Neuronové sítě

zpracoval Martin Kuba

7. dubna 1995

Obsah

1 Úvod

Tento text vznikl na podkladě přednášky doc. Hořejše "Neuronové sítě". Je doplněn poznatky, které jsem získal během vypracovávání své diplomové práce. Úseky, které doc. Hořejš nevykládal, takže je nebude požadovat u zkoušky a tudíž jsou jen doplňující, jsem označil hvězdičkou.

1.1 Historie *

Tuhle část jsem zařadil na začátek, protože je to zvykem, ale doporučuji ji číst až úplně nakonec, protože až po zvládnutí teorie vás budou zajímat drby o tom, jak vznikla.

Základy teorie položili pánové McCuloch a Pitts v roce 1943, když představili první model neuronu, který je po nich nazván. Následujících patnáct let se pilně pracovalo na sítích s prahovými neurony, kdy bylo dokázáno, že jsou schopné provádět libovolný algoritmus. V roce 1949 pan Hebb vyslovil na základě pozorování biologických neuronů své pravidlo (v anglické literatuře *Hebb learning rule*), že synaptické spojení mezi dvěma ve stejnou chvíli aktivovanými neurony se posiluje.

Kolem roku 1960 vypracovala skupina pana Franka Rosenblatta nový model neuronu nazvaný *perceptron*. Pro jednovrstevnou síť perceptronů dokonce dokázali stanovit učící algoritmus a dokázat jeho konvergenci. Bohužel pro teorii neuronových sítí v roce 1969 pánové Minsky a Papert dokázali, že jednovrstevná síť perceptronů nedokáže řešit XOR problém. A protože pan Minsky se s panem Rosenblattem nějak nepohodl a pan Misky byl ve vědě tou dobou velké zvíře, doporučil, aby se zastavilo financování výzkumu pana Rosenblatta a bylo věnováno na jiné "perspektivnější" výzkumy. Načež bylo asi dvacet let ticho po pěšině a výzkumu neuronových sítí se věnovalo jen pár nadšenců. Rosenblatt sice věděl, že vícevrstvé sítě jsou schopny jakýchkoliv výpočtů včetně XOR problému, ale nenašel pro ně učící algoritmus.

(Mimochodem, víte, proč se neuděluje Nobelova cena za matematiku ? Protože panu Nobelovi utekla žena s jedním matematikem. Cesty vývoje vědy bývají složité ...)

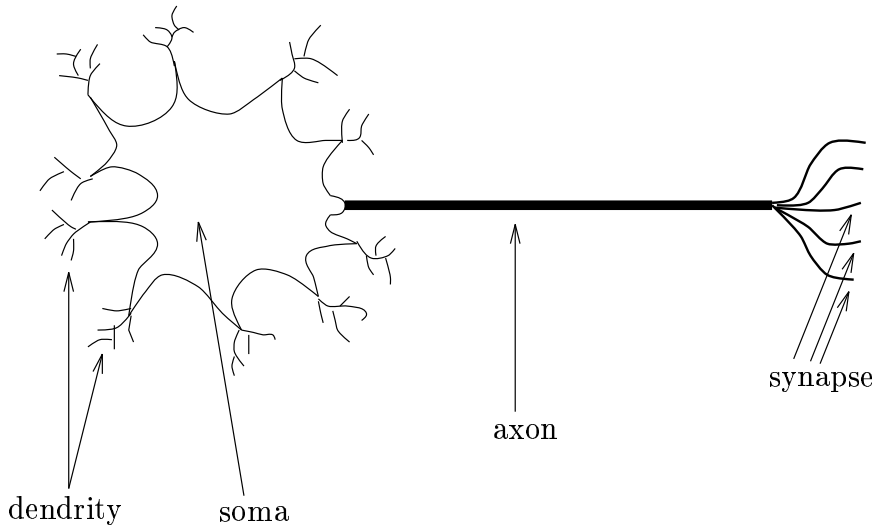
Prudký vývoj v oblasti neuronových sítí po roce 1985 navázal v podstatě tam, kde byl před 20 lety utnut. Algoritmus učení vícevrstvých sítí, nezbytný pro další vývoj a známý jako *back-propagation* (zpětné šíření chyby), poprvé objevil pan Werbos v roce 1974 a znovu byl nezávisle objeven v roce 1985 pány Rumelhartem, Hintonem a Williamsem a také panem Parkerem. Ačkoliv back-propagation není ještě ideálním obecným algoritmem pro učení neuronových sítí, mimo jiné protože zcela jistě není tím algoritmem, který k učení používají biologické sítě neuronů, dokáže řešit mnoho problémů (např. XOR), které jednovrstevné perceptrony nedokážou řešit. Většina současného výzkumu je zaměřena na back-propagation a jeho rozšíření.

1.2 Krátká exkurze do biologie *

Během svého vývoje mnohobuněčné živočichové začali potřebovat řídící informační systém. Vyvinuli dva: pomalý všesměrový chemický, kdy do krve jsou uvolňovány molekuly hormonů, které jsou roznášeny do celého těla a některé buňky pak na ně reagují svým způsobem. Například na adrenalin reagují buňky cév smrštěním a tím zvyšují krevní tlak a buňky srdce reagují tak, že srdce začne rychleji bít. Druhý informační systém je rychlý přesně směrovaný nervový. Prvotní reflexní oblouk vznikl u žahavců (medůzy) podle tohoto schematu: po podráždění receptorové buňky, reagující například na dotyk (tlak), tato buňka vyšle dlouhým vláknem vzruch do svalové buňky a ta se smršťí, takže živočich uhne, popřípadě chňapne kořist. Tato nervová soustava sestávala z osamocených nervových buněk a nazývá se *rozptýlená*. Později bylo třeba reagovat složitěji, takže vznikla *žebříčková* nervová soustava, kterou má například žížala. Dalším pokrokem bylo vytvoření nervových *uzlin*, které má třeba ploštěnka. Následovala *nervová trubice* jdoucí přes celé tělo, která se dalším vývojem diferencovala na *mozek* a *míchu*.

Biologický neuron (viz obrázek 1.2) má *tělo* (soma) veliké několik mikrometrů, které obsahuje organely nutné pro život buňky. Z něj vybíhají tisíce krátkých výběžků – *dendritů* – které tvoří vstup neuronu. Z těla vybíhá jedno vlákno (*axon*), obalené bílým tukovým ochranným obalem (myelinem). Toto vlákno může být mnoho metrů dlouhé, u žirafy vede z hlavy až do konce zadních nohou. Nervy jsou tvořeny svazky těchto vláken a je v silách dnešní chirurgie je navazovat, například při zpětném přišívání amputované ruky. Konec axonu se větví do mnoha výběžků, které končí *synapsemi*. Synapse přiléhají na dendrity dalších neuronů.

Mozek člověka je tvořen asi 10^{11} neurony, axon každého neuronu přiléhá synapsemi na dendrity několika desítek tisíc dalších neuronů. Šedá hmota mozková je tvořena šedými těly neuronů, bílá hmota mozková je tvořena axony oněch neuronů. Celková hmotnost lidského mozku je 1350g u mužů a 1200g u žen, hustota neuronů je 8.10^4 na $1mm^3$, v kůře mozkové (cortexu) je ještě řádově vyšší.



Obrázek 1: Biologický neuron

Mozek je vzhledem ke své váze největším spotřebitelem kyslíku – ačkoliv jeho hmota představuje 2% z celkové hmoty člověka, spotřebuje až 23% z celkového objemu vdechovaného kyslíku, což odpovídá příkonu asi 20W.

Zemřelé neurony se neobnovují, u člověka jich denně zahyne asi 10 000, což za 75 let života činí asi 0.2 až 0.5% celkového počtu.

1.2.1 Funkce biologického neuronu

Vlákno (axon) v klidovém stavu má rozdíl elektrického potenciálu mezi svým vnitřkem a vnějškem asi 70mV, který je způsoben přečerpáváním kladných iontů Na^+ ven a záporných iontů K^- dovnitř. Po povrchu axonu je rozprostřena vodivá membrána, která je schopna se při zvýšení elektrického potenciálu uvnitř těla neuronu nad *prahovou* úroveň prudce depolarizovat, čímž se vytvoří elektický impuls šířící se po axonu jako potenciálová vlna rychlostí od několika až do 120 metrů za sekundu. Mechanismus šíření vlny není ještě zcela objasněn. Pozoruhodné je, že při něm nedochází ke snižování amplitudy impulsu.

Impuls běžící po axonu dorazí k synapsi, ze které se v místě dotyku s dendritem následného neuronu uvolní molekuly chemické látky (tzv. *mediátory* nebo *transmitery*), které způsobí lokální změnu polarizace *transmisní* membrány pokrývající tělo a dendrity. Tím vyvolají tzv. *dendriticko-somatickou* potenciálovou vlnu, která se šíří rozvětveným systémem dendritů směrem k somatu neuronu. V síti dendritů se šíří celá skupina potenciálových vln přišlých od různých synapsí (a od různých neuronů). Synapse samy mohou mít buď *excitační* (vzrušivý) nebo *inhibiční* (tlumivý) charakter podle mediátorů v nich působících, které mají depolarizační nebo hyperpolarizační účinek. Potenciálové vlny se navzájem sčítají a odčítají, až dorazí k axonovému hrbolku.

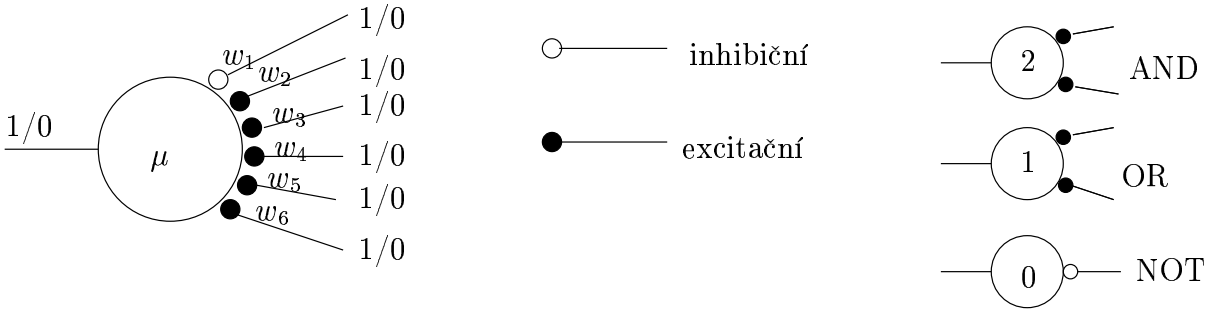
Pokud souhrn přišlých podnětů překročí určitou *prahovou úroveň*, dojde k prudké depolarizaci těla a po axonu se začne šířit potenciálová vlna (vzruch). Po vyslání impulsu se membránový potenciál neuronu vrátí na jistou zbytkovou úroveň. Po krátký čas (*refrakterní perioda*) není neuron citlivý na podněty. Po této době se membránový potenciál začne opět přibližovat prahové hodnotě. Po jejím opětovném překročení se celý děj opakuje. Neuron začne generovat celý sled impulsů, které mohou mít kmitočet až do 1000 impulsů za sekundu, a to podle toho, o kolik převažuje úhrn působení stimulujícího signálu nad prahovou úrovní. Obvyklá frekvence je několik jednotek až desítek impulsů za sekundu. Kmitočet těchto impulsů je úměrný integrálu přes tu část potenciálové vlny, která přesahuje prahovou úroveň. Aby to nebylo tak jednoduché, hodnota prahu není v čase konstantní.

U neuronových sítí živých organismů mají signály v nich působící charakter sledu impulsů. Informace může být na signálech tohoto druhu přenášena celou řadou veličin, od změny tvaru, velikosti či polohy jednotlivých impulsů až po změnu rychlosti, se kterou jsou vysílány. Lze soudit, že velká část informace je přenášena frekvenční modulací posloupnosti impulsů. Při delším zaznamenávání průběhu impulsů zjistíme, že amplituda i tvar impulsů se mění jen nepatrně, avšak výrazně se mění jejich frekvence. Ale informace může být nesena i změnou rychlosti šíření vzruchu, která se mění v rozpětí 0.5 až 2 m/s.

Když to shrneme, tak funkce reálného biologického neuronu je velice složitá a dosud ne úplně prozkoumaná. Píšou se o ní tlusté knihy a nemůžeme se tudíž divit, že formálních matematických modelů neuronu je více a žádný není dokonalý.

2 McCullochův a Pittsův neuron

První formální model neuronu uvedli v roce 1943 McCulloch a Pitts. Byl to *binární prahový neuron* (binary treshold neuron).



Obrázek 2: McCullochův a Pittsův neuron

Do neuronu přichází binární vzruchy (0 – nepřichází vzruch, 1 – přichází vzruch), každá synapse je buď excitační nebo inhibiční, tedy v případě, že přichází vzruch (1), excitační spoj přispívá +1, inhibiční -1. Působení vzruchů se sčítá. Pokud součet všech vzruchů přesáhne prahovou hodnotu μ, neuron dá na výstup 1, jinak 0. Formálně zapsáno

$$n_i(t+1) = sgn(\sum_j w_{ij} n_j(t) - \mu_i)$$

$$sgn(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

kde $n_i(t+1)$ je stav i-tého neuronu v čase $t+1$, w_{ij} je synaptická váha vstupu z j-tého do i-tého neuronu (excitační +1, inhibiční -1), μ_i je prahová hodnota pro i-tý neuron, sgn je funkce, která se v nule skokem mění z nuly na jedničku.

McCulloch a Pitts dokázali, že synchronní pole takovýchto neuronů je v principu schopno libovolného výpočtu a tudíž může provádět stejné výpočty jako digitální počítač.

Tento model neuronu má několik vad na kráse:

- Skutečné neurony mají odpověď nikoliv skokovou, ale spojitou, i když silně nelineární (graded responce).
- Mnoho buněk provádí nelineární součet vstupů.
- Reálný neuron produkuje sled pulsů, ne jen jednu úroveň výstupu. I když reprezentujeme sled pulsů reálným číslem, ignorujeme dost informace, která mohla být sledem přenášena. Někteří experti tvrdí, že fáze pulsů nehraje podstatnou roli, ale jiní experti s nimi nesouhlasí.
- Skutečné neurony nejsou ničím synchronizovány.

3 Perceptrony

Model perceptronu vymyslel v 60. letech pan Rosenblatt. Nejdříve je třeba si ujasnit, co to znamená *perceptron*. Je v tom trochu nepořádek, musel jsem tuhle kapitolu několikrát předělávat, než jsem ji přivedl do výsledného stavu. Tak tedy:

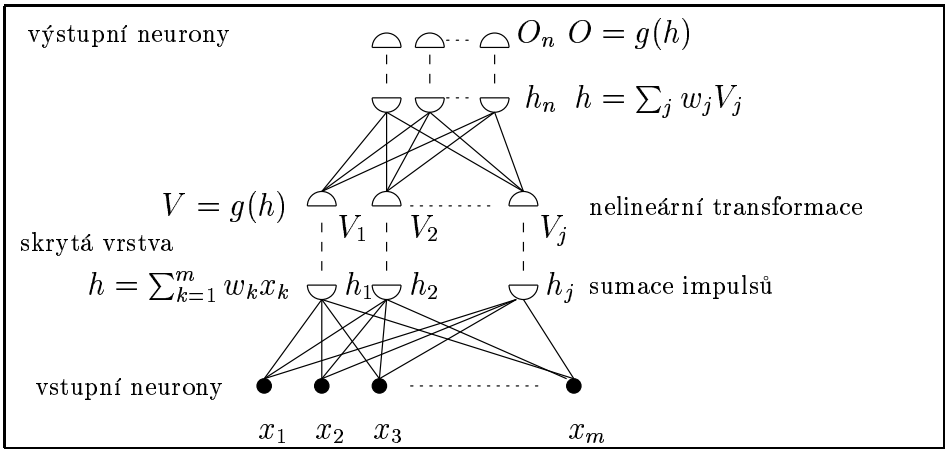
Perceptron je vícevrstvá neuronová síť s dopřednými vazbami (multilayered feed-forward network). Jsou vždy propojeny všechny neurony jedné vrstvy se všemi neurony následující vrstvy, neexistují však žádné spoje mezi vzdálenějšími vrstvami nebo mezi neurony v rámci jedné vrstvy. Do jednotlivého neuronu přichází impulzy jako reálná čísla,

každý je vynásoben jemu příslušnou *synaptickou* vahou, což je opět reálné číslo, udávající "význam" spoje. Impulzy vynásobené synaptickými vahami jsou sečteny, je odečten *práh* neuronu a na výsledek je aplikována *aktivační funkce* (gain function).

Formální zápis:

$$N_i := g(\sum_j w_{ij} N_j - \mu_i)$$

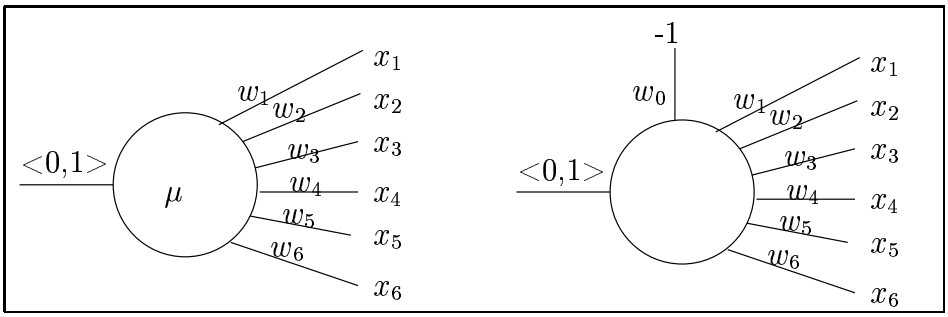
kde N_i je stav i-tého neuronu, w_{ij} je synaptická váha spoje z j-tého do i-tého neuronu, μ_i je práh neuronu i.



Obrázek 3: Obecné schéma perceptronu

Z formálních důvodů se práh μ_i nahradí fiktivním vstupem číslo nula, který má vždy hodnotu -1 a jemu příslušnou synaptickou vahou w_0, která má stejnou velikost jako μ_i. Zápis sumy se tak zjednodušší a můžeme ji pak zapsat jako skalární součin $\vec{w}.\vec{x}$ vektoru vah synapsí $\vec{w}$ a vektoru vstupních impulsů $\vec{x}$. Pak můžeme psát

$$N_i := g(\vec{w}.\vec{x})$$

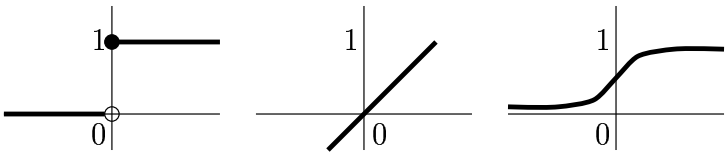


Obrázek 4: Perceptron ; Perceptron s prahem formálně jako nultým vstupem;

Aktivační funkce může být skoková, spojitá lineární nebo spojitá nelineární.

Neurony pak rozlišujeme na *prahové* (treshold units), *lineární* (linear units) a *nelineární* (non-linear units). O každém druhu zvlášť pojednávají následující tři podkapitoly.

Pokud je síť jen jednovrstevná, mluvíme speciálně o jednoduchých perceptronech (simple perceptrons). U lineárních jednotek ani nemá smysl vytvářet více vrstev, protože vrstva lineárních neuronů provádí vlastně lineární zobrazení a skládáním více lineárních zobrazení dostaneme zase jen lineární zobrazení, tedy více lineárních vrtev může dělat jen to, co zvládne i jedna vrstva. A nakonec, pro zmatení, budeme jednotlivé neurony v perceptronech nazývat taky perceptrony.



Obrázek 5: Aktivační funkce: skoková, lineární a sigmoida

3.1 Prahové perceptrony

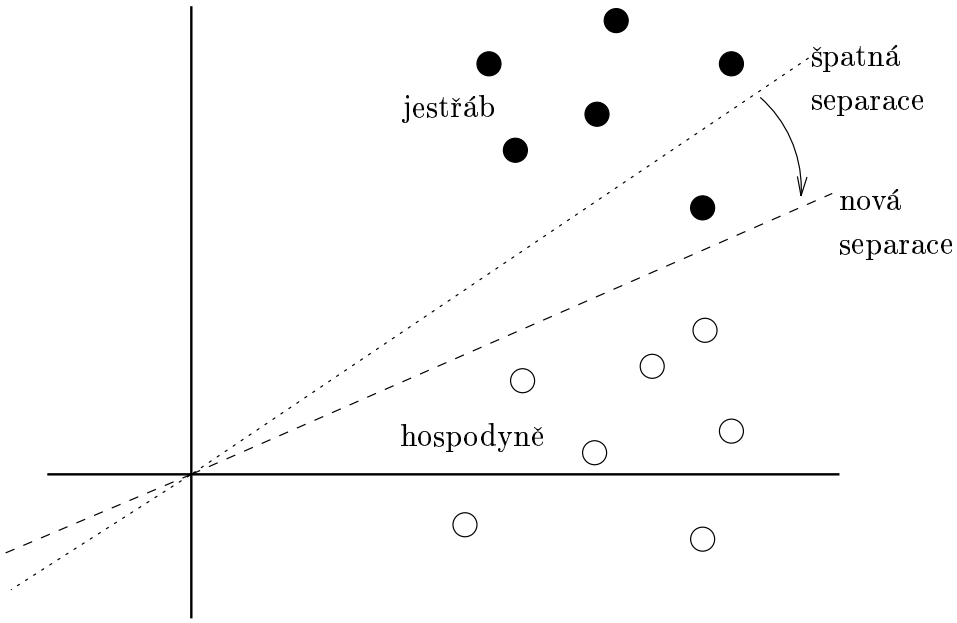
U prahových perceptronů je aktivační funkce skoková:

$$g(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Neuron tedy dává na výstup jen buď 1, pokud součet přesáhl prahovou hodnotu, nebo 0, pokud nepřesáhl.

3.1.1 Lineární separabilita

Jeden jediný prahový perceptron může řešit jen problémy, které jsou *lineárně separabilní.*



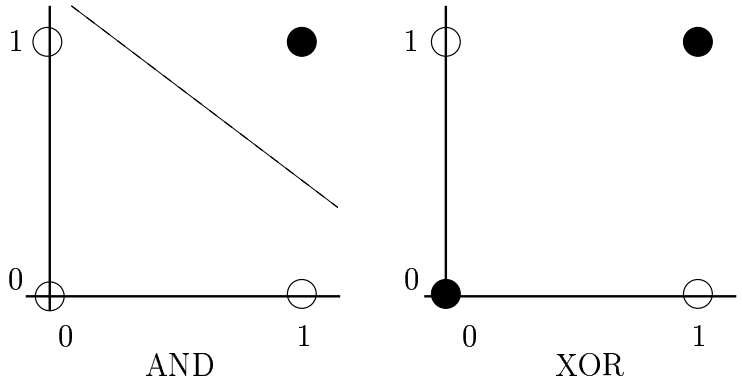
Obrázek 6: Lineární separabilita

Na obrázku je $n + 1$-rozměrný prostor možných podnětů přišlých do neuronu, kde n je počet rozměrů podnětu a jeden rozměr je přidán zavedením prahové hodnoty jako váhy. Mohli bychom tentýž obrázek nakreslit také v n rozměrném prostoru. Pak by separující nadrovina nemusela procházet počátkem souřadných os a její posun by byl úměrný velikosti prahu. Podněty, které jsou důležité, jsou označeny kolečky. Plná kolečka jsou ty podněty, na které neuron musí zareagovat, prázdná naopak ty, na které nemá reagovat. V podání pana Hořejše je tohle neuron způsobující útěk kuřete do bezpečí. Plná kolečka jsou vjemy signalizující jestřába, který chce kuřátko sežrat, prázdná kolečka jsou vjemy signalizující hospodyni, která kuře krmí a proto se před ní nemá utíkat.

Neuron se svými synaptickými vahami (a prahovou hodnotou) určuje *nadrovinu* (vyznačenou přerušovanou čarou), která odděluje (separuje) poloprostory podnětů, na které se reaguje a na které ne. Pokud nadrovina odděluje bezchybně podněty jestřába od podnětů hospodyně, kuře reaguje správně. Stačí vzít vektor $\vec{x}$ vjemů a vynásobit ho s vektorem vah $\vec{w}$. Pokud $\vec{w}.\vec{x} > 0$, neuron zareaguje. (Proč ? Protože lineární algebra a geometrie – skalárním součinem získávám

vzdálenost bodu $\vec{x}$ od roviny $\vec{w}$. Pokud je výsledek kladný, leží bod v jednom poloprostoru, pokud nulový, leží přímo v nadrovině, a pokud je záporný, leží v druhém poloprostoru.)

Co se však stane, když taková dělící nadrovina neexistuje ? Pak nelze správně podněty odlišit a kuře bude sežráno. Jinak řečeno, taková úloha je neřešitelná jedním perceptronem. Příkladem lineárně separabilní úlohy je funkce AND, příkladem funkce, která není lineárně separabilní, je XOR funkce, právě ta, kterou jako důkaz neschopnosti perceptronů předložili Minsky a Papert. Funkce XOR je zvláštním případem *paritní funkce*, která dává jedna, pokud počet jedničkových bitů je lichý, jinak nulu. Je to jeden z nejtěžších problémů pro neuronové sítě, protože výstup musí měnit svou hodnotu po každé změně vstupu.



Obrázek 7: AND je lineárně separabilní, XOR není.

3.1.2 Učení (adaptace)

Učení jednoho perceptronu spočívá v nalezení takových vah, že dělící nadrovina správně rozdělí podněty. Postupně pro všechny podněty, na které má neuron reagovat, se podíváme, jestli neuron dává správný výstup. Pokud na nějaký podnět $\vec{x}$ neuron nereaguje a měl by, pak vektor vah $\vec{w}_{old}$ změníme přičtením vektoru podnětu $\vec{x}$, čímž dostaneme nový vektor vah $\vec{w}_{new}$,

$$\vec{w}_{new} = \vec{w}_{old} + \vec{x}$$

se kterým už neuron na onen podnět reaguje. Proč teď reaguje ? Protože

$$\vec{w}_{new}.\vec{x} = (\vec{w}_{old} + \vec{x}).\vec{x} = \vec{w}_{old}.\vec{x} + \vec{x}.\vec{x} \geq 0$$

Pokud to takhle uděláme pro všechny podněty, na které se má reagovat, nakonec dostaneme takový vektor vah, že nadrovina správně oddělí všechny tyto podněty do jednoho poloprostoru.

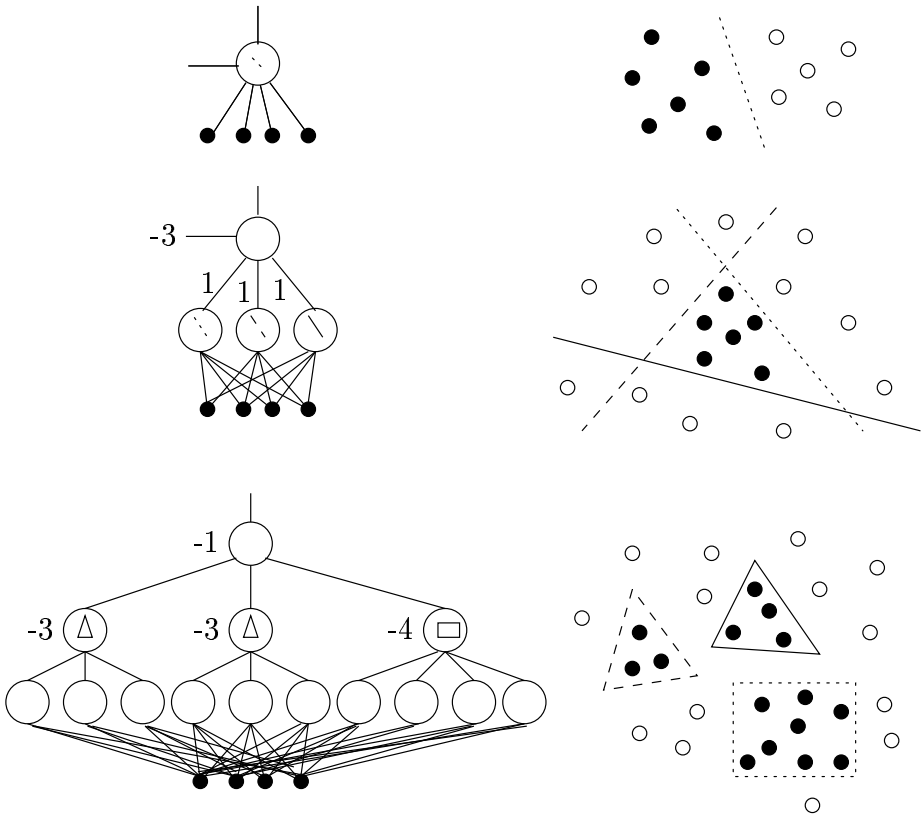
3.1.3 Vícevrstvé sítě perceptronů

Už víme, že samotný perceptron nemůže řešit problémy, které nejsou lineárně separabilní. Nemůže je tedy řešit ani jednovrstevná síť perceptronů. Ale naštěstí je může vyřešit vícevrstvá síť.

Intuitivní důkaz je naznačen na obrázku 8. Jeden perceptron mi rozdělí prostor na dva poloprostory nadrovinou. Zareaguje pak jen na podněty nacházející se jen v jednom poloprostoru. Jsou označeny plnými kolečky.

Dvouvrstevná síť může vymezit lib. konvexní útvar. Každý neuron první vrstvy vymezí jeden poloprostor, neuron druhé vrstvy pak vyhodnocuje průnik těchto poloprostorů. Základní prostor podnětů má být diskrétní, aby k vymezení libovolné konvexní množiny stačil konečný počet poloprostorů. Naštěstí reálný svět nepotřebuje matematicky přesná vymezení, stačí aproximace konečným počtem poloprostorů.

Třívrstevná síť rozezná lib. počet konvexních útvarů. Neurony první vrstvy opět vymezují poloprostory, neurony druhé vrstvy rozlišují průniky těchto poloprostorů, čímž vymezují konvexní množiny, a neuron třetí vrstvy provádí sjednocení těchto konvexních množin.



Obrázek 8: Schopnosti jedno-, dvou- a třívstevné sítě.

3.2 Lineární perceptrony *

Lineární perceptrony používají lineární aktivační funkci

$$g(x) = x$$

Lineární síť není nutné učit, příslušné synaptické váhy se dají přímo vypočítat. Nicméně vrstva lineárních perceptronů provádí jen lineární zobrazení, což není nic nového a neznámého, proto tento typ sítě nemá velké praktické použití.

Pro výpočet vah je nutné, aby vzory byly navzájem *lineárně nezávislé*, což nastane jen tehdy, pokud je vzorů míň než neuronů v síti.

3.3 Nelineární perceptrony

Nyní se dostáváme k nejpoužívanějšímu typu sítě. Nelineární perceptrony používají jako aktivační funkci tzv. *saturační funkci*, která součet impulzů transformuje do intervalu $< 0, 1 >$ a to tak, že v blízkosti nuly funkce stoupá velmi prudce, zatímco u vysokých hodnot stoupá jen nepatrně, takže případný rozdíl se tak moc neprojeví.

Tato vlastnost byla převzata od biologických neuronů. Například pro hladového studenta znamená jeden rohlík mnohem víc, než pro přejedeného.

Jako saturační funkce se nejčastěji používá *sigmoida*:

$$g(x) = \frac{1}{1 + e^{-\lambda x}}$$

Dá se dokázat, že *třívstevná* síť těchto neuronů může aproximovat *libovolné* reálné zobrazení mezi prostorem vstupů a prostorem výstupů. Jinak řečeno, ke každé funkci $f : R^n \to R^m$ existuje 3-vstevná síť, která ji realizuje.

Důkaz stojí na *Kolmorogově teorému*, který říká:

Kolmogorovův teorém * Existují spojité rostoucí funkce ϕ_{ij} na $\langle 0,1\rangle$ tak, že každá spojitá funkce f na $< 0,1 >^n$ může být zapsána ve tvaru

$$f(x_1,\ldots,x_n) = \sum_{i=1}^{2n+1} \alpha_i(\sum_{j=1}^{n} \phi_{ij}(x_j))$$

kde α_i jsou vhodně zvolené spojité funkce jedné proměnné.

3.3.1 Adaptace s učitelem

Víme tedy, že vícevrstvé sítě mohou provést libovolné zobrazení (výpočet). Ke každému problému máme zajištěnu existenci řešící sítě, ale jak ji získáme ? Přímý výpočet synaptických vah neuronů je nevhodný a ve většině případů neuskutečnitelný. Nastupuje učení sítě za použití sady *příkladů*. Mějme vícevrstvou síť s m vstupními a n výstupními neurony. *Trénovací množinou* nazveme množinu dvojic vektorů

$$T = \{[\vec{x}^1,\vec{y}^1],\ldots,[\vec{x}^P,\vec{y}^P]\}$$

kde $\vec{x} = (x_1,x_2,\ldots,x_m)$ je vždy vektor vstupních hodnot a $\vec{y} = (y_1,y_2,\ldots,y_n)$ je vektor požadovaných výstupních hodnot.

Trénovací množina obsahuje P příkladů, které se síť musí naučit. Takovémuto učení se říká *učení s učitelem*, protože musí existovat nějaká vyšší autorita, která síti předkládá příklady dvojic vstupů a výstupů a pokud síť odpovídá špatně, informuje ji o chybě a modifikuje ji. Existuje i učení bez učitele, kdy síť nedostává dvojice vstup-výstup, ale jen množinu vstupů a sama si je rozděluje do skupin podle podobnosti.

Celková chyba sítě se počítá jako

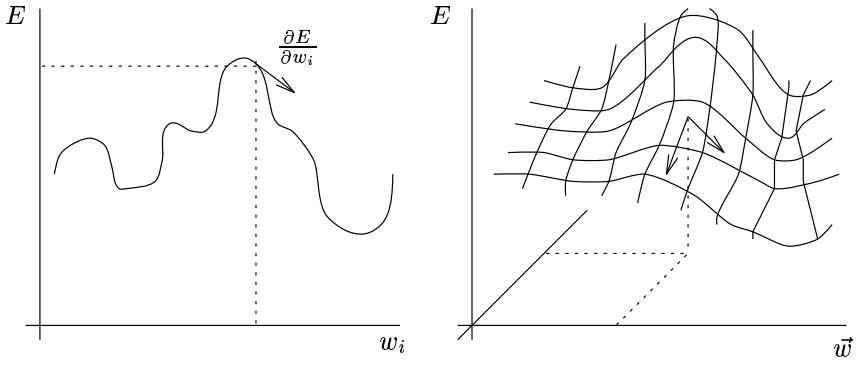
$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{i=1}^{n}(O_i^p - y_i^p)^2$$

kde O_i^p je skutečný výstup i-tého neuronu ve výstupní vrstvě při předložení vstupu p-tého příkladu na neurony vstupní vrstvy. Pro přehlednost se můžete podívat zpátky na obrázek 3. Pak $(O_i^p - y_i^p)^2$ je rozdíl očekávaného a požadovaného vstupu, suma pro i sčítá přes všech n výstupních neuronů a suma pro p sčítá přes všech P příkladů.

(*) Funkce E se v anglické literatuře nazývá **"cost function"** a je určena pro měření chyby sítí s *asymetrickými* spoji. Je to něco jiného než tzv. "energy function" zavedená v Hopfieldově modelu, která je určena pro měření stavu sítí se symetrickými spoji.

3.3.2 Gradient descent learning

Učení sítě vychází z následující myšlenky. Pokud síti předložíme pevný vektor $\vec{x}$ na vstup, chyba sítě E je funkcí všech synaptických vah neuronů. A protože aktivační funkce neuronů (sigmoidy) jsou derivovatelné, je i E derivovatelná podle jednotlivých vah. Znázorníme si to na grafu:



Obrázek 9: Error landscape

Vodorovná osa představuje mnoharozměrný prostor vah neuronů sítě, svislá osa je hodnota chyby E pro dané váhy. Vstupní vektor $\vec{x}$ je pevný, teď učíme síť jeden konkrétní příklad. Graf E je mnoharozměrná — a co je důležité, *derivovatelná* — plocha, tzv. ERROR LANDSCAPE, krajina chyb.

Pro nějaké konkrétní nastavení vah jsme někde v této krajině. Naším úkolem je sejít co nejníž, minimalizovat chybu. Ideální by bylo sejít až na "úroveň moře", na nulovou hodnotu chyby, ale to se podaří jen málokdy. Budeme se chovat jako turista v horách, nebo jako voda, která chce stéct do nížin. Vydáme se po spádnici dolů. Spádnici najdeme zderivováním krajiny (chyby) v místě, kde stojíme. Uděláme krok směrem po spádnici, dostaneme se tak na nové místo a postup opakujeme.

Pokud bude krok příliš dlouhý, například delší než svah kopce, na kterém stojíme, můžeme se dotat až na vedlejší kopec a dokonce výš, než jsme byli. V takovém případě se vrátíme a uděláme kratší krok. Tak se postupně dostaneme na nejnižší místo, ze kterého všechny cesty povedou nahoru. Pokud to bude globální minimum chyby, jsme doma, pokud to je lokální minimum — místní bezodtokové údolí, máme smůlu a musíme se dostat někam pryč. Zvolíme směr a skočíme mnoho mil a budeme doufat, že z nového místa se k moři dostaneme. Zkrátka a dobře, jsme v pozici turisty v mlze, ale s výškoměrem.

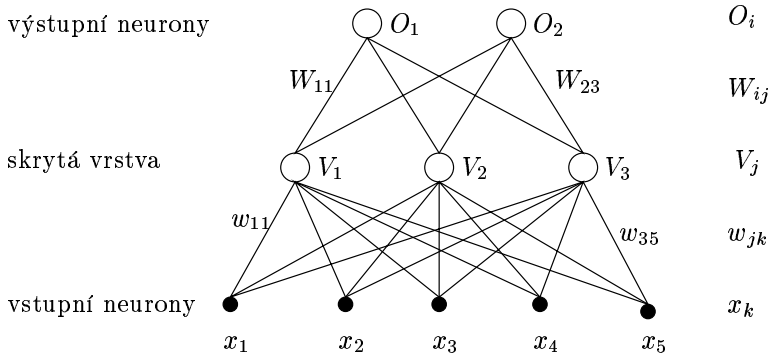
Formálně zapsán vypadá předpis pro změnu vah takto:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} + \alpha[w_i - w_i^{old}]$$

tedy změna konkrétní váhy je dána parciální derivací chyby E podle této váhy, vynásobenou délkou kroku η. Druhý člen s parametrem α je ve výrazu jako "setrvačnost" pohybu, aby se eliminovalo malé houpání terénu. Když jdu z dlouhého prudkého svahu, taky si nevšímám malých kamínků, jejichž strana obrácená ke mně má opačný sklon než svah.

3.3.3 Back-Propagation

Teď teoreticky víme, jak síť učit pomocí metody "svahového sestupu" nebo "sestupu svahem", jak se dá "gradient descent" přeložit. Prakticky je ale velice důležité, jakým způsobem budeme synaptické váhy měnit. Byl to vynález metody *zpětného šíření chyby*, který způsobil obrovský nárůst zájmu o neuronové sítě po roce 1985.



Obrázek 10: Dvouvrstevná síť ukazující označení neuronů a vah

Algoritmus back-propagation si předvedeme na dvouvrstevné síti, nakreslené na obrázku 10. Máme ji naučit trénovací množinu P příkladů $T = \{[\vec{x}^1, \vec{y}^1], \ldots, [\vec{x}^P, \vec{y}^P]\}$, kde $\vec{x} = (x_1, x_2, \ldots, x_m)$ a $\vec{y} = (y_1, y_2, \ldots, y_n)$.

Výstupní neurony jsou označeny O_i, skryté neurony V_j a vstupní neurony označíme rovnou hodnotami vstupů x_k. Spoje w_{jk} vedou ze vstupní vrstvy do skryté vrstvy, spoje W_{ij} ze skryté do výstupní. Index i vždy označuje výstupní neuron, j skrytý a k vstupní. Písmeno p je používáno pro indexování příkladů.

Při zadání vektoru $\vec{x}^p$ na vstup sítě skrytý neuron j obdrží vstup

$$h_j^p = \sum_k w_{jk} x_k^p$$

a po transformaci aktivační funkcí (sigmoidou) dává výstup

$$V_j^p = g(h_j^p) = g(\sum_k w_{jk} x_k^p)$$

. Pak ovšem výstupní neuron i obdrží vstup

$$h_i^p = \sum_j W_{ij} V_j^p = \sum_j W_{ij} g(\sum_k w_{jk} x_k^p)$$

a po transformaci aktivační funkcí (sigmoidou) dává výstup

$$O_i^p = g(h_i^p) = g(\sum_j W_{ij} V_j^p) = g(\sum_j W_{ij} g(\sum_k w_{jk} x_k^p))$$

Prahy neuronů zadáme jako zvláštní vstupní neurony nastavené na -1, jak jsme předvedli již dříve.

Celková chyba sítě měřená jako

$$E[\mathbf{w}] = \frac{1}{2} \sum_{pi} [y_i^p - O_i^p]^2$$

se po dosazení mění na

$$E[\mathbf{w}] = \frac{1}{2} \sum_{pi} [y_i^p - g(\sum_j W_{ij} g(\sum_k w_{jk} x_k^p))]^2$$

To je zřejmě spojitá diferencovatelná funkce všech vah, takže můžeme použít gradient descent algoritmus pro nalezení správných vah.

Až dosud byly všechny rovnice jednoduché a průhledné, protože jsme jen dosazovali za výrazy. Ale teď se soustřeďte, teď to bude horší.

Změna vah pro výstupní neurony Pro spoje mezi skrytými a výstupními neurony podle gradient descent vychází změna konkrétní váhy W_{ij} na

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}}$$

. Záleží na délce kroku η, což je námi zvolená konstanta, a derivaci chyby E podle této váhy.

Po parciálním zderivování chyby E podle váhy W_{ij} nám vyjde[1]

$$\frac{\partial E}{\partial W_{ij}} = -\sum_p [y_i^p - O_i^p] g'(h_i^p) V_j^p$$

Zavedeme označení

$$\delta_i^p = g'(h_i^p)[y_i^p - O_i^p]$$

[1] Pokud mi nevěříte, tak

$$E[\mathbf{w}] = \frac{1}{2} \sum_p \sum_i [y_i^p - g(\sum_j W_{ij} V_j^p)]^2$$

suma pro i je až na jeden člen tvořena konstantami

$$\frac{\partial E}{\partial W_{rs}} = \frac{1}{2} \sum_p \left(0 + \ldots + 0 + \frac{\partial [y_r^p - g(\sum_j W_{rj} V_j^p)]^2}{\partial W_r s} + 0 + \ldots + 0 \right)$$

a po zderivování tohoto členu dostanu

$$= \sum_p [y_r^p - g(\sum_j W_{rj} V_j^p)](0 - g'(\sum_j W_{rj} V_j^p)(0 + \ldots + V_s^p + \ldots + 0))$$

z čehož dostanu

$$\frac{\partial E}{\partial W_{rs}} = -\sum_p [y_r^p - O_r^p] g'(h_r^p) V_s^p$$

kde δ_i^p je chyba i-tého (tedy výstupního) neuronu, závisící na sklonu sigmoidy v místě hodnoty vstupu do neuronu h_i^p a na rozdílu skutečné a požadované hodnoty výstupu $[y_i^p - O_i^p]$.

Odtud složením dostáváme

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} = \eta \sum_p \delta_i^p V_j^p. \tag{1}$$

Změna vah pro vnitřní neurony Spoje w_{jk} mezi vstupními a vnitřními neurony jsou ve vzorci pro E mnohem hlouběji.

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} = -\eta \sum_p \frac{\partial E}{\partial V_j^p} \frac{\partial V_j^p}{\partial w_{jk}}$$

$$= \eta \sum_p i[y_i^p - O_i^p] g'(h_i^p) W_{ij} g'(h_j^p) x_k^p$$

$$= \eta \sum_p i \delta_i^p W_{ij} g'(h_j^p) x_k^p$$

Zavedeme označení

$$\delta_j^p = g'(h_j^p) \sum_i W_{ij} \delta_i^p$$

. kde δ_j^p je chyba j-tého (tedy vnitřního) neuronu, a závisí na sklonu sigmoidy v místě hodnoty vstupu do neuronu h_j^p a na chybách neuronů, do kterých posílá svůj výstup násobených příslušnými vahami spojů.

Dostáváme tedy

$$\Delta w_{jk} = \eta \sum_p \delta_j^p x_k^p. \tag{2}$$

Princip back-propagation Všimněme si, že předpisy 1 a 2 jsou stejné až na definici δ. Všeobecně, při jakémkoliv počtu vrstev, předpis pro změnu vah bude mít tvar

$$\Delta w_{pq} = \eta \sum_p \delta_{output} \times V_{input}$$

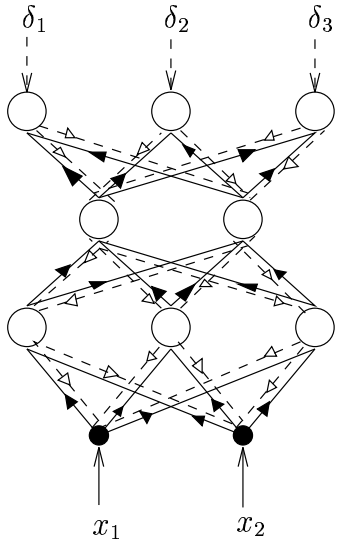
kde q je neuron, ve kterém spoj začíná a v p končí, V je výstup neuronu q. Význam δ závisí na tom, o kterou vrstvu se jedná. Chyba výstupních neuronů se spočítá z rozdílu mezi skutečným a požadovaným vstupem a pak se zpětně (back) předává (propagation) vnitřním neuronům, přičemž se násobí hodnotou váhy spoje. Váhy spojů se tak používají dvakrát — při šíření signálů jedním směrem a při šíření chyby druhým směrem. Obrázek 11 ukazuje tuto myšlenku.

Ačkoliv jsme napsali pravidla pro změnu vah jako sumy přes všechny příklady, v praxi se nejdřív předloží jeden příklad, upraví se všechny váhy a pak se teprve předloží další příklad. Dokonce je výhodné nepředkládat příklady ve stále stejném pořadí, ale vybírat je náhodně. Náhodný výběr pořadí činí cestu váhovým prostorem náhodnou a umožňuje širší prozkoumání chybové krajiny.

To, že potřebné hodnoty derivací chybové funkce mohou být vypočítány pomocí zpětného šíření chyby je velmi dobré. Má to dva důležité následky:

- Pravidlo pro změnu synaptických vah je *lokální*. Pro výpočet změny konkrétní váhy potřebujeme jen hodnotu šířené chyby δ na jednom jejím konci a hodnotu přenášeného signálu V na druhém jejím konci. Tím pádem se dá výpočet back-propagation paralelizovat.
- Výpočetní složitost je menší, než bychom mohli očekávat. Pokud máme n spojů, výpočet chyby E stojí n operací. Výpočet n derivací přímo by pak potřebovalo n^2 operací, zatímco pomocí back-propagation nám postačí n operací pro změnu hodnoty *všech* vah.

Bohužel, back-propagation není tím algoritmem, který používá pro učení sítí příroda. Jeho výpočet je sice lokální, ale lokálnost je pro biologickou implementaci podmínkou nutnou, nikoliv však postačující. Problém spočívá v obousměrnosti spojů, kdy se po nich zpětně šíří chyba. Reálné axony *nejsou* v žádném případě obousměrné.



Obrázek 11: Back-propagation ve vícevrstvé síti. Plné šipky ukazují směr šíření signálů, zatímco čárkované směr šíření chyby.

Jako aktivační funkce $g(h)$ se používají funkce

$$g_1(h) = \frac{1}{1+e^{-\lambda h}}$$

a

$$g_2(h) = tanh(\lambda h)$$

protože jejich derivace je snadné vyjádřit pomocí jich samých jako $g_1'(h) = \lambda g_1(1-g_1)$ a $g_2'(h) = \lambda(1-g_2^2)$. Proto je často v literatuře rovnice 1 vidět zapsána jako

$$\delta_i^p = (y_i^p - O_i^p)O_i^p(1-O_i^p)$$

pro neurony s výstupem $0/1$ a pro $\lambda = 1$.

Algoritmus back-propagation Protože je back-propagation nejdůležitějším a nejpoužívanějším algoritmem pro učení sítí, zapíšeme ho zde po jednotlivých krocích.

Mějme síť s L vrstvami $l = 1, \ldots, L$ a značíme V_i^l výstup i-tého neuronu v l-té vrstvě. V_i^0 znamená x_i, tedy i-tý vstup. Označení w_{ij}^l znamená spoj z V_j^{l-1} do V_i^l. Pak algoritmus zní:

1. inicializuj váhy na malá náhodná čísla
2. Zadej příklad $\vec{x}^p$ na vstup sítě (vrstva $l = 0$), takže
$$V_k^0 = x_k^p$$
3. Nech šířit signály sítí
$$V_i^l = g(h_i^l) = g(\sum_j w_{ij}^l V_j^{l-1})$$
4. Vypočítej delty pro výstupní vrstvu
$$\delta_i^M = g'(h_i^M)[y_i^p - V_i^M]$$
5. Vypočítej delty pro předcházející vrstvy zpětným šířením chyby
$$\delta_i^{l-1} = g'(h_i^{l-1}) \sum_j w_{ji}^l \delta_j^l$$

6. Změň váhy podle pravidla

$$\Delta w_{ij}^l = \eta \delta_i^l V_j^{l-1}$$
$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$$

7. Běž na bod 2 a opakuj pro další příklad.

Vylepšení back-propagation O jednom vylepšení jsme se již zmínili. Spočívá v zavedení momentu setrvačnosti ke změnám vah, kdy změna váhy záleží i na velikosti předchozí změny.

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} + \alpha \Delta w_i^{old}$$

I druhé vylepšení spočívá v určování délky kroku. Pokud jdeme po planinách, můžeme krok prodlužovat, pokud narzíme na členitou krajinu, krok zkrátíme. Další vylepšení může být takové, že na začátku učení sítě budeme používat povlovnější aktivační funkci s menším λ, ke konci učení pak budeme "přitvrzovat", zvýšíme λ".

3.4 Aplikace vícevrstvých sítí

V této kapitole sítí rozumíme síť nelineárních perceptronů učenou pomocí back-propagation.

Hlavní důvod, proč neuronové sítě používáme, je jejich schopnost *generalizace*. Spočívá v tom, že síť představuje spojité zobrazení z prostoru vstupů do prostoru výstupů. Když naučíme síť správně odpovídat na určitou množinu příkladů, určili jsme vlastně hodnotu tohoto zobrazení v několika diskrétních bodech prostoru vstupů. Spojité zobrazení reprezentované sítí je vlastně hyperplocha proložená těmito body. Proto když zadáme síti vstupní hodnotu, která nebyla mezi příklady, které umí, síť dá nějaký výstup. Zajímavé je to, že tento výstup je v lidském smyslu *rozumný*. Kromě trénovací množiny se tak zavádí pojem *testovací* množiny, což jsou dvojice vstupů a výstupů, které také mají smysl, ale síť na ně není naučena. Pokud mezi příklady v trénovací množině byla nějaká závislost, síť ji zjistí. Na testovací množině se pak zjistí, nakolik je sítí zjištěná závislost tou závislostí, kterou jsme chtěli zjistit.

Aplikace tohoto fantastického rysu neuronových sítí je nabíledni. Pokud máme množinu dat, o kterých víme, že je mezi nimi nějaká závislost, ale neumíme ji objevit, zadáme tato data síti a ona nám tu závislost najde.

Uvedme si několik příkladů použití n. sítě s uvedením funkce, kterou síť provádí.

3.4.1 Benzínové pumpy – příklad generalizace

Jedna z prvních aplikací neuronových sítí byla tato. V americké firmě provozující síť benzínových pump byl zaměstnán člověk, který už dlouhá léta určoval, jak moc se má která pumpa předzásobit, například že před svátky se mají předzásobit pumpy na výpadovkách z města a podobně. Tuto práci dělal velice dobře a nikdo jiný to neuměl tak dobře jako on. Tento člověk měl ale odejít do důchodu. Nebyl však schopen popsat, podle jakých pravidel rozdělování benzínu určuje, protože to dělal na základě dlouholeté zkušenosti intuitivně. Proto nebylo možné ani setavit klasický expertní systém, protože ten potřebuje přesně definovaná pravidla.

Vyrobili tedy neuronovou síť, která měla jako vstup stejné údaje, jako měl onen člověk, tzn. datum, počasí a já nevím co ještě. Jako výstup se síť učila dávat stejné rozdělení distribuce benzinu jako člověk. Zkrátka a dobře, asi půl roku síť pozorovala, jak to ten člověk dělá, a naučila se to taky. On mohl jít klidně do důchodu a síť místo něj pracuje bez nároku na mzdu dodnes.

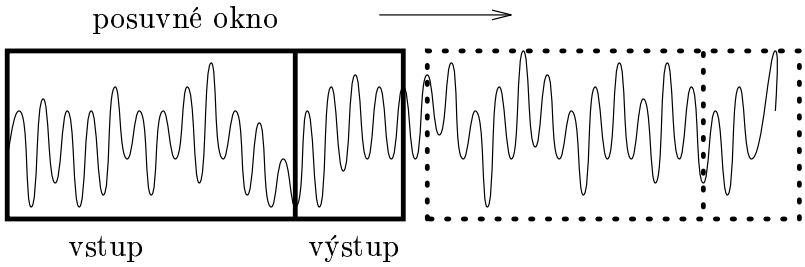
3.4.2 Srdeční arytmie – příklad klasifikace

Skupina lékařů vzala mnoho záznamů EKG a u každého z nich určila, jestli člověk, kterému záznam patřil, trpí srdeční arytmií nebo ne. Pak tyto záznamy dávali neuronové síti na vstup a učili ji správně odpovídat, jestli daný záznam vykazuje arytmii. Až byla naučená na této trénovací množině záznamů, začali jí předkládat záznamy, které předtím neviděla. A ukázalo se, že síť poznává arytmii velice dobře, dokonce lépe než 90% normálních lékařů. Je to tím, že síť rozhoduje stejně dobře, jako by rozhodovala skupina lékařů, kteří vybírali trénovací množinu, což jednak byli špičkoví odborníci a jednak jich bylo víc, takže rozhodovali lépe než jednotlivec.

Síť tak vlastně prováděla *klasifikaci* záznamů na dvě skupiny, jednu vykazující arytmii a druhou nevykazující. Klasifikovat pomocí sítě můžeme i na více skupin, než dvě.

3.4.3 Předpovídání budoucnosti – predikce

Predikce dosáhneme jednoduchým trikem. Mějme záznam průběhu nějaké veličiny (třeba teploty), jejíž hodnota nějak závisí na předchozích hodnotách. Pak jako trénovací množinu používáme jakési plovoucí okno, pohybující se po záznamu, a síť učíme odpovídat na přední část tohoto okna jeho zadní částí, viz obrázek 12

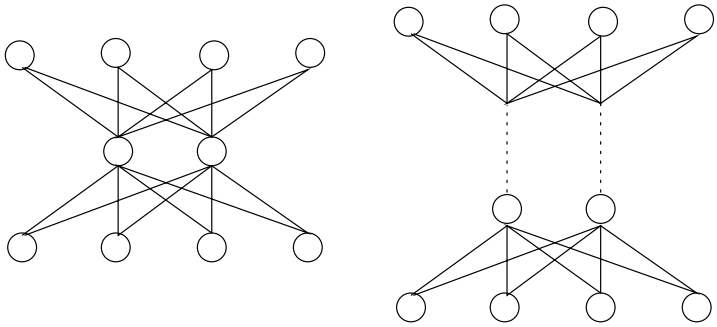


Obrázek 12: Princip učení při predikci

Například vezmeme záznamy o teplotě vzduchu za posledních dvěstě let, a síti vždy zadáme na vstup teploty v sedmi po sobě následujících dnech a učíme ji odpovídat na výstupu teplotou následujícího osmého dne. Až je naučená, zadáme jí teploty za posledních sedm dní a dozvíme se, jak bude zítra. (Nevím, jestli zrovna tenhle příklad doopravdy funguje, protože teplota nezávisí jen na předchozích teplotách, ale jako ilustrace se hodil).

3.4.4 Komprese dat

Komprese se provádí tak, že síť učíme na identitu, tedy jako příklady má stejný vstup a výstup. Vnitřní vrstva neuronů musí mít méně neuronů než vstupní a výstupní, protože pak je síť nucena vícerozměrný vstup transformovat do méněrozměrného prostoru reprentovaného vnitřní vrstvou. Síť se pak "rozstřihne" ve střední vrstvě a její přední část se používá pro kompresi a její zadní část pro dekompresi.



Obrázek 13: 4-2-4 enkoder a rozstřihnutá síť

3.4.5 NETtalk

Projekt NETtalku měl za úkol vyřešit vyslovování psaného anglického textu. Vstupem bylo plovoucí okno sedmi znaků pohybující se po textu. Výstupem byl foném předávaný generátoru řeči. Síť měla 7×9 vstupních neuronů kódujících sedm znaků z 29 možných, 80 skrytých neuronů a 26 výstupních neuronů kódujících fonémy. Síť byla trénována na 1024 slovech, po deseti trénovacích epochách produkovala srozumitelnou řeč a po 50ti epochách měla na trénovací množině 95%ní správnost. Nejdříve se naučila výrazné rysy jako mezery mezi slovy a potom postupně vylepšovala rozlišování, takže to znělo jako dítě učící se mluvit. Zjistilo se,že některé vnitřní neurony reprezentují významné věci jako třeba rozdíl mezi samohláskami a souhláskami. Po naučení byla síť zkoušena na dalších slovech, na kterých vykázala 78%ní správnost a produkovala celkem srozumitelnou řeč.

Je zajímavé porovnat NETtalk s komerčním DEC-talkem, který je založen na ručně zadaných pravidlech. Zatímco NETtalk se naučil z příkladů, DEC-talk je výsledek desetileté práce mnoha lingvistů. DEC-talk mluví rozhodně lépe, ale úsilí do něj vložené je také mnohem větší.

Toto byl příklad toho, jak jsou použitelné neuronové sítě. Použití neuronových sítí je druhým nejlepším řešením problému. Prvním nejlepším řešením je použít známý optimální algoritmus.

3.4.6 Rozpoznávání objektů na sonaru *

Gorman a Sejnovski v roce 1988 učili dvouvrstevnou síť rozpoznávat odezvy sonaru na dva druhy předmětů ležících na dně Chesapeakeského zálivu, na kameny a kovové válce. Před vstupem do sítě byla na datech provedena Fourierova transformace, kterou se získalo frekvenční rozložení signálů. Tato transformace mohla být v principu také provedena neuronovou sítí, ale takhle se ušetřilo několik vrstev a rychlost.

Síť měla 60 vstupních neuronů a dva výstupní, jeden pro kameny, druhý pro válce. Počet neuronů skryté vrstvy se měnil od žádného do 24. Bez vnitřních neuronů síť velice rychle dosáhla 80% úspěšnosti, ale dál se nedokázala zlepšit. Se 12ti vnitřními neurony vždy síť dosáhla skoro 100%ní úspěšnosti, při přidávání dalších už se její výkon nezlepšoval.

Po natrénování byla síť testována na nových datech, na kterých dosahovala 85%ní úspěšnosti. Podařilo se ji zlepšit až na 90% pečlivějším výběrem trénovací množiny.

3.4.7 Řízení auta *

Pomerlau v roce 1989 vytvořil síť pro řízení auta. Vstup byly 30×32 pixelové obrázky z videokamery na střeše auta a 8×32 obrázky detektoru vzdáleností. Tyto vstupy byly vedeny do vnitřní vrstvy s 29 neurony a odtud do výstupní vrstvy s 45 neurony uspořádanými do řady. Prostřední výstupní neuron znamenal jízdu přímo, zatímco míra zatáčení byla reprezentována vzdáleností výstupního neuronu od prostředního.

Síť byla naučena na 1200 obrázcích. Po naučení byla síť schopna řídit rychlostí 5 km/h na cestě vedoucí lesem v okolí Carnegie-Mellonovy univerzity. Rychlost byla omezena malým počítačem Sun-3, který byl použit k propagaci signálů přes síť, a mohla by být zvýšena specializovaným hardwarem. Každopádně to byla rychlost dvakrát vyšší, než jaké se podařilo dosáhnout jinými (nesíťovými) algoritmy.

3.5 Vhodná velikost sítě

Pro řešení konkrétního problému je vhodná jen síť určité velikosti. Pokud bychom použili síť příliš malou, nebyla schopna správné generalizace, nedokázala by proložit plochu všemi body příkladů. Naopak, pochud bychom použili síť příliš velkou, došlo by také ke špatné generalizaci.

První důvod je ten, že data příkladů z reálného světa vždy obsahují jistý šum, neodpovídají přesně té závislosti, kterou chceme zjistit, například v důsledku nepřesnosti měření hodnot. Příliš velká síť by se naučila z příkladů i tomuto šumu.

Druhý důvod je ten, že opravdu příliš velká síť by se naučila odpovídat naprosto správně jen na předložené příklady, ale ve zbytku prostoru vstupů by měla generalizační hyperplocha nesmyslný tvar. Odpovídá to situaci, kdy se student naučí nazpaměť zadané příklady, ale na otázku, která nebyla v příkladech, nedokáže odpovědět.

V ilustračním obrázku jsou případy sítě příliš malé, správně velké, trochu nadměrné a příliš velké.

4 Hopfieldův model

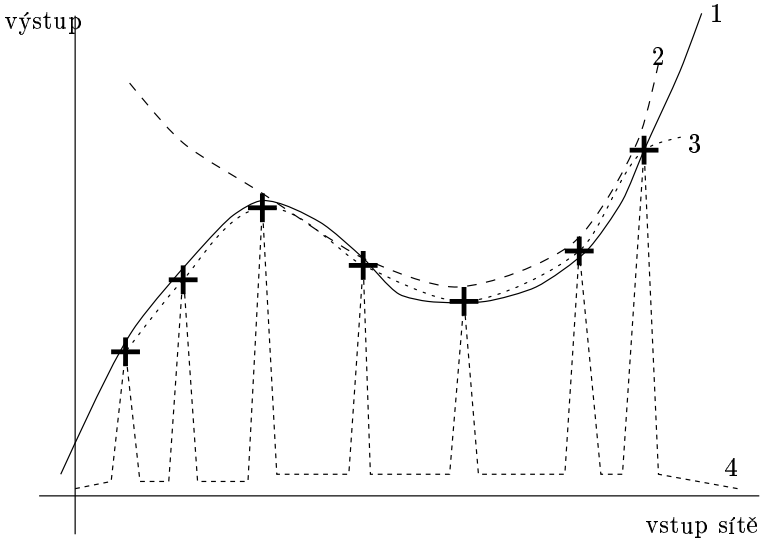
Hopfieldův model je krok směrem od biologické reality. Používá totiž *symetrické* spoje mezi neurony, které v přírodě neexistují. Patří spíš do teorie nelineárních dynamických systémů. Proto si nejdřív zavedeme několik pojmů z této teorie.

4.1 Pojmy nelineárních dynamických systémů

Nelineární dynamický systém sestává z množiny stavů P a z funkce $f : P \to P$, která určuje přechody mezi stavy.

Stacionární bod je takový stav, ze kterého se nikam jinam nedostaneme ($x \in P : f(x) = x$).

Atraktor je takový stacionární bod, že existuje nějaké jeho okolí (v nějaké metrice na množině stavů), z jehož všech stavů vždy skončíme v atraktoru. Atraktor je od slova atraktivní, je to stav, kterého se snaží systém dosáhnout a v něm zůstat. Okolí atraktoru se anglicky nazývá *basin of attraction.*



Obrázek 14: Příklady chybných generalizací. Křížky jsou příklady, čára 1 je správná generalizace, čára 2 odpovídá malé síti, čára 3 trochu větší a čára 4 příliš velké.

Cyklus je uzavřená cesta mezi stavy. *Globální atraktor* je cyklus, který má nějaké okolí, z něhož vždycky skončíme v cyklu.

4.2 Asociativní paměť

Hopfieldův model neuronové sítě byl vytvořen jako asociativní paměť. Je tvořena neurony, které jsou spojeny symetrickými spoji každý s každým[2]. Může být reprezentován symetrickou maticí vah s nulovou hlavní diagonálou.

Neurony mají dva stavy +1 (aktivovaný) a -1 (neaktivovaný) a provádějí prahovaný vážený součet

$$S_i := sgn(\sum_j w_{ij} S_j - \mu_i)$$

$$sgn(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

.

Hodnota prahu bude nulová. Učení spočívá v tom, že vezmeme nějaký vzor (pattern), například černobílý obrázek, provedeme přiřazení jeho pixelů na neurony a nastavíme je černá -1, bílá +1. Učící pravidlo je Hebbovo. Pan Hebb vyslovil na základě pozorování biologických neuronů pravidlo, že synaptické spojení mezi dvěma neurony, které jsou aktivovány ve stejnou chvíli, se posiluje.

4.2.1 Fáze učení

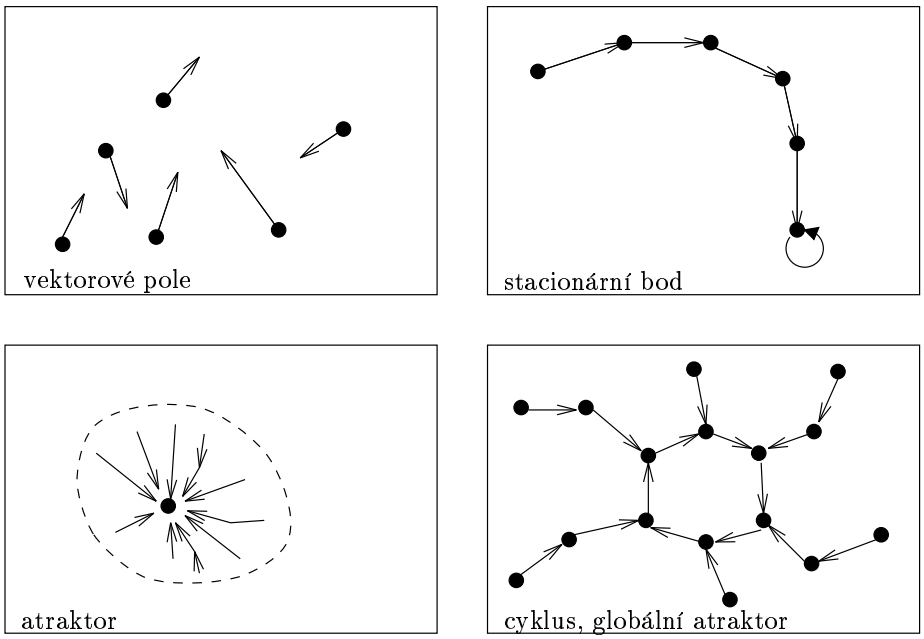
Na začátku budou všechny váhy synaptických spojení inicializovány na nulu. Přiřadíme neuronům hodnoty $\{+1; -1\}$. Změníme *všechny* váhy tak, že pokud spojuje neurony se stejnou hodnotou, zvýšíme hodnotu váhy o jedničku, pokud spojují neurony s rozdílnými hodnotami, hodnotu váhy o jedničku snížíme. Formálně zapsáno

$$\Delta w_{ij} = x_i . x_j$$

Tohle pravidlo už je mimo původní Hebbovu hypotézu, protože váha se mění i když oba neurony jsou neaktivní (-1,-1), což nemá biologické opodstatnění.

Pak vezmeme další vzor, přiřadíme hodnoty neuronům, změníme váhy. A tak dále se všemi vzory. Po naučení hodnota váhy vyjadřuje rozdíl počtu vzorů, ve kterých se jí spojené neurony shodly svými hodnotami, a počtu vzorů,

[2]U Hopfílda se spojí každý s každým – grupáč bez rozlišení pohlaví – mnemotechnická pomůcka



Obrázek 15: Pojmy dynamických nelineárních systémů

ve kterých se neshodly. Pokud jsme například měli 11 vzorů a neurony číslo 3 a 5 se u šesti vzorů shodli a u pěti vzorů neshodli, pak váha $w_{35} = w_{53}$ bude mít hodnotu +1. Kdyby se desetkrát neshodli a jednou shodli, bude váha mít hodnotu -9.

4.2.2 Fáze vybavování

Vezmeme některý vzor, který jsme síť naučili a poškodíme ho, například invertujeme třetinu bitů nebo jich náhodné množství nastavíme na náhodné hodnoty. Tento poškozený vzor přiřadíme na neurony.

Pak donekonečna opakujeme následující postup: Náhodně vybereme jeden neuron. Hodnoty ostatních neuronů vynásobíme synaptickými vahami, sečteme a pokud výsledek bude nezáporný, přiřadíme neuronu hodnotu +1, jinak -1. Jeho starou hodnotu zapomeneme a od teď používáme tuto novou hodnotu. Pak vezmeme další neuron a tak dále.

4.2.3 Princip vybavování

Tento postup vychází z následující úvahy. Měněný neuron se zeptá všech ostatních, jakou hodnotu by měl mít. Neuron, se kterým je spojen spojem s vahou 15 mu říká:

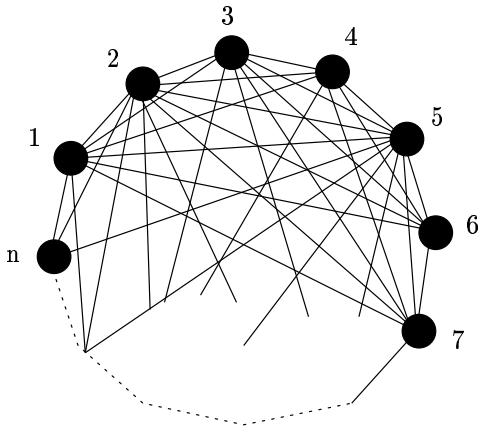
"Podívej, o patnáct víckrát jsme měli stejnou hodnotu než ji měli rozdílnou, tak je pravděpodobné, že ji budeme mít zase stejnou. Tak se laskavě nastav na stejnou hodnotu, jako mám já. A tuhle moji radu ber vážněji než rady neuronů, kteří jsou s tebou spojeni slapšími vahami."

Jiný neuron, se kterým je spojen vahou -4 povídá:

"Hele, my jsme se shodovali a neshodovali tak půl na půl, ale přece jenom počet neshod byl o čtyři vyšší, takže asi budeme mít rozdílné hodnoty. Ale tohohle doporučení si moc nevšímej, váha -4 není zas tak moc."

Náš neuron se tak zeptal všech, podle znamínka váhy určoval , jestli má mít stejnou nebo rozdílnou hodnotu než neuron, kterého se ptá, a protože byl rozený demokrat, každému dal v tomhle hlasování tolik hlasů, kolik činila absolutní hodnota váhy jeho spoje. Pak výsledky sečetl a podle výsledku se rozhodl buď pro +1 nebo -1.

Tento postup vybavování se vždy po konečném počtu kroků zastaví v některém z naučených vzorů. To, že jsme brali neurony jeden po druhém a ne naráz má velký význam, protože kdybychom nejdřív vypočítali nové hodnoty všech neuronů a pak je teprve nastavili, nebyla by konečnost zajištěna.



Obrázek 16: Hopfielův model. $w_{ij} = w_{ji}, w_{ii} = 0$

Konečnost postupu si dokážeme zavedením *energetické funkce*

$$E = -\sum_{ij} w_{ij} S_i S_J$$

Vypadá sice podobně jako chybová funkce ve vícevrstvých sítích, ale je to něco jiného. Energetickou funkci můžeme zavést jen u sítě se *symetrickými* váhami. Energetická funkce měří *energii* systému a platí, že *vždy klesá nebo zůstává stejná, pokud se systém vyvíjí podle svého dynamického pravidla.*

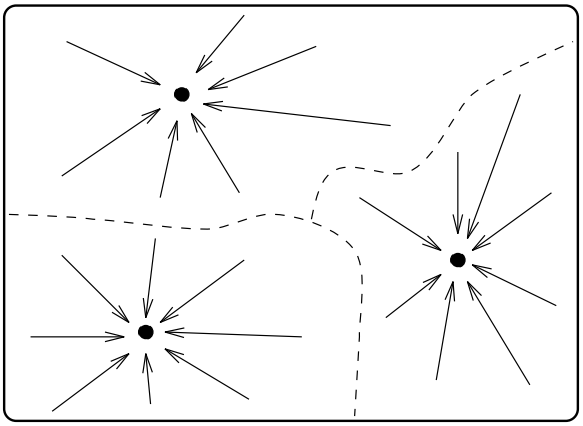
Důkaz konečnosti počtu kroků vybavování * Dokážeme si, že energie klesne vždy, když se změní hodnota neuronu. Nechť je S'_x nová hodnota stavu S_x pro nějaký neuron x.

$$S'_x = sgn(\sum_j w_{xj} S_j) \tag{3}$$

.

Pokud $S'_x = S_x$ (neuron se nezměnil), energie zůstala stejná. V opačném případě $S'_x = -S_x$, takže rozdíl energií je

$$\begin{aligned}
E' &= -(\sum_{i\neq x}\sum_{j\neq x} w_{ij} S_i S_j + \sum_i w_{ix} S_i S'_x + \sum_j w_{xj} S'_x S_j - w_{xx} S'_x S'_x) && (4)\\
&= -(\sum_{i\neq x}\sum_{j\neq x} w_{ij} S_i S_j + \sum_k (w_{kx} + w_{xk}) S_k S'_x - 0) && (5)\\
&= -(\sum_{i\neq x}\sum_{j\neq x} w_{ij} S_i S_j + \sum_k 2 w_{kx} S_k S'_x) && (6)\\
E &= -(\sum_{i\neq x}\sum_{j\neq x} w_{ij} S_i S_j + \sum_k 2 w_{kx} S_k S_x) && (7)\\
E' - E &= -\sum_k 2 w_{kx} S_k S'_x + \sum_k 2 w_{kx} S_k S_x && (8)\\
&= \sum_k 2 w_{kx} S_k (-S'_x + S_x) && (9)\\
&= \sum_k 2 w_{kx} S_k (2 S_x) && (10)\\
&= 4 S_x \sum_k w_{kx} S_k && (11)\\
E' - E &= 4 S_x \sum_j w_{jx} S_j && (12)
\end{aligned}$$



Obrázek 17: Prostor stavů sítě se třemi atraktory.

kde $\sum_j w_{jx} S_j$ má opačné znaménko než S_x podle 3, takže celý výraz je záporný. Tedy energie klesne pokaždé, když se změní hodnota S_x nějakého neuronu.

Stav binární Hopfieldovy sítě je ale vlastně binární číslo, tedy stavů je konečný počet. Když tedy síť mění svůj stav klesá její energetická funkce a proto nemůže dojít k zacyklení stavů, takže po konečném počtu kroků se musí dostat do stavu, kdy už žádné změny neuronů nemohou proběhnout a v něm se zastaví.

4.2.4 Kapacita asociativní paměti *

Energetickou funkci si opět můžeme představit jako plochu nad prostorem stavů sítě, jako "energetickou krajinu" (ENERGY LANDSCAPE). Lokální minima – údolí krajiny – jsou atraktory, do kterých se síť vždy dostane. Celý prostor stavů je tak rozdělen na bazény atraktorů. Schematicky to zobrazuje obrázek 17.

Otázka zní, jestli atraktory jsou zrovna ty vzory, které jsme si chtěli zapamatovat. Odpověď není jednoduchá. Volně zde opisuji závěr kapitoly plné výpočtů z knihy "Introduction to the theory of neural computation,Addison-Wesley, 1992":

Kapacita p_{max} (počet uschovatelných vzorů) je v přímé úměrnosti k N (počet neuronů), ale nikdy vyšší než $0.138N$, pokud se spokojíme s malým množstvím chybných bitů v každém vzoru; nebo je v přímé úměrnosti k $N/log(N)$, pokud trváme na tom, aby většina vzorů byla zapamatována bezchybně.

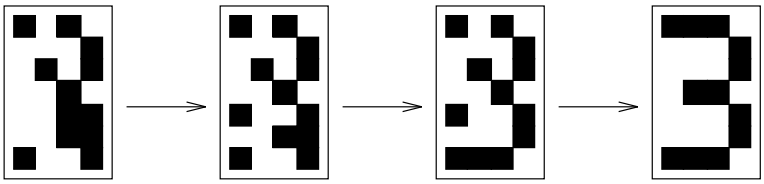
Kromě atraktorů odpovídajících zapamatovaným vzorům jsou ještě jiné atraktory.

- Ke každému zapamatovanému vzoru S^p je atraktorem také *inverzní stav* $(-S^p)$, který má stejnou energii. To není pro praktické použití zase takový problém, stačí všechny hodnoty zinvertovat.
- Stabilní jsou také tzv. *mixture states* (mixované stavy) S^{mix}, které odpovídají lineárním kombinacím lichého počtu vzorů. Nejjednodušší případ je symetrická kombinace tří zapamatovaných vzorů:

$$S_i^{mix} = sgn(\pm S_i^{p_1} \pm S_i^{p_2} \pm S_i^{p_3})$$

  Všech osm kombinací je možných. Hammingova vzdálenost (počet lišících se bitů) stavu S^{mix} od všech tří vzorů $S_i^{p_1}, S_i^{p_2}, S_i^{p_3}$ je $N/4$, leží tedy na místě stejně vzdáleném od všech komponent. Podobně je to u 5,7,... stavů.
- Pro příliš velké počty zapamatovaných stavů existují ještě další atraktory, které nemají nic společného se zapamatovanými vzory, říká se jim *spin glass states*, což je převzato odněkud ze statistické mechaniky.

Naštěstí *mixture states* a *spin glass states* mají mnohem menší bazény(?) atrakce než zapamatované vzory, takže je to celkem smůla, když do nich spadneme. Navíc byly vypracovány technické triky, které umožňují tato minima zmenšit nebo odstranit.



Obrázek 18: Postupné vybavování zapamatované číslice

4.2.5 Shrnutí

Na rozdíl od vícevrstvých sítí perceptronů, které dávají odpověď ihned, Hopfieldův model potřebuje nějaký čas, aby se ustálil v nějakém stabilním stavu. Postup vybavování tvarů zapamatovaných číslic je na obrázku 18. Kromě základního Hopfieldova modelu existují jeho rozšíření, která umožňují používat místo binárních reálné hodnoty, nebo které si místo jednotlivých stabilních stavů pamatují celé *sekvence* stavů. Hardwarové implementace asociativní paměti jsou různé, v současnosti je ve vývoji metoda, která jako matici vah používá hologram. To umožňuje dosáhnout hustoty až 10^{12} spojů na cm^3, bohužel vybavování je částečně destruktivní, takže uschované vzory musí být po čase obnovovány.

4.3 Optimizace pomocí sítě Hopfieldova typu

Hopfieldova síť je vlastně dynamický systém, u kterého umíme měřit jeho energii. O té víme, že nikdy neroste. To vedlo k nápadu použít tuto síť pro řešení problémů, ve kterých jde o minimalizaci nějaké funkce. Použitelnost si ukážeme na problému obchodního cestujícího.

Obchodní cestující má za úkol procestovat m měst tak, aby v každém byl pouze jednou a urazil co nejmenší vzdálenost. Problém vyřešíme pomocí Hopfieldovi sítě s n neurony, kde $n = m^2$ a každý neuron reprezentuje spoj mezi dvěma městy. Neurony máme uspořádány do čtvercové matice a hodnota neuronu udává, jak moc daný spoj mezi městy patří do výsledné cesty.

Ve výsledné matici musí být vždy nanejvýš jeden nenulový neuron v každém řádku a nanejvýš jeden nenulový neuron v každém sloupci, aby se matice dala interpretovat jako okružní cesta mezi městy.

Úkolem sítě je minimalizovat energetickou funkci E navrženou tak, aby byla minimální právě když jsou splněny požadavky na výslednou matici.

$$E = \underbrace{A\sum_{r}\sum_{ij} x_{ri}.x_{rj}}_{1} + \underbrace{B\sum_{r}\sum_{ij} x_{ir}.x_{jr}}_{2} + \underbrace{C((\sum_{ij} x_{ij}) - m)^2}_{3} + \underbrace{D\sum_{rs}(\sum_{i} d(r,s)x_{ri}(x_{s,i+1} + x_{s,i-1}))}_{4}$$

1 První člen roste s počtem nenulových neuronů v jednom řádku.

2 Druhý člen roste s počtem nenulových neuronů v jednom sloupci.

3 Třetí člen je nejmenší pro právě m spojů mezi městy.

4 Kdybychom použili jen první tři členy, výsledkem by byla nulová matice. Proto čtvrtý člen počítá délku cesty.

A, B, C, D jsou parametry (konstanty) určující, jak velký důraz dávám na jednotlivá omezení. Síť není binární, neuron může určovat, že spoj mezi městy patří do hledané cesty tak napůl.

Jak teď získám síť minimalizující právě tuto zběsilou energetickou funkci ? Jednoduše, všimnu si, že moje energetická funkce

$$E_1 = A\sum\sum x_{ri}x_{rj} + B\sum\sum + C(\sum)^2 + D\sum$$

a energetická funkce Hopfieldovi sítě

$$E_2 = -\sum\sum w_{ij}x_i x_j$$

jsou obě funkce ve stejných proměnných x, takže váhy w_{ij} v E_2 můžu rovnou vypočítat z E_1. Takže síť pro řešení problému obchodního cestujícího neučím, ale váhy rovnou vypočítám a mám síť hotovou. Potom jenom nechám síť ustálit a jedničkové neurony mi označují přímé cesty mezi městy, které tvoří krátkou okružní jízdu. Tento postup dává dobrá suboptimální řešení.

5 Učení samoorganizací

V následujících kapitolách opustíme učení s učitelem a zaměříme se na sítě, které se učí *bez učitele.*

Zavedeme si zkratky LTM (long term memory) a STM (short term memory). Dlouhodobá paměť LTM sestává z nastavení synaptických vah, které se pomalu mění, kdežto krátkodobá paměť STM je tvořena okamžitým stavem vzruchů, který se každým okamžikem proměňuje.

Při učení bez učitele síť dostává na vstup množinu podnětů, které si sama utřídí. Například rozdělí podněty do skupin podle podobnosti a určí typického zástupce skupiny (model ART), nebo síť svou konfigurací začne vystihovat topologii prostoru vstupů (Kohonen).

Při učení samoorganizací jsou vektory LTM a STM stejného typu. Učení probíhá podle vzorce

$$LTM_{new} = LTM_{old} + \alpha(STM - LTM_{old})$$

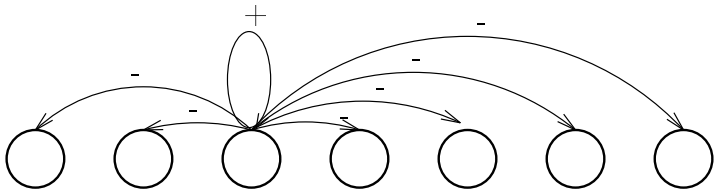
tedy dlouhodobá paměť se změní úměrně rozdílu přicházejícího podnětu od zapamatovaných hodnot. Parametr α se nazývá *koeficient plasticity* a jeho extrémní hodnoty jsou nula, kdy se síť není schopná učit, a jedna, kdy si síť nic nezapamatuje dlouhodobě, nové podněty všechno přemazávají.

5.1 Laterální inhibice

Laterální znamená vrstvový, inhibice je působení, které něco snižuje, můžeme tedy laterální inhibici volně přeložit jako *vzájemné tlumení v rámci vrstvy.*

Při laterální inhibici jsou neurony v jedné vrstvě propojeny navzájem tak, že každý neuron působí kladnou (excitační) vazbou sám na sebe a zápornou (inhibiční) vazbou na ostatní neurony. Aktivovaný neuron tak svoji vlastní aktivaci posiluje, zatímco aktivaci ostatních neuronů tlumí. Vazby mezi neurony jsou naznačeny na obrázku 19.

Výsledek vzájemného působení neuronů při laterální inhibici je, že nejsilněji aktivovaný neuron utlumí všechny ostatní a zůstane jediným aktivovaným neuronem. Tento princip se používá tehdy, když mnoho neuronů reaguje na stejný podnět a my chceme zjistit, který z nich zareagoval nejvíce. Princip jediného zvítězivšího neuronu se v angličtině označuje jako WINNER TAKES ALL.



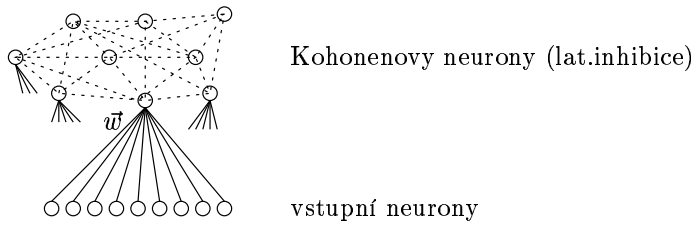
Obrázek 19: Laterální inhibice

6 Kohonenovy mapy

Kohonenova síť je nakreslena na obrázku 20. Je tvořena vrstvou *n vstupních neuronů*, které slouží jen k načtení podnětů představovaných n-prvkovými vektory $\vec{x} = (x_1, \ldots, x_n)$ a druhou vrstvou *kohonenových neuronů*, které jsou vzájemně spojeny vazbami laterální inhibice.

Do každého z kohonenových neuronů přichází spoje ze všech vstupních neuronů, každý kohonenův neuron tedy čte vstupní vektor $\vec{x}$. Vstupy jsou násobeny synaptickými vahami, takže každému kohonenovu neuronu i přísluší *vektor synaptických vah* $\vec{w}_i$. Podstata Kohonenova modelu spočívá v tom, že vektor vah je stejně jako vektor vstupu n-prvkový. Pro velké formální zjednodušení budeme používat *normalizované* vektory $\vec{w}$ a $\vec{x}$, tedy $|\vec{x}| = |\vec{w}| = 1$.

Prostor vah je totožný s prostorem vstupů. Protože jsou vektory normalizované, je tento prostor povrch n-rozměrné hyperkoule. Vektory vah $\vec{w}$ jsou body na této hyperkouli.

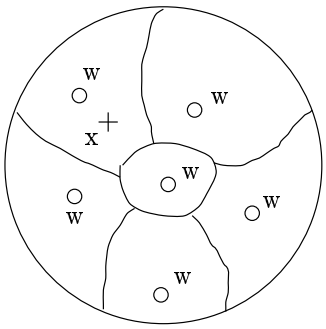


Obrázek 20: Kohonenova síť

6.1 Funkce Kohonenovy sítě

Přišlý vstup $\vec{x}$ aktivuje všechny kohonenovy neurony. Hodnota excitace je dána skalárním součinem $\vec{w}.\vec{x}$ [3] Největší excitaci má ten neuron, který je na hyperkouli nejblíž vstupu. [4]

Kohonenovy neurony se navzájem ovlivňují laterální inhibicí, takže po krátké době se síť ustálí ve stavu, kdy bude excitovaný jen jeden neuron a to ten, který byl na počátku excitovaný nejvíc, tedy ten nejbližší vstupu. Tomuto aktivovanému neuronu se říká *grandmother cell*, což pochází z diskusí, jestli je v mozku neuron reagující vždy právě ve chvíli, když člověk vidí svoji babičku.



Obrázek 21: Rozdělení prostoru vstupů na oblasti příslušné jednotlivým kohonenovým neuronům

Povrch hyperkoule je tak rozdělen na jakési Voroného oblasti příslušné kohonenovým neuronům, jak je naznačeno na obrázku 21. Na vstup zareaguje vždy jen ten neuron, do jehož oblasti vstup spadl.

6.2 Učení kohonenovy sítě

Objasnili jsme si, že v Kohonenově síti reaguje na vstup jen ten neuron, který je vstupu "nejpodobnější". Teď si povíme, jak rozmístění neuronů v prostoru vstupů vlastně vzniká.

Na počátku, při vzniku sítě, jsou neurony shromážděny blízko sebe někde na hyperkouli. Při příchodu vstupu $\vec{x}$ nejvíce zareaguje neuron s příslušným vektorem synaptickýh vah $\vec{w}$. Tento vybraný neuron se zadaptuje podle předpisu

$$\vec{w}_{new} = \vec{w}_{old} + \alpha(\vec{x} - \vec{w}_{old})$$

tedy posune se směrem k bodu představujícímu vstup. O kolik se posune záleží na parametru α. Při $\alpha = 1$ se přesune přímo do bodu vstupu, při $\alpha = 0$ se ani nehne a při $\alpha = 0.5$ se posune na poloviční vzdálenost.

Zde přichází takzvaný *konflikt stability a plasticity*. Při malém α se síť rychle učí, ale kvůli novým poznatkům zapomíná staré, při velkém α naopak. Používá se tedy postupně klesající α, v "mládí" sítě vysoké, ve "stáří" nízké.

[3] Pro zapomětlivé: skalární součet je suma součinů odpovídajících si složek vektorů, tedy $\vec{w}.\vec{x} = \sum_{i=1}^{n} w_i x_i$

[4] Vektor vstupu představuje bod na hyperkouli. Vektor vah $\vec{w}$ nějakého neuronu si lze představit buď také jako bod na hyperkouli, nebo jako vektor vycházející ze středu koule do bodu na hyperkouli. Vektor vah tak vlastně určuje rovinu, která prochází středem hyperkoule a vektor vah je jejím normálovým vektorem. Při vynásobení $\vec{w}.\vec{x}$ je výsledek vzdáleností bodu $\vec{x}$ od roviny určené vektorem $\vec{w}$. Vzhledem k tomu, že bod $\vec{x}$ musí ležet na hyperkouli, bude jeho vzdálenost od roviny největší (a rovna 1) právě tehdy, když bude totožný s průsečíkem hyperkoule s vektorem $\vec{w}$, tedy když bude totožný s bodem $\vec{w}$.

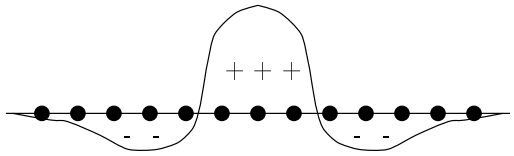
Kohonenova síť se učí celý život, nemá fázi učení a fázi používání jako perceptrony nebo Hopfieldův model, učí se za provozu.

Smyslem Kohonenovy sítě je vystihnout charakter množiny vstupů. Představme si, že množina vstupů tvoří na hyperkouli jakési shluky, tzv.*klastry* (clusters). Pokud je kohonenových neuronů stejný počet jako shluků, každý neuron si najde "vlastní" shluk a usídlí se v jeho středu, stane se jeho typickým zástupcem.

Navíc tam, kde bude větší hustota vstupů, bude i více neuronů. Dá se to představit na analogii s obchody s potravinami. Tam, kde je lidí málo, na samotách, obchody skoro vůbec nebudou, naopak ve městech, kde je vysoká hustota lidí, bude obchodů mnoho a jejich hustota na čtvereční kilometr bude zhruba odpovídat hustotě lidí.

Kohonenova síť se používá na roztřizování vstupů do skupin, přičemž tyto skupiny si síť sama vytvoří. Nedostatkem Kohonenova modelu je to, že počet neuronů musíme určit *předem*. Tento nedostatek odstraňuje ART v další kapitole.

Vylepšení učení Kohonenovy sítě se dá dosáhnout tím, že na vstup nebude reagovat jen jeden neuron, ale víc. Kohonenovy neurony kromě *logického* umístění na hyperkouli mají také *fyzické* umístění v reálném prostoru, třeba mozku. (Tím se neříká, že v mozku jsou Kohonenovy neurony, ale co kdyby byly.) Při laterální inhibici neoslabuje všechny neurony stejně, ale v závislosti na fyzické vzdálenosti podle křivky na obrázku 22. Velikost fyzického okolí se časem zmenšuje kvůli stabilitě.

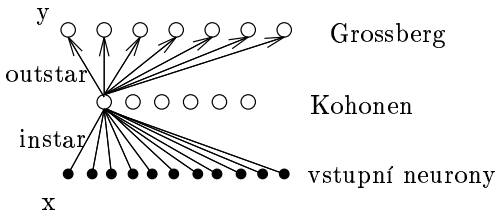


Obrázek 22: Křivka "mexického klobouku" při laterální inhibici

Kohonen zkoušel vyrobit phonetic typewriter — psací stroj ovládaný hlasem.

6.3 Counter propagation

Counter propagation je rychlejší než back-propagation, ale vytváří méně přesné prototypy. Používá Kohonenovu síť nadstavenou o další vrstvu výstupních neuronů, které se učí Grossbergovým algoritmem. Spoje sbíhající se ze vstupních neuronů do jednoho kohonenova neuronu se nazývají *instar* (do-hvězda), spoje od kohonenova neuronu ke Grossbergovým neuronům se nazývají *outstar*.



Obrázek 23: Counter-propagation

Na vstup a výstup dáme tréninkovou dvojici $(\vec{x}, \vec{y})$. V kohonenově vrstvě na vstup zareaguje právě jeden neuron, který vyšle jedničkový impuls přes "outstar" k horní vrstvě Grossbergových neuronů, impuls je vynásoben synaptickými vahami "outstar". Výstup se porovná s požadovaným výstupem $\vec{y}$ a modifikují se váhy "outstar" podle Grossbergova předpisu

$$\vec{w}' = \vec{w} + \beta(\vec{y} - \vec{w}).k$$

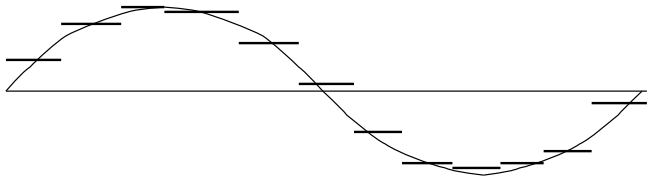
Takže síť vlastně vytváří průměrnou odpověď na členy nějakého klastru. Kohonenova vrstva rozděluje vstupy do klastrů, na všechny vstupy z jednoho klastru zareaguje vždy ten stejný neuron. Grossbergova vrstva pak pro

klastr vyrábí takovou odpověď, která vznikla postupným spojením vektorů $\vec{y}$ příslušných vstupům v klastru. Síť opět nemá odděleny fáze učení a práce, učí se po celou dobu. Je samozřejmě možné od určitého okamžiku zastavit učení nastavením parametrů α a β na nulu.

Counter propagation je možné modifikovat neúplnou laterální inhibicí nebo další transformací na Grossbergových neuronech.

Counter propagation je velice rychlý, používá se před backem pro orientaci. Kohonenova vrstva zajišťuje vystižení struktury vstupního prostoru, Grossbergova vrstva transformuje zástupce klastrů do výstupního prostoru.

Příklad — síť byla učena na funkci *sin*, jako vstupy jí byly předkládány hodnoty z intervalu $< 0, 2\pi >$, jako výstupy odpovídající hodnoty *sin*. Neurony Kohonenovy vrstvy se rovnoměrně rozprostřely po intervalu, rozdělily ho na mnoho malých podintervalů. Pro každý podinterval pak Grossbergova vrstva vyrobila průměrnou odpověď. Výsledek je na obrázku 24.



Obrázek 24: Výsledek conterpropagation při učení funkce *sin*

7 Adaptive Resonance Theory

Tento model se používá pro *kategorizaci* vstupních dat do kategorií se současným tvořením *prototypů* – idealizovnaých zástupců kategorií.

Pokud předkládáme Kohonenově síti další a další data, nemáme zajištěnu *stabilitu* již vytvořených kategorií dat. Nová data mohou kategorie změnit. Můžeme samozřejmě snížit učící parametry na nulu a tím zmrazit stav sítě. Ale pak síť ztratí svoji *plasticitu*, schopnost reagovat na nová data. Není jednoduché mít zároveň stabilitu již naučeného a plasticitu reagování na nové podněty. Toto je Grossbergovo **dilema plasticity a stability**.

Abychom vytvořili nějaký reálný systém schopný fungovat a přitom reagovat na měnící se prostředí, musíme se s tímto dilematem vyrovnat. Další problém spočívá v tom, kolik výstupních buněk (každá pro jednu kategorii) máme vlastně použít. Pokud jich dáme pevný počet, pak pokud bude kategorií více než buněk, bude kategorizace špatná. Naopak, pokud jich dáme nekonečně mnoho, pro každý vstup se vytvoří vlastní kategorie, což je taky k ničemu. Dobré řešení je mít zásobárnu nekonečně mnoha buněk, ale *nepoužívat je, dokud nejsou potřeba.* V každém okamžiku bude použito právě tolik buněk, kolik je dosud známých kategorií, a pokud se objeví nová kategorie, použijeme další buňku.

Pánové Carpenter a Grossberg v roce 1988 vyvynuli modely ART1 a ART2, které se chovají tímto způsobem. ART akceptuje vstup jen tehdy, je-li dostatečně podobný některému již existujícímu prototypu nějaké kategorie. Pokud není, vytvoří se nová kategorie se vstupem jako svým prototypem a je použita nová výstupní buňka.

Význam slov "dostatečně podobný" záleží na parametru *vigilance* (bdělosti) ρ, kde $0 < \rho \leq 1$. Pokud je vigilance velká, je podobnost posuzována velmi přísně a vzniká mnoho přesně vymezených kategorií, při malé vigilanci dochází k lepší abstrakci. Hodnotu vigilance je možno během učení měnit.

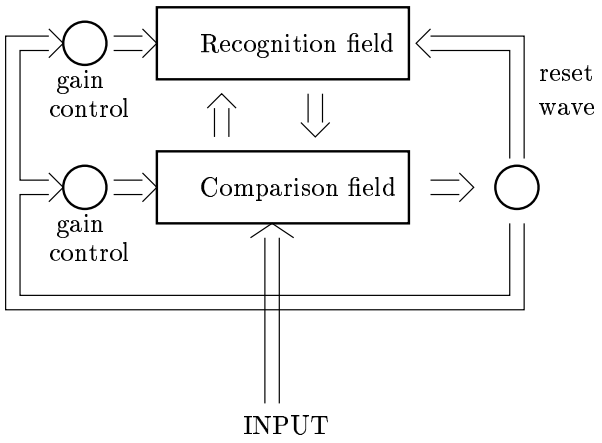
ART1 pracuje s binárními vstupy, ART2 s reálnými. Budeme se zabývat ART1, protože je jednodušší.

Popišme si ART1 jako algoritmus. Mějme vstupní vektory $\vec{x}$ a uložené vektory prototypů $\vec{w}_i$, všechny N-prvkové s binárními hodnotami. i indexuje výstupní neurony (tedy kategorie), každý z nich může být buď *enabled* nebo *disabled.*

Začneme nastavením $\vec{w}_i = \vec{1}$ na jedničkové vektory pro všechna i; jedničkový vektor bude reprezentovat nepoužitý (uncommited) výstupní neuron, ne kategorii. Pak předložíme síti vstup $\vec{x}$ a provádíme algoritmus:

1. Označme všechny výstupní neurony jako *enabled.*

2. Najděme mezi *enabled* neurony *babičku* i^*, přičemž babička je neuron s maximálním $\overline{\vec{w}}_i.\vec{x}$, kde $\overline{\vec{w}}_i$ je normalizovaný $\vec{w}_i$. Všimněme si, že nepoužitý neuron nakonec zvítězí (stane se babičkou), pokud se předtím nenajde žádný vhodnější.



Obrázek 25: Adaptive Resonance Theory

3. Otestujme, jestli je shoda mezi vstupním vektorem $\vec{x}$ a prototypovým vektorem $\vec{w}_{i^*}$ dostatečná výpočtem poměru

$$r = \frac{\vec{w}_{i^*}.\vec{x}}{\sum_j x_j}$$

Je to poměr jedničkových bitů z $\vec{x}$ které jsou zároveň v $\vec{w}_{i^*}$. Pokud je $r \geq \rho$, tedy větší než vigilance, dochází ke shodě (resonanci) a pokračuj krokem 4. Jinak je prototypový vektor $\vec{w}_{i^*}$ odmítnut. Označ neuron i^* jako *disabled* a vrať se na 2.

4. Upravíme babičku $\vec{w}_{i^*}$ vynulováním všech bitů, které nejsou zároveň v $\vec{x}$. Je to operace AND, nazývá se to *maskováním* vstupu.

Tento algoritmus může skončit jedním ze tří způsobů. Buď nalezneme odpovídající prototypový vektor, upravíme ho (pokud je to nutné) a výstupem je kategorie i^*. Nebo nenalezneme žádný prototyp, pak je použit jeden z ještě nepoužitých vektorů a je nastaven na $\vec{x}$; výstupem je nová kategorie i^*. Nebo v kroku dva už nezbývá žádný volný nepoužitý neuron, a pak vstup ignorujeme.

Smyčka z kroku 3 zpět do kroku 2 hledá mezi prototypovými vektory ten nejbližší, druhý nejbližší, atd. dokud nenalezne ten, který splňuje kritérium $r \geq \rho$. Toto hledání je pomalé, ale provádí se jen do doby, než se vytvoří všechny kategorie obsažené ve vstupních datech. Po vytvoření všech kategorií se najde příslušná kategorie na první pokus a skok z kroku 3 na 2 se již nikdy neprovádí. Všechny kategorie budou vytvořeny po konečném počtu kroků, což plyne z binarity hodnot a toho, že v kroku 4 se bity vždy jen odstraňují a nikdy nepřidávají.

Konkrétní implementace neuronovou sítí je na obrázku 25. Síť sestává ze dvou vrstev, *comparison field* a *recognition field*. Comparison field dostane vstup. Pošle ho recognition field, kde jeden neuron (babička) zareaguje a pošle zpátky své očekávání, tedy svůj prototypový vektor. Comparison field porovná vstup s očekáváním babičky a pokud jsou si dostatečně podobné, babička se zadaptuje. Pokud ne, babička se zneaktivní (disabled) a znovu se vstup pošle do recognition field. To se opakuje tak dlouho, až se nalezne vhodná babička. Pokud se žádná nenajde, vytvoří se nový neuron, který dostane vstup jako svůj prototyp. Celý systém je možné realizovat čistě neuronálně, včetně cyklů, k tomu slouží pomocné neurony označené gain control.

Adaptace se od Kohonena liší v tom, že babička se nemění jako vektor, ale začne tolerovat i věci, které se jí předtím moc nelíbily. Problém stability a plasticity je vyřešen jednak tím, že přibývají výstupní neurony a také tím, že při učení se zvyšuje míra abstrakce.

8 Genetické algoritmy

Mějme množinu P o r objektech, kde každý objekt závisí na n parametrech $(x_1, \ldots, x_n)$, přičemž o každém objektu můžeme říci jak moc je dobrý pomocí kriteriální hodnotící funkce $f(x_1, \ldots, x_n)$. Úkol je najít hodnoty parametrů tak, aby f byla maximální.

Metody jsou: systematické prohledávání prostoru parametrů, náhodný výběr, gradientní metody.
Nebo můžeme použít genetický algoritmus. Nejdřív musíme binárně zakódovat parametry.

Binární kódování

O1=101110011010101100=4
O2=000110001010001101=3
O3=000101011010001100=4.2

Každý rys jedince (parametr) x_i je kódován řetězcem nul a jedniček. Různé parametry jsou různě významné (barva očí, délka drápů). Množina P je *populace*, množina jedinců.

Genetický algoritmus

1. Náhodně zvolíme nultou gewneraci P_0 a stanovíme síly jedinců
2. tvoříme nové jedince křížením, např O1,O2 – rodiče, O4 – potomek:
   O2=0001/1000/1010/0011/01=3
   O3=0001/0101/1010/0011/00=4.2
   O4=0001/0101/1010/0011/01=
3. provedeme náhodnou mutaci z O4 na O5:
   O4=0001/0101/1010/0011/01=
   O5=0001/0101/1010/1011/01=
   Volby rodičů, genů i mutací jsou pravděpodobnostní.
4. Odstraníme jedince malé síly z populace tak, aby velikost populace zůstala zachována, např. z $P_0 = (O1, O2, O3)$ vznikne $P_1 = (O1, O3, O4)$, když síla $O4 = 3.7$.
5. a opakujeme

Občas musíme v populaci zachovávat i mrzáky, aby se prostor jedinců prohledával šířeji, jinak skončíme rychle, ale v lokálním maximu.