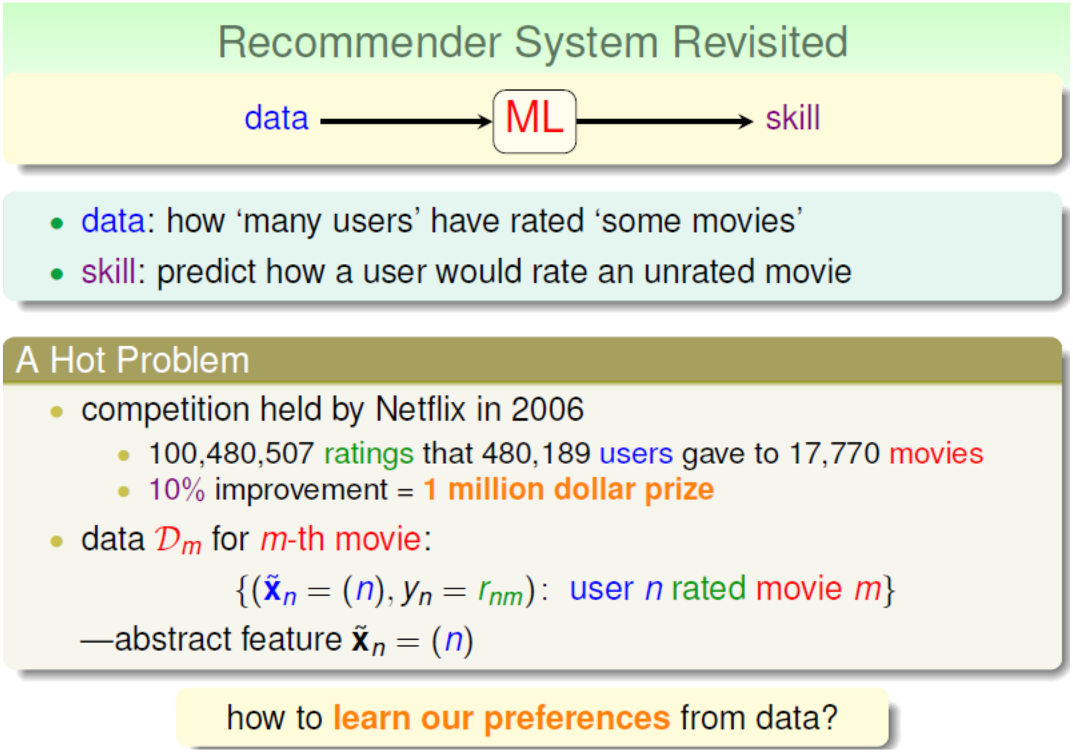
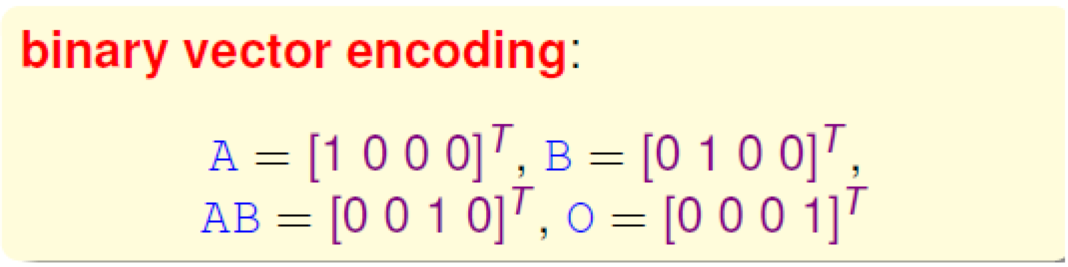
Matrix Factorization

一、Linear Network Hypothesis

1. 回顾一个基石里面的实际应用：Netflix的比赛，通过每个用户给许多电影的打分做一个推荐系统



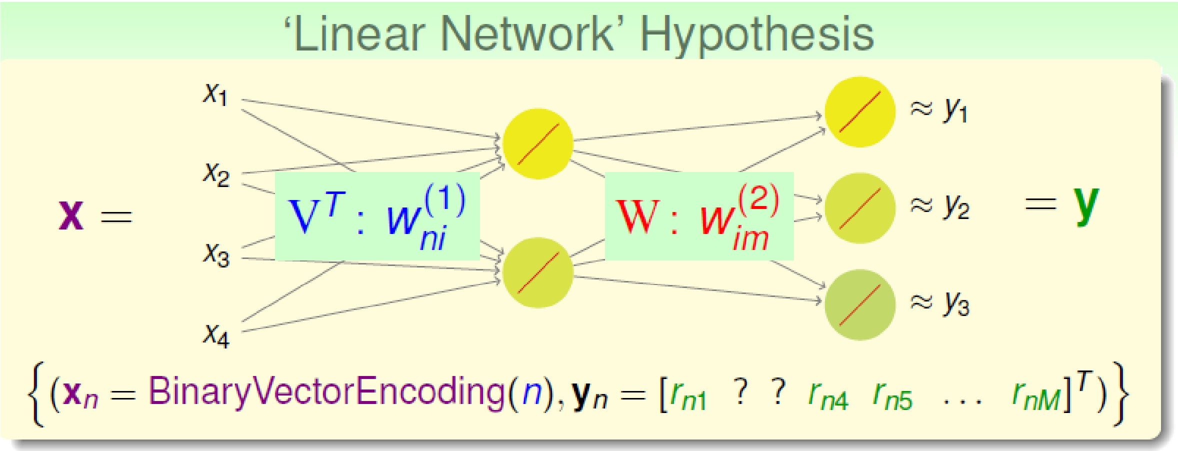
这里我们的输入就简单的是类别编号n——啥意思没有。。。我们回顾以前学习的模型，这些模型往往有一个特点——就是它们都青睐数值型的数据（好像决策树系列除外）那么我们有没有一个好方法来对付这类categorical features（离散的类别名称）呢？接下来介绍binary vector encoding。

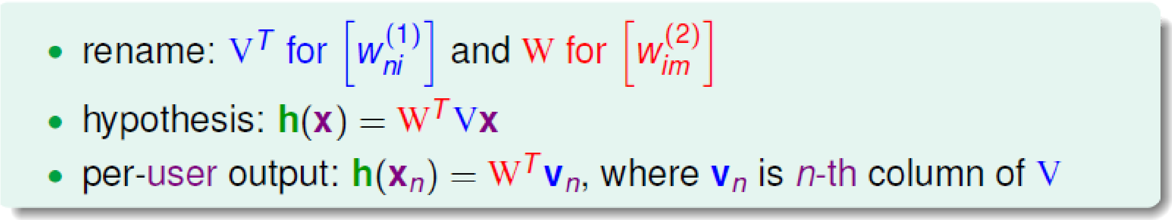


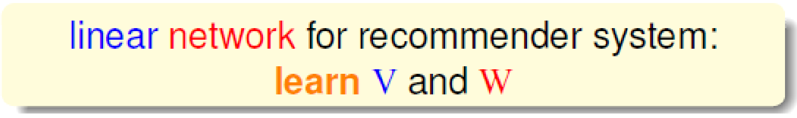
我们对于输入输出采取这样的操作：

$$x_n = [0, 0, .., 1, ..., 0]^T, y_n = [r_{n1}, ?, ?, r_{n4}, ..., r_{nM}]^T$$

确定好input output之后我们来用一个$N-\tilde{d}-M$ NNet(为了简便，放弃bias项)来学习这些特征。注意到输出传入进去的实际只有一项是有效的，所以非线性在这里没有太大的必要——我们可以放弃non-linear transform：







我们定义两个矩阵来表示这个权重：

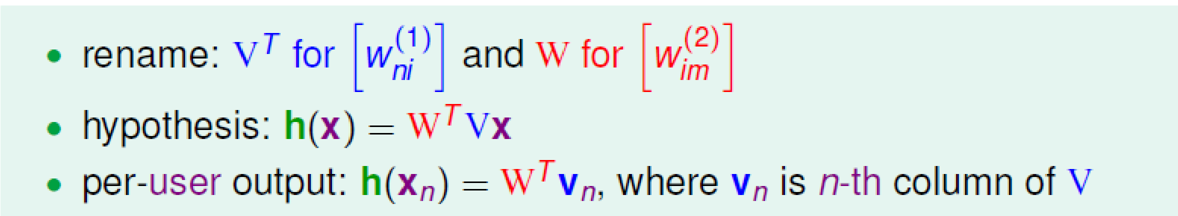
- $V^T : W_{ni}^{(1)}$
- $W : W_{im}^{(2)}$

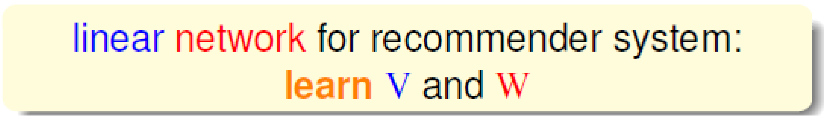
我们作运算，

$$h(x_n) = W^T(Vx_n) \tag{1}$$

注意到Vx_n实际上就是对于V的column space的线性组合，那么x_n里面只有一项非零，我们把相应的乘积（V中的第n列）叫做v_n。那么：

$$h(x_n) = W^T v_n \tag{2}$$



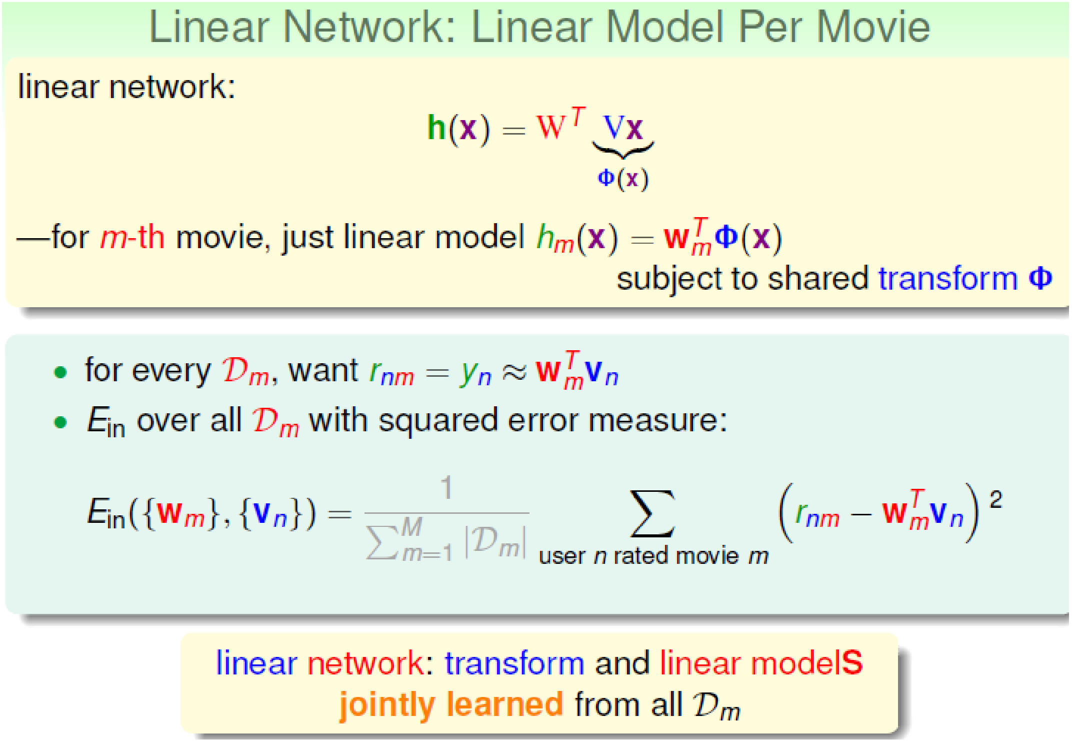


那么我们需要做的事情无非两件事：学习 V 和 W。

二、Basic Matrix Factorization

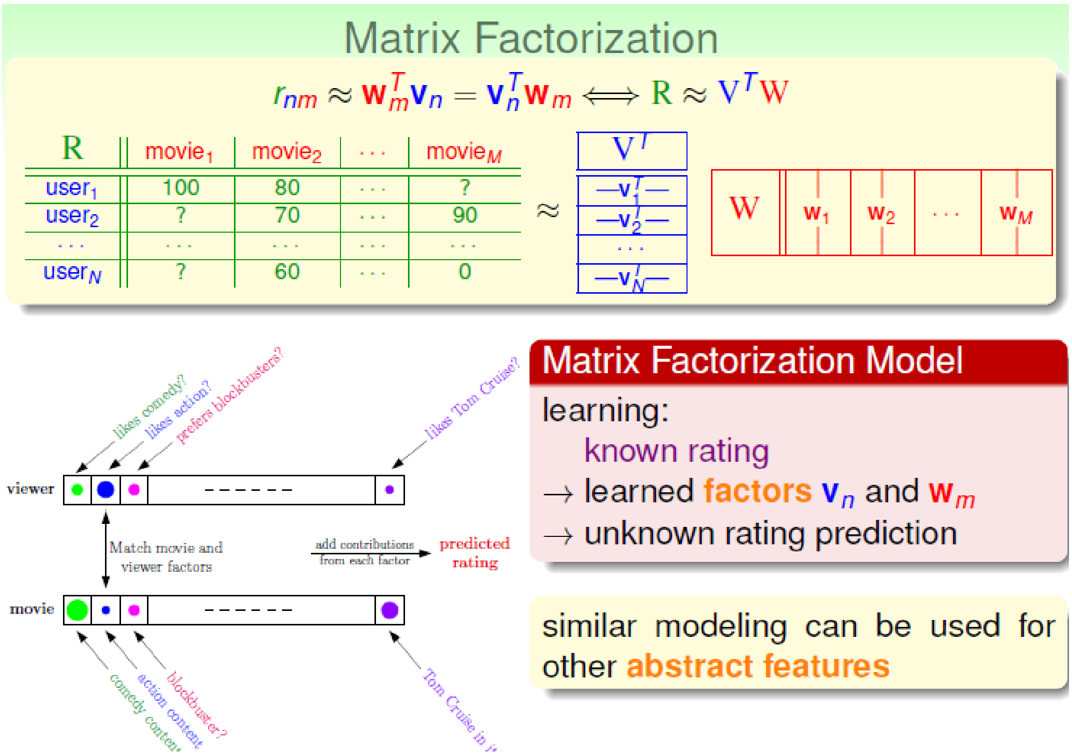
1. 我们因为是在一个线性模型中，所以NN就是一个linear network，我们对于每个电影进行考虑。考虑E_{in}

$$E_{in}(w_m, v_n) = \frac{1}{\sum_{m=1}^{M}|D_m|}\sum_{user\ n\ rated\ movie\ m}\left(r_{nm} - w_m^T v_n\right)^2 \tag{3}$$



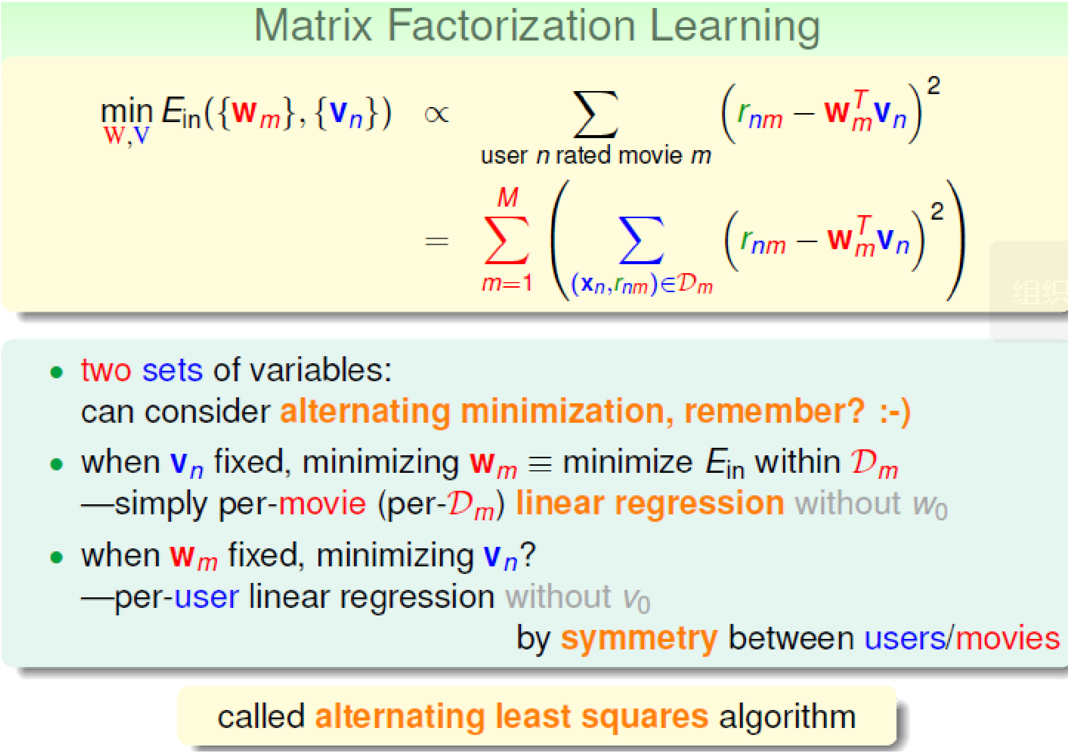
2. Matrix Factorization:

我们列一下电影打分表格：



我们预测$r_{nm} \approx w_m^T v_n \Leftrightarrow R \approx V^T W$ 本质就是matrix factorization

3. 看一下最佳化的问题：

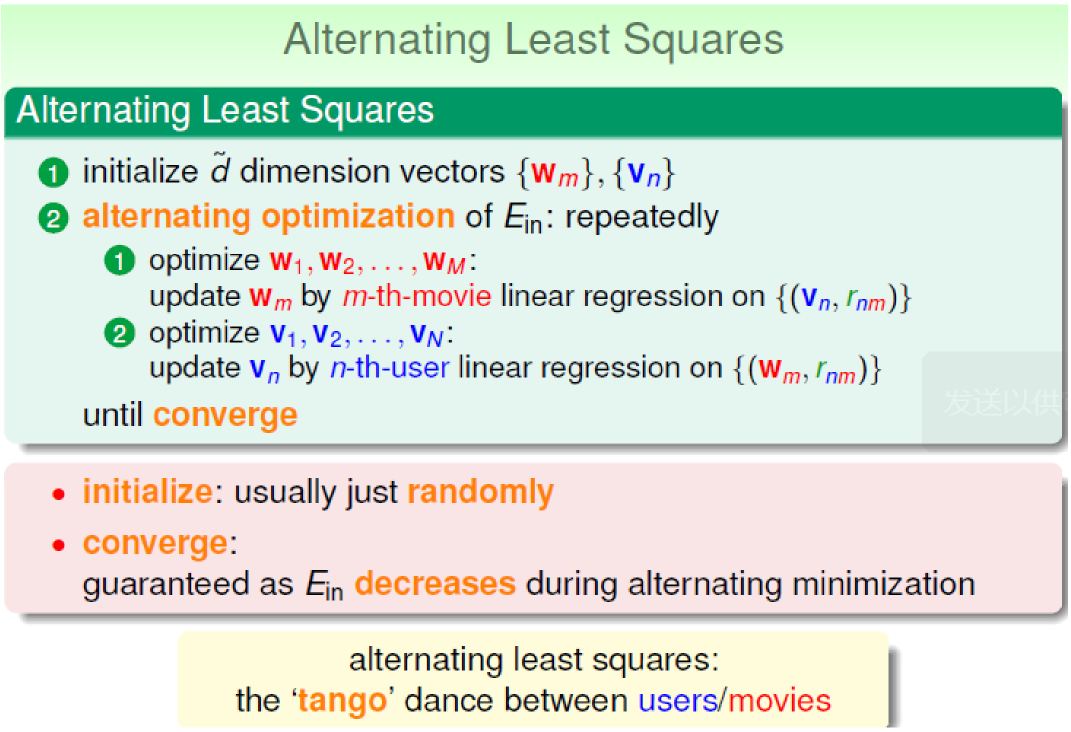


和之前K-Means一样我们都是两组变量的优化问题，我们还是使用调整法：

- 固定V，我们对W（省略bias）进行linear regression
- 固定W，我们对V（省略bias）进行linear regression

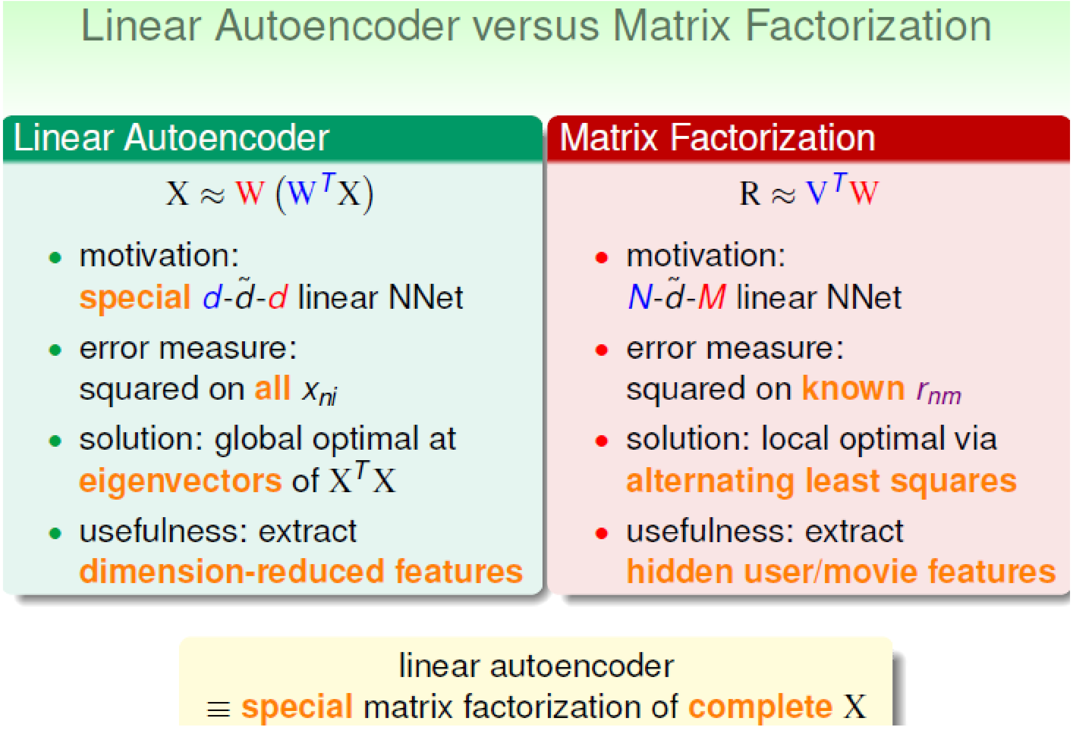
  实际上！两个矩阵是对称的 😄这个最优化的策略和之前一样都是一个alternating的策略，特别的治理叫做alternating least squares algorithm

我们总结一下算法的流程：



- 随机初始化V, W
- alternating optimization：双重linear regression（矩阵求解）
- 直到收敛就停止（看看E_{in}变了没有）

4. 这个方法实际上让我们想起了之前在PCA里面的转化



我们做一个对比：

	PCA	Matrix Factorization
网络结构	$d - \tilde{d} - d$ linear NNet	$N - \tilde{d} - M$ linear NNet
误差衡量	x_{ni}	r_{nm}
解决办法	$X^T X$的最大特征值	轮流均方误差优化
用武之地	降维特征提取	（电影）特征提取

三、Stochastic Gradient Descent

1. 我们除了用这个矩阵方法解决最优化的问题，实际上我们还可以利用老朋友SGD，实际上这个更流行：

Another Possibility: Stochastic Gradient Descent

$$E_{\text{in}}(\{\mathbf{w}_m\},\{\mathbf{v}_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \underbrace{\left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)^2}_{\text{err(user } n, \text{ movie } m, \text{rating } r_{nm})}$$

SGD: randomly pick **one example** within the $\sum$ &
update with **gradient to per-example** err, **remember? :-)**

- **'efficient'** per iteration
- **simple** to implement
- easily extends to **other** err

next: **SGD** for matrix factorization

2. 我们先求一下梯度:

Gradient of Per-Example Error Function

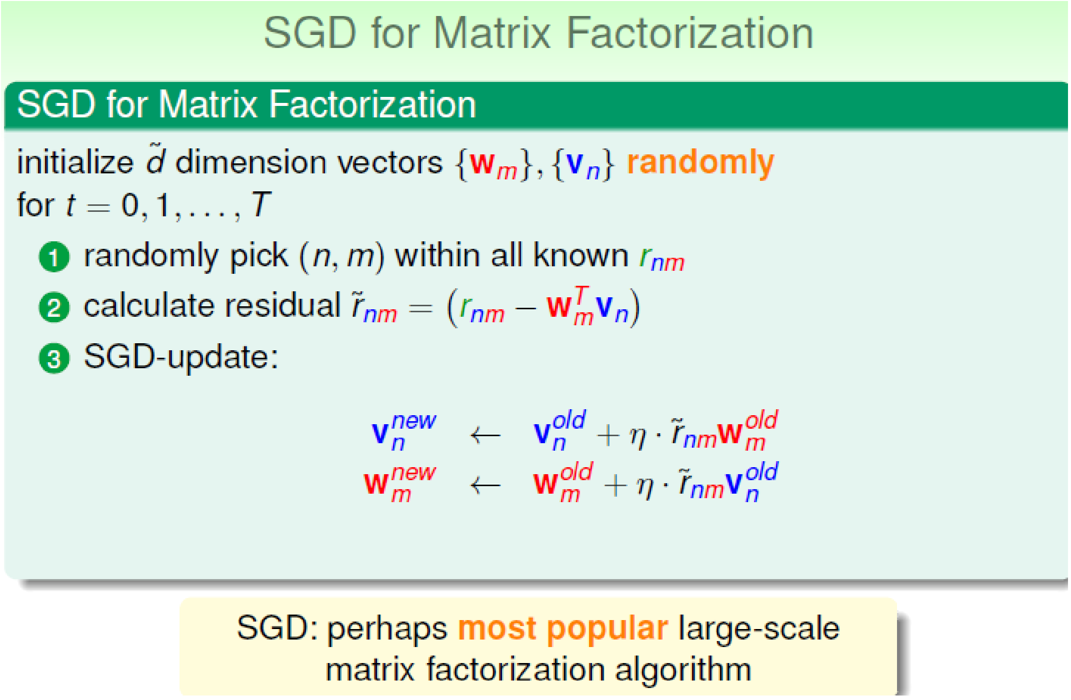
$$\text{err(user } n, \text{ movie } m, \text{rating } r_{nm}) = \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)^2$$

$$\begin{aligned}
\nabla_{\mathbf{v}_{1126}} \quad & \text{err(user } n, \text{ movie } m, \text{rating } r_{nm}) = \mathbf{0} \text{ unless } n = 1126 \\
\nabla_{\mathbf{w}_{6211}} \quad & \text{err(user } n, \text{ movie } m, \text{rating } r_{nm}) = \mathbf{0} \text{ unless } m = 6211 \\
\nabla_{\mathbf{v}_n} \quad & \text{err(user } n, \text{ movie } m, \text{rating } r_{nm}) = -2\left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right) \mathbf{w}_m \\
\nabla_{\mathbf{w}_m} \quad & \text{err(user } n, \text{ movie } m, \text{rating } r_{nm}) = -2\left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right) \mathbf{v}_n
\end{aligned}$$

per-example gradient
$\propto -$(**residual**)(**the other feature vector**)

求导之后出来的结果是余数项（残差）和向量的内积

3. 总结一下流程：只不过我们这个地方一次优化双份。。。



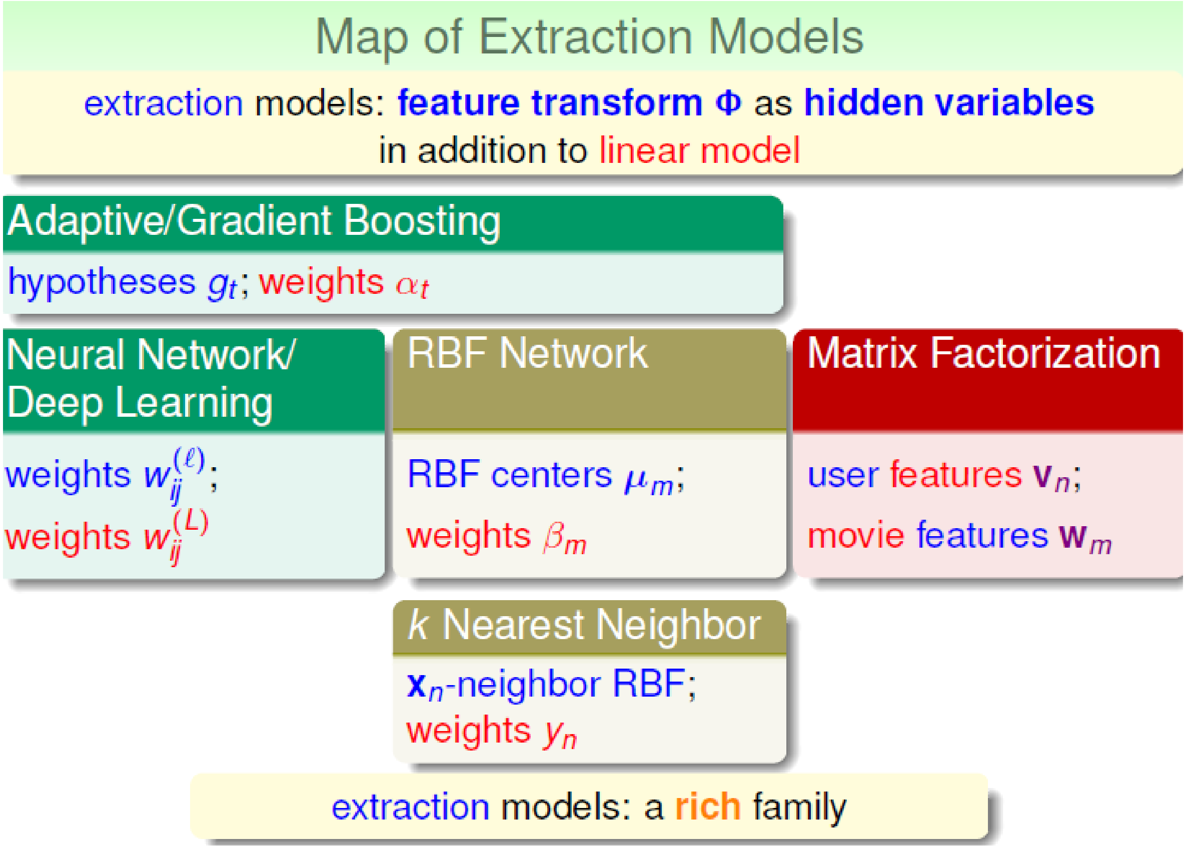
4. 鲜活的例子——SGD应用：

对于电影评价的时候，我们往往最近（时间靠后）的电影打分应该来说评分是更重的，但是我们在权重上应该怎么体现呢？我们直到SGD是对选定的一个点进行梯度下降，所以这个点而言梯度下降的意义非凡。我们希望SGD选的点都是靠后时间的点！

妙啊😸

四、Summary of Extraction Model

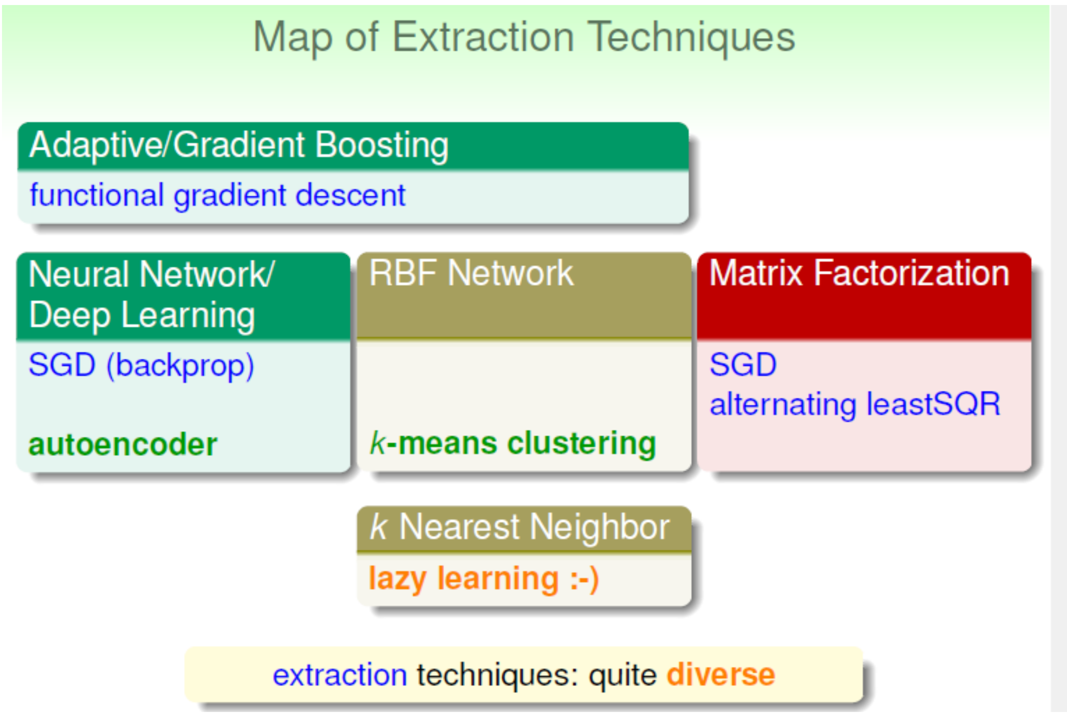
1. 总结一下extraction models!



神经网络、RBF网络、k近邻法、矩阵分解

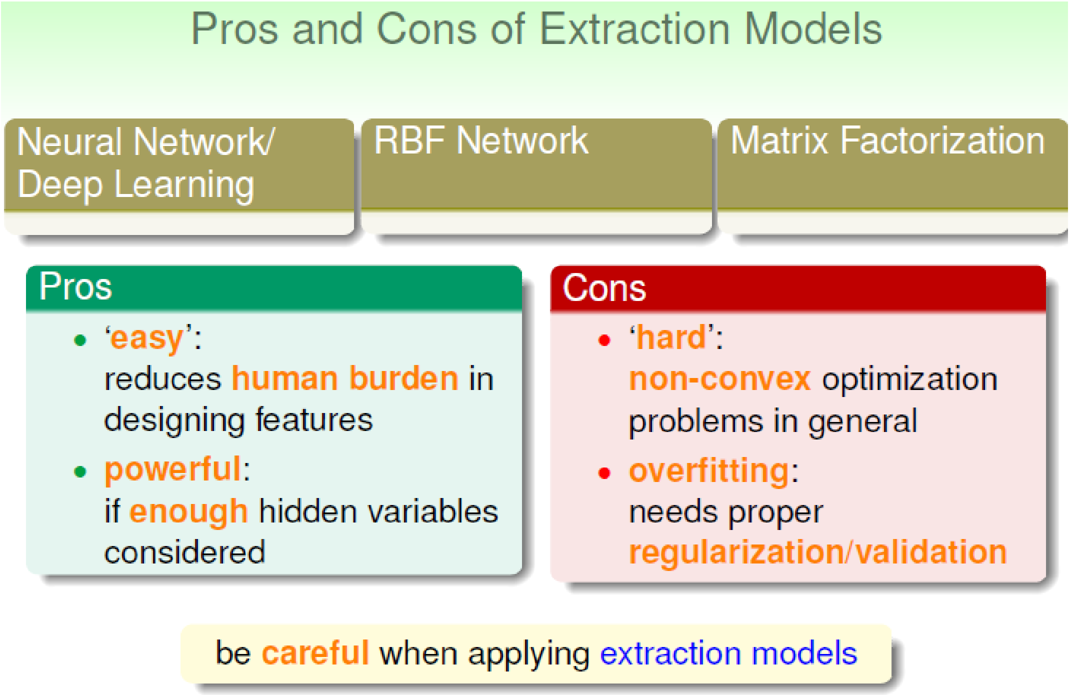
其实 boosting (AdaBoost、Gradient Boost) 的方法也可以看作是一种 extraction （因为实际上权重就是一种特征的提取转换操作！）

2. 同时我们在extraction models中蕴含着很多extraction techniques：



- GB: 函数梯度下降
- NN： SGD + autoencoder (PCA、denoising autoencoder)
- RBF Network: K-Means Clustering
- K-NN (lazy learning emoji😚
- Matrix Factorization: SGD + alternating least square optimization

3. extraction models的优缺点：



四字箴言：小心起见