

Głębokie sieci neuronowe

Głębokie sieci neuronowe - Plan

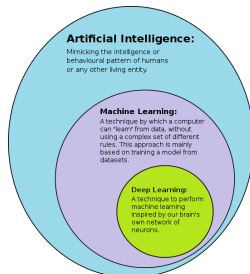
1. Głębokie MLP (DNN)
2. Sieci splotowe (CNN)
3. Sieci rekurencyjne (RNN) i uczenie sekwencji
4. Autoenkodery (AE), Deep belief network (DBN)
5. Generative Adversarial Networks (GAN)
6. Modele z uwagą i transformery

Literatura

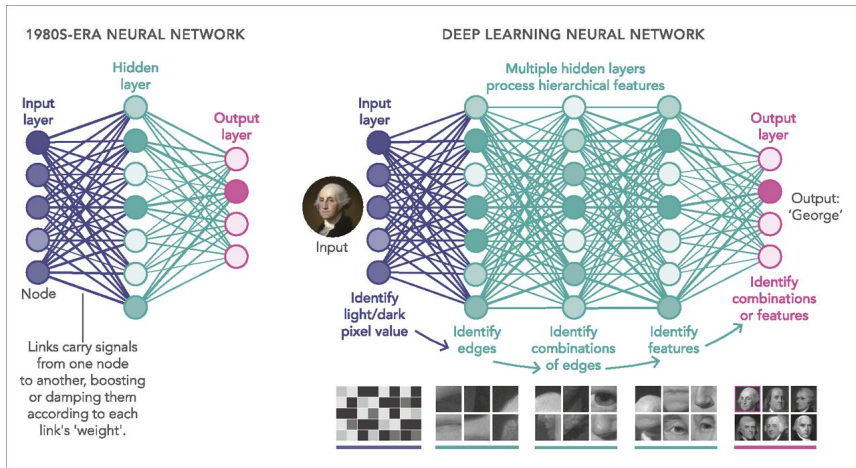
-  Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*. MIT Press, 2016,
<http://www.deeplearningbook.org>
-  Michael Nielsen, *Neural Networks and Deep Learning*,
<http://neuralnetworksanddeeplearning.com/>
-  Aston Zhang, Zachary C. Lipton, Mu Li, Alexander J. Smola, *Dive into Deep Learning*,
<https://d2l.ai/>
-  Julien Vitay, *Neurocomputing*,
<https://julien-vitay.net/lecturenotes-neurocomputing/>
-  Denny Britz, *Deep Learning Glossary*,
<http://www.wildml.com/deep-learning-glossary/>

Głębokie uczenie to metody uczenia maszynowego

- zbudowane z **wielu warstw realizujących nieliniowe transformacje**
- tworzą **model hierarchiczny**
warstwy odwzorowują kolejne poziomy abstrakcji, najniższe warstwy tworzą reprezentują najprostszyszych cech (*surowego*) sygnału wejściowego, wyższe warstwy - generują bardziej ogólne koncepty bazując na relacjach z poprzednich warstw
- automatycznie **uczą się reprezentacji** (wykrywają cechy)
- **Transfer learning** - wykorzystanie wiedzy zdobytej przy rozwiązywaniu innego (podobnego) problemu, np. „przeszczep” wag wybranych warstw sieci



Hierarchia reprezentacji

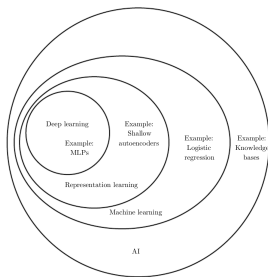


Rys: 📖 What are the limits of deep learning? by Mitchell Waldrop

Representation learning

Metody uczące się reprezentacji

metody uczenia maszynowego posiadające mechanizmy automatycznego tworzenia takiej reprezentacji danych (ekstrakcji cech), która istotnie poprawia zdolność algorytmów w rozwiązywaniu poszczególnych zadań.



Głębokie vs. płytkie uczenie

Krótką (nieformalna) definicja: głębokie uczenie, gdy model ma więcej niż 2 nieliniowości w najkrótszej ścieżce prowadzącej od wejścia do wyjścia

Płytkie modele

- do 2 warstw (nieliniowych transformacji), np. regresja logistyczna, SVM, MLP z jedną warstwą ukrytą
- wymagają odpowiednio przygotowanych cech

Głębokie modele

- więcej niż 2 warstwy przetwarzania, np. wielowarstwowe MLP, sieci rekurencyjne RNN, sieci splotowe CNN
- model hierarchiczny, wiele transformacji od wejścia do wyjścia
- każda „warstwa” tworzy reprezentację danych na innym poziomie abstrakcji
- możliwość pracy na „surowych” danych → **modele end-to-end**

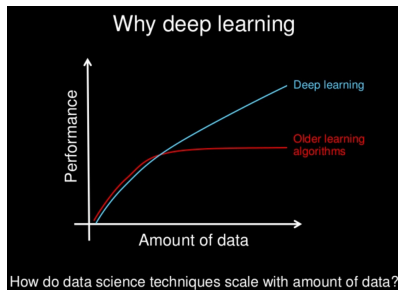
Po co nurkować w głąb?

- **Uniwersalny aproksymator:** sieć MLP z jedną warstwą ukrytą jest w stanie aproksymować dowolną funkcję z dowolną dokładnością
- Głębokie uczenie trudniejsze w realizacji od uczenia płytkich modeli, problemy nie tylko z treningiem ale z interpretacją działania modeli

Po co więc tworzyć głębsze sieci?

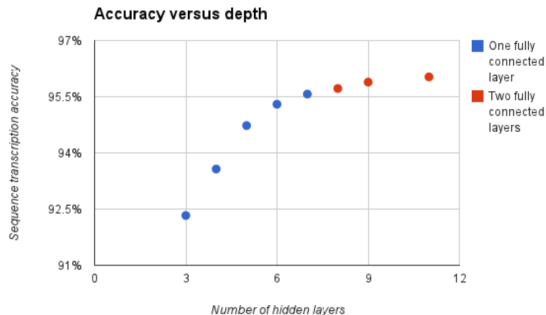
- „Płytke” modele wymagają o wiele (wykładniczo) więcej neuronów do osiągnięcia tej samej dokładności co głębokie modele
- „Płytke” a szerokie modele łatwiej się przetrenowują, znalezienie rozwiązania złożonego jest trudniejsze niż w modelach głębszych
- Złożone problemy wymagają złożonych modeli

Skalowalność wyników



- duża dostępna ilość danych (z etykietami) - Big Data, sieci głębokie wymagają dużej liczby danych
- duża dostępna moc obliczeniowa (GPU, TPU), głębokie uczenie wiąże się z wykonywaniem wielu operacji mnożenia macierzy
- nowe metody pozwalające skutecznie uczyć głębokie modele

Głębokość a poprawność klasyfikacji



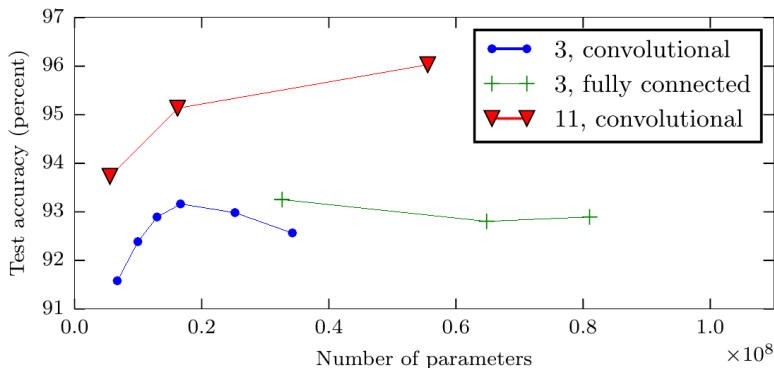
Street View House Numbers dataset (SVHN)
200k numerów domów



Głębokość niezbędna do zbudowania odpowiedniej reprezentacji i osiągnięcia lepszej poprawności

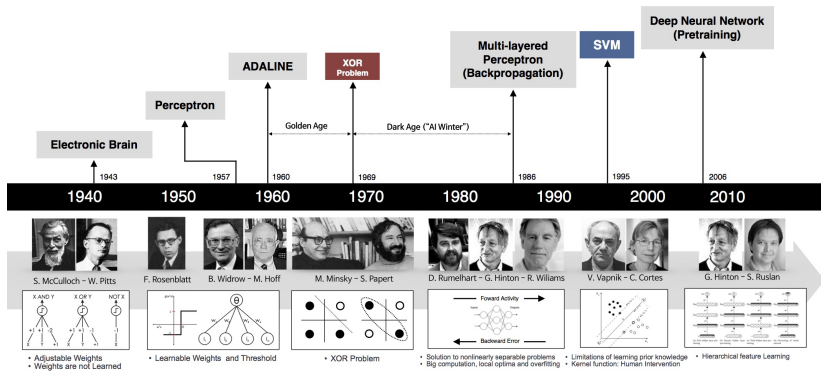
Goodfellow, et al. (2014) Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks

Głębokość ma znaczenie



Zwiększanie liczby parametrów płytkich modeli nie wystarczy do osiągnięcia poprawności modeli głębokich
Głębokość może mieć większe znaczenie od liczby parametrów modelu

Goodfellow, et al. (2014) Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks

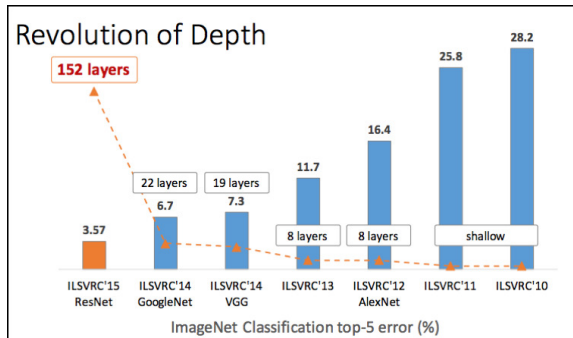


Geoffrey E. Hinton, et.al. „A fast learning algorithm for deep belief nets”, 2006
 R. Salakhutdinov, G. Hinton „Deep boltzmann machines”, 2009

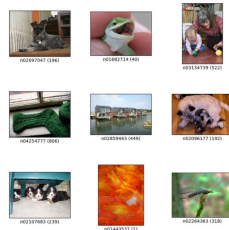
https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

State of art results

👉 ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



Dane ImageNet 2012:
klasyfikacja obrazów,
trening 1.26 M zdjęć,
1000 kategorii, poziom
człowieka $\approx$ 5% błędów



- klasyfikacja obrazów 👉 Are we there yet?
- rozpoznawanie mowy 👉 WER are we?
- 👉 Browse State-of-the-Art

Rys: 👉 The revolution of depth, Armando Vieira

Wszędzie AI

- rozpoznawanie obiektów (twarzy, emocji), etykietowanie i lokalizacja obiektów, segmentacja obrazów
- automatyczne tłumaczenie, analiza tekstu, mowy,
- modele języka, chatboty (ChatGPT, BERT)
- przewidywanie wyników wyborów, trzęsień ziemi, sekwencjonowanie białek (AlphaFold)
- sterowanie robotami, samochodami, gry komputerowe (AlphaGo, AlphaStar)
- kompresja sygnałów, rekonstrukcja obrazów, odszumianie sygnałów, kolorowanie czarno-białych filmów, zwiększanie rozdzielczości ...
- generowanie i synteza sygnałów: muzyki, obrazów (Modjourney, DALL-E2), tekstu, pisma ręcznego, mowy, kodu komputerowego,
- AI tworzące AI (Google AutoML), AI rozmawiające z AI (Google Duplex)
- 🖱️ 30 amazing applications of deep learning
- 🖱️ Browse State-of-the-Art

DNN

Głębokie MLP

MLP przypomnienie

Typy neuronów

- liniowe, sigmoidalne, tanh
- funkcje sigmoidalne nasycają się, co utrudnia optymalizację
- ReLU, Maxout - szybsza zbieżność, nie nasycają się

Optymalizacja metodami spadku gradientu lub innymi:

- wsteczna propagacja
- SGD, trening z momentem, RPROP
- metody 2 rzędu: Newtona, Levenberg-Marquarda, gradientów sprzężonych, kosztowne - wymagają obliczenia odwrotności Hessianu
- Adam, AdaGrad, moment Nesterova - nowe algorytmy uczenia

Metody regularyzacji:

- L_1 , L_2 , wczesne zatrzymanie
- dropout, batch normalization, gradient clipping, data augmentation,

Funkcje kosztu - przypomnienie

Błąd średniokwadratowy (MSE)

$$E = \frac{1}{N} \sum (f(\mathbf{x}) - y)^2$$

- problemy aproksymacji (wyjścia ciągłe)
- liniowe neurony wyjściowe

Entropia krzyżowa, Cross-Entropy (CE)

$$E = - \sum y \log f(\mathbf{x})$$

- klasyfikacja (wyjścia binarne)
- softmax lub sigmoidalne neurony wyjściowe

Wsteczna propagacja błędu - przypomnienie

1. Zainicjuj wagi $w_{ij}^{(k)}$ małymi losowymi wartościami
2. Dopóki nie spełniony warunek stopu wykonuj
 - propagacja sygnału od wejścia do wyjścia

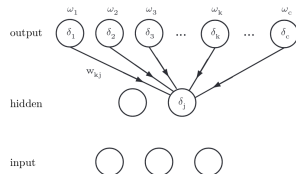
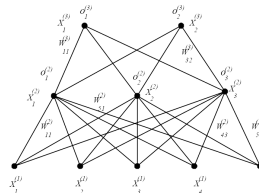
$$o_i^{(K)} = \sigma \left(\sum_j w_{ji}^{(K)} o_j^{(K-1)} \right)$$

- propagacja sygnału błędu od warstwy wyjściowej do wejściowej

$$\delta_i^{(K)} = \sum_j \delta_j^{(K+1)} w_{ij}^{(K)}$$

- aktualizuj wagi

$$\Delta w_{ij}^{(K)} = \eta \delta_j^{(K)} o_i^{(K-1)}$$



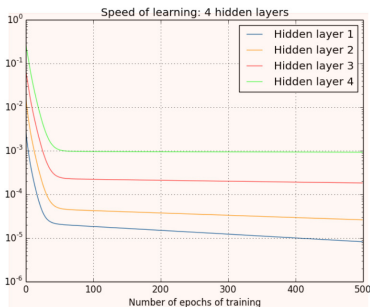
Problemy związane z uczeniem (głębokich) sieci

- **zanikający gradient** - błąd propagowany od warstwy wyjściowej zanika przy niższych warstwach
- funkcje ograniczone (sigmoidalna, tangens hiperboliczny) „nasycają się” i posiadają niezerowy gradient tylko w wąskim przedziale aktywacji bliskim 0
- **przeuczenie** - większa liczba parametrów sprzyja przetrenowaniu, wymagane są techniki regularyzacyjne
- większa ilość parametrów + duże dane → długi czas uczenia i duże wymagania dotyczące pamięci

Zanikający gradient

Przykład:

MNIST data, MLP 784x30x30x30x30x10, regularyzacja $\|\mathbf{w}\|^2$

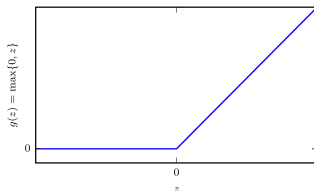


Szybkość uczenia $\|\delta\|$ drastycznie maleje w głębszych warstwach

Nielsen, 2016 [2]

Rectified linear unit (ReLU)

$$g(z) = \max(0, z), \quad g'(z) = \begin{cases} 1 & \text{dla } z > 0 \\ 0 & \text{dla } z < 0 \end{cases}$$



X. Glorot, A. Bordes and Y. Bengio (2011). Deep sparse rectifier neural networks

Korzyści ReLU

- gradient nie znika, gdy jednostka jest aktywna, ograniczenie problemu znikającego gradientu
- fragmentami liniowa, skuteczniejsza optymalizacja metodami gradientowymi
- efektywna obliczeniowo: mnożenie, dodawanie, warunek
- głębokie sieci ReLU nie wymagają pre-treningu (Glorot, 2011)
- rzadka reprezentacja (*sparse representation*), tylko część jednostek jest aktywna w czasie przetwarzania sygnału

X. Glorot, A. Bordes and Y. Bengio (2011). Deep sparse rectifier neural networks

Sparse representation

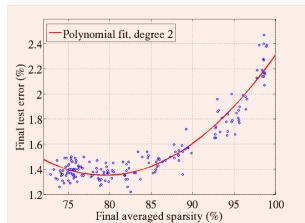
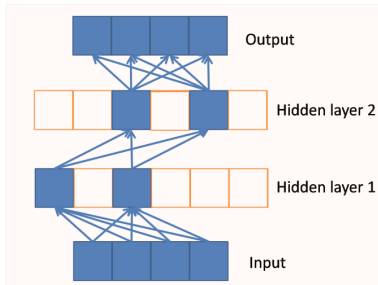


Figure 3: **Influence of final sparsity on accuracy.** 200 randomly initialized deep rectifier networks were trained on MNIST with various L_1 penalties (from 0 to 0.01) to obtain different sparsity levels. Results show that enforcing sparsity of the activation does not hurt final performance until around 85% of true zeros.

Rzadka reprezentacja dodaje nadmiarowość do modelu, zmiana wywołana przez sygnał wejściowy wprowadza zmiany tylko w podzbiorze neuronów

ReLU networks results

Sieci z jednostkami ReLU uzyskują dobre wyniki bez konieczności stosowania pre-treningu, czyli metody ustalenia początkowych wartości wag w procesie wstępnego treningu (zazwyczaj nienadzorowanego)

Neuron	MNIST	CIFAR10	NISTP	NORB
<i>With</i> unsupervised pre-training				
Rectifier	1.20%	49.96%	32.86%	16.46%
Tanh	1.16%	50.79%	35.89%	17.66%
Softplus	1.17%	49.52%	33.27%	19.19%
<i>Without</i> unsupervised pre-training				
Rectifier	1.43%	50.86%	32.64%	16.40%
Tanh	1.57%	52.62%	36.46%	19.29%
Softplus	1.77%	53.20%	35.48%	17.68%

Problemy ReLU

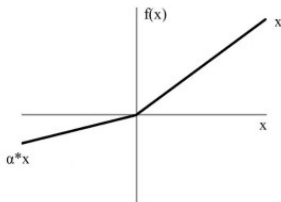
- nie jest różniczkowalne dla $z = 0$
- nie jest symetryczna i wycentrowana w 0
- nieograniczona, może powodować nieograniczony wzrost wartości aktywacji
- **problem umierających ReLU** - gdy neurony osiągną stan permanentnego braku aktywacji $g(z) = 0$ i ich wagi nie są modyfikowane w trakcie uczenia
- druga pochodna równa 0, utrudnia to stosowanie metod gradientowych 2 rzędu

Uogólnienia ReLU

- **Absolute ReLU** (Jarret, 2009), $g(z) = |z|$ skuteczna w szczególnych zastosowaniach dla obrazów z symetrią
- **Leaky ReLU** (Maas, 2013), wprowadza małą (np. $\alpha = 0.01$) wartość aktywacji dla $z < 0$

$$g(z) = \begin{cases} z & \text{dla } z > 0 \\ \alpha z & \text{dla } z < 0 \end{cases}$$

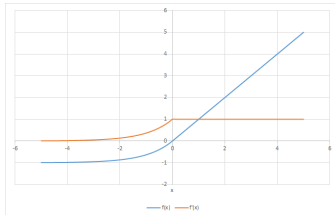
- **Parametric ReLU** odmiana Leaky ReLU, gdzie α podlega optymalizacji w trakcie uczenia



Modyfikacje ReLU

- **Noisy ReLU** dodaje niewielki szum z rozkładu normalnego
- **Exponential linear unit (ELU)** (Clevert, 2015), średnia aktywacja bliższa 0, szybsza zbieżność w stosunku do ReLU

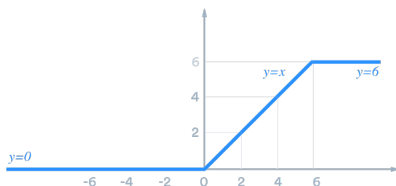
$$g(z) = \begin{cases} z & \text{dla } z > 0 \\ \alpha(e^z - 1) & \text{dla } z < 0 \end{cases} \quad g'(z) = \begin{cases} 1 & z > 0 \\ \alpha e^z, & z < 0 \end{cases}$$



Rys:  ELU as a Neural Networks Activation Function by Sefik

ReLU6

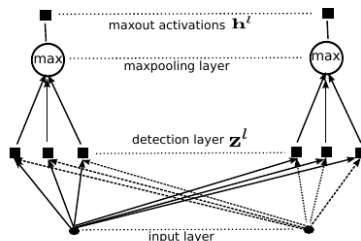
$$g(z) = \begin{cases} 6 & \text{dla } z > 6 \\ z & \text{dla } 0 < z < 6 \\ 0 & \text{dla } z < 0 \end{cases}$$



Górne ograniczenie przyspiesza tworzenie rzadkiej reprezentacji,
dobre wyniki na CIFAR-10 (Krizhevsky)

Maxout

$$g_i(\mathbf{z}) = \max_{j \in G_i} z_j$$



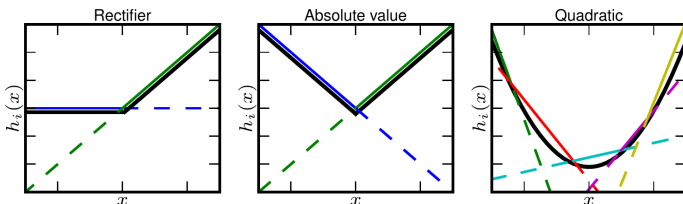
Zwraca wyjście najaktywniejszego neuronu z grupy k neuronów w warstwie, gdzie G_i to zbiór neuronów należących do i -tej grupy

$$g'_i(z_j) = \begin{cases} 1 & \text{dla } j = \arg \max_i z_j \\ 0 & \text{dla } j \neq \arg \max_i z_j \end{cases}$$

Rys: Paweł Świtojowski, *Investigation of maxout networks for speech recognition*.
Goodfellow, et al, (2013). "Maxout Networks". *JMLR WCP*. 28 (3): 1319–1327.

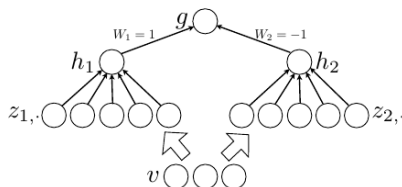
Maxout

- realizuje funkcję wypukłą fragmentami liniową, złożoną z k fragmentów, w efekcie uczony jest kształt funkcji aktywacji



- ReLU i PReLU to szczególne przypadki Maxout
- fragmentami liniowa
- rzadka reprezentacja: większość neuronów ma niezerowe wyjścia ale tylko sygnał zwycięzcy z grupy jest propagowany, sygnał błędu również jest propagowany tylko przez jednostkę zwycięską

Sieć Maxout



- złożenie funkcji wypukłych pozwala modelować dowolną funkcję ciągłą
- sieć z neuronami maxout jest uniwersalnym aproksymatorem
- aktywacja zależy od grypy wag, więc wprowadza do modelu redundancję, która pozwala niwelować fenomen „katastrofalnego zapominania”, gdy sieć zapomina wyuczone wcześniej wzorce

Rys: Goodfellow, et al, (2013). "Maxout Networks". JMLR WCP. 28 (3): 1319–1327.

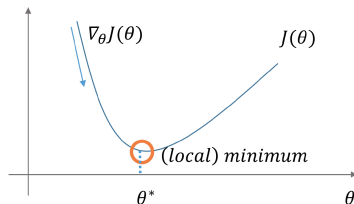
Maxout netowrk on MNIST

METHOD	TEST ERROR
RECTIFIER MLP + DROPOUT (SRI-VASTAVA, 2013)	1.05%
DBM (SALAKHUTDINOV & HINTON, 2009)	0.95%
Maxout MLP + dropout	0.94%
MP-DBM (GOODFELLOW ET AL., 2013)	0.91%
DEEP CONVEX NETWORK (YU & DENG, 2011)	0.83%
MANIFOLD TANGENT CLASSIFIER (RIFAI ET AL., 2011)	0.81%
DBM + DROPOUT (HINTON ET AL., 2012)	0.79%

Goodfellow, Maxout Networks, et al., 2013

Optymalizacja spadkiem gradientu

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E$$
$$E = \frac{1}{N} \sum_{\mathbf{x}} E(f(\mathbf{x}), y)$$



- **Batch gradient descent** - dla całego zbioru, niepraktyczne dla dużych danych
- **SGD on-line** - stochastic gradient descent dla $N = 1$, duża wariancja
- **SGD mini-batch** estymacja błędu na podstawie paczki zawierającej n losowych przypadków

Jak dobrać stałą uczenia η ?

Rys: S. Ruder, „Optimising for Deep Learning”

SGD z momentem (Polyak, 1964)

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

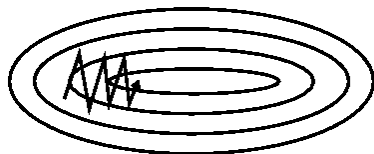
- $v(t) = \Delta w(t)$ prędkość uczenia akumuluje historyczne wartości gradientów (średnia krocząca)

$$v(t) = \gamma v(t-1) - \eta \nabla E(t)$$

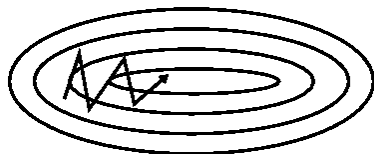
$$w(t+1) = w(t) + v(t)$$

- pęd zwiększa krok, gdy gradient nie zmienia kierunku
- redukuje oscylacje (spowalnia uczenie), gdy gradient zmienia kierunek
- gdy $\nabla E = 0$ wagi są nadal modyfikowane (bezwładność)
- typowo współczynnik $\gamma = 0.9$

Przyspieszenie zbieżności SGD za pomocą momentu



SGD bez momentu



SGD z momentem

Nesterov's accelerated gradient (NAG)

- przyspieszenie uczenia metodą momentu Nesterova (1968)

$$\begin{aligned}v(t) &= \gamma v(t-1) - \eta \nabla E(w(t) + \gamma v(t-1)) \\w(t+1) &= w(t) + v(t)\end{aligned}$$

- gradient $\nabla E(\hat{w}(t))$ jest liczony dla przybliżonej przyszłej pozycji

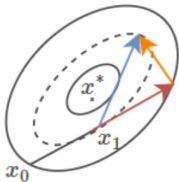
$$\hat{w}(t) = w(t) + \gamma v(t-1)$$

- trening szybciej jest w stanie zareagować na zmiany gradientu

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. Doklady ANSSSR (translated as Soviet.Math.Docl.), vol. 269, pp. 543–547.

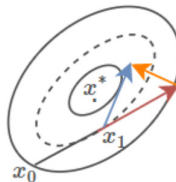
Polyak vs Nesterov

Polyak's Momentum



$$x_{t+1} = x_t - \alpha \nabla f(x_t) + \mu(x_t - x_{t-1})$$

Nesterov Momentum



$$x_{t+1} = x_t + \mu(x_t - x_{t-1}) - \gamma \nabla f(x_t + \mu(x_t - x_{t-1}))$$

Dla małych wartości η metody stają się równoważne

Rys: Ioannis Mitliagkas, *Theoretical principles for deep learning*, 2018

AdaGrad (Duchi, 2011)

$$w(t+1) = w(t) - \frac{\eta}{\sqrt{v(t)}} \nabla E(t)$$

gdzie $v(t)$ akumuluje historyczne wartości gradientów

$$v(t) = v(t-1) + [\nabla E(t)]^2 = \sum_{t' < t} [\nabla E(t')]^2$$

Efektywny współczynnik uczenia $\frac{\eta}{\sqrt{v(t)}}$ jest ustalana dla każdego optymalizowanego parametru niezależnie

- strome spadki funkcji błędu $\Rightarrow$ duży gradient $\Rightarrow$ mniejszy krok uczenia, szybka zbieżność dla funkcji wypukłych
- małe gradienty $\Rightarrow$ większy krok uczenia przeciwdziała efektowi znikającego gradientu
- domyślna wartość $\eta = 0.01$, zazwyczaj nie wymaga zmiany
- wada: mianownik może szybko przybrać duże wartości, co powoduje przedwczesny zanik uczenia (problem ten niwelują algorytmy RMSProp, Adadelta, Adam)

RMSProp (Hinton, 2012)

$$w(t+1) = w(t) - \frac{\eta}{\sqrt{v(t)}} \nabla E(t)$$

gdzie $v(t)$ jest wykładniczo ważoną średnią kroczącą kwadratów gradientów (EMA, *exponential moving average*)

$$v(t) = \beta v(t-1) + (1 - \beta) [\nabla E(t)]^2$$

- $RMS(\nabla E(t)) = \sqrt{v(t)}$ estymacja RMS (*root mean square*)
- modyfikacja AdaGrad zapewniająca lepszą zbieżność dla nie-wypukłych funkcji
- największy wpływ na wielkość $v(t)$ mają gradienty z ostatnich iteracji
- β kontroluje skalę długości ruchomej średniej, typowo $\beta = 0.9$ (średnia z 10 kroków)
- dobre wyniki w połączeniu z momentem Nesterova

Adadelta (Zeiler, 2012)

$$w(t+1) = w(t) - \frac{\sqrt{m(t)}}{\sqrt{v(t)}} \nabla E(t)$$

gdzie $\sqrt{v(t)} = RMS(\nabla E(t))$ wyznaczany średnią kroczącą

$$v(t) = \beta v(t-1) + (1 - \beta) [\nabla E(t)]^2$$

$\sqrt{m(t)} = RMS(\Delta w(t-1))$ wyznaczany średnią kroczącą kwadratów zmian parametrów Δw przybliżoną wartością z poprzedniej iteracji

$$m(t) = \beta m(t-1) + (1 - \beta) [\Delta w(t-1)]^2$$

- brak globalnej stałej uczenia η ale należy dobrać współczynnik β (typowo $\beta = 0.9$)
- metoda jest heurystycznym przybliżeniem metod kwadratowych (bez wyznaczania Hessianu)
- krok Δw wykonywany jest w tych samych hipotetycznych jednostkach (gdzie inne metody 1 rzędu $\nabla E \approx \frac{1}{w}$ nie zachowują jednostki)

Adaptive Moment Estimation (Adam)

$$w(t) = w(t-1) - \frac{\eta}{\sqrt{\hat{v}(t)}} \hat{m}(t)$$

Estymaty pierwszego (średnia) i drugiego (wariancja) momentu gradientów

$$m(t) = \beta_1 m(t-1) + (1 - \beta_1) \nabla E(t)$$

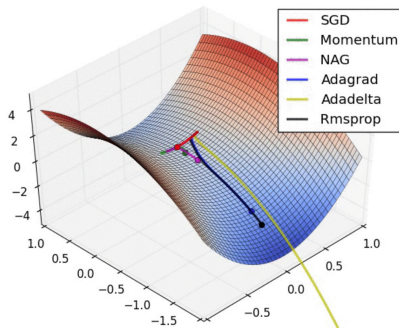
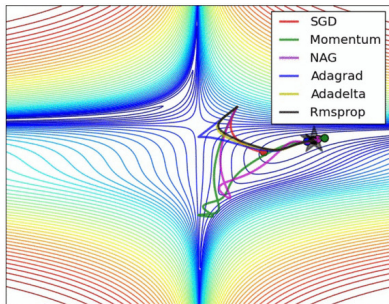
$$v(t) = \beta_2 v(t-1) + (1 - \beta_2) [\nabla E(t)]^2$$

Poprawka na odchylenie w stronę wartości 0 (z powodu zerowych początkowych wartości estymaty)

$$\hat{m}(t) = \frac{m(t)}{1 - \beta_1}, \quad \hat{v}(t) = \frac{v(t)}{1 - \beta_2}$$

współczynniki zanikania $\beta_1 = 0.9$, $\beta_2 = 0.999$
stała uczenia $\eta = 0.01$

Porównanie



👉 An overview of gradient descent optimization algorithms

Podsumowanie

- DNN dobre do analizy danych bez wyraźnej struktury np. przestrzennej lub czasowej (sekwencje)
- w większości zastosowań aktywacja ReLU sprawdza się bardzo dobrze oraz algorytm adam z domyślnymi parametrami η, β_1, β_2
- SGD z zanikającą wykładniczo wartością stałej uczenia może dorównać algorytmom takim jak adam