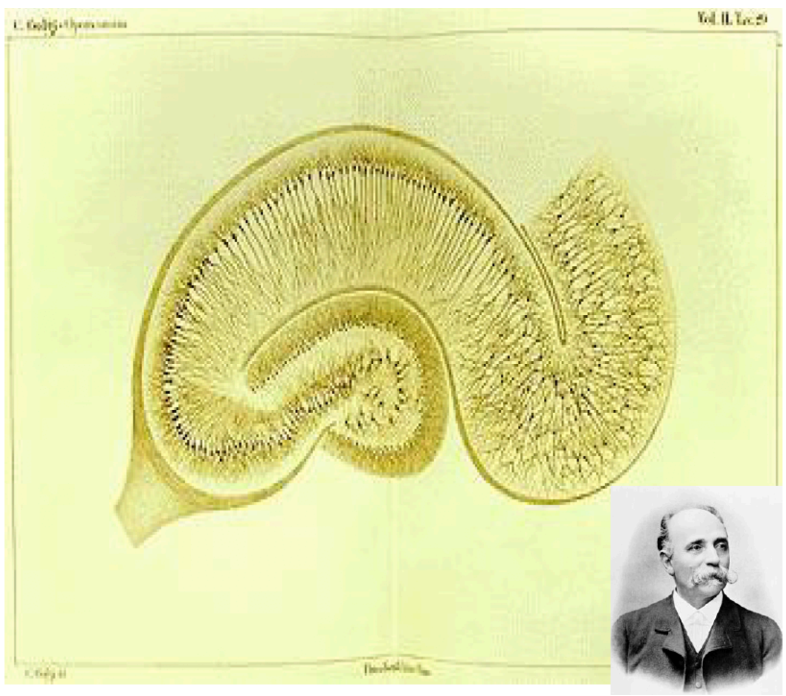
Biological Neurons

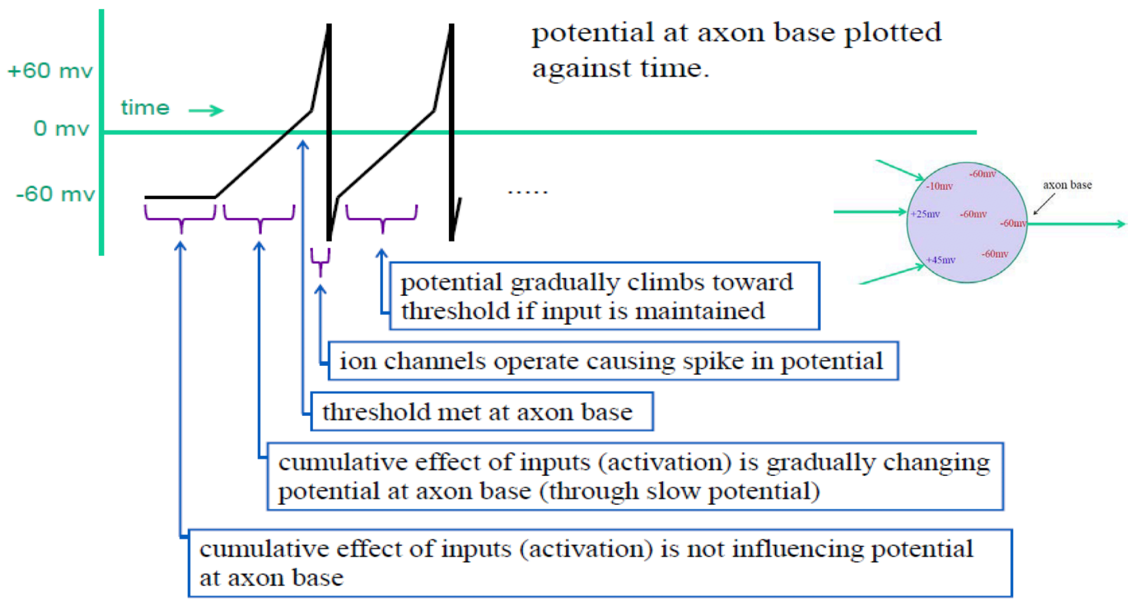
- The human brain has about **10^{11}** very tiny neurons. Each neuron is connected to about **10^{4}** other neurons.
- Each neuron **only connects directly to a very small portion of other neurons** <=> as if you could only communicate directly with 3 other people in Canada.
- 神经元(Neuron)在大脑中的不同位置呈现出来的形状会相当的不同
- The **cerebral cortex consists of many layers of neurons** with one layer feeding into the next. Layer boundaries are not rigid and some connections jump across layers. 在大脑皮层中有着许多层的神经元，层层相连。层之间的边界不是固定的，有些链接跨越层
- 每个神经元与同一层的许多（但不是全部）临近的神经元相连
- 1900年，*Camillo Golgi*发明了一种给神经物质染色的方法：*black reaction*



- 没有用的有趣的知识
  - 大脑占据着人类体重的1%，却消费着25%的身体能量
  - 几乎所有的神经元在人类出生之前就已经存在了，和身体的其他细胞不一样的是，神经元只有在婴儿发育期间才可以分裂以产生新的神经元。过了这个发育的时期之后，是不会再有新的神经元产生。他们只有形状会发生改变，这些神经元是要持续一辈子的。
  - 在童年期间，特别是在青春期，一个叫做*Synaptic Pruning（突出修剪）*的过程会发生。虽然大脑一直在发育，但是神经元和突触的数量最多会减少50%，减去了没有用的神经元，用更佳复杂和有效的结构去代替他们，这些结构更加适合成年生活

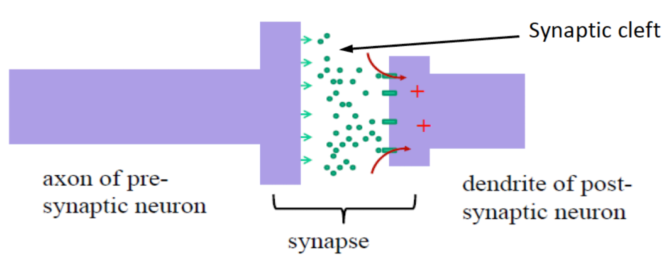
- 每个神经元的细胞壁（cell membrane）所包围着的细胞质（cytoplasm）中溶解了离子（ions）。神经元本身处于一种salt solution中，这个salt solution就是一个extracellular fluid（细胞外的流体），在这些细胞外的流体中，主要包括了**正钠离子（positively charged sodium ions）**和**负氯离子（negatively charged chloride ions）**。细胞质中的离子主要包括了**正钾离子（positively charged potassium ions）**和**负有机分子（large negatively charged organic molecules）**（比如蛋白质）。
- ***Resting Potential***: Unstimulated, 神经元维持着一个constant的potential，这个potential叫做**Resting Potential**, 在细胞中永远是负的，从-40到-90 millivolts， 如果一个神经元stimulated，那么在神经元里面的Resting Potential会根据刺激变得more or less negative.
    - 带电的物质无法通过由lipids（油脂）组成的细胞壁， 他们必须通过channel proteins（通道蛋白：**通道蛋白**是一类跨越细胞膜磷脂双分子层的蛋白质）that extend through the membrane.
    - 在一个unstimulated的神经元中，只有钾离子才可以通过一种特定的蛋白质：potassium channel（钾通道）来穿越细胞壁，虽然在细胞壁上面，sodium channel（钠通道）同样存在，不过在一个unstimulated的神经元上面，他是关上的。因为只有钾离子可以穿越细胞壁，并且在神经元中钾离子更加集中，所以他们会慢慢的散出神经元，只剩下大的负有机离子，越来越多的正钾离子离开，所以在这个神经元中变得越来越负，但是因为“同性相斥，异性相吸”，所以现在这个神经元对于离去的钾离子形成了一个强大的吸引力，这个强大的吸引力tends to把离去的钾离子拉回来。在某个阶段，钾离子的散开和这个吸引力会平衡，当达到这个平衡的时候，那么这个神经元就会达到**Resting Potential**
    - 这样的方式来达到Resting Potential并不需要改变神经元内外的钾浓度，只有大概万分之一的钾离子需要离开来达到Resting Potential（-60 millivolts）。不过在神经系统中，真正携带信息的是Action Potential
- ***Action Potential（动作电位）***: 在被stimulated之后，如果神经元里面的Resting Potential变得足够less negative，并且达到**threshold**的值，那么一个**Action Potential**就会被triggered。在这个Action Potential被triggered的期间，这个neuron的potential会变成20到50 millivolts inside. Action Potential只会持续很短的一个时间，然后这个neuron会恢复成他的negative resting potential。
    - 动作电位怎么产生?
        1. 当神经元受到了刺激（神经传递物质/电刺激），细胞膜上的"化学敏感通道Na^+"开启
        2. Na^+的通透性增加，就是说会有更多的Na^+流入细胞内
        3. 细胞膜内外的电位差减小
        4. 此时的膜电位状态的变化被称为**“去极化”**，也就是说，由原来的外正内负变成“外没那么正，内没那么负”
        5. 若这个去极化上升到阈值的程度，会引发大量的“电压敏感Na^+通道”开启，一旦达到这个状态 >>> 全开!
        6. ***产生非常剧烈的去极化，这个如此剧烈的去极化被称为：动作电位***
        7. 在这个状态下，当细胞膜电位逐渐增加，"电压敏感K^+通道"开启
        8. K^+会跑出去，会慢慢减缓去极化的作用
        9. K^+持续流出细胞，细胞外正电荷增加，细胞膜电位逐渐降低，那么会恢复静止膜电位，这个过程叫做**“再极化”**，但是这个电压敏感K^+通道关闭的很慢，那么细胞内的钾离子会流逝的太多，造成**“过极化”**，就是细胞内的膜电位降到低于原来的Resting Potential
        10. Na^+/K^+泵恢复离子分布，慢慢恢复成原来的**极化**状态。

- Thus a spike in potential (the action potential) moves gradually down the axon. **Note that after operating, the ion pumps need a short period to recover before they can operate again. This prevents the spike in potential from moving backwards on the axon. Also note that the potential at the axon base falls slightly when the spike is being generated. <u>If the neuron returns to the threshold value, another spike of activity will start on the axon</u>.**
- 基本上动作电位产生的过程的膜电位变化如图所示：

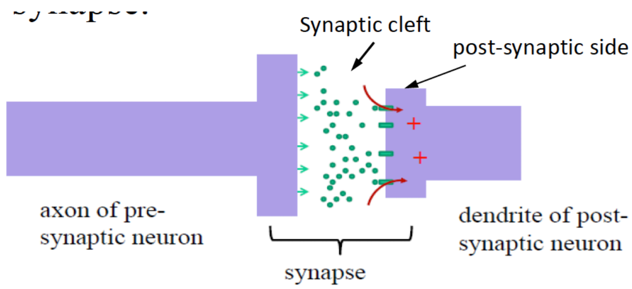


- "被stimulated"这个过程实际上是一个非常复杂的过程
  1. 当有stimulate来的时候，这个signal会改变位于dendrites（树突）的potential，当这个potential坚持了下来，整个神经元内的potential都会被改变。这是因为"positive and negative ions attract each other"，所以他们会结合成一个统一。这个改变被称为"slow potential"。
  2. 当位于Axon（轴突）Base的potential的和大于阈值threshold的时候，一个"gate"会被打开，离子会流入轴突

- 当***Action Potential***到达轴突的末端，会导致neurotransmitters（神经传导物质）的释放。这些神经传导物质漂流满本神经元的轴突和接收信息的神经元的树突之间



- 在接受Action Potential的神经元的synapse中，会有通道是被激活的，然后这些离子会被pump进接受信息的树突，然后上面的过程会在下一个神经元循环。



- 每一个生成的动作电位都是一样的，所以真正matter的是：***动作电位发送的频率***
    - The **maximum firing rate is about 200 per second**.
- 带电的物质无法通过由lipids（油脂）组成的细胞壁， 他们必须通过channel proteins（通道蛋白：**通道蛋白**是一类跨越细胞膜磷脂双分子层的蛋白质）that extend through the membrane.
- 虽然对于某个特定的发出信息的神经元来说，它发出的信息都是一样的，**<u>不过对于不同的接受者来说，这“一样”的信息会被不一样的接受</u>**。这些信息或许会提高原有的potential（amplified），或许会降低原有的potential（de-amplified），甚至有可能被negated。这些都决定于各种不一样的synapses。
- 这个”一样的信息“被不一样的解释的这一现象**is modelled by assigning each synapse a number (called a <u>weight</u>, or synaptic strength) and *multiplying it by the signal* to determine the resulting local potential at the post-synaptic neuron.**
- **Excitatory neurotransmitters** have excitatory effects on the neuron. This means they increase the likelihood that the neuron will fire an action potential.
- **Inhibitory neurotransmitters** have inhibitory effects on the neuron. This means they decrease the likelihood that the neuron will fire an action.
- There are some **"veto" neurons** that have the overwhelming power of neutralizing the effect of a large number of excitatory inputs to a neuron.
- Some **indirect self-excitation** also occurs when one node's activation excites its neighbour, which excites the first node again. 就是一些神经元会激活他邻近的神经元，这些邻近的神经元再次发送信号给这个神经元来激活他

Summary

- Incoming signals create potentials at the synapse sites of a neuron.
- Each of these potentials influences (through slow potential) the potential at the base of the axon. The sum of all these influences is called the **activation**.
- When activation exceeds some **threshold, action potentials** are generated, and propagate down the axon.
- The same signal transmitted to all of recipient neurons, can cause different effects on the recipient neurons depending on type and ions at the synapses and the neurotransmitters.

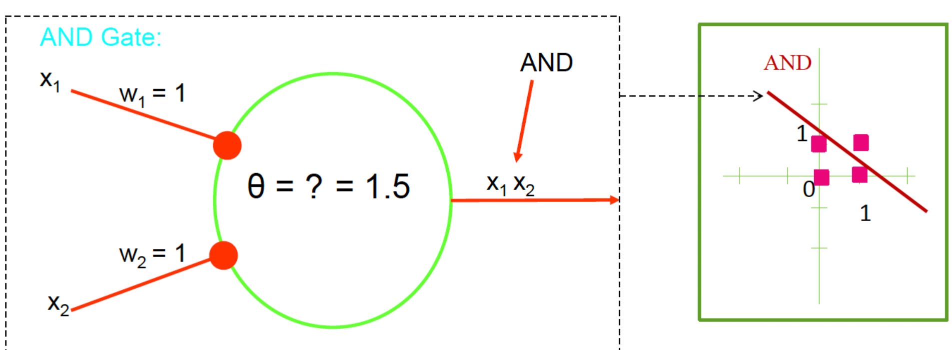
Learning in Artificial Networks and Linear Separability

- **Classification** - Grouping and recognizing the label attached to the group
  - Label is known
  - Identify letters as 'A' and 'B' which denote class labels
- **Clustering** - Grouping based on distinctive and common features
  - Label is not known
  - Identify both as letters but different based on shape, name of the letter is not known, define central properties.
- 上面这两个方式有着本质上的区别，第一种问题是分类问题，他是把东西归类到已知的版块，而第二种问题则是聚类问题，他是根据特征把不同的东西分开，label是不知道的

Threshold Logic Units(TLUs)

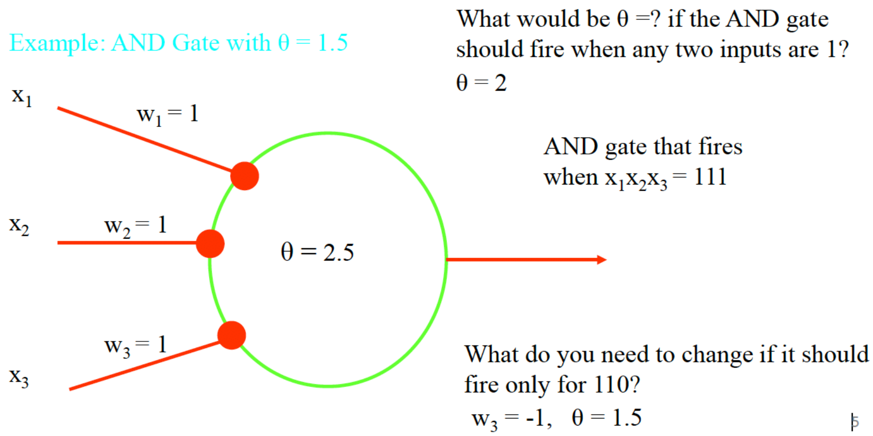
- *TLUs CANNOT* realize functions that are NOT**linearly separable**
- TLU有点像是threshold neuron，只不过他接受的是二进制的输入（0，1）
- ***AND Gate***

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1



  - 所以把它想像成一个threshold neuron的话，那么我们需要决定的就是阈值的大小，为了让当x_1=1和x_2=1并且两个的权重都是1的情况下才被激活，他的阈值可以设定在1-2之间，因为如果设定在0-1之间的话，只需要一个x就可以激活，如果设定在2以上的话，怎么样都无法激活
  - 很巧妙的，如果有图像来表示的话，每个点的坐标由（x_1, x_2）来表示，可以看到正好有一条分割线把这些点隔开了

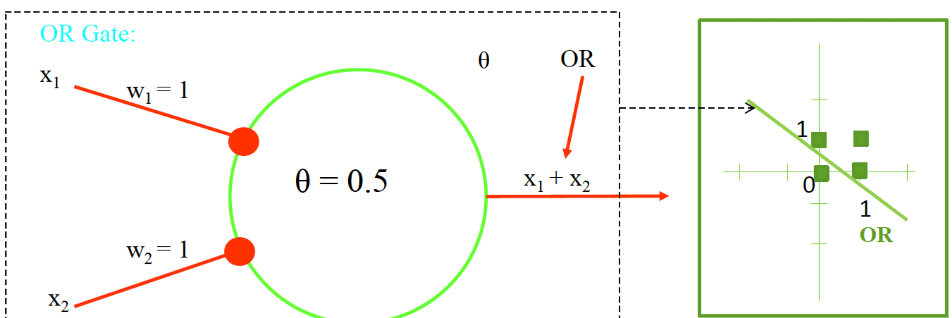
- ***AND Gate > 2 inputs***



- 如上图所示，对于不同的要求，都可以通过调整权重以及阈值的大小来完成目标

- ***OR Gate***

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1



Input space

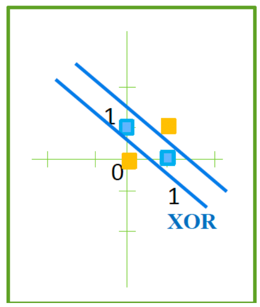
- 和上面的一样，因为or gate的意思是，只需要一个input是1那么他的neuron就可以被激活，所以在权重都是1的情况下，把阈值设定在0-1是唯一的选择

- ***Linearly Separable***：A function f:{0, 1}^{n}→{0, 1} is linearly separable if the space of input vectors yielding 1 can be separated from those yielding 0 by a linear surface having (hyperplane) *n-1*dimensions. 就是说对于一个f:{0, 1}^{n}→{0, 1}这样子的函数来说，线性可分割指的就是在映射为1的数据和映射为0的数据中正好存在一个属于n - 1维度的平面把他们分开

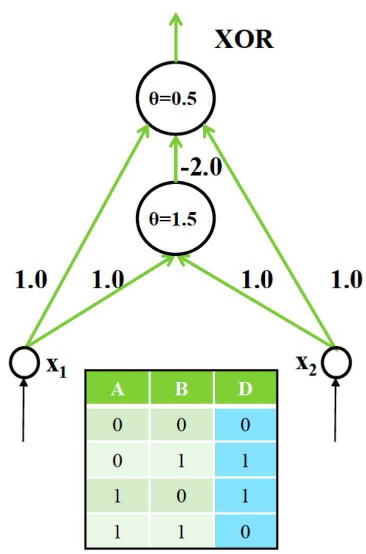
- ***XOR Gate***

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

- 把他的图表画出来是这个样子的：

- 对于这些点来说，并不存在一条线把他们分开，所以他们是线性不可分，所以用TLU logic根本表示不出来，所以就得用多个layer，即除了input layer和output layer之外，还要有一个third layer，如图所示：

- **像这样三个layer的logic gate的话是有能力表示全部的逻辑的**

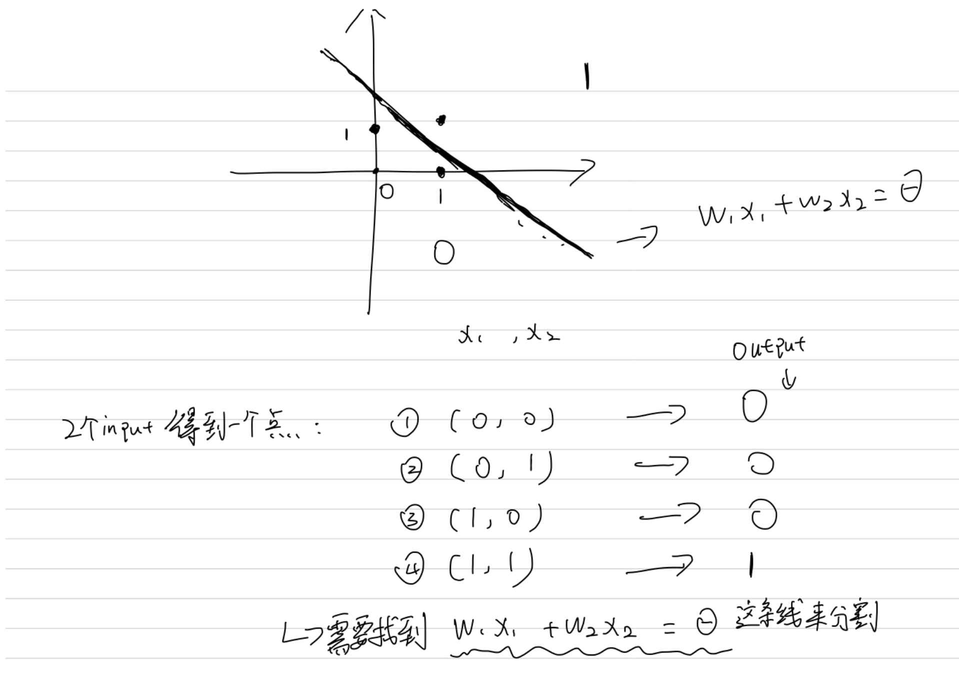
* 一些历史：
    - In 1969, ***Marvin Minskyand Seymour Papert***, two “PSS” researchers at MIT studied the ANNs and revealed that a two-layered (input and output) network cannot handle all logical relations –specifically the XOR.
        - It implies that ANNslacked the power of a Turing machine.
    - Federal funding for ANNs immediately stopped
    - ***David Rumelhartand Jim McClelland*** developed ***Parallel Distributed Processing (PDP)***.

- ***Linear Separability as Functional Mapping***:
    - 首先，先有一个比较基础的output function:

      $if \sum_{i=1}^{n} w_i x_i \geq \theta$: $f(x_1, x_2, \ldots, x_n) = 1$

      $if \sum_{i=1}^{n} w_i x_i < \theta$: $f(x_1, x_2, \ldots, x_n) = 0$

      也就是说这个函数是$\mathbb{R}^n \rightarrow 0, 1$，他的output只有0和1
    - 想象一下，假设此时只有 x_1, x_2，那么此时，如果$w_1 x_1 + w_2 x_2 \geq \theta$的话，那么函数的结果会是1，反之为0，所以可以看出，当$w_1 x_1 + w_2 x_2 = \theta$的这条线，就是这个把这个函数的的点都给分开的一个separate line，拿之前的and gate来举个例子：

      如果把x_2看作是y，那么可以写成$y = mx + c$的形式来plot：

      $w_1 x_1 + w_2 x_2 = \theta$

      $w_2 x_2 = \theta - w_1 x_1$

      $x_2 = \frac{\theta}{w_2} - \frac{w_1}{w_2} x_1$

      $x_2 = (\frac{-w1}{w_2}) + \frac{\theta}{w_2}$
    - 上面最后一行就是ppt给出的线的表达式
    - 通过改变weight的值和θ的值，可以得到二维平面上的直线
    - θ被称为bias
- 所以***Training a network means finding the best fitting values of θ and the weights to categorize a given set of values***
- 二维平面可以被一条直线分割，三维平面可以被一个二维的平面分割，所以：***An n-dimensional input space can be divided by an (n - 1) -dimensional plane or hyperplane***

- The net input signal is the sum of all inputs:

$$\text{net}_i(t) = \sum_{j=1}^{n} w_{i,j}(t)x_j(t)$$

This can be viewed as computing the **inner product** of the **vectors** $\mathbf{w_i}$ and $\mathbf{x}$: https://en.wikipedia.org/wiki/Dot_product

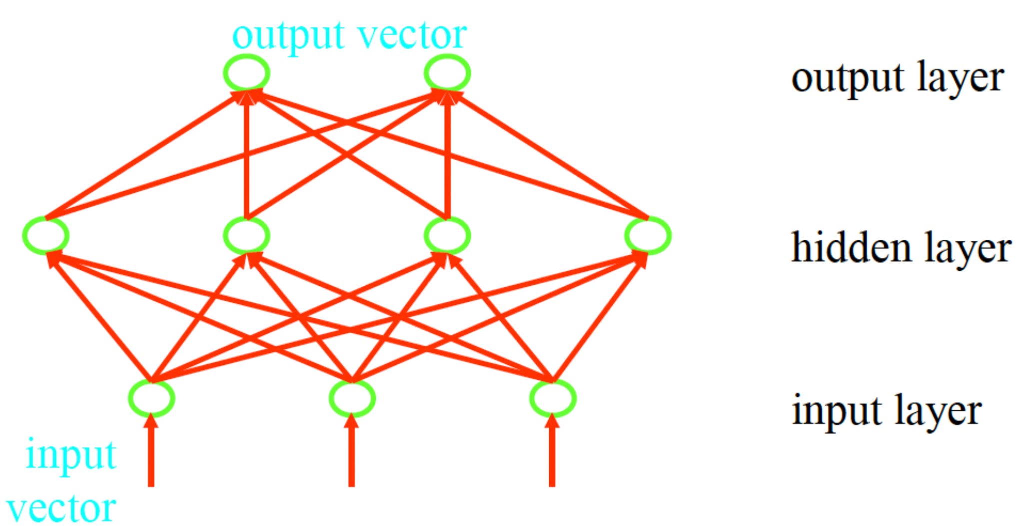
$$\text{net}_i(t) = \| w_i(t) \| \cdot \| x(t) \| \cdot \cos\alpha$$

where α is the **angle** between the two vectors and

$||a|| = \text{sqrt}(a.a)$

17

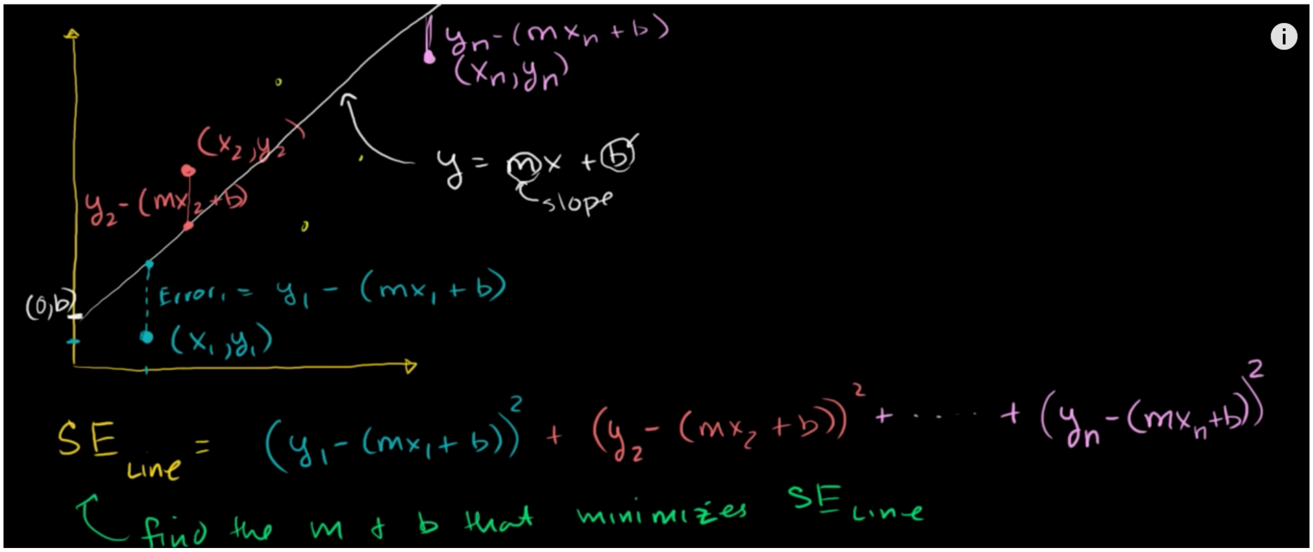
- 可以使用**multiple artificial neurons in multiple layeers to form networks with increased capabilities**
- 在一个layer里面可以有多个neuron，假设拿到一个二维数据，那么每个neuron都可以看成是一条分割线
- 一些Terminology：
  - $f: \mathbb{R}^3 \rightarrow (0,1)^2$: **means that the network takes 3-dimensional real values as input and generates two dimensional binary output values.**



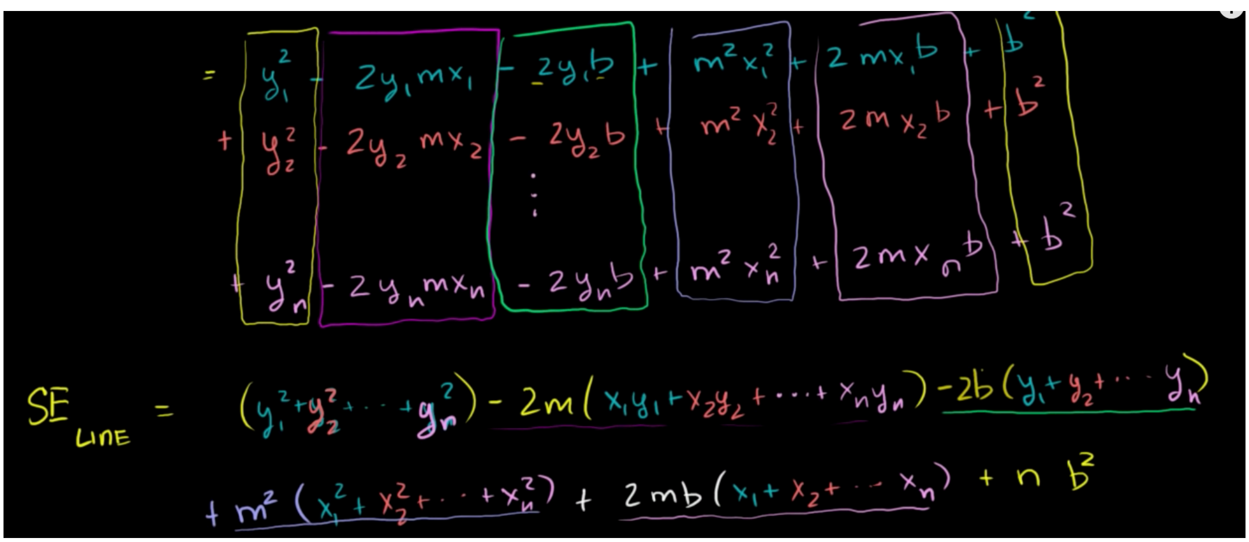
- Arrows indicate the direction of data flow.
- The *input layer*just contains the input vector and*does not perform any computations other than distributing inputs to the next layers (used optionally)*.
- The intermediate layers, termed *hidden layers*, receives input from the input layer or previous hidden layers and *sends output to the final output layer*.
- After applying their *activation function*, the neurons in the final *output layer* generate the output vector.

Adaline

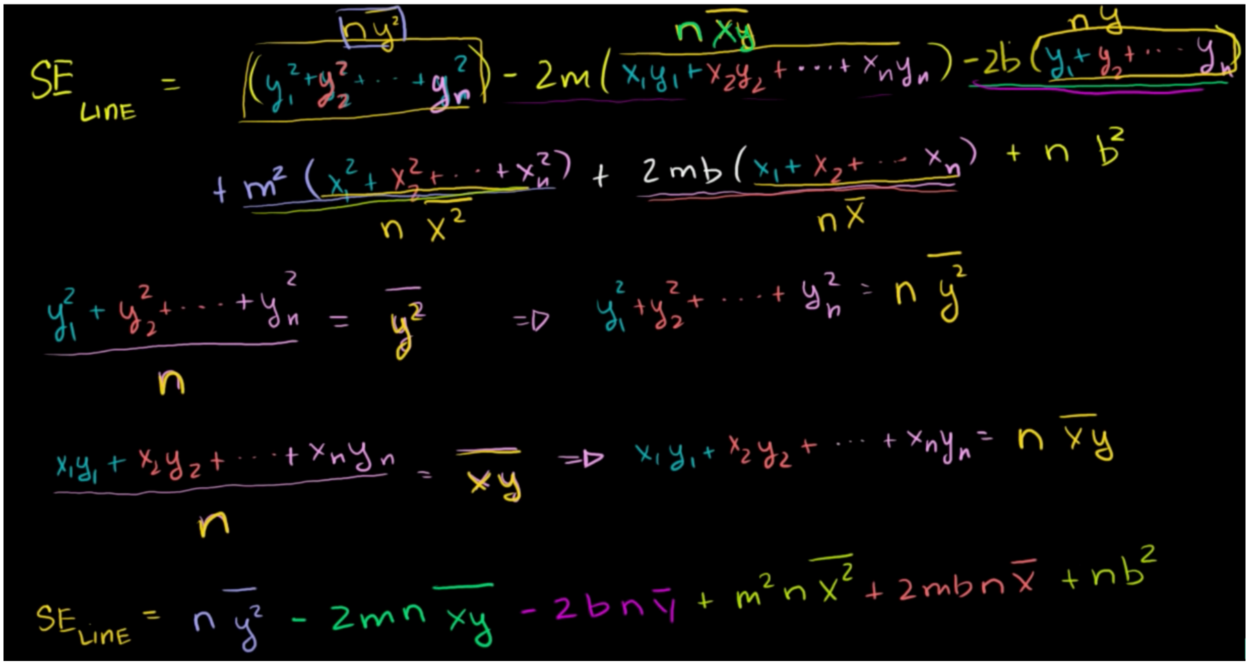
Mathematical Point of View:



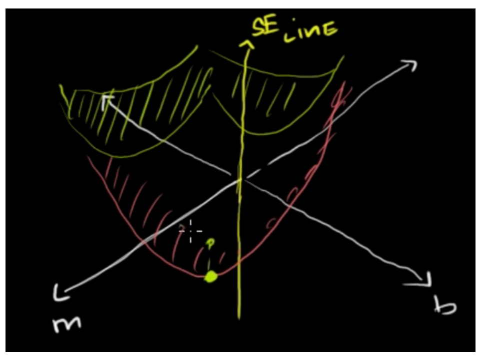
- 假设有许多二维定义的点:$\{x_n, y_n\}$，在这些点之间存在一条直线$y = mx + b$，这条是希望求得的线，他的作用是：minimize the squared distances to these different points
- 对于每个点来说，"Error"被定义成这个点到这条线的距离，比如如果是对于点x_1，那么他的error就是：$y_1 - (m * x_1 + b)$这个的值，所以他的***Squared Error***就是$(y_1 - (m * x_1 + b))^2$，当我们把这些点每个的Squared Error都取下来的话，我们会得到$\sum_{i=0}^{n} SquaredError_i$，**然后现在要找到的就是一个适当的m和b值来minimize这个Squared Error的sum的值**



- 经过拆分+合并同类项之后，SE_{line}的这个公式可以被拆解成这个样子



- 之后注意在每个括号内的式子，都是同样的式子加n遍，所以如图所示很容易得出他们的平均数，把平均数代入公式，就可以得到一个更加简洁的公式，在得到了最 下面这个式子之后，就可以进行optimize（找到能把这个表达式minimize的m和b）



- **Error Surface**的概念，有了上面的SE_{line}的这个式子之后，其实可以放一放。想象一个三维平面，其中m-axis和b-axis如图所示，SE_{line}也自成一个axis，在这个式子中，放入不同的m值和b值，可以得到不同的SE的值，如果把m和b所有的可能都求得，那么output成的SE的值可以组成一个平面，这个就是error surface，其实简单的来说就是把这个式子可视化之后他就是一个平面就是error surface。
- 那么这个时候希望做到的是找到m和b的值去找到这个平面的极小值，下面是关于如何利用导数求极值的笔记：

1、先求一次导数，这个一次导数，全名叫一次导函数(first derivative, 或 first differentiation)；

2、令一次导函数为0，解出来的x，称为静态点(stationary point)；

3、继续对一次导函数求导，求出来的是二次导函数。

将刚才的静态点的x，代入到二次导函数中，

如果大于零，刚才的静态点为极小值点；

如果小于零，刚才的静态点为极大值点；

如果等于零，刚才的静态点既非极大值点，也非极小值点，称为拐点，

拐点 = POI = Point of Inflexion = 图像上凹下凹的转折点。

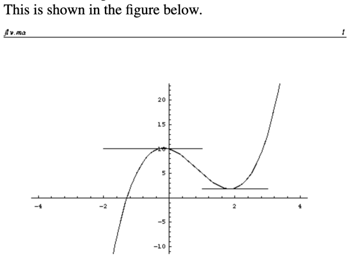
4、将静态点的坐标代入到原函数，就得到了最大或最小值。

- 这个只是one variable的，那么当有two variable，甚至multiple variable的时候：

Maxima and Minima of Functions of Two Variables

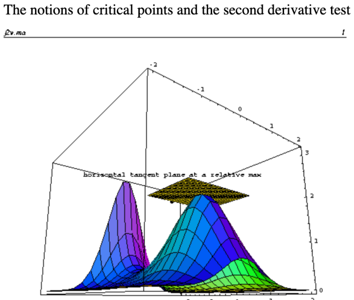
The problem of determining the maximum or minimum of function is encountered in geometry, mechanics, physics, and other fields, and was one of the motivating factors in the development of the calculus in the seventeenth century.

Let us recall the procedure for the case of a function of one variable y=f(x). First, we determine points x_c where f'(x)=0. These points are called critical points. At critical points the tangent line is horizontal. This is shown in the figure below.



The second derivative test is employed to determine if a critical point is a relative maximum or a relative minimum. If f''(x_c)>0, then x_c is a relative minimum. If f''(x_c)<0, then x_c is a maximum. If f''(x_c)=0, then the test gives no information.

The notions of critical points and the second derivative test carry over to functions of two variables. Let z=f(x,y). Critical points are points in the xy-plane where the tangent plane is horizontal.



Since the normal vector of the tangent plane at (x,y) is given by

$$f_x(x,y)\mathbf{i} + f_y(x,y)\mathbf{j} - \mathbf{k}$$

The tangent plane is horizontal if its normal vector points in the z direction. Hence, *critical points are solutions of the equations*:

$$f_x(x,y) = 0 \quad \textit{and} \quad f_y(x,y) = 0$$

because horizontal planes have normal vector parallel to z-axis. The two equations above must be solved *simultaneously*.

Example

Let us find the critical points of

$$z = f(x,y) = \exp\left(-\frac{1}{3}x^3 + x - y^2\right)$$

The partial derivatives are

$$f_x(x,y) = (-x^2+1)\exp\left(-\frac{1}{3}x^3 + x - y^2\right)$$

$$f_y(x,y) = -2y\exp\left(-\frac{1}{3}x^3 + x - y^2\right)$$

f_x=0 if 1-x^2=0 or the exponential term is 0. f_y=0 if -2y=0 or the exponential term is 0. The exponential term is not 0 except in the degenerate case. Hence we require 1-x^2=0 and -2y=0, implying x=1 or x=-1 and y=0. There are two critical points (-1,0) and (1,0).

The Second Derivative Test for Functions of Two Variables

How can we determine if the critical points found above are relative maxima or minima? We apply a second derivative test for functions of two variables.

Let (x_c,y_c) be a critical point and define

$$D(x_c,y_c) = f_{xx}(x_c,y_c)f_{yy}(x_c,y_c) - [f_{xy}(x_c,y_c)]^2.$$

We have the following cases:

- If D>0 and f_xx(x_c,y_c)<0, then f(x,y) has a relative maximum at (x_c,y_c).
- If D>0 and f_xx(x_c,y_c)>0, then f(x,y) has a relative minimum at (x_c,y_c).
- If D<0, then f(x,y) has a saddle point at (x_c,y_c).
- If D=0, the second derivative test is inconclusive.

An example of a saddle point is shown in the example below.

Example: Continued

For the example above, we have

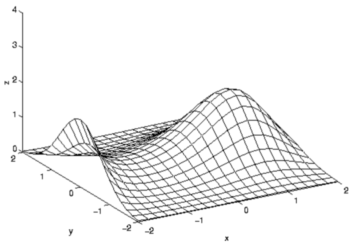
$$f_{xx}(x,y) = (-2x + (1-x^2)^2) exp\left(-\frac{1}{3}x^3 + x - y^2\right),$$

$$f_{yy}(x,y) = (-2 + 4y^2) exp\left(-\frac{1}{3}x^3 + x - y^2\right),$$

$$f_{xy}(x,y) = -2y(1-x^2) exp\left(-\frac{1}{3}x^3 + x - y^2\right),$$

For x=1 and y=0, we have D(1,0)=4exp(4/3)>0 with f_xx(1,0)=-2exp(2/3)<0. Hence, (1,0) is a relative maximum. For x=-1 and y=0, we have D(-1,0)=-4exp(-4/3)<0. Hence, (-1,0) is a saddle point.

The figure below plots the surface z=f(x,y).



Notice the relative maximum at (x=1,y=0). (x=-1,y=0) is a relative maximum if one travels in the y direction and a relative minimum if one travels in the x-direction. Near (-1,0) the surface looks like a saddle, hence the name.

Maxima and Minima in a Bounded Region

Suppose that our goal is to find the global maximum and minimum of our model function above in the square -2<=x<=2 and -2<=y<=2? There are three types of points that can potentially be global maxima or minima:

1. Relative extrema in the interior of the square.
2. Relative extrema on the boundary of the square.
3. Corner Points.

We have already done step 1. There are extrema at (1,0) and (-1,0). The boundary of square consists of 4 parts. Side 1 is y=-2 and -2<=x<=2. On this side, we have

$$z = f(x,-2) = exp\left(-\frac{1}{3}x^3 + x - (-2)^2\right) = g(x)$$

The original function of 2 variables is now a function of x only. We set g'(x)=0 to determine relative extrema on Side 1. It can be shown that x=1 and x=-1 are the relative extrema. Since y=-2, the relative extrema on Side 1 are at (1,-2) and (-1,-2).

On Side 2 (x=-2 and -2<=y<=2)

$$z = f(-2,y) = exp\left(-\frac{1}{3}(-2)^3 - 2 - y^2\right) = h(y).$$

We set h'(y)=0 to determine the relative extrema. It can be shown that y=0 is the only critical point, corresponding to (-2,0).

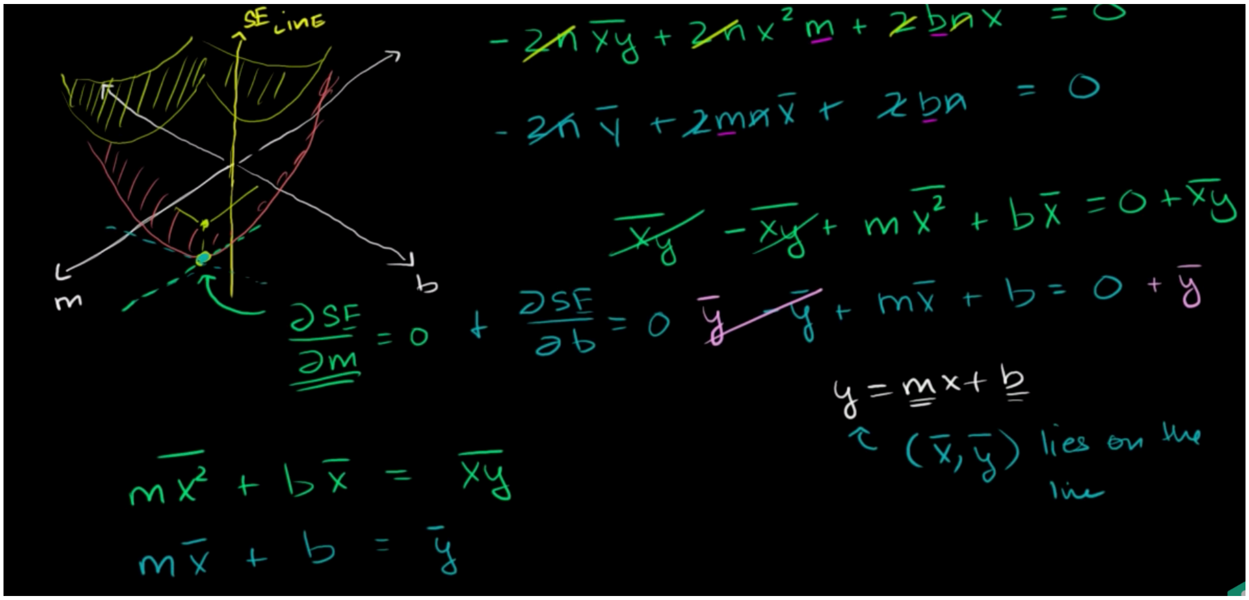
We play the same game to determine the relative extrema on the other 2 sides. It can be shown that they are (2,0), (1,2), and (-1,2).

Finally, we must include the 4 corners (-2,-2), (-2,2), (2,-2), and (2,2). In summary, the candidates for global maximum and minimum are (-1,0), (1,0), (1,-2), (-1,-2), (-2,0), (2,0), (1,2), (-1,2), (-2,-2), (-2,2), (2,-2), and (2,2). We evaluate f(x,y) at each of these points to determine the global max and min in the square. The global maximum occurs (-2,0) and (1,0). This can be seen in the figure above. The global minimum occurs at 4 points: (-1,2), (-1,-2), (2,2), and (2,-2).

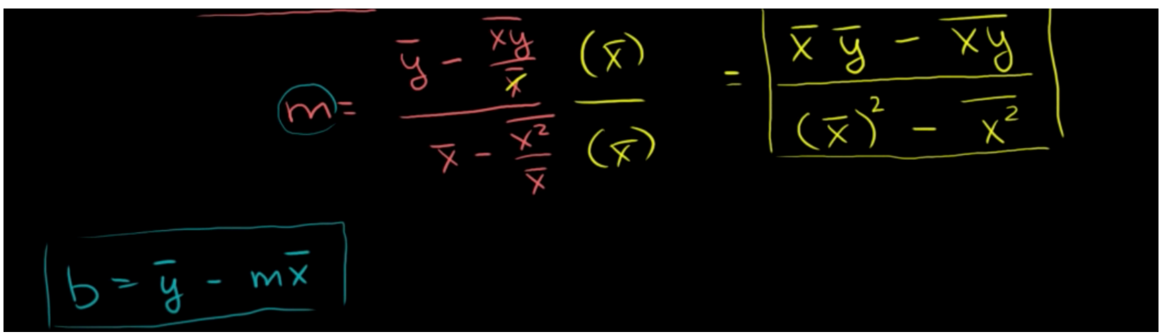
Example: Maxima and Minima in a Disk

Another example of a bounded region is the disk of radius 2 centered at the origin. We proceed as in the previous example, determining in the 3 classes above. (1,0) and (-1,0) lie in the interior of the disk.

- 所以可以看到，为了求出一个平面的极值，需要通过求出这个平面的导数（分别对应不同的variable）来找出
- Partial Derivative is just like taking regular derivative, just assume everything but the variable you are doing the partial derivative is constant



- 在做partial derivative的时候又发现了一个很有趣的现象：最下面那个$m * \bar{x} + b = \bar{y}$，也就是说：在那个最fit的线里面，存在一个点$(\bar{x}, \bar{y})$, $m\bar{x^2} + b\bar{x} = \bar{xy}$把这个拆成$mx + b$的形式之后也会有另外一个点：$(\frac{\bar{x^2}}{\bar{x}}, \frac{\bar{xy}}{\bar{x}})$, 有了这两个点之后事实上就已经足够解出$y = mx + b$了



- 上面这个例子实际上是optimize一个linear regression的例子

Adaline

- 和上面的数学模型不同的是，在人工智能的模型里面，error被简单的定义成$e = d - y$
- 对于p个data point，这个mean squared error是:

$$MSE = \frac{1}{p} * \sum^{p} (d - y)^2$$

- 在这里讨论的前提是：

$$y = f(net) = a = \sum_{i=1}^{n} x_i w_i$$

  也就是说是一个linear output function

- 和上面的步骤一样，为了最小化error，就要求这个的导数，找到最适合的weight和bias unit

Some history

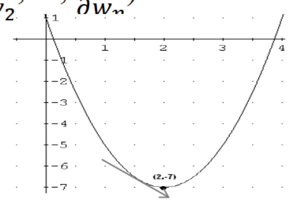
- An **Adaptive Linear Element** tries to minimize the total **amount of error** instead of aiming to reduce the number of misclassifications.
- The Adaline, proposed by Widrow(1959,1960), modifies weight in such a way to diminish the mean square error (MSE) at every data point.
- The training examples are of form (x, d) where $d \in R$
- Output $y \in R$,the network outputs the activation *a*.Uses *linear output function*. y=a

Learning Using Gradient Descent

- Adaline uses ***gradient descent***(梯度下降法)
- Gradient(梯度)：**在微积分里面，对多元函数的参数求∂偏导数，把求得的各个参数的偏导数以向量的形式写出来，就是梯度。比如函数f(x,y), 分别对x,y求偏导数，求得的梯度向量就是(∂f/∂x, ∂f/∂y)T,简称grad f(x,y)或者▽f(x,y)。**对于在点(x0,y0)的具体梯度向量就是(∂f/∂x0, ∂f/∂y0)T.或者▽f(x0,y0)，如果是3个参数的向量梯度，就是(∂f/∂x, ∂f/∂y，∂f/∂z)T,以此类推。 If *f*(*w*1, ..., wn) is a *differentiable, scalar-valued output functionofseveral variables*, its **gradient**is the *vector* whose components are the *n*partial derivatives of *f*.
- 梯度的意义：

$$\text{Gradient is a vector } V = \frac{\partial f}{\partial W} = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \ldots, \frac{\partial f}{\partial w_n}\right)$$

Gradient represents the **slope of the tangent of the graph function *f*()** and points in the direction of the greatest rate of increase of the value of *f*().



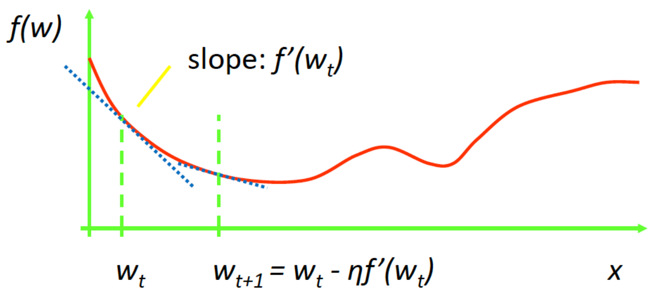
f(x) = 2 x ² - 8 x + 1

  - 他的意义从几何意义上讲，就是函数变化增加最快的地方。具体来说，对于函数f(x,y),在点(x0,y0)，沿着梯度向量的方向就是(∂f/∂x0, ∂f/∂y0)T的方向是f(x,y)增加最快的地方。或者说，***沿着梯度向量的方向***，更加容易找到函数的最大值。反过来说，沿着梯度向量相反的方向，也就是-(∂f/∂x0, ∂f/∂y0)T的方向，梯度减少最快，也就是更加容易找到函数的最小值。
  - 比如我们在一座大山上的某处位置，由于我们不知道怎么下山，于是决定走一步算一步，也就是在每走到一个位置的时候，求解当前位置的梯度，沿着梯度的负方向，也就是当前最陡峭的位置向下走一步，然后继续求解当前位置梯度，向这一步所在位置沿着最陡峭最易下山的位置走一步。这样一步步的走下去，一直走到觉得我们已经到了山脚。当然这样走下去，有可能我们不能走到山脚，而是到了某一个局部的山峰低处。 从上面的解释可以看出，梯度下降不一定能够找到全局的最优解，有可能是一个局部最优解。当然，如果损失函数是凸函数，梯度下降法得到的解就一定是全局最优解。
- Grandient Descent
  - 其实这个很好理解，首先Squared Error这个式子是根据desired value和actual output得到的，actual output是根据output function得到的，而output function里面放进去的值是net input，这是根据$\sum_i x_i w_i$得到的，也就是说现在squared error这个式子相当于一个error surface，旨在找到一个最优的weight，来另square value达到最低点
  - **Gradient descent** is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, **one takes steps proportional** to the *negative*of the gradient at the current point.
- Gradient Descent VS 另导数等于0

- 对于另导数等于0来说，首先是运算量非常大，并且对于多个variable的function也会求得多个critical value，总之就是运算量会很大，并且可能会有的函数求不出导数等于0
- 所以实际上，用Gradient Descent来慢慢逼近最优这个结果最大的好处是：如果目前的weight得到的结果离目标的结果比较远的话，那么这个slope会比较大，也就是说对于这个weight来说，调整的幅度会很大，但是，当weight慢慢逼近最优（就是接近error surface的最低点）的时候，他就会慢慢的调整，并且对于一些导数等于0计算不出/有许多的critical value的情况能够有所应对

- 所以其实在ppt里面讲到的东西，讲来讲去都是在讲具体怎么调整权重的值，而在Adaline里面，权重的调整方式就是**负梯度**，而梯度里面的偏导数是基于MSE得到的

- Finding the absolute minimum of a one-dimensional error function *f(x)*:



Repeat this iteratively until for some x_i, $f'(x_i)$ is sufficiently close to 0.

- We have to apply the *chain rule* to solve the derivative because

$E = f(e)$	given $E = e^2$
$e = f(y)$	given $e = (d - y)$
$y = f(a)$	function of activation
$a = f(w)$	given $a = \sum_{i=1}^{n} x_i w_i$

when we are computing partial derivative w.r.t. weight

$$\frac{\partial E}{\partial w_k} = \frac{\partial E}{\partial e} \cdot \frac{\partial e}{\partial y} \cdot \frac{\partial y}{\partial a} \cdot \frac{\partial a}{\partial w_k}$$

- So, considering *n* dimensional input vector and $E = e^2 = (d - a)^2$ and $a = \sum_{i=1}^{n} x_i w_i$, w_k should be changed in the direction of –ve gradient.
- Applying the *chain rule* to solve the derivative,

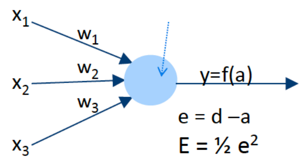
$$-\frac{\partial E}{\partial w} = -\left(\frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n}\right)$$

$$-\frac{\partial E}{\partial w_k} = -\frac{\partial E}{\partial a} \cdot \frac{\partial a}{\partial w_k}$$

$$= c(d - a).(-1).\frac{\partial(w_1 x_1 + \cdots + w_k x_k + \cdots + w_n x_n)}{\partial w_k}$$

$$= c(d - a).x_k$$

c = a small +ve constant learning rate

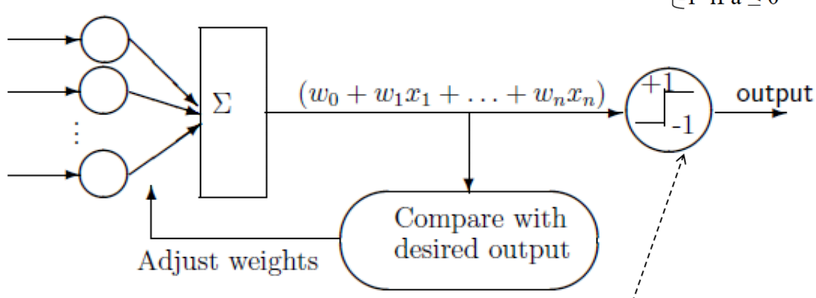


- Chain Rule：

$$D\{f(g(x))\} = f'(g(x))\, g'(x)\,.$$

- 其实对于Adaline来说，整个学习的过程都是差不多的

$$x_j = \begin{cases} +1 & \text{if } a > 0 \\ -1 & \text{if } a \leq 0 \end{cases}$$



See example 2.6 in the book. You will see that the solution applies correction only when a data point is incorrectly classified (error> threshold) to reduce the computation.

An output function can be used to generate custom output after training

- 如果有不止一个output的话，可以使用多个neuron with multiple weights，这样的方法在Assignment中也有用到

Validation of the ANN Model

- True Positive(真正，TP)：将正类预测为正类数
- True Negative(真负，TN)：将负类预测为负类数
- False Positive(假正，FP)：将负类预测为正类数误报 (Type I error)(有多少个这个东西)
- False Negative(假负，FN)：将正类预测为负类数→漏报 (Type II error)

1、准确率（Accuracy）

准确率(accuracy)计算公式为：

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

注：准确率是我们最常见的评价指标，而且很容易理解，就是被分对的样本数除以所有的样本数，通常来说，正确率越高，分类器越好。准确率确实是一个很好很直观的评价指标，但是有时候准确率高并不能代表一个算法就好。比如某个地区某天地震的预测，假设我们有一堆的特征作为地震分类的属性，类别只有两个：0：不发生地震、1：发生地震。一个不加思考的分类器，对每一个测试用例都将类别划分为0，那那么它就可能达到99%的准确率，但真的地震来临时，这个分类器毫无察觉，这个分类带来的损失是巨大的。为什么99%的准确率的分类器却不是我们想要的，因为这里数据分布不均衡，类别1的数据太少，完全错分类别1依然可以达到很高的准确率却忽视了我们关注的东西。再举个例子说明下。在正负样本不平衡的情况下，准确率这个评价指标有很大的缺陷。比如在互联网广告里面，点击的数量是很少的，一般只有千分之几，如果用acc，即使全部预测成负类（不点击）acc也有 99% 以上，没有意义。因此，单纯靠准确率来评价一个算法模型是远远不够科学全面的。

2、错误率（Error rate）

错误率则与准确率相反，描述被分类器错分的比例，error rate = (FP+FN)/(TP+TN+FP+FN)，对某一个实例来说，分对与分错是互斥事件，所以accuracy =1 - error rate。

3、灵敏度（sensitive）

sensitive = TP/P，表示的是所有正例中被分对的比例，衡量了分类器对正例的识别能力。

4、特效度（sensitive）

specificity = TN/N，表示的是所有负例中被分对的比例，衡量了分类器对负例的识别能力。

5、精确率、精度（Precision）

精确率(precision)定义为：

$$P = \frac{TP}{TP + FP}$$

表示被分为正例的示例中实际为正例的比例。

6、召回率（recall）

召回率是覆盖面的度量，度量有多个正例被分为正例，recall=TP/(TP+FN)=TP/P=sensitive，可以看到召回率与灵敏度是一样的。

F-measure

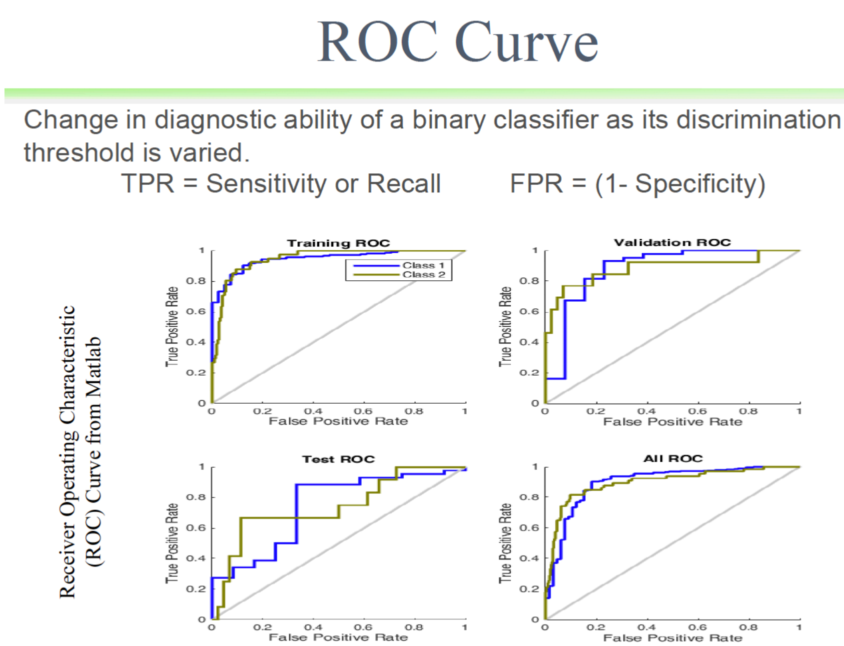
- Also called harmonic mean of precision and recall, the traditional F-measure or balanced F-score:

$$\frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Confusion matrix：

n = 165	Predicted: No	Predicted: Yes
Actual: No	50	10
Actual: Yes	5	100

- 就是一个类似这样子的表格



Perception

- Some History:
  - ***McCulloch and Pitts (1943)*** gave the ***first mathematical model of a single neuron***.
    - 在这个模型里面，连接和weight都是静态的
    - 只有一个layer，所以不能implement XOR
  - Hebb's learning rule (1949): For each input pattern presentation, increase connection weight between nodes *i*and *j* if both nodes are simultaneously ON or OFF.
    - Activation of *j* always causes an activation of *i* where w_{ji} is the weight associated with connection from *j* to *i* and x_i an x_j dare inputs to *i* and *j* respectively.
    - The strength of connections between neurons eventually comes to represent the correlations between their outputs:

$$\Delta w_{ji} = c * x_i x_j$$

  - ***Rosenblatt's "perceptrons" (1958)***: If the output is unsatisfactory, modify each weight by a quantity that is likely to improve network performance，并且还提出了supervised learning的概念，也就是正确的output是知道的，需要通过这个正确的output去整理现有的weight和θ
  - ***Widrowand Hoff's learning rule (1960, 1962)*** was also based on *gradient descent*.
  - Then ***back-propagation algorithms*** were proposed for training MULTI-LAYER networks.

Perceptron

- **Perceptron只能分两个Class！！！**
- 在***Perceptron***，learning有两种方式：
  1. 简单的**Simple Feedback Learning**：根据desired value来判断当前的threshold/weight是不是符合的，从而根据那个来调整现有的weight和threshold
  2. **Error Correction Learning**：Use an error measure to adapt the weight vector
  - 其实上面这两种学习的方式本质上是并没有什么区别的，对于Supervised Learning来说，每个一组input和一组权重会通过某种function来得到一个output，那么这个output又会对应一个desired output，如果desired output不等于现在的output，那么说明现在的权重需要被调整，Simple Feedback Learning和Error Correction Learning的区别就在于权重被调整的方式：
    - **Simple Feedback Learning**:
      - 当desired output > output: $\Delta w = c * x$
        - $w_{new} = w_{old} + \Delta w$
      - 当desired output < output: $\Delta = (-1) * c * x$(所以在这里增量变成了一个负数，所以权重会降低来把结果降低)
        - $w_{new} = w_{old} + \Delta w$
    - **Error Correction Learning**:
      - 在这种方式里面根本不需要判断output是大于还是小于output：

        - $\Delta w = (d - y) * c * x$, $w_{new} = w_{old} + \Delta$
        - 在这种方式里面，每一次循环都会加上Δw因为后面$(d - y)$的存在会让每次的调整灵活应对是正增量还是负增量
    - 使用x_i来作为一个调整的值是因为：***Use input value in calculation because if input value is high, error will be high and vice versa***

- ***Features and Functionality(Two layer network)***:
    - Two layer network
    - Applies feedforward processing - all connections go to the next layer
    - Initially w_i are **assigned random values** which results in poor initial performance(high error)
    - To improve performance, network is trained to adjust the weight values $\rightarrow$ network learns
- 因为不仅仅是要调整weight，也要同时调整θ，所以把θ放入input里面一并调整

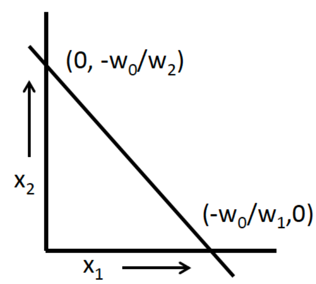
  $\sum_{i=1}^{n} x_i w_i - \theta = 0$这个是那个分割线，那么把$\theta$换成$\sum_{i=1}^{n} x_i w_i - x_0 w_0 = 0$，就可以让他一起调整$\theta$的值，在这种情况下，$x_0$必定等于1，$w_0 = -\theta$

  所以现在的分割线的函数是：$\sum_{i=0}^{n} x_i w_i = 0$
- 对于一个二维的平面，***a neuron will represent a straight line***：

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

- 以上这条线，也可以这样表示：

$$x_2 = (-\frac{w_1}{w_2})x_1 - \frac{w_0}{w_2}$$

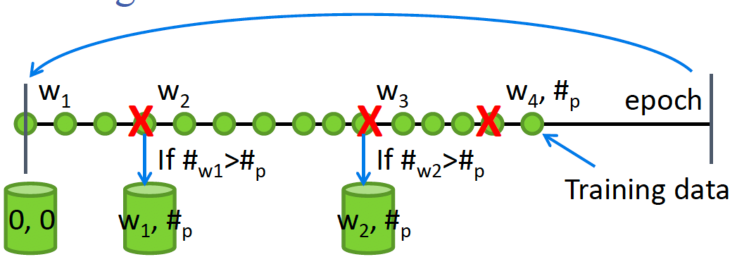


- 也就是说，evaluate每个点的output和desired output的差距/比较，从而得出需要进行多少调整
- ***<u>Perceptron Convergence Theorem</u>: It can be guaranteed that the Perceptron training algorithm will classify all the data correctly when they are linearly separable and cis sufficiently small.***
- ***Learning Rate***：这个值决定了每一次的增量的大小，如果c的值太小了，那么每一次都会对weight作出比较小的改变，在这种情况下，可能需要很长的时间去训练，如果c的值太大了，那么这条线就会晃来晃去
    - common choice is c = 1
- ***Terminating Condition***
    - 一直循环直到所有的weight都被调整好
    - 假设这组数据是一个linearly not separable的data的话，那么就会进入一个无限的loop里面，这样子的话，需要设置一个***maximum number of iteration***

- Until accepatable error level is reached: ***Error = (misclassified data points / total data points) <= threshold***
- 或者可以结合一下上面几种结束条件一起进行

The Pocket Algorithm

- The pocket algorithm is a useful modification of the perceptron training algorithm
  - Weight change mechanism is the same.
  - Identifies the weight vector ($w_1,\ldots,w_n$) with the **longest unchanged run** (# of points correctly classified: $\#_p$) as the best solution so far.
  - Stores *the best weight vector* in a "pocket" as well as *the best length of the run* associated with it.
  - Pocket contents are replaced with a new weight vector when a longer successful run is found.



Pocket Algorithm with *Ratchet*

- A lucky run of several successes may allow a poor solution to replace a better solution in the pocket.
- To avoid this, the Pocket algorithm with ratchet ensures that the pocket weights always "ratchet up"
  - w^1 in the pocket is replaced by w^2 that has longer successful run **only after testing on all training samples** whether w^2 does correctly classify a greater number of samples than w^1.
  - Expensive computation.

- 对于Pocket algorithm的话，以下这个伪代码相当的useful:

INPUT: Training Examples $\{E^k, C^k\}$. E^k is a vector with $E_0^k \equiv 1$ and other components, $E^k{}_1, \ldots, E^k{}_p$, assuming values in $\{+1, -1, 0\}$. $C^k = \{+1, -1\}$ is the desired response.

OUTPUT: $W = \langle w_{p+1,0}, w_{p+1,1}, \cdots, w_{p+1,p} \rangle$ is a vector of integral 'pocket' weights where $w_{p+1,0}$ is the bias.

TEMPORARY DATA:

π = vector of integral perceptron weights, $\langle \pi_0, \pi_1, \cdots, \pi_p \rangle$.
$\mathbf{run}_\pi$ = number of consecutive correct classifications using perceptron weights π.
$\mathbf{run}_W$ = number of consecutive correct classifications using pocket weights W.
$\mathbf{num\_ok}_\pi$ = total number of training examples that π correctly classifies.
$\mathbf{num\_ok}_W$ = total number of training examples that W correctly classifies.

1. **Set** $\pi = \langle 0, 0, \cdots, 0 \rangle$ **and** $\mathbf{run}_\pi = \mathbf{run}_W = \mathbf{num\_ok}_\pi = \mathbf{num\_ok}_W = 0$.
2. **Randomly pick a training example** E^k **(with corresponding classification** C^k**).**
3. **If** π **correctly classifies** E^k**, i.e.**
   $$\{\pi \cdot E^k > 0 \text{ and } C^k = +1\} \text{ or}$$
   $$\{\pi \cdot E^k < 0 \text{ and } C^k = -1\}$$
   Then:
   - 3a. $\mathbf{run}_\pi = \mathbf{run}_\pi + 1$.
   - 3b. **If** $\mathbf{run}_\pi > \mathbf{run}_W$
     Then:
     - 3ba. **Compute** $\mathbf{num\_ok}_\pi$ **by checking every training example.**
     - 3bb. *(Ratchet:)* **If** $\mathbf{num\_ok}_\pi > \mathbf{num\_ok}_W$
       Then:
       - 3bba. **Set** $W = \pi$
       - 3bbb. **Set** $\mathbf{run}_W = \mathbf{run}_\pi$
       - 3bbc. **Set** $\mathbf{num\_ok}_W = \mathbf{num\_ok}_\pi$
       - 3bbd. **If all training examples are correctly classified (i.e.** $\mathbf{num\_ok}_W = |\{E^k\}|$**) then stop; the training examples are separable.**

   Otherwise:
   - 3A. *(Change step:)* **Form a new vector of perceptron weights**
     $$\pi = \pi + C^k E^k$$
   - 3B. **Set** $\mathbf{run}_W = 0$
4. **End of this iteration. If the specified number of iterations has not been taken then go to 2.**

Fig. 1. Pocket algorithm with ratchet. Perceptron weights, π, are computed that occasionally replace pocket weights, W.

Iter.	π	run_π	W	run_W	Choice	OK?	Action
1.	$\langle 0\ 0\ 0 \rangle$	0	$\langle 0\ 0\ 0 \rangle$	0	E^4	no	$\pi = \pi - E^4$ $\text{Run}_\pi = 0$
2.	$\langle -1\ -1\ -1 \rangle$	0	$\langle 0\ 0\ 0 \rangle$	0	E^4	yes	$\text{Run}_\pi = \text{Run}_\pi + 1$ $W = \pi$ $\text{Run}_W = \text{Run}_\pi$
3.	$\langle -1\ -1\ -1 \rangle$	1	$\langle -1\ -1\ -1 \rangle$	1	E^2	no	$\pi = \pi + E^2$ $\text{Run}_\pi = 0$
4.	$\langle 0\ -2\ 0 \rangle$	0	$\langle -1\ -1\ -1 \rangle$	1	E^3	no	$\pi = \pi + E^3$ $\text{Run}_\pi = 0$
5.	$\langle 1\ -1\ -1 \rangle$	0	$\langle -1\ -1\ -1 \rangle$	1	E^4	yes	$\text{Run}_\pi = \text{Run}_\pi + 1$
6.	$\langle 1\ -1\ -1 \rangle$	1	$\langle -1\ -1\ -1 \rangle$	1	E^2	yes	$\text{Run}_\pi = \text{Run}_\pi + 1$ $W = \pi$ $\text{Run}_W = \text{Run}_\pi$
7.	$\langle 1\ -1\ -1 \rangle$	2	$\langle 1\ -1\ -1 \rangle$	2	E^3	yes	$\text{Run}_\pi = \text{Run}_\pi + 1$ $\text{Run}_W = \text{Run}_\pi$
8.	$\langle 1\ -1\ -1 \rangle$	3	$\langle 1\ -1\ -1 \rangle$	3	E^1	no	$\pi = \pi - E^1$ $\text{Run}_\pi = 0$
9.	$\langle 0\ 0\ 0 \rangle$	0	$\langle 1\ -1\ -1 \rangle$	3	...		

Fig. 2. Pocket algorithm iterations.

- ***Categorical Input***: 有时候一些input并不是简单的一个数值，而是一组特征，比如$color \in red, blue, green, yellow$这样子的话应该：

1. Generate four new dimensions: red, blue, green and yellow
2. Recode categorical value as a binary vector $[red, blue, green, yellow] = (0, 1)^4$
3. 所以如果(0, 0, 0, 0)四个值的顺序分别代表的是red, blue, green, yellow的话，那么表示red的就是:

   (1, 0, 0, 0)

- Also, 2 digits to 表示四种不一样的值的做法也可以被使用，例如：
  - [(0, 0), (0, 1), (1, 0), (1, 1)]分别可以表示四种不一样的值
  - 使用这种方法的话：训练的时间会变长，并且也需要多个neuron在hidden layer里面
- In the general case, if an attribute (e.g. color) can take one of *n*different values, then *n*new dimensions are obtained.

ANN-Why?

Cognitive Modeling的原因：

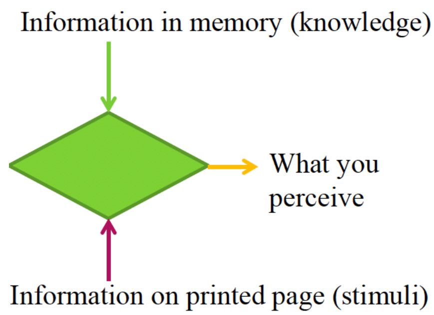
1. To understand **cognitive processes** better
    - Cognitive Process: Understanding *language*, *learning*, *perception*, *recognition*, *memory*, *logic processing*, *computation*, *attention*, *decision making*
2. To computationally implement a cognitive process.
    - 比如建造一个思想的模型
3. Compare and evaluate the various explanations of the cognitive processes
4. Predict outcomes of cognitive processes（预测认知过程的结果）
5. May not be fully accurate to account for human errors and uncertainty –different from **computational modeling**.
    - Computational modeling指的是普通的数学模型，和数学模型不同，Cognitive modeling不可以完全正确的解释人类的错误及不确定性

- 其他的方法：
    1. Rule based approaches
    2. Constraints
        - Multiple if-then-else coding can get very complicated
        - Difficult to extract rules –tacit knowledge which is difficult to transfer
        - Cannot implement **asynchronous** data flow and **parallel processing**
- ***Cybernetic model of the brain at MIT***
    - The first ANN model proposed by *Warren McCulloch, Walter Pitts, Jerry Lettvin*
- 在Brain Neuron中的“Learning”：
    - **Repeated simultaneous firing of neurons strengthens synapses**
    - Skill learning by **doing something repeatedly**
    - Skill learning, **rewarding experience**
- 在ANN中的“Learning”
    - 学习会改变/重塑ANN的结构
    - 改变阈值（threshold）
    - 通过改变不同的权重，来加强node和node之间的connection
- **ANN中三种主要的学习方式**：
    1. Error correction learning
    2. Reinforcement/Correlation/Hebbianlearning
    3. Competitive learning
        - 这个大致的概念就是：**nodes compete for the right to respond to a subset of the input data.**

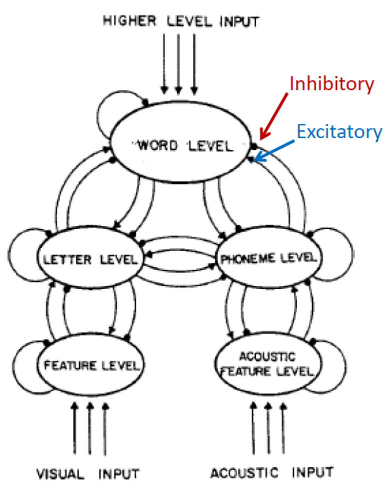
- ***Excitatory messages***: increase the activation level of their recipients
- ***Inhibitory messages***: decrease the activation level of their recipients

Interactive Activation Model

- **ANNs implement Parallel Distributed Processing (PDP) where processing is done in a distributed network in parallel.**
    - PDP同时也解释了**top-down**和**bottom-up**这些信息是怎么结合在一起来完成一个类似文字识别这样子的感知任务的



    - 基本上top-down信息可以理解成是已经习得的固有知识，而bottom-up信息可以理解成现在需要处理的信息/stimuli，比如当阅读的时候，看到了“work”这个图像，得到了这个刺激，再结合本身的习得的知识，就可以知道这个是“work”这个词
- ***Interactive Activation Model***是第一个获得显著成功的ANN by *McClelland and Rumelhart*



    - The **arrow** in the diagram represnet excitatory connections, and the **circular** ends represent inhibitory connections.
    - Note that an active word detectors mutually inhibit each other(即在特定node中，当他的值超过了阈值的时候，不仅会继续process这个information，他还会给同一个layer中的其他的node发出inhibit的讯号（比如一个单词是work，在经过了letter level之后，传输信息到word level，在word level中，如果有两个相似的node：“work”和“worR”，那么这个时候他们都会接收到讯号，但是随着“work”的阈值的达到，“work”会向“worR”发出inhibit的讯号，让他达不到阈值））
        - 所以在同一个level中的connection必定是inhibitory的
        - 不同level的connection可以是excitatory也可以是inhibitory的

- 信息的传递是**双向**的
- There are no connection between non-adjacent level
- “Higher Level Input”实际上只是指代了更高级别的层级，比如说在word level上面可能还有sentence level，paragraph level等等。他们也会向lower level传递信息（feeding downward to the lower level）。

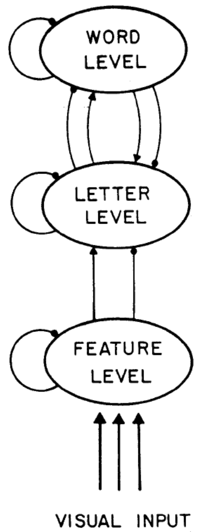
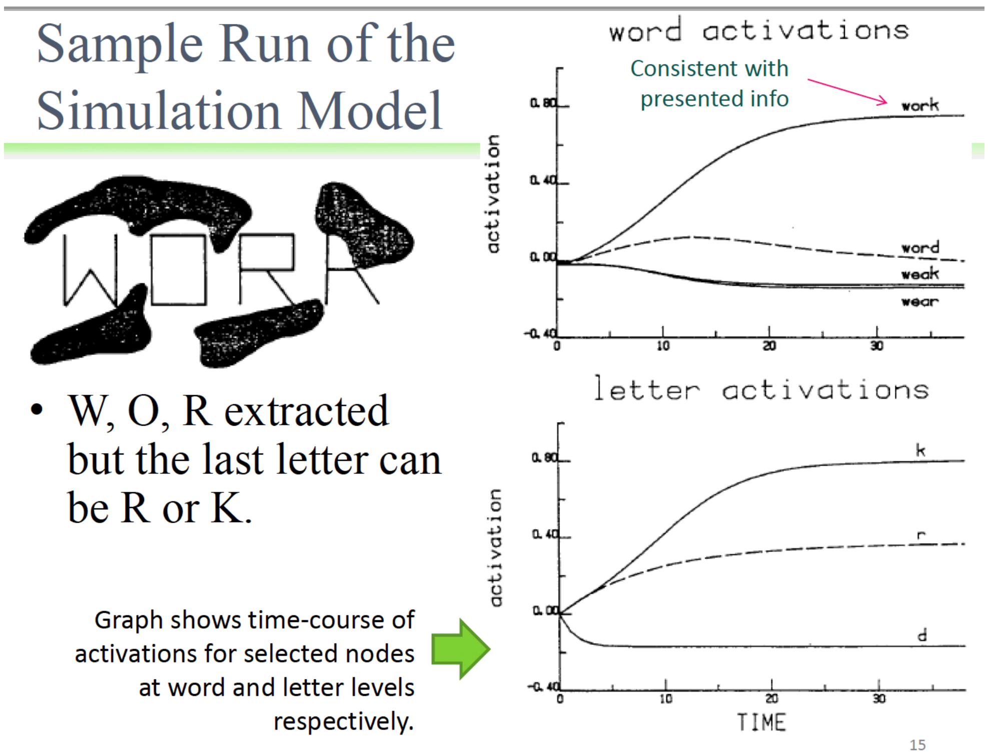
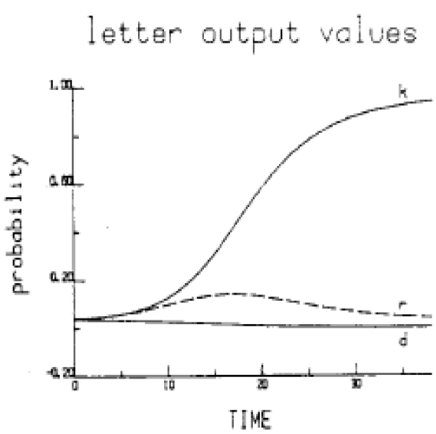


Figure 2. The simplified processing system.

- 为了简化这个模型，现在来到了这个比较简单的模型。在这个简单的模型中：
  - Delayed higher level processes and phnological processes
  - Ignored the reciprocity of activation that may occur between word and letter levels and any other levels of the system
- Input包含了visual feature detectors that are on for each letter
- ***Activation value***: Activation value is a real number. For node i, the activation value is $a_i(t)$(t指的是input). 一个具有positive activation value的node我们把他称为“active”. 如果一个node没有input，那么他们会处于“inactive”的state。那么这个resting level指的就是一个低于activation value的值。每个node的resting level都可以不一样
- ***Resting Level***: Resting level for node i is r_i. For units not at rest, decay back to the resting level occurs at some rate θ_i.
  - Nodes for high-frequency words have resting levels higher than those for low-frequency words.(????)
- ***Net input***: 当传输信息的时候，因为一个unit可能会接收到多个input，那么需要知道这个unit接收到的信息的和，就是net input: $n_i(t) = \sum_j \alpha_{ij} e_j(t) - \sum_k \beta_{ik} i_k(t)$, 其中在这个公式里面$e_j(t)$/$i_j(t)$指的是excretory/inhibitory neighbour的activation的值，a_{ij}/β_{ik}分别对应的是excretory/inhibitory neighbour的weight



- 在上面这张图中，可以看到不同的word node/letter node的activation的大小，在这里，**设定是对于word to word/letter to letter的连接中：inhibitory的值是0，也就是说，如果“k”被激活了，那么他不会向“r”发送inhibitory的信号**，所以可以看到在下图里面，r最后的增率慢慢趋近0，而不是变成负的。**那么如果inhibitory的值没有被设定成0的话，这张图就会像下面这样**：



Final output as probability of the different letters – wait until becomes stable

- ***Summary*** for IAM(model of context effects in perception of letters)

    1. Visual input excites detectors for visual features in the display

2. Active features inhibit other features and

    - Activate those letters which contain the features and
    - Inhibit letters which do not contain the features.

3. Active letters inhibit other letters and

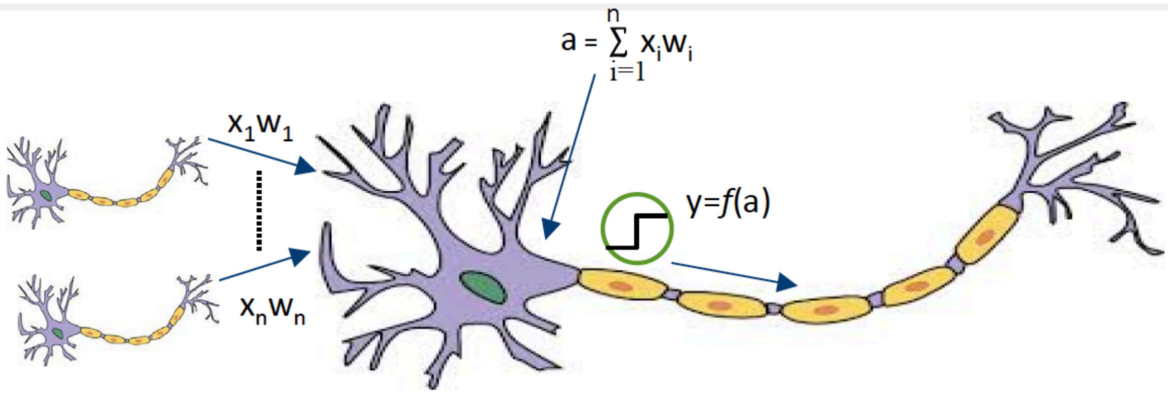
- Activate words which contain the letters and

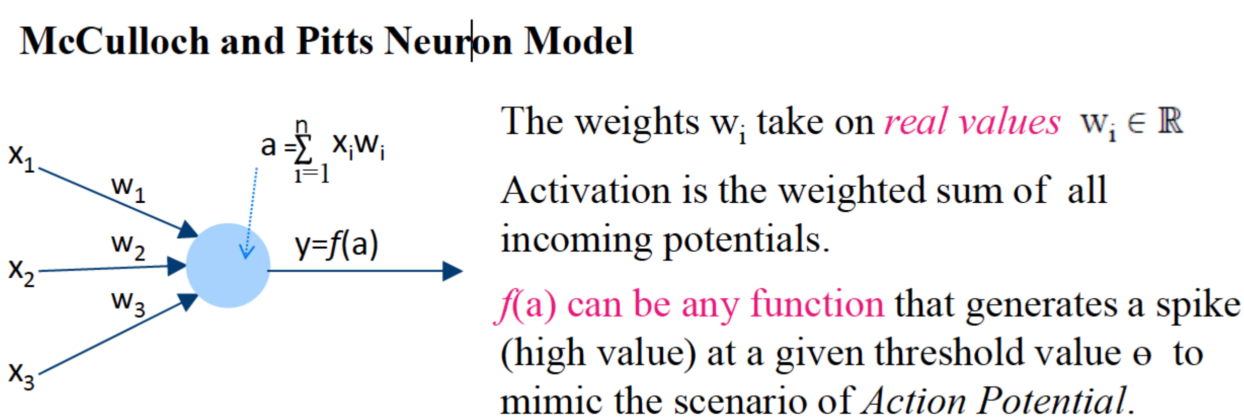
- Inhibit words which do not contain the letters

4. Active word detectors mutually inhibit each other then
   - Send feedback to letter level to activate the letters they contain and inhibit letters they do not contain.
     - Feedback strengthens perception of constituent letters(加强了组成这个单词的字母node的神经元)
5. Active letter detectors
     - Send feedback to feature level to activate the features they contain and inhibit features they do not contain
     - Feedback strengthens perception of constituent features.
6. 所以这整个其实是一个cycle：先是由最初时的信号到feature input，在这里被激活的feature会inhibit其他没有这个feature的feature node然后把信号传输到下一个letter level，经历了同样的操作之后，会传输到下一个word level，然后又会往反方向发送信号，经历同样的操作。此消彼长之后，会得到一个具有很好的能力的模型

ANN Design

- Weight w_i是一个实数：$w_i \in \mathfrak{R}$，是一个一开始自己设定，后面模型会经过学习自己调整





- 所以可以看到：使一个node被激活的方程和这个node被激活之后发出的信号的方程是不一样的
- ***The Activation Function***:

1. Threshold Function

$f_i(net_i(t)) = 1$ if $net_i(t) \geq \theta$

$f_i(net_i(t)) = 0$ if $net_i(t) \geq \theta$

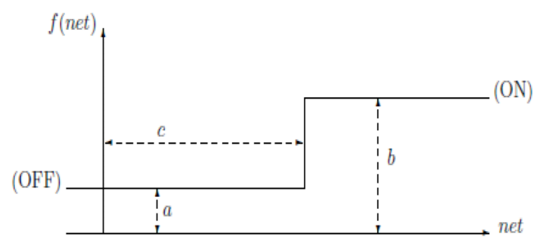
- **Output Function**:
  - The simplest node functions are:
    1. **Identity**, $f(\text{net}) = \text{net}$, and its non-negative variant $f(\text{net}) = \max(0, \text{net})$
    2. Constant functions $f(\text{net}) = c$
    3. Signum function $$f(net) = \begin{cases} +1 & \text{if } net > 0 \\ -1 & \text{if } net < 0 \\ 0 & \text{if } net = 0 \end{cases}$$
  - 在第一个function中，这是一个非负函数，可以看到当他的$f(x)$小于0的时候，它的最终output会是0

4. Step Function

- Simplest function that captures the idea of a "firing threshold"
- Can be used as a class identifier
- Problem: Very small change in $net_i(t)$ can cause a spike and hence change the output

$f(\text{net}) = a$ if $net < c$

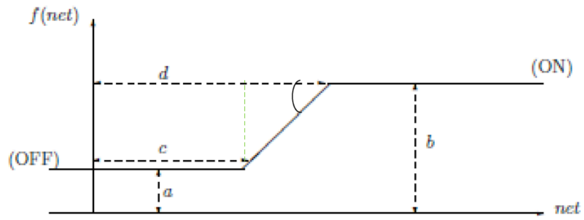
$= b$ if $net >= c$



5. a) Ramp Function

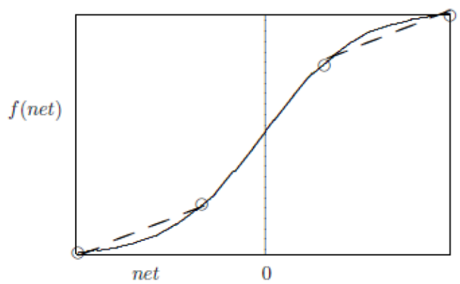
- The ramp function is continuous and almost everywhere differentiable in exchange of the simple ON/OFF description of the output.

$$f(net) = \begin{cases} a & \text{if } net \leq c \\ b & \text{if } net \geq d \\ a + \frac{(net-c)(b-a)}{d-c} & \text{otherwise} \end{cases}$$



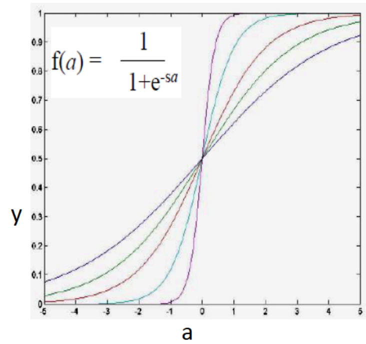
5. b) Piecewise Linear Functions

- Consist of finite number of linear segments, and are thus differentiable almost everywhere.
- Easier to compute than general nonlinear functions such as sigmoid functions.
- Can be used to avoid sudden change in output like the step function (from 0 to 1).



6. Sigmoid Function

- These functions are continuous and differentiable everywhere, and asymptotically approach saturation values (0 and 1 as shown in the picture)
- The parameter s controls the slope of the sigmoid function. Greater s value will give steeper curve.

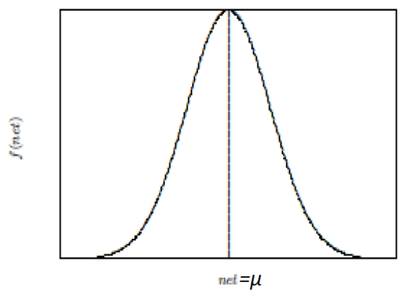


$$\lim_{net \to +\infty} f(net) = 1$$

$$\lim_{net \to -\infty} f(net) = 0$$

7. Gaussian Functions

- Continuous bell-shaped functions.
- Also called 'radial-basis' function.
- f(net) asymptotically approaches 0 (or some constant) for large magnitudes of net, with a single maximum for $net = \mu$, say $\mu = 0$. Greater σ → wider curve.



$$f(net) = \frac{1}{\sqrt{2\pi}\sigma} exp[-\frac{1}{2}(\frac{net-\mu}{\sigma})^2]$$

- 上面其实描述了很多可以用来当作***Output function***的选择
- （？？？）Differentiability of output functions is desired to manipulate behavior by adjusting parameter values

Principle Component Analysis

Feature Reduction

- 当输入向量的维度过大时，computationally expensive and time consuming
- 所以有的时候，好几个feature会被合成一个feature，或者可以认为是多维度的东西通过technique去把他的维度调整：These features are more important in the sense that they are better able to discriminate or classify the data，这样的做法叫做feature reduction

Means of feature reduction

- 二逼方法：直接删掉一个维度，直接删掉维度的话，information loss，并且即使是想要比对处理过的信息和原来的信息也很难，因为现在维度都不一样了
  - 建立在二逼方法上面的调整：Replace the componenet x_i with the average of all x_i for all data points P and then find the error

    new vector: $X' = (x_1, x_2, ..., \bar{x_i}, x_{i+1}, ..., x_n)$

Each modified input vector $\boldsymbol{x}'$ now differs from the original vector by (just one component is different)

$$(X - X')^2 = \left(\sqrt{(x_1 - x_1)^2 + ... + (x_i - \overline{x_i})^2 + ...}\right) = (x_i - \overline{x_i'})^2$$

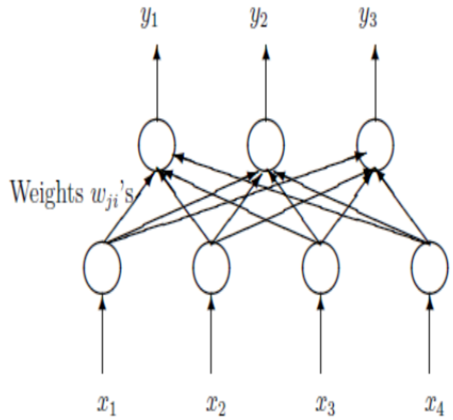
So, the mean squared error for all P transformed input vectors,

$$E(x_i - \overline{x_i'})^2 = \frac{1}{P}\sum_{j=1}^{P}(x_{ij} - \overline{x_{ij}'})^2 = \text{variance of } x_i$$

  - 通过以上的式子可以得到替换掉x_i的error和x_i的variance，那么对于每个单独的数据，都经过这样的处理之后，两两比对：
    - 删除掉最低的variance的component
    - 把那个被删掉的component替换成average feature

PCA-ANN Approach

- A **Principal Component Analysis Network** may be used to extract the desired transformation rather than resorting to the mathematical derivations.



- $y_j = \sum_{i=1}^{n} w_{ji} x_i$, y其实就是向量内积
- y个output node里面，各存着一个（1, x）的weight，最后出来y个output node，而每个output node中的这个（1, x)的weight，到时候代表着那个eigenvector的row，所你最后会出来一个(y, x)的一个eigenvector
- 要把原来的(x, 1)的数据变为(y, 1)，就直接（y,x）点乘(x, 1)就变成了（y, 1）了

- 这是一个Feedforward network with no hidden nodes
- 对于每个w的增量可以表示为：

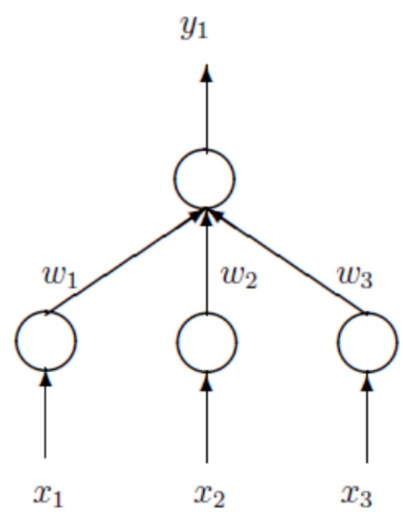
$$\Delta w_l = \eta Y_l x_l^T - K_l W_l$$

这个式子中，l = iteration，K depends on the algorithm

- One option for $K = Y_l Y_l^T$ suggested by Williams(1985)

At every iteration through the training period,

1. w_l (weight matrix) converges to $||W|| = 1$
2. The direction of W is the maximal eigen-vector of $S(T)$, the covariance matrix of the data set, so
3. W lies in the direction that maximizes variance, which minimizes the loss.



PCA network for example 5.9 and 5.10.

- For multiple output nodes j and input nodes i, an easier representation of the weight change rule is $\Delta w_{ji} = \eta Y_j (X_i - \sum_{k=1}^{j} Y_k W_{ki})$

- 看Example！！

Consider the data from Example 5.8, with $\eta_\ell = 1.0$, and initial weights $(0.3, 0.4, 0.5)$.

For the first input $x_1 = (0, 0.2, -0.7)$,
$y = (0.3, 0.4, 0.5) \cdot (0, 0.2, -0.7) =$ -0.27 Using the K_ℓ suggested by Williams,

$$\Delta W = (-0.27(0.00, 0.20, -0.70) - (-0.27)^2(0.30, 0.40, 0.50)) .$$

$$W = (0.30, 0.40, 0.50) + \Delta W = (0.278, 0.316, 0.652).$$

For the next input (x_2), $y = -0.23$ and

$$\Delta W = (-0.231(0.10, -0.20, -0.30) - (-0.231)^2\ (0.278, 0.316, 0.651))$$

$$W = (0.278, 0.316, 0.652) + \Delta W = (0.240, 0.346, 0.687).$$

Subsequent presentations of x_3, x_4, and x_5 change the weight matrix (in this case, a vector) W to $(0.272, 0.351, 0.697)$, $(0.238, 0.313, 0.751)$, and $(0.172, 0.293, 0.804)$, respectively.

This process is repeated, cycling through the input vectors $x_1, \ldots, x_5$. By the end of the second iteration, the weight vector becomes $(-0.008, 0.105, 0.989)$, and the third iteration changes it to $(-0.111, -0.028, 1.004)$, converging towards the first principal component.

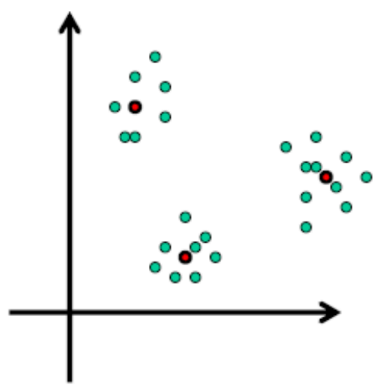
- A PCA-ANN only computes weighted sums of inputs, and carries out Hebb's Rule learning, which was originally devised as an explanation of learning and modifications in biologic neurons

Unsupervised Learning

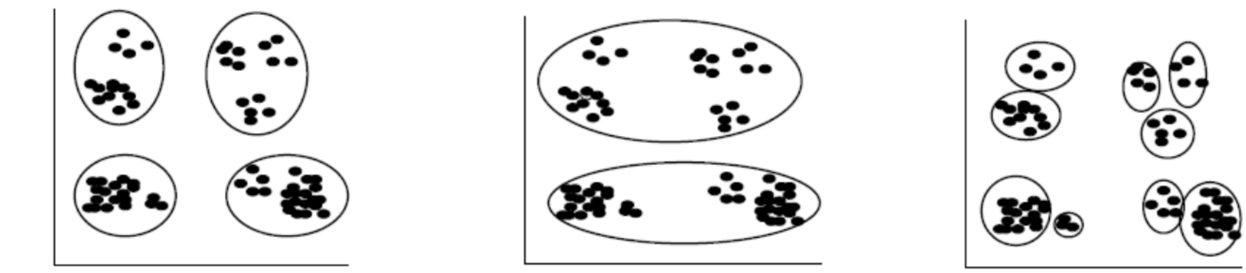
- Proceeds to discover special features and pattern from available data without using external help(也就是说，没有“desired output”)
- 应用：
  - Clustering
  - Vector quantization：向量量化，找到附近既定的点，来当做一个区间的代表，从而简化资料量
  - Approximation of data distribution
  - Feature extraction
  - Dimensionality reduction：降维，指在某些限定条件下，降低随机变量个数，得到一组“不相关”主变量的过程
- 大多数无监督学习的技术和很多现有的统计模型非常像

Clustering(聚类)

- 聚类是把相似的对象通过静态分类的方法分成不同的组别或者更多的子集)（subset），这样让在同一个子集中的成员对象都有相似的一些属性，常见的包括在坐标系中更加短的空间距离等。
- Given a number of data points, determine a set of representative centroids(其实就是在一个聚类中，找到他的“图心”，也就是说要判断未知的数据，是根据这个新的数据跟之前找到的centroids的距离来判断它属不属于这个聚类的。所以centroids也被称为prototype，cluster centers， reference vectors)
- ***Data belongs to the cluster whose centroid is the closest***



- 对同样一组数据可以进行不同的聚类分析，通过改变聚类的数量，可以实现同一组数据不同的聚类方法



Reasonable number of clusters　Small Number of Clusters　Too Many Clusters

- Clusters are evaluated by measuring the average squared distance between each input pattern and the centroid of the cluster in which it's placed

$$E_{cluster} = \frac{1}{patterns}\sum_{patterns}||(pattern - centroid)||^2$$
$$E_{total} = \sum_{cluster} E_{cluster}$$

K-Means 聚类算法

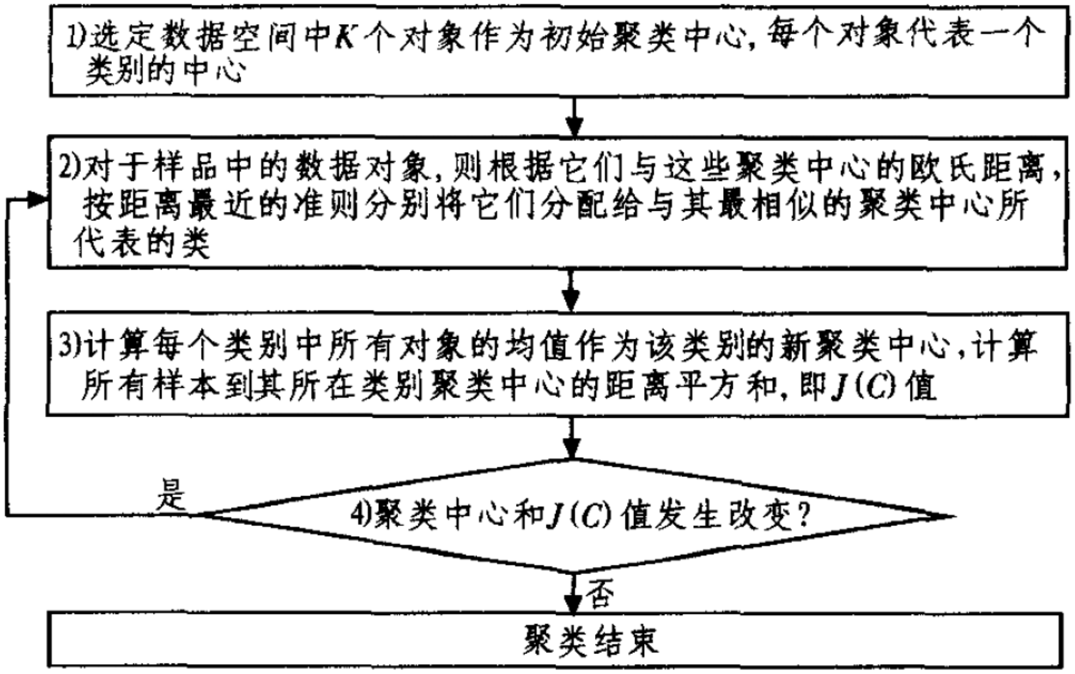
- 上面讲到的这个目标函数也同时是K-Means聚类算法的目标函数
    1. 首先先任意选出k个centroids，每个centroids代表一个聚类，所以现在有k个聚类
    2. 有input$x_1, \dots, x_n$
    3. 计算每个input距离这些centroids的距离，并且找到对于x_i离得最近的centroids，把x_i assign给距离最近的centroids代表的聚类
    4. 然后现在可以得到目标函数：

$$J(C) = \sum_{k=1}^{K} J(C_k) = \sum_{k=1}^{K}\sum_{x_i = c_k}\|x_i - \mu\|^2 = \sum_{k=1}^{K}\sum_{x_i = c_k} d_{ki}\|x_i - \mu\|^2$$

    - 其中，从右边开始看起，μ代表了每个聚类的centroid的位置；d_{ki}代表了当数据被成功归类（不管有没有正确）时为1，没有被成功归类时为0（没有影响），整个训练的目标是为了令$J(C)$，就是总的各类的距离平方和最小
    - 为了令其最小，首先选择最优的d_{ki}，这个值只要在前面的步骤能保证每个input能被分到距离最近的centroids的话就会是1，也就不用在意了
    - 然后固定d_{ki}为1，求最低的μ，这个时候将J对μ求导，得到：

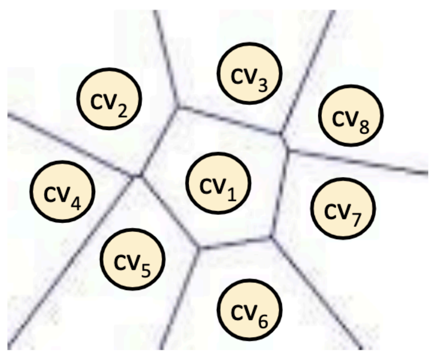
$$\mu_k = \frac{\sum_n d_{ki} x_n}{\sum_n d_{ki}}$$

      也就是说，对于每一次迭代，每个聚类中的centroid应该被更改成被分类到这个聚类的所有数据点的平均值
    - 学习会不断循环，直到centroid的位置不发生改变



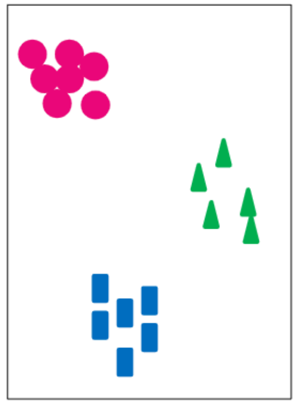
Vector Quantization

- Applies unsupervised learing to divide an input space into several connected regions called ***Voronoi Regions***, representing a quantization of the space

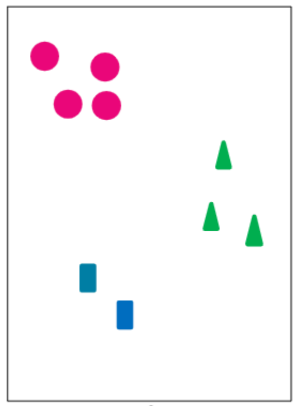


- Each region is represented using a single vector called a ***Codebook Vector(CV)*** which are also called ***Voronoi centres***
- 上面这张图被称为**Voronoi Diagram**，有八个voronoi regions和八个cv
- Every point in the input space belongs to a region and is mapped to the corresponding CV，所以说许多输入向量会被map到同一个CV
- 用监督学习和无监督学习都可以训练，监督学习的话每个输入向量都会匹配一个类似desired output的desired region；无监督学习的话使用普通的聚类算法就可以实现了
- **Therefore, the set of CVs is a compressed form of information represented by all input data.**

Approximation of Data Distribution



a



b

- The data distribution in b is a concise description of the larger amount of data in a drawn from a probability distribution
    - Data distribution in b is considered to be an approximation of that in a
    - Points in b may not be a subset of points in a, can be done using unsupervised learning
        - 如果使用无监督学习的话，每个聚类只有一个centroid

Feature Extraction and Dimensionality Reduction

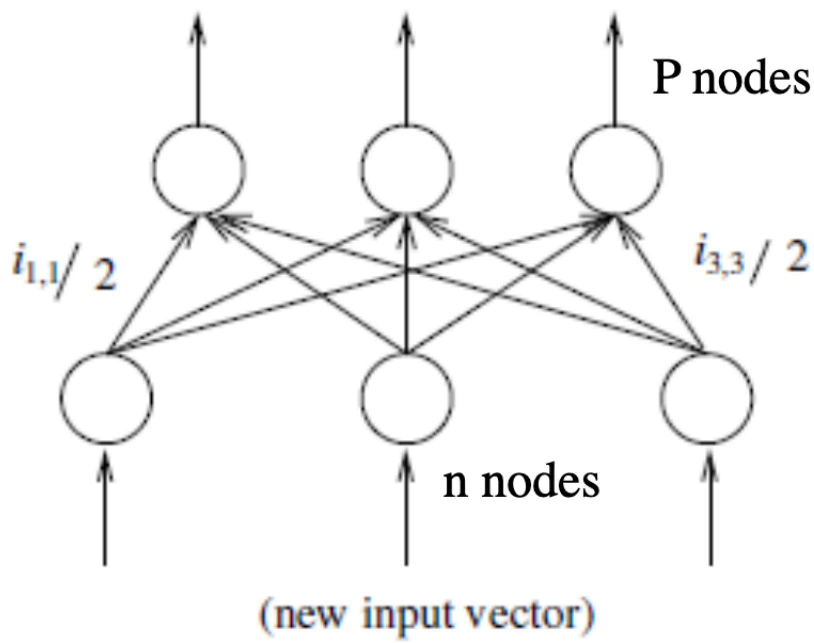
- The goal of feature extraction is to find the most important(distinguishable) features, i.e., those with the highest variation in a given population
- Important side-effect is reduction of input dimensionality and thereby, improve in processing time and cost

Winner-Take-All Networks/Competitive Learning

- During training, the output unit that provides the highest activation to a given input pattern is declared the weights of the winner and is moved closer to the input pattern, whereas the rest of the neurons are left unchanged
- This strategy is also called winner-take-all **since only the winning neuron is updated**
- Patterns in the same cluster are as alike as possible

Hamming Networks

- Hamming Distance：汉明距离，在信息论中，两个等长字符串之间的**汉明距离**（英语：Hamming distance）是两个字符串对应位置的不同字符的个数。换句话说，它就是将一个字符串变换成另外一个字符串所需要*替换*的字符个数。
    - **1011101**与**1001001**之间的汉明距离是2
    - **2143896**与**2233796**之间的汉明距离是3
    - 比如在matlab中两个向量之间的汉明距离的定义为两个向量之间不同的分量所占的百分比，比如(0, 0)和(1, 0)之间的汉明距离是0.5而(1, 1)和(0, 0)之间的汉明距离为1.0
    - 但是在这里他给的定义是：the number of differing bits, of input and stored vectors.
- 汉明网络是专门为求解二值模式识别问题而设计的（输入向量的每个元素只能是两个可能值中的一个）
- 目的是判定输入向量最接近于哪个标准向量（就是stored vectors）**A network to calculate Hamming distance (H) between stored vectors and input vector**



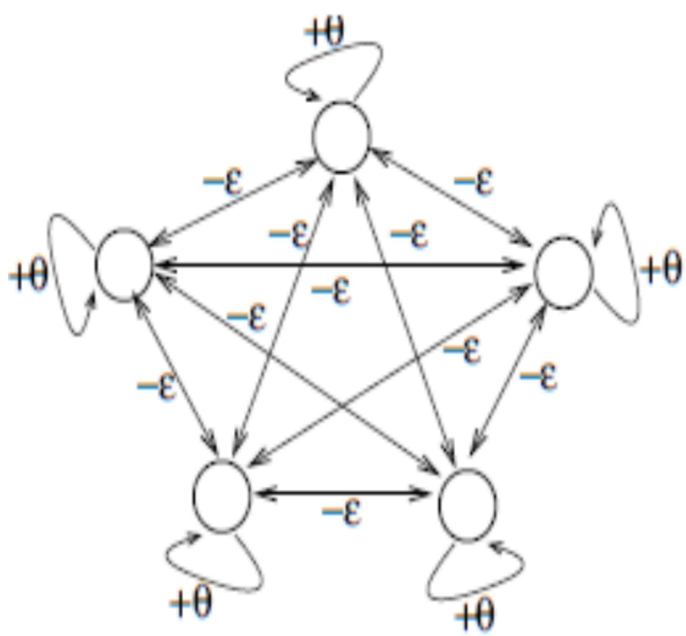
- P nodes输出层代表了有P个stored vector标准向量，而n nodes输入层其实只是代表了一个输入向量，分成n个nodes说明这个输入向量被分成n个部分（就是input vector的维度），图中的weights on links from input layer to an output layer代表了component of stored input patterns and their weight，weight是每个标准向量的那个部分除以2，而bias unit是恒定的$-\frac{n}{2}$
- output nodes generates Hamming distance between p^{th} stored pattern and the input pattern

$$W = \frac{1}{2}\begin{pmatrix} i_1^T \\ \vdots \\ i_P^T \end{pmatrix} = \frac{1}{2}\begin{pmatrix} i_{11}^T & i_{12}^T & & i_{1n}^T \\ & & \vdots & \\ i_{P1}^T & i_{P2}^T & & i_{Pn}^T \end{pmatrix} \quad \theta = \begin{pmatrix} -\frac{n}{2} \\ \vdots \\ -\frac{n}{2} \end{pmatrix}$$

Output is linear output calculated as o = WX+θ

- Example 5.1
- If all bits match($w_i = \frac{n}{2}$) then output is 0 otherwise < 0
- So we can determine which stored pattern is nearest to a new input pattern by taking the maximum of the outputs

Maxnet



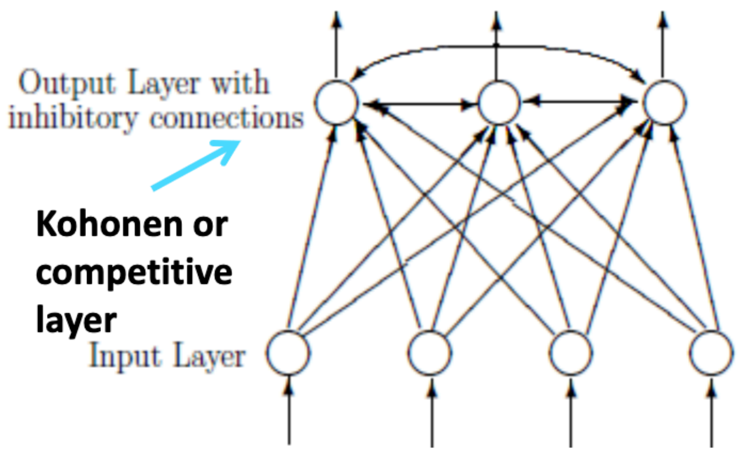
- A Maxnet is a recurrent competitive one-layer network used to determine which node has the highest initial activation.
- $\theta = 1, \epsilon \leq \frac{-1}{\#ofnodes}$, 其实这里的$\epsilon$就是w，整个运行会在只剩下一个node不为0，别的node都被减为0的时候停止，那么此时不为0的node就是winner
- 对于每个node，每一次的iteration都会接收别的node的output从而更新自己的值
    - node function $f(net) = max(0, net)$, $net = \sum_{i=1}^{n} w_i x_i$
    - 比如：initial activation values = (0.5,0.9, 1,0.9,0.9)，$\epsilon = -\frac{1}{5}$ and $\theta = +1$

- 第一次遍历：
  - $o_i = max(0, \epsilon * (\sum_n x_n) + \theta * x_i)$
  - $o_1(x_1 = 0.5) = max(0, -\frac{1}{5}(0.9 + 1 + 0.9 + 0.9) + 0.5)$
  - $o_2(x_2 = 0.9) = max(0, -\frac{1}{5}(0.5 + 1 + 0.9 + 0.9) + 0.9)$

  以此类推，每次遍历完了之后都会把现有的x_i更改为对应的o_i，直到只剩下一个位置不为0
- 虽然不知道为什么不直接用max来计算，可能是未来的数据会变得相当大？？
- 像上面讲到的Hamming net是拿来计算向量和现有的stored向量的负汉明距离的，而Maxnet则是拿来找到最大的

Simple Competitive Learning

$$\text{Output} = \begin{cases} 1 & \text{if node is a winner} \\ 0 & \text{otherwise} \end{cases}$$



Kohonen Learning

- 像上面这样子的generalize version which $\mathbb{R}^2 \rightarrow [0, 1]$也被称为Kohonen Learning
- 对于这样子的一个网络，每个output node都代表着一个聚类，而这个聚类的centroid/stored pattern/stored vector是由它的weight vector（权重向量）来代表的
- 存在一个distance measure，可以是欧式距离，曼哈顿距离，切比雪夫距离，汉明距离等等
- 对于每一个input vector，都会被传输到output layer里面去，计算这个input vector和每个output node的weight vector的距离，记为d，拥有最小的d的output node就是**Winner node**
- 因为要学习，所以必然使用Iterative Training，那么在每次遍历中，并不是对所有的权重向量都进行更改，而是只更改winner node的权重向量，旨在把权重向量拉得离被归类到这个聚类的输入向量越近越好

The jth output node is described by its weight vector from the input nodes,

$$w_j = (w_{j,1}, \ldots, w_{j,n}) \qquad l=\{1,\ldots,n\} \text{ input dimension}$$

A competition occurs to find the "winner" in the outer layer node j* whose weight vector (or "prototype" or "node position"), is nearest to the input vector i_l

$d(w_{j*}, i_l) \le d(w_j, i_l)$ where $j=\{1,\ldots,m\}$ # of output nodes

$k=\{1,\ldots,P\}$ # of data points, d is **Euclidean distance =**

$$d(w_j, i_k) = \sqrt{\sum_{l=1}^{n} (i_{k,l} - w_{j,l})^2}$$

42

- 下面这一页主要是说了一下通过net input怎么计算欧式距离

$$d(w_j, i_l) = \sqrt{\sum_{l=1}^{n} (i_{k,l} - w_{j,l})^2}$$ for kth data point and jth output node

- When expanded

$$d^2(w, i) = \boldsymbol{i_k}.\boldsymbol{i_k} + \boldsymbol{w_j}.\boldsymbol{w_j} - 2\sum i_{kl} w_{jl}$$

- Assuming $\| i_l \| = \| w_j \| = 1$ are unit vectors where

$$\|\mathbf{p}\| = \sqrt{p_1^2 + p_2^2 + \cdots + p_n^2} = \sqrt{\mathbf{p} \cdot \mathbf{p}}$$

- The distance can be calculated as

$$d^2(w, i) = 2 - 2\sum i_{lk} w_{jl} = 2 - 2\, \boldsymbol{i} \,.\, \boldsymbol{w}$$

- So, the distance will be minimum when $\boldsymbol{i.w}$ (dot product) is maximum ($\|i\| \, \|w\| \cos\theta$ is maximum at $\theta=0$).
- To generate output $\boldsymbol{i.w}$, use linear output neuron.

- 要令winner node的权重向量离data point越近，就要让权重向量沿着data point的方向移动

$$\Delta w_j = \eta(i_l - w_j)$$

$$w_{j+1} = w_j + \Delta w_j$$

- 这时候，当一个事先定义的阈值ϵ达到（$\Delta w_j < \epsilon$），那么此时w_j就会变成：

$$w_j = \frac{1}{\sum_l \delta_{j,l}} \sum_l i_l \delta_{j,l}$$

并且：

$$\delta_{j,\ell} = \begin{cases} 1 & \text{if the } j\text{th node is "winner" for input } i_\ell, \\ 0 & \text{otherwise,} \end{cases}$$

也就是说当达到这个阈值的时候，他就会变成the average of input vectors for which w_j is the winner

- 在学习的过程中，学习率可以人为的设置，可以不停的降低学习率，从而提高学习效率

Figure 5.4 Simple competitive learning algorithm

Initialize weights randomly;
repeat
- (Optional:) Adjust learning rate $\eta(t)$;
- Select an input pattern i_k;
- Find node $j*$ whose weight vector w_{j*} is closest to i_k;
- Update each weight $w_{j*,1}, \ldots, w_{j*,n}$ using the rule:

$$\Delta w_{j*,l} = \eta(\mathrm{t})(i_{k,l} - w_{j*,l}) \qquad \text{for } \ell \in \{1, \ldots, n\}$$

until network converges or computational bounds are exceeded

- 使用曼哈顿距离出来的结果也是不一样的
- Using three nodes does not guarantee that three "clusters" will be obtained by the network. **Close initial values may result in splitting of a cluster into two**
- 当学习率大的时候：sample的顺序和初始的权重向量都会很有影响

K-Means 聚类算法

- 上面提到过这里拉出来重新讲一下
- Applies batch update
- 上面讲到的这个目标函数也同时是K-Means聚类算法的目标函数
  1. 首先先任意选出k个centroids，每个centroids代表一个聚类，所以现在有k个聚类
  2. 有input$x_1, \ldots, x_n$
  3. 计算每个input距离这些centroids的距离，并且找到对于x_i离得最近的centroids，把x_i assign给距离最近的centroids代表的聚类
  4. 然后现在可以得到目标函数：

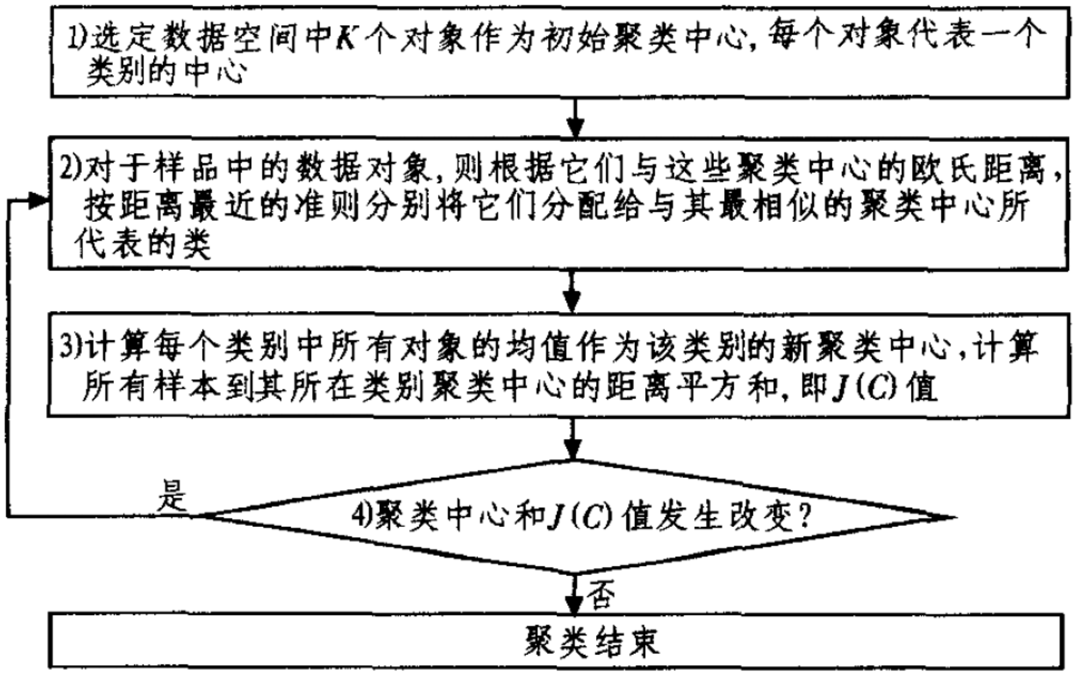
$$J(C) = \sum_{k=1}^{K} J(C_k) = \sum_{k=1}^{K} \sum_{x_i = c_k} \|x_i - \mu\|^2 = \sum_{k=1}^{K} \sum_{x_i = c_k} d_{ki} \|x_i - \mu\|^2$$

  - 其中，从右边开始看起，μ代表了每个聚类的centroid的位置；d_{ki}代表了当数据被成功归类（不管有没有正确）时为1，没有被成功归类时为0（没有影响），整个训练的目标是为了令$J(C)$，就是总的各类的距离平方和最小
  - 为了令其最小，首先选择最优的d_{ki}，这个值只要在前面的步骤能保证每个input能被分到距离最近的centroids的话就会是1，也就不用在意了
  - 然后固定d_{ki}为1，求最低的μ，这个时候将J对μ求导，得到：

$$\mu_k = \frac{\sum_n d_{ki} x_n}{\sum_n d_{ki}}$$

  也就是说，对于每一次迭代，每个聚类中的centroid应该被更改成被分类到这个聚类的所有数据点的平均值

- 学习会不断循环，直到centroid的位置不发生改变



Algorithm k-means Clustering

Figure 5.5 k-means clustering algorithm

Initialize k prototypes

$$w_j = i_\ell, j \in \{1, \ldots, k\},\ \ell \in \{1, \ldots, P\}$$

Each cluster C_j is associated with prototype w_j.

repeat

for each input vector i_ℓ **do**

Place i_ℓ in the cluster with nearest prototype w_{j*}

end for

for each cluster C_j **do**

$$w_j = \frac{1}{|c_j|} \sum_{i_l \in c_j} i_l \quad \text{where } |c_j| \text{ is the cluster size}$$

end for

$$\textbf{Compute}\quad E = \sum_{j=1}^{k} \sum_{i_l \in c_j} | i_l - w_j |^2$$

until E no longer decreases, or cluster memberships stabilize

The simple competitive learning algorithm conducts stochastic gradient descent on the *quantization error*

$$E = \sum_p \mid i_p - \mathbb{W}(i_p) \mid^2$$

where $\mathbb{W}(i_p)$ is the weight vector nearest to i_p

The number of nodes is assumed to be fixed, but the right choice may not be obvious. We may attempt to minimize $E + c(\text{number of nodes})$ instead of E, for $c > 0$.

The above adjustment is called **Regularization** (one approach).

- 也就是说simple competitive learning algorithm也可以使用随机梯度下降法来通过达到error surface的全局最小值来完成学习的目的

Learning Vector Quantizers(LVQ)

- A LVQ is an application of the above, uses winner-take-all network and illustrates **how unsupervised learning can be adapted to solve supervised learning**.
  - 其实也很简单，因为这时每组数据都已经知道了他们对应的聚类了，只要“把是这个聚类的centroid拉近，不是这个聚类的centroid推开”就好了
- Learning rate应该是随着epoch而下降的: $\eta(t) = \frac{1}{t}$或者是$\eta(t) = a[1 - (\frac{t}{A})]$，在这个里面a是个正数而A是个大于1的数
- 其实很简单，和上面的没什么差别

When pattern i from class C(i) is presented to the network, let the winner node $j^* \epsilon$ C(j^*).

If this is the correct class i.e., C(i)=C(j^*), j^* moves closer to i

$$\Delta w_{j^*,l} = \eta(t)(i_{k,l} - w_{j^*l})$$

Otherwise j^* moves away from i.

$$\Delta w_{j^*,l} = -\eta(t)(i_{k,l} - w_{j^*,l})$$

Figure 5.6 LVQ1 algorithm

```
Initialize all weights ∈ [0, 1]
repeat
  Adjust η(t);
  for each i_k do
    find node j* whose weight vector w_{j*} is closest to i_k;
  end for
  for ℓ = 1, ..., n do
    if the class label of node j* equals the desired class of i_k
    then
      Δw_{j*,l} = η(t)(i_{k,l} − w_{j*,l})
    else
      Δw_{j*,l} = − η(t)(i_{k,l} − w_{j*,l})
    end if
  end for
until network converges or computational bounds are exceeded
```

Unsupervised Learning - Part II

Adaptive Resonance Theory 自适应谐振理论

- ***Allow the number of clusters to vary with problem size***，能够自适应的当新的node进来的时候，添加新的cluster prototypes
- **Motivation**：很多神经网络是这样运作的：先训练，在训练进行到神经网络的表现到了一个比较好的地方的时候，就停止训练，并且这个神经网络就可以用来解决实际问题了，那么此时神经网络的权重和参数就不会再进行further的调整了。但是事实是，并没有那么多那么全面的training data给你去训练，例如此时出现了一个新的类别，需要去重新训练，如果只训练新的pattern，一个可能发生的情况是：Training on only the new silhouette could result in the network learning that pattern quite well, but forgetting previously learned patterns. 这时候，ART就显得很牛逼了，他可以preserve its previously learned knowledge while continuing to learn new things
- The user can control the dissimilarity between members of the same cluster by selecting a ***vigilance parameter***
- "***Resonance***" refers to the matching process
- ***ART网络的逻辑很好理解***：ART网络的思路是当网络接收新的输入时，按照预设定的参考门限检查该输入模式与所有存储模式类典型向量之间的匹配程度以确定相似度，对相似度超过门限的所有模式类，选择最相似的作为该模式的代表类，并调整与该类别相关的权值，以使后续与该模式相似的输入再与该模式匹配时能够得到更大的相似度。若相似度都不超过门限，就在网络中新建一个模式类，同时建立与该模式类相连的权值，用于代表和存储该模式以及后来输入的所有同类模式。

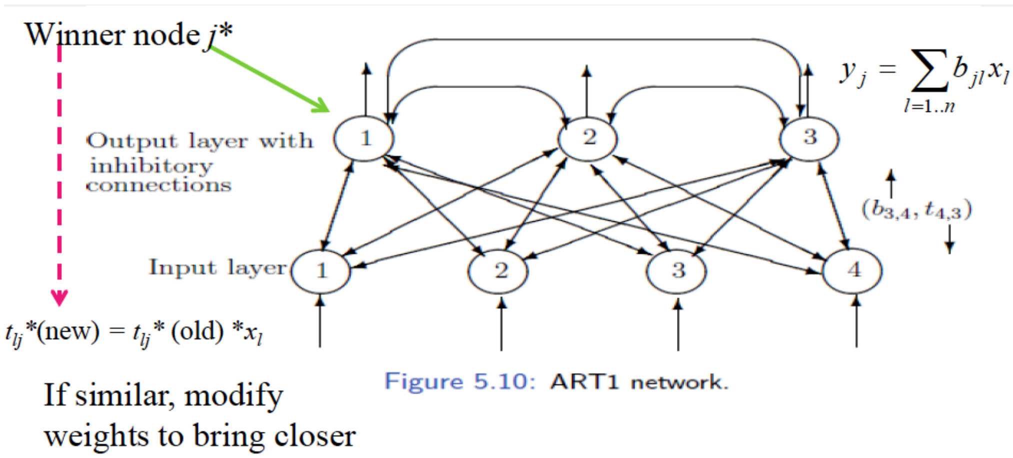


Figure 5.10: ART1 network.

  - 层数：两层，comparison layer(input layer) 和recognition layer(output layer)
  - 比较层输入：**首先全都是binary的值**，其次对于输入到第一层（comparison layer）的值，总共有三个：a component of the input pattern（一般意义上的输入，input vector here），a component of the feedback pattern（第二张图的话就是t），and a gain G1（增益控制讯号g1）
  - 比较层输出条件：只有当三个input里面的至少两个input是1（active）的状态的时候比较层才会输出
  - 认知层有m个神经元，代表m个输入模式类，m可以动态增长，以设立新的模式类。C层的输出向量C沿着认知层神经元的内星权向量到达R层神经元，经过竞争在产生获胜神经元处指示本次输入模式的所属类别。获胜神经元输出为1，其余为0。认知层每个神经元都对应着两个权向量，一个是将认知层前馈信号汇聚到R层的内星权向量，另一个是将认知层反馈信号散发到C层的外星权向量。
  - 增益控制讯号（在farhana的课件上没有被提到）：

        - G1：G1的作用就是使得比较层能够区分网络运行的不同阶段，在一开始的时候，从认知层并没有反馈信号（反馈信号是0），那么这时为了让他能够输出，G1被设定为1，可以让外界输入信号绕开和反馈信号的比较，直接输入到认知层。当认知层开始产生反馈信号的时候，G1的值被调整为0，那么这时比较层的输出全部取决于外界输入信号和反馈信号的比较：如果xi = tij，则，ci = xi，否则ci=0。可以看出控制信号G1的作用是使得比较层能够区分网络运行的不同阶段，网络开始运行阶段G1的作用是使得C层对输入信号直接输出，之后G1的作用是使C层行使比较功能，此时ci为xi和tij的比较信号，两者同时为1，则ci为1，否则为0。
        - G2：G2的作用是检查全部input pattern是否为0，所以也就是一个作用在各份量的pattern上面的一个逻辑或
    - 那么如果在第一个比较阶段里面，相似度并没有超过阈值，那么就会发出Reset信号，会再次进入之前的比较阶段，并且在这个时候，第一个阶段得出的获胜神经元将会持续收到一个抑制信号，因为这个抑制信号的存在，在第二个阶段里面，获胜神经元必定是第一阶段的次位获胜神经元。

```
"""Pseudocode"""
for x in pattern:
  for i < m: #认知层里面的神经元个数
    求得获胜神经元j
    通过t（j的回溯信号）来得到和阈值p比较的结果
    if (t*x) > p:
      这个信号被归类为j类
      调整权重b（前馈权重），t(回溯权重)
      break #开始接受下一个input
    else:
      对现在的这个获胜神经元j发射抑制信号（或者直接在剩下来的遍历中不考虑j)
      continue #开始找下一个获胜神经元并且进行比较阶段

  #如果遍历了这么多之后仍然找不到获胜神经元，那么将会学习新的cluster
```

    - Each input pattern is presented several times, and many be associative with different clusters before the network stabilizes，当然在每个成功的找到了获胜神经元的遍历中，前馈权重和和回溯权重都将会被调整
    - signals travel ***back and forth*** between the output layer and the input layer ("resonance") until a match is discovered.
    - 如果遍历了这么多之后仍然找不到获胜神经元，那么将会学习新的cluster

Figure 5.11 Algorithm for updating weights in ART1

Initialize each $t_{\ell,j}(0) = 1,\ b_{j,\ell}(0) = \frac{1}{n+1}$

while the network has not stabilized **do**

1. Let A contain all nodes;
2. For a randomly chosen input vector x, compute $y_j = b_j \cdot x$ for each $j \in A$.
3.

repeat

(a) Let j^* be a node in A with largest y_j.

(b) Compute $s* = (s_1^*, \ldots, s_n^*)$ where $s_\ell^* = t_{\ell,j^*}\ x_l$ ← Signal sent back to input

(c) If $\frac{\sum_{\ell=1}^{n} s_\ell^*}{\sum_{\ell=1}^{n} x_\ell} \le \rho$ then: remove j^* from set A ← Not enough similarity to input

else: associate x with node j^* and update weights:

Only weight of j* is updated

$$\left.\begin{aligned} b_{j^*,\ell}(\text{new}) &= \frac{t_{\ell,j^*}(\text{old})\ x_\ell}{0.5 + \sum_{\ell=1}^{n} t_{\ell,j^*}(\text{old}) x_\ell} \\ t_{\ell,j^*}(\text{new}) &= t_{\ell,j^*}(\text{old}) x_\ell \end{aligned}\right\} \quad 1$$

x and t are binary values

until A is empty or x is associated with some node

4. If A is empty, create new node with weight vector using eq. 1 for $t_{l,j^*}(\text{old})=1$

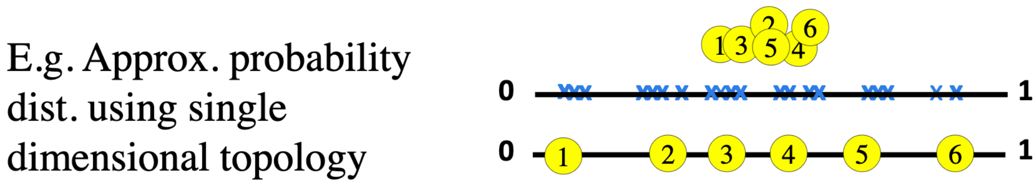
end while

- 可以看到一般首次前馈权重的定义为$\frac{1}{n+1}$
- 所以其实vigilance test指的就是相似度的比较测试
- Top-down weights are modified by computing intersections with the *input vector*, so that the *number of 1's gradually decreases or remains the same*. That is also the reason for initializing each top-down weight to 1.
- Given sufficient number of nodes, "outliers" that ought not to belong to any cluster will also be assigned separate nodes.
- 上面这几段都是quote了课件的废话

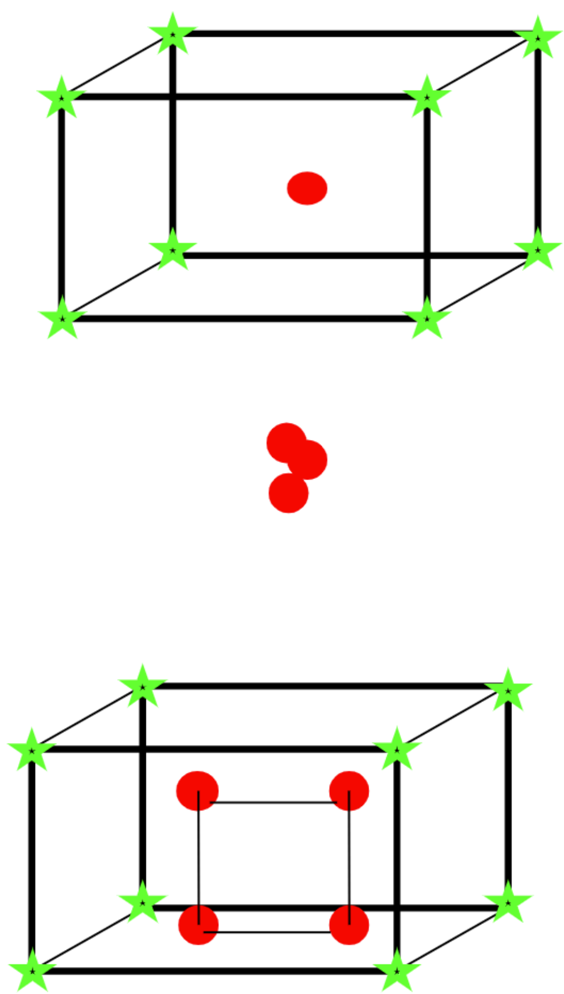
Topologically Organized Networks, Self-Organizing Maps(SOM)

- Topology：只考虑物体间的位置关系而不考虑他们的形状和大小，***“对特定对象（称为拓扑空间下不变之性质的研究，尤其是那些在特定可逆变换下不变之性质。”***
  - specified in terms of ***neighbourhood relation*** among nodes
    - can be expressed in terms of indices and links among prototypes or centroids
    - distance is measured in terms of # of links
- There is a **variation of Kohonen Learning** which combines competitive learning with topological structuring such that <u>adjacent nodes tend to have similar weight vectors.</u> It is called **Self-Organizing Maps (SOM)**, or sometimes called **Kohonen** or **Self-Organizing Feature Maps (SOFM)**.
  - Kohonen Learning在part I
  - 为什么叫self-organizing，因为learning确保了权重向量可以被找到并且在欧几里得空间里面保持拓扑关系，使用一个邻近函数来保持输入空间的拓扑性质

- Competitive learning在output layer里面有抑制连接；topological structuring requires that each node also has excitatory connections to a small number of nodes
- **和之前的学习不一样的是，之前的学习只调整了winner node，但在这里，除了调整winner node，还调整了winner node的neighbour**
- 应用：
    - clustering
    - vector quantization
    - approximation probability distribution: The centroids in a given region is roughly proportional to the number of input vectors



    - proximity of prototypes: suppose we want to interpolate the outputs of the prototypes - that is, we need to know which is closet and the one that is next closet
    - 所以拓扑信息能够让cluster保持他们的相对位置，这样子才可以predict the order of the prototypes，否则，权重随机生成的，所以无法做到



- 像上面这张图，黄色指的是sample，而红色的点是centroid，如果使用一个不保留拓扑结构的网络的话，就会像上面的图一样，在rectangle里面的那个centroid will be the winner node for all the sample，而另外三个点会越拖越远，如果保留了拓扑结构的话，not only one point but its neighbors to be pulled towards the cube像图二一样，所以如果计算时间够长，是可以分得出来的
- 具体怎么运作和example在slide
    - 一开始需要给prototype连上线，从而表示neighbour这个概念

Distance Measure

- 在聚类之后，不同的prototype会有一个index，所以通过这个index，可以找到相邻的prototype，但是！neighbour distance和欧几里得距离是完全不一样的
- An SOM's execution can be viewed as consisting of two phases:
    - "volatile" phase: prototypes move a lot
    - "sober" phase: prototypes slowly settle down as cluster centroids
- 虽然sober phase也有在converge，但是一个ordered map（也就是连了线的prototypes）的出现是基于volatile阶段
- A topologically ordered configuration is reachable, but the network may move out of it, as in general, there is no energy function that is optimized (no gradient descent to reach minimum energy state) in the weight update rule.
- 根据example → **topological adjacency no longer guarantees proximity of weight vectors**（weight vectors之间的？？距离）也就是说根据那个example，两个不相邻的centroid反而考得更近了
- Topological vicinity is defined by the **topological distance D(t) between nodes**

  – The neighbourhood $N_j(t)$ contains nodes that are within a distance of $D(t)$ from node j at time t, where $D(t)$ decreases with time.

  – $D(t)$ does NOT refer to Euclidean distance in input space, it refers only to the length of the path connecting two nodes for the pre-specified topology chosen for the network.

- To make topological adjacency represent proximity of weight vectors, *decrease neighborhood D and learning rate with time*.

- 为什么要decrease neighborhood D？？？是因为D是被定义的两个centroid之间的path的长度

Weight Change Rule

- If $N_j(t) = \{j\} \cup \{$neighbours of j at time $t\}$ where j is the winner node, and i is the input vector presented to the network at time t, then the weight change rule is:

$$w_\ell(t+1) = \begin{cases} w_\ell(\mathrm{t}) + \eta(t)\ (i - w_\ell(\mathrm{t})), & \text{if } \ell \in \mathcal{N}_j(t) \\ w_\ell(\mathrm{t}), & \text{if } \ell \notin \mathcal{N}_j(t) \end{cases}$$

- Weights vectors often become **ordered**, i.e., topological neighbours become associated with weight vectors that are near each other in the input space.

- 也就是说，调整权重的时候是全部的neighbours一起调整，如果不是neighbourhood的话就保持不变

Kohonen's SOM Learning Algorithm

Select network topology (neighbourhood relation);
Initialize weights randomly, and select $D(0) > 0$;
while computational bounds are not exceeded do

1. Select an input sample i_l
2. Find the output node j^* with minimum
   $\sum_{k=1..n}(i_{lk}(t)-w_{jk}(\mathrm{t}))^2$
3. Update weights to all nodes within a topological distance of $D(t)$ from j^* (including j^*), using
   $w_j(t+1) = w_j(t) + \eta(t)(i_l(t) - w_j(t))$;
   where $0 < \eta(t) < \eta(t\text{-}1) < 1$ and $j \in N_{j^*}(t)$;
   Weights of non-neighboring nodes are left unchanged.
4. Reduce $D(t)$ and η after a time interval t'

end while

- 然后ppt还有example