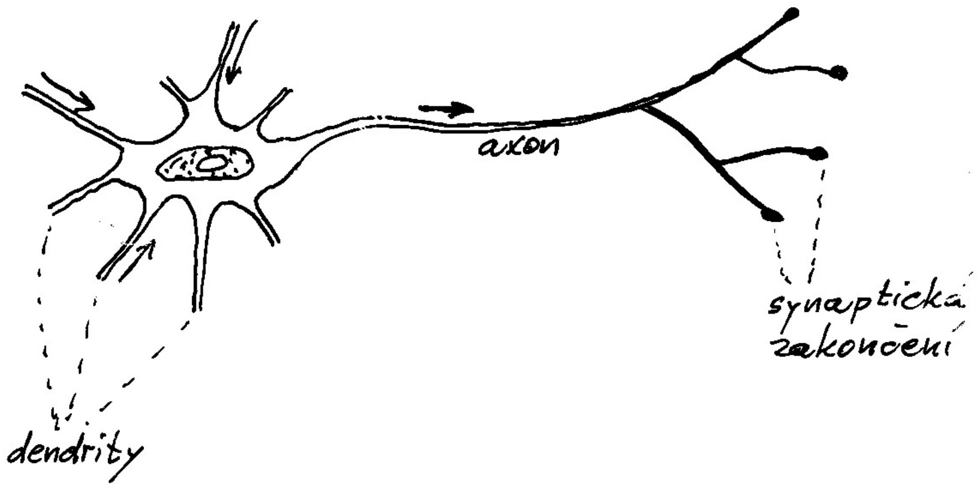
NEURON

Umělý perceptron je inspirován svým biologickým protějškem – neuronem, který je jedním z prvků nervové soustavy a tvoří základ mozku.

Neexistuje „typický neuron", avšak následující obrázek schematicky znázorňuje vlastnosti sdílené mnoha neurony:



Dendrity tvoří vstupy neuronu, axon je výstupem. Synaptická zakončení tvoří spoj k jiným neuronům či efektorům.

Aktivita receptorů modifikuje membránový potenciál dendritů a těla buňky. Elektrický impuls membránového potenciálu se šíří podél axonu a aktivuje synaptická zakončení, která následně modifikují membránový potenciál dalších neuronů či svalových vláken (efektorů).

Potenciálový rozdíl se šíří skrze membránu obalující nervovou buňku, a se vzdáleností zaniká. Pokud ovšem napětí převýší jistou hodnotu zvanou „práh", pak se elektrický signál může dále šířit bez poklesu své velikosti.

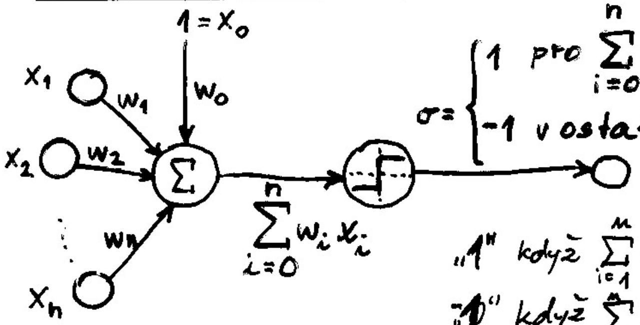
Většina synapsí má chemickou povahu. Elektrický signál, který dosáhne synapsi, způsobí uvolnění molekul zvaných transmitéry (přenašeče), které mohou mít buď excitační nebo inhibiční účinek. Excitační účinek „posune" neuron směrem k jeho prahu, zatímco inhibiční naopak.

Koncept umělého neuronu pochází z r. 1943 (autoři McCuloch a Pitts). Tehdejší umělý neuron byl popsán jako binární diskrétní (časově) element. Jeho výstup (log. „0" nebo „1") je ovlivněn hodnotou součtu vstupů – pokud součet převýší určitou prahovou hodnotu, na výstupu umělého neuronu se objeví „1", jinak „0".

V r. 1957 rozšířil Rosenblatt model umělého neuronu na tzv. perceptron zejména tím, že zavedl tzv. váhy spojů mezi neurony. V principu každá váha určuje sílu vztahu mezi dvojicí neuronů. Změna hodnoty váhy umožňuje se neuronu adaptovat na změněné podmínky a tím také se učit.

Spojením neuronů do sítě vznikne struktura schopná učení. Existuje mnoho typů sítí a učících metod.

PERCEPTRONY



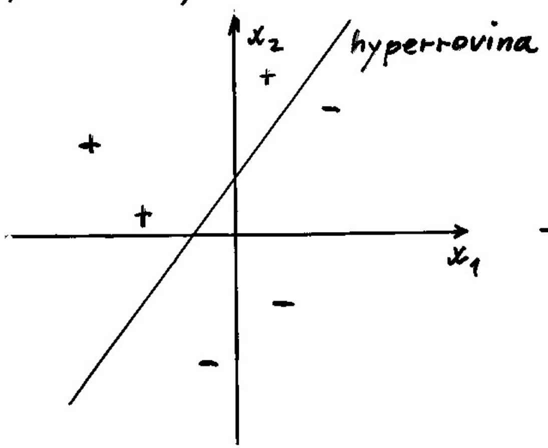
$$\sum_{i=0}^{n} w_i x_i$$

$$\sigma = \begin{cases} 1 & \text{pro } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{v ostatních případech} \end{cases}$$

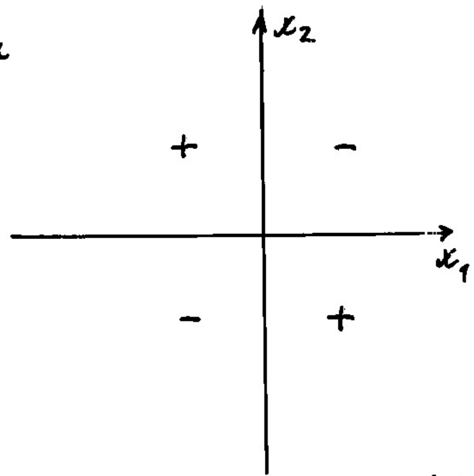
„1" když $\sum_{i=1}^{n} x_i w_i > t$ ... práh

„-1" když $\sum_{i=1}^{n} x_i w_i \leq t$ $\quad (w_0 = -t)$

Perceptron representuje rozhodovací hyperrovinu v n-rozměrném instančním prostoru (instance = bod). Výstupem perceptronu je 1 pro instance nacházející se na jedné straně hyperroviny a -1 pro instance na druhé straně:

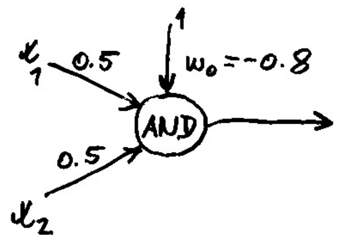


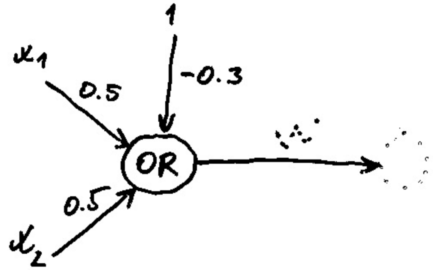
a) soubor trénovacích instancí a hyperrovina perceptronu korektně klasifikující instance



b) soubor trénovacích instancí, které nejsou lineárně separabilní (XOR)

Jednoduchý perceptron lze využít pro realizaci mnoha booleovských funkcí. Je-li TRUE = 1 a FALSE = -1, pak např. nastavení vah $w_0 = -0.8$, $w_1 = w_2 = 0.5$ realizuje AND. Změnou $w_0 = -0.3$ dosáhneme OR.





Perceptrony AND a OR jsou speciálními případy tzv. m-z-n funkcí (nejméně m z n vstupů musí být TRUE aby výstup byl TRUE).
AND odpovídá $m = n$, OR odpovídá $m = 1$.

Perceptrony (mohou) representovat primitivní booleovské funkce AND, OR, NAND (¬AND), NOR (¬OR).

Některé funkce však representovat NELZE, např. XOR (exklusivní OR: výstup je 1 pouze když $x_1 \neq x_2$).

Schopnost representace AND, OR, NAND, NOR je důležitá proto, že KAŽDOU booleovskou funkci lze realizovat nějakou sítí propojených základních jednotek (stačí dvouvrstvá síť). Lze použít např. DNF (disjunktivní normální formu) jako disjunkci (OR) konjunkcí (AND) vstupů a jejich negací (negace AND ... změna znaménka w_i).

Umělý perceptron: representace boolských funkcí

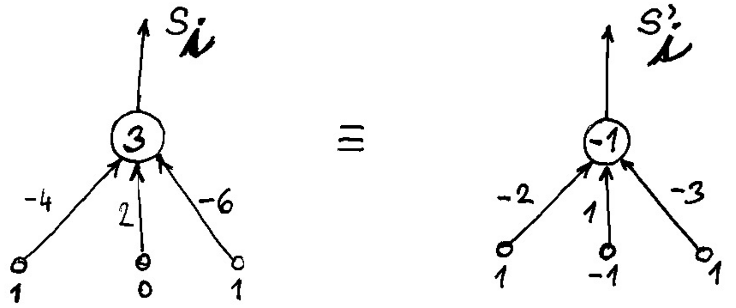
+1 ... true (logická "1")
-1 ... false (-"- "0")
(0 ... neznámo)

} možný způsob representace boolských hodnot

+1 ... T
0 ... F

} jiný možný (standardní) způsob

Obě formy jsou ekvivalentní:



$$w'_{ij} = \frac{1}{2} w_{ij} \quad \text{(verze bez posuvu)}$$

$$w'_{i0} = w_{i0} + \sum_{j=1}^{n} w'_{ij} = w_{i0} + \frac{1}{2} \sum_{j=1}^{n} w_{ij}$$

} přepočet vah (a posuvu)

$(-4 \cdot 1) + (2 \cdot 0) + (-6 \cdot 1) = -10$
$-10 + 3 = \underline{\underline{-7}}$

$(-2 \cdot 1) + (1 \cdot (-1)) + (-3 \cdot 1) = -6$
$-6 - 1 = \underline{\underline{-7}}$

$$\underline{\underline{S_i = S_i'}}$$

Pozn.: pokaždé, když změníme v systému {0, 1} vstup z F na T (0 → 1), zvýšíme S_i o w_{ij}. V systému {-1, 1} je S_i' zvýšeno o $2w_{ij}$. Z požadavku na $S_i = S_i'$ plyne výše uvedené.

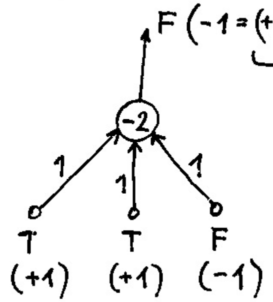
Obvykle se dává přednost systému {-1, +1}, protože v tomto případě probíhá trénování (sítě) perceptronů rychleji:

Je-li vstup $u_j = 0$ v systému {0, 1}, pak nedojde k žádné opravě váhy (násobení nulou). V systému {-1, +1} je však $u_j = -1$, takže w_{ij} se změní. To obvykle vede k rychlejší konvergenci.

Je-li zapotřebí v systému {0, 1} pracovat s hodnotou "neznámo", lze použít 1/2.

Modely s jednou jednotkou

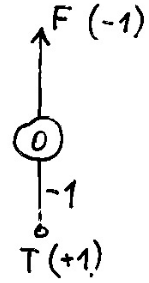
Tyto lineární modely mohou počítat většinu běžných boolských funkcí:



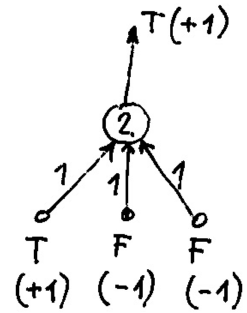
$$F\ (-1 = \underbrace{(+1 \cdot 1) + (+1 \cdot 1) + (-1 \cdot 1)}_{1+1-1=1})$$

$1 + (-2) = -1$

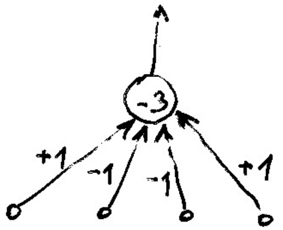
AND



NOT

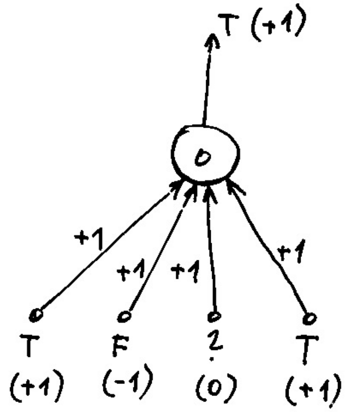


OR



SELEKTOR

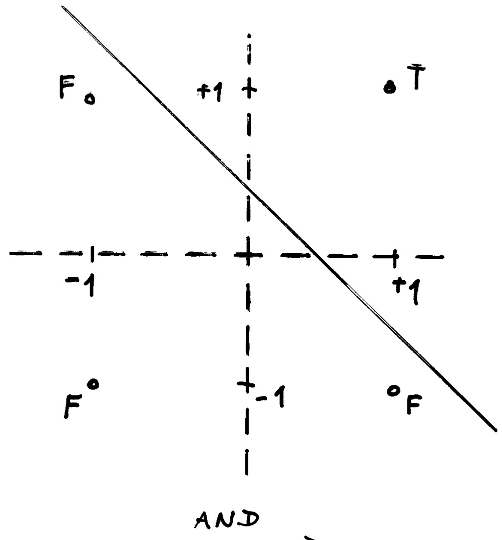
Na výstupu je +1 právě pro jedinou vstupní kombinaci (zde +1, −1, −1, +1).

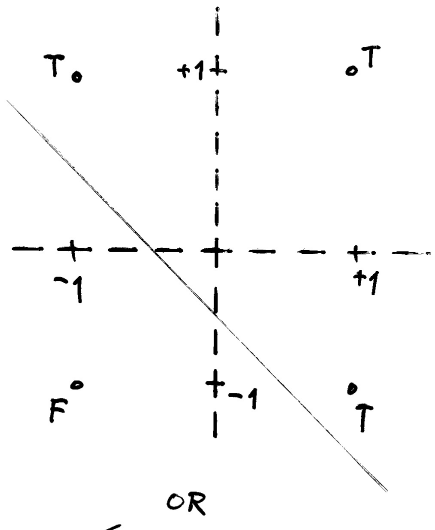


Na výstupu je T (+1) tehdy, když většina vstupů je T (více T než F), F(−1) když je více F než T, a ? (neznámo, 0) pokud je stejně T i F.

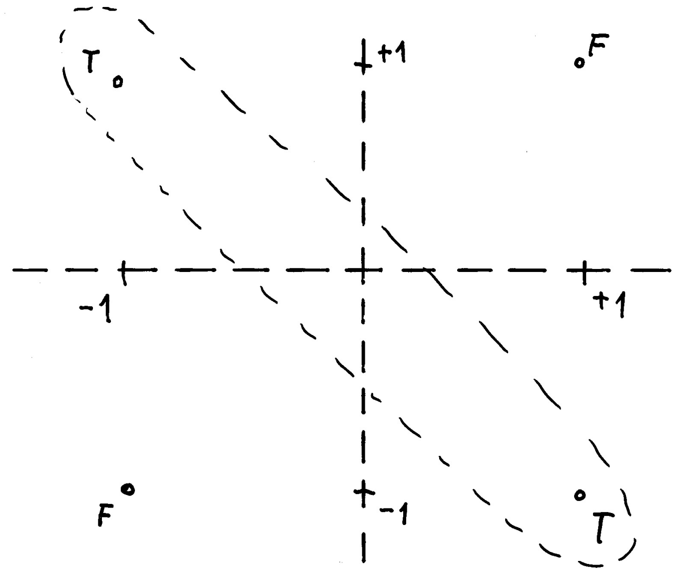
VĚTŠINA

<u>Důležité:</u> Lineární model s 1 jednotkou neumí počítat <u>všechny</u> boolské funkce. Umí počítat pouze tzv. <u>lineárně separabilní</u> funkce.





lineárně separovatelné

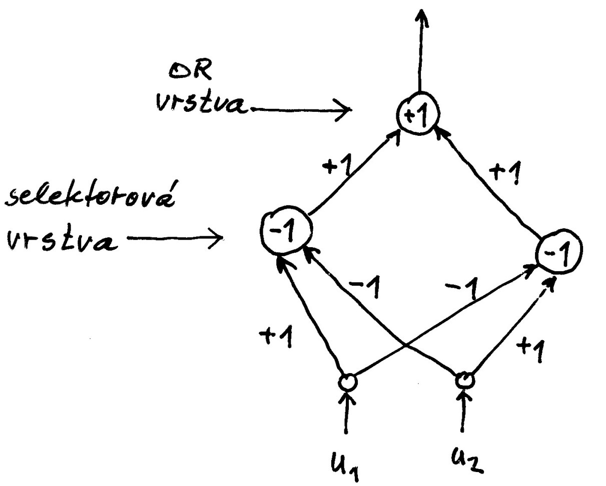


XOR → nelze zkonstruovat přímku oddělující F a T
(XOR není lineárně separovatelná fce)

XOR je z hlediska počtu vstupů nejjednodušší lineárně neseparovatelná funkce.
Tzv. paritní funkce (zobecněná XOR) je T pokud počet T-vstupů je lichý, jinak je F.

Representace libovolné boolské funkce

Vícevrstevný perceptron umí representovat libovolnou boolskou funkci f:



Perceptronová síť pro výpočet hodnoty XOR pomocí selektorů.
Výstup je T pro vstupy $\{+1, -1\}$ a $\{-1, +1\}$, jinak je F.

Věta: Jakákoliv boolská funkce s konečným počtem vstupů je representovatelná pomocí vícevrstvého lineárního perceptronu.

Důkaz: Ve struktuře OR-selektory způsobí každý vstup, že právě 1 nebo žádná selektorová jednotka bude mít na výstupu +1, v závislosti na tom, zda požadovaný výstup je T nebo F. To zaručuje, že výstup jednotky OR bude odpovídat požadovanému výstupu.

Důsledek: Je-li dán soubor nekontradiktorických trénovacích příkladů mající boolské atributy a klasifikace, pak vždy existuje vícevrstevný lineární perceptron produkující korektní výstup pro všechny trénovací příklady.

Praktická limitace: existuje mnoho funkcí s výstupem T přibližně pro 1/2 vstupů (možných vstupních vzorů), takže selektorová konstrukce by vyžadovala cca 2^{p-1} jednotek pro representaci těchto funkcí (p je počet vstupů). Např. pro pouhých 10 vstupů to je $2^{10-1} = 2^9 = 512$. V praxi lze často očekávat $p \sim 10^n$, $n \geq 2$. Dále, např. pro $p = 100$ a počet trénovacích vzorků = 1000 sice sestavíme vhodnou síť s $\leq 10^3$ selektory, ale tato síť poskytne na výstupu T pouze když vstup bude duplikovat jeden z T-trénovacích příkladů → malá robustnost.

TRÉNOVACÍ PRAVIDLO PRO PERCEPTRON

Problém učení lze definovat jako problém stanovení vektoru vah tak, aby perceptron dával na výstupu korektně hodnoty ±1 pro každou z daných trénovacích instancí.
Je známo několik algoritmů, zde uvedeme dva: perceptronové pravidlo a delta pravidlo.

Jednou z možností, jak získat přijatelný vektor vah $\vec{w}$, je začít s váhami s náhodně přiřazenými hodnotami a iterativně použít perceptron na každý trénovací příklad - bude-li klasifikován chybně, váhy je nutno modifikovat.
Tento proces se opakuje tak dlouho, dokud nejsou všechny trénovací příklady klasifikovány správně. Modifikace vah probíhá podle perceptronového trénovacího pravidla:

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta (t - \sigma) x_i$$

$$\eta > 0$$

t je očekávaná hodnota
σ je skutečná hodnota na výstupu
η je tzv. učící konstanta

Koeficient η moderuje stupeň velikosti změn vah v každém kroku. Obvykle se η nastaví na nějakou malou hodnotu, např. $\eta = 0.1$. Se vzrůstajícím počtem iteračních kroků se většinou nechává vliv η postupně vymizet.

Proč uvedené perceptronové pravidlo konverguje směrem k vytvoření váhového vektoru?

Předpokl., že ve speciálním případě je daná instance klasifikována korektně: $(t - \sigma) = 0 \Rightarrow \Delta w_i = 0$, tzn. váhy nejsou modifikovány.
Objeví-li se místo +1 na výstupu −1, je nutno váhy změnit (pro $x_i > 0$ se zvýší w_i a tím se perceptron přiblíží korektní klasifikaci daného příkladu).
w_i se zvýší, neboť $(t - \sigma) > 0$, $\eta > 0$, $x_i > 0$, např. $x_i = 0.8$, $\eta = 0.1$, $t = 1$, $\sigma = -1$:

$$\Delta w_i = \eta (t - \sigma) x_i = 0.1 (1 - (-1)) \cdot 0.8 = 0.16$$

Na druhé straně bude-li $t = -1$ a $\sigma = +1$, pak váhy příslušející pozitivním x_i budou sníženy (nikoliv zvýšeny).

V r. 1969 dokázali Minsky a Papert, že za předpokladu lineární separability trénovacích příkladů a dostatečně malého η perceptronové trénovací pravidlo konverguje. Nejsou-li data lineárně separabilní, nelze konvergenci zaručit.

Pravidlo delta a gradientní sestup

Perceptronové pravidlo může selhat v případě nesplnění podmínky lineární separability (oddělitelnosti) dat. Pravidlo delta umožňuje tuto obtíž překonat: zaručí konvergenci k nejlépe se hodící aproximaci cílového konceptu.

Podstata: pro prohledávání prostoru možných vah se použije tzv. gradientní sestup k nalezení vah, které umožní co nejlepší aproximaci.

Pozn.: tzv. BACK-PROPAGATION algoritmus pro trénování umělých neuronových sítí vzniklých propojením perceptronů je založen na pravidlu delta.

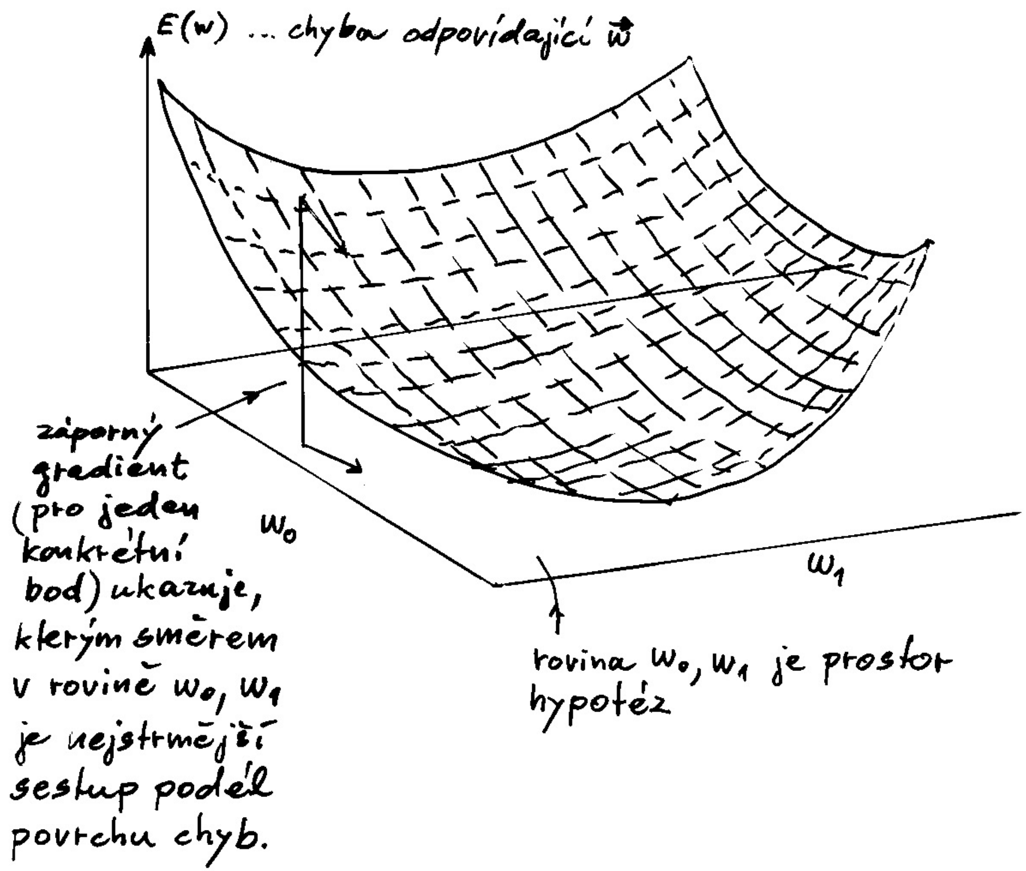
Předpokládejme lineární jednotku (perceptron bez prahu), jejíž výstup je dán vztahem

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

Je nutno stanovit nějakou míru trénovací chyby vzhledem k trénovacím příkladům. Nejčastěji se používá výhodný vztah

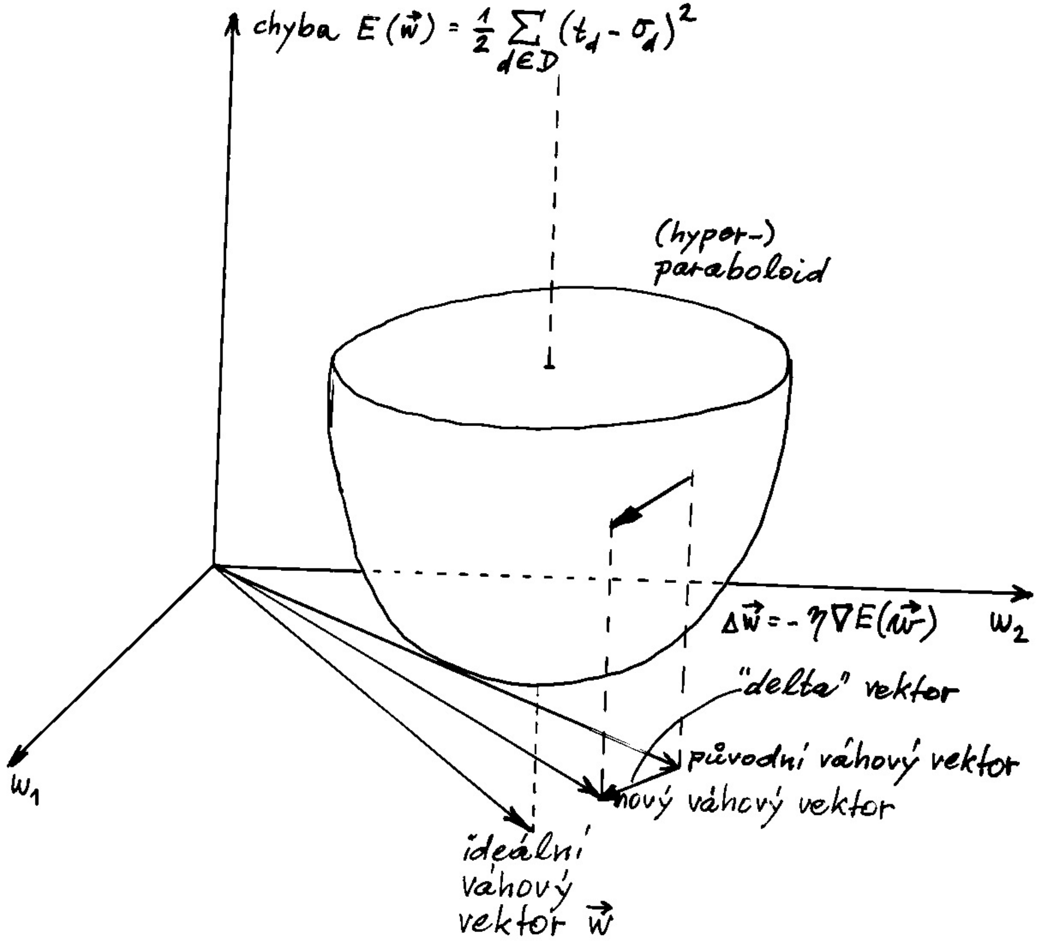
$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \left(t_d - o_d\right)^2$$

kde D je množina trénovacích příkladů, t_d je správný výstup pro trénovací příklad d, o_d je skutečný výstup získaný pro instanci d.



Cílem je najít hypotézu s minimální chybou. Pro lineární jednotky je (vzhledem k definici E) chybový povrch parabolický s jediným globálním minimem. (Parabola samozřejmě závisí na trénovací množině!)

Gradientní sestup pomáhá určit váhový vektor $\vec{w}$, jenž minimalizuje E: začne se s libovolným počátečním vektorem vah a tento je opakovaně modifikován v malých krocích. V každém kroku je $\vec{w}$ měněn ve směru, který dává nejstrmější sestup podél chybového povrchu. Konec: dosažení glob. minima.



Pravidlo "delta" posunuje váhový vektor tak, aby jeho projekce na (hyper-)paraboloid minimální chyby se pohybovala směrem negativního gradientu.

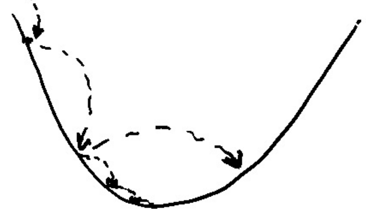
$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$	$\Delta\vec{w} = -\eta\nabla E$
$w_i \leftarrow w_i + \Delta w_i$	$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$

Postup při trénování perceptronu:

- vytvoř náhodně inicializovaný (malými hodnotami) vektor vah;
- aplikuj lineární jednotku na všechny trénovací instance;
- vypočítej Δw_i pro každou váhu w_i;
- každou váhu w_i aktualizuj přičtením Δw_i;
- celý proces opakuj.

Vzhledem k existenci jediného globálního minima algoritmus konverguje k vektoru vah pro minimální chybu bez ohledu na to zda trénovací příklady jsou lineárně separovatelné nebo ne (předpokládá se malé η, např. 0.5, 0.1 apod.)

η: je-li příliš velké, pak hrozí riziko přeskočení minima (místo jeho dosažení):



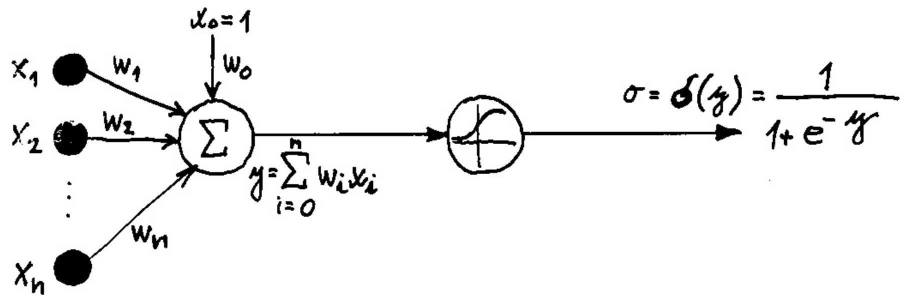
Proto se obvykle volí postupné snižování hodnoty η se vzrůstem počtu kroků.

Vícevrstvé sítě a trénovací algoritmus BP

Bylo ukázáno, že jednoduchý perceptron umožňuje pouze lineární rozhodovací hyperroviny.
Pro nelineární rozhodovací plochy je zapotřebí vytvořit z perceptronů tzv. vrstvené sítě.

Jaké jednotky jsou pro konstrukci sítě vhodné?
Lineární jednotky, pro něž bylo odvozeno Δ pravidlo, mohou vytvářet v kaskádách opět pouze lineární funkce, přičemž zapotřebí je, aby síť uměla reprezentovat rozeznávat nelineární závislosti.
Perceptron s nespojitým prahem neumožňuje výpočet derivace nutný pro stanovení gradientu.
Je proto zapotřebí vytvořit jednotku, jejíž ~~vstu~~ výstup je nelineární funkcí vstupů a zároveň jejíž výstup je diferencovatelnou funkcí vstupů.

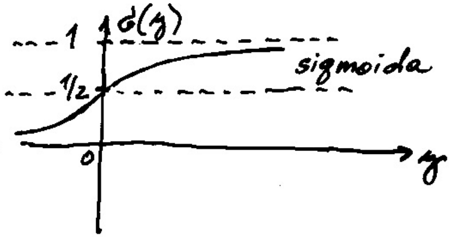
Používaným nejběžnějším řešením je tzv. sigmoidální jednotka – velmi podobná perceptronu, avšak založená na hladké diferencovatelné prahové funkci:



Zobrazená „sigmoidální jednotka" podobně jako perceptron napřed spočítá lineární kombinaci svých vstupů a pak na výsledek aplikuje práh (prahovou funkci).
Sigmoidální jednotka používá pro stanovení výstupu spojitou funkci zvanou sigmoida:

$$\sigma = \sigma(\vec{w} \cdot \vec{x})$$

$$\sigma(y) = \frac{1}{1+e^{-y}}$$



(tato funkce se někdy také nazývá „logistická funkce").
Výstup je mezi 0 a 1, roste monotónně, mapuje velmi široký rozsah vstupní domény do úzkého výstupního rozsahu („squashing function").
Derivace je snadno vyjádřitelná v termínech vstupu:

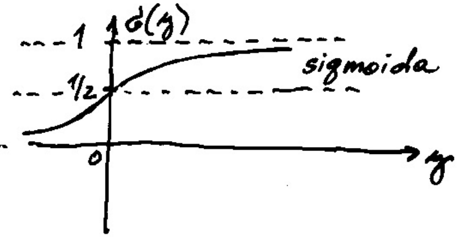
$$\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))$$

Tato derivace je tedy využitelná pro výpočet gradientu.
Pozn.: někdy se používají jiné funkce, např. člen e^{-y} bývá nahrazován e^{-ky} ($k>0$ ovlivňuje strmost prahu). Alternativou bývá místo $\sigma(y)$ také $\tanh(y)$.

Zobrazená „sigmoidální jednotka" podobně jako perceptron napřed spočítá lineární kombinaci svých vstupů a pak na výsledek aplikuje práh (prahovou funkci).
Sigmoidální jednotka používá pro stanovení výstupu spojitou funkci zvanou sigmoida:

$$o = \sigma(\vec{w}\cdot\vec{x})$$

$$\sigma(y) = \frac{1}{1+e^{-y}}$$



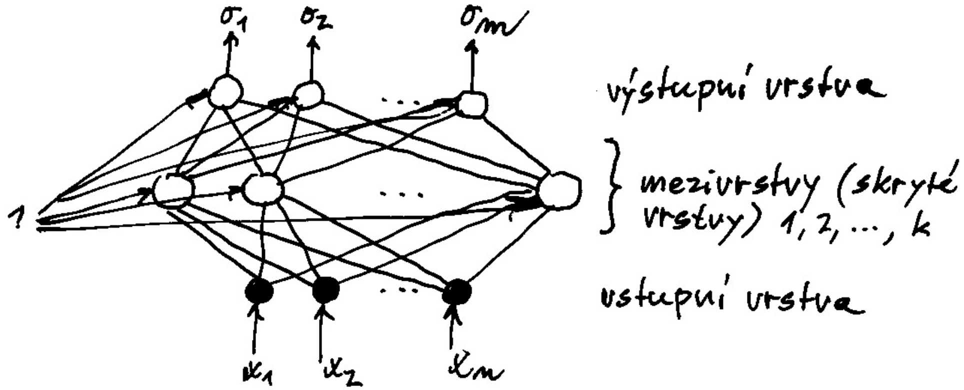
(tato funkce se někdy také nazývá „logistická funkce"). Výstup je mezi 0 a 1, roste monotónně, mapuje velmi široký rozsah vstupní domény do úzkého výstupního rozsahu („squashing function").
Derivace je snadno vyjádřitelná v termínech vstupu:

$$\frac{d\sigma(y)}{dy} = \sigma(y)\cdot(1-\sigma(y))$$

Tato derivace je tedy využitelná pro výpočet gradientu.
Pozn.: někdy se používají jiné funkce, např. člen e^{-y} bývá nahrazován e^{-ky} ($k>0$ ovlivňuje strmost prahu). Alternativou bývá místo $\sigma(y)$ také $\tanh(y)$.

Trénovací algoritmus Back Propagation (BP) (zpětné šíření)

Pomocí BP se síť učí potřebné hodnoty vah za předpokladu, že architektura sítě (počet jednotek a spojů) je neměnná.
Pro minimalizaci kvadrátu chyby (odchylky) mezi skutečným výstupem sítě a požadovaným se používá gradientní sestup.



Protože nyní uvažujeme síť s vícenásobným výstupem, definujeme chybu E jako součet chyb přes všechny výstupy jednotek:

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in \text{výstupy}}(t_{kd}-o_{kd})^2$$

kde výstupy je množina výstupních jednotek sítě, t_{kd} a o_{kd} jsou hodnoty požadované a dosažené pro $\underline{k}$-tou výst. jednotku a trénovací příklad $\underline{d}$.

Některé významnější aplikace

Algoritmus BP je nejpopulárnější učící metodou pro vícevrstvé neuronové sítě. BP a jeho variace byly použity pro řešení široké škály problémů:

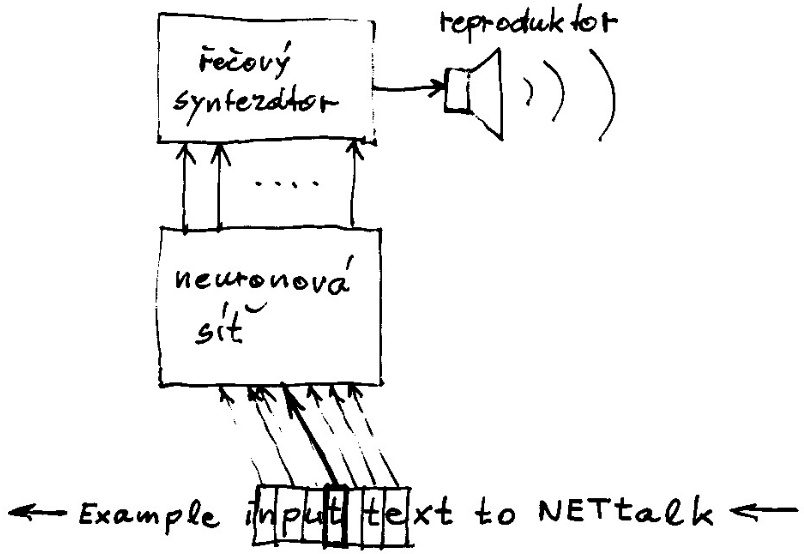
- rozpoznávání vzorů
- zpracování signálů
- komprese obrazů
- modelování nelineárních systémů
- rozpoznávání řeči
- lékařská diagnostika
- předpovědi
- řízení procesů

Nejpřitažlivější vlastností je schopnost adaptace, umožňující modelování složitých procesů pomocí učení se na základě vzorků měření či příkladů. Není nutná znalost specifických matematických modelů, ani expertní znalost.

NETtalk

Jednou z prvních aplikací bylo natrénování sítě pro konverzi anglického textu na řeč (r. 1987). Systém, známý jako NETtalk, se skládal ze dvou modulů:

- mapovací síť
- komerční řečový syntezátor



Mapovací síť měla 80 jednotek ve skryté vrstvě a 26 jednotek ve výstupní vrstvě. Výstup tvořil kód 1-z-26 pro kódování fonémů. Výstup sítě byl zároveň vstupem řečového syntetizéru, který generoval zvuky asociované se vstupními fonémy.

Vstupem do sítě byl 203-rozměrný binární vektor, který kódoval „okno" složené ze 7 následných písmen (29 bitů pro každý ze 7 znaků včetně punktuace; každý znak je kódován za použití kódu 1-ze-29 binárně).

Požadovaným výstupem byla hláska, resp. její kód poskytující výslovnost písmena nacházejícího se uprostřed okna.

Při trénování s použitím 1024 slov ze souboru příkladů anglických hlásek byl NETtalk schopen po 10 trénovacích cyklech poskytovat srozumitelnou řeč. Po 50 cyklech činila přesnost 95%. Síť byla schopna rozeznávat hranice mezi slovy a jak se postupně učila dále, silně připomínala dítě učící se mluvit. Síť uměla rozlišit samohlásky a souhlásky a při testování na novém odlišném textu dosahovala přesnosti 78%. Přidáním vhodného šumu k hodnotám vah či odstraněním několika neuronů klesala výkonnost sítě kontinuálně, nikoliv - jak je obvyklé u sériových digitálních systémů - náhle.

Obdobné komerční zařízení (DEC-talk) založené na bázi pravidel a využívající expertní systém s „ručně" zakódovanými lingvistickými pravidly je lepší. Význam NETtalku ovšem spočívá ve velmi krátké době vývoje (NETtalk se jednoduše učil z omezeného souboru příkladů, zatímco DEC-talk využíval pravidel, která vznikla jako výsledek mnohaleté analýzy mnoha jazykovědců.

Uvedená aplikace ilustruje snadnost (relativní vůči expertním systémům), jak lze pomocí umělé neuronové sítě vyvinout systém dokonce i tehdy, nerozumíme-li příliš či úplně řešenému problému.

Glove-Talk

Systém je založen na neuronové síti jako adaptivním rozhraní pro mapování gesta → řeč. Využívá se 5 dopředných sítí. Gesta jsou popsána 16 parametry (x, y, z, natočení, směr vzhledem k pevné referenci + 10 úhlů prstů). Parametry jsou měřeny každou 1/60 vteřiny.

Systém je trénován tak, že napřed je vytvořeno slovo pomocí gesta, potom pohybem vpřed či vzad v jednom ze 6 směrů se určí zakončení slova (nahoru -s [plurál], k osobě -ed, od osoby -ing, ~~k osobě~~ doprava -er, doleva -ly, dolů nic).

~~Každá~~ Např. „Hand Shape" síť má 80 skrytých jednotek plně propojených na 66 výstupních neuronů (kódování 1-z-66). Tím se vytváří 66 „kořenových" slov.

Glove-Talk je schopen konvertovat 203 gest na slova s 99% přesností.

ZIP-Code Recognition

Rozeznávání PSČ. Viz obrázek.
Trénováno na 7291 příkladech, testováno na 2007 příkladech.
Síť má celkem 1000 jednotek, 64,660 spojů.
Přesnost $\approx$ 99%.

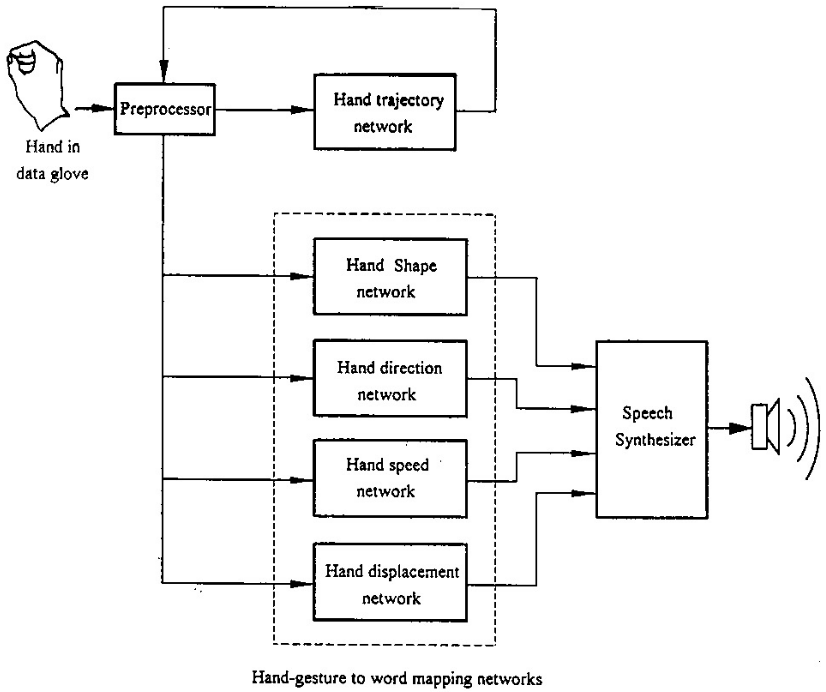


Figure 5.3.2
Glove-Talk: A neural network–based system that maps hand gestures to speech. (From S. S. Fels and G. E. Hinton, Glove-Talk: A neural network interface between a data-glove and a speech synthesizer, *IEEE Transactions on Neural Networks*, 4(1):2–8, 1993; © 1993 IEEE.)

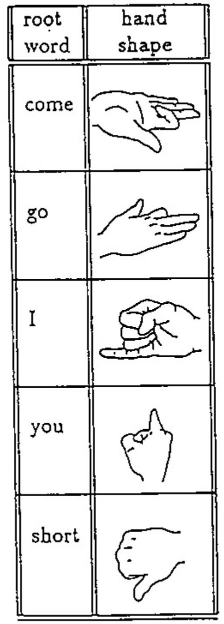


Figure 5.3.3
Examples of root words for several hand gestures (Adopted from S. S. Fels and G. E. Hinton, Glove-Talk: A neural network interface between a data-glove and a speech synthesizer, *IEEE Transactions on Neural Networks*, 4(1):2–8, 1993; © 1993 IEEE.)

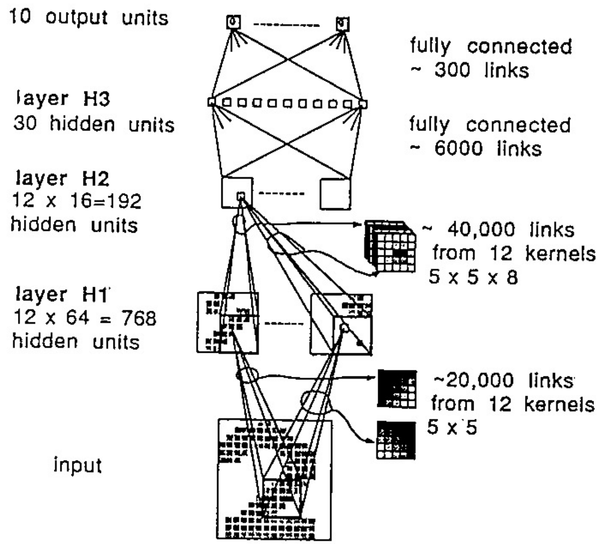


Figure 5.3.5
Network architecture for the handwritten ZIP code recognition neural network (From Y. Le Cun et al., 1989, with permission of the MIT Press.)

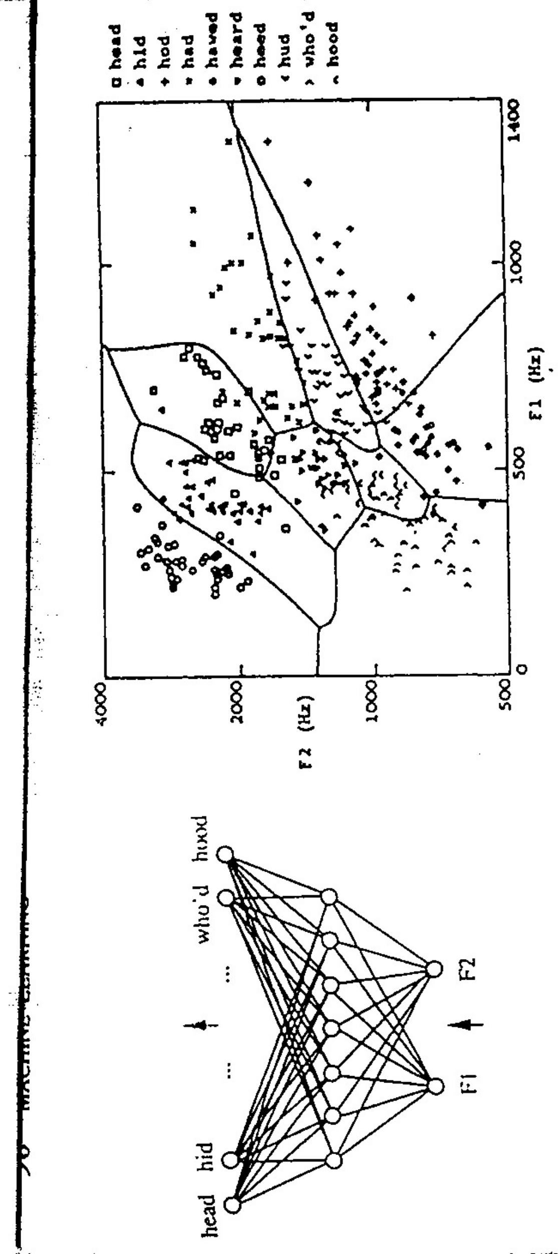


FIGURE 4.5
Decision regions of a multilayer feedforward network. The network shown here was trained to recognize 1 of 10 vowel sounds occurring in the context "h_d" (e.g., "had," "hid"). The network input consists of two parameters, F1 and F2, obtained from a spectral analysis of the sound. The 10 network outputs correspond to the 10 possible vowel sounds. The network prediction is the output whose value is highest. The plot on the right illustrates the highly nonlinear decision surface represented by the learned network. Points shown on the plot are test examples distinct from the examples used to train the network. (Reprinted by permission from Haung and Lippmann (1988).)