Kognitiv Modellering

Mårten Szymanowski

20 januari 2004

Innehåll

1 Kognitiv Modellering

Syftet med detta dokument är att (förhoppningsvis) ge läsaren:

· en inblick i varför man modellerar.

· en ökad förståelse för vissa grundläggande egenskaper hos Artificiella Neurala Nät (ANN) som de delar med ”riktiga” neurala nät.

· en terminologi som används vid kognitiv modellering.

Läsaren ombeds att läsa figurtexterna då dessa innehåller mycket information som inte återfinns i den övriga texten.

1.1 Varför?

Låt oss börja med en mycket kort och förenklad förklaring till varför kognitiv modellering är av värde. Ungefär så här resonerar de som bedriver denna typ av verksamhet:

1. En organisms psykologiska tillstånd och beteende är, i huvudsak, en konsekvens av det som sker i dess hjärna/nervsystem.

2. De byggstenar som ett ANN är uppbyggt av fångar det som sker i verkliga neurala nät (det vill säga, hjärnan). Med andra ord, den simulerade neuronen (noden) och synapsen (vikten), delar vissa pricipiella egenskaper med avseende på dess *funktion*.

3. Om 1 och 2 är sant kan ett ANNs olika aktiveringstillstånd tolkas som psykologiska tillstånd där dess beteende är en konsekvens av dessa tillstånd.

4. Om 3 är sant följer att vi kan *förklara* en organisms psykologiska tillstånd och beteende i kraft av ANN modellen. En modell i vilken varje enskild variabel är under modellerarens kontroll.

Poängen med kognitiv modellering är således att konstruera ett ANN och i kraft av de eventuella likheter mellan de data som modellen genererar och psykologiska data kunna förklara det sistnämnda. Då modellen, i sig, utgör förklaringen så är analysen och förståelsen av den primär. Att endast skapa en funktion som gör den ”rätta” mappningen mellan input och output utan någon vetskap om *varför* resultatet blev som det blev utgör således ingen förklaring. Det enda som åstadkommits är ett nytt ”beteende” som i sin tur återstår att förklaras.

2 Grundläggande psykologiska fenomen

Människor reagerar, uppfattar och kategoriserar sin omvärld på ett karaktäristiskt sätt. Nedanstående begrepp har valts då de är fundamentala för mänsklig kognition.[1]

2.1 Stimulusgeneralisering

Någon som håller på att lära sig Morse blandar ofta ihop bokstaven C ($- \cdot - \cdot$) med bokstaven Y ($- \cdot - -$). De två stimulusmönstren delar vissa element och därför så sker denna *generalisering*.[2] Ett ännu mer konkret exempel kan, t.ex., vara vilken färg du tillskriver ett stimulus med våglängden 560 nm. Med största sannolikhet säger du 'röd'. Med stor sannolikhet kallar du även ett stimulus med våglängden 562 nm för 'röd'.

Ett barn som får lära sig att säga 'röd' när denne tittar på en röd bok kan komma att kalla *alla* böcker för röda. Det vill säga, en klass av flera stimuli som delar någon eller några likheter, vilka ger upphov till samma respons. Detta är ett exempel på *över*generalisering. Delvis beror detta på att böckerna, oavsett färg, delar vissa särdrag med avseende på dess form.

2.2 Stimulusdiskriminering

Motpolen till generalisering är *diskriminering*. Ett barn som t.ex. kallar alla böcker, oavsett färg, för röda och sedan lär sig att bara kalla röda böcker för röda, har lärt sig att diskriminera. Det vill säga, att kunna differentiera mellan olika stimuli även om de delar vissa särdrag. Barnet har således lärt sig att diskriminera med avseende på vissa karaktäristiska skillnader i särdrag.

2.3 Abstraktion/Klassifikation

När endast en *egenskap* hos ett objekt ger upphov till en respons kallas det för en *abstraktion*. Ett exempel på detta är numerositet, vilket är en abstrakt egenskap hos en mängd — inte en inneboende egenskap hos de ingående elementen i sig. Ett konkret exempel: Någon som lärt sig vad numerositeten för talet '2' kan tala om när det står *två* objekt på, låt säga, ett bord oavsett objektens färg, form, vikt, densitet, placering etc.

[1] Denna uppräkning är givetvis inte på något sett uttömmande.

[2] Holland, J.G. and Skinner, B.F. (1961). *The Analysis of Behavior: A Program for Self-Instruction*. New York: McGraw-Hill Book Company, Inc.

3 Artificiella neurala nät

Psykologer kan konstatera *att* ovanstående fenomen existerar. En ANN modell som uppvisar samma (el. liknade) egenskaper kan således ge *möjliga* förklaringar till *varför* och *hur* dessa fenomen framträder/uppkommer.

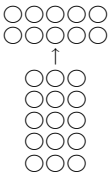
Ett ANN implementerar en funktion $f(x)$ som givet en viss indata ger en viss utdata. Denna funktion beror av i) nätverkets topologi (hur det är sammankopplat), ii) nodernas respektive aktiveringsfunktion och iii) rådande viktuppsättning. Viktuppsättningen kan förändras genom att nätet "tränas" på en träningsmängd. Vad som är viktigt att komma ihåg är att i) – iii) *tillsammans* utgör $f(x)$. Om nätet inte beter sig på önskat sätt så kan det mycket väl "bara" bero på att, till exempel, träningsmängden elller den slumpmässiga viktuppsättningen varit "felaktig".

Nedan följer två ANN-implementationer. Dessa är tänkta att i) belysa vissa karaktäristiska egenskaper hos ANN vilka sammanfaller med de psykologiska fenomenen som beskrivits ovan, samt ii) visa hur man kan tolka och analysera ett ANN. Med andra ord, få en inblick i varför nätet beter sig som det gör.

3.1 Generalisering och Diskriminering

Detta experiment gjordes med BP++ modulen vilket ingår i PDP++ (v.2.2).

Topologi Nätverkets struktur består av två lager, ett indatalager (15 noder) och ett utdatalager (10 noder) där *alla* noder i inputlagret är kopplade till *alla* noder i utdatalagret.



In-/Utdata Nätverket tränas på följande datamängd om 10 par input-/output mönster, en för vardera siffra ('•' innebär maximal aktivation och '○' ingen aktivitet):

Output/Respons: 10 mönster (0-9), lokalistisk representation

●○○○○	○●○○○	○○●○○	○○○●○	○○○○●	○○○○○	○○○○○	○○○○○	○○○○○	○○○○○
○○○○○	○○○○○	○○○○○	○○○○○	○○○○○	●○○○○	○●○○○	○○●○○	○○○●○	○○○○●

●●●	○○●	●●●	●●●	●○●	●●●	●●●	●●●	●●●	●●●
●○●	○○●	○○●	○○●	●○●	●○○	●○○	○○●	●○●	●○●
●○●	○○●	●●●	●●●	●●●	●●●	●●●	○○●	●●●	●●●
●○●	○○●	●○○	○○●	○○●	○○●	●○●	○○●	●○●	○○●
●●●	○○●	●●●	●●●	○○●	●●●	●●●	○○●	●●●	○○●

Indata/Stimulus: 10 mönster (0-9), distribuerad representation

Vid olika indatamönster är *samma* nod delaktig i att representera *olika* siffror. Detta kallas för att representationen är *distribuerad*. Vad beträffar utdata så representerar varje enskild nod ett (och endast ett) begrepp, i detta fall en siffra, vilket kallas för att representationen är *lokalistisk*. Aktiveringen hos respektive nod kan då tolkas som en "hypotesdetektor". Det vill säga, varje enskild nods aktivation ses som en indikator på "styrkan" hos det begrepp som representeras.

Träningsförfarandet Respektive siffra presenterades för nätet (stimulus) varpå nätets svar (respons) jämförs med det önskade svaret. Skillnaden mellan önskat och erhållet svar (felet[3]) används av en viktjusteringsalgoritm, som förändrar vikterna så att felet (förhoppningsvis) minskar. I detta fall uppdateras vikterna med Backpropagation algoritmen. Vid varje epok så presenterades nätet för samtliga exempel i träningsmängden en gång i en permuterad ordning. Vilket har som följd att inget enskilt träningsexempel presenteras fler (eller färre) gånger än något annat och presentationen sker i slumpmässig ordning.

Resultat Resultatet presenteras i form av tre klusterdiagram (fig. 1 och 2). Klusterdiagram[4] visar, genom en trädstruktur, likhetsförhållandet (avståndet) mellan ett antal vektorer. Två siffermönster (vektorer) som är nära varandra är det på grund av att det euklidiska[5] avståndet mellan dessa två vektorer är minst i förhållande till alla övriga kombinationer. De är, med andra ord, mest lika *varandra*. Då Y-axeln endast är ett index över de olika vektorerna är den därför inget att fästa vikt vid. X-axeln däremot visar avståndet mellan mönster inom ett kluster. Det vill säga avståndet mellan två mönster kan utläsas genom att "vandra" i klusterdiagrammet och summera alla horisontella linjer som existerar mellan dessa. Detta är inget som i praktiken behöver utföras, den stora fördelen med denna typ av diagram är att det erbjuder ett (relativt) enkelt sätt att få en överskådlig inblick i hur nätet behandlar (internt kategoriserar) sin input. Som en konsekvens av detta är det även (i viss utsträckning) också möjligt att förutsäga hur nätet kommer att behandla aldrig tidigare presenterad input. Observera figurerna nedan och jämför hur respektive kluster relaterar till respektive

[3]Vanligen brukar en av följande två felfunktioner användas, Cross entropy Error (CE) eller Summed Squared Error (SSE). För ett träningsexempel beräknas CE respektive SSE enligt:

$$CE = -\sum_k (t_k \log o_k + (1 - t_k) \log(1 - o_k)$$

$$SSE = \sum_k (t_k - o_k)^2$$

där k är ett index över alla element i utdatavektorn, o_k är nätets output för element k och t_k är det önskade svaret för samma element. Om skillnaden mellan o_k och t_k är noll så kommer CE resp. SSE vara noll. Skillnaden mellan funktionerna är följande: Om skillnaden ökar så ökar SSE proportionellt mot skillnaden, CE däremot ökar logaritmiskt. Således, om man använder sig av en lokalistisk utdatarepresentation är det oftast bättre att beräkna utdatafelet med CE istället för SSE. Att notera är att Leabra++ (i PDP++ paketet) endast använder sig av SSE. Det går givetvis bra att använda sig av en lokalistisk utdatarepresenation i Leabra++, förutsatt att man sätter k (i kWTA) till ett lämpligt värde för outputlagret.

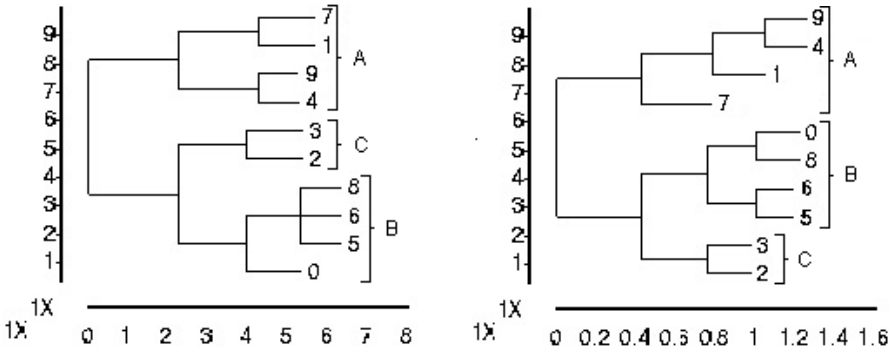
[4]Ett klusterdiagram erhålls genom att, från nätverksfönstret, skapa en `NewMonitor` detta genom att välja vilka noder du vill föra statisk över och därefter välja `Monitor Values`. Skapa därefter en `GridLog`. Presentera alla träningsexempel för nätet, *en gång*. Därefter väljs `Analyze -> Cluster Plot` i `GridLog`. I `GridLog`-fönstret ska följande parametrar ställas in. I `data`-fältet anges alla noder som klusteranalysen ska utföras på (det existerar inga om en `NewMonitor` inte skapats för dessa), därefter; `label` = `Event`, `disp_log` = `NULL`, `graphic` markeras, `metric` = `EUCLIDIAN`, `norm` avmarkeras och `tol` = 0.

[5]Det euklidiska avståndet är ett mått på likheten mellan två mönster (vektorer) och beräknas enligt:

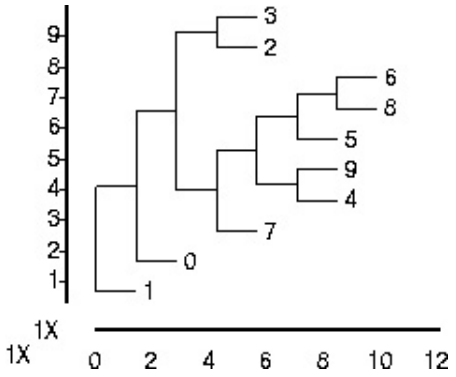
$$d = \sqrt{\sum_i (x_i - y_i)^2}$$

där x_i är värdet på element i i den ena vektorn och y_i är värdet på samma element i den andra vektorn.

indata mönster. Exempelvis så skiljer sig ‘6’ och ‘8’ respektive ‘5’ och ‘6. på ett ställe, ‘5’ och ‘8’ däremot, skiljer sig på två.



Figur 1: **T.v.** ett klusterdigram över *indata*vektorerna. Det euklidiska avståndet mellan vektorn som t.ex. representerar siffran ’1’ respektive ’7’ är mindre i jämförelse med dess övriga kombinationer. De siffror som ryms under A är längre ifrån de som inryms under B respektive C. **T.h.** ett klusterdiagram över *utdata*vektorerna då vikterna är satta *slumpmässigt* ($U(-0.5, 0.5)$) vid 0 epoker, d.v.s. innan träning. Som synes finns det en betydande likhet mellan hur vektorerna grupperat sig i de bägge diagrammen! Detta är en systemegenskap som ANN besitter.

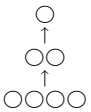


Figur 2: Klusterdiagram över *utdatalagret* efter 200 epoker. Nätet kan här ge en korrekt respons. Med korrekt respons avses: att de önskade noderna är mest aktiva, vanligen > 0.5 medan de övriga är < 0.5. Så trots att nätet ger rätt respons så finns en viss aktivitet i de noder som inte önskas vara aktiva. Den klusterhierarki som finns i utdatalagret vid 0 epoker återfinns, till viss del, även här, vilket visar att nätets generaliseringsförmåga kvarstår trots att det har lärt sig att diskriminera.

3.2 Abstraktion/Klassifikation

Ett exempel på hur ett ANN lär sig diskriminera m.a.p. numerositeten för talet '2'. Nätet ska således kunna klassificera ett stimulus med avseende på en egenskap, en egenskap som *inte* beror av de ingående särdragens *placering*. Dessa är ovidkommande, i detta fall, är det endast *antalet* som är av vikt.

Topologi 4 inputnoder kopplade till 2 dolda noder vilka är kopplade till 1 utnod, (4x2x1):



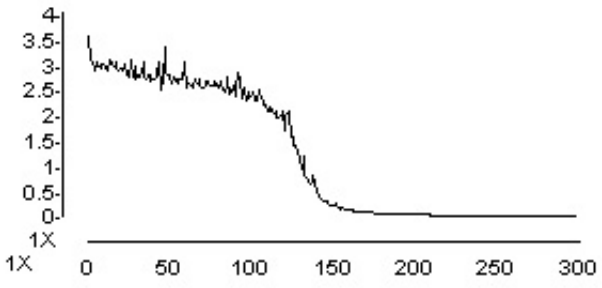
In-/Utdata representation och Träningsmängd Indatarepresentationen är distribuerad och utgörs av 4 noder vilket ger 16 möjliga indatamönster.[6] Utdatarepresentation är lokalistisk och utgörs av en nod, som när den är aktiv motsvarar numerositeten hos talet '2'. Då den ej är aktiv motsvarar det alla övriga fall, d.v.s. '0', '1', '3' och '4'. De två träningsexempel som är markerade (-) utgör inte en del av träningsmängden. Dock så är dessa två med då vi prövar om nätet har lärt sig att generalisera. D.v.s. om det klarar av att avgöra numerositeten hos aldrig tidigare presenterade indatamönster.

```
Indata   ->  0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
Utdata   ->   0    0    0    1    0    1    1    0    0    1    1    0    1    0    0    0
Numerositet->"0"  "1"  "1"  "2"  "1" -"2"  "2"  "3"  "1"  "2"  "2"  "3"  -"2" "3"  "3"  "4"
```

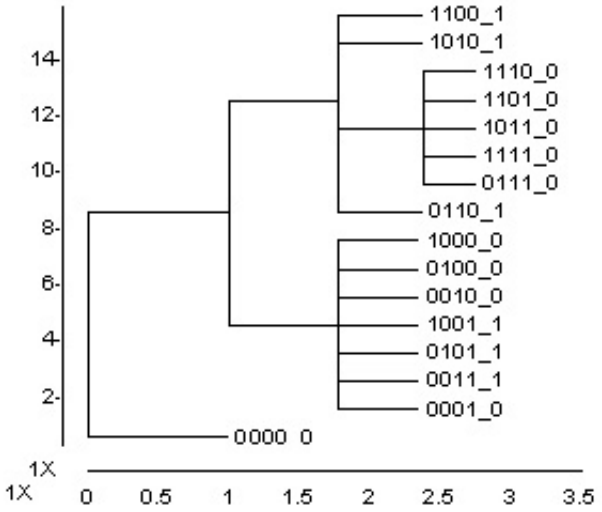
Träningsförfarande De 14 mönstren presenteras för nätet i permuterad ordning. Därefter testades det på alla möjliga kombinationer, det vill säga hela träningsmängden samt `[0101]` och `[1100]`.

[6] Det är 16 möjliga indatamönster förutsatt att vi endast tillåter en binär representation för respektive inputnod. I PDP++ kan inputmönster även anta värden i intervallet $[-1, 1]$ om så önskas.

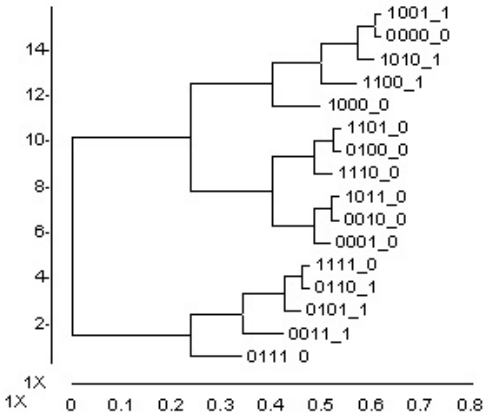
Resultat Mindre än 300 epoker var tillräckligt för att nätet skulle klara av uppgiften (fig. 3). Figur 4 - 6 visar nätets initiala likhetsstruktur före och efter träning. Se figurtext för vidare förklaring.



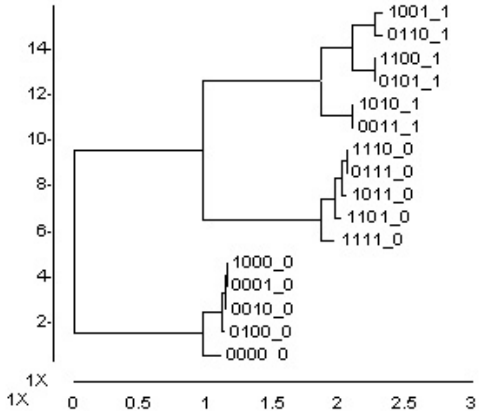
Figur 3: Felkurvan efter 300 epoker.



Figur 4: Klusterdiagram över de fyra indatanoderna då de presenteras för de 16 olika indatamönstren. De vektorer som är mest lika varandra har grupperats tillsammans. Av digrammet framgår att indatamönstren i sig innehåller viss information som redan från början är till gagn för nätet. Flera indatamönster grupperas dock helt felaktigt.



Figur 5: Klusterdiagram över de två dolda noderna med vikterna satta slumpmässigt ($U(-0.5, 0.5)$), vilket de är innan träning.



Figur 6: Klusterdiagram över de två dolda noderna efter 300 epoker. Tre distinkta kluster kan urskiljas. I det översta klustret har alla indata vektorer med två element grupperat sig. I det mittersta har alla indatavektorer med tre eller fyra element grupperat sig. Samt det nedersta, där alla övriga återfinns, det vill säga, de med ett eller inga element. Jämför detta med klusterdiagrammet över de de dolda noderna innan träning (figur 5). Detta trots att nätet inte ”tränats” på vektorerna `[0101]` och `[1100]`. Märk väl att nätet tränats på att diskriminera mellan vektorer innehållandes två element, trots detta har nätet grupperat de andra vektorerna på motsvarande sätt. Detta kan tillskrivas nätets generaliseringsförmåga. Nätet kan således abstrahera numerositeten ’2’ hos ett indatamönster.

3.3 Constraint Satisfaction

Ett centralt begrepp med avseende på *återkopplade* neurala nät är egenskapen 'constraint satisfaction'. Vad som avses är när aktivationstillstånd hos noder ömsesidigt påverkar varandras respektive aktivationstillstånd. Detta får till följd att aktivationstillstånden stabiliseras, över tid, på grund av den rådande viktuppsättningen och aktuell extern input. Notera att någon form av dämpande faktor (inhibition) måste ingå i systemet. Om inte så kommer aktiveringen att konstant öka tills ingående noder når sin maximala aktivering.

Ett exempel för att åskådliggöra 'constraint satisfaction' skulle kunna vara att släppa en kula i ett handfat. Kulan kommer, över tid, att stabilisera sig på handfatets lägsta punkt [7]

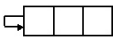
Den hamnar där som en konsekvens av alla de krafter (friktion, gravitation, kulans massa, handfatets utformning) som inverkar på densamma. Dessa krafter begränsar/avgör kulans trajektoria. På motsvarande sätt så utgör vikterna mellan de återkopplade noderna (och den externa aktivationen), de begränsningar som aktivationstillstånden måste satisfiera.

I BP++ och CS++ sker inhibering via negativa vikter. I Leabra++ kan inhibering också ske via negativa vikter men i huvudsak sker det med hjälp av en Winner-Take-All algoritm vid namn kWTA, vilken inhiberar hela nodlager.

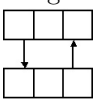
Ofta vill man modellera något fenomen som inbegriper försökspersoners reaktionstid. Detta görs lämpligen med något verktyg som är anpassat för att ta hänsyn till tid. Modulerna CS++ och Leabra++, i PDP++ paketet beräknar nätets aktivation på ett sätt som explicit tar hänsyn till temporala aspekter.[8] Nätet uppdateras ett antal cykler (`Cycles_To_Settle`) tills dess att nätets aktivationsskillnader mellan respektive cykel är under en viss nivå (eller `max` antal `cycles` uppnåtts). Variabeln `Cycles_To_Settle` kan här tolkas som nätets reaktionstid.

Några olika alternativ för att erhålla olika typer av 'constraint satisfaction', är följande.

a) Ett lager om tre noder återkopplat till sig själv (`Self Prjn`), vilket innebär att alla noder inom lagret är kopplade till alla övriga utom sig själva:



b) Ett lager om tre noder kopplat till ett annat och åter:



c) Olika kombination av a) och b).

[7] Vilket skulle utgöra en s.k. 'fixed point attractor', d.v.s. oavsett hur och varifrån kulan släpps, så kommer den att hamna i handfatets lägsta punkt. Det finns andra typer av rörelser i vektor rymden, såsom 'cyclic', MERA strange, chaos.

[8] Det går även att modellera temporala aspekter i BP++. Dock så är det omständligt, kräver mer arbete och tar längre tid.

3.3.1 Winner-take-all

Om man vill styra aktivationen i ett lager så kan man göra det på lite olika sätt. Ett sätt är att implementera en Winner-Take-All (WTA) mekanism. I Leabra++ så finns, för varje nodlager, parametern `k` som styr hur många noder (antal eller procent) som ska vara aktiva. Algoritmen kallas därför för `kWTA`. I CS++ finns inte denna algoritm att tillgå. Istället kan man erhålla ett liknande resultat genom att återkoppla ett lager till sig själv (`Self Prjn`, vilket innebär att alla noder inom lagret är kopplade till alla övriga utom sig själva). Därefter *låses* alla vikterna i projektionen till ett och *samma negativa* viktvärde. Detta får till följd att noderna (inom lagret) kommer att ”tävla” om att få uppnå sin respektive aktiveringsnivå. Den nod som initialt hade högst aktivation kommer att vinna. Detta genom att undertrycka alla de övriga, därav namnet, 'Winner-take-all'. En intressant egenskap som ett sådant lager uppvisar är bland annat att om fler noder i lagret initialt har hög aktivation kommer det att ta längre tid att utse en ”vinnare”.[9] Märk väl att med `kWTA`-algoritmen så sker denna dämpning momentant. Det är således en skillnad mellan att implementera en WTA funktion med negativa vikter och att använda sig av `kWTA`.

[9] McClelland, J.L., & Rumelhart, D.E. (1986). An Interactive Activation Model of Context Effects in Letter Perception: Part 1. An Account of Basic Findings. *Psychological Review*, *88*(5), 375-407.