Konekcionizmus

Prúd v rámci Umelej inteligencie a Kognitívnych vied, ktorý pracuje s konceptom mnohých identických výpočtových jednotiek ktoré sú spojené v nejakej sieti vzťahov. Väčšinou sa predpokladá, že takto možno vysvetliť mentálne javy. Konekcionizmus je zaujímavý z hľadiska biologickej relevancie a to z viacerích dôvodov:

1. Jednotky môžu predstavovať neuróny a spojenia synapsie,
2. Jednotky majú aktivačnú funkciu, ktorá rozhoduje o výstupnej aktivite jednotky (porovnanie s neurónmi a dynamikou ich vybudenia)
3. Spojenia medzi jednotkami sa premieňajú v čase v závislosti od chyby výstupu – učenie
4. Informácie sú uložené distribuovane medzi jednotkami – analógia ku distribuovanej povahe mozgu.
5. Výpočet prebieha (aspoň teoreticky) paralelne

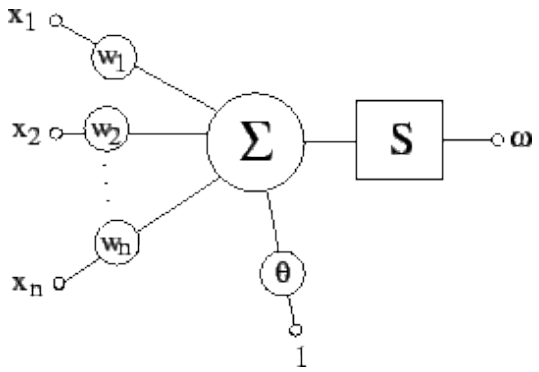
Konekcionistické architektúry sú hlavne zastúpené neuronovými sieťami. História neurónových sietí sa ťahá až do polovice 20tého storočia, kedy ich základy položil Rosenblatt svojou prácou na Perceptrone.

Perceptron

Perceptron je zjednodušený model neurónu. Tak ako neurón má vstupy od iných neŕonov (synapsie), centrálnu časť ktorá vstupy nejako spracuje (soma) a produkuje výstup (axón). Perceptrón môže byť diskrétny alebo lineárny – rozdiel je v tom, akým spôsobom sa zo vstupných dát produkuje výstup.

Diskrétny perceptron má prahovanie: pokiaľ suma váhovaných vstupov neprekročí nejaký prah, tak na výstup pošle nulu (prípadne -1, (prípadne hocičo, to je vlastne jedno)). Tým pádom je jasné, že diskrétny perceptron pracuje nespojito – buď vypľuje a, alebo b, ale nič medzi tým.

Spojitý perceptrón naproti tomu nemá prahovaciu funkciu (aj keď neskôr spomeniem tzv. Bias) a výstup je spojitou funkciou sumy váhovaných vstupov. Ako aktivačná funkcia sa najčastejšie sa používa sigmoida (link na wolfram), ale sú aj iné aktivačné funkcie (http://en.wikipedia.org/wiki/Activation_function). Dôležité vlastnosti dobrej aktivačnej funkcie sú, že sa dá derivovať, je ohraničená a monotónna.



Štruktúra perceptrónu je na obráyku hore. Perceptrónu sa prezentuje nejaký vstup, ktorý je vektor a ma X[i], I = 1...n zložiek (napr. Vektor (1,0,1,1,0) má päť zložiek). w[i] je váha vstupu x[i]. Každá zložka

vstupu prispeje do sumácie hodnotou w[i] * x[i]. Od sumz sa ešte odráta hodnota bias (povedzme jedna), a celá hodnota sa pošle aktivačnej funkcií f(), ktorá vyráta výstupnu aktivitu perceptrónu. Výsledná aktivita perceptrónu je teda:

$$y = f(\Sigma^{n}_{j=1} w_j x_j - \theta)$$

Toto je všeobecná aktivačná funkcia perceptrónu. Diskrétny perceptrón má aktivačnú funkciu prahovanie, tj. ak suma vážených vstupov prekročí nejaký prah (povedzme 0.5 alebo 0), tak až potom dá perceptron nejaký výstup, inakšie dá nulu (alebo -1). Spojitý má napríklad už spomínanú sigmoidálnu funkciu.

Ako bolo spomenuté, neurónové siete sa učia, tj. pomocou nejakého pravidla učenia si upravujú svoje váhy. Podľa toho akým spôsobom prebieha učenie môžeme neurónové siete rozdeliť na

1. siete s učením s učiteľom
2. siete s reinforcement učením
3. siete s učením bez učiteľa

Perceptrón je zástupcom sietí v ktorých učenie prebieha s učiteľom. Základná myšlienka takéhoto upravovania váh je taká, že perceptrónu prezentujeme nejaký vstup, a on vypľuje výstup. Získaný výstup porovnáme s očakaváným výstupom (tj. takým aký by sme si želali) a vypočítame chybu, ktorú perceptrón spravil (napr. odrátame želaný výstup od získaného). Túto chybu potom použijeme na úpravu váh.

Konkrétne pravidlo učenia v diskrétnom perceptróne je:

$$w(j) := w(j) + \alpha(y - f(x))x(j) \quad (j = 1, \ldots, n)$$

Kde w(j) je j-ta váha, alfa je nejaký koeficient učenia, y je želaný výstup a f(x) je získaný výstup. x(j) je j-ty komponent vstupu.

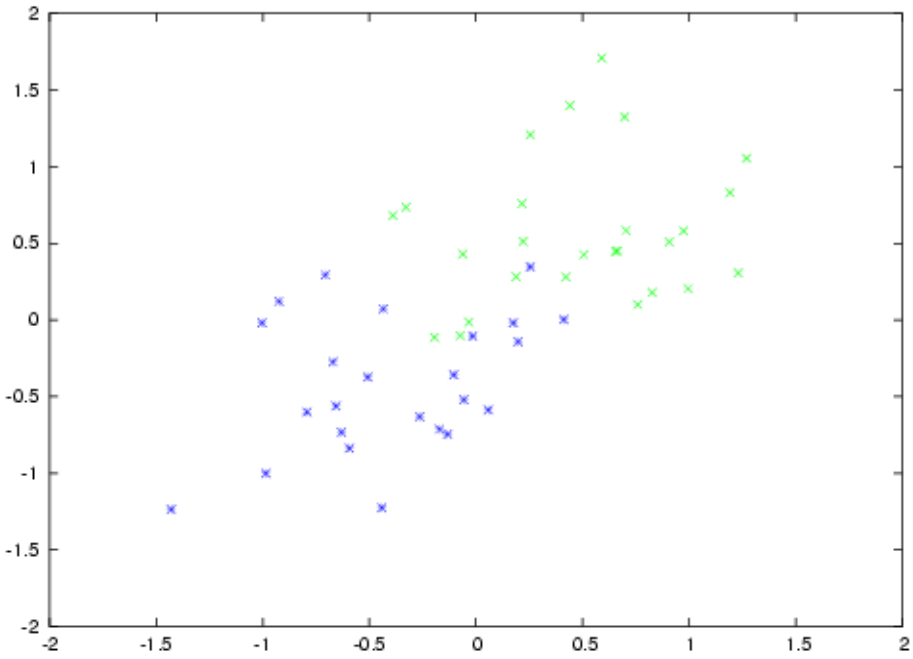
Učenie v spojitom perceptróne je kúsok komplikovanejšie. Všeobecne sa tejto metóde hovorí gradient descent, pretože sa pre jej metaforické vyjadrenie používa predstava chybového priestoru, ktorý vyzerá ako lievik, na ktorého dne je optimálna (nulová) hodnota chyby. Gradient descent po tomto lieviku kráča tak, že sa snaží ísť stále smerom dolu, čo nás raz privedie na dno lievika. Toto je robené cez derivácie (pamätáme si, že nulová hodnota derivácie nejakej funkcie bola práve v bode nejakého maxima/minima). Konkrétny vzorec učenia je:

$$w_j(t+1) = w_j(t) + \alpha \ (d^{(p)} - y^{(p)}) f' x_j$$

Všimnime si že používame deriváciu aktivačnej funkcie. D je želaný výstup, y je yískaný. Tie horné indexy p znamenajú “pre daný pattern” – tj. nejaký konkrétny vstup.

Diskrétne aj spojité perceptróny slúžia ako binárne klasifikátory. Majme nejakú skupinu dvoch rôznych entít, po natrénovaní sa perceptrón naučí rozoznávať medzi nimi. Problém nastáva, keď entity niesú

lineárne separovateľné: tj. nedá sa medzi nimi natiahnúť rovná deliaca čiara. Ako príklad viď ďalší obrázok, na ktorom sú zelené a modré bodky pomiešané medzi sebou:

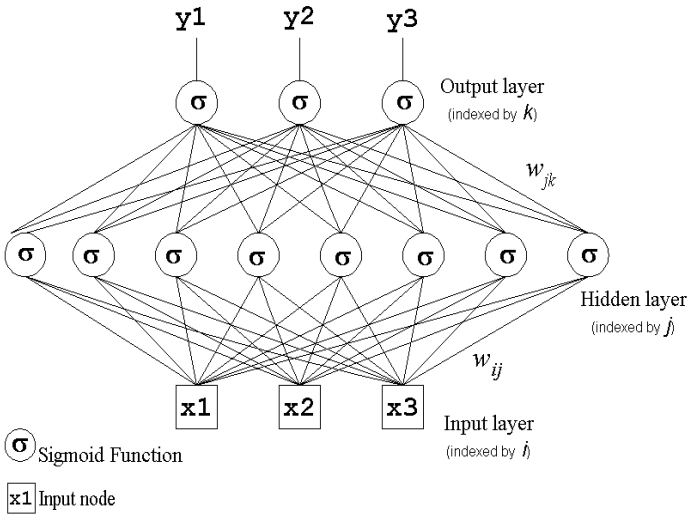


Medzi zelenými a modrými bodkami sa nedá viesť priama čiara v 2D priestore a túto klasifikačnú úlohu by perceptróny vyriešiť nevedeli. Toto koncom 60tých rokov ukázali v knihe *Perceptrons* Minsky a Papert, s tým že ako konkrétny príklad použili fakt, že perceptrón sa nedokáže naučiť funkciu XOR. Ďalej predpokladali, že sa takéto problémy nenaučia ani viacvrstvové perceptróny.

Tento problém v skutočnosti možno riešiť dvoma spôsobmi. Buď rozšírime dimenziu v ktorej pracujeme (v tomto prípade povedzme z 2D na 1000D), praktika, ktorá sa volá kernelová metóda, alebo použijeme, navzdory predpokladom Minského a Paperta viacvrstvové perceptróny.

Viacvrstvové perceptróny

Viacvrstvové perceptróny sa už prestali volať perceptróny, ale hovoríme im neurónové siete. Klasický perceptrón mal vlastne vstupy priamo prepojené na výstup, pričom viacvrstvové prinášajú medzivrstvu/y. Týmto sa hovorí skrytá vrstva (hidden layer).



Viacvrstvové perceptróny vedia robiť všelijaké pekné veci, ako napríklad naučiť sa funkciu XOR, klasifikovať vzory, etc. Ich používanie je zhruba rovnaké ako u jednovrstvých, avšak líšia sa v jednom dôležitom bode. Pravidlo učenia nemožno použiť na váhy medzi vstupnou a skrytou vrtstvou, keďže nevieme aký je želaný výstup zo skrytej vrstvy. Preto sa ako pravidlo učenia používa backpropagation, teda spätné šírenie chybového signálu naprieč sieťou. Viac o backpropagácií tuna http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html
V skratke ide o to, že sa na výstupnej vrstve vypočíta štandardným spôsobom chyba (želaný výsledok mínus získaný), táto chyba sa sieťou šíri naspäť tak že sa prenásobí váhami medzi daným neurónom v nižšej vrstve a neurónom vo vyššej vrstve, keď dorazí na začiatok, použije sa gradientová metóda a postupne sa upravia váhy medzi vrstvami.

Voľba optimálneho modelu, zovšeobecnenie. Regresia.

Voľba optimálneho modelu vlastne znamená, akú architektúru siete a reprezentácie vstupných a výstupných dát si zvolíme. Napríklad pokiaľ je náš klasifikačný problém lineárne separovateľný, je zbytočné pridávať skryté vrstvy. Farkaš ma v slajdoch rozdelenie podľa reprezentácie výstupných dát, plus aktivačnej funkcie.

- Na binárne klasifikačné úlohy použijeme sigmoidu a povedzme prahovanie.
- Na klasifikačné úlohy s vyšším počtom tried použijeme 1-of-M kódovanie, teda výstupná vrstva bude mať M neurónov a pokiaľ budé vstup patriť do n-tej triedy, tak n-tý výstupný neurón dá jednotku, ostatné daju nulu. Inými slovami výstupný vektor bude mať samé nuly a len jednu jednotku na n-tom mieste. Aktivačnú funkciu softmax.
- Pre spojité ciele, ktoré sú ohraničené použijeme sigmoidu alebo tanh
- pre zhora neohraničené ciele exponenciálnu aktivačnú funkciu.

Sieť pokiaľ je navrhnutá správne má vlastnosť, ktorej sa hovorí zovšeobecnenie. Pokiaľ ju natrénujeme na rozlišovanie medzi štvorcami a kruhmi, tak že štvorec bude na výstupe 0 a kruh 1, správne natrénovaná sieť by pravidelnému n-uholníku mala priradiť nejaké čislo medzi nula a 1, pričom by sa výsledok s rastúcim počtom strán asi približoval ku jednotke. Zovšeobecnenie vlastne znamená, že si sieť vztvorí akúsi reprezentáciu vstupnej funkcie, ktorú má aj pre jej časti, na ktorých nebola trénovaná.

S generalizáciou súvisí aj fenomén, ktorý sa volá overfitting. Pokiaľ použijeme moc veľa neurónov a vrstiev, može sa stať, že sieť po čase začne aproximovať oveľa komplikovanejšiu funkciu ako od nej

chceme. Toto sa rieši tým že prebytočné neuróny a vrstvy odstraňujeme – tomuto sa hovorí network pruning.

Úspešnosť natrénovanej siete overíme tak, že jej prezentujeme vstupy a sledujeme ako ich dokáže klasifikovať. Pokiaľ by sme jej ale prezentovali vstupy na ktorých sieť bola trénovaná, nemôžme z určitosťou povedať, že sieť sa niečo naučila. Preto sa celý proces učenia väčšinou robí tak, že vstupnú množinu dát si rozdelíme na dve polovice, na jednej z nich sieť trénujeme, a po natrénovaní siete druhou polkou dát sieť odskúšame. Pokiaľ dostaneme správne výsledky aj pre doteraz neznáme vstupy, je sieť natrénovaná dobre. Bystrý čítateľ pochopil, že vlastne opäť hovorím o zovšeobecňovaní.

Regresia je asi lineárna regresia, ale niesom si istý, napíšem Farkašovi a moc sa mi to nechce, to je hnusná lineárna algebra/blbost.