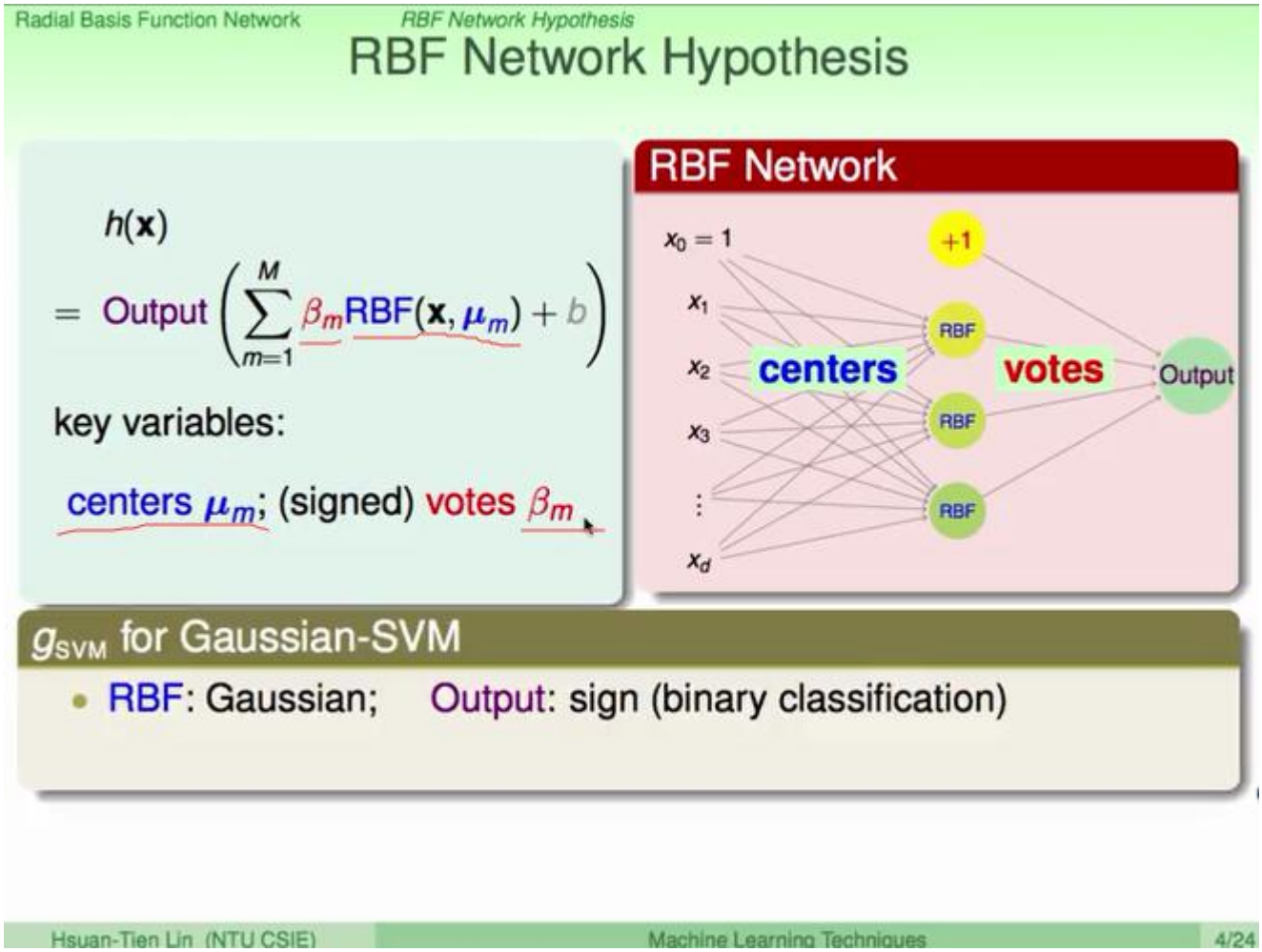


从 ppt 这一页可以知道 RBFNN 关键点是两个：

1、第一层：得到 RBF 函数的中心 μ_m（视频中后面讲解了几种求解方法（我们用 Kmean））

2、第二层：计算投票权重 β_m（也就是线性组合的系数）。

```
def __init__(self, k=10, delta=0.1):
    '''
    delta: 高斯函数中的扩展参数
    beta: 隐层到输出层的权重
    k: 中心的个数
    '''
    self._delta = delta
    self._beta = None
    self._hidden_num = k
    self.kms = KMeans(k)
    pass
```

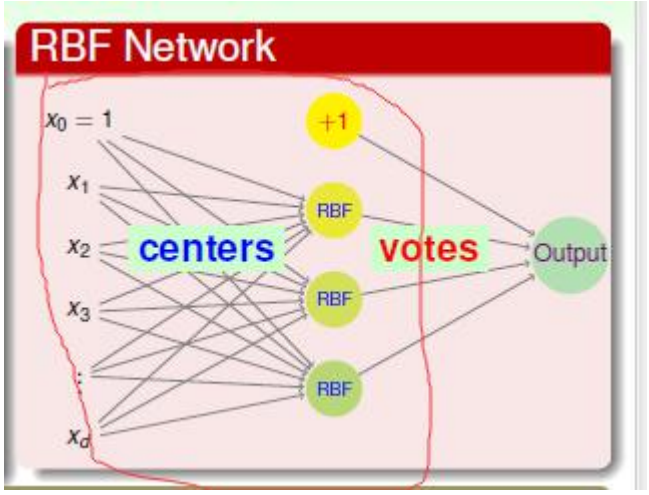
初始化了我们需要的参数，其中 delta：高斯核函数中的 λ 参数。beta：ppt 中提到的 β_m（也就是线性组合的系数），然后下面的就是计算中心点 μ_m（用 Kmean 计算）

```
def _calRBF(self,x,c):
    '''
    计算RBF函数的输出，这里使用高斯函数
    '''
    return np.exp(-self._delta* np.sqrt(np.sum(np.square(x-c))))
```

高斯核函数公式：$$\text{Gaussian}(\mathbf{x},\mathbf{x}') = \exp(-\gamma\|\mathbf{x}-\mathbf{x}'\|^2)$$

代码这一段实现了 RBF 函数（后面备用），其中 x 表示样本数据，c 表示求得的中心。

```
def _calG(self, X):
    '''
    输入层到隐层的特征转换
    G相当于公式中的大写的Z=[z1,z2,z3...zN], N为数据样本里
    G维度: N * hidden
    '''
    num, dim = X.shape
    G = np.empty((num, self._hidden_num))
    for i in range(num):
        for j in range(self._hidden_num):
            # 计算每一个数据与所有的重心的RBF输出，作为隐层神经元的输出
            G[i,j] = self._calRBF(X[i,:], self._centers[j])
    return G
```



$$\mathbf{z}_n = [\text{RBF}(\mathbf{x}_n,\mathbf{x}_1), \text{RBF}(\mathbf{x}_n,\mathbf{x}_2), \ldots, \text{RBF}(\mathbf{x}_n,\mathbf{x}_N)]$$

这部分展示了第一层的计算过程我们用 Kmean 求得中心点之后就可以计算 RBF 值了，图中得全连接网络在数学上就是一个矩阵维度为：[num(样本数)，hidden_num(隐层神经元数)]。这样就得到了 RBF 第一层的输出。

```
def _calPseudoInvese(self,x):
    '''
    计算矩阵伪逆
    '''
    return np.linalg.pinv(x)
```

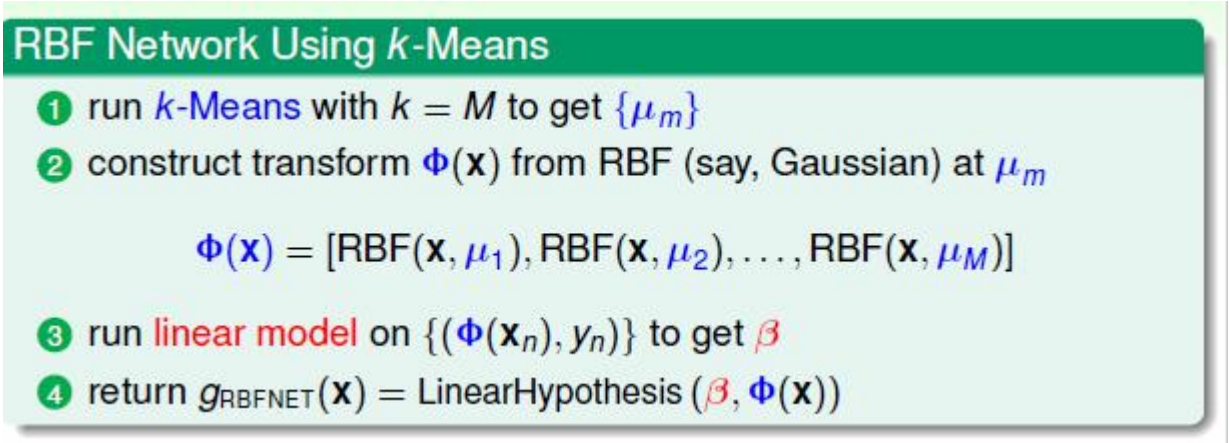
这段代码实现了一个矩阵求逆的函数（后面备用）

```
def fit(self, train_x, train_y):
    '''
    训练函数
    '''
    num, dim = train_x.shape

    # 使用KMeans无监督确定中心
    self.kms.train(train_x)
    self._centers = self.kms._centers
     self.kms.fit(train_x)
     self._centers = self.kms.cluster_centers_

    # 计算Z
    self.G = self._calG(train_x)

    # 计算权重矩阵,其中包含一个求伪逆的过程
    self._beta = self._calPseudoInvese(np.dot(np.transpose(self.G), self.G))
    self._beta = np.dot(self._beta, np.transpose(self.G))
    self._beta = np.dot(self._beta, train_y)
```



上面的代码实现了整个流程：

1、Kmean 求得中心点；

2、计算第一层输出：G 表示第一层输出。

3、计算第二层线性组合优化（目的求得我们需要的参数：线性组合权重矩阵 β_m ）。

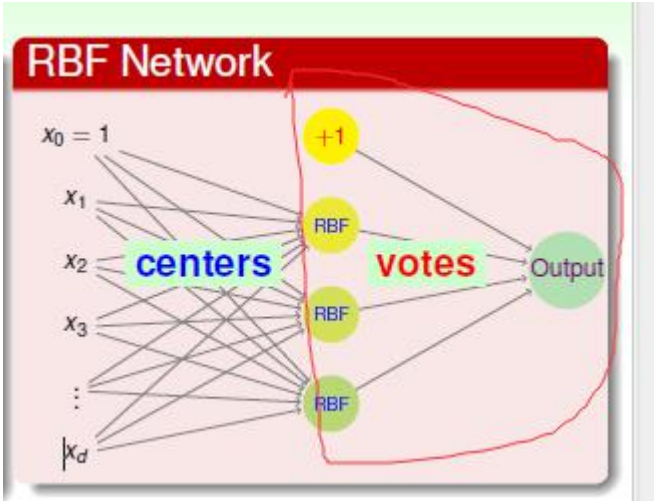
- optimal β? $\beta = (Z^T Z)^{-1} Z^T y$, if $Z^T Z$ invertible, remember? :-)

求解 β_m 的公式：线性组合参数通过残差最小方式优化是典型的方法（可以百度最小二乘求解）。上面的公式公式展示了 β_m 的计算公式（中间推导省略）。
通过这个函数我们就会得到我们需要的两个关键（开始说过了），中心点 μ_m 和权重 β_m 。

```
def predict(self, test_x):
    '''
    预测
    test_x: 可以是多个x
    '''
    if not isinstance(test_x, np.ndarray):
        try:
            test_x = np.asarray(test_x)
        except:
            raise TypeError('np.ndarray is necessary')
    if len(test_x.shape) == 1:
        test_x = test_x.reshape(1, test_x.shape[0])

    # 计算输入x的隐层的神经元的值
    # 相当于公式中\phi(X)
    G = self._calG(test_x)

    #计算最终输出
    Y = np.dot(G, self._beta)
    return Y
```



这段代码表示对数据的预测函数，可以看出来其中直接用了我们在 fit()函数中求得的参数输出结果。