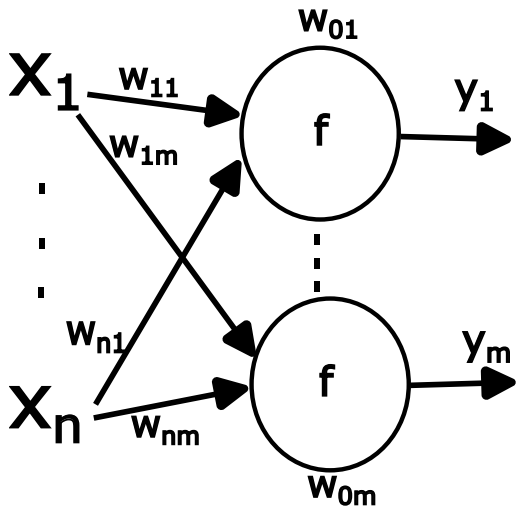
Co jsme probírali minule

1. Neuronová síť s jednou vrstvou neuronů
   - Mnohorozměrná lineární regrese (lineární neuronová síť)
   - Lineární klasifikace do více tříd / rozpoznávání vzorů (jednovrstvý perceptron)



1. Vícevrstvá (vrstevnatá) neuronová síť (MLP)
2. Algoritmus zpětného šíření (backpropagation) - nedokončili jsme

Vrstevnatá neuronová síť (multi-layer neural network, MLP)

Neuronová síť s hierarchickou architekturou:

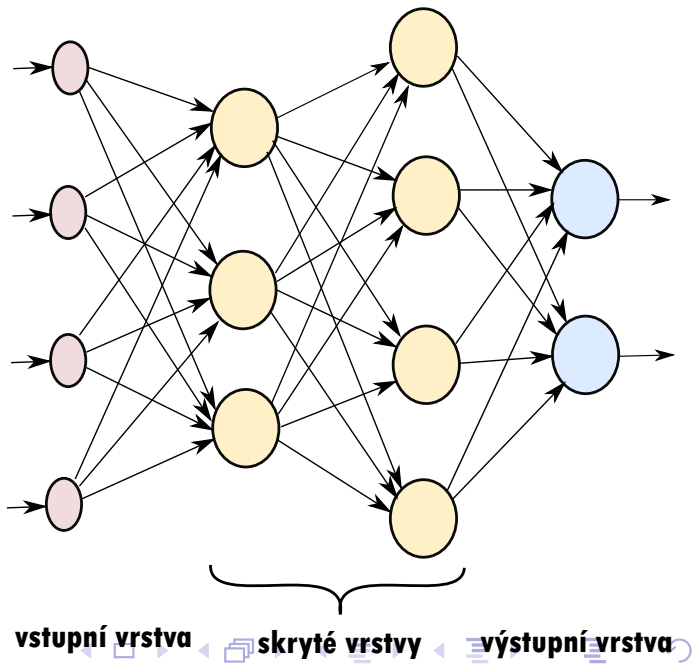
- neurony jsou uspořádány do vrstev
- všechny neurony v jedné vrstvě jsou propojeny právě se všemi neurony z následující vrstvy

Speciální vstupní vrstva:

- odpovídá vstupům neuronové sítě

Výstupní vrstva

- výstup (odezva) neuronové sítě odpovídá výstupům (aktivitám) výstupních neuronů

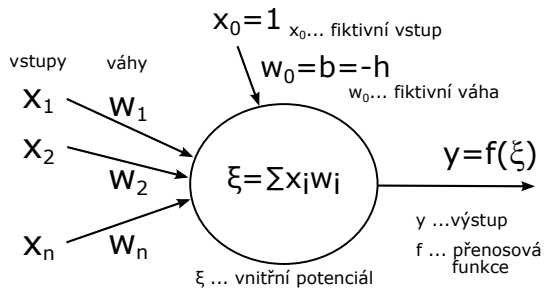


Připomenutí: neuron a zjednodušení výpočtu díky zavedení fiktivního vstupu

Formální neuron

- vnitřní potenciál: $\xi = \sum_{i=1}^{n} w_i.x_i + b = \vec{x}\vec{w} + b$
- výstup: $y = f(\xi)$
- trénovací množina $T = (X, d)$:

x_{11}	...	x_{1n}	d_1
...		...	...
x_{N1}	...	x_{Nn}	d_N



Formální neuron - alternativně:

- rozšířený příznakový prostor ... $\vec{x} = (x_0 = 1, x_1, ..., x_n)$
- rozšířený prostor vah ... $\vec{w} = (w_0 = b, w_1, ..., w_n)^T$
- vnitřní potenciál: $\xi = \sum_{i=0}^{n} w_i.x_i = \vec{x}\vec{w}$
- výstup: $y = f(\xi)$
- trénovací množina $T = (X, d)$:

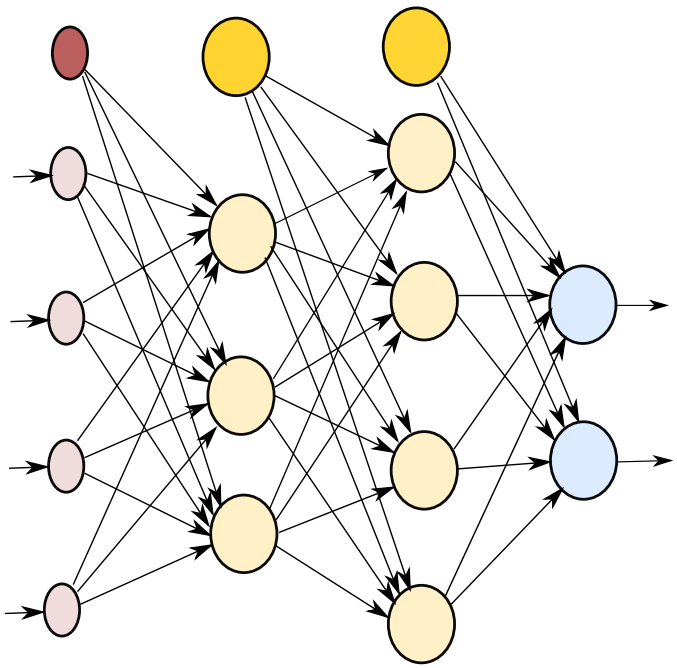
$x_{10} = 1$	x_{11}	...	x_{1n}	d_1
...	...		...	...
$x_{N0} = 1$	x_{N1}	...	x_{Nn}	d_N

Vrstevnatá neuronová síť (multi-layer neural network, MLP)

- provedeme obdobné zjednodušení jako u jednoho neuronu

Fiktivní neurony:

- ke každé skryté vrstvě (podobně jako ke vstupní) přidáme jeden fiktivní neuron s konstantním výstupem 1, který bude reprezentovat prahy (biasy) neuronů v následující vrstvě

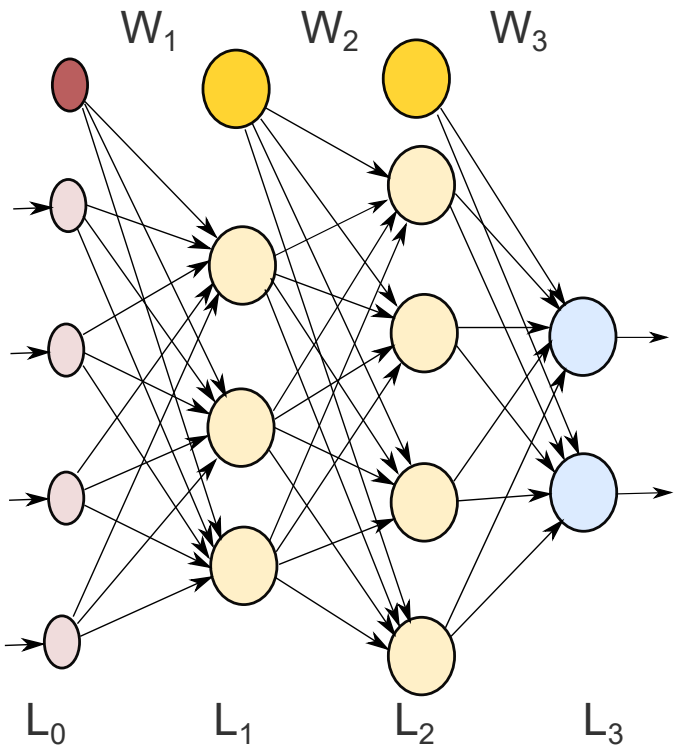


Vrstevnatá neuronová síť (multi-layer neural network, MLP)

Maticová reprezentace vrstevnaté neuronové sítě:

- mějme vrstevnatou neuronovou síť s vrstvami L_0 (vstupní),...,L_{max} (výstupní)
- váhy všech neuronů můžeme reprezentovat pomocí matic $W_1, ..., W_{L_{max}}$
- W_L ... matice vah mezi vrstvou $L-1$ a L o rozměrech $(n_{L-1}+1) \times n_L$

(podobně jako pro jednovrstvou neuronovou síť)



Vrstevnatá neuronová síť (multi-layer neural network, MLP)

Konfigurace neuronové sítě:

- vektor vah všech neuronů v síti $\vec{w} \sim$ vektor všech parametrů modelu

Jak vrstevnatou neuronovou síť budeme učit?

- můžeme použít gradientní metodu pro zvolenou chybovou funkci E a vektor všech parametrů modelu $\vec{w}$:

$$\vec{w}(t+1) = \vec{w}(t) - \alpha \nabla E(\vec{w})$$

- výpočet parciálních derivací a proces adaptace vah budou složitější než u jedné vrstvy neuronů
- výpočet značně zjednoduší algoritmus zpětného šíření chyby (backpropagation)

Algoritmus zpětného šíření (Backpropagation)

(Werbos, Rumelhart, 1974-1986)

Máme k dispozici

- Trénovací množina T s N trénovacími vzory $(\vec{x}_p, \vec{d}_p)$.
  - $x^p = (x_1^p, ..., x_n^p)$ ... vstupní vzor
  - $d^p = (d_1^p, ..., d_m^p)$ ... požadovaný výstup

$x_{10} = 1$	x_{11}	...	x_{1n}	d_{11}	...	d_{1m}
...	...		...	...		...
$x_{N0} = 1$	x_{N1}	...	x_{Nn}	d_{N1}	...	d_{Nm}

- Vrstevnatá neuronová síť s danou architekturou a s $n + 1$ vstupními a m výstupními neurony. Neurony musí mít spojitou, diferencovatelnou přenosovou funkci.

Cíl

- Nastavit váhy všech neuronů v síti tak, aby byl skutečný výstup sítě stejný jako požadovaný.

Algoritmus zpětného šíření (Backpropagation)

Cílová (chybová) funkce

- místo SSE se často používá její zobecnění MSE (mean squared error) ... průměrná chyba přes všechny vzory $E_{MSE} = E_{SSE}/N$
  - pro jeden trénovací vzor:

$$E_p(\vec{w}) = \frac{1}{2}\sum_{j=1}^{m}(d_{pj} - y_{pj})^2$$

  - pro celou trénovací množinu:

$$E(\vec{w}) = \frac{1}{N}\sum_{p=1}^{N} E_p = \frac{1}{2N}\sum_{p=1}^{N}\sum_{j=1}^{m}(d_{pj} - y_{pj})^2$$

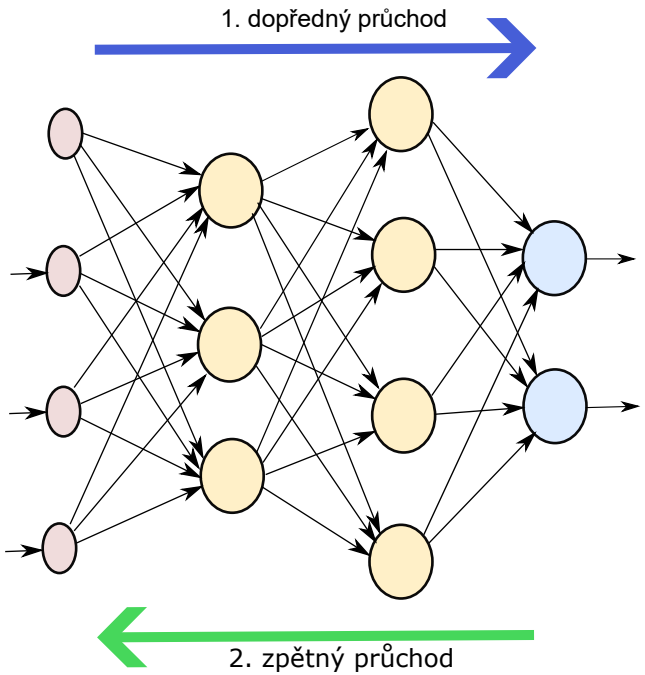
Cíl algoritmu zpětného šíření

- minimalizace chybové funkce E na dané trénovací množině T
- obecně lze ale použít i jinou chybovou funkci (např. cross-entropy)

Algoritmus zpětného šíření (Backpropagation)

Základní princip

1. Spočteme skutečnou odezvu sítě pro daný trénovací vzor.
   - od vstupní vrstvy směrem k výstupní
2. Porovnáme skutečnou a požadovanou odezvu sítě.
3. Adaptujeme váhy a prahy:
   - proti směru gradientu chybové funkce
   - od výstupní vrstvy směrem ke vstupní



Výpočet odezvy neuronové sítě - nejprve ,,po jednom"

3. Předložíme vstupní vzor $\vec{x}_t$
4. Postupujeme ve směru od vstupní vrstvy k výstupní a pro každý neuron j spočteme (a zapamatujeme si) jeho výstup y_{tj} (využijeme k tomu aktivity neuronů v předchozí vrstvě, včetně fiktivního)

$$y_{tj} = f(\xi_{tj}) = f(\sum_i w_{ij} y_{ti})$$

   (i je index přes neurony ve vrstvě předcházející neuronu j)
5. Výstup sítě $\vec{y} = (y_1, ..., y_m)$ tvoří výstupy neuronů ve výstupní vrstvě

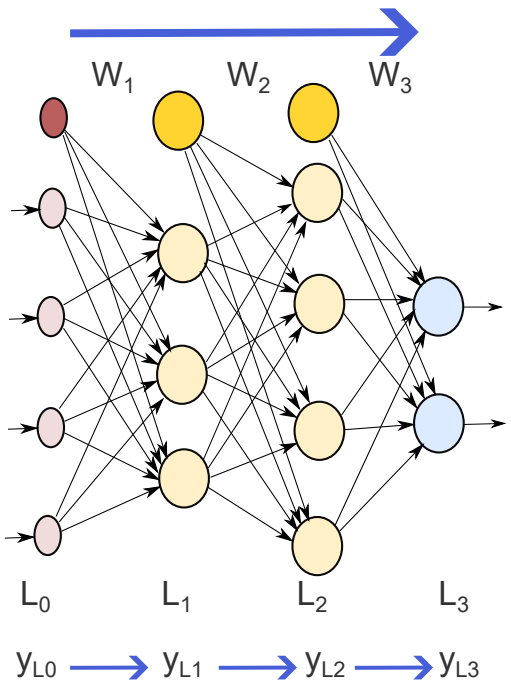
Výpočet odezvy neuronové sítě - maticově I.

1. Předložíme vstupní vzor $\vec{x}$
2. Pro vrstvy $L = L_0, L_1, ..., L_{max}$ postupně spočítáme vektory výstupů těchto vrstev $\vec{y}_{L_0}, ..., \vec{y}_{L_{max}}$:
   - Pro $L = L_0$ (vstupní vrstva): $\vec{y}_{L_0} = \vec{x}$
   - Pro $L = L_1 ..., L_{max}$:

$$\vec{z}_L = f(\vec{\xi}_L) = f(\vec{y}_{L-1} W_L)$$

$$\vec{y}_L = (1|\vec{z}_L)$$

   W_L je rozšířená matice vah mezi vrstvou (L-1) a L
3. Výstup sítě $\vec{y} = (y_1, ..., y_m) = \vec{y}_{L_{max}}$ tvoří výstupy neuronů ve výstupní vrstvě



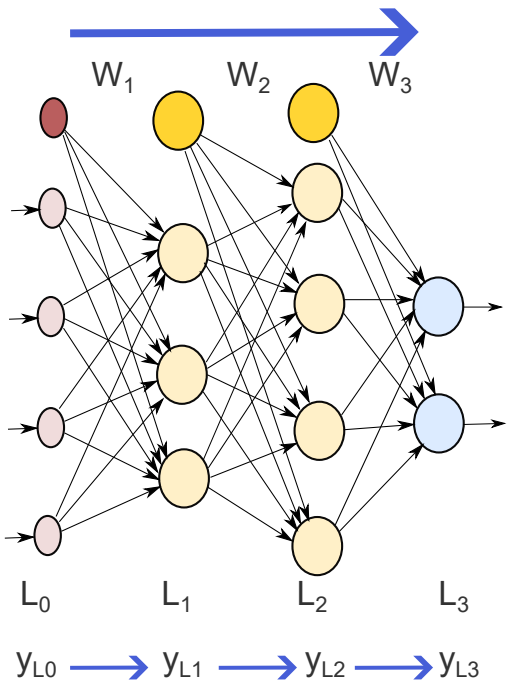
Výpočet odezvy neuronové sítě - maticově II.

1. Předložíme matici vstupních vzorů X
2. Pro vrstvy $L = L_0, L_1, ..., L_{max}$ postupně spočítáme matice výstupů těchto vrstev $Y_{L_0}, ..., Y_{L_{max}}$:
   - Pro $L = L_0$ (vstupní vrstva): $Y_{L_0} = X$
   - Pro $L = L_1, .., L_{max}$:

$$Z_L = f(\xi_L) = f(Y_{L-1} W_L)$$

$$Y_L = (1|Z_L)$$

   W_L je rozšířená matice vah mezi vrstvou (L-1) a L
3. Výstup sítě $Y = Y_{L_{max}}$ tvoří výstupy neuronů ve výstupní vrstvě



Backpropagation - Adaptační pravidla

- Chybová funkce pro jeden trénovací vzor $(\vec{x}_t, \vec{d}_t)$ a skutečný výstup sítě $\vec{y}_t$:

$$E_t = \frac{1}{2}\sum_{j=1}^{m}(d_{tj} - y_{tj})^2$$

- Parciální derivace:

$$\frac{\partial E_t}{\partial w_{ij}} = \frac{\partial E_t}{\partial y_{tj}}\frac{\partial y_{tj}}{\partial \xi_{tj}}\frac{\partial \xi_{tj}}{\partial w_{ij}}$$

- Adaptační pravidlo pro váhu z neuronu i do neuronu j v čase t:

$$w_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t),$$

- $\triangle w_{ij}(t)$ je přírůstek váhy w_{ij} přispívající k minimalizaci E_t:

$$\triangle w_{ij}(t) = -\alpha\frac{\partial E_t}{\partial w_{ij}} = -\alpha\frac{\partial E_t}{\partial y_{tj}}\frac{\partial y_{tj}}{\partial \xi_{tj}}\frac{\partial \xi_{tj}}{\partial w_{ij}}$$

- α je parametr učení (learning rate)

Backpropagation - Adaptační pravidla

Označme

$$\delta_{tj} = \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} = \frac{\partial E_t}{\partial \xi_{tj}} \text{ ... chybový člen pro nejuron } j$$

Pak

$$\triangle w_{ij}(t) = -\alpha \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} \frac{\partial \xi_{tj}}{\partial w_{ij}} = -\alpha \delta_{tj} \frac{\partial \xi_{tj}}{\partial w_{ij}} = -\alpha \delta_{tj} y_{ti}$$

kde:

$$\frac{\partial \xi_{tj}}{\partial w_{ij}} = \frac{\partial (\sum_k w_{kj} y_{tk})}{\partial w_{ij}} = y_{ti}$$

(k je index přes všechny neurony ve vrstvě předcházející j)

Backpropagation - Adaptační pravidla

I. Pro neurony j ve výstupní vrstvě:

$$E_t = \frac{1}{2}\sum_{j=1}^{m}(d_{tj} - y_{tj})^2$$

$$\delta_{tj} = \frac{\partial E_t}{\partial y_{tj}}\frac{\partial y_{tj}}{\partial \xi_{tj}} = -(d_{tj} - y_{tj})f'(\xi_{tj})$$

Pro váhy w_{ij} mezi poslední skrytou vrstvou a výstupní:

$$\triangle w_{ij}(t) = -\alpha\delta_{tj}y_{ti} = \alpha(d_{tj} - y_{tj})f'(\xi_{tj})y_{ti}$$

Back-propagation - Adaptační pravidla

Pro neurony j v poslední skryté vrstvě:

$$\begin{aligned} E_t &= \frac{1}{2}\sum_{k=1}^{m}(d_{tk} - y_{tk})^2 = \frac{1}{2}\sum_{k=1}^{m}(d_{tk} - f(\xi_{tk}))^2 \\ &= \frac{1}{2}\sum_{k=1}^{m}(d_{tk} - f(\sum_{j} w_{jk} y_{tj}))^2 \end{aligned}$$

(k je index přes všechny výstupní neurony, j je index přes všechny neurony v poslední skryté vrstvě)

$$\begin{aligned} \frac{\partial E_t}{\partial y_{tj}} &= \sum_{k=1}^{m} \frac{\partial E_t}{\partial y_{tk}} \frac{\partial y_{tk}}{\partial y_{tj}} = \sum_{k=1}^{m} \frac{\partial E_t}{\partial y_{tk}} \frac{\partial y_{tk}}{\partial \xi_{tk}} \frac{\partial \xi_{tk}}{\partial y_{tj}} \\ &= \sum_{k=1}^{m} \delta_{tk} \frac{\partial \xi_{tk}}{\partial y_{tj}} = \sum_{k=1}^{m} \delta_{tk} w_{jk} \end{aligned}$$

Back-propagation - Adaptační pravidla

Pro neurony j v poslední skryté vrstvě:

$$\frac{\partial E_t}{\partial y_{tj}} = \sum_{k=1}^{m} \delta_{tk} w_{jk}$$

$$\delta_{tj} = \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} = \left(\sum_{k=1}^{m} \delta_{tk} w_{jk} \right) f'(\xi_{tj})$$

Pro váhy w_{ij} mezi předposlední a poslední skrytou vrstvou:

$$\triangle w_{ij}(t) = -\alpha \delta_{tj} y_{ti} = -\alpha \left(\sum_{k=1}^{m} \delta_{tk} w_{jk} \right) f'(\xi_{tj}) y_{ti}$$

(k je index přes všechny výstupní neurony)

Back-propagation - Adaptační pravidla

Pro neurony j v libovolné skryté vrstvě:

$$\frac{\partial E_t}{\partial y_{tj}} = \sum_k \frac{\partial E_t}{\partial y_{tk}} \frac{\partial y_{tk}}{\partial y_{tj}} = \sum_k \frac{\partial E_t}{\partial y_{tk}} \frac{\partial y_{tk}}{\partial \xi_{tk}} \frac{\partial \xi_{tk}}{\partial y_{tj}}$$

$$= \sum_k \delta_{tk} \frac{\partial \xi_{tk}}{\partial y_{tj}} = \sum_k \delta_{tk} w_{jk}$$

(index k jde přes všechny neurony v následující vrstvě)

$$\delta_{tj} = \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} = (\sum_k \delta_{tk} w_{jk}) f'(\xi_{tj})$$

Back-propagation - Adaptační pravidla

Celkem:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha\delta_{tj}y_{ti},$$

kde

- pro výstupní neuron j:

$$\delta_{tj} = f'(\xi_{tj})(y_{tj} - d_{tj}).$$

- pro skrytý neuron j:

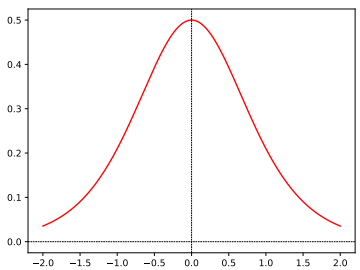
$$\delta_{tj} = f'(\xi_{tj})\sum_{k}(\delta_{tk}w_{jk}).$$

Připomenutí: Výpočet derivace přenosové funkce

Sigmoidální ... logsig

$$y = f(\xi) = \frac{1}{1 + e^{-\lambda\xi}}$$

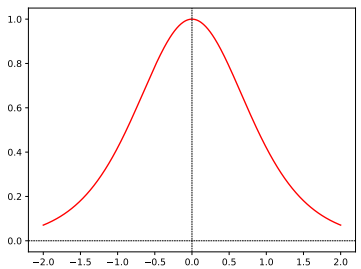
$$f'(\xi) = \lambda y(1 - y)$$



Hyperbolický tangens ... tanh

$$y = f(\xi) = \frac{1 - e^{-2\lambda\xi}}{1 + e^{-2\lambda\xi}}$$

$$f'(\xi) = \lambda^2(1 + y)(1 - y)$$



Algoritmus zpětného šíření (online varianta)

1. **Inicializace sítě**
   Inicializujeme váhy malými náhodnými hodnotami
2. **Předložíme další trénovací vzor** ve tvaru $(\vec{x}_t, \vec{d}_t)$
3. **Dopředný výpočet:**
   Postupujeme ve směru od vstupní vrstvy k výstupní a pro každý neuron j spočteme (a zapamatujeme si) jeho výstup y_{tj} (využijeme k tomu aktivity neuronů v předchozí vrstvě, včetně fiktivního)

$$y_{tj} = f(\xi_{tj}) = f(\sum_i w_{ij} y_{ti})$$

(i je index přes neurony ve vrstvě předcházející neuronu j)

Algoritmus zpětného šíření (online varianta)

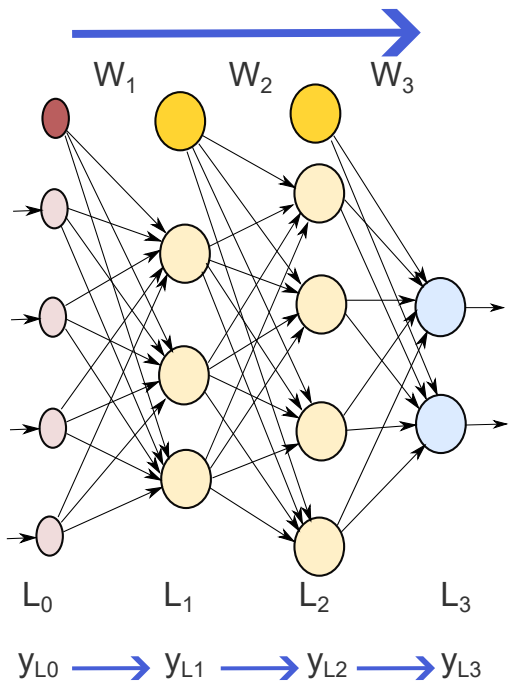
3. **Dopředný výpočet maticově:**
   Pro vrstvy $L = L_0, L_1, ..., L_{max}$ postupně spočítáme vektory výstupů těchto vrstev $\vec{y}_{L_0}, ..., \vec{y}_{L_{max}}$:

- Pro $L = L_0$ (vstupní vrstva): $\vec{y}_{L_0} = \vec{x}$
- Pro $L = L_1 ..., L_{max}$:

$$\vec{z}_L = f(\vec{\xi}_L) = f(\vec{y}_{L-1} W_L)$$

$$\vec{y}_L = (1|\vec{z}_L)$$

W_L je rozšířená matice vah mezi vrstvou (L-1) a L



- Výstup sítě $\vec{y} = (y_1, ..., y_m) = \vec{y}_{L_{max}}$ tvoří výstupy neuronů ve výstupní vrstvě

Algoritmus zpětného šíření (online varianta)

4. **Zpětný výpočet**
   Postupujeme ve směru od výstupní vrstvy k první skryté a pro každý neuron j spočteme (a zapamatujeme si) chybový člen δ_{tj} a aktualizujeme všechny váhy w_{ij} vedoucí do vrcholu j.
   - Pro výstupní neurony j spočteme:
   $$\delta_{tj} = f'(\xi_{tj})(y_{tj} - d_{tj})$$
   - Pro skryté neurony j spočteme:
   $$\delta_{tj} = f'(\xi_{tj}) \sum_k (\delta_{tk} w_{jk}).$$
   (k je index přes neurony ve vrstvě následující po vrstvě obsahující neuron j)
   - Pro váhu každé hrany z nějakého neuronu i (včetně fiktivního) do neuronu j:
   $$w_{ij}(t+1) = w_{ij}(t) - \alpha \delta_{tj} y_{ti}$$

Algoritmus zpětného šíření (online varianta)

4. **Zpětný výpočet maticově**
   Pro vrstvy $L = L_{max}, ..., L_1$ postupně spočítáme chybové členy $\vec{\delta_L}$ a aktualizujeme matice vah W_L mezi vrstvami (L-1) a L:

- Pro výstupní vrstvu L_{max} spočteme:

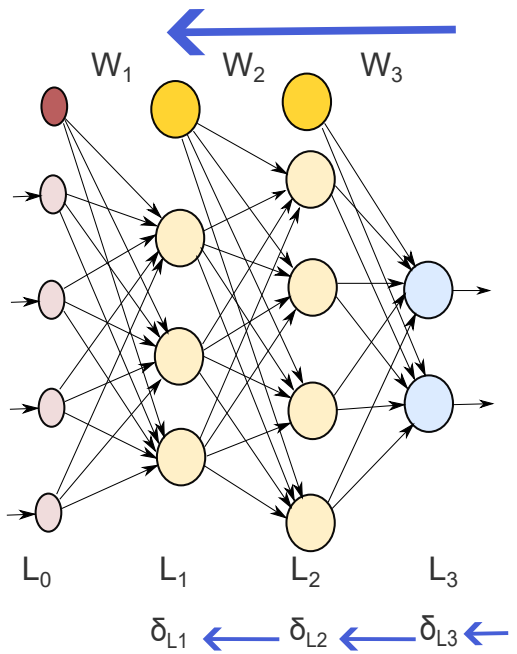
$$\vec{\delta}_{L_{max}} = f'(\vec{\xi}_{L_{max}}) \circ (\vec{y}_t - \vec{d}_t)$$

- Pro skryté vrstvy L spočteme:

$$\vec{\delta}_L = f'(\vec{\xi}_L) \circ \left(\vec{\delta}_{L+1} W_{L+1}^T\right)$$

- Aktualizujeme matici vah W_L mezi vrstvami (L-1) a L:

$$W_L(t+1) = W_L(t) - \alpha \vec{y}_{L-1}^T \vec{\delta}_L$$



Algoritmus zpětného šíření (online varianta)

5. **Ukončující podmínka**
   Pokud není splněna ukončující podmínka, vrátíme se zpět ke kroku 2.
   - Předem daný maximální počet epoch.
   - Časový limit.
   - Jakmile je průměrná chyba na trénovací množině dostatečně malá ... $E < E_{min}$
   - Jakmile přestane klesat průměrná chyba na validační množině dat .... early stopping
   - Jakmile je přírůstek vah Δw moc malý ... $|\Delta w| < \Delta_{min}$

Algoritmus zpětného šíření - učící strategie

Jak předkládat trénovací vzory?

1. **iterativně po epochách (online GD)**: během jedné epochy se každý vzor předloží právě jednou, v rámci každé epochy vzory náhodně uspořádáme
   - maximální počet epoch .... kolikrát se předloží celá trénovací množina
2. **dávkově po epochách (batch GD)**:
   - celá trénovací množina se předloží najednou a váhy se adaptují najednou pro celou trénovací množinu
3. **dávkově po mini-batchích (SGD, stochastic gradient descent)**
   - v každé epoše se trénovací množina náhodně rozdělí na malé podmnožiny vzorů (mini-batch) a ty se iterativně předloží (v rámci mini-batche dávkově)

Algoritmus zpětného šíření (dávková - batch varianta)

Dopředný výpočet maticově pro dávkový (batch) GD:

1. Předložíme matici vstupních vzorů X
2. Pro vrstvy $L = L_0, L_1, ..., L_{max}$ postupně spočítáme matice výstupů těchto vrstev $Y_{L_0}, ..., Y_{L_{max}}$:
   - Pro $L = L_0$ (vstupní vrstva): $Y_{L_0} = X$
   - Pro $L = L_1, .., L_{max}$:

$$Z_L = f(\xi_L) = f(Y_{L-1} W_L)$$

$$Y_L = (1|Z_L)$$

W_L je rozšířená matice vah mezi vrstvou (L-1) a L

3. Výstup sítě $Y = Y_{L_{max}}$ tvoří výstupy neuronů ve výstupní vrstvě

Algoritmus zpětného šíření (dávková - batch varianta)

Zpětný výpočet maticově pro dávkový (batch) GD:

4. Pro vrstvy $L = L_{max}, ..., 1$ postupně spočítáme chybové členy Δ_L a aktualizujeme matice vah W_L mezi vrstvami (L-1) a L:
   - Pro výstupní vrstvu L_{max} spočteme:

$$\Delta_{L_{max}} = f'(\xi_{L_{max}}) \circ (Y - D)$$

   - Pro skryté vrstvy L spočteme:

$$\Delta_L = f'(\xi_L) \circ \left(\delta_{L+1} W_{L+1}^T\right)$$

   - Aktualizujeme matici vah W_L mezi vrstvami (L-1) a L:

$$W_L(t+1) = W_L(t) - \alpha Y_{L-1}^T \Delta_L$$

Algoritmus zpětného šíření - diskuse učící strategie

- Online GD
  - rychlé učení, ale poměrně nestabilní (algoritmus snižuje chybu pro aktuální vzor $\rightarrow$ chyba se může z zvýšit pro ostatní vzory)
  - vyšší citlivost na odlehlé vzory a na volbu hyperparametrů, náhodnost (ale možnost úniku z lokálních minim)
- Batch GD
  - stabilnější, efektivní pro malá data
  - výpočetně a paměťově náročný pro velká data
  - vyšší citlivost na šum v datech
- Mini-batch SGD
  - spojuje výhody obou předchozích strategií
  - používá se pro velké datové sady a hluboké sítě

Dnešní hodina

- Vrstevnaté neuronové sítě s algoritmem zpětného šíření
- **Analýza modelu**
  - Nejprve obecně
  - Rychlost učení a aproximační schopnost
  - Schopnost zobecňovat (příště)

Vrstevnaté neuronové sítě s algoritmem zpětného šíření

- Oblíbený model neuronové sítě
- Jednoduchý algoritmus učení s vysokou variabilitou
- Poměrně dobré aproximační a generalizační schopnosti

Nevýhody

- Lokální metoda učení $\rightarrow$ nemusí najít globální minimum chybové funkce
- Učení s učitelem. Trénovací množina by měla být dostatečně velká a vyvážená.
- Citlivost na inicializaci a hyperparametry.
- Pomalá konvergence, lokální minima, nebezpečí přeučení,
- Interní reprezentace znalostí (černá skříňka).
- Nemá zabudované mechanismy pro zohlednění prostorové struktury dat

Algoritmus zpětného šíření - analýza

Jak zjistit, zda a jak bylo učení úspěšné?

- rychlost učení
- schopnost aproximace (tj. schopnost naučit se danou úlohu)
- schopnost zobecňovat

Co je zásadní pro úspěch algoritmu zpětného šíření?

- normalizace vstupních příznaků
- vhodná inicializace vah (např. $\sim N(0, 1)$)
- nastavit správně hyperparametry pro danou úlohu: architektura (počet vrstev a počet neuronů v jednotlivých vrstvách), přenosové funkce, chybová funkce, parametr učení, ...

Aproximační schopnosti neuronových sítí

- **Už víme:** libovolnou logickou funkci lze reprezentovat pomocí sítě s jednou skrytou vrstvou

Věta (Kolmolgorov)

- Libovolnou spojitou funkci $f : [0,1]^n \to (0,1)^m$ lze vyjádřit (aproximovat libovolně přesně) pomoci vrstevnaté neuronové sítě s odpovídajícím počtem neuronů a s vhodnou přenosovou funkcí.
  - Pro neurony s přenosovou funkcí logsig, tanh, radiální (RBF) stačí **jedna** skrytá vrstva [1989, 1990].
  - Pro další přenosové funkce stačí dvě skryté vrstvy

$\to$ zdálo by se, že nepotřebujeme hluboké architektury

V praxi

- Hlubší architektury mohou vést k lepší reprezentaci dat a vyšší účinnosti učení, zejména pokud jsou trénovány na velkých a složitých datasetech.

NP-úplnost problému učení

Věta

- Obecný problém učení umělých neuronových sítí je NP-úplný. Výpočetní náročnost roste exponenciálně s počtem parametrů.

Poznámky

1. Věta platí i pro model vrstevnaté neuronové sítě a problém učení logických funkcí.
2. Pro některé speciální typy jednoduchých neuronových sítí je problém učení řešitelný v polynomiálním čase (pomocí metod lineárního programování).

Architektura vrstevnaté neuronové sítě

- **mělká (shalllow)** - model s jednou skrytou vrstvou
- **hluboká (deep)** - model s více (nebo i mnoha) skrytými vrstvami

$\rightarrow$ dnes se budeme zabývat modelem s jednou nebo dvěma skrytými vrstvami

Architektura vrstevnaté neuronové sítě

Mělká architektura modelu

- je vhodnější pro jednodušší úlohy - model se pro ně učí rychleji a lépe zobecňuje
- vystačí si s menšími trénovacími daty (naopak velká trénovací data mu nesvědčí)
- je snazší ho pochopit a interpretovat
- **Ale:** hůře se učí složitější vztahy mezi daty
- **Ale:** složité úlohy se učí pomalu a s obtížemi

Architektura vrstevnaté neuronové sítě

- mělká (shalllow) - model s jednou skrytou vrstvou
- hluboká (deep) - model s více (mnoha) skrytými vrstvami

Hluboká architektura modelu

- je vhodnější pro složitější úlohy s velkými trénovacími daty - učí se pro ně lépe
- je schopna zachytit i složité vztahy mezi daty
- vyžaduje jiné učící mechanismy a potýká se v praxi s jinými problémy než mělký model

Vrstevnatá neuronová síť - jaké zvolit přenosové funkce?

Jakou přenosovou funkci použít ve výstupní vrstvě?

- regresní úloha: lineární
- klasifikace do dvou tříd: tanh (případně logsig)
- klasifikace do k tříd: k neuronů s tanh, případně softmax

Jakou přenosovou funkci použít ve skrytých vrstvách?

- tanh - stabilní, ale možné problémy se saturací neuronů
- v případě hlubokých sítí i jiné přenosové funkce, např. ReLU (pozitivně lineární) - rychlejší, efektivnější pro hluboké sítě, ale asymetrie a omezená schopnost reprezentace dat

Vrstevnatá neuronová síť - jakou zvolit cílovou (chybovou) funkci

MSE

- velmi vhodná pro regresní úlohy
- pro klasifikační úlohy lze použít (pro sítě bez výstupní softmax vrstvy)
- citlivá k odlehlým hodnotám

Cross-Entropy

- vhodná pro klasifikaci do více tříd ve spojení se softmax přenosovou funkcí ve výstupní vrstvě
- robustnější k odlehlým hodnotám

Binární Cross-Entropy

- varianta Cross-Entropy vhodná pro klasifikaci do dvou tříd společně s logsig přenosovou funkcí ve výstupní vrstvě

Vrstevnaté neuronové sítě - analýza modelu
Rychlost učení a aproximační schopnosti

- Algoritmus zpětného šíření 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)

Vrstevnaté neuronové sítě - analýza modelu
Rychlost učení a aproximační schopnosti

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.

Vrstevnaté neuronové sítě - analýza modelu
Rychlost učení a aproximační schopnosti

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

- Vhodná inicializace vah a prahů
- Předzpracování a normalizace vstupních dat
- 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 logsig/ tanh 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 přenosové funkce je téměř nulová
    $\rightarrow$ **paralýza sítě** a nekontrolovaný růst vah

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

Vhodná inicializace vah a prahů

Jak snížit riziko saturace neuronů?

I. Vhodná inicializace vah a prahů - např. podle nějaké heuristiky:

- Základ: $w_{ij}(0) \sim N(0,1)$
- Dobrá heuristika, např. volit počáteční váhy z intervalu $[-\frac{2.4}{\sqrt{n_i}}, \frac{2.4}{\sqrt{n_i}}]$, kde n_i je počet hran, které vedou do daného neuronu
- Metoda Nguyen-Widrow - snaží se neurony z dané vrstvy rozhodit ve vstupním prostoru zhruba rovnoměrně

II. Normalizace vstupních dat

- Trénovací vzory by měly být v každém příznaku normalizované v intervalu $[-1,1]$ resp. $[0,1]$ (v závislosti na zvolené přenosové funkci).

Volba vhodného parametru učení

- pro učení je zásadní správné nastavení parametru učení α (learning rate)

Jak volit parametr učení?

- 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

Co pomůže?

- nastavení optimální hodnoty parametru učení pro danou úlohu
- učení s momentem
- adaptivní parametr učení

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 udržovat 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):

$$\begin{aligned} \triangle w_{ij}(t+1) &= -\alpha \frac{\partial E_t}{\partial w_{ij}} + \alpha_m \triangle w_{ij}(t) \\ &= -\alpha \frac{\partial E}{\partial w_{ij}} + \alpha_m (w_{ij}(t) - w_{ij}(t-1)) \end{aligned}$$

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

Algoritmus zpětného šíření s momentem

Moment učení

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

- 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

Jak volit parametr učení a moment učení?

- 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):

$$\triangle w_{ij}(t+1) = -\alpha_{ij,t+1}\frac{\partial E_t}{\partial w_{ij}} + \alpha_m \triangle w_{ij}(t)$$

Adaptivní parametr učení

Heuristika:

- Parametr učení by měl klesat s rostoucím počtem epoch
- **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:
    $\sum_{t=0}^{\infty} \alpha_t = \infty$,
    $\sum_{t=0}^{\infty} \alpha)_t^2 < \infty$

Resilent propagation (Rprop) - Silva & Almeida

Algoritmus založený na změně znaménka parciální derivace:

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

Více variant:

- Silva & Almeida
- Super SAB
- Rprop+
- ...

Resilent propagation - Silva & Almeida

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.

Resilent propagation - 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: $w_{ij}(t+1) = w_{it}(t) - \triangle w_{ij}(t)$
  - Zmenši parametr 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

Resilent propagation - Rprop+

- Parametr učení není řízený znaménkem parciální derivace chybové funkce, ale změnou velikosti chyby
- Rychlejší než Super SAB
- Pro dávkový i online algoritmus zpětného šíření

Algoritmus

- Nastav všechny α_{ij}^0 na počáteční hodnotu α_{start}.
- Proveď krok (t) algoritmu zpětného šíření s momentem.
- Pokud $E_t < E_{t-1}$: $\alpha_{ij,t+1} = u \cdot \alpha_{ij,t}$.
- Pokud $E_t > c \cdot E_{t-1}$:
  - Anuluj předchozí změnu vah: $w_{ij}(t+1) = w_{it}(t) - \triangle w_{ij}(t)$
  - Zmenši parametr učení: $\alpha_{ij,t+1} = d \cdot \alpha_{ij,t}$
- Konstanty c, u, d jsou pevně zvolené tak, že $c > 1$, $u > 1$, $d < 1$

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

- vhodné pro učení ,,mělkých" neuronových sítí
- 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 parciálních 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

Malá odbočka: algoritmy učení hlubokých neuronových sítí

- vycházejí z GD a často používají adaptivní a lokální parametr učení
  - Gradient Descent (základní algoritmus zpětného šíření)
  - SGD (stochastický algoritmus zpětného šíření využívající mini-batche)
  - AdaGrad
  - RMSprop
  - Adadelta
  - Adam (aktuálně zřejmě nejpopulárnější)
  - AdaMax
  - NAdam
  - FTRL ...