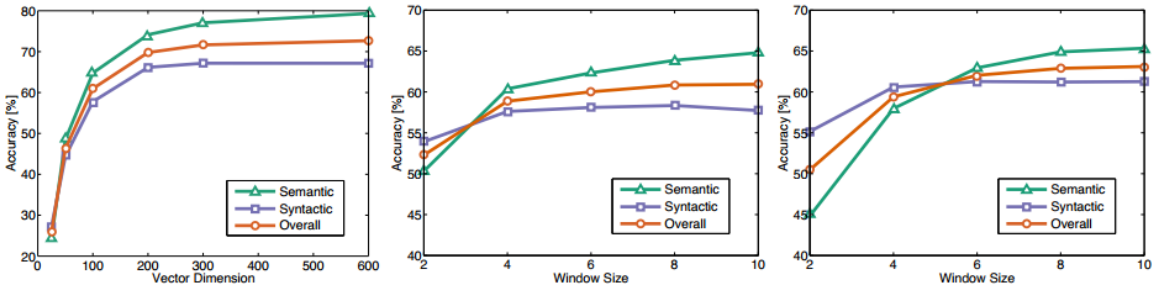
Figure 6.11: intrinsic word vector evalution

比如 man:woman::king:? 这种结果可视化出来后，往往都是平行的向量

Extrinsic：通过对外部实际应用的效果提升来体现。耗时较长，不能排除是否是新的词向量与旧系统的某种契合度产生。需要至少两个 subsystems 同时证明。这类评测中，往往会用 pre-train 的向量在外部任务的语料上 retrain。

调参

如果考虑到成本的话，大约 300 维，窗口为 8 的对称窗口的效果是最好的



(a) Symmetric context (b) Symmetric context (c) Asymmetric context

Figure 6.12: 调参

6.2 文本分类

Introduction

本章内容主要介绍一些基本概念

6.2.1 Softmax

第七章　Deep Learing

Introduction

本章节内容主要是来自于深度学习中遇到的一些坑以及问题

7.1　一些小的 trick

1. 一定要对数据进行归一化

2. 训练可能会出现 loss 一开始迅速下降，然后稳定在一定的值上，这时候不要以为网络已经收敛了，有时候会出现一个拐点，即在训练很后面的时候会重新出现一个新的下降的区间，这时候网络基本上会收敛。但是不排除会出现多的拐点的情况。结论：训练过程中一定要耐心耐心再耐心。

7.2　weight decay

在损失函数中，weight decay 是放在正则项（regularization）前面的一个系数，正则项一般指示模型的复杂度，所以 weight decay 的作用是调节模型复杂度对损失函数的影响，若 weight decay 很大，则复杂的模型损失函数的值也就大。我所理解的 weight decay 就是一个调节正则化项的系数，增大则对正则化项的依赖会更高，不容易过拟合，反之则反之。

利用 weight decay 给损失函数加了个惩罚项，使得在常规损失函数值相同的情况下，学习算法更倾向于选择更简单（即权值和更小）的 NN。是一种减小训练过拟合的方法。

Weight decay is equivalent to L2 regularizer.

在训练神经网络的时候，可以先设置 weight decay 为 0，然后使网络过拟合，查看测试结果。结果正确后，设置 weight decay 为大一点的值，即加入正则化项，这样可以防止过拟合现象。具体大小需要在网络中调试。

遇到的问题：在训练 U-net 的时候，出现训练结果全是黑图的情况。原因是 weight decay 设置过大，导致网络结果没有过拟合。解决方案：设置 weight decay 为更小的值。

7.3 momentum

momentum 是梯度下降法中一种常用的加速技术。对于一般的 SGD，其表达式为 $\boldsymbol{x} \leftarrow \boldsymbol{x} - \boldsymbol{\alpha} * \boldsymbol{dx}$,x 沿负梯度下降。而带 momentum 项的 SGD 则写生如下形式：$\boldsymbol{v} = \boldsymbol{\beta} * \boldsymbol{v} - \boldsymbol{a} * \boldsymbol{dx}, \boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{v}$, 其中 $\boldsymbol{\beta}$ 是 momentum 系数，通俗的理解上面式子就是，如果上一次的 momentum（即 v）与这一次的负梯度方向是相同的，那这次下降的幅度就会加大，所以这样做能够达到加速收敛的过程。

神经网络的训练过程（也就是梯度下降法）是在高维曲面上寻找全局最优解的过程（也就是寻找波谷），每经过一次训练 epoch，搜寻点应该更加靠近最优点所在的区域范围，这时进行权重衰减便有利于将搜寻范围限制在该范围内，而不至于跳出这个搜索圈，反复进行权重衰减便逐渐缩小搜索范围，最终找到全局最优解对应的点，网络收敛。momentum 是冲量单元，也就是下式中的 m，作用是有助于训练过程中逃离局部最小值，使网络能够更快速地收敛，也是需要经过反复地 trial and error 获得的经验值

主要作用：防止陷入局部最小值

第八章　Caffe

Introduction

本章节内容主要是关于 caffe 框架的一些知识。

8.1　lmdb

lmdb 格式的文件在使用 Python 程序进行生成的时候，如果需要重复生成，则需要先删除原来的。否则会在原先的 lmdb 上重新添加文件。

8.2　net protobuf

此部分内容有待添加

8.3　solver

solver 文件为 caffe 的训练参数的文件，主要存储一些训练的超参数

运行代码为： caffe train –solver=*slover.prototxt

一个例子：

```
1     train_net: "lenet_train.prototxt"
2     test_net: "lenet_test.prototxt"
3         test_iter: 100
4         test_interval: 500
5
6         base_lr: 0.01
7         lr_policy: "fixed"
8
9         momentum: 0.9
```

```
10          type: SGD
11          weight_decay: 0.0005
12          display: 100
13          max_iter: 20000
14          snapshot: 5000
15          snapshot_prefix: "models"
16          solver_mode: CPU
```

下面来一个一个解释这些程序的意思

1.

train_ net: "lenet_ train.prototxt"

test_ net: "lenet_ test.prototxt"

这两行用于定于训练网络和测试网络，可以是同一个网络，用 net：train_ test.prototxt 来表示, 为上一节的内容。注意：文件的路径要从 caffe 的根目录开始，其他所有的配置都是这样的。

2. 接下来 test_ iter 表示一次训练需要加载多少个数据，许训练中的 batch size 是一致的；test_ iterval 表示经过多少此训练的 Iteration 后进行一次测试，如 500 表示没经过 500 个 Iteration 进行一次测试。另外，如果网络不想进行测试的话，可以在 solver 文件中加入如下的参数 test_ initialization: false 这样不管经过多少次的 Iteration 都不会进行测试。

3. 有关于 learing rate 的东西

base_ lr 表示初始的一个 learing rate，如果 lr_ policy 如果设置为 fixed，训练过程中会一直维持这个 learing rate 不再改变，其他的都是会在训练过程中逐渐变化的。

lr_ policy 可设置的值如下所示：

- fixed: 保持 base_ lr 不变
- step: 如果设置为 step, 则还需要设置一个 stepsize, 返回 $base\_lr * gamma^{floor * \frac{iter}{stepsize}}$, 其中 iter 表示当前的迭代次数。如设置 gamma=0.9，stepsize=100 表示在第一百次迭代后 learning rate 下降到原来的 0.9 倍
- exp: 返回 $base\_lr * gamma^{i}ter$，iter 为当前迭代次数
- inv: 如果设置为 inv, 还需要设置一个 power, 返回 $base_{l}r * (1 + gamma * iter)^{-power}$
- multistep: 如果设置为 multistep, 则还需要设置一个 stepvalue。这个参数和 step 很相似，step 是均匀等间隔变化，而 multistep 则是根据 stepvalue 值变化
- poly: 学习率进行多项式误差, 返回 $base\_lr(1 - iter/max\_iter)^{power}$
- sigmoid: 学习率进行 sigmod 衰减，返回 $base\_lr(1/(1 + exp(-gamma * (iter - stepsize))))$

需要设置参数的数量随着 lr pilicy 的不同而有所变化。如设置为 fixed 则不需要添加任何参数，设置为 step 则需要添加 gamma 和 stepsize 两个参数，设置为 step 的策略后 solver 配置如下所示：

```
1          base_lr: 0.01
2          lr_policy: "step"
3          gamma: 0.9
4          stepsize: 100
```

4. 对于 momentum，一般取值在 0.5–0.99 之间。通常设为 0.9，momentum 可以让使用 SGD 的深度学习方法更加稳定以及快速。详细的资料，参考 Hinton 的论文《A Practical Guide to Training Restricted Boltzmann Machines》

5. type:SGD

表示优化算法，总共有六种：SGD、AdaDelta、AdaGrad、Adam、NAG、RMSprop

6.weight_ decay 为权重衰减项，详细的内容已经在上一章解释过了。

7.display：100 表示没训练 100 个 Iteration 显示一次 loss

8.max_ iter:2000 表示最大的迭代此时为 2000

9.

snapshot:500

snapshot_ prefix :"models"

表示没训练 500 个 Iteration，保存一次网络的参数数据，保存路径为 models。同时会保存另外一个 solverstate 文件，以便下次训练的时候可以从这一步继续训练。

10.solver_ mode: CPU 设置运行模式为 CPU

8.4 一些其他的问题

8.4.1 loss=87.3365

- 如果一开始就出现这种情况，有可能是参数的初始化方式不对，使用 xavier 可以避免这种情况
- 在 caffe 中数据的标签必须从 0 开始，且为整数（1.0,2.0 也可以）
- 数据可能会出现问题（一定要反复确定数据没有问题才可以）
- 如果是图片数据可能是图片没有进行归一化，最好是进行归一化并且减去均值
- 如果一开始的 loss 是在下降的，但是训练到后期出现了 87.3365 这个值，代表网络发散了，需要调整 learing rate 到一个较小的值，建议从 0.01 开始，到 0.0001 即可适合

大部分网络结构

8.4.2 一些差别

solver.step(1) 表示网络进行一次迭代，即进行一次前向过程和一次反向传播

solver.net.forward() 表示网络只进行一次前向的过程，可用于网络的预测

8.4.3 多 GPU 计算

如果使用 solver 文件：build/tools/caffe train –solver=models/bvlc_ alexnet/solver.prototxt –gpu=0,1

如果使用 Python 文件： python yourpythonfile.py CUDA_ VISIBLE_ DEVICES=0, 1

Bibliography