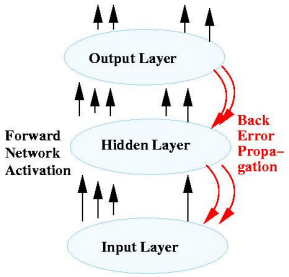
Uczenie sieci neuronowych

Ważniejsze metody uczenia perceptronów wielowarstwowych

- *Back Propagation* (wstecznej propagacji błędów),
- *Quick Propagation* (szybkiej propagacji błędów).
- *Conjugate Gradients* (gradientów sprzężonych),
- *Quasi-Newton* (metoda zmiennej metryki),
- *Levenberg – Marquardt* (metoda paraboloidalnych modeli funkcji błędów),
- Inne metody „nieklasyczne” (angażujące zmienne metryki, metody bezgradientowe itp.)

Zaczniemy od omówienia metody *backpropagation*



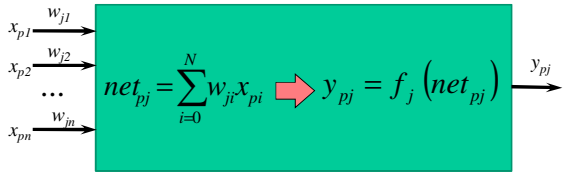
Dla poprawnej interpretacji następnych wzorów (których będzie niestety sporo...) wprowadzimy jednolity system oznaczeń:

Rozmiar warstwy wejściowej sieci oznaczymy ***N*** oraz rozmiar warstwy wyjściowej oznaczymy ***M***.

Oba te rozmiary są zależne od sposobu postawienia rozwiązywanego zadania.

Natomiast rozmiar warstwy ukrytej oznaczony ***L*** jest zależny o twórcy sieci.

Działanie pojedynczego neuronu o numerze j w chwili p będzie wykonywane wg. schematu:



x_{pi}, - wartość sygnału na i-tym wejściu neuronu w chwili czasowej p

w_{ji}, - wartość wagi na i-tym wejściu neuronu o numerze j

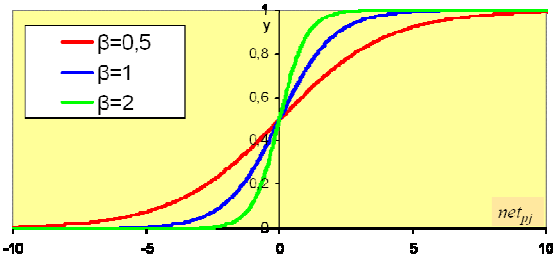
Agregacja sygnałów wejściowych net_{pj} oznacza potencjał pomocniczy

Wyznaczany jest sygnał wyjściowy neuronu y_{pj}

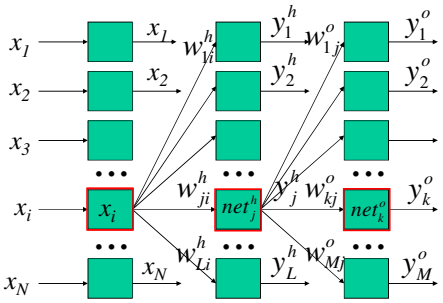
Funkcja $f_j(\ldots)$ jest nieliniowym (najczęściej) przekształceniem, opisującym proces generacji sygnału wyjściowego y_{pj}.

Używanych jest wiele różnych funkcji, dzięki czemu można mówić o wielu typach sieci neuronowych, jednak najpowszechniej używana jest funkcja logistyczna, nazywana sigmoidą

$$f_j(net_{pj}) = \frac{1}{1+\exp(-\beta\, net_{pj})}$$



Struktura sieci neuronowej

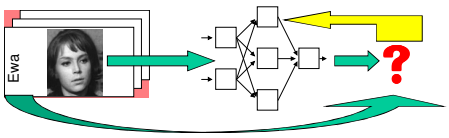


Opis działania rozważanych algorytmów uczenia sieci przedstawiony zostanie w oparciu o ciąg uczący o postaci:

$$\langle x_1, d_1\rangle, \langle x_2, d_2\rangle, \ldots, \langle x_p, d_p\rangle, \ldots, \langle x_U, d_U\rangle$$

gdzie wartości wejściowe $x_p \in R^N$, zaś oczekiwane (zakładane) wartości wyjściowe $d_p \in R^M$ określone są dla każdego $p = 1, \ldots, U$.

Przykład uczenia sieci zadania rozpoznawania twarzy

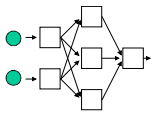


Liczebność zbioru uczącego U zależy od tego, ile wiedzy posiadamy a'priori o prawidłowym (wzorcowym) rozwiązaniu tego zadania.

Oznaczenie p używane będzie stale jako numer aktualnie rozważanego kroku procesu uczenia.
Oczywiście $p = 1, \ldots, U$

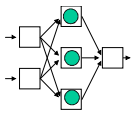
W trakcie działania sieci neuronowej wyróżnić można następujące etapy:

•wprowadzenie na wejście sieci p - tego wektora wejściowego (x_p),



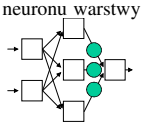
•obliczenie dla każdego neuronu warstwy ukrytej wartości net_{pj}^h zgodnie ze wzorem:

$$net_{pj}^h = \sum_{i=0}^{N} w_{ji}^h x_{pi}$$



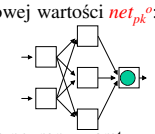
•obliczenie wartości wyjściowej y_{pj}^h dla każdego neuronu warstwy ukrytej:

$$y_{pj}^h = f_j^h\left(net_{pj}^h\right)$$



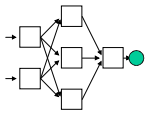
•obliczenie dla każdego neuronu warstwy wyjściowej wartości net_{pk}^o:

$$net_{pk}^o = \sum_{j=0}^{L} w_{kj}^o y_{pj}^h$$



•obliczenie wartości wyjściowej y_{pk}^o dla każdego neuronu warstwy wyjściowej:

$$y_{pk}^o = f_k^o\left(net_{pk}^o\right)$$



Po wykonaniu wszystkich czynności wchodzących w skład wszystkich tych etapów sieć ustala swój sygnał wyjściowy y_{pk}^{o}.

Sygnał ten może być prawidłowy lub nie, ale w tym właśnie celu mamy **proces uczenia**, żeby **rzeczywiste** działanie sieci dopasować do działania **pożądanego**.

W prawidłowo działającej sieci wektor wartości wyjściowych y_p powinien być identyczny lub bardzo zbliżony do wektora wartości oczekiwanych d_p.

Aby do tego doprowadzić stosuje się opisane dalej algorytmy uczenia.

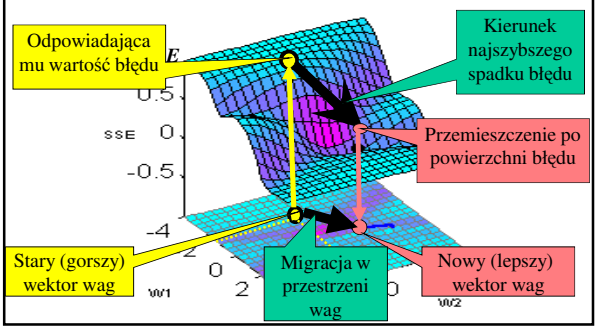
Klasyczna metoda wstecznej propagacji błędów

Celem procesu uczenia jest taki dobór wag, który zapewni minimalizację wartości błędu E będącego sumą błędów E_p obliczonych dla każdego elementu ciągu uczącego.

Błąd E_p dla p-tego elementu ciągu uczącego wyraża formuła:

$$E_p = \frac{1}{2}\sum_{k=1}^{M}\left(d_{pk} - y_{pk}^{o}\right)^2$$

Algorytm wstecznej propagacji błędów polega na szukaniu kierunku spadku E i na takim zmienianiu wartości wag w_1, w_2, żeby zmniejszać wartość funkcji błędu w kierunku jej najszybszego spadku



Po prezentacji p-tego elementu ciągu uczącego modyfikacja wektora wag odbywa się w tej metodzie zgodnie ze wzorem:

$$\mathbf{w}^{(p+1)} = \mathbf{w}^{(p)} - \eta \nabla E_p\left(\mathbf{w}^{(p)}\right)$$

gdzie:

$\mathbf{w}^{(p)}$ - wektor wszystkich wag sieci w p - tym kroku działania algorytmu,

$\nabla E_p(\mathbf{w}^{(p)})$ - gradient funkcji E_p w punkcie $\mathbf{w}^{(p)}$,

η - współczynnik uczenia będący stałą z przedziału (0, 1).

W trakcie działania algorytmu modyfikowane być muszą zarówno neuronów warstwy wyjściowej jak i wagi neuronów warstwy ukrytej, co stanowi główne źródło trudności.

Modyfikacja wag przebiega zgodnie z wzorami:

$$w_{kj}^{o}(p+1) = w_{kj}^{o}(p) - \eta\frac{\partial E_p}{\partial w_{kj}^{o}}$$

$$w_{ji}^{h}(p+1) = w_{ji}^{h}(p) - \eta\frac{\partial E_p}{\partial w_{ji}^{h}}$$

W celu obliczenia wartości $\frac{\partial E_p}{\partial w^o_{kj}}$

najpierw oblicza się pochodną $\frac{\partial E_p}{\partial y^o_{pk}}$

$$\frac{\partial E_p}{\partial y^o_{pk}} = \frac{\partial}{\partial y^o_{pk}}\left(\frac{1}{2}\sum_{k=1}^{M}\left(d_{pk} - y^o_{pk}\right)^2\right) = -\left(d_{pk} - y^o_{pk}\right)$$

następnie

$$\frac{\partial E_p}{\partial net^o_{pk}} = \frac{\partial E_p}{\partial y^o_{pk}}\frac{\partial y^o_{pk}}{\partial net^o_{pk}} = -\left(d_{pk} - y^o_{pk}\right)\frac{\partial \mathrm{f}\left(net^o_{pk}\right)}{\partial net^o_{pk}}$$

W ostatnim wzorze pojawia się **pochodna** nieliniowej **funkcji przejścia neuronu** względem jej argumentu: $\frac{\partial \mathrm{f}\left(net^o_{pk}\right)}{\partial\, net^o_{pk}}$

W praktyce wyznaczenie wartości tej pochodnej nie jest jednak trudne, ponieważ funkcje przejścia dobiera się w taki sposób, aby miały **wygodne** (z punktu widzenia tych rozważań) formuły wyrażające ich pochodne.

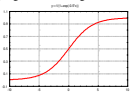
Dla uproszczenia notacji na chwilę zastąpmy wyrażenie net^o_{pk} jakimś pojedynczym symbolem, na przykład s.

Związek między sumarycznym pobudzeniem neuronu s a jego sygnałem wyjściowym, który też w uproszczeniu zapiszemy na chwilę jako y wyraża się wtedy prostym wzorem:

$$y = f(s)$$

Dla często stosowanej w sieciach neuronowych funkcji przejścia w kształcie ***sigmoidy*** mamy następującą formułę:

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



Znajdując pochodną tej funkcji mamy:

$$f'(s) = \frac{d}{ds}(1+e^{-s})^{-1} = -(1+e^{-s})^{-2}e^{-s}(-1)$$

Powyższa formuła jest konstruktywna, to znaczy mając do dyspozycji

$$s = net^o_{pk}$$

można wyznaczyć potrzebną pochodną $f'(s)$ dla każdego neuronu o numerze k i dla każdego kroku procesu uczenia p.

Można jednak uzyskać formułę jeszcze wygodniejszą w użyciu.

Przekształcając wyprowadzony wyżej wzór na pochodną otrzymujemy kolejno:

$$f'(s) = -(1+e^{-s})^{-2}e^{-s}(-1) = \frac{1}{1+e^{-s}}\frac{e^{-s}}{1+e^{-s}} =$$

$$= \frac{1}{1+e^{-s}}\frac{1+e^{-s}-1}{1+e^{-s}} = \frac{1}{1+e^{-s}}\left(1-\frac{1}{1+e^{-s}}\right) = y(1-y)$$

Ostatni wzór przepisany w postaci:

$$f'(s) = y\,(1-y)$$

Pokazuje, że można niezwykle łatwo wyznaczyć potrzebną pochodną, ponieważ sygnał wyjściowy y jest w oczywisty sposób znany dla każdego neuronu o numerze k i dla każdego kroku procesu uczenia p

Inne typowo używane funkcje przejścia także są tak dobierane, że mają stosunkowo proste formuły dla pochodnych – i to takie, które łatwo jest wyznaczyć w praktyce na podstawie dostępnych w sieci sygnałów.

Na przykład dla chętnie używanej funkcji

$$f(s) = tanh(s)$$

$$f'(s) = (1+y)\,(1-y)$$

Po wprowadzeniu dodatkowej zmiennej:

$$\delta^o_{pk} = \left(d_{pk} - y^o_{pk}\right)\frac{\partial \mathrm{f}\left(net^o_{pk}\right)}{\partial\, net^o_{pk}}$$

otrzymujemy wygodne podstawienie:

$$\frac{\partial E_p}{\partial net^o_{pk}} = -\delta^o_{pk}$$

i wtedy mamy ostatecznie wzór:

$$\frac{\partial E_p}{\partial w^o_{kj}} = \frac{\partial E_p}{\partial net^o_{pk}}\frac{\partial net^o_{pk}}{\partial w^o_{kj}} = -\delta^o_{pk}\frac{\partial}{\partial w^o_{kj}}\left(w^o_{k0} + w^o_{k1}y^h_{p1} + \ldots + w^o_{kj}y^h_{pk} + \ldots + w^o_{kL}y^h_{pL}\right)$$

Ostateczny wzór opisujący zmianę wag neuronów **warstwy wyjściowej** przedstawia się następująco:

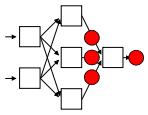
$$w_{kj}^{o}(p+1) = w_{kj}^{o}(p) + \eta \delta_{pk}^{o} y_{pj}^{h}$$

gdzie wartość

$$\delta_{pk}^{o} = \left(d_{pk} - y_{pk}^{o}\right)\frac{\partial \mathrm{f}\left(net_{pk}^{o}\right)}{\partial net_{pk}^{o}} = \left(d_{pk} - y_{pk}^{o}\right) y_{pk}^{o}\left(1 - y_{pk}^{o}\right)$$

(ostatnie podstawienie ważne tylko dla sigmoidy!)

może być wyznaczona dla warstwy wyjściowej **bezpośrednio**, bo tam znamy d_{pk}



Mamy więc efektywny wzór dla neuronów warstwy wyjściowej:

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta\left(d_{pk} - y_{pk}^{o}\right) y_{pk}^{o}\left(1 - y_{pk}^{o}\right) y_{pj}^{h}$$

W analogiczny sposób wyznaczyć można formułę określającą zmiany wag w neuronach warstwy ukrytej.

Zmiana wartości wyjściowej j - tego neuronu warstwy ukrytej wpływa na zmianę E_p następująco:

$$\frac{\partial E_p}{\partial y_{pj}^{h}} = \frac{\partial}{\partial y_{pj}^{h}}\left(\frac{1}{2}\sum_{k=1}^{M}\left(d_{pk} - y_{pk}^{o}\right)^2\right) = \sum_{k=1}^{M}\frac{\partial}{\partial y_{pj}^{h}}\left(\frac{1}{2}\left(d_{pk} - y_{pk}^{o}\right)^2\right) =$$

$$= -\sum_{k=1}^{M}\left(d_{pk} - y_{pk}^{o}\right)\frac{\partial \mathrm{f}\left(net_{pk}^{o}\right)}{\partial y_{pj}^{h}} = -\sum_{k=1}^{M}\left(d_{pk} - y_{pk}^{o}\right)\frac{\partial \mathrm{f}\left(net_{pk}^{o}\right)}{\partial net_{pk}^{o}}\frac{\partial net_{pk}^{o}}{\partial y_{pj}^{h}} =$$

$$= -\sum_{k=1}^{M}\left(d_{pk} - y_{pk}^{o}\right)\frac{\partial \mathrm{f}\left(net_{pk}^{o}\right)}{\partial net_{pk}^{o}} w_{kj}^{o} = -\sum_{k=1}^{M}\delta_{pk}^{o} w_{kj}^{o}$$

Następnie oblicza się pochodną E_p względem sumy iloczynów wejść i wag dla neuronu warstwy ukrytej:

$$\frac{\partial E_p}{\partial net_{pj}^{h}} = \frac{\partial E_p}{\partial y_{pj}^{h}}\frac{\partial y_{pj}^{h}}{\partial net_{pj}^{h}} = -\left(\sum_{k=1}^{M}\delta_{pk}^{o} w_{kj}^{o}\right)\frac{\partial \mathrm{f}\left(net_{pj}^{h}\right)}{\partial net_{pj}^{h}}$$

po wprowadzeniu dodatkowej zmiennej:

$$\delta_{pj}^{h} = \left(\sum_{k=1}^{M}\delta_{pk}^{o} w_{kj}^{o}\right)\frac{\partial \mathrm{f}\left(net_{pj}^{h}\right)}{\partial net_{pj}^{h}}$$

mamy:

$$\frac{\partial E_p}{\partial net_{pj}^{h}} = -\delta_{pj}^{h}$$

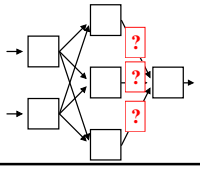
Ostatnim etapem jest wykonanie obliczeń wartości wyrażenia:

$$\frac{\partial E_p}{\partial w_{ji}^{h}} = \frac{\partial E_p}{\partial net_{pj}^{h}}\frac{\partial net_{pj}^{h}}{\partial w_{ji}^{h}} = -\delta_{pj}^{h} x_{pi}$$

wzór na zmianę wag dla neuronów warstwy ukrytej:

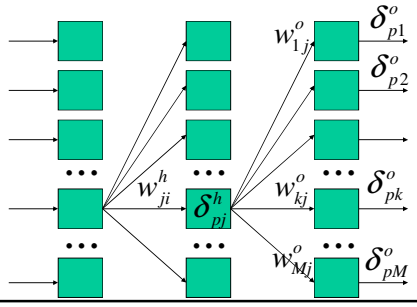
$$w_{ji}^{h}(p+1) = w_{ji}^{h}(p) + \eta \delta_{pj}^{h} x_{pi}$$

problemem jest tu wyznaczenie wartości błędów neuronów ukrytych, bo ich sygnały wyjściowe nie podlegają bezpośredniej ocenie



Z pomocą przychodzi tu formuła wstecznej propagacji błędów, od której cała metoda bierze swoją nazwę:

$$\delta_{pj}^{h} = \left(\sum_{k=1}^{M}\delta_{pk}^{o} w_{kj}^{o}\right)\frac{\partial \mathrm{f}\left(net_{pj}^{h}\right)}{\partial net_{pj}^{h}} = \left(\sum_{k=1}^{M}\delta_{pk}^{o} w_{kj}^{o}\right) y_{pj}^{h}\left(1 - y_{pj}^{h}\right)$$

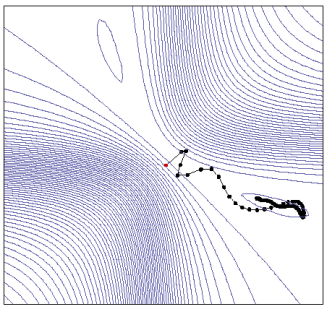


Ostateczny wzór, pozwalający wyznaczać poprawki wag neuronów warstwy ukrytej:

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta\delta_{pj}^{h}x_{pi} = w_{ji}^{h}(t) + \eta\left(\sum_{k=1}^{M}\delta_{pk}^{o}w_{kj}^{o}\right)\frac{\partial \mathrm{f}\left(net_{pj}^{h}\right)}{\partial net_{pj}^{h}}x_{pi} =$$

$$= w_{ji}^{h}(t) + \eta\left(\sum_{k=1}^{M}\delta_{pk}^{o}w_{kj}^{o}\right)y_{pj}^{h}\left(1-y_{pj}^{h}\right)x_{pi}$$

Backpropagation jest generalnie metodą dosyć wolną w działaniu:



91 iterations

Niektórych badaczy to ogromnie denerwuje, więc na jednej konferencji naukowej rozprowadzane były takie ulotki:

W celu przezwyciężenia wad algorytmu *backpropagation*, takich jak:

- powolność algorytmu,
- tendencja do zatrzymywania się algorytmu w minimach lokalnych funkcji E,

najczęściej stosuje się dodatkowy składnik nazywany pędem *(momentum):*

$$\Delta\mathbf{w}^{p} = -\eta\nabla E\left(\mathbf{w}^{p}\right) + \mu\Delta\mathbf{w}^{p-1}$$

gdzie μ jest współczynnikiem *momentum,* określającym wpływ poprzedniej modyfikacji wektora wag na aktualny wektor poprawek.

Żeby **przyspieszyć** działanie metody *backpropagation* stosuje się specjalne metody doboru współczynnika uczenia

Podstawowy pomysł polega na tym, że gdy w kolejnym kroku algorytmu wartość funkcji błędu **maleje**, to współczynnik uczenia η jest nieznacznie **zwiększany**, natomiast gdy wartość błędu **wzrasta** w znacznym stopniu w stosunku do poprzedniego kroku algorytmu (prawdopodobnie następuje oddalanie się od minimum funkcji błędu), wartość współczynnika η jest **zmniejszana**.

Sposoby zwiększania i zmniejszania współczynnika η najczęściej są arbitralne.

Na przykład w pakiecie MATLAB wartość współczynnika uczenia jest zwiększana poprzez przemnożenia aktualnej wartości współczynnika przez stałą większą od jedności (domyślna wartość tej stałej wynosi 1.05).

Z kolei Autorzy Neural Network Toolbox przyjęli zasadę, że jeżeli iloraz uzyskanego błędu do błędu uzyskanego w poprzednim kroku algorytmu przekroczy pewną wartość (domyślnie 1.04), to wówczas współczynnik uczenia jest przemnażany przez pewną stałą mniejszą od jedności (domyślnie 0.7).

Z kolei w metodzie, którą zaproponowali F. Silva i L. Almeida różnicuje się współczynniki uczenia η związane z poszczególnymi składowymi wektora wag:

$$\Delta w_i^{(t)} = -\eta_i^{(t)} \cdot \nabla_i E\left(w^t\right)$$

W początkowej fazie algorytmu wszystkim współczynnikom uczenia nadaje się taką samą wartość, będącą niewielką liczbą dodatnią $\eta_i^{(0)}$. Po wykonaniu kroku t dokonuje się zmiany wartości współczynnika uczenia w następujący sposób:

$$\eta_i^{(t+1)} = \begin{cases} \eta_i^{(t)} \cdot u & gdzie \quad \nabla_i E\left(w^t\right) \cdot \nabla_i E\left(w^{t-1}\right) \geq 0 \\ \eta_i^{(t)} \cdot d & gdzie \quad \nabla_i E\left(w^t\right) \cdot \nabla_i E\left(w^{t-1}\right) < 0 \end{cases}$$

gdzie wartości u oraz d są odpowiednio dobranymi stałymi spełniającymi warunki: $u > 1$ oraz $d < 1$.

Nieco inna jest reguła modyfikacji współczynnika uczenia η podana przez Jacobsa:

$$\eta_i^{(t+1)} = \eta_i^{(t)} - \gamma \frac{\partial E\left(w^t\right)}{\partial \eta_i^{(t)}}$$

gdzie γ jest pewnym dodatnim, stałym współczynnikiem z przedziału [0, 1].

Czasami zmianę współczynnika uczenia opiera się na wzorze:

$$\eta_i^{(t+1)} = \eta_i^{(t)} + \gamma \frac{\partial E\left(w^t\right)}{\partial w_i^t} \cdot \frac{\partial E\left(w^{t-1}\right)}{\partial w_i^{t-1}}$$

Wynika z niego, że współczynnik uczenia rośnie, jeżeli odpowiednie składowe wektora gradientu nie zmieniają znaku w dwóch kolejnych iteracjach. W przeciwnym przypadku współczynnik uczenia maleje.

W praktyce stosuje się często inną, podaną tajże przez Jacobsa, modyfikację opisanej reguły zmian η zwanej *delta-bar-delata*:

$$\eta_i^{(t+1)} = \begin{cases} \eta_i^{(t)} + u & gdzie \quad \nabla_i E\left(w^t\right) \cdot \delta_i^{(t-1)} > 0 \\ \eta_i^{(t)} \cdot d & gdzie \quad \nabla_i E\left(w^t\right) \cdot \delta_i^{(t-1)} < 0 \\ \eta_i^{(t)} & gdzie \quad \nabla_i E\left(w^t\right) \cdot \delta_i^{(t-1)} = 0 \end{cases}$$

gdzie u oraz d są wartościami stałymi (najczęściej $0 < u < 0.05$ oraz $0.1 < d < 0.3$) zaś $\delta_i^{(t)}$ dane jest wzorem:

$$\delta_i^{(t)} = (1-\phi)\nabla_i E\left(w^t\right) + \phi \delta_i^{(t-1)}$$

Powiększanie wartości współczynnika uczenia η następuje w sposób liniowy poprzez dodanie stałej u, zaś zmniejszanie współczynnika dokonywane jest w sposób wykładniczy poprzez przemnożenie przez mniejszą od jedności stałą d.

Czasem dla polepszenia jakości działania algorytmu ***backpropagation*** stosuje się też mechanizm zanikania wag:

$$\Delta \mathbf{w}^t = -\eta \nabla E\left(\mathbf{w}^t\right) - \beta \mathbf{w}^t$$

albo technikę kumulowania poprawek (*cumulative weight adjustment*) polegającą na tym, że po prezentacji każdego wzorca obliczany jest wektor zmiany wag $\Delta \mathbf{w}_p$, ale modyfikacja wartości wag dokonywana jest po prezentacji wszystkich elementów ciągu uczącego (raz w ciągu całej epoki). Wagi zmieniane są wtedy o wartość:

$$\Delta \mathbf{w} = \sum_{p=1}^{K} \Delta \mathbf{w}_p$$

Wszystkie te pomysłowe metody w naprawdę trudnych zadaniach dają raczej **mizerne** wyniki.

Dlatego wymyślono metodę przyspieszonej wstecznej propagacji, zwaną ***Quickprop***:

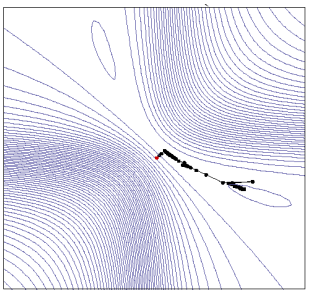
W metodzie tej zakłada się, że funkcja błędu jest lokalnie ***paraboloidalna***.

Korzystając z tego założenia wartość wektora poprawki **Δw** oblicza się w taki sposób, aby w trakcie **jednej iteracji** osiągnąć minimum funkcji błędu (minimum paraboloidy). Wartość poprawki dana jest wzorem:

$$\Delta \mathbf{w}^t = \frac{\nabla E\left(\mathbf{w}^t\right)}{\nabla E\left(\mathbf{w}^{t-1}\right) - \nabla E\left(\mathbf{w}^t\right)} \Delta \mathbf{w}^{t-1}$$

Chociaż zazwyczaj przytoczone wyżej założenie nie jest dokładnie spełnione, to jednak metoda ta znacznie przyspiesza proces uczenia

Quickprop



63 iterations

Kolejna modyfikacja metody wstecznej propagacji błędów, zwana ***RPROP*** *(resilient backpropagation)*, której autorami są M. Riedmiller oraz H. Braun wykorzystuje minimalizację **zmodyfikowanej** funkcji błędu o postaci:

$$E = \sum_{p=1}^{R}\sum_{k=1}^{M}\left(d_{pk} - y_{pk}^{o}\right)^2 + 10^{-\alpha}\sum w_{ij}^2$$

gdzie:

α - parametr określający tempo zanikania wag,

$\sum w_{ij}^2$ - suma kwadratów wszystkich wag występujących w sieci.

Modyfikacja wartości wag w algorytmie RPROP dokonywana jest po prezentacji wszystkich elementów ciągu uczącego (uczenie skumulowane).

W metodzie ***RPROP*** każda z modyfikowanych wag w_{ij} występujących w sieci zmieniana jest w następujący sposób:

$$\Delta w_{ij}^{t} = \begin{cases} -\Delta_{ij}^{t}, & gdy \quad \dfrac{\partial E(w^t)}{\partial w_{ij}} > 0 \\ \Delta_{ij}^{t}, & gdy \quad \dfrac{\partial E(w^t)}{\partial w_{ij}} < 0 \\ 0, & gdy \quad \dfrac{\partial E(w^t)}{\partial w_{ij}} = 0 \end{cases}$$

gdzie Δ_{ij}^{t} jest wielkością, o którą zmienia się wartość wagi w_{ij} w t - tej epoce procesu uczenia.

W metodzie ***RPROP*** każdorazowo po dokonaniu zmiany wag następuje proces modyfikowania wielkości Δ_{ij}.

Dokonuje się on w następujący sposób:

$$\Delta_{ij}^{t+1} = \begin{cases} \theta^{+} * \Delta_{ij}^{t}, & gdy \quad \dfrac{\partial E(w^t)}{\partial w_{ij}} \dfrac{\partial E(w^{t+1})}{\partial w_{ij}} > 0 \\ \theta^{-} * \Delta_{ij}^{t}, & gdy \quad \dfrac{\partial E(w^t)}{\partial w_{ij}} \dfrac{\partial E(w^{t+1})}{\partial w_{ij}} < 0 \\ \Delta_{ij}^{t}, & gdy \quad \dfrac{\partial E(w^t)}{\partial w_{ij}} \dfrac{\partial E(w^{t+1})}{\partial w_{ij}} = 0 \end{cases}$$

gdzie $0 < \theta^{-} < 1 < \theta^{+}$.

Powyższa reguła stosowana w metodzie ***RPROP*** powoduje, że w przypadku, gdy odpowiednia pochodna cząstkowa zmieniła swój znak, następuje zmniejszenie wartości Δ_{ij}. Jeżeli znak pochodnej nie uległ zmianie, to wartość Δ_{ij} jest nieznacznie zwiększana.

Jeżeli nastąpiła zmiana znaku pochodnej cząstkowej to w następnym kroku algorytmu nie przeprowadza się modyfikacji wielkości Δ_{ij}.

Parametrami metody RPROP są również wartość początkowa Δ_{ij}^{0} oraz wartość maksymalna Δ_{max} i wartość minimalna Δ_{min}.

Po osiągnięciu jednej z podanych wartości granicznych wielkość Δ_{ij} nie jest już więcej modyfikowana.

Algorytmu *QRprop*

Autorami są M. Pfister i R. Rojas. Łączy on cechy metody *Quickprop* i *RPROP*. Sposób modyfikacji wag:

- jeżeli składowe gradientu $\nabla_i E(\mathbf{w}^{(t)})$ oraz $\nabla_i E(\mathbf{w}^{(t-1)})$ w kolejnych iteracjach nie zmieniły znaku (ich iloczyn jest **dodatni**) to stosowana jest modyfikacja wag zgodna z metodą *RPROP*,
- zmiana znaku składowych gradientu (iloczyn **ujemny**) oznacza, że nastąpił przeskok nad minimum lokalnym - w takiej sytuacji stosowana jest modyfikacja zgodna z regułą *Quickprop*, zaś wartości składowej gradientu $\nabla_i E(\mathbf{w}^{(t)})$ przypisuje się wartość 0,
- w przypadku, gdy iloczyn jest równy **zero** (co ma miejsce wówczas, gdy w kroku nadano składowej gradientu wartość 0 lub wtedy, gdy osiągnięto minimum lokalne) wykonuje się krok zgodnie z metodą *Quickprop* wykorzystując składowe wektorów gradientu $\nabla_i E(\mathbf{w}^{(t)})$ oraz $\nabla_i E(\mathbf{w}^{(t-2)})$

Wszystkie te sztuczki nie zmieniają faktu, że wszystkie algorytmy bazujące na koncepcji ***backpropagation*** działają jednak raczej wolno.

Stąd poszukuje się stale dalszych nowych algorytmów.

Ważną techniką istotnie różną od ***backpropagation*** jest metoda gradientów sprzężonych

Idea metody jest następująca: kierunek poszukiwań **p** w *t* - tym kroku algorytmu uczenia wyznacza się zgodnie ze wzorem:

$$\mathbf{p}^t = -\nabla E\left(\mathbf{w}^t\right) + \beta^t \mathbf{p}^{t-1}$$

gdzie:

$\mathbf{p}^{t-1}$ jest wektorem poszukiwań w kroku poprzednim,

β^t jest współczynnikiem sprzężenia

β^t jest współczynnikiem sprzężenia, którego wartość może być dobierana według różnych strategii, najczęściej w oparciu o wzór Polaka - Ribiery:

$$\beta^t = \frac{\nabla E\left(w^t\right)^T \left(\nabla E\left(w^t\right) - \nabla E\left(w^{t-1}\right)\right)}{\left(\nabla E\left(w^{t-1}\right)\right)^T \left(\nabla E\left(w^{t-1}\right)\right)}$$

Istota metody polega na tym, że dzięki zastosowaniu przy wyznaczaniu kierunku poszukiwań składnika uwzględniającego współczynnik sprzężenia β^t, metoda ta w kolejnych krokach algorytmu dobiera maksymalnie różne kierunki poszukiwania.

Dokładniej - w kolejnych krokach algorytmu dobierane są takie kierunki poszukiwania minimum funkcji błędu *E*, które pozwalają poszukiwać optymalnego punktu w kierunku wnoszącym maksymalnie duży zysk w postaci zmalenia wartości *E* **w stosunku do kierunku wcześniej eksplorowanego.**

Po wyznaczeniu aktualnego kierunku poszukiwań $\mathbf{p}^t$ wyznacza się wektor zmiany wag:

$$\Delta\mathbf{w}^t = \eta\mathbf{p}^t$$

dobierając współczynnik η w taki sposób, który zapewnia **minimalizację** funkcji *E* w wyznaczonym kierunku poszukiwań **p**.

Jest to bardzo ważne, bo w kolejnych krokach wybierane będą kierunki ***sprzężone*** w stosunku do **p,** czyli **maksymalnie od p odmienne**. Jeśli więc algorytm nie „wyciśnie" maksimum poprawy wartości *E* w ciągu tego jednego jedynego kroku w kierunku **p**, to potem już nie będzie okazji, żeby to poprawić!

Dobór wartości współczynnika η może być dokonany na przykład przy upraszczającym założeniu, że przekrój funkcji błędu w kierunku **p** ma kształt paraboli. **Uwaga:** zadanie to jest uproszczone, bo przekrój funkcji *E* w kierunku **p** jest zwykłą funkcją jednej zmiennej!

Po wyznaczeniu kierunku poprawy $\mathbf{p}_k$ w metodzie gradientów sprzężonych należy znaleźć minimum funkcji kryterium $E(\mathbf{w}^k)$ w tym kierunku.

Definiując rozwinięcie w szereg Taylora funkcji $E(w)$ w kierunku $\mathbf{p}_k$ uzyskuje się zależność:

$$E(\mathbf{w}_k + \eta\mathbf{p}_k) \approx E(\mathbf{w}_k) + \eta\nabla E(\mathbf{w}_k)^T \mathbf{p}_k + \tfrac{1}{2}\eta^2 \mathbf{p}_k^T \mathbf{H}(\mathbf{w}_k)\mathbf{p}_k$$

gdzie **H** jest hesjanem funkcji błędu.

Po oszacowaniu parametrów paraboli można w sposób analityczny wyznaczyć jej minimum. Na tej podstawie wartość η wyznaczana jest w taki sposób, aby **w jednym kroku** zapewnić przejście z punktu Δw^t do punktu Δw^{t+1} położonego w punkcie, w którym wyznaczona parabola osiąga minimum.

Warunkiem koniecznym osiągnięcia minimum funkcji błędu jest:

$$\frac{\partial E(\mathbf{w}_k + \eta\mathbf{p}_k)}{\partial(\eta\mathbf{p}_k)} = \nabla E(\mathbf{w}_k) + \eta\mathbf{H}(\mathbf{w}_k)\mathbf{p}_k = 0$$

Na tej podstawie można wyznaczyć potrzebną długość kroku:

$$\eta = -\frac{\mathbf{p}_k^T \nabla E(\mathbf{w}_k)}{\mathbf{p}_k^T \mathbf{H}(\mathbf{w}_k)\mathbf{p}_k} = \frac{\mu_k}{\delta_k}$$

W celu wyznaczenia wartości η należy więc obliczyć:

$$\mu_k = -\mathbf{p}_k^T \nabla E(\mathbf{w}_k) \qquad \text{oraz} \qquad \delta_k = \mathbf{p}_k^T \mathbf{H}(w_k)\mathbf{p}_k$$

i obliczyć ich stosunek.

Stosowanie opisanej metody w praktyce stwarza dwa poważne problemy:

- wyznaczenie wartości hesjanu jest nieopłacalne z uwagi na dużą złożoność obliczeniową oraz zajętość pamięci komputera,
- warunkiem koniecznym do tego, aby podany wzór prowadził do minimum funkcji celu jest dodatnia określoność hesjanu .

Rozwiązaniem problemu może być zastosowanie przybliżonej (numerycznej) metody wyznaczania wartości pomocniczej:

$$\mathbf{H}(\mathbf{w}_k)\mathbf{p}_k$$

Przybliżenie to wyznacza się na podstawie wartości wektora gradientu w punktach: $\mathbf{w}_k$ oraz $\mathbf{w}_k + \sigma_k\mathbf{p}_k$

gdzie wartość σ_k spełnia warunek: $0 < \sigma_k << 1$

Na podstawie tych wartości można wyznaczyć przybliżoną wartość:

$$\mathbf{H}(\mathbf{w}_k)\mathbf{p}_k \approx s_k = \frac{\nabla E(\mathbf{w}_k + \sigma_k \mathbf{p}_k) - \nabla E(\mathbf{w}_k)}{\sigma_k}$$

Wartość ta może być poddana dodatkowo regularyzacji, której celem jest zapobieganie utracie warunku dodatniej określoności wyznaczanej aproksymacji hesjanu.

Regularyzację wykonuje się zgodnie ze wzorem:

$$s_k = \frac{\nabla E(\mathbf{w}_k + \sigma_k \mathbf{p}_k) - \nabla E(\mathbf{w}_k)}{\sigma_k} + \lambda_k \mathbf{p}_k$$

Parametr regularyzacji λ_k musi być dobrany w taki sposób, aby:

$$\delta_k = \mathbf{p}_k^T s_k > 0$$

Jeżeli przyjęty w k - tym kroku algorytmu parametr regularyzacji λ_k nie zapewnia spełnienia warunku:

$$\delta_k = \mathbf{p}_k^T s_k > 0$$

to należy go zwiększyć do takiej wartości $\lambda_k^{(r)}$ dla której ten warunek będzie spełniony.

Wartość s_k dla parametru $\lambda_k^{(r)}$ obliczana jest ze wzoru:

$$\mathbf{s}_k^{(r)} = \mathbf{s}_k + \left(\lambda_k^{(r)} - \lambda_k\right)\mathbf{p}_k$$

gdzie s_k jest wartością obliczoną zgodnie ze wzorem

$$s_k = \frac{\nabla E(\mathbf{w}_k + \sigma_k \mathbf{p}_k) - \nabla E(\mathbf{w}_k)}{\sigma_k} + \lambda_k \mathbf{p}_k$$

dla parametru regularyzacji równego λ_k.

Analogicznie parametr δ_k odpowiadający wartości $\lambda_k^{(r)}$ obliczany jest ze wzoru:

$$\delta_k^{(r)} = \delta_k + \mathbf{p}_k^T\left(\lambda_k^{(r)} - \lambda_k\right)\mathbf{p}_k = \delta_k + \left(\lambda_k^{(r)} - \lambda_k\right)|\mathbf{p}_k|^2$$

Ze wzoru tego między innymi wynika, że

$$\lambda_k^{(r)} > \lambda_k - \frac{\delta_k}{|\mathbf{p}_k|^2}$$

Zależność ta wskazuje kierunek zmian dla parametru λ_k zapewniający poprawne działanie algorytmu. Moller w związku z tym zaproponował następującą regułę:

$$\lambda_k^{(r)} = 2\left(\lambda_k - \frac{\delta_k}{|\mathbf{p}_k|^2}\right)$$

Przyjmując regułę Mollera dla wyznaczenia wartości $\lambda_k^{(r)}$ otrzymuje się formułę pozwalającą na wyznaczanie wartości parametru δ_k po regularyzacji:

$$\delta_k^{(r)} = \delta_k + \left(2\lambda_k - 2\frac{\delta_k}{|\mathbf{p}_k|^2} - \lambda_k\right)|\mathbf{p}_k|^2 = -\delta_k + \lambda_k|\mathbf{p}_k|^2$$

a to pozwala ustalić wartość parametru η

$$\eta = \frac{\mu_k}{\delta_k^{(r)}} = \frac{\mu_k}{-\mathbf{p}_k^T\mathbf{s}_k + \lambda_k|\mathbf{p}_k|^2}$$

Z zależności powyższej wynika, że zwiększenie wartości parametru regularyzacji λ_k powoduje zmniejszenie wartości η, a tym samym zmniejszenie kroku w kierunku $\mathbf{p}_k$.
Jest to taktyka rozsądnie ostrożna.

Inna rozsądnie ostrożna taktyka nakazuje przyjąć metodę doboru wartości parametru λ_k, która oparta jest na wartości współczynnika Δ_k danego wzorem:

$$\Delta_k = \frac{E(\mathbf{w}_k) - E(\mathbf{w}_k + \eta\mathbf{p}_k)}{E(\mathbf{w}_k) - E_a(\eta\mathbf{p}_k)}$$

gdzie $E_a(\eta\mathbf{p}_k)$ jest aproksymowaną wartością funkcji celu liczoną po zastąpieniu hesjanu jego wartością zregularyzowaną.

Wzór dla obliczania wartości współczynnika Δ_k można też zapisać:

$$\Delta_k = \frac{2\delta_k^{(r)}\left(E(\mathbf{w}_k) - E(\mathbf{w}_k + \eta\mathbf{p}_k)\right)}{\mu_k^{(r)2}}$$

Im wartość współczynnika Δ_k jest bliższa jedynki, tym mniejsze jest zniekształcenie wywołane regularyzacją

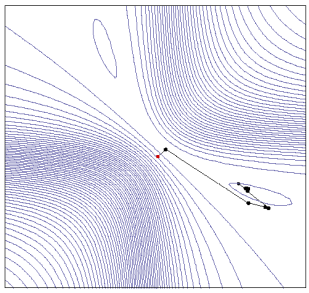
Zmiana wartości współczynnika λ_k odbywa się według następującej reguły:

- jeśli $\Delta_k \geq 0.75 \Rightarrow \lambda_k = 0.25\ \lambda_k$,
- jeśli $\Delta_k < 0.25 \Rightarrow \lambda_k = \lambda_k + \frac{\delta_k(1-\Delta_k)}{|\mathbf{p}_k|^2}$
- w pozostałych przypadkach λ_k pozostaje bez zmian.

Wszystkie przytoczone tu reguły należy traktować jako heurystyki, które w razie potrzeby mogą zostać zmienione!

Tak naprawdę w dziedzinie metody gradientów sprzężonych istotne jest to, że metoda ta odwołuje się do **drugich pochodnych funkcji błędu,** (gdyż obliczane lub aproksymowane są hesjany) podczas gdy klasyczna ***backpropgation*** oraz wszystkie jej modyfikacje odwołują się do pierwszych pochodnych (gradientów) funkcji błędów.

Jako metoda drugiego rzędu technika **gradientów sprzężonych** działa oczywiście szybciej, niż metody oparte na pierwszych pochodnych:



22 iterations

Główne zalety metody gradientu sprzężonego są następujące:

- ze względu na to, że dla funkcji n zmiennych istnieje **nie więcej** niż n sprzężonych kierunków, proces uczenia zawsze zatrzyma się wykonawszy najwyżej n kroków (gdzie n jest liczbą wyznaczanych wag)
- osiągnięta w kolejnym kroku algorytmu wartość błędu E^t **nigdy** nie jest wyższa od wartości błędu w kroku poprzednim E^{t-1},
- metoda gradientu sprzężonego nie posiada parametrów, których wartość musiałaby być ustalona przez użytkownika, a które wpływałyby istotnie na szybkość działania i osiągnięty efekt końcowy, jest więc łatwa i prosta w stosowaniu.

W trakcie realizacji omawianych dotychczas metod uczenia sieci wykonuje się kolejne przesunięcia po powierzchni błędu mające prowadzić do punktu położonego najniżej. Zakłada się, że minimum zostanie osiągnięte po wykonaniu pewnej skończonej (chociaż nieznanej z góry) liczby kroków, co realizujemy wieloetapowo zbliżając się do poszukiwanego punktu minimum poprzez stopniowe, ale systematyczne zmniejszanie wartości funkcji błędu.

Alternatywne podejście do zagadnienia minimalizacji funkcji błędu polega na próbie osiągnięcia położonego najniżej punktu powierzchni błędu **w jednym kroku** - co jest osiągalne przy założeniu, że **znany** jest kształt tej powierzchni.

Strategię taką stosuje się w *metodach opartych na modelu obszaru* (*model-trust region approach*).

Biorąc za punkt wyjścia pewien początkowy wektor wag $\mathbf{w_k}$ i zakładając paraboloidalny kształt minimalizowanej funkcji błędu, minimum tej funkcji osiągnąć można w jednym kroku przesuwając się zgodnie z formułą:

$$\mathbf{w} = \mathbf{w_0} - \mathbf{H}^{-1}\nabla E(\mathbf{w_0})$$

Formuła przytoczona wyżej wynika z następującego rozumowania:

Przyjmując, że wektor $\mathbf{w}$ jest wektorem wszystkich wag (w całej sieci) można wyrazić wartość gradientu funkcji kryterium w otoczeniu pewnego punktu $\mathbf{w_0}$ w postaci szeregu Taylora:

$$\nabla E(\mathbf{w}) = \nabla E(\mathbf{w_0}) + \mathbf{H}(\mathbf{w} - \mathbf{w_0}) + \ldots$$

gdzie $\boldsymbol{H}$ jest macierzą drugich pochodnych (hesjanem) funkcji błędu E. Dalszych wyrazów szeregu nie uwzględniamy, zakładając, że funkcja błędu ma kształt funkcji kwadratowej (paraboloidy).

W punkcie $\mathbf{w}$, w którym funkcja $E(\boldsymbol{w})$ osiąga minimum wartość gradientu wynosi 0, przeto właśnie

$$\mathbf{w} = \mathbf{w_0} - \mathbf{H}^{-1}\nabla E(\mathbf{w_0})$$

W praktyce funkcja błędu prawie nigdy nie ma kształtu paraboloidy, dlatego zwykle nie udaje się trafić w punkt rzeczywistego minimum „jednym strzałem”

Oznacza to konieczność iterowania rozwiązania zgodnie ze wzorem:

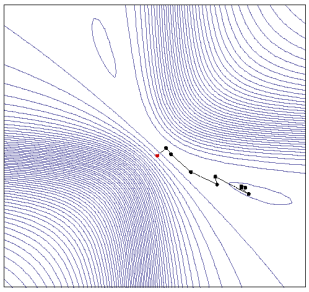
$$\mathbf{w^t} = \mathbf{w^{t-1}} - \eta\mathbf{H}^{-1}\nabla E(\mathbf{w^{t-1}})$$

Metoda ta jest bardzo kosztowna numerycznie, ponieważ w każdym kroku stosowania algorytmu należy odwrócić macierz drugich pochodnych.

Jak wiadomo odwrócenie macierzy o wymiarach n x n wymaga około n^3 działań.

Uzyskany wynik nazywany jest wyznaczaniem minimum funkcji *metodą Newtona*.
Metoda ta pozwala na osiągnięcie w jednym kroku minimum funkcji błędu pod warunkiem, że funkcja ta ma kształt paraboloidalny.
Niestety, w ogólnym przypadku minimalizowana w trakcie uczenia sieci neuronowej funkcja błędu odbiega swym kształtem od zakładanej paraboloidy, co powoduje, że wyznaczenie jej minimum w jednym kroku nie jest możliwe.

Metoda Newtona



11 iterations

W opisanej metodzie wymagane jest obliczanie macierzy drugich pochodnych, co bywa trudne.

Dlatego metoda Newtona bywa **niestabilna numerycznie**, zwłaszcza w przypadku, gdy punkt startowy nie jest położony dostatecznie blisko poszukiwanego rozwiązania.

Rozwiązaniem tych problemów może być zastosowanie *metody Levenberga-Marquardta*.

W pobliżu minimum posiada ona cechy metody Newtona, zaś wraz z oddalaniem się punktu pracy od poszukowanego minimum sposób jej działania zbliża się do metody najszybszego spadku.

Wzór opisujący ten algorytm wygląda następująco:

$$\mathbf{w}^{t} = \mathbf{w}^{t-1} - (\mathbf{H} + \mu\mathbf{I})^{-1}\nabla E(\mathbf{w}^{t-1})$$

W zapisie formalnym różnica pomiędzy formułą Newtona i Levenberga-Marquardta jest niewielka.

Polega ona wyłącznie na tym, że w tej drugiej metodzie do hesjanu dodawana jest dodatkowo macierz jednostkowa przemnożona przez pewien dodatni współczynnik μ, którego wartość modyfikowana jest w trakcie uczenia.

$$\mathbf{w}^{t} = \mathbf{w}^{t-1} - (\mathbf{H} + \mu\mathbf{I})^{-1}\nabla E(\mathbf{w}^{t-1})$$

Spróbujmy przeanalizować wpływ wartości μ na sposób działania algorytmu.

Jeśli μ zbliża się do zera, to wektor wag uzyskany przy pomocy metody Levenberga-Marquardta będzie bardzo zbliżony do wartości wyznaczonych przy zastosowaniu prostej formuły Newtona.

Zwiększaniu wartości μ towarzyszy wzrost znaczenia kierunku poprawy wyznaczonego w oparciu o gradient funkcji błędu.

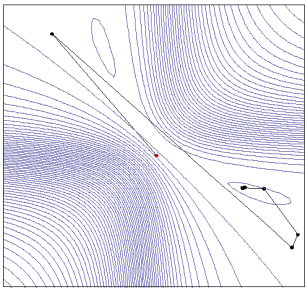
Warto zauważyć, że μ nie tylko spełnia funkcję „współczynnika wagowego", określającego udział wspomnianych dwóch strategii w wyznaczeniu rzeczywistego kierunku zmian, ale również wyznacza wielkość kroku wykonywanego w kierunku poprawy bazującym na gradiencie - im większa wartość μ, to tym mniejszy jest ten krok, a to jest korzystne.

W typowych przypadkach **algorytm Levenberga-Marquardta jest najszybszym algorytmem uczenia** wszystkich sieci jednokierunkowych (typu *feedforward*). Niestety posiada on kilka ważnych ograniczeń. Wśród nich można wymienić następujące:

- Algorytm ten może być użyty wyłącznie dla sieci z pojedynczym neuronem wyjściowym i funkcją błędu w postaci sumy kwadratów odchyleń. Algorytm ten nie nadaje się zatem do uczenia sieci o wielu wyjściach lub takich sieci, których cel działania nie daje się wyrazić w formie błędu średniokwadratowego.

- Wymagania rozważanego algorytmu w zakresie pamięci są bardzo duże. Precyzyjnie mówiąc, wymagania pamięciowe algorytmu są proporcjonalne do W^2, gdzie W jest liczbą parametrów (współczynników wagowych) występujących w całej sieci. Powoduje to, że jest on niepraktyczny w przypadku dużych sieci, zawierających dużą liczbę współczynników wag.

Algorytm Levenberga-Marquardta



7 iterations

Kolejna grupa metod uczenia odwołuje się do zmiennej metryki

Podstawowa metoda z tej grupy jest zbliżona do metody Newtona z tą różnicą, że występującą w tej metodzie odwrotność hesjanu zastępuje się jej przybliżeniem $\mathbf{V}$

Modyfikacja wag dokonywana jest zgodnie ze wzorem:

$$\mathbf{w}^t = \mathbf{w}^{t-1} - \eta \mathbf{V}^t \nabla E(\mathbf{w}^{t-1})$$

gdzie macierz $\mathbf{V}^t$ jest przybliżeniem macierzy $\mathbf{H}^{-1}$ w trakcie realizacji t - tego kroku algorytmu.

Wzór opisujący sposób obliczania macierzy $\mathbf{V}^t$ ma charakter **rekurencyjny**, to znaczy macierz $\mathbf{V}^t$ wyrażana jest jako funkcja macierzy $\mathbf{V}^{t-1}$. W pierwszej iteracji przyjmuje się, że macierz $\mathbf{V}^0 = \mathbf{I}$. Dalej poszczególne metody różnią się między sobą sposobem wyznaczania przybliżenia $\mathbf{V}^t$.

W podanym niżej opisie poszczególnych metod przyjęto następujące pomocnicze oznaczenia:

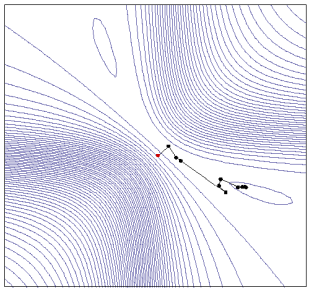
$$\mathbf{s}^t = \mathbf{w}^t - \mathbf{w}^{t-1}$$

$$\mathbf{r}^t = \nabla E\left(\mathbf{w}^t\right) - \nabla E\left(\mathbf{w}^{t-1}\right)$$

Przy zastosowaniu tych oznaczeń najbardziej znaną metodę zmiennej metryki: Davidona - Fletchera - Powella **(DFP)** można wyrazić wzorem:

$$\mathbf{V}^t = \mathbf{V}^{t-1} + \frac{\mathbf{s}^t\left(\mathbf{s}^t\right)^T}{\left(\mathbf{s}^t\right)^T\mathbf{r}^t} - \frac{\mathbf{V}^{t-1}\mathbf{r}^t\left(\mathbf{r}^t\right)^T\mathbf{V}^{t-1}}{\left(\mathbf{r}^t\right)^T\mathbf{V}^{t-1}\mathbf{r}^t}$$

Metoda Davidona - Fletchera - Powella



11 iterations

Są oczywiście także inne metody:

Dosyć popularna jest metoda Broydena - Fletchera - Goldfarba - Shanno (BFGS), który zakłada użycie wzoru:

$$\mathbf{V}^t = \mathbf{V}^{t-1} + \left(1 + \frac{(\mathbf{r}^t)^T \mathbf{V}^{t-1} \mathbf{r}^t}{(\mathbf{s}^t)^T \mathbf{r}^t}\right) \frac{\mathbf{s}^t (\mathbf{s}^t)^T}{(\mathbf{s}^t)^T \mathbf{r}^t} - \frac{\mathbf{s}^t (\mathbf{r}^t)^T \mathbf{V}^{t-1} + \mathbf{V}^{t-1} \mathbf{r}^t (\mathbf{s}^t)^T}{(\mathbf{s}^t)^T \mathbf{r}^t}$$

Popularna jest też metoda Wolfe'a - Broydena - Davidona (WBD), która korzysta ze wzoru

$$\mathbf{V}^t = \mathbf{V}^{t-1} + \frac{(\mathbf{s}^t - \mathbf{V}^{t-1} \square \mathbf{r}^t)(\mathbf{s}^t - \mathbf{V}^{t-1} \square \mathbf{r}^t)^T}{(\mathbf{s}^t - \mathbf{V}^{t-1} \square \mathbf{r}^t)^T \mathbf{r}^t}$$

Stosowane są też strategie mieszane, na przykład aktualizacja wag zgodnie z algorytmem WBD/BFGS jest przeprowadzana w zależności od potrzeb w sposób zgodny z metodą WBD lub BFGS.

Wybór odpowiedniego sposobu postępowania jest uzależniony od spełnienia następującego warunku:

$$(\mathbf{s}^t - \mathbf{V}^{t-1} \mathbf{r}^t)^T \mathbf{r}^t > 0$$

Jeżeli warunek ten jest spełniony, to kolejne przybliżenie macierzy $\mathbf{V}^t$ jest obliczane zgodnie z regułą WBD, w przeciwnym przypadku macierz $\mathbf{V}^t$ obliczana jest zgodnie z regułą BFGS

Istotną cechą wszystkich omówionych wyżej algorytmów zmiennej metryki jest tak zwana ***odnowa***, polegająca na wykonaniu podstawienia $\mathbf{V}^t = \mathbf{I}$. Odnowę przeprowadza się wówczas, gdy wyznaczony kierunek zmian

$$-\mathbf{V}^t \nabla E(\mathbf{w}^{t-1})$$

nie jest kierunkiem poprawy, jak również wtedy, gdy liczba iteracji od czasu ostatniej odnowy przekroczyła wartość $2n+1$, gdzie n jest liczbą szacowanych parametrów sieci.

Współczynnik uczenia sieci η powinien być dobrany w taki sposób, aby zapewniał minimalizację funkcji błędu $E(\eta)$ w kierunku

$$-\mathbf{V}^t \nabla E(\mathbf{w}^{t-1})$$

W celu wyznaczenia odpowiedniej wartości współczynnika η można zastosować jedną z klasycznych metod minimalizacji w kierunku, na przykład *metodę złotego podziału* lub *metodę interpolacji kwadratowych*

Godne rozważenia są też metody całkowicie niezależne od wstecznej propagacji błędów

Opisywane wyżej metody każdorazowo odwoływały się do pojęcie wyboru kierunku zmian wag gwarantującego szybkie malenie funkcji błędu.

Czasami warto jednak sięgnąć do metod, które tego elementu w sposób jawny nie zawierają, a mimo to (a często właśnie dlatego) pozwalają na w miarę szybki postęp procesu uczenia.

Wśród tych metod na uwagę z pewnością zasługuje rekurencyjna metoda najmniejszych kwadratów (**RLS** - *Recursive Least Squares Algorithm*).

W trakcie uczenia jednokierunkowej sieci wielowarstwowej przy zastosowaniu algorytmu RLS wagi dobierane są w taki sposób, aby zminimalizować wartość funkcji kryterium o postaci:

$$Q(n) = \sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \varepsilon_j^{(L)^2}(t)$$

gdzie:

- t - indeks kolejnej iteracji algorytmu, $t = 1, ..., n$,
- $Q(n)$ - wartość funkcji kryterium po przeprowadzeniu n kroków uczenia,
- λ - dodatni współczynnik (tzw. stała zapominania) z przedziału (0, 1). Wyrażenie λ^{n-t} sprawia, że na wartość $Q(n)$ mają większy wpływ błędy uzyskane w późniejszych krokach algorytmu,
- L - liczba warstw w sieci,
- k - numer warstwy, $k = 1, ..., L$,
- N_k - liczba neuronów w warstwie k,
- $\varepsilon_i^{(k)}(n) = d_i^{(k)}(n) - y_i^{(k)}(n)$ - błąd sygnału wyjściowego dla i - tego neuronu warstwy k po przeprowadzeniu n - tej iteracji algorytmu, błąd ten liczony jest jako różnica pomiędzy wartością zakładaną (d) i wartością wyjściową (y) dla tego neuronu.

Warunkiem minimalizacji formuły

$$Q(n)=\sum_{t=1}^{n}\lambda^{n-t}\sum_{j=1}^{N_L}\varepsilon_j^{(L)^2}(t)$$

jest zerowanie się pierwszych pochodnych funkcji *Q(n)* liczonych względem wag neuronów poszczególnych warstw, czyli

$$\frac{\partial Q(n)}{\partial w_i^{(k)}(n)}=0$$

Po dokonaniu odpowiednich przekształceń równanie normalne zapisać można w postaci:

W przytoczonych wzorach istotną rolę odgrywa

$$b_i^{(k)}=f^{-1}\left(d_i^{(k)}\right)$$

sygnał wzorcowy części liniowej *i* - tego neuronu w warstwie *k*.

Ostateczną postać formuły obliczeniowej dla rozważanej tu metody uczenia sieci przedstawiają wzory podane na kolejnym slajdzie

$$\mathbf{w}_i^{(k)}(n)=\mathbf{w}_i^{(k)}(n-1)+\mathbf{g}_i^{(k)}(n)\varepsilon_i^{(k)}(n)$$

$$\varepsilon_i^{(k)}(n)=\begin{cases}d_i^{(L)}(n)-y_i^{(L)}(n) & dla \quad k=L\\ \sum_{j=1}^{N_{k+1}} f^{\copyright}\left(s_j^{(k+1)}(n)\right)w_{ji}^{(k+1)}(n)\varepsilon_j^{(k+1)}(n) & dla \quad k=1,\ldots,L-1\end{cases}$$

$$\mathbf{g}_i^{(k)}(n)=\frac{f^{\copyright}\left(s_i^{(k)}(n)\right)\mathbf{P}_i^{(k)}(n-1)\mathbf{x}^{(k)}(n)}{\lambda+f^{\copyright^2}\left(s_i^{(k)}(n)\right)\mathbf{x}^{(k)^T}(n)\mathbf{P}_i^{(k)}(n-1)\mathbf{x}^{(k)}(n)}$$

$$\mathbf{P}_i^{(k)}(n)=\lambda^{-1}\left[\mathbf{I}-f^{\copyright}\left(s_i^{(k)}(n)\right)\mathbf{g}_i^{(k)}(n)\mathbf{x}^{(k)^T}(n)\right]\mathbf{P}_i^{(k)}(n-1)$$

Przy prowadzeniu iteracyjnych obliczeń zmierzających do ustalenia wartości wag według wzoru

$$\mathbf{w}_i^{(k)}(n)=\mathbf{w}_i^{(k)}(n-1)+\mathbf{g}_i^{(k)}(n)\varepsilon_i^{(k)}(n)$$

w charakterze wartości początkowych przyjmuje się:

$$\mathbf{w}_i^{(k)}(0)=0$$

$$\mathbf{P}_i^{(k)}(0)=\delta\mathbf{I}$$

gdzie stała $\delta > 0$ jest dobierana do warunków zadania

$$\mathbf{r}_i^{(k)}(n)=\mathbf{R}_i^{(k)}(n)\mathbf{w}_i^{(k)}(n)$$

gdzie:

$$\mathbf{R}_i^{(k)}=\sum_{t=1}^{n}\lambda^{n-t}f^{\copyright^2}\left(s_i^{(k)}(t)\right)\mathbf{x}^{(k)}(t)\mathbf{x}^{(k)^T}(t)$$

$$\mathbf{r}_i^{(k)}(n)=\sum_{t=1}^{n}\lambda^{n-t}f^{\copyright^2}\left(s_i^{(k)}(t)\right)b_i^{(k)}(t)\mathbf{x}^{(k)}(t)$$

f - funkcja aktywacji neuronu,

$s_i^{(k)}(t)$ - suma ważonych wejść dla *i* - tego neuronu warstwy *k*,

$\mathbf{x}^{(k)}$ - wektor sygnałów wejściowych dla neuronów warstwy *k*,

Jak wiemy, ciągłą funkcję błędu *E*, mającą w punkcie $\mathbf{w_k}$ dobrze określone wszystkie pochodne przedstawić można w postaci sumy szeregu potęgowego. Wyraża to ogólnie znany *wzór Taylora*.

Z uwagi na postulowany paraboloidalny kształt funkcji pominiemy w tym wzorze wyrazy szeregu rzędu wyższego niż dwa. Prowadzi to do formuły:

$$E(w^{k+1}) = E(w^k) + \nabla E(w^k)(w^{k+1} - w^k) + \frac{1}{2}(w^{k+1} - w^k)^T H(w^k)(w^{k+1} - w^k)$$

Jeśli w punkcie $\mathbf{w_{k+1}}$ funkcja ma osiągnąć minimum, to wektor pierwszych pochodnych wyznaczony w tym punkcie powinien być wektorem zerowym. Obliczmy więc $E'(w_{k+1})$ i przyrównajmy otrzymane wyrażenie do zera.
Następnie z uzyskanego równania wyznaczmy wektor $\mathbf{w_{k+1}}$.