INT301 W12

UNSUPERVISED LEARNING: COMPETITIVE LEARNING

Unsupervised Competitive Learning

在 Hebbian networks 中，所有神经元可以同时"激发"。

而竞争性学习 (competitive learning) 意味着每个时间步长中，每组只有一个神经元被激发。这意味着输出单元相互竞争。

These are **winner-takes-all (WTA)** units.

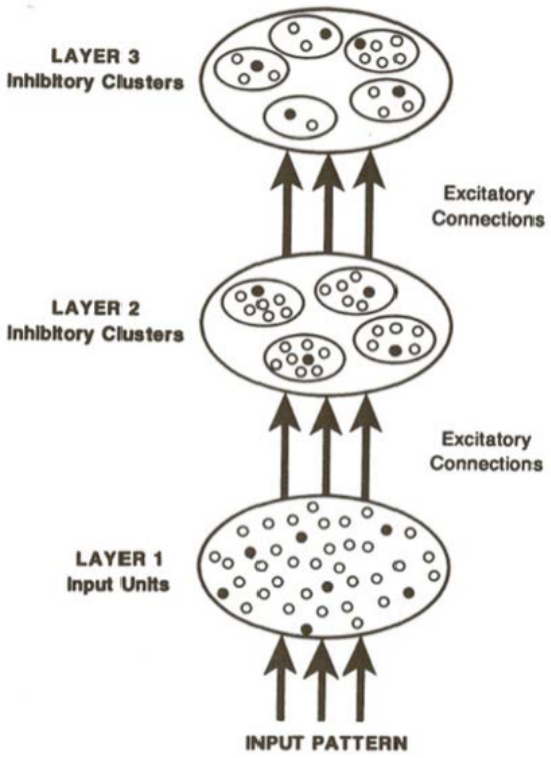
竞争对神经网络很重要：

- 在生物神经系统中观察到神经元之间的竞争
- 竞争对于解决许多问题很重要

为了更好的理解，for example：

- 将一个输入模式进行分类，一共 m 类
    - 理想的情况：只有一个节点输出 1，其他所有节点都是 0
    - 不过，通常会有多个节点具有非零输出
    - 这时候可以引入竞争机制。如果这些类节点相互竞争，也许最终只有一个会赢，而所有其他节点都会输（winner-takes-all）。这样获胜者所代表的类就是最终的结果。

The architecture of competitive learning mechanism



竞争学习机制的架构如上图：

- unit（激活的或非激活的）在图中表示为点
  - 激活的 unit 由黑点表示，非激活的由白点表示
- 在 layer 中的 unit 可以：
  - 接收来自上一次中所有单元的输入，将输出传递给下一层中的所有单元
- 层之间的连接是刺激性的 (excitatory)
- 层内的连接是抑制性的 (inhibitory)：
  - 每层由一组相互抑制的 unit 的聚类组成
  - 集群中的 unit 相互抑制，使得每个聚类只有一个 unit 处于激活状态

Winner-takes-all (WTA)

实现 WTA 的最简单方法：有一个外部的中央仲裁员（一个程序）通过比较竞争对手的当前输出来决定获胜者（任意地打破平局）。

这在生物学上是不合理的（在生物神经系统中不存在这样的外部仲裁员）。

Simple Competitive Learning (SCL)

SCL 的过程如下：

- 初始化 k 个 prototype vectors (这里和 K-means 类似，K-means 是随机选 k 个中心，这里的 prototype vectors 也可以看作中心)
- 提供一个 example，即拿一个数据出来
- 把 example 和每一个 prototype vectors 的距离进行对比，离 example 最近的 prototype vectors 叫做 winner (这里和 k-means 又不一样，k-means 是把所有的数据都对比，而这里只对比一个)
- 将 winner 向 example 移动，移动多少距离可以自定义

直观清晰、合理的程序：

- 将原型放置在数据密度高的区域
- 识别最相关的 feature 组合

Winner:

Winner = 输出节点的传入权重，与输入向量的欧氏距离最短

$$h_j = \sum_i w_{ji} x_i$$

$$w_{j^*} \cdot x \geq w_j \cdot x \quad \forall x$$

注：第一个公式中，h_j 是第 j 个输出节点的值，其中 w 是 prototype vectors，x 是数据，$w_{ji}x_i$ 用来衡量数据和 prototype vectors 的距离 (注意，这里是 inner product，即当两个向量的角度接近 0，那么 x 和 w 很接近，则结果很大；如果角度接近 90，那么 x 和 w 很远，则结果是 0)。

第二个公式中，这个公式是确定 winner 的。其中 w_{j^*} 是 winner，w_j 代指剩余的 unit。这个公式的意思就是，传入权重与输入向量的欧氏距离最短的那个 unit，就是 winner。

Update rule for all neurons

$$\Delta w_{j^*i} = \eta y_j (x_i - w_{j^*i})$$
$$\begin{cases} y_{j^*} = 1 \\ y_j = 0 \ \text{ if } \ j \neq j^* \end{cases}$$

仅修改 winner 节点的传入权重。

除了上面这种，还可能有其他的 update rule：

$$\Delta w_{j^*i} = \eta x_i$$

$$\Delta w_{j^*i} = \eta' \left(\frac{x_i}{\sum_i x_i} - w_{j^*i} \right)$$

$$\Delta w_{j^*i} = \eta \left(x_i - w_{j^*i} \right)$$

SCL Example

6 Cases:

(0 1 1) (1 1 0.5)
(0.2 0.2 0.2) (0.5 0.5 0.5)
(0.4 0.6 0.5) (0 0 0)

$$\Delta w_{j^*i} = \eta y_j \left(x_i - w_{j^*i}\right)$$

$$\begin{cases} y_{j^*} = 1 \\ y_j = 0 \quad if \quad j \neq j^* \end{cases}$$

Learning Rate: 0.5
Initial Randomly-Generated Weight Vectors:
[0.14 0.75 0.71]
[0.99 0.51 0.37] Hence, there are 3 classes to be learned
[0.73 0.81 0.87]

Training on Input Vectors
Input vector # 1: [0.00 1.00 1.00]
Winning weight vector # 1: [0.14 0.75 0.71] Distance: 0.41
Updated weight vector:
[0.07 0.87 0.85][0.99 0.51 0.37] [0.73 0.81 0.87]

Input vector # 2: [1.00 1.00 0.50]
Winning weight vector # 3: [0.73 0.81 0.87] Distance: 0.50
Updated weight vector:
[0.07 0.87 0.85][0.99 0.51 0.37]**[0.87 0.90 0.69]**

注：这里以 input vector 1 为例，先计算出 Δw_{j^*i}。由于 $\begin{cases} y_{j^*} = 1 \\ y_j = 0 \text{ if } j \neq j^* \end{cases}$，因此我们只考虑当前的 winner。因此 $\Delta w_{j^*i} = 0.5 \times 1 \times (x_i - w_{j^*i})$，最后的结果是：[-0.07 0.125 -0.145]，然后用 winner 加上 Δw_{j^*i}，即得到结果。

```
Input vector # 3:   [ 0.20  0.20  0.20 ]
   Winning weight vector # 2: [ 0.99  0.51  0.37 ] Distance:  0.86
   Updated weight vector:
   [ 0.07  0.87  0.85 ][ 0.59  0.36  0.29 ][ 0.87  0.90  0.69 ]

Input vector # 4:   [ 0.50  0.50  0.50 ]
   Winning weight vector # 2: [ 0.59  0.36  0.29 ] Distance:  0.27
   Updated weight vector:
   [ 0.07  0.87  0.85 ][ 0.55  0.43  0.39 ][ 0.87  0.90  0.69 ]

Input vector # 5:   [ 0.40  0.60  0.50 ]
   Winning weight vector # 2: [ 0.55  0.43  0.39 ] Distance:  0.25
   Updated weight vector:
   [ 0.07  0.87  0.85 ][ 0.47  0.51  0.45 ][ 0.87  0.90  0.69 ]

Input vector # 6:   [ 0.00  0.00  0.00 ]
   Winning weight vector # 2: [ 0.47  0.51  0.45 ] Distance:  0.83
   Updated weight vector:
   [ 0.07  0.87  0.85 ][ 0.24  0.26  0.22 ][ 0.87  0.90  0.69 ]
Finish of Epoch 1

Clusters after epoch 1:
 Weight vector # 1: [ 0.07  0.87  0.85 ]
    Input vector # 1:   [ 0.00  1.00  1.00 ]
 Weight vector # 2: [ 0.24  0.26  0.22 ]
    Input vector # 3:   [ 0.20  0.20  0.20 ]
    Input vector # 4:   [ 0.50  0.50  0.50 ]
    Input vector # 5:   [ 0.40  0.60  0.50 ]
    Input vector # 6:   [ 0.00  0.00  0.00 ]
 Weight vector # 3: [ 0.87  0.90  0.69 ]
    Input vector # 2:   [ 1.00  1.00  0.50 ]

Weight Vectors after epoch 2:
  [ 0.03  0.94  0.93 ]
  [ 0.19  0.24  0.21 ]
  [ 0.93  0.95  0.59 ]

Clusters after epoch 2:
   unchanged.
```

以上方法，形成的聚类对初始权重向量高度敏感 (不同的初始 prototype vectors 会得到不同的聚类)。

Enforcing fairer competition

output unit 的权重矢量的初始位置可能位于模式很少的区域。

有些 unit 可能永远不会或很少成为 winner，因此权重矢量可能不会更新，从而阻止它找到更丰富的模式空间部分。***DEAD UNIT***

更有效率，以确保更公平的竞争，每个 unit 都有平等的机会代表训练数据的某些部分。

Leaky learning

这个是对上面 SCL 的修改：同时更新 winner 和 losing (其余) unit 的权重，但学习率不同：

$$w(t+1) = w(t) + \begin{cases} \eta_w(x - w(t)) \\ \eta_L(x - w(t)) \end{cases}$$
$$where\ \eta_w \gg \eta_L$$

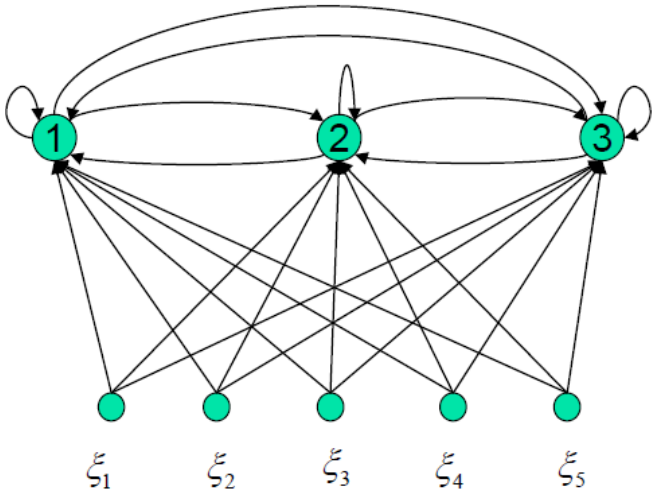
它具有将 losing unit 缓慢地向更密集区域模式空间移动的效果。

Maxnet

Lateral inhibition

执行 Winner Take All (WTA) 竞争的特定竞争网络是 Maxnet。

每个节点的输出通过抑制性连接（负权重）馈送到其他节点 (同一个 layer 之间)。

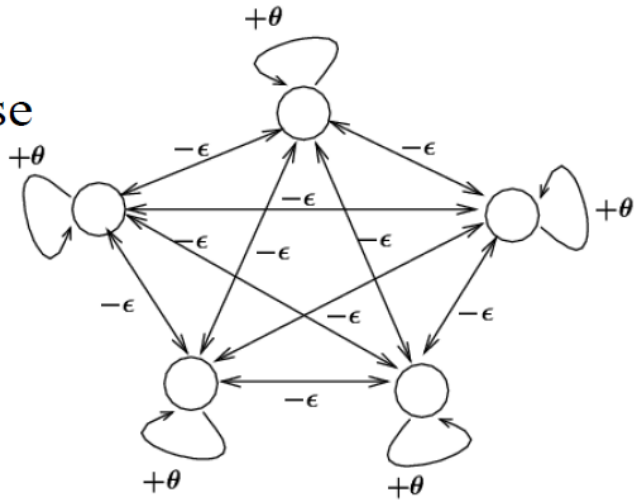


Maxnet

竞争对手之间的横向抑制 (lateral inhibition)，下面这些都是同一层内的 unit。

$$\text{weights}: w_{ji} = \begin{cases} \theta & \text{if } i = j \\ -\varepsilon & \text{otherwise} \end{cases}$$

$$\text{node function}: f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



经过上面的步骤，最终只有一个 node 可以输出 1，其他的都是 0。

Notes:

- 竞争：进行迭代，直到网络稳定（最多一个具有正激活的节点）$0 < \varepsilon < 1/m$, where m is the number of competitors
- ε 太小：收敛时间太长
- ε 太大：可能会抑制整个网络 (没有 winner)

Mexican Hat Network

在所有竞争节点中，只有一个会赢，所有其他节点都会输。

- 在输出层的所有神经元中，激活水平最高的神经元成为赢家。该神经元是唯一产生输出信号的神经元。所有其他神经元的活动在竞争中被抑制。

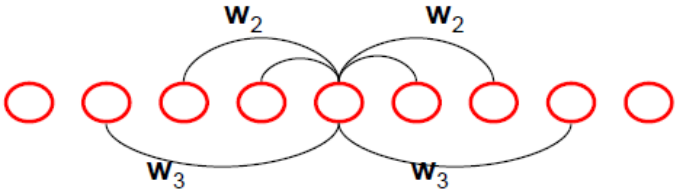
我们主要处理单 Winner WTA，但多个 winners WTA 是可能的。

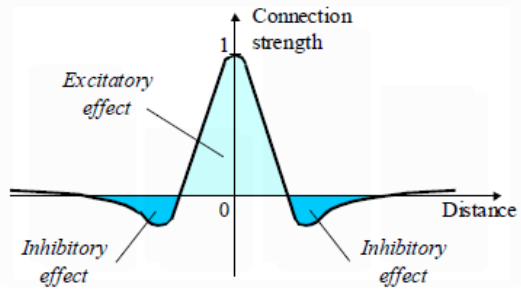
- 侧向连接产生刺激或抑制作用，这取决于与 winner 神经元的距离。
- 这是通过使用 Mexican Hat function 来实现的，该函数描述了输出层中神经元之间的突触权重。

Architecture

对于一个给定的 node (近交远攻):

- close neighbors: cooperative (合作，相互刺激，w > 0)
- distant neighbors: competitive (竞争，相互抑制，w < 0)
- too far away neighbors: irrelevant (w = 0)





需要一个距离（neighborhood）的定义：

- 一维：按索引排序（1，2, ... n）
- 二维：lattice (格子，可以当作距离)

均衡 (Equilibrium)：

- negative input = positive input for all nodes (刺激和抑制的数量相等)
- winner has the highest activation
- its cooperative neighbors all have positive activations
- its competitive neighbors all have negative (or zero) activations

weights

$$w_{ij} = \begin{cases} c_1 & \text{if} \quad \text{distance}(i,j) < k \ (c_1 > 0) \\ c_2 & \text{if} \quad \text{distance}(i,j) = k \ (0 < c_2 < c_1) \\ c_3 & \text{if} \quad \text{distance}(i,j) > k \ (c_3 \le 0) \end{cases}$$

activation function

$$f(x) = \begin{cases} 0 & \textit{if} \quad x < 0 \\ x & \textit{if} \quad 0 \le x \le \max \\ \max & \textit{if} \quad x > \max \end{cases}$$

ramp function :

Vector Quantization

Idea：

- 使用竞争学习算法将给定的输入向量集分为 M 类
- 仅通过其所属的类来表示任何向量

竞争学习的重要应用（特别是在数据压缩 (data compression) 中）：

- 将整个模式空间划分为多个单独的子空间
- M 个 units 的集合表示 prototype vectors 的集合 - ***CODEBOOK***
- 新模式 x 基于其与 prototype vectors 的接近程度，使用欧式距离分配给聚类

A codebook：

- a set of centroids/codewords/codevector：

$$\{m_1, m_2, \ldots m_k\}$$

注：上面的元素都是每个类的 prototype vector

A quantization function (量化函数):

$$q\left(x_i\right) = m_k$$

根据上面的 quantization function，我们可以对 new pattern x 进行分类。方程 q 会找到 x 的 nearest-neighbor。

K-means 可用于构建 codebook。