Řešení domácí úlohy

Příklad 1 - Začátek

- p = [-100:100; -1.*ones(1,100) 0 ones(1,100)];
  t = [100:-1:0 1:100];
- Vstupní i výstupní vzory normalizujte pomocí mapminmax a náhodně je seřaďte (pomocí randperm, p i t stejně!!).
- Vytvořte pro tuto trénovací množinu neuronovou síť s jednou skrytou vrstvou, 5 neurony v této skryté vrtsvě, přenosovou funkcí tansig ve skryté vrstvě a purelin ve výstupní vrstvě. Učící algoritmus nastavte na traingd.
- Prohlédněte si vytvořenou síť. Jak zjistit hodnoty vah a prahů?

Řešení domácí úlohy

Příklad 1 - Pokračování

- Nastavte u vytvořené sítě maximální počet cyklů na 1500. Nastavte maximální počet zhoršení na validační množině na 10.
- Zrušte automatickou transformaci vstupních a výstupních dat.
- Síť naučte se zapnutým grafickým výstupem a prohlédněte si všechny zobrazené grafy. Rozumíte jim?
- Z jakého důvodu se učení zastavilo? Jaká je výsledná chyba na trénovací, validační a testovací množině?
- Zobrazte výstupy sítě pro vstupní vzory do grafu. Naučila se síť dobře?
- Zkuste skript pustit několikrát. Liší se výsledky?

Řešení domácí úlohy

```
x = [-100:100; -1.*ones(1,100) 0 ones(1,100)];
tx = [100:-1:0 1:100];
  % Normalizace dat
[x,pp] = mapminmax(x); [tx,tp] = mapminmax(tx);
  % Náhodně uspořádám vzory
P=randperm(size(x,2)); p = x(:,P); t = tx(:,P);
  % Vytvořím neuronovou síť
net = newff(p,t,[5],{'tansig','purelin'},'traingd');
net.outputs{2}.processFcns = {}; net.inputs{1}.processFcns = {};
net.trainParam.epochs = 1500; net.trainParam.max_fail = 10;
  % Naučím neuronovou síť
[net1,tr] = train(net,p,t);
  % Zobrazím výstup sítě
y = sim(net,x);
hold on; plot(x(1,:),tx,'bx'); plot(x(1,:),y,'ro'); hold off;
```

Řešení domácí úlohy – K-násobná křížová validace

Příklad 1 - Dokončení

- Napište funkci v = CrossVal(), která porovná pro tuto úlohu síťě se 2, 3 a 5 neurony metodou k-násobné křížové validace (pro k=10).
- Funkce vrátí matici, kde každý řádek bude odpovídat jednomu z modelů a ve sloupcích budou hodnoty:
  v(1,:) = průměr (mean) a směrodatná odchylka (std) výsledné chyby na trénovací a testovací množině (mse)
  v(2,:) = totéž pro druhý model
  v(3,:) = totéž pro třetí model
- Doporučuji pro urychlení vypnout při učení grafický výstup (trainParam.showWindow)
- Jak dopadlo srovnání?

Řešení domácí úlohy – K-násobná křížová validace

Příklad výsledku experimentu - rozšířený o čas a počet cyklů a o další možnosti

neur	α	E_{tr}	E_t	čas(s)	počet cyklů
2	0.01	0.182 ± 0.095	0.196 ± 0.090	19.1 ± 0.7	1245 ± 516
3	0.01	0.105 ± 0.106	0.065 ± 0.115	21.9 ± 0.1	1500 ± 0
5	0.01	0.065 ± 0.082	0.074 ± 0.113	21.9 ± 0.4	1482 ± 56
7	0.01	0.025 ± 0.049	0.032 ± 0.065	22.4 ± 0.5	1500 ± 0
9	0.01	0.012 ± 0.065	0.016 ± 0.015	22.6 ± 0.1	1500 ± 0
5	1	2.11 ± 1.44	2.18 ± 1.92	1.5 ± 0.1	0 ± 0
5	0.1	0.028 ± 0.002	0.021 ± 0.060	20.2 ± 0.4	1362 ± 437
5	0.01	0.065 ± 0.082	0.074 ± 0.113	21.9 ± 0.5	1482 ± 56
5	0.001	0.248 ± 0.061	0.258 ± 0.078	20.5 ± 0.3	1405 ± 300

BP sítě - analýza modelu

- Jeden z nejpoužívanějších modelů
- Jednoduchý algoritmus učení
- Poměrně dobré výsledky - aproximační a generalizační schopnosti

Nevýhody

- Interní reprezentace znalostí (černá skříňka).
- Učení s učitelem. Trénovací množina by měla být dostatečně velká a vyvážená.
- Aproximační a generalizační schopnosti: volba vhodné architektury a dalších parametrů (přenosové funkce, parametr učení, strmost sigmoidy) pro danou úlohu.
- Pomalá konvergence, lokální minima, nebezpečí přeučení

BP sítě - analýza modelu

Rychlost učení standardního algoritmu zpětného šíření

- Algoritmus je poměrně pomalý.
- Špatná volba počátečních parametrů ho ještě zpomalí.
- Přesto dosahuje často lepších výsledků než mnohé „rychlé algoritmy“
  (hlavně v případě, že má úloha realistickou úroveň složitosti a velikost trénovací množiny přesáhne kritickou mez)

BP sítě - analýza modelu

Jak urychlit učení a zajistit dobrou aproximaci?

1. Algoritmy, které zachovávají pevnou topologii (architekturu) sítě.
2. Současná adaptace parametrů (vah, prahů) i architektury sítě.
3. Modulární sítě - výrazně zlepšují aproximační schopnosti neuronových sítí.

BP sítě - analýza modelu

Jak urychlit učení a zajistit dobrou aproximaci při zachování pevné topologie?

- Vhodná inicializace vah a prahů
- Vhodná volba parametru učení
- Použití rychlého algoritmu učení (metody druhého řádu,...)

Vhodná inicializace vah a prahů

Pravidlo

- Váhy a prahy by měly být malé, náhodné, rovnoměrně rozdělené, se střední hodnotou 0.

Proč střední hodnota 0?

- Očekávaná hodnota potenciálu každého neuronu bude 0.
- Derivace sigmoidy je maximální pro 0 (~ 0.25) $\rightarrow$ výraznější změny vah na začátku

Proč náhodné?

- Snaha omezit symetrii. Skryté neurony by neměly mít navzájem podobnou funkci

Vhodná inicializace vah a prahů

Problém vah v průběhu učení

- Jsou-li váhy a prahy moc malé, šíří se sítí příliš malá chyba a učení je moc pomalé.
- Moc velké váhy naopak vedou k tzv. saturaci neuronů a pomalému učení (v plochých oblastech chybové funkce)
  - Neurony jsou hodně aktivní nebo hodně pasivní pro všechny trénovací vzory a nelze je dále učit, protože derivace sigmoidy je téměř nulová
  - Paralýza sítě $\rightarrow$ nekontrolovaný růst vah

$\rightarrow$ **proces učení je pak zastaven v suboptimálním lokálním minimu chybové funkce**

- Jak riziko snížit? ... Vhodná inicializace vah a prahů

Vhodná inicializace vah a prahů

Heuristiky

- Volit počáteční váhy z intervalu $[-\frac{2.4}{\sqrt{n}}, \frac{2.4}{\sqrt{n}}]$, n je počet hran, které vedou do daného neuronu

V Matlabu

- *initnw* (Nguyen-Widrow) - snaží se neurony z dané vrstvy rozhodit ve vstupním prostoru zhruba rovnoměrně

Obdobně pro vstupní vzory

- Trénovací vzory by měly být normalizované a v intervalu $[0, 1]$ resp. $[-1, 1]$.

Algoritmus zpětného šíření s momentem

Problém

- V úzkých údolích chybové funkce může vést sledování gradientu k náhlým, velkým a častým oscilacím během učení

Řešení

- Zavedeme člen odpovídající **momentu** (kromě aktuální hodnoty gradientu chybové funkce bereme v úvahu i předchozí změny vah)
- Větší setrvačnost, umožňuje održovat směr, oslabuje oscilace během učení

Algoritmus zpětného šíření s momentem

Nové adaptační pravidlo

- Změna váhy hrany z neuronu i do neuronu j v čase (t+1):

$$\triangle w_{ij}^{t+1} = -\alpha \frac{\partial E}{\partial w_{ij}} + \alpha_m \triangle w_{ij}^{t} = -\alpha \frac{\partial E}{\partial w_{ij}} + \alpha_m (w_{ij}^{t} - w_{ij}^{t-1})$$

- α ... parametr učení
- α_m ... moment učení
- Optimální hodnoty α a α_m a jejich optimální poměr záleží na charakteru dané úlohy

Algoritmus zpětného šíření s momentem

Jak volit parametr učení a moment?

- Parametr učení α
  - moc malý ... pomalé učení (malé změny vah) - nebezpečí uvíznutí v suboptimálním lokálním minimu
  - moc velký ... velké skoky - nebezpečí oscilací, mohu přeskočit lokální minimum chybové funkce

Algoritmus zpětného šíření s momentem

Jak volit parametr učení a moment?

- Moment učení α_m
  - Udržuje směr v úzkých údolích chybové funkce
  - Snižuje nebezpečí ustálení v nestabilních stavech (sedla)
  - Zvyšuje rychlost konvergence (delší úseky beze změny směru přírůstku vah)
  - Moc velký α_m - moc velká setrvačnost, mohu přeběhnout minimum chybové funkce

Algoritmus zpětného šíření s momentem

V plochých oblastech chybové funkce (sedla)

- Zde je gradient chybové funkce téměř nulový a to může vést k oscilacím $\rightarrow \alpha$ by měl být velký (chceme se dostat pryč)

Ve strmých oblastech chybové funkce

- Zde by měl být α naopak malý, α_m pak brání oscilacím.

Řešení:

- adaptivní a lokální parametr učení

Adaptivní parametr učení

Lokální parametr učení pro každou váhu

- Změna váhy z neuronu i do neuronu j v čase (t+1):

$$\triangle w_{ij}^{t+1} = -\alpha_{ij}^{t+1} \frac{\partial E}{\partial w_{ij}} + \alpha_m \triangle w_{ij}^{t}$$

Adaptivní parametr učení

Heuristika

- Parametr učení klesá s rostoucím počtem cyklů.

Počáteční parametr učení $0 << \alpha < 1$

- Umožňuje přeskočit mělká lokální minima
- Rychlý vývoj vah sítě

Závěrečný parametr učení $\alpha \sim 0$

- Zabraňuje oscilacím
- Neměl by klesat moc rychle, stačí:
  $\sum_{t=0}^{\infty} \alpha^t = \infty$,
  $\sum_{t=0}^{\infty} (\alpha^t)^2 < \infty$

Adaptivní parametr učení

Varianty

- Silva & Almeida
- Delta-bar-delta
- Super SAB

Silva & Almeida - heuristika

- Inicializuj α_{ij}^0 malými náhodnými hodnotami
- Zrychluj, pokud se za poslední dvě po sobějdoucí iterace nezměnilo znaménko parciální derivace $\frac{\partial E}{\partial w_{ij}}$
- Zpomaluj, pokud se znaménko změnilo

Algoritmus Silvy & Almeidy

Adaptace parametru učení v čase (t+1)

- $\alpha_{ij}^{t+1} = u \cdot \alpha_{ij}^{t}$, jestliže $\frac{\partial E^t}{\partial w_{ij}} \cdot \frac{\partial E^{t-1}}{\partial w_{ij}} > 0$
- $\alpha_{ij}^{t+1} = d \cdot \alpha_{ij}^{t}$, jestliže $\frac{\partial E^t}{\partial w_{ij}} \cdot \frac{\partial E^{t-1}}{\partial w_{ij}} < 0$
- Konstanty u, d jsou pevně zvolené tak, že $u > 1$, $d < 1$

Problémy

- Parametr učení roste i klesá exponenciálně vzhledem k u a d

$\rightarrow$ Problémy mohou nastat, jestliže po sobě následuje mnoho urychlovacích kroků.

Delta-bar-delta

- Větší důraz na urychlování

Adaptace parametru učení v čase (t+1)

- $\alpha_{ij}^{t+1} = \alpha_{ij}^{t} + u$, jestliže $\frac{\partial E^t}{\partial w_{ij}} \cdot \delta_{ij}^{t-1} > 0$
- $\alpha_{ij}^{t+1} = d \cdot \alpha_{ij}^{t}$, jestliže $\frac{\partial E^t}{\partial w_{ij}} \cdot \delta_{ij}^{t-1} < 0$
- $\alpha_{ij}^{t+1} = \alpha_{ij}^{t}$ jinak

Kde

- $\delta_{ij}^{t} = (1 - \phi) \cdot \frac{\partial E^t}{\partial w_{ij}} + \phi \cdot \delta_{ij}^{t-1}$
- Konstanty u, d, ϕ jsou pevně zvolené

Super SAB

Algoritmus

- Nastav všechny α_{ij}^0 na počáteční hodnotu α_{start}.
- Proveď krok (t) algoritmu zpětného šíření s momentem.
- Pokud $\frac{\partial E^t}{\partial w_{ij}} \cdot \frac{\partial E^{t-1}}{\partial w_{ij}} > 0$, $\alpha_{ij}^{t+1} = u \cdot \alpha_{ij}^t$.
- Pokud $\frac{\partial E^t}{\partial w_{ij}} \cdot \frac{\partial E^{t-1}}{\partial w_{ij}} < 0$:
  - Anuluj předchozí změnu vah: $w_{ij}^{t+1} = w_{ij}^{t+1} - \triangle w_{ij}^t$
  - Zmenši paremetr učení: $\alpha_{ij}^{t+1} = d \cdot \alpha_{ij}^t$

Vlastnosti

- Řádově rychlejší než standardní algoritmus zpětného šíření
- Poměrně stabilní
- Robustní k volbě počátečních parametrů

Jak je to v Matlabu

- *traingdm* ... algoritmus zpětného šíření s momentem
  - net.trainParam.lr ... parametr učení
  - net.trainParam.mc ... moment učení
- *traingda* ... algoritmus zpětného šíření s adaptivním parametrem učení
  - net.trainParam.lr_inc
  - net.trainParam.lr_dec
  - net.trainParam.max_perf_inc
  - pokud se zmenšila chyba, vynásob parametr učení pomocí lr_inc
  - pokud se zvětšila chyba o alespoň max_perf_inc, vynásob parametr učení pomocí lr_dec a anuluj poslední změnu
- *traingdx* ... algoritmus zpětného šíření s adaptivním parametrem učení a momentem

Cvičení

Příklad 1 z minula - pokračování

- Zkuste nahradit funkci 'traingd' za 'traingdx' (resp. 'traingda', 'traingdm') a zvyšte počet neuronů na 5.
- Podívejte se na průběh chyby (Performance) a další grafy (Plots). Vidíte nějaké změny oproti 'traingd'?
- Zkuste skript pustit několikrát a srovnejte výsledky.
- Zkuste různé hodnoty parametrů (net.trainParam.mc, net.trainParam.lr, atd.) a podívejte se, jaký mají vliv na dosažené výsledky

Cvičení

Příklad 1 - pokračování

- Uvažujeme síť s 5 skrytými neurony a stejný příklad (data).
- Napiště skript (nebo i pomocné funkce), který naučí k (=100) sítí každou ze 4 metod (pro vhodně zvolené parametry trainParam) a spočítá:
  - průměr (mean) a směrodatnou odchylku (std) počtu cyklů a času (tr.best_epoch, tr.time)
  - průměr (mean) a směrodatnou odchylku (std) výsledné chyby na trénovací, testovací a validační množině (tr.perf, tr.vperf, tr.tperf pod indexem tr.best_epoch)
- Alternativně lze použít k-násobnou křížovou validaci jako v minulém dom. úkolu (pro k=10) - získáme méně stabilní odhad.
- Jak dopadlo srovnání jednotlivých metod?

Další finty pro zlepšení učení

Výpočet lze pustit opakovaně

- pro různé počáteční váhy, vyberu síť s nejmenší výslednou chybou

Pokud síť nekonverguje nebo konverguje velmi pomalu

- zkusit více neuronů ve vrstvách, popř. více vrstev

Častější předkládání důležitých vzorů

- snížení chyby pro důležité vzory

Další finty pro zlepšení učení

Při ustálení vah a prahů s vysokou chybou

- 'žíhání sítě' = přidání náhodného šumu
- Adaptace váhy hrany z neuronu i do neuronu j podle: $w_{ij}^{t+1} = w_{ij}^{t} + random(-\epsilon, \epsilon)$
- Parametr ϵ je třeba volit opatrně:
  - moc malé ϵ ... vrátí se zpět
  - moc velké ϵ ... musí se znova dlouho adaptovat

Další strategie pro urychlení učení

Algoritmy druhého řádu

- berou v úvahu více informací o tvaru chybové funkce: nejen gradient, ale i zakřivení
- Kvadratická aproximace chybové funkce
  - $\vec{w}$ ... vektor všech vah a prahů sítě (délky n)
  - $E(\vec{w})$ ... chybová funkce
- Taylorův rozvoj:
  $E(\vec{w}+\vec{h}) \approx E(\vec{w}) + \bigtriangledown E(\vec{w})^T \vec{h} + \frac{1}{2}\vec{h}^T \bigtriangledown^2 E(\vec{w})\vec{h}$

Algoritmy druhého řádu

Taylorův rozvoj

$E(\vec{w}+\vec{h}) \approx E(\vec{w}) + \bigtriangledown E(\vec{w})^T \vec{h} + \frac{1}{2}\vec{h}^T \bigtriangledown^2 E(\vec{w})\vec{h}$

- $\vec{w}$ ... vektor všech vah a prahů sítě (délky n)
- $\vec{h}$ ... změna vektoru vah (délky n)
- $E(\vec{w})$ ... chybová funkce
- $\bigtriangledown E(\vec{w})$ je vektor derivací délky n ... $(\frac{\partial E}{\partial w_i})_{i=1}^n$
- $\bigtriangledown^2 E(\vec{w})$ je Hessovská matie $(n \times n)$ parciálních derivací druhého řádu ... $(\frac{\partial^2 E}{\partial w_i \partial w_j})_{i,j=1}^n$

Algoritmy druhého řádu

Gradient chybové funkce

- Taylorův rozvoj:
  $E(\vec{w}+\vec{h}) \approx E(\vec{w}) + \bigtriangledown E(\vec{w})^T \vec{h} + \frac{1}{2}\vec{h}^T \bigtriangledown^2 E(\vec{w})\vec{h}$
- Spočteme gradient chybové funkce:
  $\bigtriangledown E(\vec{w}+\vec{h})^T \approx \bigtriangledown E(\vec{w})^T + \vec{h}^T \bigtriangledown^2 E(\vec{w})$
- Gradient by měl být nulový (hledáme minimum E):
  $\vec{h} = -(\bigtriangledown^2 E(\vec{w}))^{-1} \bigtriangledown E(\vec{w})$

Newtonovská metoda

- Adaptace vah (a prahů) iterativně podle:
  $\vec{w}^{t+1} = \vec{w}^t - (\bigtriangledown^2 E(\vec{w}))^{-1} \bigtriangledown E(\vec{w})$
- Rychlá konvergence, ale problémem může být výpočet inverzní Hessovské matice

Algoritmy druhého řádu

Pseudonewtonovské metody

- Pracují se zjednodušenou aproximací Hessovské matice
- Např. mohu brát v úvahu jen prvky na diagonále (ostatní nulové): $(\frac{\partial^2 E}{\partial w_i^2})$
- Adaptace vah (a prahů) iterativně podle: $\vec{w}_i^{t+1} = \vec{w}_i^t - (\frac{\partial^2 E}{\partial w_i^2})^{-1} \frac{\partial E}{\partial w_i}$
- Odpadá výpočet inverzní Hessovské matice
- Nemusí fungovat dobře, pokud se chybová funkce hodně liší od kvadratické

Algoritmy druhého řádu

Pseudonewtonovské metody - zrychlení a zpřesnění výsledku

- **Quickprop**
  - nepočítá přímo ani prvky Hessovské matice na diagonále
  - používá diskrétní aproximaci parciální derivace druhého řádu

    $$\frac{\partial^2 E^t}{\partial w_i^2} \approx \frac{\frac{\partial E_i^t}{\partial w_i} - \frac{\partial E_i^{t-1}}{\partial w_i}}{\triangle w_i^{t-1}}$$
  - Adaptace vah (a prahů) iterativně podle:

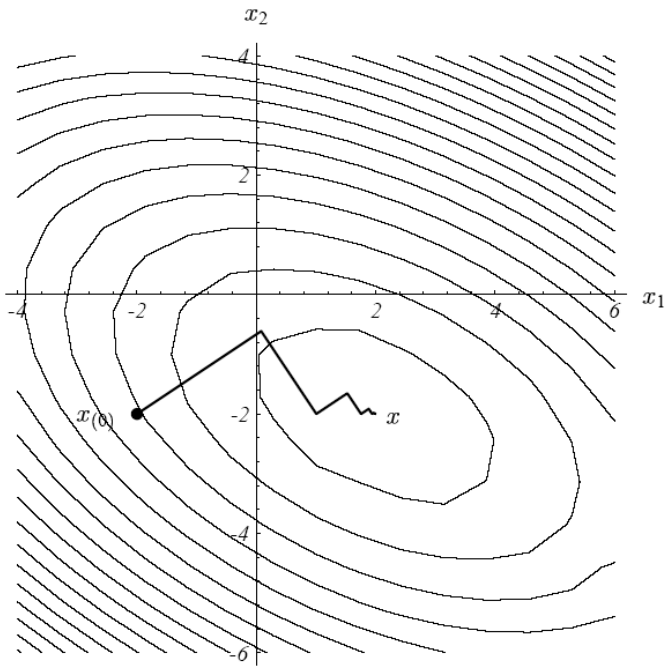
    $$w_i^{t+1} = w_i^t - \frac{\triangle w_i^{t-1}}{\frac{\partial E_i^t}{\partial w_i} - \frac{\partial E_i^{t-1}}{\partial w_i}} \frac{\partial E_i^t}{\partial w_i}$$

Algoritmy druhého řádu

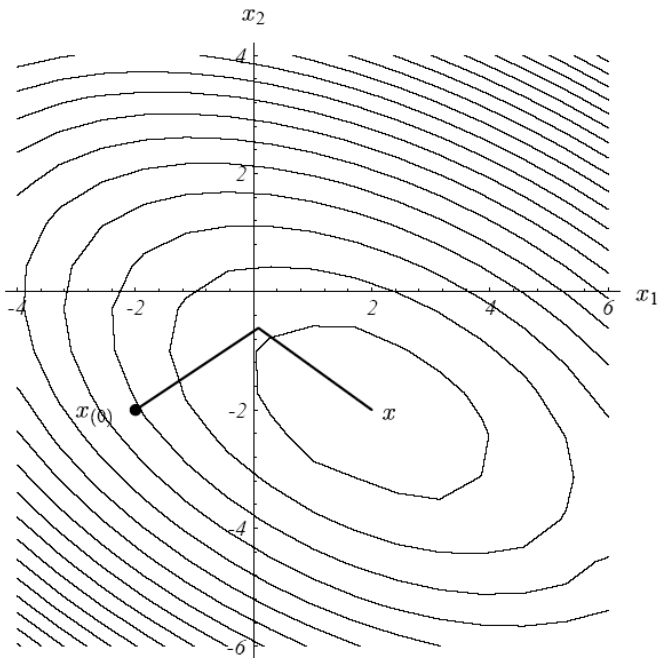
Pseudonewtonovské metody - zrychlení a zpřesnění výsledku

- **Levenberg-Marquardtův algritmus**
  - rychlejší a přesnější v oblasti minima chybové funkce
  - kombinace gradientní a Newtonovy metody
  - pro jeden výstup y: $\frac{\partial^2 E}{\partial w_i \partial w_j} \approx \frac{\partial y}{\partial w_i} \frac{\partial y}{\partial w_j}$

Metody konjugovaných gradientů



(a) BP

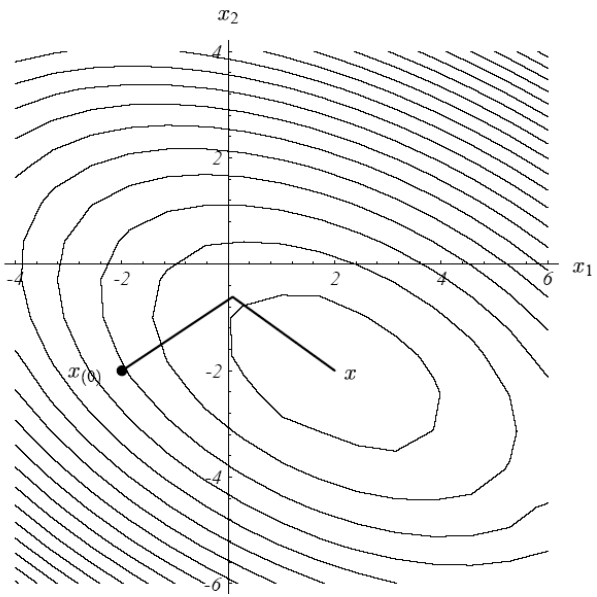


(b) SCG

Rychlá a robustní učící metoda, robustní k volbě počátečních parametrů

Metody konjugovaných gradientů

- Adaptace vah (a prahů) iterativně podle:
  $w_i^{t+1} = w_i^t + \alpha^t \vec{g}^t$
  - α^t ... délka kroku.
  - $\vec{g}^t$ ... směr kroku.
- Vektory $\vec{g}^t$ jsou spočteny tak, any byly navzájem konjugované dáno Hessovská matice $\bigtriangledown^2 E(\vec{w})$:
  $\vec{g}^{tT} \cdot \bigtriangledown^2 E(\vec{w}) \cdot \vec{g}^t = 0$



Relaxační metody

Základní metoda

- V každém kroku se aktualizuje jedna náhodně zvolená váha w_i
- Počítá diskrétní aproximaci gradientu: $\triangle \vec{w_i} = -\alpha \frac{E(\vec{w}+\vec{\beta}) - E(\vec{w})}{\beta_i}$
- $\vec{\beta}$ ... k váze w_i je přičtena malá náhodná perturbace

Rychlejší metoda

- V každém kroku se perturbuje výstup jednoho (i-tého) neuronu o_i o hodnotu $\triangle o_i$
- Požadovaný nový potenciál neuronu i bude: $\sum_j w_{ji}^{t+1} x_i = f^{-1}(o_i + \triangle o_i)$
- Adaptace váhy w_{ki} podle: $w_{ki}^{t+1} = w_{ki}^t \frac{f^{-1}(o_i + \triangle o_i)}{\sum_j w_{ji} x_i}$

Jak je to v Matlabu

Pseudonewtonovské metody

- *trainlm* ... Levenberg-Marquardtův algoritmus
- *trainloss* ... Quickprop
- *trainbfg*, *trainbfgc* ... další pseudonewtonovské metody

Metody konjugovaných gradientů

- *trainscg* ... Moeller ...Metoda škálovaných konjugovaných gradientů
- *traincgf* ... Fletcher-Reeves
- *traincgp* ... Polak-Ribiere
- *traincgb* ... Powell-Beale

Relaxační metoda

- *trainrp* ... Resilent method

Srovnání

User Guide → Backpropagation → Speed and Memory Comparisons

Cvičení

Příklad 1 z minula - pokračování

- Zkuste nahradit funkci 'traingd' za tyto učící funkce.
- Jaké mají tyto metody volitelné parametry? (*net.trainParam*)
- Podívejte se na průběh chyby (Performance) a další grafy (Plots). Vidíte nějaké změny oproti 'traingd'?
- Zkuste skript pustit několikrát a srovnejte výsledky.
- Zkuste různé hodnoty parametrů (net.trainParam.*, atd.) a podívejte se, jaký mají vliv na dosažené výsledky

Cvičení

Příklad 1 - pokračování

- Uvažujeme síť s x skrytými neurony a stejný příklad (data).
- Napiště skript (nebo i pomocné funkce), který naučí k (=100) sítí 4 z těchto metod (pro vhodně zvolené parametry trainParam) a spočítá:
  - průměr (mean) a směrodatnou odchylku (std) počtu cyklů a času (tr.best_epoch, tr.time)
  - průměr (mean) a směrodatnou odchylku (std) výsledné chyby na trénovací, testovací a validační množině (tr.perf, tr.vperf, tr.tperf pod indexem tr.best_epoch)
- Alternativně lze použít k-násobnou křížovou validaci jako v minulém dom. úkolu (pro k=10) - získáme rychlejší, ale méně stabilní odhad.
- Jak dopadlo srovnání jednotlivých metod?