Transparanten bij het vak Inleiding Adaptieve Systemen: Unsupervised Leren/ Self organizing networks. M. Wiering

---

Unsupervised Learning en Self Organizing Networks

Leerdoelen:

- Weten wat unsupervised learning is
- Weten hoe K-means clustering gaat
- Competitive Learning begrijpen en kunnen uitleggen
- LVQ begrijpen en kunnen toepassen
- Kohonen netwerken begrijpen en de leerformules kennen
- Kunnen uitrekenen wat gewichtenveranderingen in een competitief leersysteem zijn

---

Unsupervised Leren

In Unsupervised leren krijgen we enkel patronen x^p als input en geen doel output.

De geleerde informatie moet dus volledig uit de inputpatronen gehaald worden

Unsupervised leren kan gebruikt worden voor:

- Clustering: Groepeer de data in clusters.
- Vector quantisation: Discretiseer een continue inputruimte
- Dimensionaliteits reductie: groepeer de data in een subruimte van lagere dimensie dan de dimensionaliteit van de data
- Feature extraction: Extraheer kenmerken uit de data

K-means clustering

K-means clustering kan gebruikt worden op continue attributen.

Het algoritme begint met K prototype vectoren $w^1, \ldots, w^K$ welke elk een cluster $(C^1, \ldots, C^K)$ representeren.

Herhaal tot clusters niet meer veranderen:

1. Bereken welke voorbeelden $x^1, \ldots, x^n$ in elk van de clusters vallen.

   Een voorbeeld x^i valt in een cluster C^j als w^j de prototype vector is met de kleinste Euclidische afstand tot het voorbeeld:

   $$d(w^j, x^i) \leq d(w^l, x^i) \quad \text{Voor alle l} \neq \text{j}$$

   Met $d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$: de Euclidische afstand tussen x en y.

2. Zet de prototype vectoren op het centrum van de input voorbeelden in de betreffende cluster. Voor alle clusters C^j doe:

   $$w_i^j = \frac{\sum_{k \in C^j} x_i^k}{|C^j|}$$

Voorbeeld

We hebben vier voorbeelden:
(1,2)
(1,4)
(2,3)
(3,5)

We initialiseren 2 prototype vectoren $w_1 = (1, 1)$ en $w_2 = (3, 3)$

We zien dat de voorbeelden in de volgende clusters vallen:
(1,2) $\rightarrow$ 1
(1,4) $\rightarrow$ 2
(2,3) $\rightarrow$ 2
(3,5) $\rightarrow$ 2

Nu berekenen we de nieuwe prototype vectoren en krijgen: $w_1 = (1, 2)$ en $w_2 = (2, 4)$.

Hierna vallen alle voorbeelden weer in dezelfde clusters dus zijn we klaar.

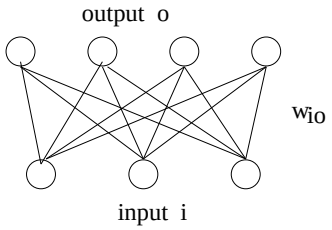
Competitive learning

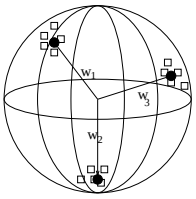
Competitief leren verdeelt de input ruimte over clusters in de input data

Er worden enkel input patronen x^p aangeboden. De output van het leernetwerk op een input patroon is de cluster waarin x^p valt.

In een simpel competitief leernetwerk worden alle inputs i met alle outputs o verbonden.

Hoe wordt de actieve (winnende) cluster bepaald?





We veronderstellen allereerst dat gewichten en inputs genormaliseerd worden tot lengte 1.

Adaptieve stappen voor het netwerk

(1) Elke output unit o berekent zijn activatie y_o door het "dot-product":

$$y_o = \sum_i w_{io} x_i = w_o x$$

(2) Vervolgens wordt de output neuron k met maximale activatie gekozen:

$$\forall o \neq k : \qquad y_o \leq y_k$$

Activaties worden gereset zodat $y_k = 1$ en $y_{o \neq k} = 0$.

(3) Tenslotte worden de gewichten van de winnende neuron k veranderd door de volgende leerregel:

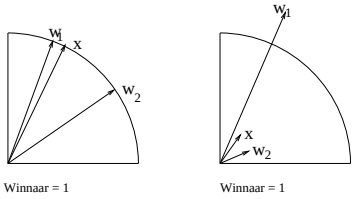
$$w_k(t+1) = \frac{w_k(t) + \gamma(x(t) - w_k(t))}{\|w_k(t) + \gamma(x(t) - w_k(t))\|}$$

De deler zorgt ervoor dat de gewichten vector genormaliseerd wordt.

Werking van competitive learning

In principe worden de gewichten vectoren in de input ruimte gedraaid naar het voorbeeld

Dus gaan gewichten vectoren wijzen naar gebieden waar veel inputs verschijnen
Het kan mis gaan als inputs en gewichten vectoren niet genormaliseerd zijn:
Het ongewenste effect is dat grote gewichten vectoren het winnen tegen kleine gewichten vectoren.



Selecteren van de winnaar met de Euclidische afstand

Als inputs en gewichten vectoren niet genormaliseerd zijn, kunnen we de Euclidische afstand nemen om de winnaar te bepalen:

$$\text{Winnaar} \quad k : \|w_k - x\| \leq \|w_o - x\| \quad \forall o.$$

Als de vectoren genormaliseerd zijn, geeft het gebruik van de Euclidische afstand dezelfde winnaar terug als het gebruik van het dot-produkt.

De gewichten update-regel schuift de gewichten vector van de winnende neuron naar het inputpatroon:

$$w_k(t+1) = w_k(t) + \gamma(x(t) - w_k(t)) \qquad (1)$$

Competitive Learning (niet genormaliseerd)

Er zijn K clusterpunten (neuronen) w^i, $1 \leq i \leq K$.

Eerst berekenen we voor elke clusterpunt de Euclidische afstand naar een voorbeeld met:

$$d(w^i, x^p) = \sqrt{\sum_j (w_j^i - x_j^p)^2}$$

De winnende neuron k heeft de minimale $d(w^k, x^p)$

Vervolgende verschuiven we de winnende neuron k naar voorbeeld x^p:

$$w_i^k = w_i^k + \gamma(x_i^p - w_i^k)$$

Voorbeeld: We hebben 2 (K = 2) neuronen met geinitialiseerde waarden: $w^1 = (1, 1)$ en $w^2 = (3, 2)$.

Nu krijgen we 4 leervoorbeelden:
$x^1 = (1, 2)$
$x^2 = (2, 5)$
$x^3 = (3, 4)$
$x^4 = (2, 3)$

We zetten de leersnelheid $\gamma = 0.5$. Dan krijgen we:

$x^1 = (1, 2) \rightarrow d(w^1, x^1) = 1, d(w^2, x^1) = 2$. Dus: Winnaar $w^1 = (1, 1)$ Toepassen van de update vergelijking geeft:
$w^1 = (1, 1) + 0.5((1, 2) - (1, 1)) = (1, 1.5)$.

$x^2 = (2, 5) \rightarrow d(w^1, x^2) = \sqrt{13.25}, d(w^2, x^2) = \sqrt{10}$. Dus: Winnaar $w^2 = (3, 2)$ Toepassen van de update vergelijking geeft:
$w^2 = (3, 2) + 0.5((2, 5) - (3, 2)) = (2.5, 3.5)$.

$x^3 = (3, 4) \rightarrow d(w^1, x^3) = \sqrt{10.25}, d(w^2, x^3) = \sqrt{0.5}$. Dus: Winnaar $w^2 = (2.5, 3.5)$ Toepassen van de update vergelijking geeft:
$w^2 = (2.5, 3.5)+0.5((3, 4)-(2.5, 3.5)) = (2.75, 3.75)$.

Probeer het nu zelf voor het laatste voorbeeld.

Initialisatie

Een probleem van de recursieve clustering methodes is de initialisatie: Het kan gebeuren dat een neuron nooit winnaar wordt en dus niet leert.

Twee oplossingen:

- Initialiseer een neuron op een inputpatroon
- Gebruik "leaky learning". Hier leren alle neuronen op alle inputpatronen met een zeer kleine leersnelheid $\gamma' << \gamma$:

  $$w_l(t+1) = w_l(t) + \gamma'(x(t) - w_l(t)), \forall l \neq k$$

Minimalisering van de kosten functie

Clustering houdt in dat overeenkomsten tussen inputs in dezelfde cluster veel groter zijn dan inputs in andere clusters.

De mate van overeenkomst wordt vaak uitgedrukt door de (inverse van de) Euclidische afstandsmaat te gebruiken

Een gebruikelijke maat om de kwaliteit van een clustering te berekenen is gegeven door de volgende kwadratische-fout kosten functie:

$$E = \frac{1}{2} \sum_p \|w_k - x^p\|^2$$

Hier is k weer de winnende neuron voor inputpatroon x^p.

We kunnen aantonen dat competitive learning het minimum van de kosten functie zoekt door de negatieve afgeleidde te volgen.

Bewijs:

De foutfunctie voor patroon x^p:

$$E = \frac{1}{2} \sum_i (w_{ik} - x_i^p)^2$$

waar k een winnende neuron is, wordt geminimaliseert door de gewichten update-regel van vergelijking (1).

Bewijs: we berekenen het effect van een gewichten verandering op de foutfunctie:

$$\Delta_p w_{io} = -\gamma \frac{\partial E^p}{\partial w_{io}}$$

Nu hebben we als gedeeltelijke afgeleidde van E^p:

$$\begin{aligned} \frac{\partial E^p}{\partial w_{io}} &= w_{io} - x_i^p, \quad \text{Als unit } o \text{ wint} \\ &= 0, \quad \text{anders} \end{aligned} \tag{2}$$

Hieruit volgt (voor winnaar o):

$$\Delta_p w_{io} = \gamma(x_o^p - w_{io})$$

Dus: de kosten functie wordt geminimaliseerd door herhaalde gewichten updates.

Vector quantisatie

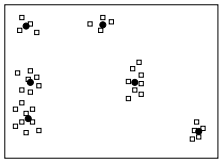
Een ander belangrijk gebruik van competitief leren is vector quantisatie.

Een vector quantisatie verdeelt de input ruimte in een aantal verschillende subruimtes.

Het verschil met clustering is dat we nu niet zo geinteresseerd zijn in clusters van gelijke data, maar meer in het quantificeren van de hele input ruimte.

De quantisatie welke geleerd wordt door competitive leren respecteert de verdeling van inputs: meer inputs in een regio leidt tot meer clusters

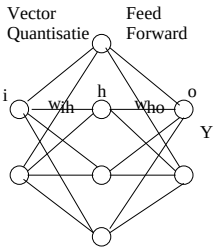
Een voorbeeld vector quantisatie:



Combinatie met Supervised leren

Vector quantisatie kan gebruikt worden als preprocessing stadium voor een supervised lerend systeem.

Een voorbeeld is het volgende netwerk dat vector quantisatie in de eerste laag combineert met supervised leren in de 2e laag:



We kunnen eerst de vector quantisatie uitvoeren en dan de supervised leerstap maken, of we kunnen beide lagen tegelijk aanpassen.

Leeralgoritme supervised vector quantisatie

- Presenteer het netwerk met input x en doel waarde $d = f(x)$.
- Voer de unsupervised quantisatie stap uit. Bereken de afstand van x naar elke gewichten vector en bepaal de winnaar k. Stel de gewichtenvector bij met de unsupervised leerregel.
- Voer de supervised approximatie leerstap uit:

$$w_{ko}(t+1) = w_{ko}(t) + \gamma(d - w_{ko}(t))$$

Dit is simpelweg de delta-regel met $y_o = w_{ko}$ waar k de winnende neuron is.

Als we een functie $g(x,k)$ definieren als:

$$\begin{aligned} g(x,k) &= 1, && \text{Als } k \text{ de winnaar is} \\ &= 0, && \text{Anders} \end{aligned}$$

Dan kan aangetoond worden dat de bovenstaande leerprocedure convergeert naar:

$$W_{ho} = \frac{\int_{\Re^n} y_o g(x,h) dx}{\int_{\Re^n} g(x,h) dx}$$

Dus: elke tabel-entry (gewicht van hidden unit h naar output unit o) convergeert naar het gemiddelde van de doelwaarde voor alle keren dat de neuron wint.

Supervised Vector Quantization

De winnende neuron verschuift volgens dezelfde update regel als bij vector quantisatie, maar bovendien wordt er een output y^k voor de winnende neuron w^k na het leervoorbeeld bijgesteld met:

$$y^k = y^k + \alpha(D^p - y^k)$$

Stel nu weer 2 clusterpunten :
$w_1 = (1,1), y_1 = 0$ en $w_2 = (3,2), y_2 = 0$.

We krijgen de volgende leervoorbeelden:
$(x^1 \rightarrow D^1) = (1, 2 \rightarrow 3)$
$(x^2 \rightarrow D^2) = (2, 5 \rightarrow 7)$
$(x^3 \rightarrow D^3) = (3, 4 \rightarrow 7)$
$(x^4 \rightarrow D^4) = (2, 3 \rightarrow 5)$

Stel we zetten de leersnelheid $\gamma = 0.5$ en $\alpha = 0.5$. Dan krijgen we:

$x^1 = (1,2) \rightarrow d(w^1, x^1) = 1, d(w^2, x^1) = 2$.
Dus: Winnaar $w^1 = (1,1)$ Toepassen van de update vergelijking geeft:
$w^1 = (1,1) + 0.5((1,2) - (1,1)) = (1, 1.5)$.
Dit is precies als hiervoor.

Het enige verschil is dat we nu ook de output van de winnende neuron moeten bijstellen:
$y^1 = 0 + 0.5(3 - 0) = 1.5$

We laten nu enkel zien hoe de output waarden van de neuronen veranderen, de gewichtenvectoren w^i veranderen net als hiervoor.

$x^2 = (2, 5)$. Winnaar is neuron 2.
$y^2 = 0 + 0.5(7 - 0) = 3.5$.

$x^3 = (3, 4)$. Winnaar is neuron 2.
$y^2 = 3.5 + 0.5(7 - 3.5) = 5.25$.

Probeer nu zelf de update voor voorbeeld 4 te berekenen.

Learning Vector Quantisatie (LVQ)

Is eigenlijk een supervised leeralgoritme voor discrete outputs

Deze netwerken proberen "decision boundaries" te leren aan de hand van gelabelde voorbeelden, zodat elk voorbeeld in een regio valt met de juiste klasse-label.

Het algoritme ziet er als volgt uit:

1. Associeer met elke output neuron o een klasse label y_o
2. Presenteer een leervoorbeeld (x^p, d^p)
3. Gebruik een afstandsmaat tussen gewichtenvectoren en inputvector x^p om de winnende neuron k_1 en de op een na beste neuron k_2 te vinden

   $$\|x^p - w_{k_1}\| < \|x^p - w_{k_2}\| < \|x^p - w_i\| \forall i \neq k_1, k_2$$

4. De labels y_{k_1} en y_{k_2} worden vergeleken met d^p, waaruit een gewichtenverandering wordt bepaald.

Update regels

- Als $y_{k_1} = d^p$: Voer de gewichten update regel voor k_1 uit:

  $$w_{k_1}(t+1) = w_{k_1}(t) + \gamma(x^p - w_{k_1}(t))$$

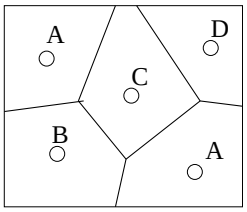
- Anders, als $y_{k_1} \neq d^p$ en $y_{k_2} = d^p$: Voer de gewichten update regel voor k_2 uit:

  $$w_{k_2}(t+1) = w_{k_2}(t) + \gamma(x^p - w_{k_2}(t))$$

  en verwijder de winnende neuron van het voorbeeld:

  $$w_{k_1}(t+1) = w_{k_1}(t) - \gamma(x^p - w_{k_1}(t))$$

Dus: de neuron met het juiste label (als de winnaar of de 1-na beste dit zijn) wordt geschoven naar het input patroon.



LVQ: voorbeeld

We hebben nu K clusterpunten (neuronen) met een gelabelde output. We berekenen de dichtsbijzijnde neuron w^{k1} en de op 1 na dichtsbijzijnde neuron w^{k2}.

Stel we beginnen met de volgende clusterpunten: $w^1 = (1, 1)$ en label $y^1 = A$. $w^2 = (3, 2)$ en label $y^2 = B$.

We krijgen de volgende leervoorbeelden:
$(x^1 \rightarrow D^1) = (1, 2 \rightarrow A)$
$(x^2 \rightarrow D^2) = (2, 5 \rightarrow B)$
$(x^3 \rightarrow D^3) = (3, 4 \rightarrow A)$
$(x^4 \rightarrow D^4) = (2, 3 \rightarrow B)$

Dan krijgen we: $(1, 2 \rightarrow A)$, winnaar is neuron 1 een na beste is neuron 2. De label van neuron $1 = D^1$. Dus: neuron 1 verschuift naar het voorbeeld:
$w^1 = (1, 1) + 0.5((1, 2) - (1, 1)) = (1, 1.5)$.

$x^2 = (2, 5)$ Winnaar is neuron 2. 1 na beste is neuron 1. De label van neuron 2 is gelijk aan de label D^2, dus neuron 2 verschuift naar het voorbeeld:
$w^2 = (3, 2) + 0.5((2, 5) - (3, 2)) = (2.5, 3.5)$.

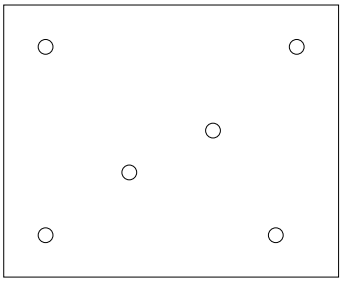
$x^3 = (3, 4)$. Winnaar is neuron 2. 1 na beste is neuron 1. De label van neuron 2 is niet gelijk aan de label D^3. De label van neuron 1 is wel gelijk aan D^3. Dus verschuiven we neuron 1 naar het voorbeeld en neuron 2 weg van het voorbeeld:
$w^1 = (1, 1.5) + 0.5((3, 4) - (1, 1.5)) = (2, 2.75)$.
$w^2 = (2.5, 3.5) - 0.5((3, 4) - (2.5, 3.5)) = (2.25, 3.25)$.

Bepaal nu zelf de updates voor voorbeeld 4.

Vraag: tekenen decision boundaries

De geleerde partitie van de input ruimte wordt ook wel een Voronoi diagram genoemd.

Vraag: Gegeven de volgende plaatsing van output neuronen, teken (ongeveer) het bijpassende Voronoi diagram



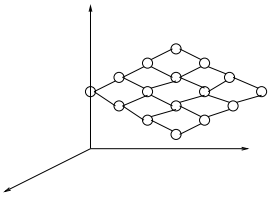
Kohonen netwerk

In een Kohonen netwerk zijn de output units geordend op een bepaalde manier, b.v. in een 2-dimensionale grid.

Deze ordening bepaalt welke neuronen buren van elkaar zijn.

Inputs die dicht bij elkaar vallen moeten gemapped worden op output units (in S) welke dicht bij elkaar liggen (dezelfde neuron of buren)

Vaak wordt een Kohonen netwerk van lagere dimensionaliteit gebruikt dan de inputvectoren. Dit is vooral handig als de inputs in een subruimte van $\Re^n$ vallen:



Kohonen netwerk leeralgoritme

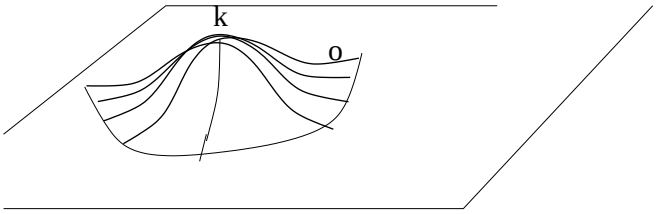
Voor een leervoorbeeld wordt weer de winnende neuron k berekend met de bekende Euclidische afstandsmaat.

Vervolgens worden de inputs van de winnende neuron en zijn (niet enkel directe) buren bijgesteld door:

$$w_o(t+1) = w_o(t)+\gamma g(o,k)(x(t)-w_o(t)) \quad \forall o \in S.$$

Hier is $g(o,k)$ een afnemende functie van de afstand tussen units o en k zodat $g(k,k) = 1$. Bijvoorbeeld:

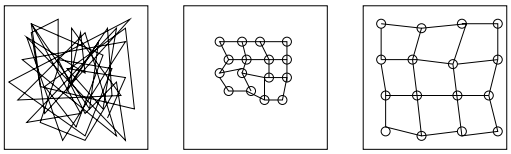
$$g(o,k) = exp(-buurafstand(o,k))^2$$



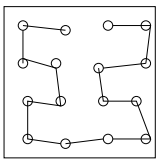
Voorbeeld leerproces

Door deze collectieve leermethode worden inputs die dichtbij elkaar vallen gemapped op output neuronen die dicht bij elkaar zitten.

Hierdoor blijft de topologie inherent in de inputsignalen bewaard in het geleerde Kohonen netwerk:



Als de instrinsieke dimensionaliteit in S kleiner is dan N, worden de neuronen van het netwerk "gevouwen" in de input ruimte:



Kohonen netwerk: Voorbeeld

We hebben een Kohonen netwerk met 3 neuronen verbonden in een lijn. We maken gebruik van de burenrelaties door $g(k,k) = 1$ te nemen en $g(h,k) = 0.5$ te zetten als h en k directe buren zijn, anders is $g(h,k) = 0$.

Nu berekenen we weer eerst de winnende neuron k, en vervolgens updaten we alle neuronen als volgt:

$$w^i = w^i + \gamma g(i,k)(x^p - w^i)$$

Nu initialiseren we $w^1 = (1,1)$, $w^2 = (3,2)$, $w^3 = (2,4)$. Weer zetten we $\gamma = 0.5$.

We krijgen weer de voorbeelden:
$x^1 = (1,2)$
$x^2 = (2,5)$
$x^3 = (3,4)$
$x^4 = (2,3)$

Op $x^1 = (1,2)$ wint neuron 1. Dit resulteert in de update:
$w^1 = (1,1) + 0.5 * 1((1,2) - (1,1)) = (1,1.5)$.
We moeten ook de buren updaten. $g(2,1) = 0.5$ en $g(3,1) = 0$. Dus updaten we neuron 2:
$w^2 = (3,2) + 0.5 * 0.5((1,2) - (3,2)) = (2.5,2)$.

Op $x^2 = (2,5)$ wint neuron 3. Dit resulteert in de update:
$w^3 = (2,4) + 0.5 * 1((2,5) - (2,4)) = (2,4.5)$.
We moeten ook de buren updaten. $g(2,3) = 0.5$ en $g(1,3) = 0$. Dus updaten we neuron 2:
$w^2 = (2.5,2) + 0.5 * 0.5((2,5) - (2.5,2)) = (2.375,2.75)$.

Op $x^3 = (3,4)$ wint neuron 3. Dit resulteert in de update:
$w^3 = (2,4.5) + 0.5 * 1((3,4) - (2,4.5)) = (2.5,4.25)$.
We moeten ook de buren updaten. $g(2,3) = 0.5$ en $g(1,3) = 0$. Dus updaten we neuron 2:
$w^2 = (2.375,2.75) + 0.5 * 0.5((3,4) - (2.375,2.75)) = (2.53,3.06)$.

Probeer het nu zelf voor het laatste voorbeeld.

Kohonen netwerk: Supervised leren

Een Kohonen netwerk kan ook gebruikt worden voor supervised leren. Hiervoor kunnen we elke output neuron h met een tabel-entry (w_{ho}) verschaffen

Voor het bepalen van de totale output y kunnen we de outputs van buren mee laten tellen door:

$$y = \frac{\sum_{h \in S} g(h,k) w_{ho}}{\sum_{h \in S} g(h,k)}$$

Elke neuron leert dan de gemiddelde output over alle gewogen input patronen:

$$w_{ho} = w_{ho} + \gamma(d - w_{ho}) \frac{g(h,k)}{\sum_{i \in S} g(i,k)}$$

De unsupervised leerstap kan eventueel aangepast worden om de beste neuronen het snelst te verschuiven naar het leervoorbeeld.

Conclusie

- Unsupervised leermethoden kunnen gebruikt worden voor: Clustering, Vector quantisation, Dimensionaliteits reductie en Feature extraction.
- In competitief leren strijden de neuronen om geactiveerd te worden.
- De unsupervised leermethoden kunnen uitgebreid worden met een extra output laag om ook supervised te kunnen leren. Hiervoor wordt de delta regel voor de nieuwe laag gebruikt.
- De getoonde leeralgoritmen kunnen het best omgaan met continue inputs, voor discrete inputs zijn extra aanpassingen nodig
- De leeralgoritmen respecteren het lokaliteits principe: inputs die dicht bij elkaar liggen worden samen gegroepeerd.
- Voor Supervised leren zijn de getoonde leeralgoritmen geschikt als de functie erg grillig (niet smooth) is. Door extra neuronen toe te voegen kan een goede approximatie van een functie geleerd worden.