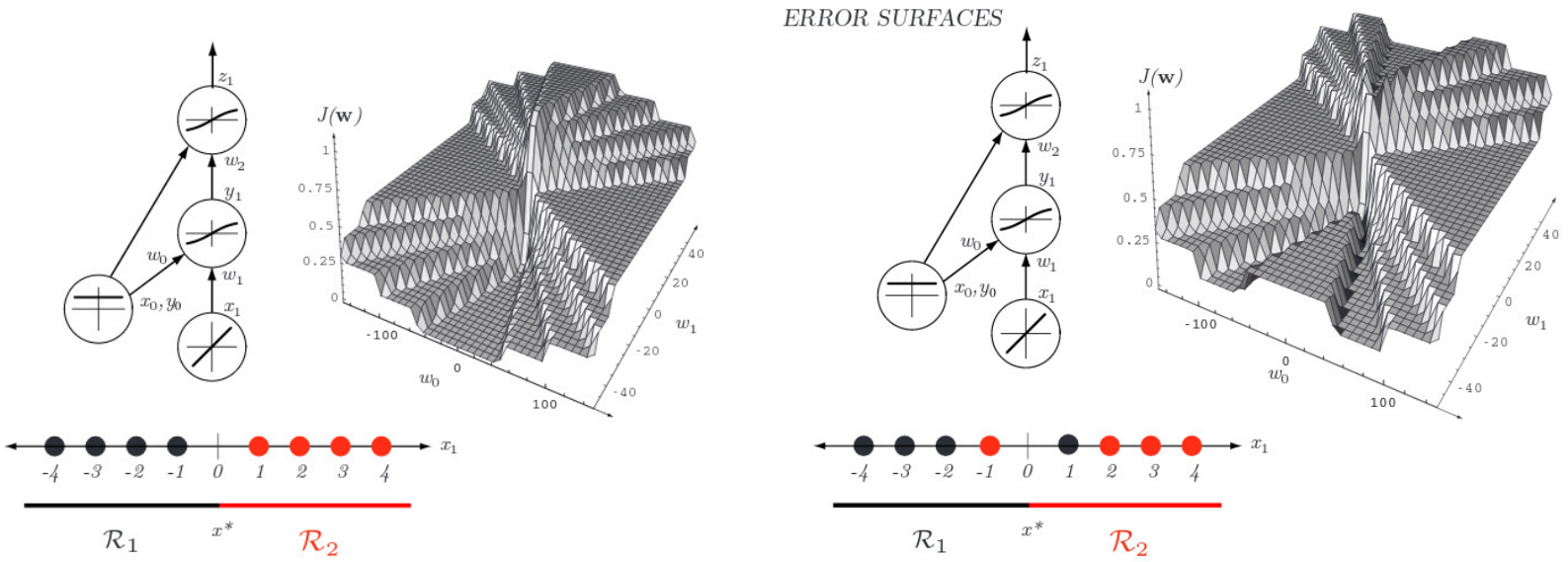
# Ulepszenia treningu sieci MLP

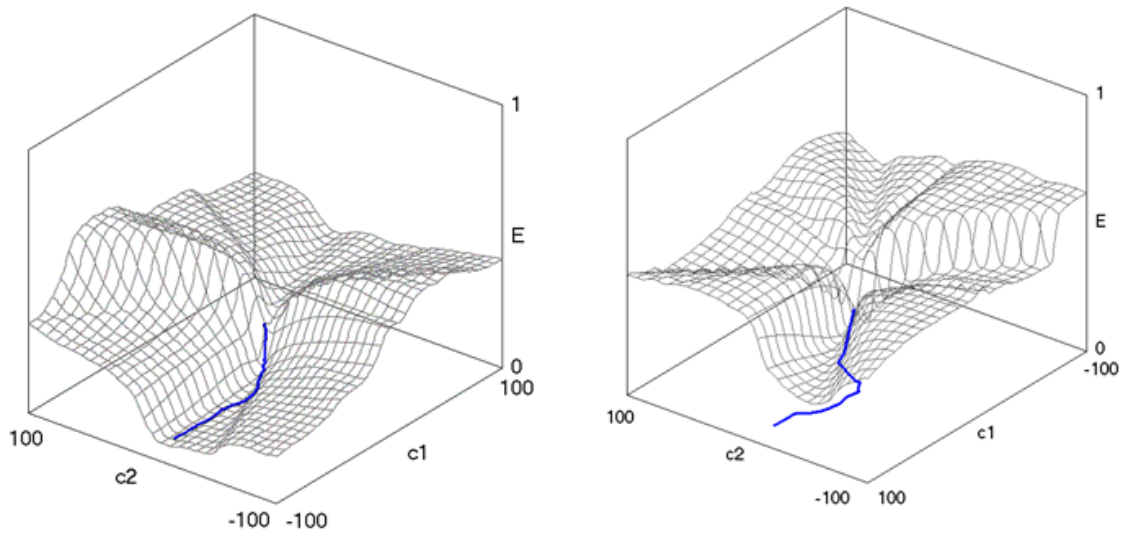
# Powierzchnia błędu




- minima lokalne i minimum globalne
- „wzgórza” i„doliny” odpowiadające lokalnym optimom
- płaskie regiony, płaskowyże (*plateau*), regiony o małej zmienności błędu względem wag
- nierównomierna krzywizna, szybkie zmiany w jednych kierunkach, wolne w innych
- wysoka wymiarowość, miliony parametrów w sieciach neuronowych

# Trajektorie zbieżności

PCA na macierzy kowariancji wag w kolejnych krokach iteracji




- wiele płaskowyżów i kanionów.
- dane leżące daleko od granicy mają mały wpływ na powierzchnie błędu
- przy końcu uczenia główna redukcja błędu wynika ze wzrostu wartości wag, czyli wyostrzania się sigmoid aż zrobią się prawie skokowe

# Jak omijać minima lokalne?

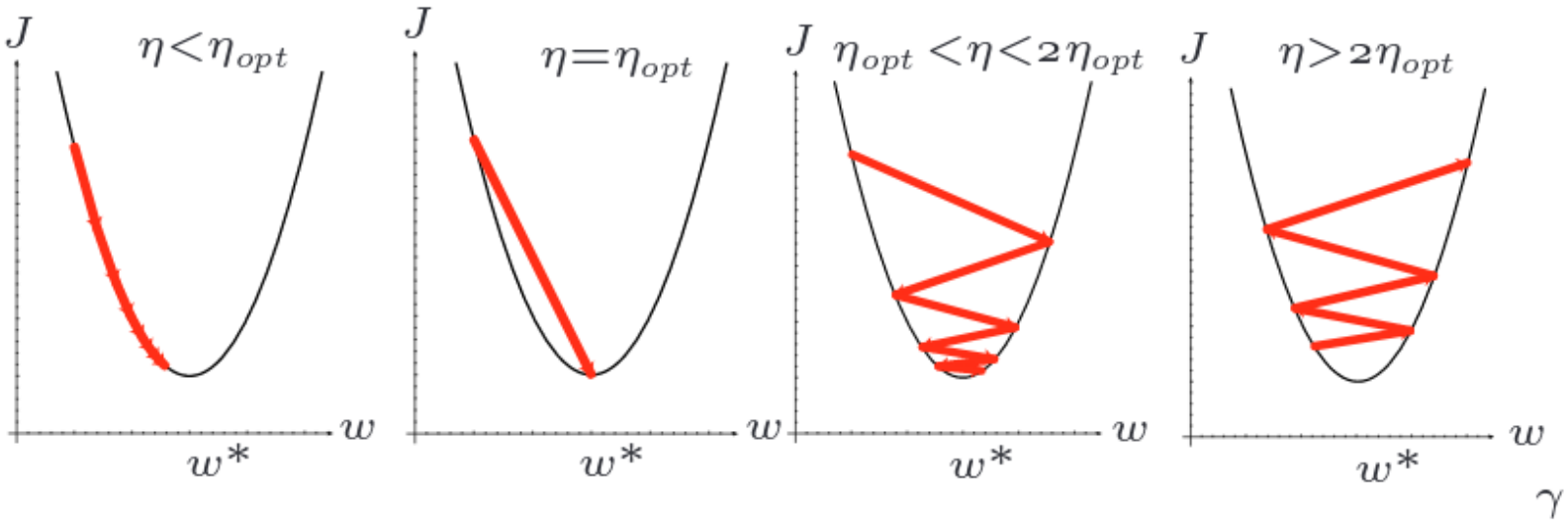
- **losowa inicjalizacja**: rozrzucanie punktów startowych w różnych regionach przestrzeni wag
- **wielokrotny start** z różnymi wartościami początkowymi - najprostsza ale skuteczna metoda
- szum dodawany do wag lub szum dodany do danych pozwala wygładzić funkcję błędu i uciec z płytszych minimów – formalnie jest to równoważne **regularyzacji**, czyli dodaniu dodatkowego członu wygładzającego do funkcji błędu
- losowa kolejność prezentowania przypadków
- modyfikacje algorytmów gradientowych (np. momentum)
- metody optymalizacji globalnej

# Metody globalnej optymalizacji

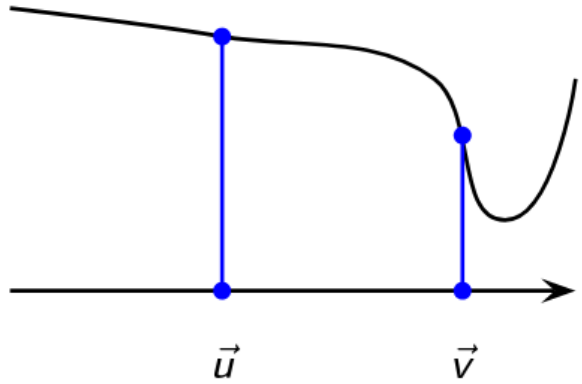
- Metody globalnej optymalizacji: wiele metod, opartych na przeszukiwaniu przestrzeni wag
- Monte Carlo, symulowane wyżarzanie, metody multisympleksowe, optymalizacja rojem cząstek (PSO), algorytmy genetyczne, minimalizacja Tabu, homotopia ...
- Dużo prac łączących algorytmy genetyczne z sieciami MLP
- Zalety: globalne, proste w realizacji, niektóre nie potrzebują gradientu, inne łączą zalety metod gradientowych z metodami globalnymi
- Wady: zwykle kosztowne obliczeniowo i czasochłonne

# Dobór współczynnika uczenia $\eta$

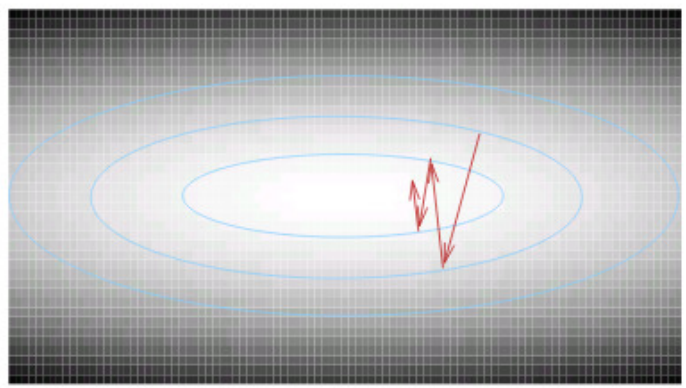
- zbyt mała wartość $\eta \rightarrow$ powolna zbieżność
- za duża wartość $\eta \rightarrow$ niestabilny proces uczenia, oscylacje
- wartość kroku może być zmieniana w czasie uczenia, różne podejścia:
  - zmniejszana w czasie treningu
  - zwiększana dla płaskich powierzchni błędu
  - dobierana indywidualnie dla każdej warstwy lub pojedynczych neuronów i wag



# Płaskie powierzchnie i strome zbocza dolin




Powierzchnia błędu w wielu wymiarach ma różne nachylenie wzdłuż różnych kierunków



*Rys: Riedmiller, Machine Learning*

# Trening z momentem

- prędkość uczenia zależna od poprzednich wartości gradientów

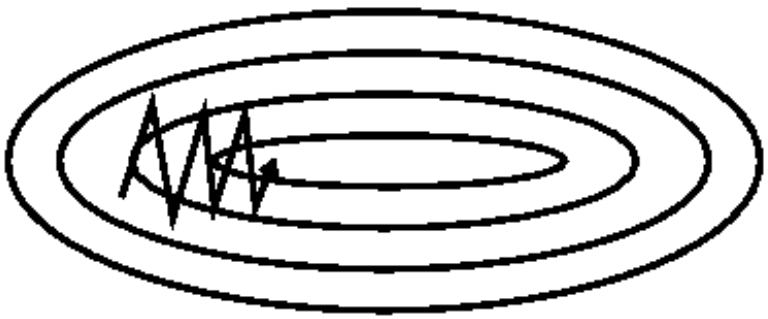
$$\Delta w(t) = -\eta \nabla E(t) + \gamma \Delta w(t-1)$$

  gdzie $0 < \gamma < 1$, typowo $\gamma = 0.9$
- czynnik *momentum* zwiększa krok uczenia, gdy gradient nie zmienia kierunku
- przyśpieszenie uczenia na płaskich odcinkach (gdzie $\nabla E$ się nie zmienia) wynosi w przybliżeniu

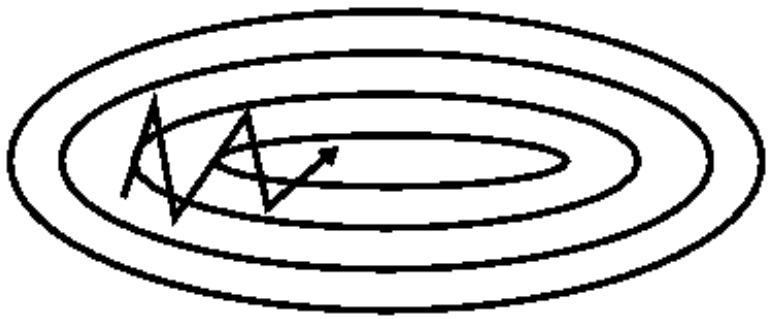
$$\Delta w(t) = -\frac{\eta}{1-\gamma} \nabla E$$

- redukuje oscylacje (spowalnia uczenie), gdy gradient zmienia kierunek
- gdy $\nabla E = 0$ wagi są nadal modyfikowane (bezwładność uczenia), może to ułatwić opuszczenie regionu „przyciągania” do minimum lokalnego

# Trening z momentem



SGD bez momentu



SGD z momentem

# Inicjalizacja wag

- Losowe wartości z rozkładu jednostajnego w okolicach 0
- Za duże wagi powodują duże wartości aktywacji i ryzyko nasycenia funkcji sigmoidalnych
- Dla $d$ wejść można wybrać wartości z zakresu

$$-\frac{1}{\sqrt{d}} < w_{ij} < \frac{1}{\sqrt{d}}$$

  co przy standaryzacji danych wejściowych daje uśrednioną aktywację neuronów w zakresie liniowym $[-1, 1]$ sigmoidy
- Analogicznie można inicjować wagi w kolejnych warstwach

# Skalowanie wartości wejściowych

- Odmienne zakresy wartości mierzonych cech $x_i$ (różne skale i jednostki), rozkłady wartości zmiennych dalekie od rozkładu normalnego, wartości odstające - mogą negatywnie wpływac na proces uczenia
- Dane wejściowe $\mathbf{x}_i$ powinny posiadać zbliżone zakresy wartości aby uniknąć problemów zbieżności
- **Standaryzacja**

$$x_s = \frac{x - \mu}{\sigma}$$

  $\mu$ - wartość średnia, $\sigma$ - odchylenie standardowe
- **Normalizacja** w zakresie $[-1, +1]$

$$x_n = 2\frac{x - x_{min}}{x_{max} - x_{min}} - 1$$

# RPROP

**Resilent BP** (Riedmiller, Braun, 1992)

- heurystyczny algorytm uczenia, który ma przyśpieszać proces uczenia i być odporny na problemem zanikającego (oraz eksplodującego) gradientu
- aktualizacja wag $\Delta w_{ij}(t)$ zależy od znaku gradientu a nie od wartości
- współczynnik uczenia $\eta_{ij}$ jest dobierany dla każdej wagi $w_{ij}$ niezależnie, jego wartość zmienia się w czasie uczenia

$$\Delta w_{ij}(t) = -\eta_{ij}(t) \cdot \text{sgn}\left(\frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}}\right)$$

# RPROP

- aktualizacja współczynnika zależy od zmian znaku gradient w dwóch ostatnich krokach $t, t-1$
- $\eta_{ij}$ rośnie, gdy brak zmiany znaku gradientu
- $\eta_{ij}$ maleje, gdy następuje zmiana znaku gradientu

$$\Delta w_{ij}(t) = -\eta_{ij}(t) \cdot \text{sgn}\left(\frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}}\right)$$

gdzie

$$\eta_{ij}(t) = \begin{cases} \min(a \cdot \eta_{ij}(t-1), \eta_{max}) & \text{dla} \quad \frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}} \frac{\partial E(\mathbf{w}(t-1))}{\partial w_{ij}} > 0 \\ \max(b \cdot \eta_{ij}(t-1), \eta_{min}) & \text{dla} \quad \frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}} \frac{\partial E(\mathbf{w}(t-1))}{\partial w_{ij}} < 0 \\ \eta_{ij}(t-1) & \text{w pozostałych przypadkach} \end{cases}$$

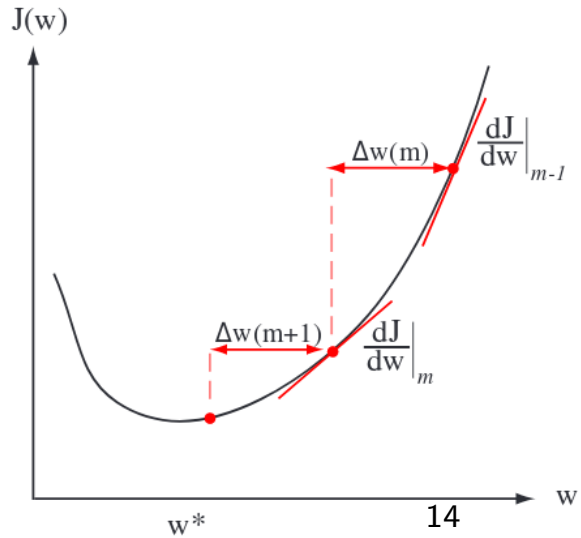
$a = 1.2, \quad b = 0.5, \quad \eta_{min} = 10^{-6}, \quad \eta_{max} = 50$

# Quickprop (Fahlman, 1988)

- założenie: funkcja błędu lokalnie jest kwadratowa i może być przybliżona parabolą
- przybliżenie położenia wierzchołka paraboli za pomocą wartości wag i gradientów (nachylenia) funkcji kosztu w 2 punktach $w(t)$ i $w(t-1)$

$$\Delta w_{ij}(t) = \frac{\nabla_{ij} E(t)}{\nabla_{ij} E(t-1) - \nabla_{ij} E(t)} \Delta w(t-1)$$

- zbieżność kwadratowa (szybciej niż liniowa metoda największego spadku)
- trening może być niestabilny z powodu dużych kroków, założenia metody mogą być spełnione tylko w pobliżu minimum



# Metody 2 rzędu

Rozwinięcie w szereg Taylora funkcji kosztu w sąsiedztwie $\mathbf{w}$ w kierunku $\Delta\mathbf{w}$:

$$E(\mathbf{w}+\Delta\mathbf{w}) \approx E(\mathbf{w}) + \nabla E(\mathbf{w})^T \Delta\mathbf{w} + \frac{1}{2}\Delta\mathbf{w}^T \nabla^2 E(\mathbf{w}) \Delta\mathbf{w}$$

gdzie $\nabla^2 E(\mathbf{w}) = \mathbf{H}$ jest macierzą Hessego (taw. hesjan) zawierającą $n \times n$ pochodnych 2 rzędu

$$H_{ij} = \frac{\partial^2 E(\mathbf{x};\mathbf{w})}{\partial w_i \partial w_j}$$

stąd gradient funkcji kosztu:

$$\nabla E(\mathbf{w}+\Delta\mathbf{w})^T \approx \nabla E(\mathbf{w})^T + \Delta\mathbf{w}^T \mathbf{H}$$

Minimum funkcji wymaga aby $\nabla E(\mathbf{w}+\Delta\mathbf{w}) = 0$, stąd

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

rozwiązanie w jednym kroku, jeżeli potrafimy obliczyć $\mathbf{H}^{-1}$

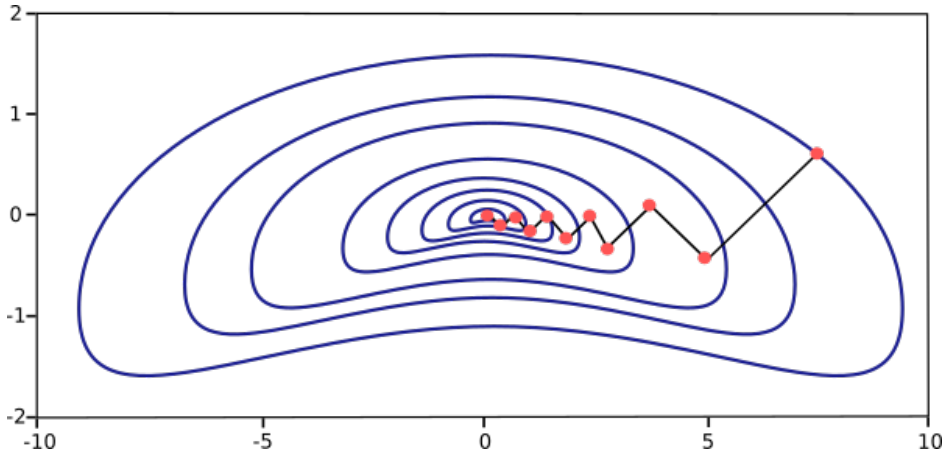
# Metoda Newtona

Iteracyjna metoda 2 rzędu:

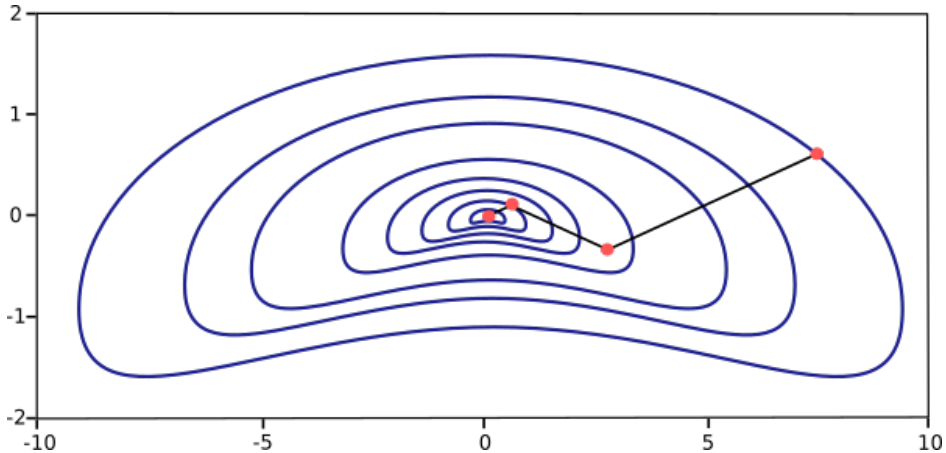
$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mathbf{H}^{-1}\nabla E(\mathbf{w}(t))$$

- metoda kosztowna czasowo $O(n^3)$, wymaga odwracania macierzy w każdej iteracji
- zbieżność kwadratowa, szybciej niż metoda gradientu prostego
- kosztowna pamięciowo, hesjan $O(n^2)$, gdzie $n$ to liczba wag
- praktyczne zastosowania tylko dla małych sieci
- hesjan $\mathbf{H}$ musi pozostać dodatnio okreslony aby zapewnić zbieżność do minimum i bezpieczne odwracanie macierzy
- w praktyce nierealne jest zapewnienie dodatniej określoności hesjanu w każdym kroku iteracji, dlatego stosuje się metody przybliżone

## Spadek gradientu




## Metoda Newtona




*Rys: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network*

# Metody quasi-Newtona

Przybliżenia hesjanu

- zaniedbanie pozadiagonalnych elementów

$$\Delta w_i = -\frac{\partial E}{\partial w_i} \Big/ \frac{\partial^2 E}{\partial w_i^2}$$

- metoda **zmiennej metryki** - iteracyjne przybliżenie $\mathbf{H}^{-1}$, kwadratowo zbieżna, dwie główne odmiany:
  - Davidon-Fletcher-Power (DFP)
  - Broyden-Fletcher-Goldfarb-Shanno (BFGS)
- metoda **Levenberg-Marquardta** oparta na przybliżeniu Gaussa-Newtona

# Metoda Levenberg-Marquardta

Metoda znajduje rozwiązanie zadań optymalizacji, które daje się sprowadzić do postaci

$$E = \sum_{i=1}^{p} e_i^2$$

Dla $n$ przypadków uczących i sieci posiadającej $m$ neuronów wyjściowych funkcja kosztu MSE zawiera sumę $p = n \cdot m$ czynników rezydualnych $e_i$:

$$E = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} (f_j(\mathbf{x}_i) - y_{ij})^2$$

Niech $\mathbf{e} = [e_1, e_2, \ldots, e_p]$ tworzy **wektor rezydualny** zaś **J** to macierz Jacobiego (**jakobian**) zawierająca pochodne cząstkowe wartości rezydualnych względem wag:

$$J_{ij} = \frac{\partial e_i}{\partial w_j}, \qquad i = 1, \ldots, p \qquad j = 1, \ldots, k$$

gdzie $k$ to ilość wszystkich wag.

# Metoda Levenberg-Marquardta

Gradient i hesjan funkcji błędu można wyrazić wówczas w postaci:

$$\nabla E = 2\mathbf{J}^T\mathbf{e} \qquad\qquad \mathbf{H} = 2\mathbf{J}^T\mathbf{J}$$

Metoda zastępuje hesjan w metodzie Newtona przybliżeniem, które zawiera czynnik regularyzacyjny zapewniający dodatnią określoność hesjanu:

$$\mathbf{H} \approx 2\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}$$

gdzie parametr tłumienia $\mu > 0$ jest dostosowywany w czasie uczenia.

Reguła aktualizacji wag:

$$\Delta\mathbf{w} = -2\left(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}\right)^{-1}\mathbf{J}^T\mathbf{e}$$

# Metoda Levenberg-Marquardta

$$\Delta \mathbf{w} = -2 \left( \mathbf{J}^T \mathbf{J} + \mu \mathbf{I} \right)^{-1} \mathbf{J}^T \mathbf{e}$$

- dla dużych $\mu$ czynnik kwadratowy zanika i uzyskujemy metodę gradeirntu prostego
- dla $\mu = 0$ mamy metodę Newtona o kwadratowej zbieżności
- algorytm uczenia startuje z dużą wartością $\mu$ a następnie w trakcie uczenia, w miarę zbliżania się do rozwiązania, $\mu$ jest zmniejszane
- bardzo szybko zbieżna metoda dla funkcji, które są sumą kwadratów (MSE)
- nie nadaje się do zastosowań z funkcją Cross Entropy
- nie praktyczne dla dużych danych, duży koszt pamięci, Jakobian ma wymiar $p \times k$, gdzie $p = n \cdot m$ - ilość wektorów razy ilość wyjść, $k$ - ilość parametrów (wag)

# Metoda Levenberg-Marquardta

Algorytm doboru współczynnika $\mu$ w $i$-tym kroku, dla danego $\mathbf{w}(i)$:

1. oblicz wartość $\mathbf{w}(i+1)$ zgodnie z regułą uczenia LM
2. oblicz wartość błędu w punkcie $\mathbf{w}(i+1)$
3. jeśli błąd wzrósł, wróć do poprzedniej wartości $\mathbf{w}(i)$ i zwiększ wartość $\mu$ $k$-krotnie i wróć do kroku 1 (przybliżenie liniowe minimalizowanej funkcji w otoczeniu $\mathbf{w}(i)$ okazało się nie dość ścisłe, więc zwiększamy „wpływ" metody gradientu prostego)
4. jeśli błąd zmalał, zaakceptuj ten krok i zmniejsz wartość $\mu$ $k$-krotnie (założenie o liniowości minimalizowanej funkcji w otoczeniu $\mathbf{w}(i)$ okazało się wystarczająco ścisłe, więc zwiększamy „wpływ" metody Gaussa-Newtona).

Typowo $k = 10$.

# Metoda gradientów sprzężonych

Metoda **gradientów sprzężonych** (*conjugated gradients*) poszukuje minimum wzdłuż kierunku $\Delta\mathbf{w}(m)$ ortogonalnego i sprzężonego do wszystkich kierunków z poprzednich iteracji.

Kierunki $\Delta\mathbf{w}(m)$ i $\Delta\mathbf{w}(m-1)$ są sprzężone gdy:

$$\Delta\mathbf{w}^T(m-1)\mathbf{H}\Delta\mathbf{w}(m) = 0$$

Kierunek wyszukiwania minimum w iteracji $m$ wyznaczamy z reguły

$$\Delta\mathbf{w}(m) = -\nabla E(m) + \beta(m)\Delta\mathbf{w}(m-1)$$

gdzie **współczynnik sprzężenia** $\beta$ kumuluje informacje o kierunkach z poprzednich iteracji $m-1, ..., 1$, zapewniając warunek sprzężenia.

# Metoda gradientów sprzężonych

Przykładowe postaci współczynnika sprzężenia:

- reguła Fletchera-Reevesa

$$\beta(m) = \frac{\nabla E(m)^T \nabla E(m)}{\nabla E(m-1)^T \nabla E(m-1)}$$

- reguła Polaka-Ribiera

$$\beta(m) = \frac{\nabla E(m)^T (\nabla E(m) - \nabla E(m-1))}{\nabla E(m-1)^T \nabla E(m-1)}$$

# Algorytm metody gradientów sprzężonych

**Pierwszy kierunek**

1. zainicjuj $\mathbf{w}(0)$ i znajdź pierwszy kierunek $\Delta\mathbf{w}(0)$ metodą największego spadku $\Delta\mathbf{w}(0) = -\nabla E(0)$
2. wykonaj liniowe przeszukiwanie wzdłuż $\Delta\mathbf{w}(0)$ $\alpha = \arg\min_\alpha E(\mathbf{w}(0) + \alpha\Delta\mathbf{w}(0))$
3. zaktualizuj wagi $\mathbf{w}(1) = \mathbf{w}(0) + \alpha\Delta\mathbf{w}(0)$

**Kolejne kierunki** $m = 1, ..., d$

1. wyznacz kierunek $\Delta\mathbf{w}(m) = -\nabla E(m)$ metodą największego spadku w punkcie $\mathbf{w}(m)$
2. wyznacz współczynnik sprzężenia $\beta(m)$
3. zaktualizuj kierunek czyniąc go kierunkiem sprzężonym $\Delta\mathbf{w}(m) = -\nabla E(m) + \beta(m)\Delta\mathbf{w}(m-1)$
4. wykonaj liniowe przeszukiwanie wzdłuż $\Delta\mathbf{w}(m)$ $\alpha = \arg\min_\alpha E(\mathbf{w}(m) + \alpha\Delta\mathbf{w}(m))$
5. zaktualizuj wagi $\mathbf{w}(m+1) = \mathbf{w}(m) + \alpha\Delta\mathbf{w}(m)$

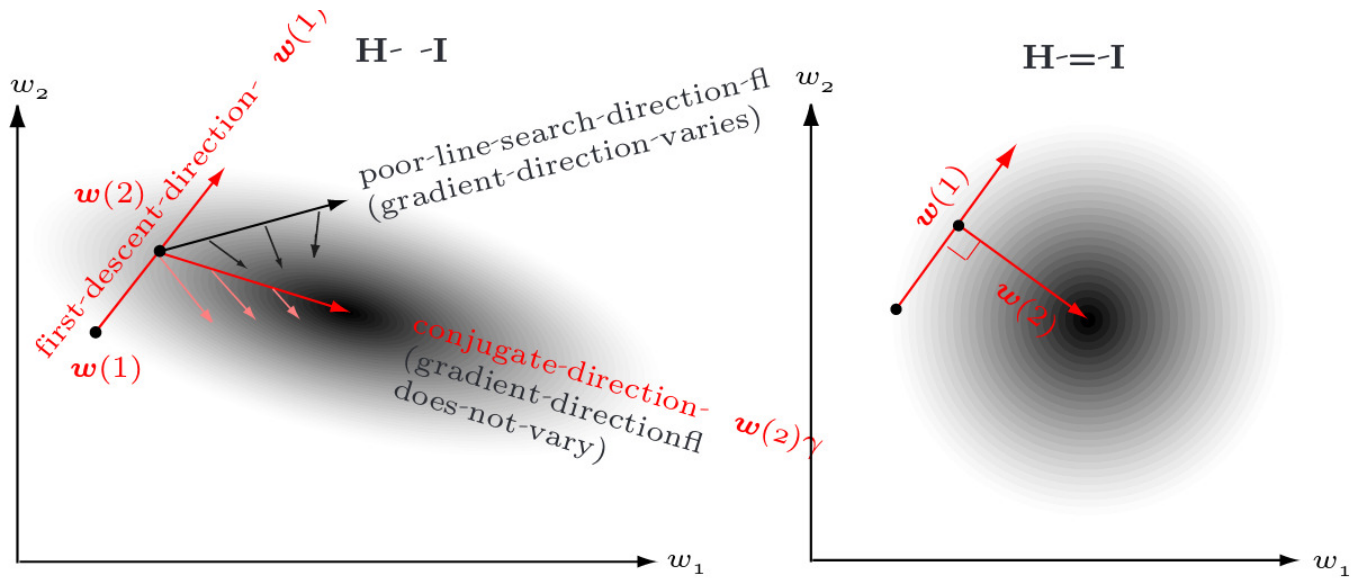
# Metoda gradientów sprzężonych

- na skutek przybliżeń w kolejnych iteracjach kierunki mogą zatracić ortogonalność, dlatego po $n$ krokach (gdzie $n$ jest liczbą optymalizowanych parametrów) metodę inicjujemy ponownie startując z ostatnio znalezionego punktu
- dla kwadratowej funkcji kosztu w $n$ wymiarach metoda CG osiąga minimum w $n$ krokach
- zbieżność liniowa lecz znacznie szybsza od spadku gradientem, brak konieczności wyznaczania odwrotności hesjanu $\mathbf{H}^{-1}$
- małe zapotrzebowanie pamięciowe, stosunkowo mała złożoność obliczeniowa pozwala stosować do bardzo złożonych problemów optymalizacyjnych

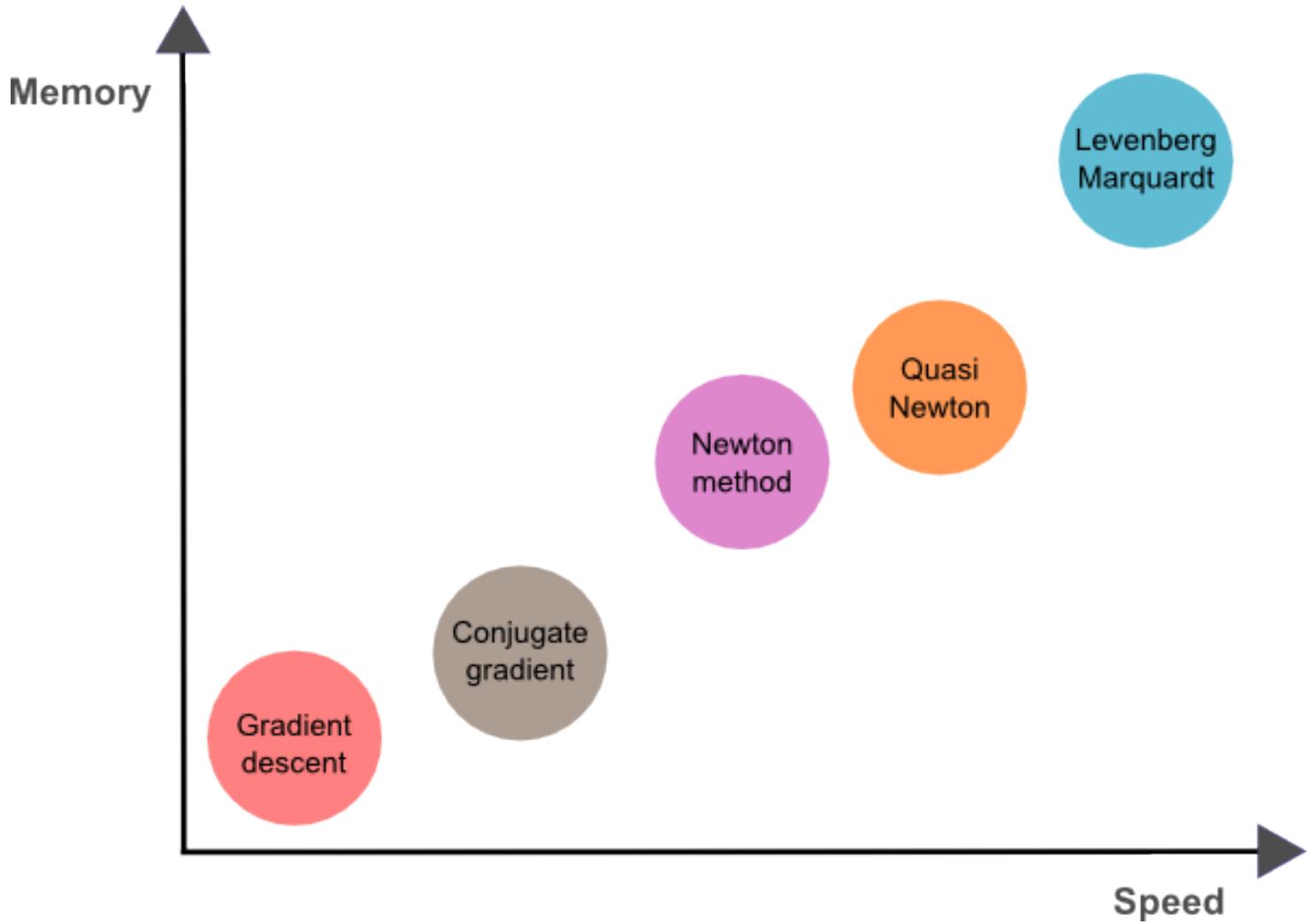
*https://en.wikipedia.org/wiki/Nonlinear_conjugate_gradient_method*

# Minimalizacja CG

- kolejny kierunek sprzężony nie wpływa ujemnie na wartość błędu wzdłuż poprzednio wyszukanych kierunków
- dla **H** diagonalnej kolejne kierunki są ortogonalne względem siebie




*Duda, Hart, Pattern recognition*




*Rys: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network*