Uczenie sieci neuronowych

ciąg dalszy...

Dobrze i prosto wytłumaczone w Aurelien Geron, „Uczenie Maszynowe z użyciem ScikitLearn i Tensorflow”, Helion 2020

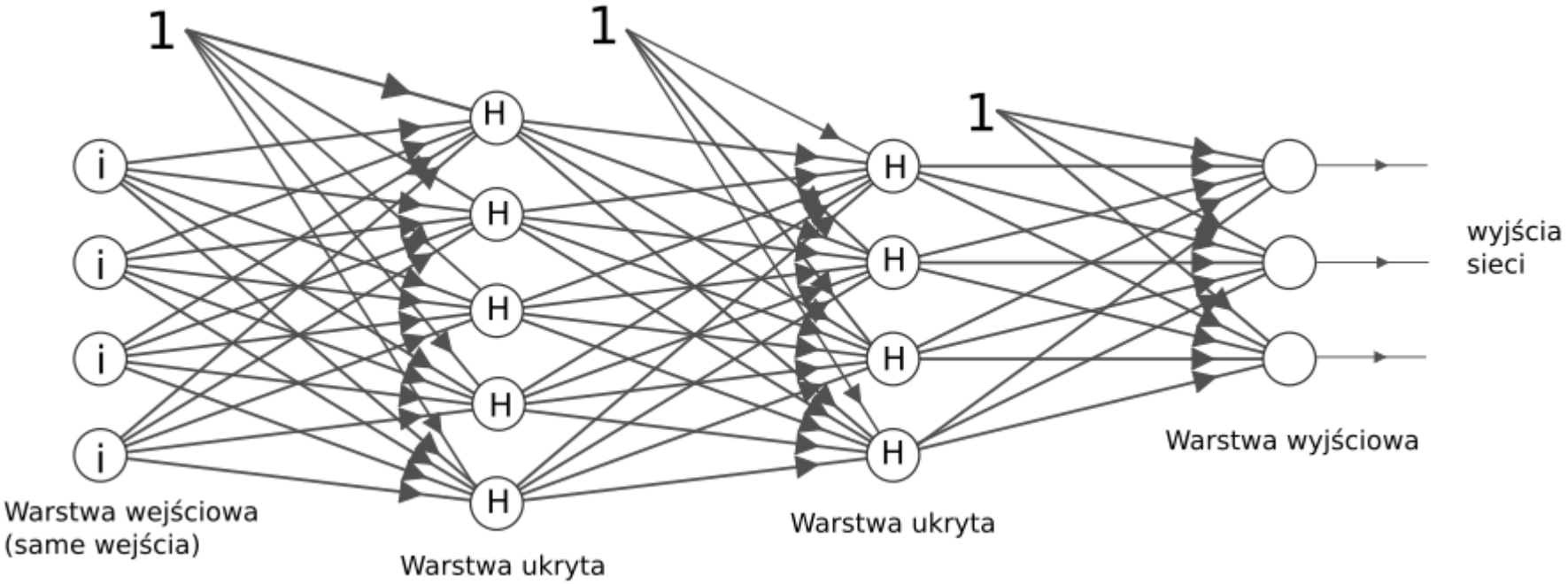
- SGD, SGD z momentem
- SGD z momentem Nestorowa (1983)
- AdaGrad (nie jest stosowany do głębokich, ale ułatwia zrozumienie kolejnych metod)
- RMSProp
- ADAM (ang. Adaptive momentum estimation) 2014r.
- NADAM – Adam z użyciem momentu Nestorowa 2016r.

Krótkie przypomnienie co było poprzednio...

- Algorytmy gradientowe
- Te alg. optymalizacji nie tylko do sieci...
- Stachurski A., Wierzbicki A.: *Podstawy optymalizacji*, Oficyna Wydawnicza PW, 1999
- Alg. gradientowe II rzędu - Stanisław Osowski, „Sieci neuronowe do przetwarzania informacji”, Oficyna wydawnicza 2020

Wielowarstwowy perceptron MLP (ang. Multilayer perceptron)

co najmniej jedna warstwa ukryta!



$$\left\{\mathbf{X}_k, \mathbf{T}_k\right\}_{k=1}^{N}$$

Sieć jednokierunkowa (ang. feedforward network)
i – wejścia sieci
h – neurony w warstwach ukrytych (ang. hidden layers)
W warstwach ukrytych są nieliniowe funkcje aktywacji np. tanh lub relu

Problem zanikających/eksplodujących gradientów (ang. vanishing / exploding gradient)

- problem zanikających gradientów (coraz mniejszych w niższych warstwach) przez długie lata uniemożliwiał skuteczną naukę sieci z większą liczbą warstw (dlatego przeważnie stosowano do dwóch warstw ukrytych w MLP).
- problem eksplodujących gradientów może występować w sieciach rekurencyjnych

Walka z problemem zanikającego gradientu

- Nienasycające funkcje aktywacji np. (nie sigmoidalne)
- Odpowiednia inicjalizacja wag (np. He lub Glorota)
- Normalizacja wsadowa, obcinanie gradientu
- Połączenia nie tylko między sąsiednimi warstwami (np. resnet)

Uczenie

Sieci MLP i CNN są uczone iteracyjnie algorytmami gradientowymi.

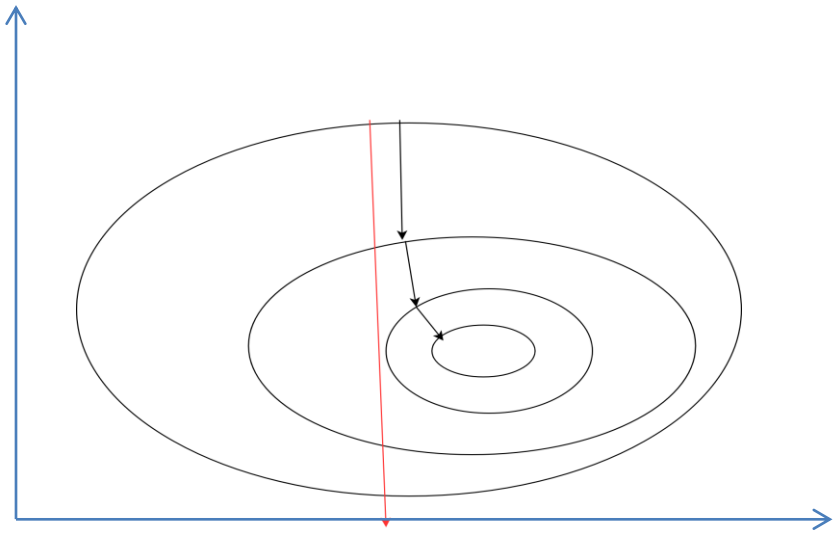
Zmiany wag w kolejnych iteracjach metody gradientu prostego następują w kierunku antygradientu (najszybszego spadku f.celu):

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta_i \nabla E(\mathbf{w}_i),$$

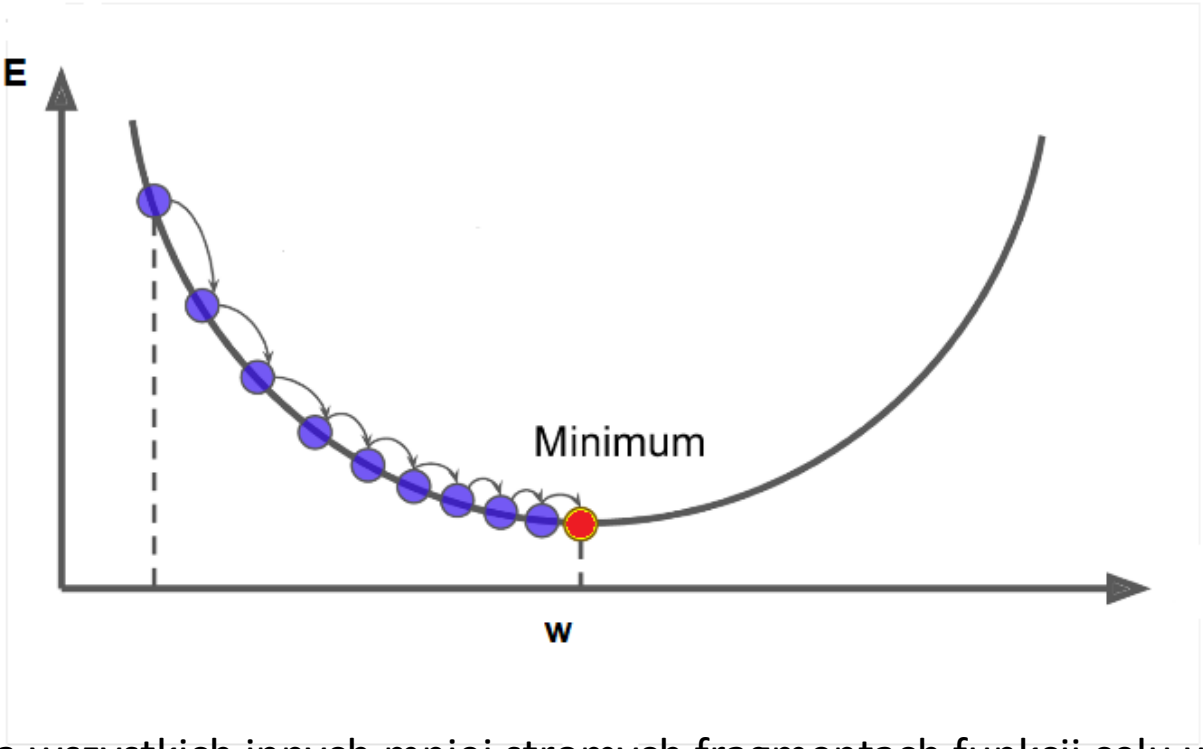
gdzie gradient funkcji straty

$$\nabla E(\mathbf{w}_i) = \left[\frac{\partial E}{\partial w_{i,1}}, \frac{\partial E}{\partial w_{i,2}}, \cdots, \frac{\partial E}{\partial w_{i,n}}\right]^T,$$

$w_{i,j}$ – oznacza jtą wagę w iteracji i.



Gradient prosty



Blisko minimum i na wszystkich innych mniej stromych fragmentach funkcji celu wolno zbiega

Im większa stromizna – tym większe nachylenie (gradient) – szybkie zmiany f. celu.
Blisko minimum często płasko i duża liczba kroków jest potrzebna.

Zbieżność liniowa. Przy stałym współczynniku uczenia osiągnięcie minimum z dokładnością ε może zająć O(1/ ε) iteracji. Uzyskanie 10 razy większej dokładności może wymagać 10 razy więcej iteracji.

Uczenie metoda gradientu prostego

- Metoda gradientu prostego działa na ogół nienajgorzej, gdy ***w*** *jest daleko od* minimum E(***w***)*.* Jeśli bieżący punkt jest blisko minimum, zmiana gradientu często jest mała z powodu małego nachylenia, więc krok jest niewielki.

Dobór współczynnika uczenia η (ang. learning rate)

- Współczynnik uczenia powinien maleć podczas uczenia, żeby umożliwić dobre zbliżenie sie do minimum lokalnego (a nie odbijanie się pomiędzy ścianami doliny, w której znajduje sie minimum)

Harmonowanie uczenia (ang. learning scheduling)

- Harmonowanie potęgowe (ang. power scheduling)

$$\eta_i = \frac{\eta_0}{(1 + i / c_1)^{C_2}},$$

gdzie

i oznacza nr iteracji,

c_1 i c_2 są stalymi (hiperparametrami). Zazwyczaj $c_2 = 1$.

W Kerasie c_2=1 i decay=1/c_1:

```
optimizer = SGD(lr=0.01, decay=1e-4)
```

Harmonogramowanie stałoprzedziałowe (ang. piecewise constant scheduling)

- Stała wartość współczynnika uczenia przez określoną liczbę kolejnych epok np. przez pierwsze kilka epok może on wynosić 0,1, przez kolejne kilkanaście 0,01 a następnie 0,001 itd.

```
W Kerasie można użyć własnej funkcji:
def piecewise_constant(epoch):
  if epoch < 5:
      return 0.1
  elif epoch < 20:
       return 0.01
   else:
      return 0.001

lr_scheduler =
keras.callbacks.LearningRateScheduler(piecewise_constant)
history = model.fit(X, T, ...,  callbacks=[lr_scheduler])
```

Harmonogramie wydajnościowe

- Jeżeli wartość funkcji celu na zbiorze walidacyjnym przestaje maleć w kolejnych epokach, to zmniejszam współczynnik uczenia.

- W matlabie jeżeli w kolejnej epoce E maleje to współczynnik uczenia jest zwiększany o 5%, a w przeciwnym razie jest zmniejszany o 30%.

Gradient prosty
(ang. gradient descent)

Załóżmy, że funkcją straty (celu) jest MSE (ang. Mean Square Error), siec ma jedno wyjście i mamy 20000 przykładów w zbiorze uczącym

$$\left\{ \mathbf{X}_k, T_k \right\}_{k=1}^{20000}$$

Gradient prosty (ang. gradient descent)
Załóżmy, że f. straty jest MSE

Wsadowy gradient prosty (ang. batch gradient descent)

$$E = \frac{1}{20000} \sum_{k=1}^{20000} \left(T_k - Y_k\right)^2$$

jedna epoka = jedna iteracja

Gradient prosty z minigrupami (miniwsadami) (ang. mini-batch gradient descent)

$$E = \frac{1}{100} \sum_{k=1}^{100} \left(T_k - Y_k\right)^2$$

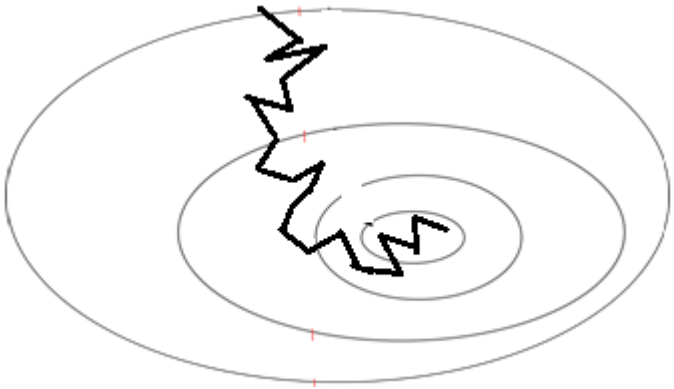
jedna epoka = 20000/100=200 iteracji
elementy wybierane losowo

Stochastyczny spadek wzdłóż gradientu (ang. stochastic gradient descent)

$$E = \left(T_k - Y_k\right)^2$$

jedna epoka = 20000/1=20000 iteracji
elementy wybierane losowo

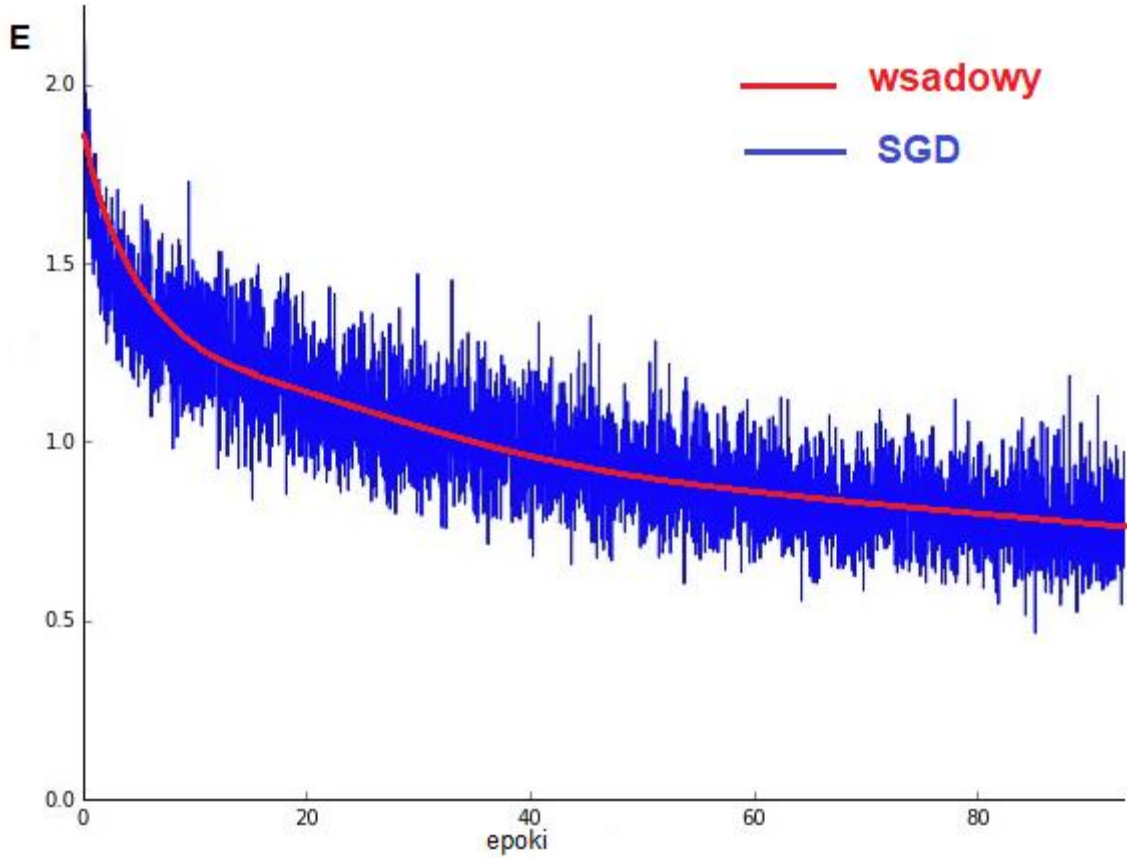
Stochastyczny spadek wzdłuż gradientu (ang. SGD Stochastic gradient descent)



SGD – może czasami uciec z minimum lokalnego (ma większe szanse na znalezienie minimum globalnego niż wsadowy gradient prosty)
Przez losowość nie osiąga minimum (skacze koło minimum), jeżeli jest stały współczynnik uczenia η. Jednym z rozwiązań jest zmniejszanie współczynnika uczenia podczas nauki

Funkcja określająca zmiany współczynnika uczenia w każdym przebiegu nazywana jest harmonogramem uczenia (ang. learning scheduling)

SGD Stochastic Gradient Descent



W przypadku schodzenia po gradiencie z minigrupami sytuacja pośrednia między wsadowym gradientem prostym a SGD

Uczenie – częsta poprawa - gradient prosty z momentem

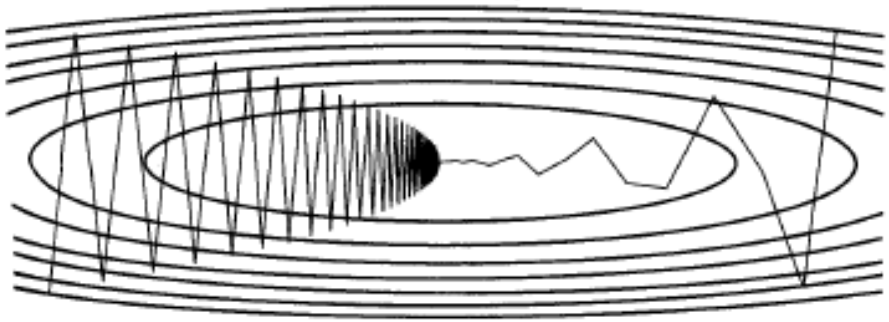
Z momentem:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta \cdot \nabla E_k + \beta(\mathbf{w}_i - \mathbf{w}_{i-1}),$$
gdzie $\beta \in [0,1]$

β bliskie 1 duża bezwładność

Bez momentu, jeżeli gradient i współczynnik uczenia duże, to mogą występować oscylacje wokół minimum

Moment ułatwia przeskoczenie minimum lokalnego



Gdy kolejne kierunki (gradienty) podobne, wówczas ich działanie sie kumuluje i prędkość coraz większa. Wtedy nawet zmiana gradientu na przeciwny nie od razu spowoduje zawrócenie. Gdy kolejne kierunki się znacznie różnią to następuje hamowanie.

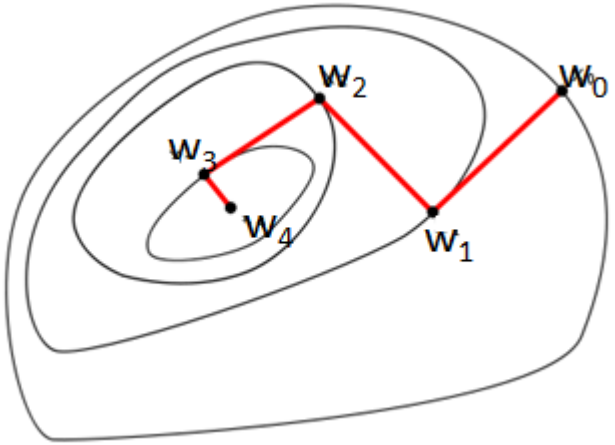
Uczenie – metoda najszybszego spadku

działa jak gradient prosty, ale **z minimalizacją kierunkowa**

$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta_i \nabla E(\mathbf{w}_i)$,
gdzie gradient funkcji straty

$$\nabla E(\mathbf{w}_i) = \left[\frac{\partial E}{\partial w_{i,1}}, \frac{\partial E}{\partial w_{i,2}}, \cdots, \frac{\partial E}{\partial w_{i,n}}\right]^T,$$

$w_{i,j}$ – oznacza jtą wagę w iteracji i.



Wolna zbieżność – liniowa

Prosty

Nie wymaga dużej pamięci ani mocy obliczeniowej

zbieżność liniowa - przy spełnieniu założeń metody, odległości pomiędzy kolejnymi przybliżeniami a minimum funkcji, które znajduje sie w punkcie **w***, maleją liniowo:

$$\|\mathbf{w} * - \mathbf{w}_{i+1}\| \leq c\|\mathbf{w} * - \mathbf{w}_i\| \qquad \frac{\|\mathbf{w} * - \mathbf{w}_{i+1}\|}{\|\mathbf{w} * - \mathbf{w}_i\|} \leq c$$

Norma L2 - RMSE

Uczenie

Rozwinięcie w szereg Taylora funkcji celu $E(\mathbf{w})$ w kierunku $\mathbf{p}$

$$E(\mathbf{w}+\mathbf{p}) = E(\mathbf{w}) + \left[\nabla E(\mathbf{w})\right]^T \cdot \mathbf{p} + \frac{1}{2}\mathbf{p}^T \mathbf{H}(\mathbf{w})\mathbf{p} + \ldots, \qquad (1)$$

gdzie

$$\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \cdots, \frac{\partial E}{\partial w_n}\right]^T,$$

gradient

$$\mathbf{H}(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1 \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_n \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_n \partial w_n} \end{bmatrix}.$$

Hesjan, macierz Hessego – macierz kwadratowa drugich pochodnych cząstkowych funkcji straty (celu)

Uczenie – alg. gradientowe pierwszego rzędu

Liniowe przybliżenie funkcji celu $E(\mathbf{w})$ w najbliższym sąsiedztwie punktu $\mathbf{w}$

$$E(\mathbf{w}+\mathbf{p}) \approx E(\mathbf{w}) + \left[\nabla E(\mathbf{w})\right]^T \cdot \mathbf{p} \qquad (1)$$

Chcemy, aby $E(\mathbf{w}+\mathbf{p}) < E(\mathbf{w})$

Metoda gradientu prostego:

$$\mathbf{p} = -\nabla E(\mathbf{w})$$

$-[\nabla E(\mathbf{w})]^T \cdot \mathbf{p} < 0,$
ponieważ
$[\nabla E(\mathbf{w})]^T \cdot \nabla E(\mathbf{w}) > 0$

Uczenie – alg. gradientowe drugiego rzędu

Kwadratowe przybliżenie funkcji celu $E(\mathbf{w})$ w najbliższym sąsiedztwie punktu $\mathbf{w}$

$$E(\mathbf{w}+\mathbf{p}) \approx E(\mathbf{w}) + \left[\nabla E(\mathbf{w})\right]^T \cdot \mathbf{p} + \frac{1}{2}\mathbf{p}^T \mathbf{H}(\mathbf{w})\mathbf{p} \qquad (2)$$

Minimum (2) wymaga, aby $\dfrac{dE(\mathbf{w}+\mathbf{p})}{d\mathbf{p}} = 0$

Z powyższego warunku

$$\nabla E(\mathbf{w}) + \mathbf{H}(\mathbf{w})\mathbf{p} = 0$$

$$\mathbf{p} = -\left[\mathbf{H}(\mathbf{w})\right]^{-1} \nabla E(\mathbf{w})$$

Są też inne metody...

Metody gradientowe drugiego rzędu korzystają z Hesjanu lub jego przybliżenia

$$\mathbf{H}(\mathbf{w}_i) = \begin{bmatrix} \frac{\partial^2 E}{\partial w_{i,1} \partial w_{i,1}} & \cdots & \frac{\partial^2 E}{\partial w_{i,1} \partial w_{i,n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_{i,n} \partial w_{i,1}} & \cdots & \frac{\partial^2 E}{\partial w_{i,n} \partial w_{i,n}} \end{bmatrix}$$

Hesjan (macierz Hessego) – macierz kwadratowa drugich pochodnych cząstkowych

W metodzie newtonowskiej zmiany wag następują zgodnie z formułą

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta_i [\mathbf{H}(\mathbf{w}_i)]^{-1} \nabla E(\mathbf{w}_i).$$

Uczenie metoda Newtona

Kwadratowe przybliżenie f.straty (f.kosztu, f.celu)

$$E(\mathbf{w}+\mathbf{p}) \approx E(\mathbf{w}) + \left[\nabla E(\mathbf{w})\right]^T \cdot \mathbf{p} + \frac{1}{2}\mathbf{p}^T \mathbf{H}(\mathbf{w})\mathbf{p} \qquad (2)$$

Minimalizujemy przybliżenie

$$\frac{\partial\left(E(\mathbf{w}) + \left[\nabla E(\mathbf{w})\right]^T \cdot \mathbf{p} + \frac{1}{2}\mathbf{p}^T \mathbf{H}(\mathbf{w})\mathbf{p} \right)}{\partial \mathbf{p}} = 0, \quad \frac{\partial^2 \text{licznik}}{\partial \mathbf{p}^2} \text{ dodatnio okreslona} \qquad (3)$$

$$\mathbf{p} = -\left[\mathbf{H}(\mathbf{w})\right]^{-1} \nabla E(\mathbf{w})$$ ←**Newtonowski algorytm optymalizacji**

- Jeżeli **gradient=**0, to macierz **H**[(**w**)] musi być dodatnio określona. to żeby punkt ten był minimum. Można domniemywać, że prawdopodobnie będzie dodatnio określona w otoczeniu minimum lokalnym ze względu na duże prawdopdobieństwo, że przekroje f.celu będą tam miały kształt zbliżony do dna paraboli.
- Metoda Newtona jest wiec dobra, kiedy punkt jest blisko minimum.
- Jesli macierz $[\mathbf{H}(\mathbf{w})]^{-1}$ jest półdodatnio okreslona, kierunek **p** *jest kierunkiem* spadku.

- problem z odwracaniem Hesjanu – wymagana spora liczba działań – złożoność $O(n^3)$ lub dla lepszego alg. $O(n^{2,7})$.
- problem z dodatnią określonością Hesjanu
- zbieżność kwadratowa (tzn. odległości pomiędzy kolejnymi przybliżeniami a minimum funkcji celu maleją kwadratowo

$$\left\|\mathbf{w}^* - \mathbf{w}_{k+1}\right\| \leq c\left\|\mathbf{w}^* - \mathbf{w}_k\right\|^2$$

Uczenie – metoda zmiennej metryki zamiast Hesjanu **H** stosuje się jego przybliżenie **G**

$$E(\mathbf{w}+\mathbf{p}) \approx E(\mathbf{w}) + \left[\nabla E(\mathbf{w})\right]^T \cdot \mathbf{p} + \frac{1}{2}\mathbf{p}^T \mathbf{H}(\mathbf{w})\mathbf{p} \qquad (2)$$

$$\mathbf{p} = -\left[\mathbf{H}(\mathbf{w})\right]^{-1} \nabla E(\mathbf{w})$$ ←**Newtonowski algorytm optymalizacji**

Broydena-Goldfarba-Fletchera-Shanno (BFGS):

$$\mathbf{G}(\mathbf{w}_k) \approx \mathbf{H}(\mathbf{w}_k)$$

$$\mathbf{s}_k = \mathbf{w}_k - \mathbf{w}_{k-1}$$

$$\mathbf{r}_k = \nabla E(\mathbf{w}_k) - \nabla E(\mathbf{w}_{k-1})$$

$$\mathbf{V}_0 = \mathbf{I}$$

$$\mathbf{V}_k \approx \left[\mathbf{G}(\mathbf{w}_k)\right]^{-1}$$

$$V_k = V_{k-1} + \left[1 + \frac{r_k^T V_{k-1} r_k}{s_k^T r_k}\right] \frac{s_k s_k^T}{s_k^T r_k} - \frac{s_k r_k^T V_{k-1} + V_{k-1} r_k s_k^T}{s_k^T r_k}$$

Uczenie – metoda zmiennej metryki

$$E(\mathbf{w}+\mathbf{p}) \approx E(\mathbf{w}) + \left[\mathbf{g}(\mathbf{w})^T\right]\cdot\mathbf{p} + \frac{1}{2}\mathbf{p}^T\mathbf{H}(\mathbf{w})\mathbf{p} \qquad (2)$$

$$\mathbf{p} = -\left[\mathbf{H}(\mathbf{w})\right]^{-1}\mathbf{g}(\mathbf{w})$$

Broydena-Goldfarba-Fletchera-Shanno (BFGS):

$$\mathbf{G}(\mathbf{w}_k) \approx \mathbf{H}(\mathbf{w}_k)$$

$$\mathbf{s}_k = \mathbf{w}_k - \mathbf{w}_{k-1}$$

$$\mathbf{r}_k = \mathbf{g}(w_k) - \mathbf{g}(w_{k-1})$$

$$\mathbf{V}_k \approx \left[\mathbf{G}(\mathbf{w}_k)\right]^{-1}$$

$$\mathbf{V}_0 = \mathbf{I}$$

$$V_k = V_{k-1} + \left[1 + \frac{r_k^T V_{k-1} r_k}{s_k^T r_k}\right]\frac{s_k s_k^T}{s_k^T r_k} - \frac{s_k r_k^T V_{k-1} + V_{k-1} r_k s_k^T}{s_k^T r_k}$$

Uczenie – metoda zmiennej metryki

Davidona-Fletchera-Powella (DFP):

$$V_k = V_{k-1} + \frac{s_k s_k^T}{s_k^T r_k} - \frac{V_{k-1} r_k r_k^T V_{k-1}}{r_k^T V_{k-1} r_k}$$

Zarówno w DFP jak i BFGS przy **V**o=**I** i minimalizacji kierunkowej w każdym kroku można zapewnić dodatnią określoność aproksymowanej macierzy hesjanu.
Minimalizacja kierunkowa – zamiast stałej wartości współczynnika sprawdzanie jak daleko można podążyć w danym kierunku, zanim wartość funkcji straty (kosztu, celu) zacznie rosnąć

Algorytm Levenberga-Marquardta Połączenie metody najszybszego spadku i Newtona dla **MSE**

$$E = \frac{1}{2}\sum_{j=1}^{q}(\mathbf{y}_j - \mathbf{d}_j)^2 \qquad \mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_q \end{bmatrix}$$

$$\mathbf{J}(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2 e_1}{\partial w_1} & \frac{\partial^2 e_1}{\partial w_2} & \cdots & \frac{\partial^2 e_1}{\partial w_n} \\ \frac{\partial^2 e_2}{\partial w_1} & \frac{\partial^2 e_2}{\partial w_2} & \cdots & \frac{\partial^2 e_2}{\partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 e_q}{\partial w_1} & \frac{\partial^2 e_q}{\partial w_2} & \cdots & \frac{\partial^2 e_q}{\partial w_n} \end{bmatrix}.$$

$$\nabla E(\mathbf{w}_k) = \left[\mathbf{J}(\mathbf{w}_k)\right]^T \mathbf{e}(\mathbf{w}_k)$$

$$\mathbf{G}(\mathbf{w}) = \left[\mathbf{J}(\mathbf{w})_k\right]^{\mathbf{T}} \mathbf{J}(\mathbf{w}_k) + v_k \mathbf{I}$$

Daleko od minimum v_k duże i kierunek podobny jak w metodzie gradientu prostego!!!

Wartość skalara v_k zwanego parametrem Levenberga-Marquardta, jest zmieniana w trakcie procesu optymalizacyjnego.

Algorytm Levenberga-Marquardta

O skuteczności alg. decyduje odpowiedni dobór wartości v_k.
Duża początkowo wartość v_k musi ulegać redukcji aż do wartości zerowej przy rozwiązaniu bliskim optymalnemu.
Sposoby zmiany wartości v_k są podane m.in. w książce
Stanisław Osowski, „Sieci neuronowe do przetwarzania informacji”, Oficyna wydawnicza 2020
Są w niej dobrze opisane alg. gradientowe II rzędu.

Dlaczego z sieciami głębokimi są stosowane algorytmy gradientowe pierwszego rzędu?

Dlaczego proste algorytmy pierwszego rzędu są stosowane do uczenia sieci głębokich?

Istotne wady i zalety metod gradientowych pierwszego rzędu

Zalety:

- Złożoność pamięciowa O(n), co jest kluczowe dla sieci głębokich! Głęboka sieć konwolucyjna VGG16 ma 138357544 parametrów (wag) zmienianych podczas nauki. Macierz hesjanu miałaby $138357544^2 = 19142809981711936 \approx 19 \cdot 10^{15}$ elementów czyli około 19 biliardów elementów (19 peta elementów)!
- Prosta implementacja

Wady:
wolna zbieżność (liniowa).

W lewej kolumnie są linki do dokumentacji sieci w Kerasie

Model	Rozmiar	Top-1 Dokładność (ang. Accuracy)	Top-5 Dokładność	Parametery	Głębokość
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88

Tabela: Dane sieci z `https://keras.io/applications/`

Dokładność Top 5 - prawidłowa klasa wśród pięciu najbardziej prawdopodobnych klas wskazywanych przez sieć

Dokładność Top 1 - prawidłowa klasa wskazana przez sieć jako najbardziej prawdopodobna

Najmniej parametrów ma MobileNetV2. Nawet dla tej sieci hesjan miałby aż $3538984^2 = 3538984 \approx 13 \cdot 10^{12}$ elementów (ponad 12 teta elementów)!

Dobrze i łatwo wytłumaczone w Aurelien Geron, Uczenie Maszynowe z użyciem ScikitLearn i Tensorflow, Helion 2020

- SGD, SGD z momentem
- SGD z momentem Nestorowa (1983)
- AdaGrad (nie jest stosowany do głębokich ale ułatwia zrozumienie kolejnych metod)
- RMSProp – przeważnie najlepsze rezultaty zanim nie wynaleziono ADAMa
- ADAM (ang. Adaptive momentum estimation) 2014r.
- NADAM – Adam z użyciem momentu Nestorowa 2016r.
- Dla niektórych problemów wyszukane adaptacyjne szwankuje i wtedy zawsze można spróbować SGD z momentem (zwykłym lub Nestorowa)

Uczenie – gradient prosty

$$\mathbf{w}_{i+1} = \mathbf{w}_k - \eta \cdot \nabla E(\mathbf{w}_i),$$

UWAGA
Na dalszych slajdach będzie uproszczona notacja bez indeksów wskazujących numer iteracji

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla E(\mathbf{w}),$$

Gradient prosty z momentem

$$\mathbf{m} \leftarrow \beta\mathbf{m} - \eta \cdot \nabla E(\mathbf{w}),$$
$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{m}$$

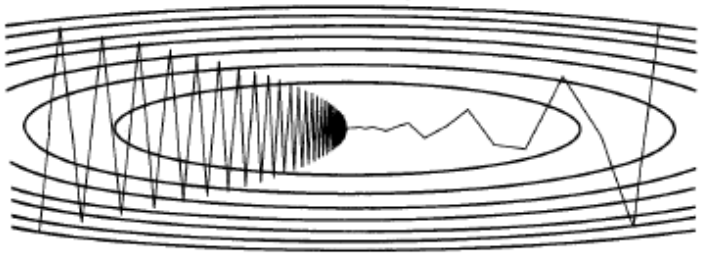
Uwzględniane są poprzednie gradienty – w każdej iteracji iloczyn gradientu i współczynnika uczenia ηodejmowany jest od wektora momentu **m.**

Wagi zmieniane są poprzez dodanie tego momentu.

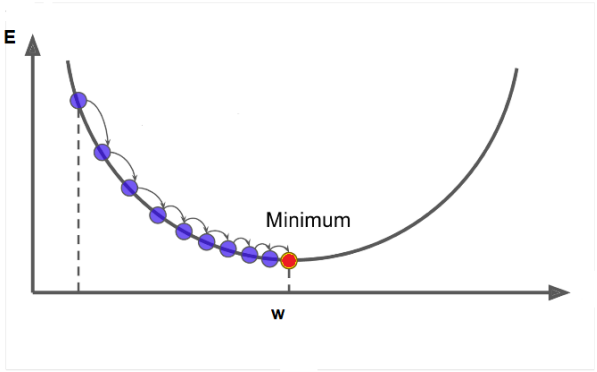
Gradient wykorzystywany jest jako przyspieszenie, a nie prędkość.

Hiperparametr β (zwany też momentem) zapobiega nadmiernemu wzrostowi prędkości przypomina trochę zjawisko tarcia. Jego wartość mieści się między 0 (duże tarcie) i 1 (brak tarcia).

Typowa wartość β=0,9



Załóżmy, że jesteśmy blisko minimum lokalnego koło którego nachylenie jest niewielkie – zmiany nachylenia są bardzo małe.



Zobaczmy, że gdybyśmy dla uproszczenia założyli, że nachylenie jest stałe (stały gradient), to dla $\beta=0{,}9$ zmiany wag będą 10 razy szybsze niż przy gradiencie prostym !

$$\mathbf{m} = \beta\mathbf{m} - \eta \cdot \nabla E(\mathbf{w}),$$
$$\mathbf{m} - \beta\mathbf{m} = -\eta \cdot \nabla E(\mathbf{w}),$$
$$\mathbf{m}(1-\beta) = -\eta \cdot \nabla E(\mathbf{w}),$$
$$\mathbf{m} = \frac{-\eta \cdot \nabla E(\mathbf{w})}{1-\beta}$$

W kerasie:

```
optimizer = SGD (lr=0.001, momentum=0,9)
model.compile(loss='mse', optimizer=optimizer, metrics=['mse'])
```

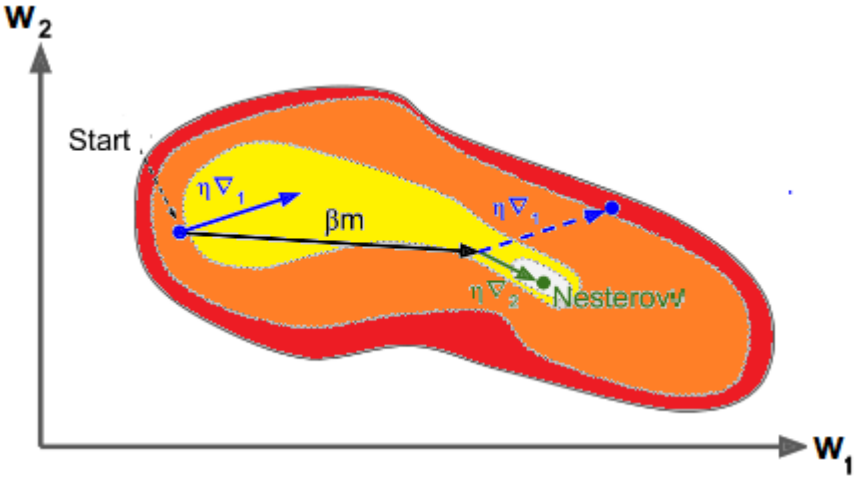
Algorytm Nesterowa zwany też przyspieszonym spadkiem wzdłuż gradientu (NAG ang. Nestorov accelerated gradient) 1983r.

Gradient funkcji straty nie w obecnej pozycji określonej przez przez wektor wag **w**, ale nieco z przodu w kierunku pędu $\mathbf{w}+\beta\mathbf{m}$.

$$\mathbf{m} \leftarrow \beta\mathbf{m} - \eta \cdot \nabla E(\mathbf{w} + \beta\mathbf{m}),$$

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{m}$$

Ta modyfikacja zazwyczaj przyspiesza, ponieważ wykorzystanie informacji o gradiencie nieco z przodu pozwoli trochę lepiej dobrać kierunek.




```
optimizer = SGD (lr=0.001, momentum=0.9, Nestorov=True)
model.compile(loss='mse', optimizer=optimizer, metrics=['mse'])
```

AdaGrad

W sieciach konwolucyjnych mamy różnego typu warstwy np. konwolucyjne, gęste...

Dla rożnych wag przydałaby się inne wartości współczynnika uczenia.

Algorytm AdaGrad redukuje współczynnik uczenia szybciej dla wymiarów (każda waga związana jest z innym wymiarem) o stromym przebiegu funkcji straty niż dla wymiarów o bardziej łagodnym przebiegu – umożliwia wcześniejsze wykrycie zmianę nachylenia i korygowanie kierunku zmian w stronę minimum.

AdaGrad sprawdza się dobrze w nieskomplikowanych zadaniach np. doborze współczynników regresji liniowej, ale potrafi zatrzymać się zbyt wcześnie podczas uczenia sieci neuronowych.

Jego poznanie ułatwia zrozumienie lepszych algorytmów.

AdaGrad

Łatwo zrozumieć, jeżeli pamięta się o związku pochodnej i nachylenia funkcji i wiedząc, że elementy gradientu są pochodnymi cząstkowi E

$$\mathbf{s} \leftarrow \mathbf{s} + \nabla E(\mathbf{w}) \otimes \nabla E(\mathbf{w}), \qquad (1)$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w}) \div \sqrt{\mathbf{s} + \varepsilon}, \qquad (2)$$

gdzie

$\otimes$ oznacza mnożenie poszczególnych elementów wektorów,

$\div$ oznacza dzielenie poszczególnych elementów wektorów.

Wektor **s** kumuluje kwadraty gradientu – jeżeli funkcja straty E jest stromo nachylona w i-tym wymiarze, to w kolejnych iteracjach i-ty element wektora s będzie miał coraz większą wartość.

Łatwo zauważyć, że im bardziej strome nachylenie f. straty E w *i*-tym wymiarze, tym większa wartość *i*-tego elementu wektora $\nabla E(\mathbf{w}) \otimes \nabla E(\mathbf{w})$

Wystarczy zauważyć, że gradient zawiera pochodne cząstkowe i wiedzieć o związku pochodnej z nachyleniem

Wzór (2) podobny do gradientu prostego, ale wektor gradientu jest zmniejszany przez $\sqrt{\mathbf{s} + \varepsilon}$

ε jest bardzo małą wartością, która zabezpiecza przed dzieleniem przez zero

W kolejnych alg. też m.in. jest wykorzystywany wektor kwadratów gradientu

Podczas analizy kolejnych algorytmów również warto pamiętać, że **im bardziej strome nachylenie f. straty E w *i*-tym wymiarze, tym większa wartość *i*-tego elementu wektora**

$$\nabla E(\mathbf{w}) \otimes \nabla E(\mathbf{w})$$

RMSProp (ang. root mean squared propagation 2012r. (znacznie lepszy od AdaGrad)

- AdaGrad może zwolnić zbyt szybko i skończyć działanie nie osiągając minimum lokalnego.
- Problem ten rozwiązano w RMSProp poprzez gromadzenie głównie aktualnych gradientów z ostatnich iteracji (w AdaGrad stare i nowe gradienty wpływaja na **s** w jednakowym stopniu. W RMSprop zmodyfikowano równanie (1) z AdaGrad)

$$\mathbf{s} \leftarrow \rho\mathbf{s} + (1-\rho)\nabla E(\mathbf{w}) \otimes \nabla E(\mathbf{w}),$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla E(\mathbf{w}) \div \sqrt{\mathbf{s} + \varepsilon},$$

gdzie

$\otimes$ oznacza mnożenie poszczególnych elementów wektorów,

$\div$ oznacza dzielenie poszczególnych elementów wektorów.

Współczynnik rozpadu ρ zazwyczaj ma wartość 0,9.

```
optimizer = RMSprop(lr=0.001, rho=0.9)
```

Wyjaśnienie nazwy RMSProp (ang. root mean squared propagation)

- The **root-mean-square deviation** (**RMSD**) or **root-mean-square error** (**RMSE**) is a frequently used measure of the differences between values (sample or population values) predicted by a model and the values observed. The RMSD represents the square root of MSE.

$$RMSE = \sqrt{MSE}$$

W RMSprop jest w drugim równaniu pierwiastek z kwadratu elementów gradientu:

$$\mathbf{s} \leftarrow \rho \mathbf{s} + (1-\rho) \nabla E(\mathbf{w}) \otimes \nabla E(\mathbf{w}),$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w}) \div \sqrt{\mathbf{s} + \varepsilon},$$

ADAM (ang. adaptive moment estimation) 2014r. – adaptacyjne szacowanie momentu

ADAM łączy koncepcję optymalizacji momentem i optymalizacji RMSProp:

$$\mathbf{m} \leftarrow \beta\mathbf{m} + (1-\beta)\nabla E(\mathbf{w}),$$
$$\mathbf{s} \leftarrow \rho\mathbf{s} + (1-\rho)\nabla E(\mathbf{w}) \otimes \nabla E(\mathbf{w}),$$
$$\hat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1-\beta},$$
$$\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho},$$
$$\mathbf{w} \leftarrow \mathbf{w} + \eta\hat{\mathbf{m}} \div \sqrt{\mathbf{s} + \varepsilon},$$

Wzmocnienie **m** i **s** na początku treningu, (wektory m i s są inicjowane zerami)

gdzie

$\otimes$ oznacza mnożenie poszczególnych elementów wektorów,

$\div$ oznacza dzielenie poszczególnych elementów wektorów.

Współczynnik rozkładu momentu β ma zazwyczaj wartość 0.9, a ρ ma często wartość 0,99.9

ADAM

- To jest bardzo dobry i popularny algorytm uczenia sieci neuronowych.

NADAM

- ADAM ze sztuczką (momentem) Nestorowa.

Alg. gradientowe

- Co byłoby gdyby początkowe wartości wag wszystkich neuronów były takie same (i nie było dropoutu) ?
- Wagi wszystkich neuronów po treningu miałyby takie same wartości

- Warunki zatrzymania procesu uczenia (np. mała wartość normy z gradientu, maksymalna liczba epok, zadana dokładność, norma z różnicy wektorów wag z kolejnych iteracji, itp.)

Dziękuję za uwagę