Leren in netwerkmodellen 1b

Fleur Zeldenrust
Leren & Geheugen, 2017

Soorten leren

We kunnen 3 soorten leren onderscheiden:

1. **Unsupervised learning**: netwerk leert op basis van de input alleen (bijvoorbeeld receptive field)

2. **Supervised learning**: er is een leraar die vertelt hoe het netwerk het fout had (veel gebruikt in AI, misschien in cerebellum?)

3. **Reinforcement learning**: er is een (eventueel vertraagde) beloning of straf (bijvoorbeeld leren fietsen)

‘Eenvoudige' Hebb

Zorgt voor:

- cell assemblies / pattern completion
- leert correlaties in input

Problemen:

- Instabiel (gewichten blijven groeien)
- Alleen LTP, geen LTD (correlaties positief)
- lokale minima: ‘vergeten' nodig, of noisy updates
- instabiliteit patronen die neuronen 'delen'

Programma

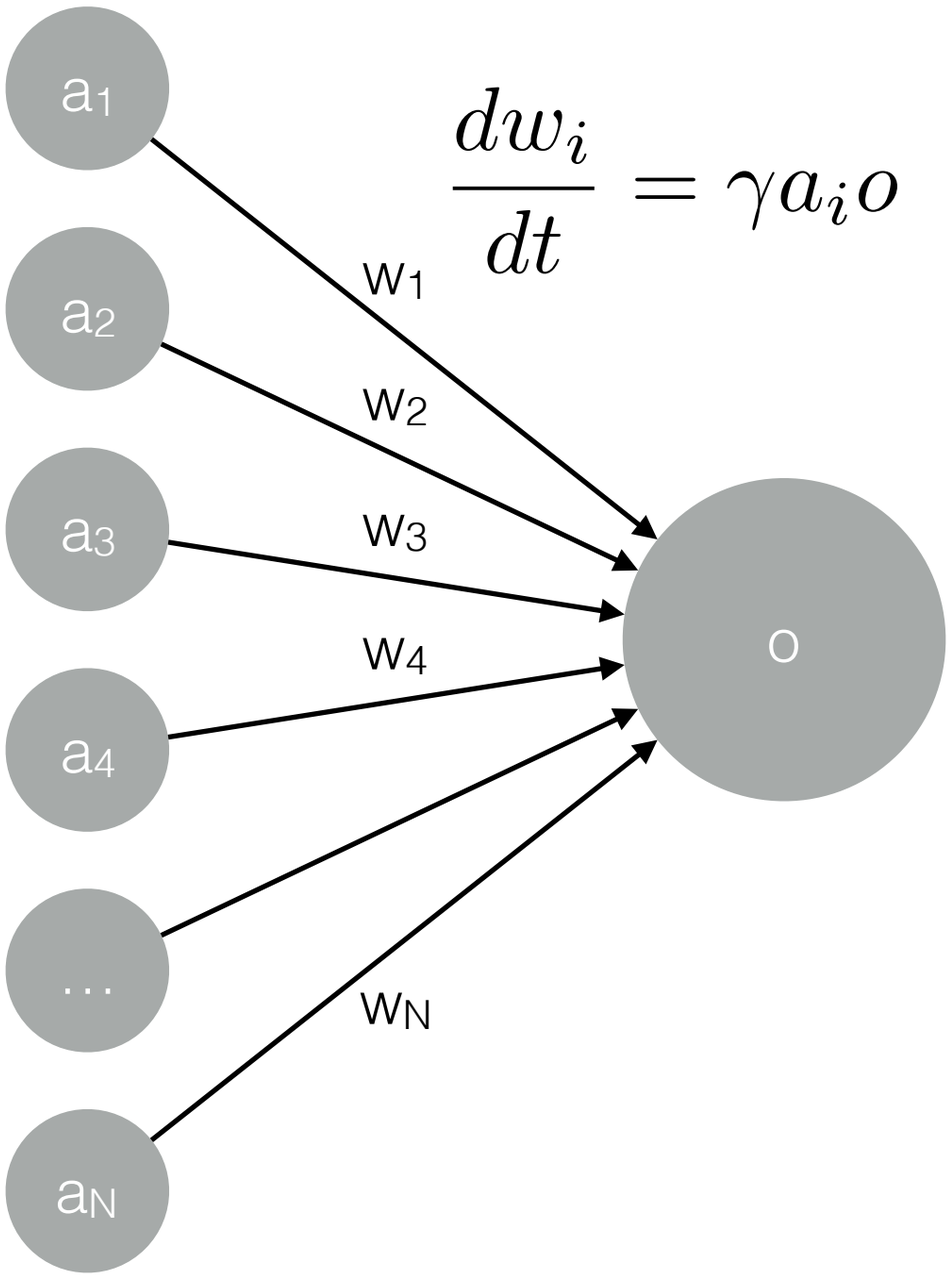
Vandaag

1. **Unsupervised learning**
    - LTP/LTD modellen: rate-based Hebbian learning
    - Andere ‘Hebb-achtige’ regels
    - STDP: spike-based Hebbian learning
2. **Supervised learning**

College 2

3. **Reinforcement learning**

LTD voor lage activiteit



$$\frac{dw_i}{dt} = \gamma a_i o$$

- Verander naïeve regel door drempel toe te voegen

$$\frac{dw_i}{dt} = \gamma \langle a_i (o - T_l) \rangle_{\text{patronen}}$$

- waar T_l een drempelwaarde is: eronder LTD, erboven LTP
- Let op! T_l is drempel voor leren, niet voor activatie neuron!
- Neem nu $T_l = \langle o \rangle$ (dus gemiddelde activiteit output neuron)
- Dan vinden we nu de **covariance rule**

Covariance rule

$$\frac{dw_i}{dt} = \gamma \sum_j C_{ij} w_j$$

- C is covariantie matrix

$$C_{ij} = \langle (a_i - \langle a_i \rangle)(a_j - \langle a_j \rangle) \rangle$$

- aanname: activiteiten a zijn constant in de tijd
- Hierbij is er dus wel depressie/LTD (want C kan negatief zijn)
- Maar: nog steeds **instabiliteit** (geen grens aan gewichten)

Maak leren stabiel

Maak een leerregel waarbij:

- Er LTD is (dus gewichten kunnen ook omlaag)
- Gewichten begrensd zijn (dus niet voorbij bepaalde waarde kunnen groeien)

Twee leerregels

- BCM regel
- Oja regel

Maak leren stabiel

Maak een leerregel waarbij:

- Er LTD is (dus gewichten kunnen ook omlaag)
- Gewichten begrensd zijn (dus niet voorbij bepaalde waarde kunnen groeien)

Twee leerregels

- BCM regel
- Oja regel

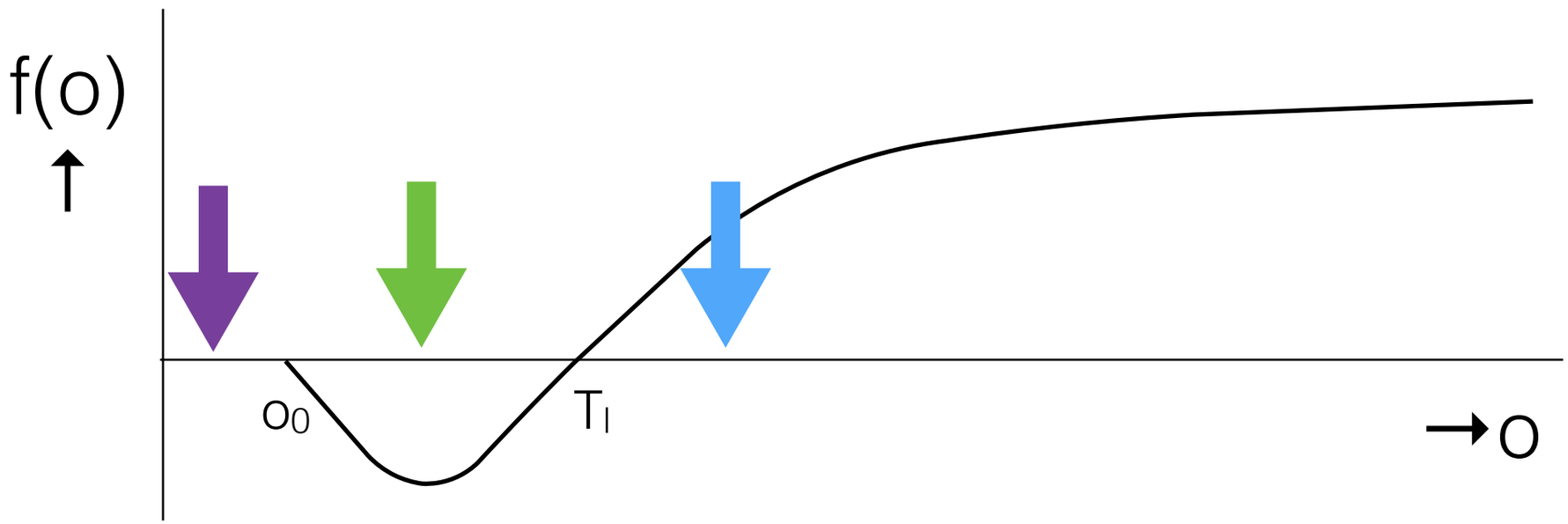
BCM rule: stabiel

$$\frac{dw_i}{dt} = \gamma a_i f(o - T_l)$$

Bienenstock, Cooper and Munro (1982)

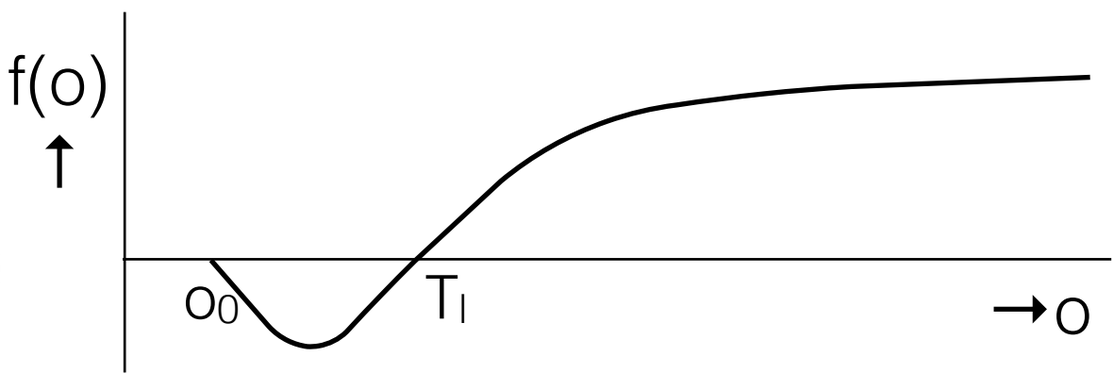
f is niet-lineaire functie van activiteit output-neuron o en drempel T_l

- $o < o_0$: geen leren
- $o_0 < o < T_l$: LTD
- $o > T_l$: LTP



BCM rule: stabiel

$$\frac{dw_i}{dt} = \gamma a_i f(o - T_l)$$



Bienenstock, Cooper and Munro (1982)

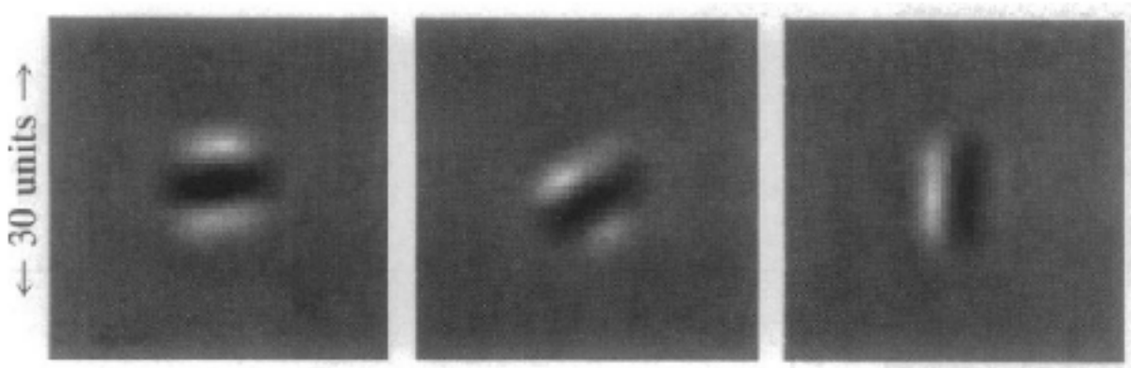
f is niet-lineaire functie van activiteit output-neuron o en drempel T

- Pas T aan om gewichten in te perken:
    - T neemt toe met gemiddelde postsynaptische vuurfrequentie o.
- Dit geeft **competitie** tussen synapsen: synaps wordt sterker → activiteit o output-neuron groter → T hoger → moeilijker om (andere) synapsen te versterken
- Dus: zorgt voor **selectiviteit**: output-neuron reageert op den duur alleen op één (set van) presynaptische neuronen

BCM theorie

Verklaring eigenschappen V1:

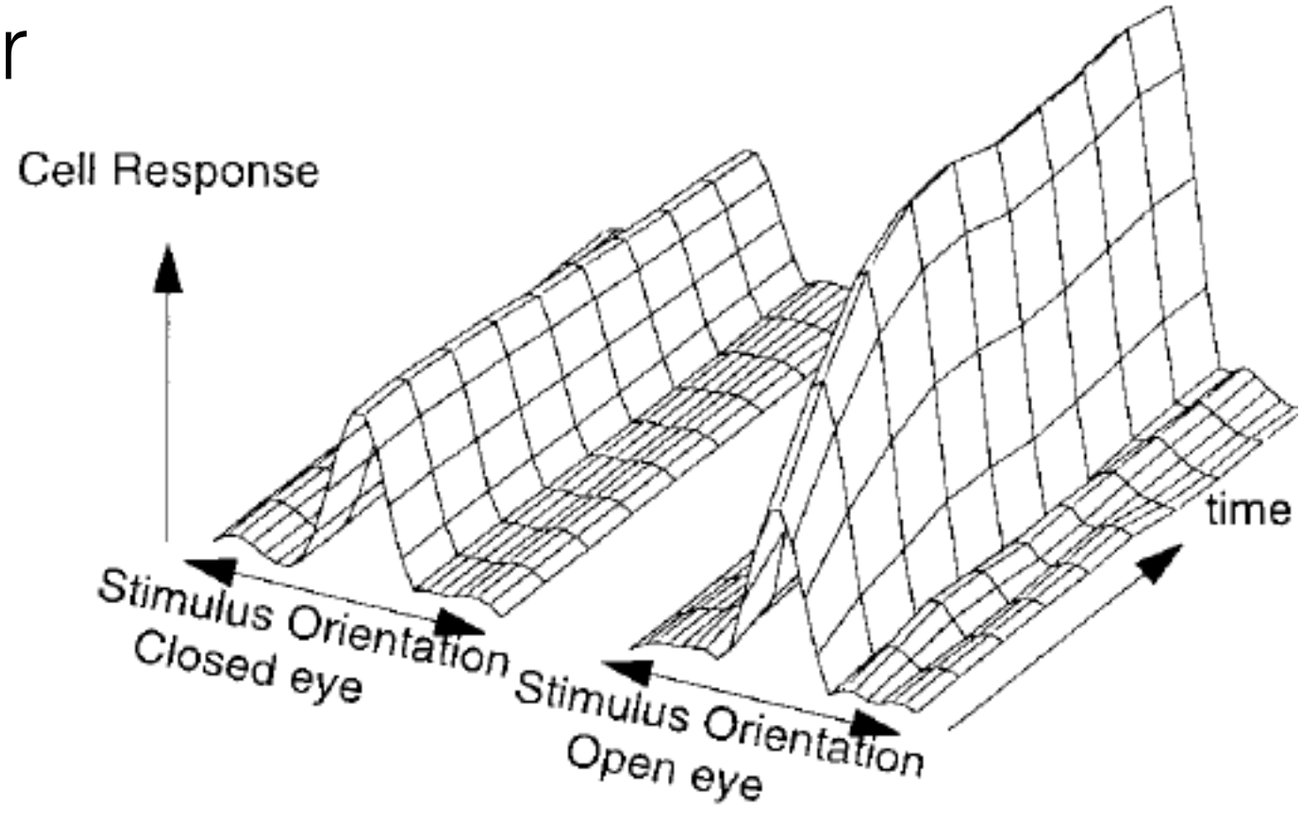
- 'realistische input' & BCM resulteert formatie 'realistische' receptive fields (direction selectivity) (Law & Cooper, 1994)



BCM theorie

Verklaring eigenschappen V1:

- Ocular dominance shift in deprivation en formatie ocular dominance columns kan verklaard worden door BCM (Blais et al 1999)



Maak leren stabiel

Maak een leerregel waarbij:

- Er LTD is (dus gewichten kunnen ook omlaag)
- Gewichten begrensd zijn (dus niet voorbij bepaalde waarde kunnen groeien)

Twee leerregels

- BCM regel
- Oja regel

Normalisatie van gewichten: Oja rule

$$\frac{dw_i}{dt} = \gamma(oa_i - \alpha o^2 w_i)$$

- Andere oplossing instabiliteit
- Aanname: kwadratische som van de gewichten blijft constant:

$$\sum_i |w|_i^2 = \frac{1}{\alpha}$$

- Dit geeft ook weer **competitie** tussen synapsen: als één toeneemt, moet de rest afnemen, en dus **selectiviteit**

Overzicht ‘Hebb’-achtige regels

- ‘Naïeve’ regel: alleen LTP (geen LTD), instabiel (synapsen stoppen nooit met groeien)

- Covariance regel: ook LTD, maar nog steeds instabiel

- BCM regel:
    - LTD & stabiel
    - perk gewichten in door dynamische drempel T
    - competitie en selectiviteit

- Oja regel:
    - LTD & stabiel
    - kwadratische som gewichten constant
    - competitie en selectiviteit

Conclusie unsupervised Hebbian learning

Rate based

- Klassiek Hebbiaans leren zorgt voor **cell assemblies** en **pattern completion**
- 'Naïef' Hebb leerregels geven gewichten die een reflectie zijn van correlaties in de input, maar is **instabiel**
- Een activiteit-afhankelijke drempel (BCM rule) of normalisatie gewichten (Oja rule) stabiliseert gewichten en zorgt voor **competitie** en **selectiviteit**

Programma

Vandaag

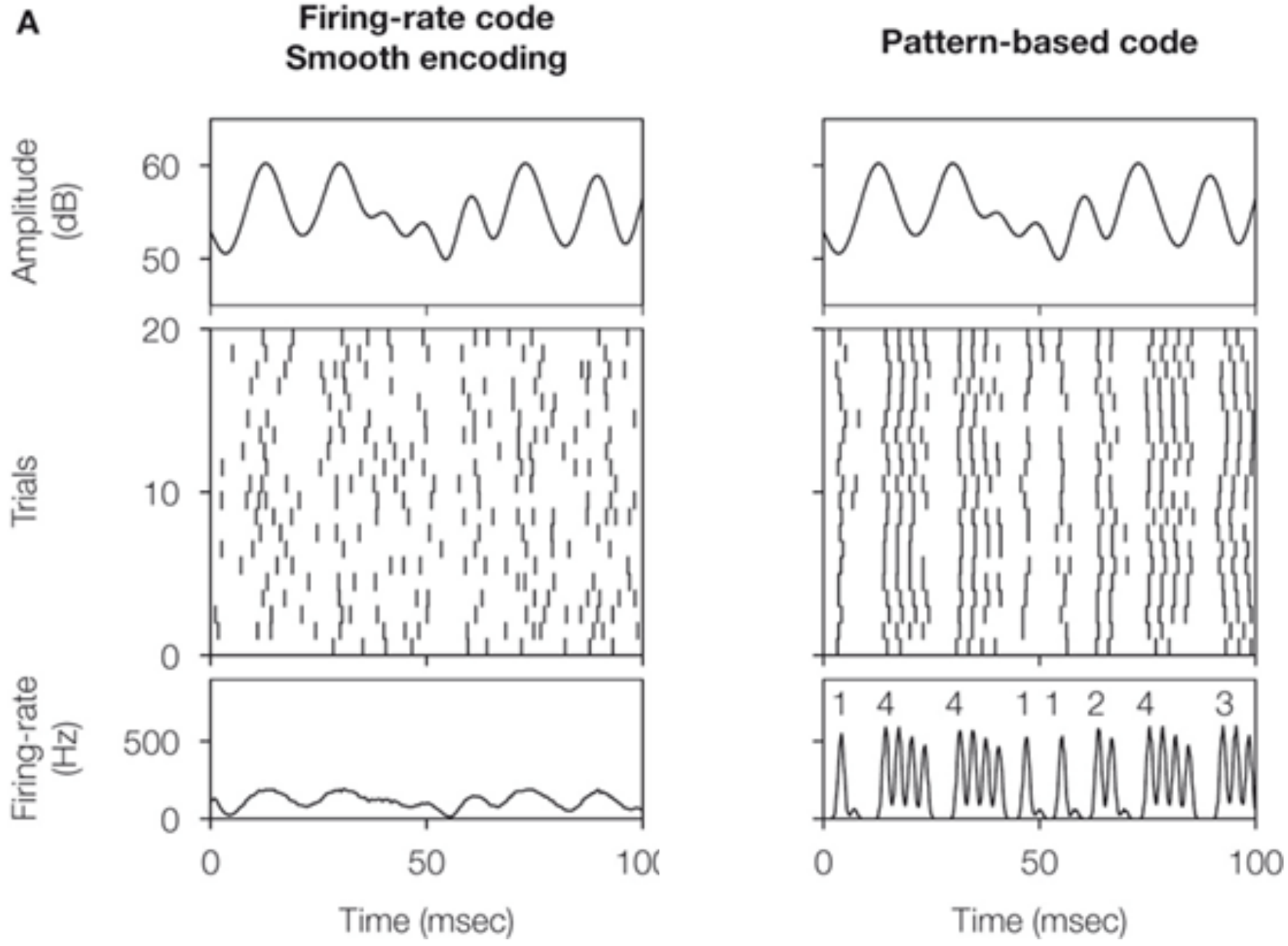
1. **Unsupervised learning**
    - LTP/LTD modellen: rate-based Hebbian learning
    - Andere ‘Hebb-achtige’ regels
    - STDP: spike-based Hebbian learning
2. **Supervised learning**

College 2

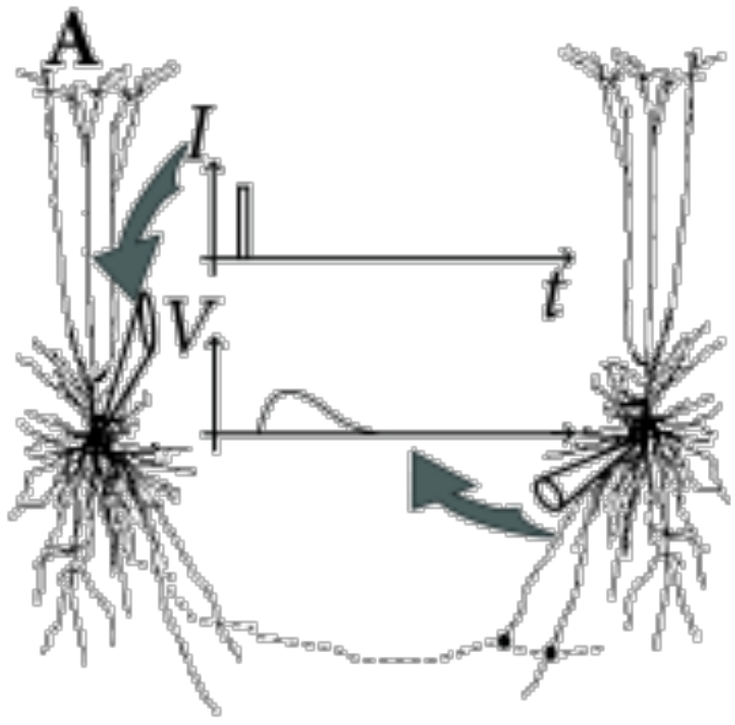
3. **Reinforcement learning**

Coding

- rate
- temporal
- Hoe leer je een temporal code?

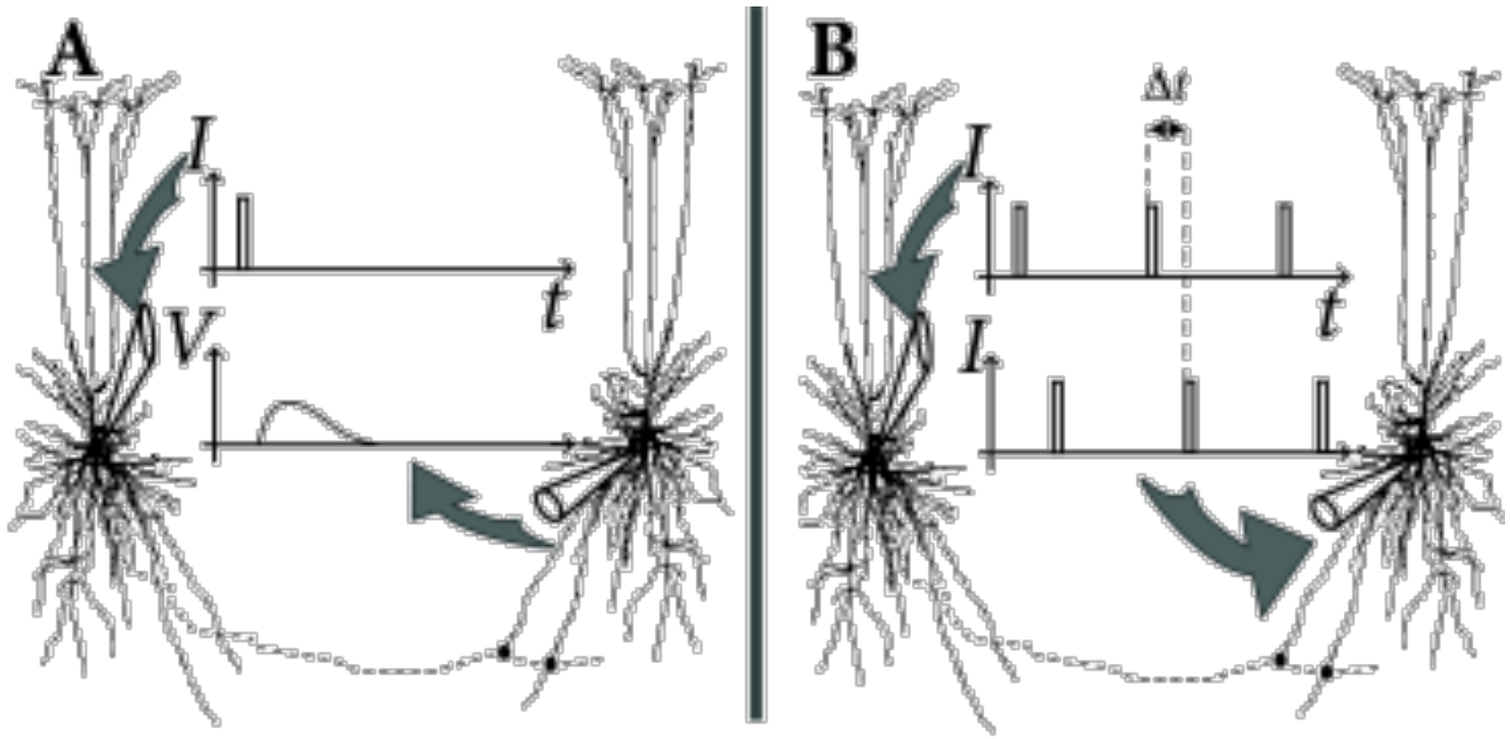


Spike Timing Dependent Plasticity



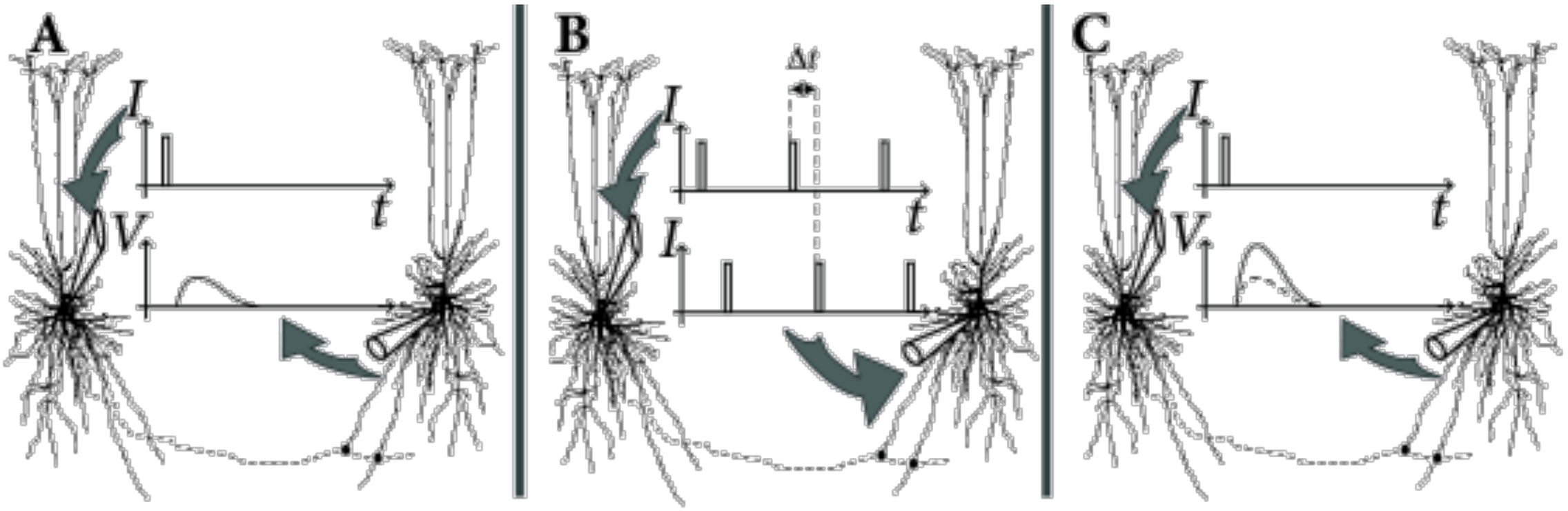
Als je cel A stimuleert zie je EPSP in cel B: verbonden

Spike Timing Dependent Plasticity



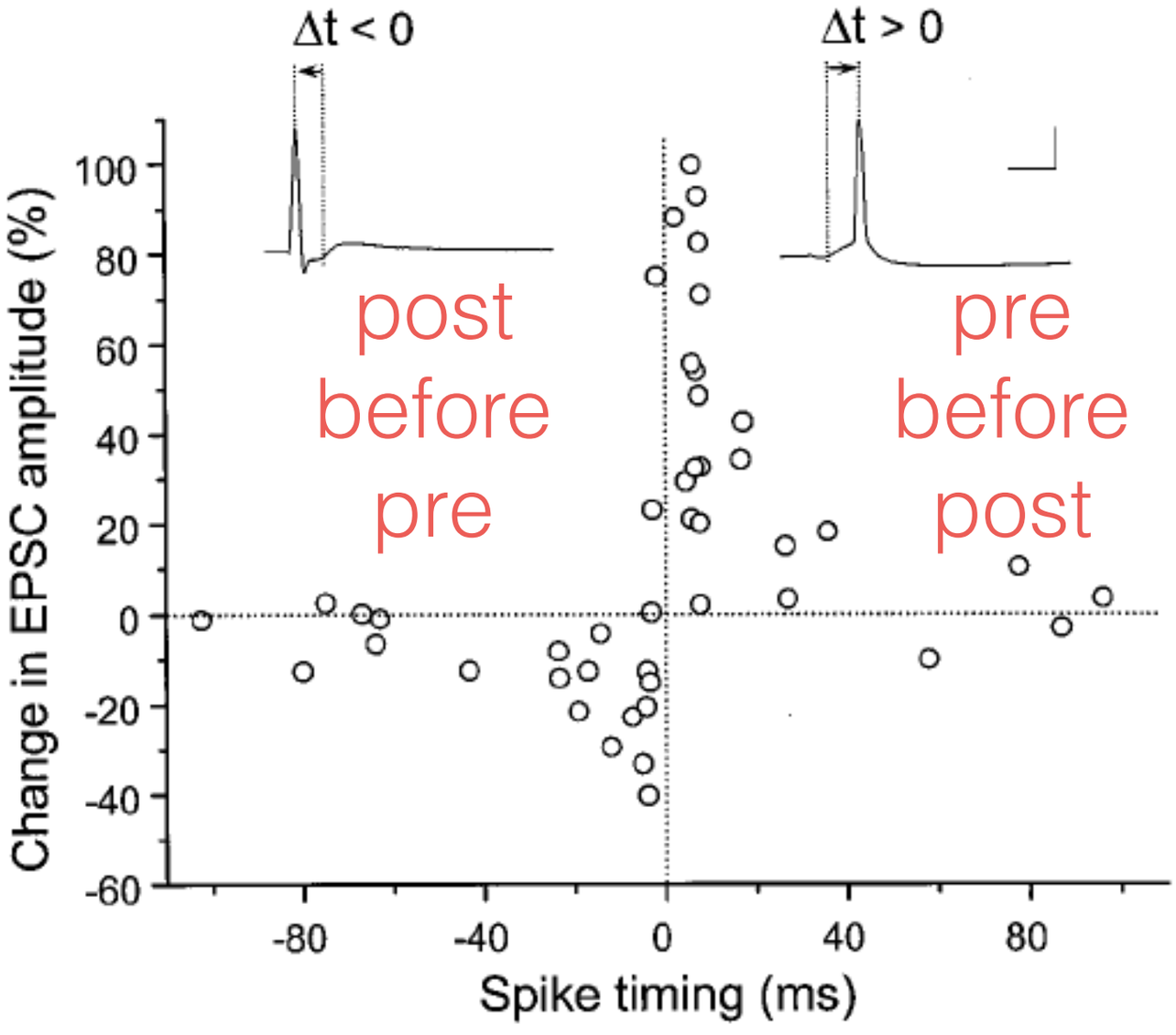
Als je cel A stimuleert zie je EPSP in cel B: verbonden
Stimuleer steeds beide cellen en varieer tijd tussen stimulatie

Spike Timing Dependent Plasticity



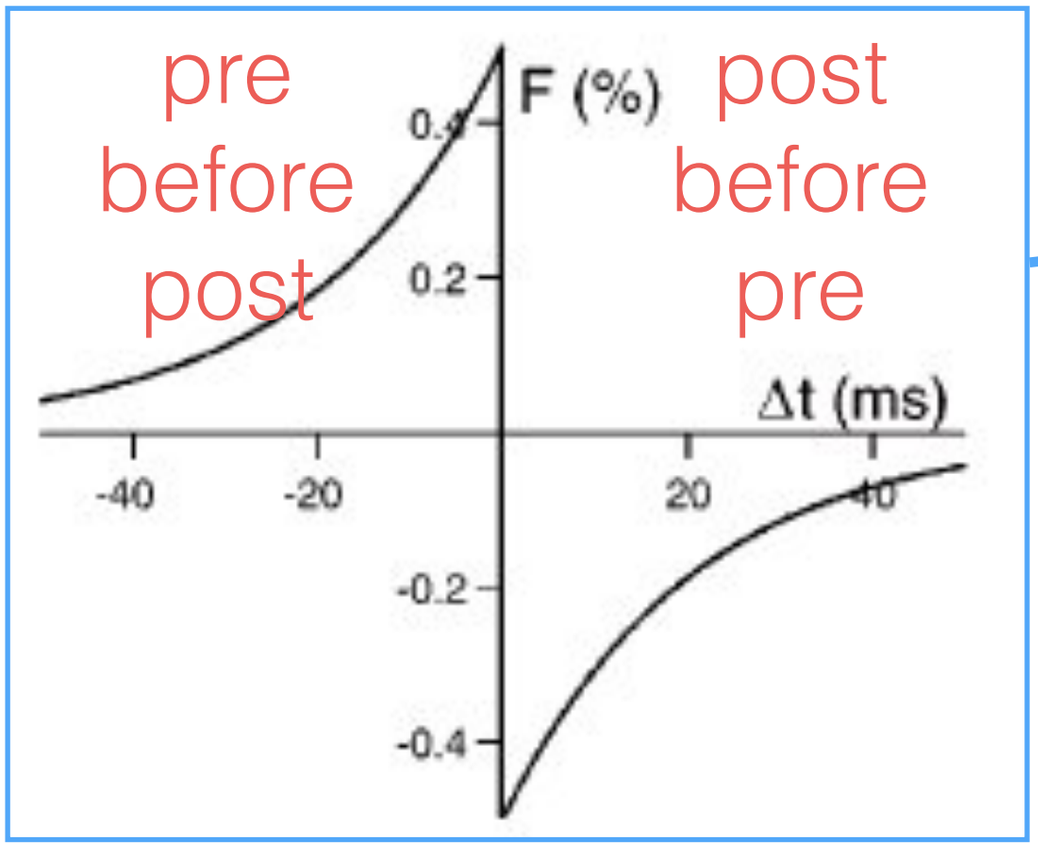
Als je cel A stimuleert zie je EPSP in cel B: verbonden
Stimuleer steeds beide cellen en varieer tijd tussen stimulatie
Is de synaps-sterkte tussen de cellen veranderd?

Spike Timing Dependent Plasticity



Bi&Poo, 1998

Model: Pairwise STDP



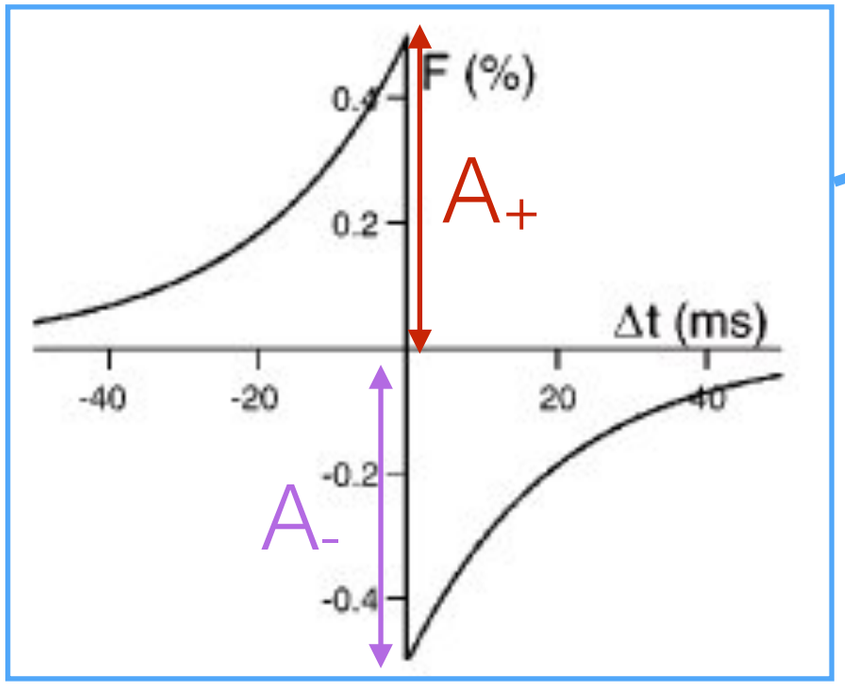
Let op: Δt soms ‘andersom' gedefinieerd!

‘STDP window’
F(Δt)

$$\Delta w_{ij} = \sum_{\substack{\text{spikes} \\ \text{neuron } i}} \sum_{\substack{\text{spikes} \\ \text{neuron } j}} F(\Delta t)$$

$$F(\Delta t) = \begin{cases} A_+ \exp(-|\Delta t|/\tau_+) & \text{if } \Delta t < 0 \\ -A_- \exp(-|\Delta t|/\tau_-) & \text{if } \Delta t > 0 \end{cases}$$

Model: Pairwise STDP



$$\Delta w_{ij} = \sum_{\substack{\text{spikes} \\ \text{neuron } i}} \sum_{\substack{\text{spikes} \\ \text{neuron } j}} F(\Delta t)$$

'STDP window' F(Δt)

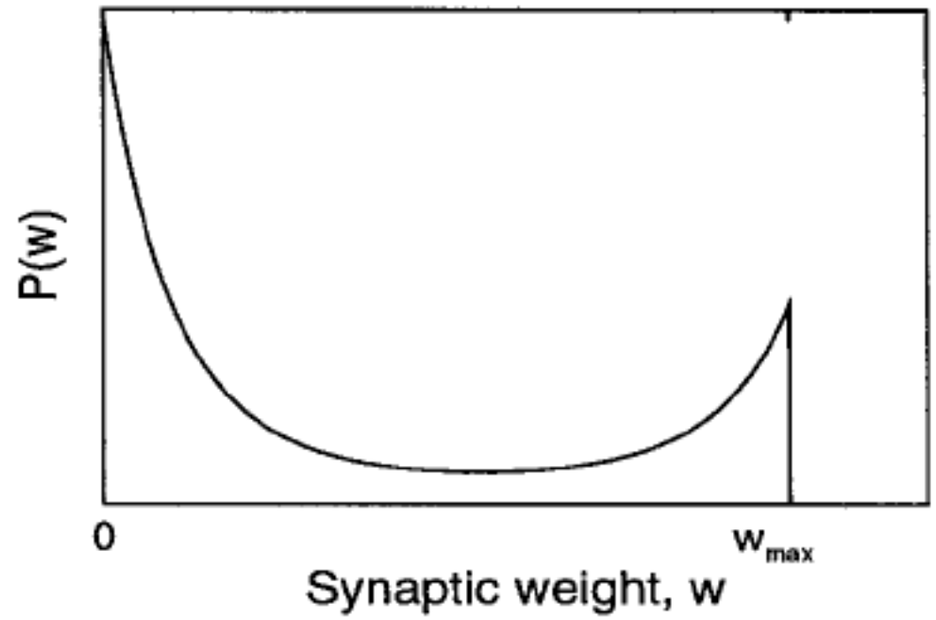
$$F(\Delta t) = \begin{cases} A_+ \exp(-|\Delta t|/\tau_+) & \text{if } \Delta t < 0 \\ -A_- \exp(-|\Delta t|/\tau_-) & \text{if } \Delta t > 0 \end{cases}$$

- kijk steeds naar één post-synaptische en één pre-synaptische spike
- tijd ertussen Δt bepaalt sterkte/richting verandering gewicht
- sterkte verandering gewicht (A_+ en A_-) kan ook van de synaps-sterkte afhangen:
  - zo niet: **additive** STDP (harde grens)
  - zo wel: **multiplicative** STDP (zachte grens)

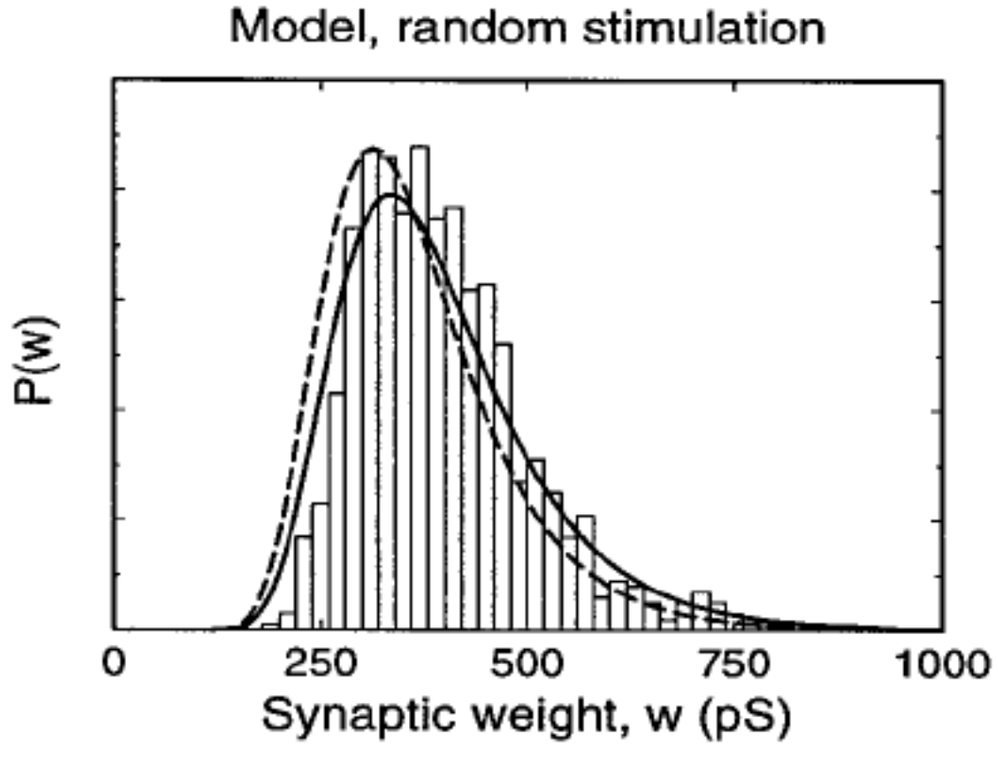
Additive en multiplicative STDP

van Rossum et al 2000

- Additive STDP (harde grens)
  - Als $w>w_{max}$: $\Delta w=0$
  - resultaat: gewichten 0 of w_{max}: bimodale verdeling
  - competitie



- Multiplicative STDP (zachte grens)
  - $\Delta w=A^*(w_{max}-w)$
  - resultaat: gewichten unimodale verdeling
  - geen competitie

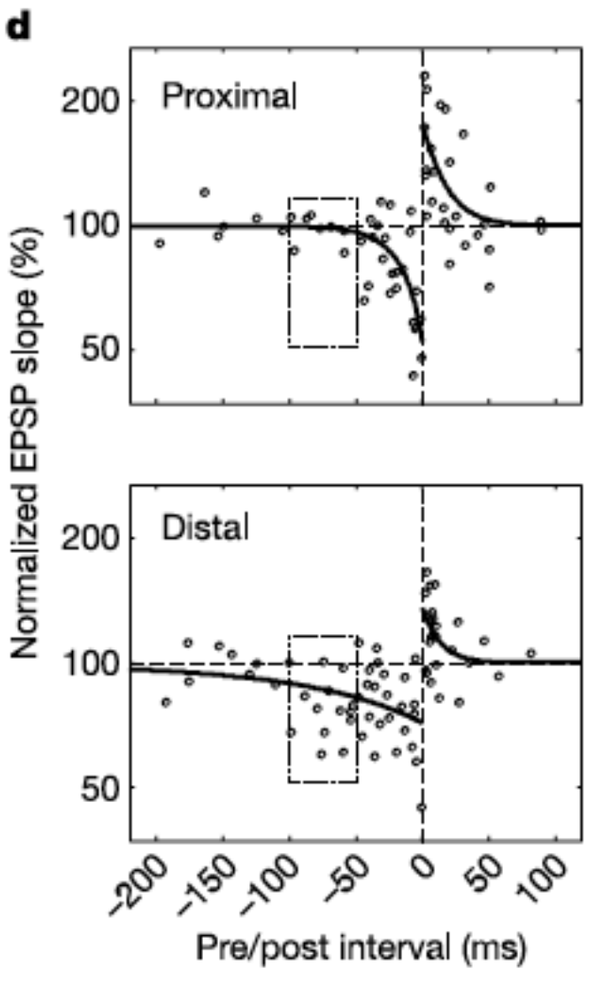


STDP window: hangt af van

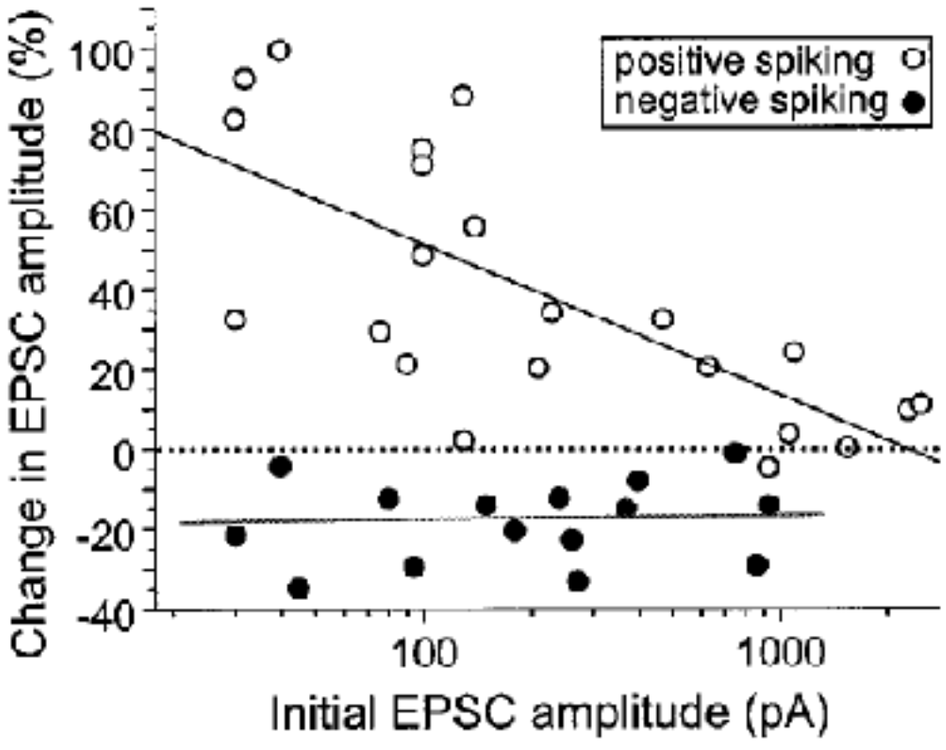
locatie op dendriet

sterkte synaps

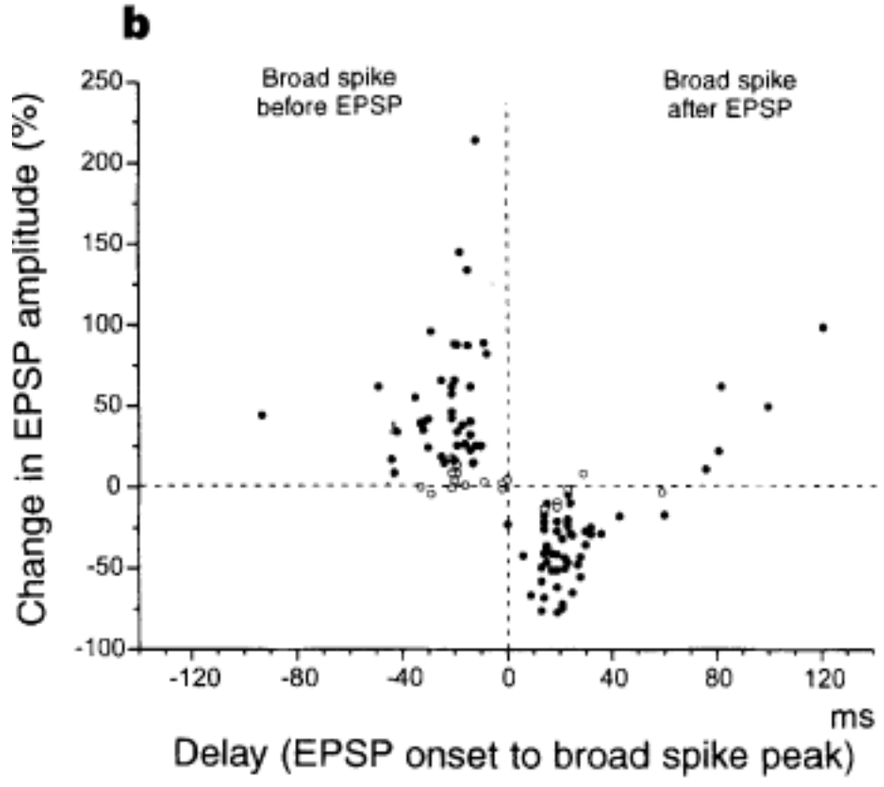
locatie brein



Froemke et al. 2005



Bi&Poo, 1998



Purkinje cells, Bell et al. 1997

Gevolg STDP leren

- Conclusies van rate based learning gelden nog steeds: geleerde gewichten geven correlaties input weer

- Maar STDP heeft nu een tijd-component toegevoegd: als neuronen ongeveer tegelijkertijd vuren, wordt zowel LTP als LTD sterker

- STDP kan gebruikt worden voor modellen die **temporal coding** gebruiken

Conclusie unsupervised (Hebbian) learning

Rate based

- Klassiek Hebbiaans leren zorgt voor **cell assemblies** en **pattern completion**
- Klassieke Hebb leerregels geven gewichten die een reflectie zijn van correlaties in de input, maar is **instabiel**
- Een activiteit-afhankelijke drempel (BCM rule) of normalisatie gewichten (Oja rule) stabiliseert gewichten en zorgt voor **competitie**

Spike based

Spike Timing-Dependent Plasticity

- additive (harde grens, competitie, bimodale distributie)
- multiplicative (zachte grens, geen competitie, unimodaal)

Wat moet je kennen?
Unsupervised learning

- ‘Naïeve' Hebb regel
    - toegepast op Hopfield netwerk
- Covariance regel
- BCM regel
- Oja regel
- STDP
    - learning window
    - additief
    - multiplicatief

Programma

Vandaag

1. **Unsupervised learning**

2. **Supervised learning**
   - perceptron regel
   - delta-regel
   - error-backpropagation

College 2

3. **Reinforcement learning**

Leren

- **unsupervised learning:**
  - op basis van correlaties in input (receptive fields)
  - classificatie, herkennen: heb ik deze input al eens gezien?

- **supervised learning:**
  - op basis van feedback van omgeving
  - hoe kun je een (kunstmatig) systeem probleem laten oplossen in interactie met omgeving?
  - Draait allemaal om ‘beloning' (‘reward’ en ‘punishment/straf’), acties zijn niet neutraal
  - Veel gebruikt in AI / informatica!

Supervised learning

- Wat als je een netwerk wilt trainen voor een bepaalde taak?
    - curve fitting (statistiek, data analyse)
    - spraakherkenning (Alexa, Siri)

- Je wilt elk input-patroon koppelen aan een output-patroon
- Je geeft steeds aan of en hoe het output-patroon correct was of niet

koppeling 1

input	netwerk output	gewenste output	verschil
a	o	t	δ
0	0	1	+1
1	0	0	0
0	1	1	0
1	1	0	-1

koppeling 2

input	netwerk output	gewenste output	verschil
a	o	t	δ
0	1	0	-1
0	0	0	0
0	0	0	0
1	1	1	0

Programma

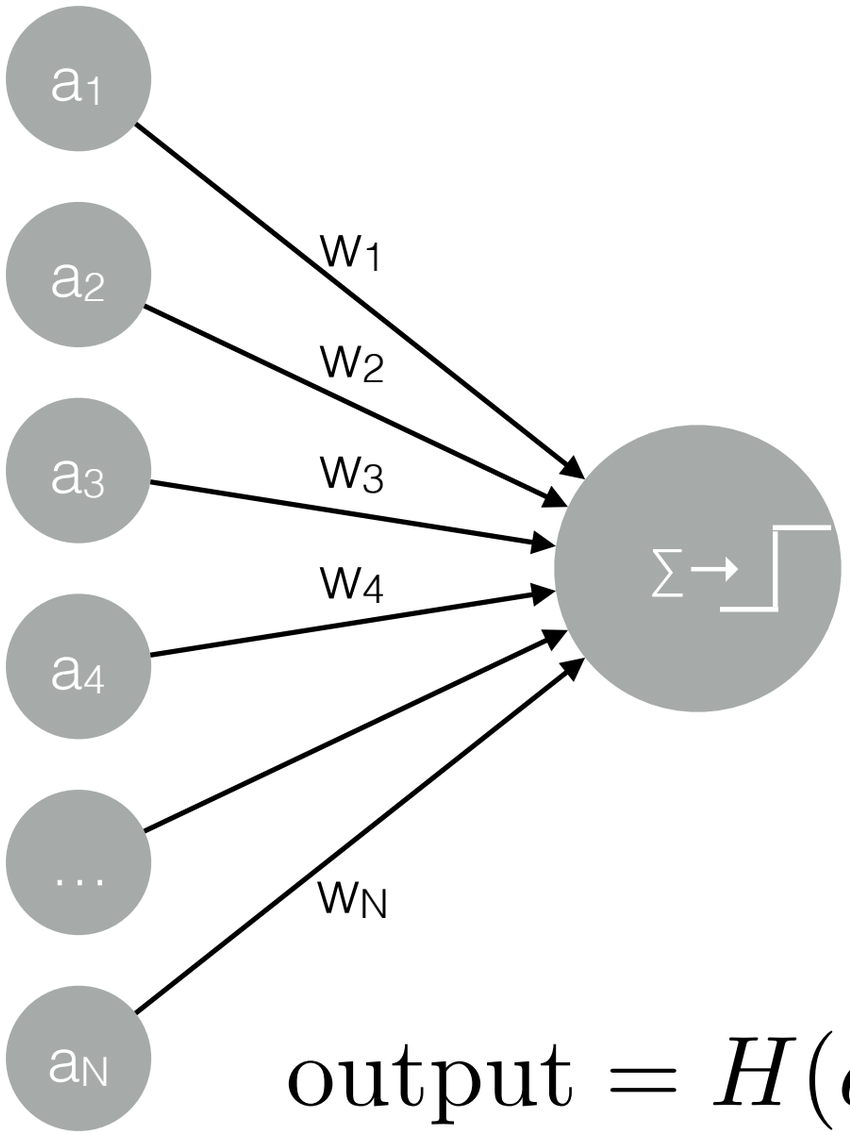
Vandaag

1. **Unsupervised learning**

2. **Supervised learning**
   - perceptron regel
   - delta-regel
   - error-backpropagation

College 2

3. **Reinforcement learning**



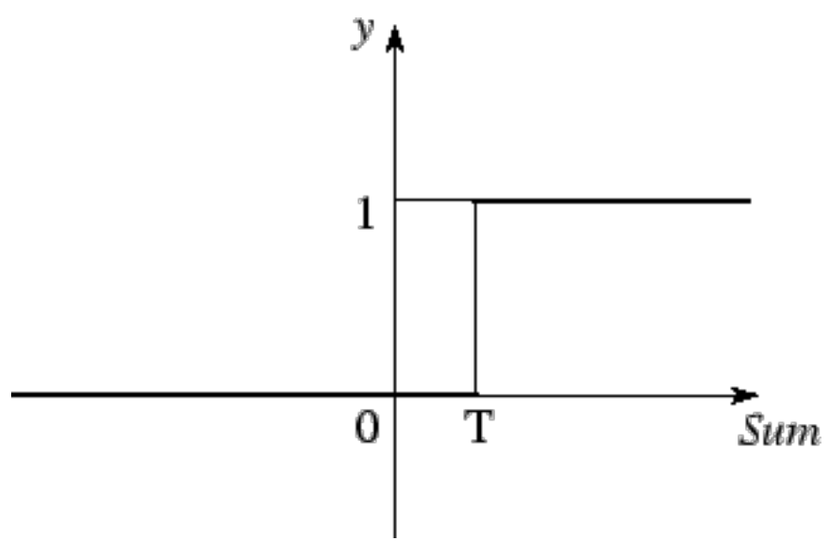
Herhaling: Perceptron

Input-activiteit a_i 0 of 1
Output-activiteit o 0 of 1

$$\text{output} = H(a_1 * w_1 + a_2 * w_2 + ... + a_N * w_N - T)$$

$$= H(\sum_{n=1}^{N} a_n w_n - T)$$

Heaviside step y =H(x-T)

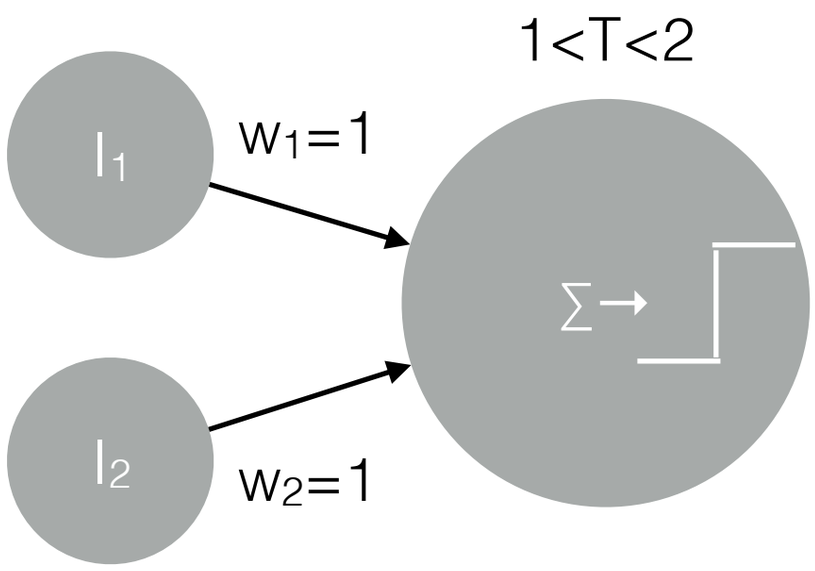


Booleaanse logica met MPN

- Kan elke Booleaanse functie oplossen, mits het meerdere lagen heeft

- Hier bijvoorbeeld AND

- Hoe worden gewichten geleerd?

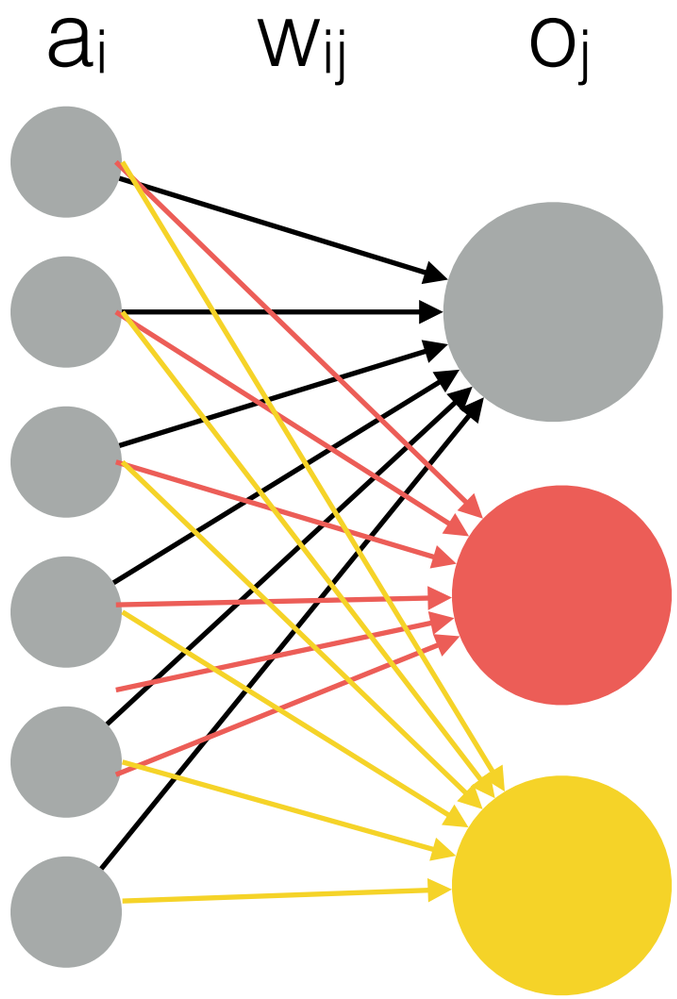


input 1	input 2	output
1	1	1
1	0	0
0	1	0
0	0	0

Perceptron regel

Hoe leer ik de gewichten w?

1. Geef de input a
2. Bereken de output o
3. Vergelijk voor elk output neuron de output o met de gewenste output t
    A. gelijk: doe niets
    B. Als 1 (had 0 moeten zijn): verlaag gewichten
    C. Als 0 (had 1 moeten zijn): verhoog gewichten



$$w_{ij} \rightarrow w_{ij} + \Delta w_{ij}$$

$$\Delta w_{ij} = \gamma(t_j - o_j)a_i$$

Perceptron regel

- Je geeft steeds aan of en hoe het output-patroon correct was of niet

- Dus: verander gewichten met δ (verschil netwerk output en gewenste output)

koppeling 1

input	netwerk output	gewenste output	verschil
a	o	t	δ
0	0	1	+1
1	0	0	0
0	1	1	0
1	1	0	-1

$$w_{ij} \rightarrow w_{ij} + \Delta w_{ij}$$
$$\Delta w_{ij} = \gamma(t_j - o_j)a_i$$

Perceptron regel

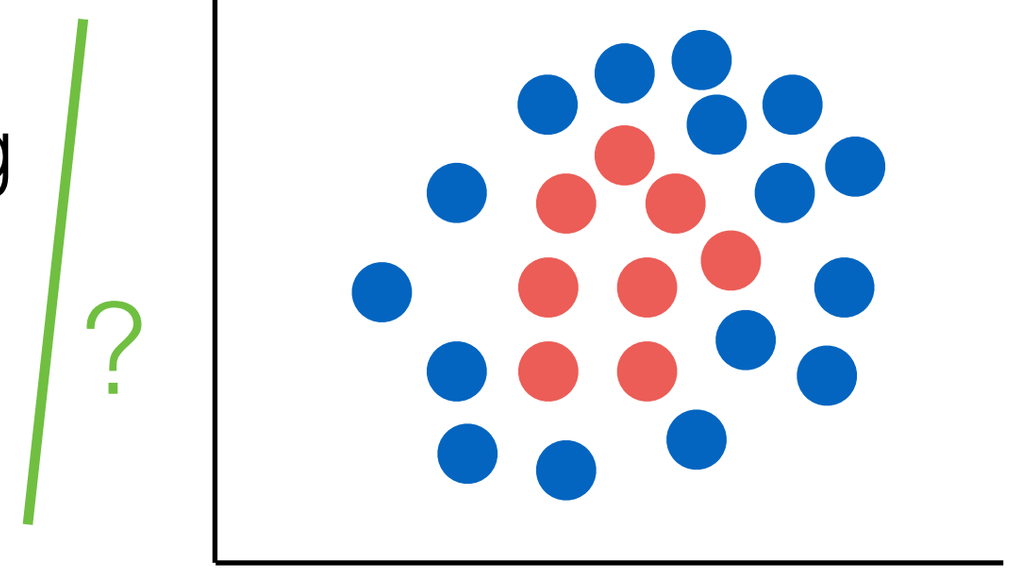
- Je kunt wiskundig aantonen dat de perceptron regel altijd de oplossing vindt, *als die bestaat*

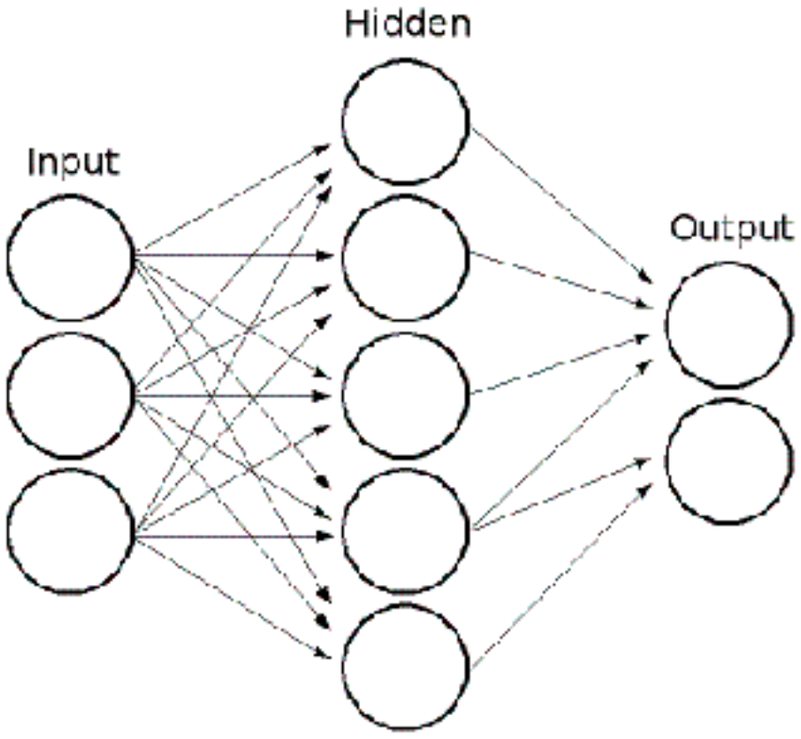
- Bestaan alle oplossingen voor enkellaags perceptron?

- Nee! (XOR, lineair niet separabel, zie: Perceptie)

- Hoe kon je dit oplossen?

- ‘Verborgen' laag





Supervised learning

- Perceptron regel werkt voor enkellaags perceptron, met binaire neuronen (McCulloch & Pitts neuron MPN)
  - hoe dan voor rate neuronen? → Delta rule
  - hoe leert de ‘verborgen laag’ → Error-backpropagation

Programma

Vandaag

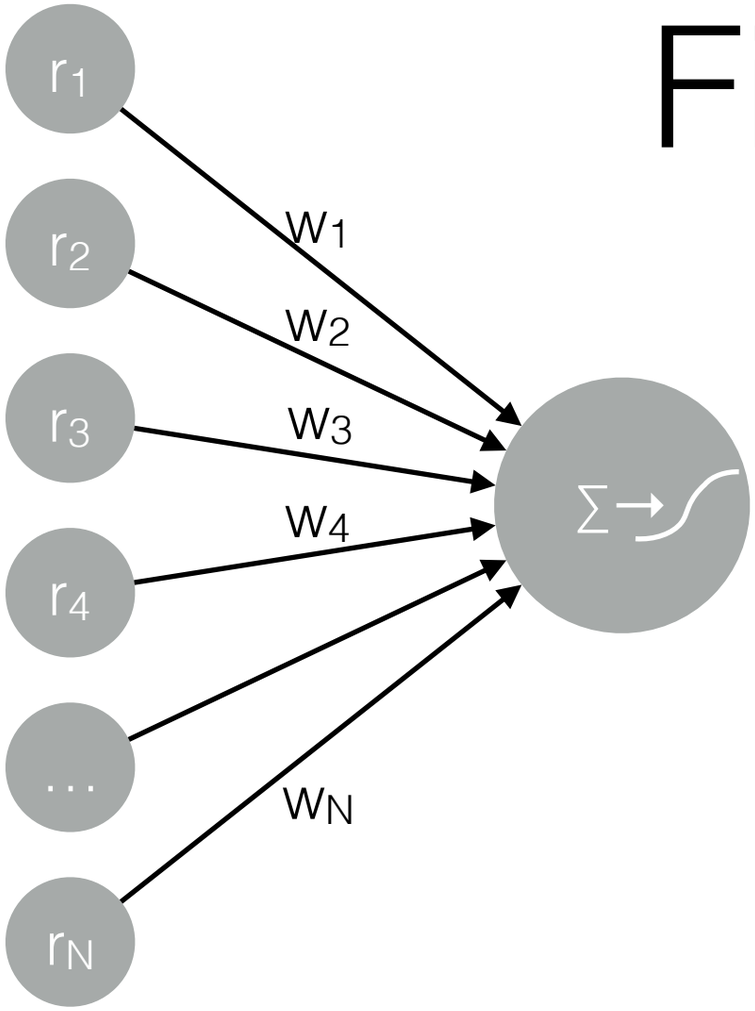
1. **Unsupervised learning**

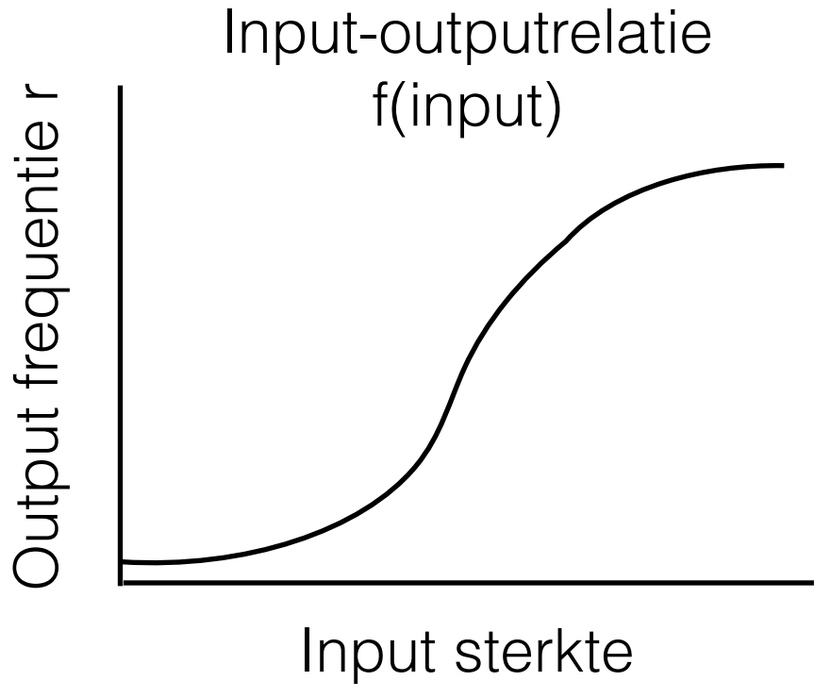
2. **Supervised learning**
   - perceptron regel
   - delta-regel
   - error-backpropagation

College 2

3. **Reinforcement learning**

Firing Rate Neuron (FRN)





$$\tau \frac{dr_{\text{output}}(t)}{dt} = f(h, \text{neuron parameters})$$

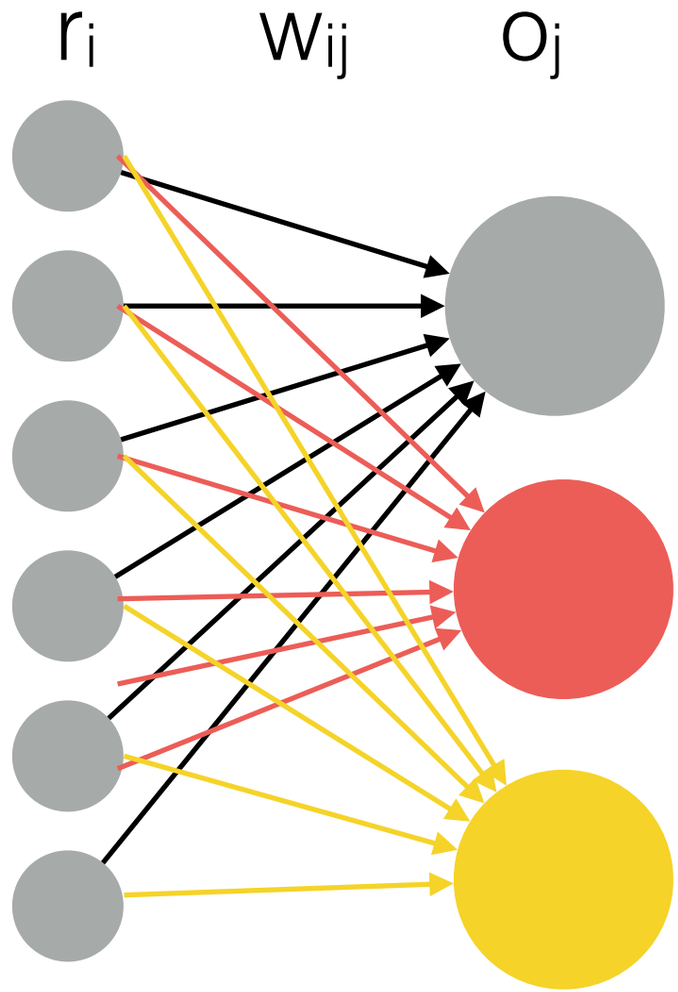
$$h = \sum_{n=1}^{N} r_n(t) * w_n$$

totale input naar output neuron

Delta regel

Hoe leer ik de gewichten w?

1. Geef de input r
2. Bereken de output o
3. Vergelijk voor elk output neuron de output o met de gewenste output t
    A. gelijk: doe niets
    B. Als te hoog: verlaag gewicht
    C. Als te laag: verhoog gewicht
4. Maar hou rekening met input-outputrelatie!



$$w_{ij} \rightarrow w_{ij} + \Delta w_{ij}$$

$$\Delta w_{ij} = \gamma(t_j - o_j)a_i$$

$$w_{ij} \rightarrow w_{ij} + \Delta w_{ij}$$

$$\Delta w_{ij} = \gamma(t_j - o_j)r_i \frac{\partial f(h_j)}{\partial h_j}$$

Delta regel

- Waar komt die afgeleide vandaan?

- Elke keer dat je gewichten update verklein je de **error**

$$E = \sum_j \frac{1}{2}(t_j - o_j)^2 = \sum_j \frac{1}{2}(t_j - f(h_j))^2$$

- Je kunt bewijzen: gewichten veranderen niet meer op minimum error-functie (niet in tentamen, bewijs einde slides)

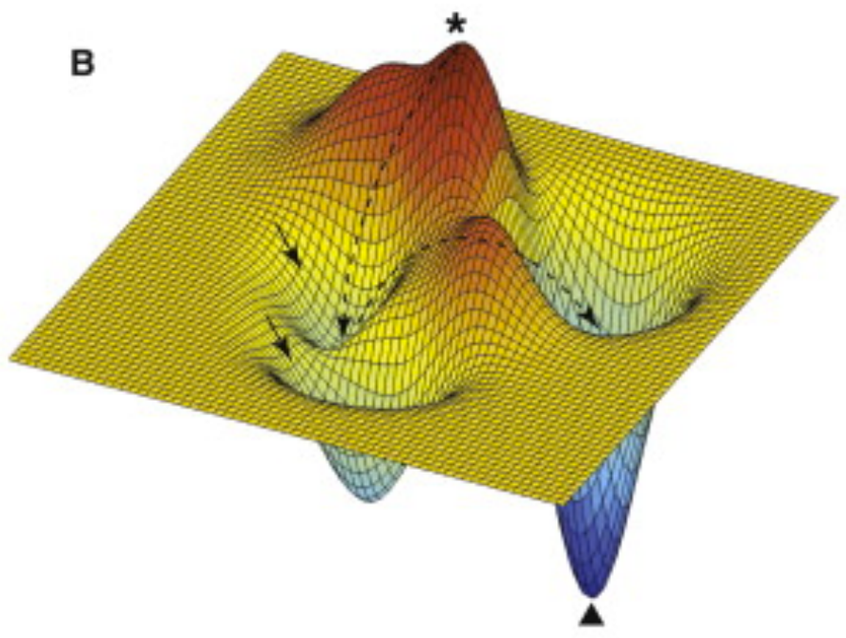
- Dus heel effectieve leerregel: error (verschil echte en target output) zo klein mogelijk!

- Waar lijkt dit op?

Hopfield netwerk	**Delta regel leren**
Minimaliseren Energie	Minimaliseren Error
Updates: activiteit neuronen	Updates: sterkte gewicht
Probleem: lokale minima (‘valse herinneringen’)	Probleem: lokale minima (niet beste oplossing)

Lokale minima bij Delta regel: je vindt niet de ‘beste’ oplossing (kleinste error), maar blijft ‘hangen’ in een minder goede oplossing

Daardoor geef je niet bij elke input de juiste output



Delta regel

- Problemen:
    - Kan blijven hangen lokale minima
    - Biologisch onrealistisch (target = vuurfrequentie *elk neuron*)
    - Enkele laag kan geen lineair separabele problemen oplossen. Hoe leer je verborgen laag?

Programma

Vandaag

1. **Unsupervised learning**

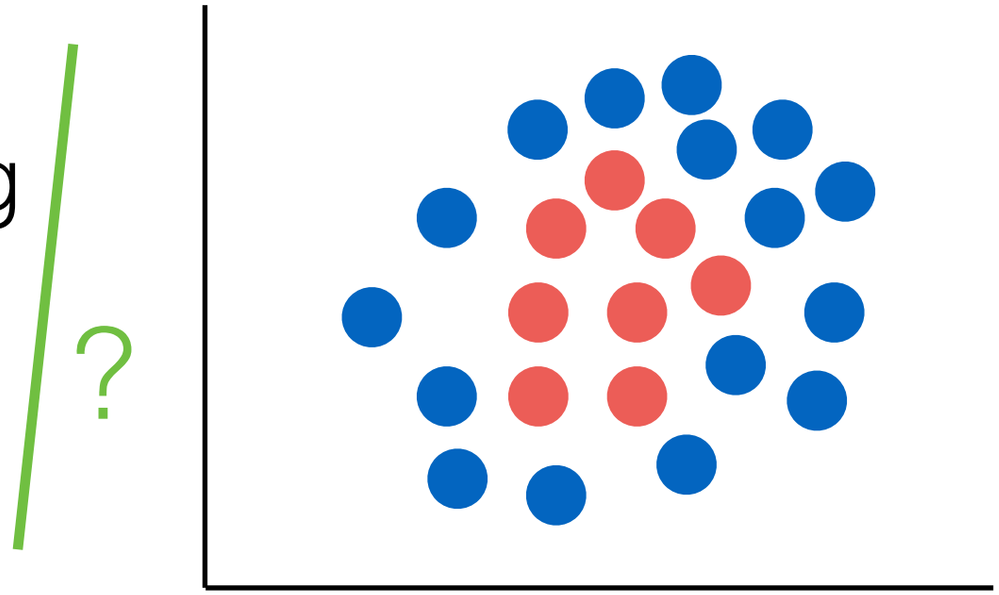
2. **Supervised learning**
   - perceptron regel
   - delta-regel
   - error-backpropagation

College 2

3. **Reinforcement learning**

Perceptron regel

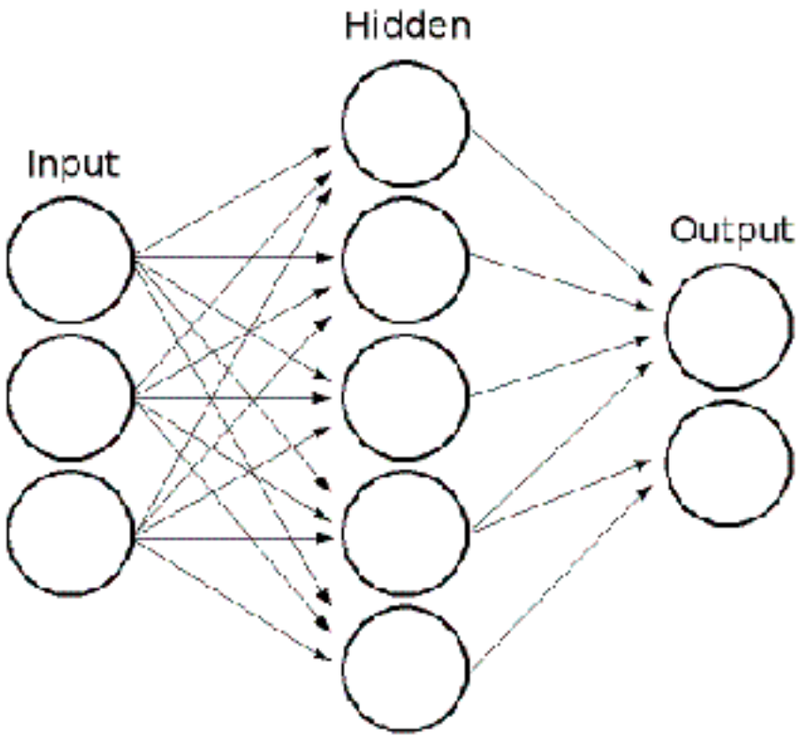
- Je kunt wiskundig aantonen dat de perceptron regel altijd de oplossing vindt, *als die bestaat*



- Bestaan alle oplossingen voor enkellaags perceptron?

- Nee! (XOR, lineair niet separabel, zie: Perceptie)

- ‘Verborgen' laag

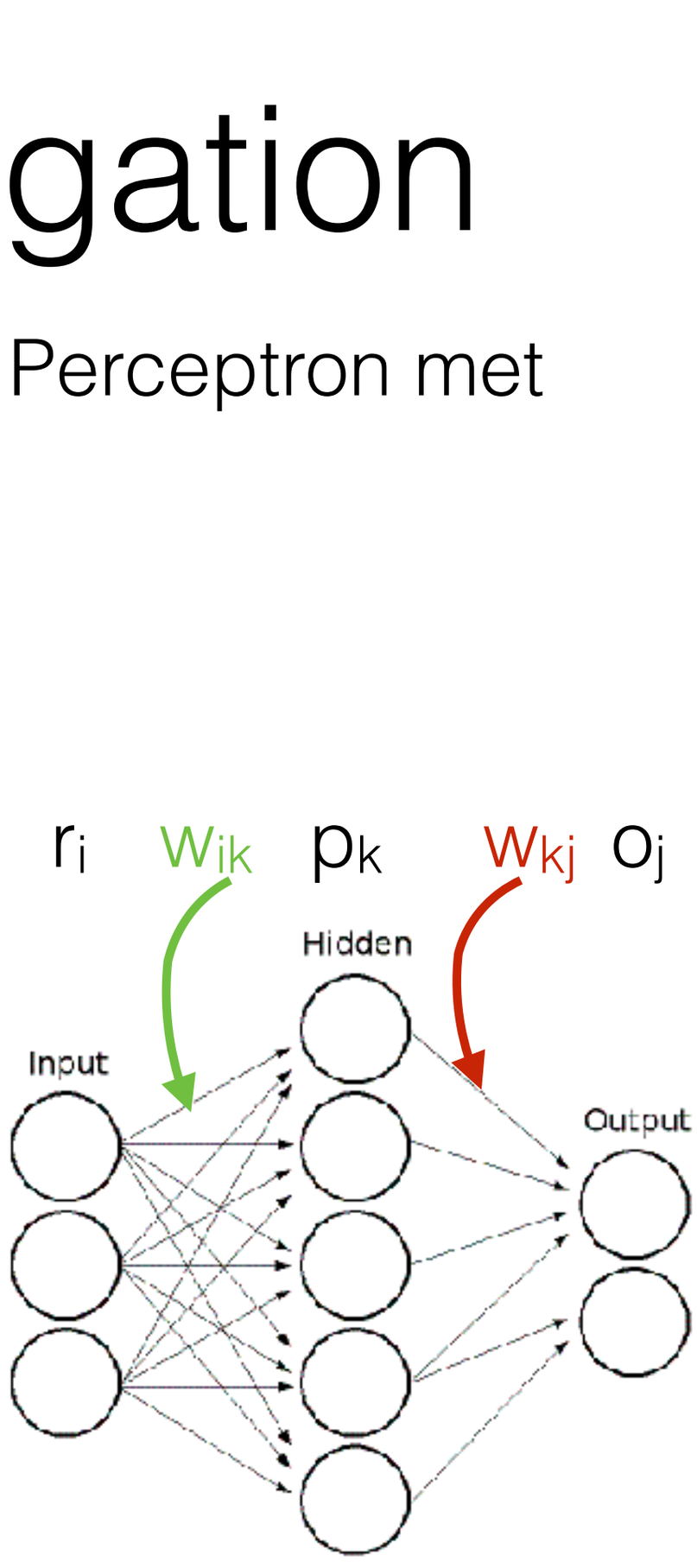


Error-backpropagation

Hoe kan ik de gewichten leren met een Perceptron met een ‘verborgen laag’?

1. Geef de input r
2. Bereken de output o
3. Bereken verandering gewichten w_{kj} van verborgen naar output laag net als bij **delta regel**
4. Bereken verandering gewichten w_{ik} van de input naar de verborgen laag:

$$\Delta w_{ik} = \sum_j \left((t_j - o_j) \frac{\partial f(h_j)}{\partial h_j} w_{ij} \right) \frac{\partial f(h_k)}{\partial h_k} p_k$$



Error-backpropagation

$$\Delta w_{ik} = \sum_j \left((t_j - o_j) \frac{\partial f(h_j)}{\partial h_j} w_{ij} \right) \frac{\partial f(h_k)}{\partial h_k} p_k$$

- Dus voor gewichten naar 'hidden neurons':
  1. bepaal hoeveel een ‘hidden neuron’ bijdroeg aan de output
  2. bereken hoeveel deze output bijdroeg aan de error.

- Zo verklein je de error steeds, net als bij de delta regel

- Zo laat je het error-signaal ‘terug' gaan door het netwerk: error-backpropagation

- Dit werkt goed, maar er zijn een paar problemen

Catastrophic forgetting

- Stel: het netwerk leert een lijst input-output patronen, lijst A
- Daarna leert het netwerk een lijst *andere* input-output patronen, lijst B
- Wat blijkt: het netwerk is nu de lijst A vergeten
- Dit is niet wat je bij mensen / dieren ziet; dus onrealistische vorm van vergeten
    - mensen vergeten vooral als input op elkaar lijkt maar gewenste output niet
    - netwerk vergeet ook als A en B **andere** input en gewenste output gebruiken

Overtraining

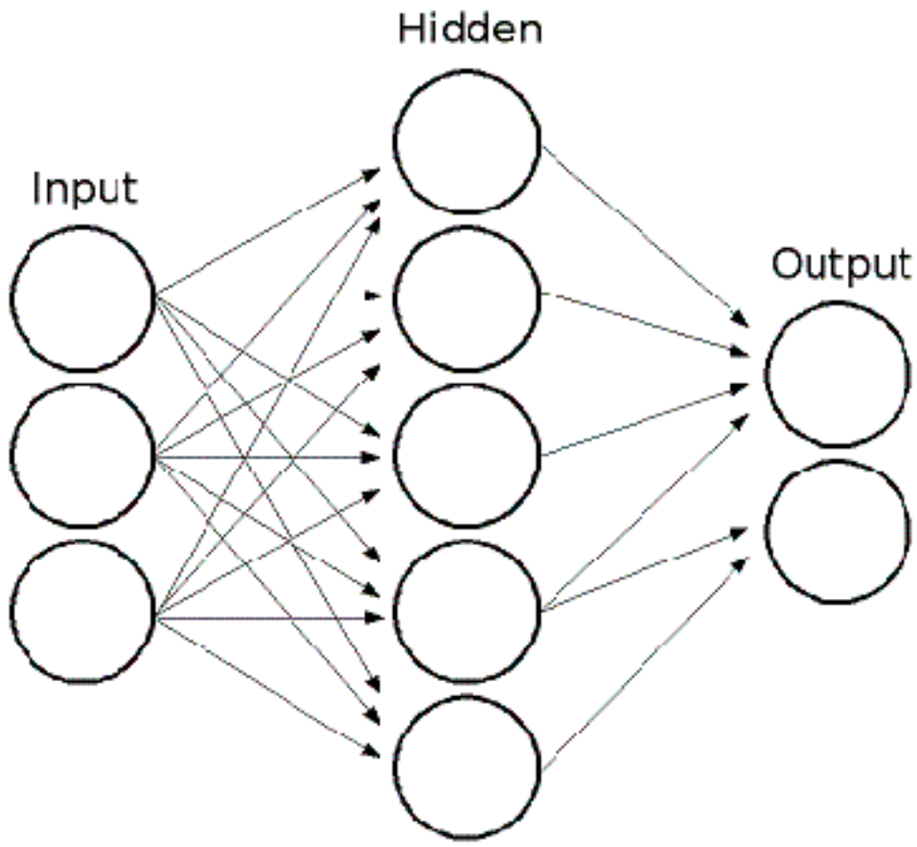
- Stel: het netwerk leert een lijst input-output patronen, lijst A
- Als het netwerk dit goed leert, dan geeft het voor elk inputpatroon het goede outputpatroon
- Maar wat nu als je een input geeft die lijkt op een geleerde input, maar een klein beetje anders is?
    - ‘goed geleerd’ netwerk geeft dan output die hoorde bij geleerde input
    - ‘**overtraind**' netwerk geeft een willekeurige output; dit netwerk kan alleen met exact dezelfde input als het geleerd heeft omgaan
    - het netwerk leert als het ware te precies
    - dit is vooral een probleem als je te veel ‘hidden' neuronen hebt

Error-backpropagation

Error (verschil echte en gewenste output) wordt geminimaliseerd: heel krachtige leerregel
Veel gebruikt in AI / informatica: werkt goed

Problemen:

- Kan blijven hangen lokale minima
- Biologisch onrealistisch
    - gewenste output = vuurfrequentie *elk neuron*
    - voor gewichten input-hidden laag is activiteit output neuronen nodig: niet lokaal
- Catastrophic forgetting
- Overtraining



Samenvatting supervised learning

- **Perceptron regel:** voor binair neuron, enkellaags perceptron

- **Delta regel:** voor rate neuron, enkellaags perceptron

- **Error-backpropagation:** voor rate neuron, meerlaags perceptron

Gradient descent

- (Niet voor tentamen)

$$h = \sum_{n=1}^{N} r_n(t) * w_n$$

- Waar komt die afgeleide vandaan in delta regel?

- ‘error’ E: verschil output en gewenste output

$$E = \sum_j \frac{1}{2}(t_j - o_j)^2 = \sum_j \frac{1}{2}(t_j - f(h_j))^2$$

- Die wil je zo klein mogelijk maken

- Dus: je wilt de gewichten w vinden waarvoor error E het kleinst

- Hoe vond je ook alweer maxima en minima?

Gradient descent

- Je wilt de gewichten w vinden waarvoor error E het kleinst
- Hoe vond je ook alweer maxima en minima?
- Neem afgeleide!

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}(\sum_j \frac{1}{2}(t_j - f(h_j))^2$$

$$= (t_j - f(h_j))\frac{\partial f(h_j)}{\partial w_{ij}}$$

$$= (t_j - f(h_j))\frac{\partial f(h_j)}{\partial h_j}\frac{\partial h_j}{\partial w_{ij}}$$

$$= (t_j - f(h_j))\frac{\partial f(h_j)}{\partial h_j} r_i$$

$$\frac{\partial h_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_i r_i w_{ij}$$

$$= r_i$$

Dit is de delta regel!

Gradient descent

- Je wilt de gewichten w vinden waarvoor error E het kleinst
- Dat is waarvoor afgeleide E naar w gelijk 0 is
- Dus: verander gewichten net zolang *in die richting* tot dit waar is
- NB Updaten gewichten hidden neurons kan ook zo afgeleid worden

$$\frac{\partial E}{\partial w_{ij}} = 0$$

$$(t_j - f(h_j)) \frac{\partial f(h_j)}{\partial h_j} r_i = 0$$

$$(t_j - f(h_j)) \frac{\partial f(h_j)}{\partial h_j} r_i = \Delta w_{ij}$$