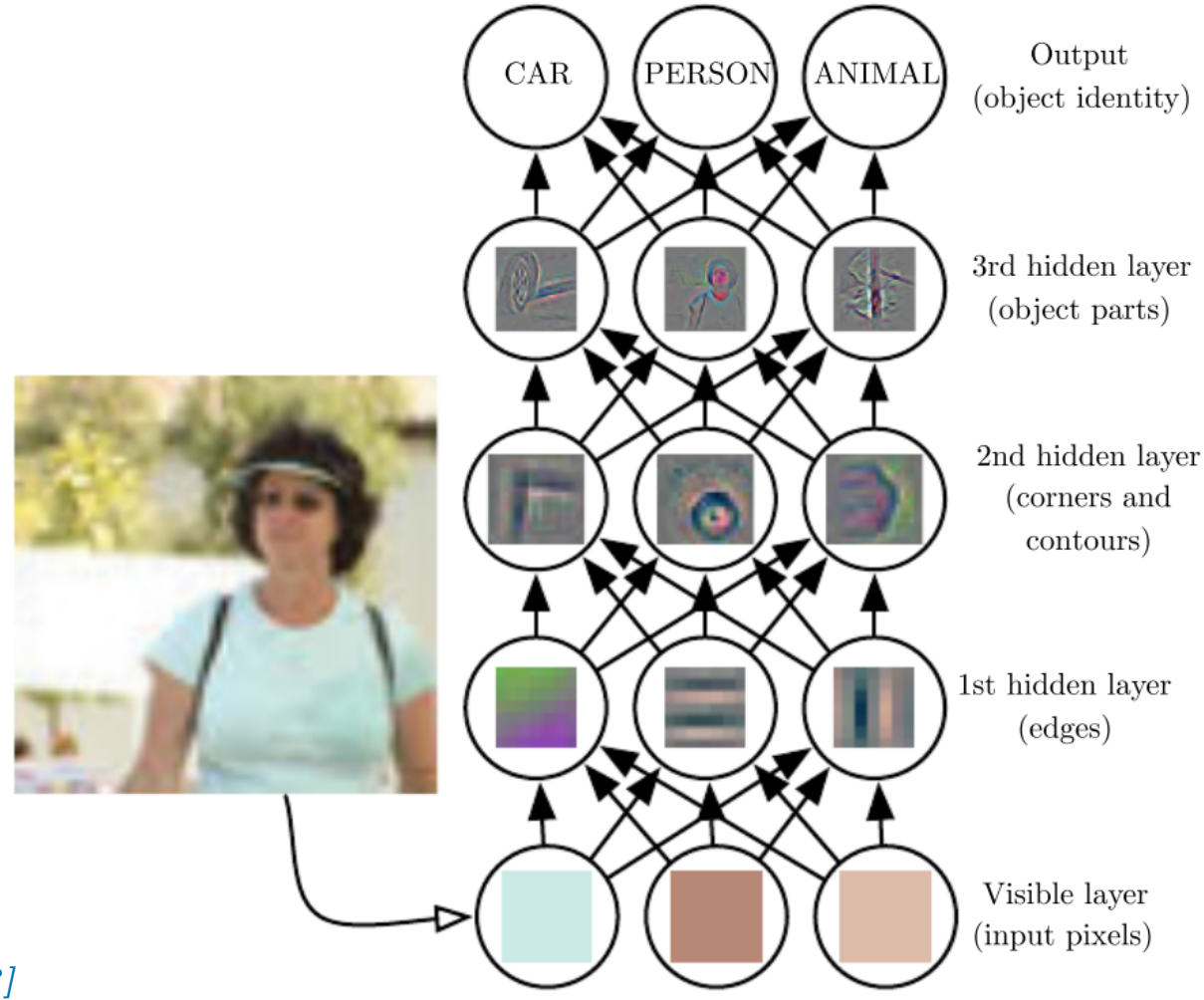
Głebokie uczenie: co to takiego?

- **Głębokie uczenie** - metody uczenia maszynowego, które zbudowane są z **wielu warstw realizujących nieliniowe transformacje**.
- Warstwy odwzorowują kolejne poziomy abstrakcji tworząc **model hierarchiczny**):
  najniższe warstwy tworzą reprezentuję najprostszych cech z sygnałue wejściowego, wyższe warstwy - generują bardziej ogólne koncepty bazując na relacjach z poprzednich warstw.

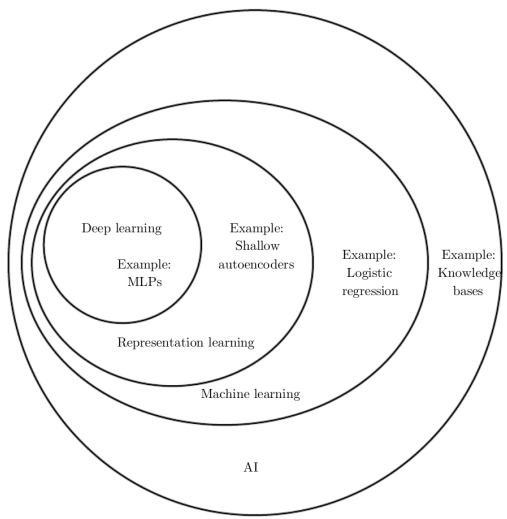
Hierarchia reprezentacji



Gooffellow, 2016 [?]

Representation learning

Metody uczące się reprezentacji (*representaion learning*) - *metody uczenia maszynowego posiadające mechanizmy automatycznego tworzenia takiej reprezentacji danych (ekstrakcji cech), która istotnie poprawia zdolność algorytmów w rozwiązywaniu poszczególnych zadań.*



Gooffellow, 2016 [?]

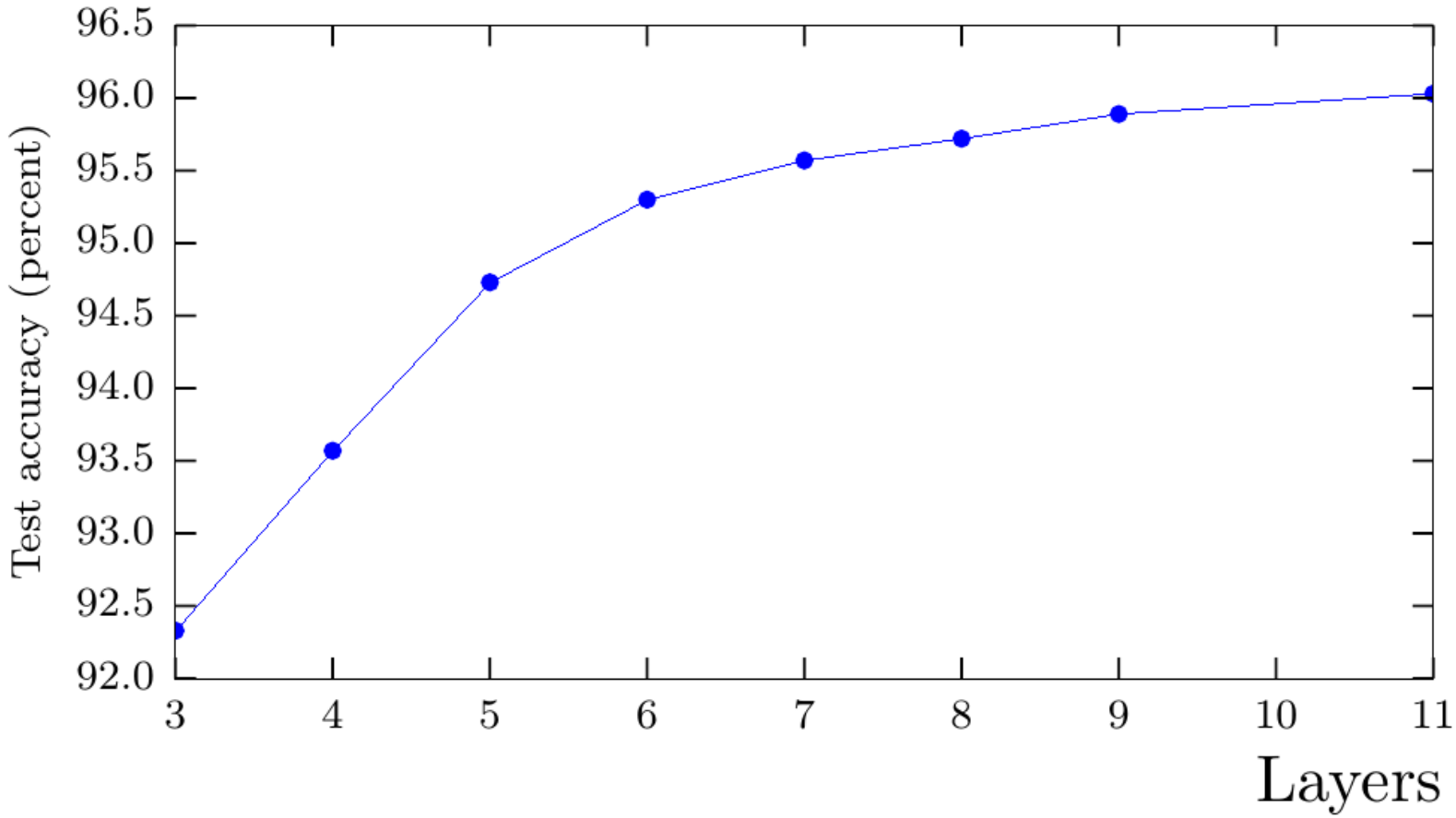
Głębokie vs. płytkie uczenie

- Krótka (nieformalna) definicja:
  głębokie uczenie, gdy model ma więcej niż 2 nieliniowośc
- **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

Po co nurkować w głąb?

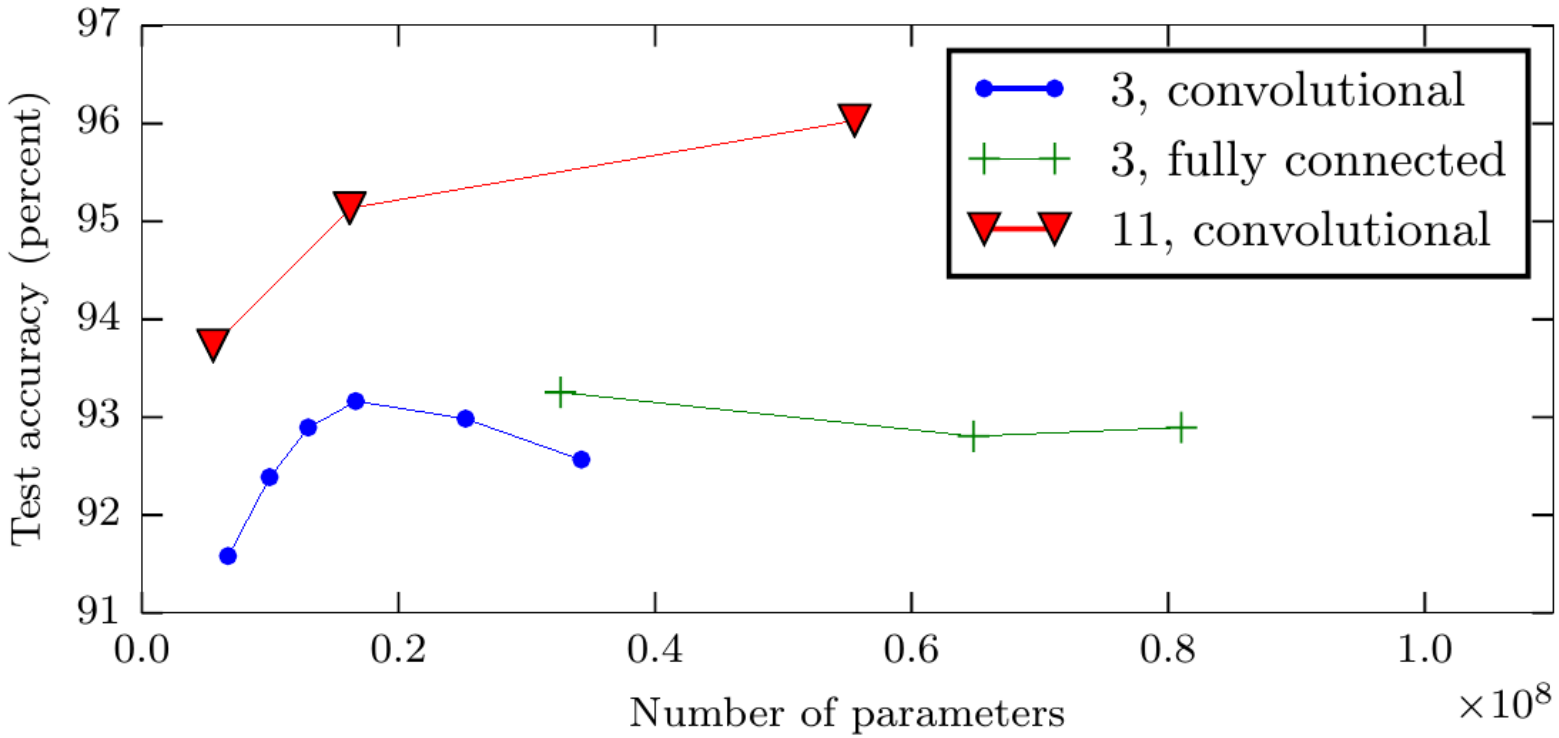
- **Uniwersalny aproksymator:** sieć MLP z jedną warstwą ukrytą jest w stanie aproksymować dowolną funkcję z dowolną dokładnością
- Deep learning też nie taki prosty, problemy nie tylko z treningiem ale z interpretacją działania
- Po co więc tworzyć głębsze sieci?
- „Płytkie" modele wymagają o wiele (wykładniczo) więcej neuronów
- „Płytkie" a szerokie modele łatwiej się przetrenowują, nie są w stanie odszukać lepszego rozwiązania

Głębokość a poprawność klasyfikacji

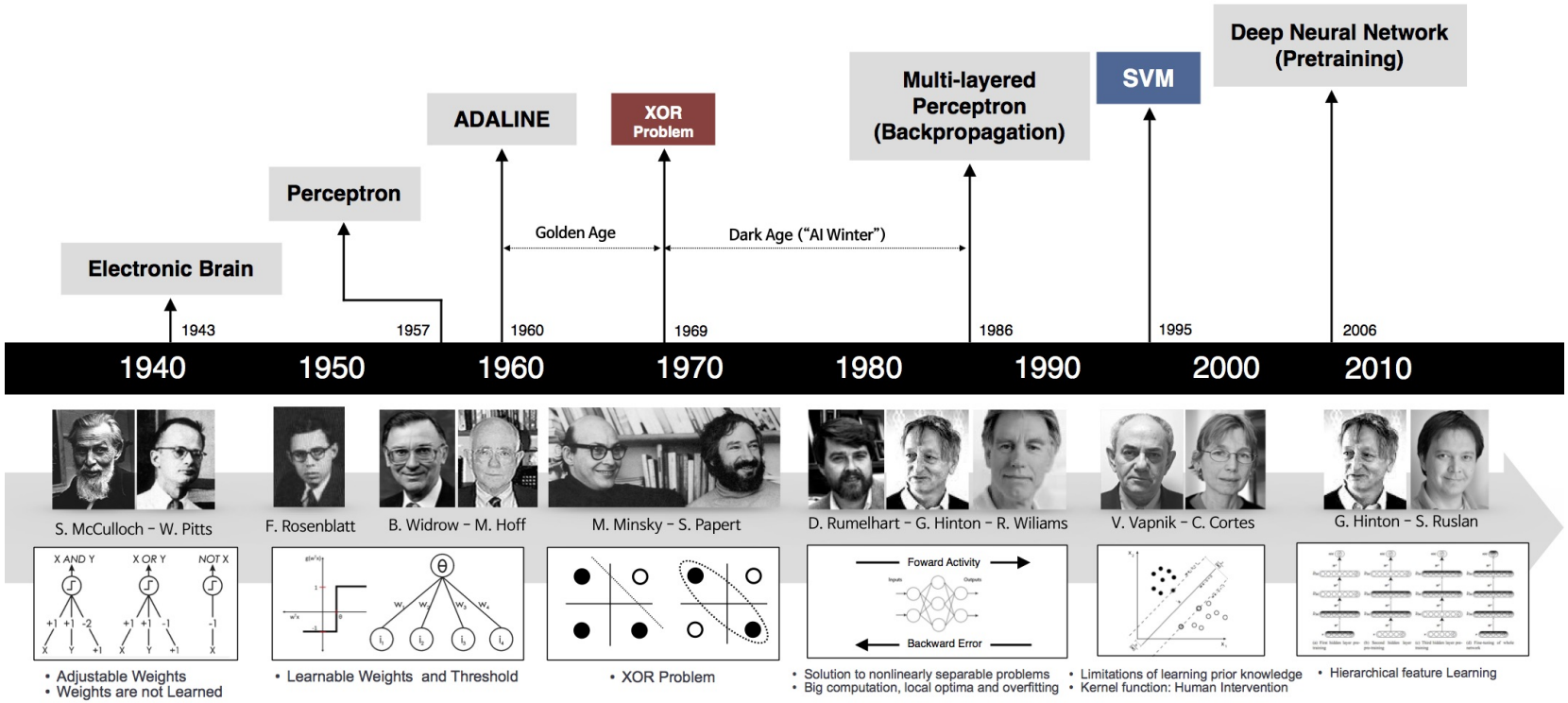


Gooffellow, 2016 [?]

Głębokość ma znaczenie



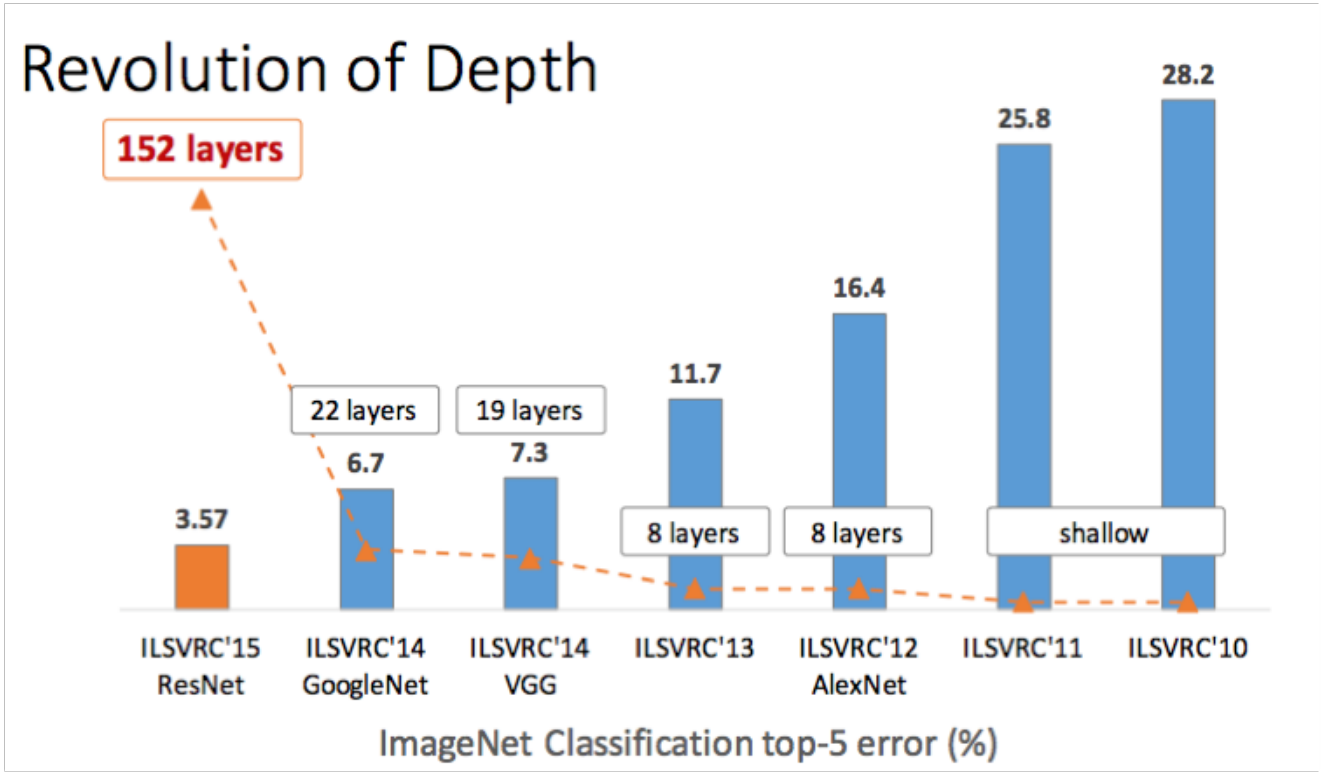
Gooffellow, 2016 [?]



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

State of art results

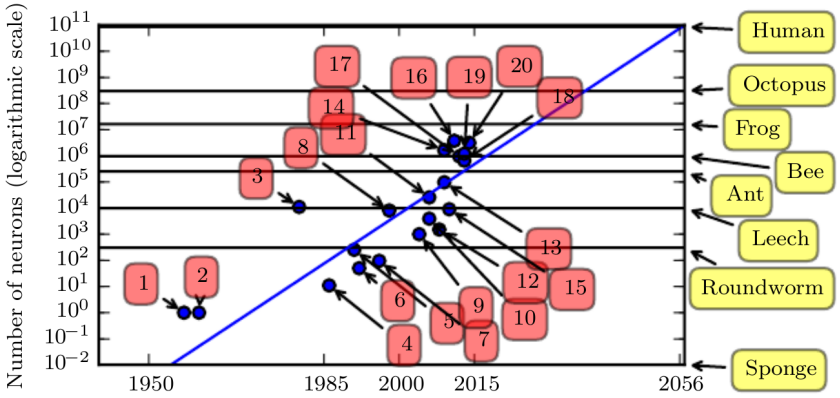
ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



☞ Are we there yet ?

☞ *The revolution of deepth, Armando Vieira*

Ilość neuronów w modelach neuronowych



1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart *et al.*, 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition (Bengio *et al.*, 1991)
7. Mean field sigmoid belief network (Saul *et al.*, 1996)
8. LeNet-5 (LeCun *et al.*, 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton *et al.*, 2006)
11. GPU-accelerated convolutional network (Chellapilla *et al.*, 2006)
12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina *et al.*, 2009)
14. Unsupervised convolutional network (Jarrett *et al.*, 2009)
15. GPU-accelerated multilayer perceptron (Ciresan *et al.*, 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le *et al.*, 2012)
18. Multi-GPU convolutional network (Krizhevsky *et al.*, 2012)
19. COTS HPC unsupervised convolutional network (Coates *et al.*, 2013)
20. GoogLeNet (Szegedy *et al.*, 2014a)

Liczba neuronów podwaja się co 2.5 roku

Gooffellow, 2016 [?]

Wszędzie AI

- rozpoznawanie obiektów (twarzy, emocji), etykietowanie i lokalizacja obiektów
- automatyczne tłumaczenie, analiza tekstu, mowy, chatboty
- przewidywanie wyników wyborów, trzęsień ziemi, ...
- sterowanie robotami, samochodami, gry komputerowe (AlphaGo)
- kompresja sygnałów, rekonstrukcja odszumianie sygnałów, odszumianie, kolorowanie czarno-białych filmów, ...
- generowanie i synteza sygnałów: muzyki, obrazów, tekstu, pisma ręcznego, mowy, kodu komputerowego, ...
- AI tworzące AI (Google AutoML), AI rozmiawiające z AI (Google Duplex)
- ☞ 30 amazing applications of deep learning

DNN

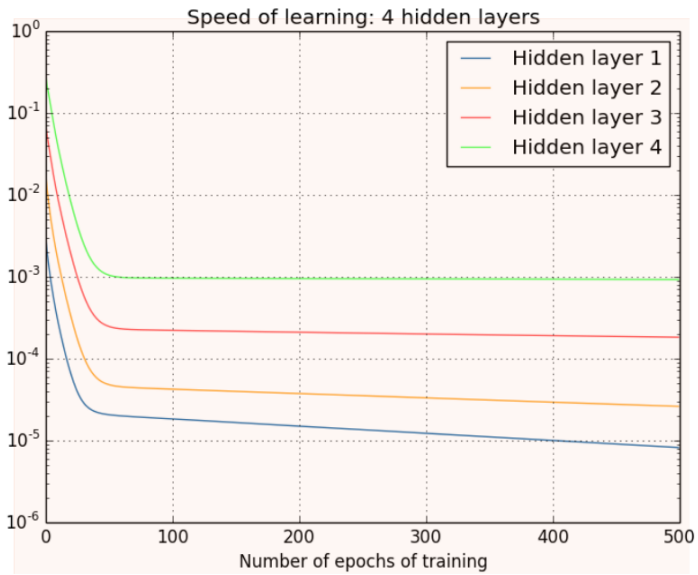
Głębokie MLP

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 tyko w wąskim przedziale aktywacji bliskim 0
- **przeuczenie** - większa liczba parametrów sprzyja przetrenowaniu, wymagane są techniki regularyzacyjne
- funkcja MSE jako funkcja błędu nie jest najlepszym wyborem w problemach klasyfikacji, wybór funkcji kosztu uzależniony od typu neuronów wyjściowych
- większa ilość parametrów + duże dane $\rightarrow$ długi czas uczenia i duże wymagania dotyczące pamięci

Zanikający gradient

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



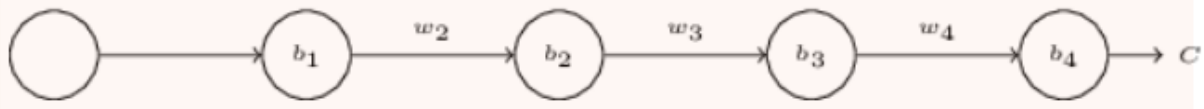
Szybkość uczenia $||\blacktriangleleft||$ drastycznie maleje w głębszych warstwach

Nielsen, 2016 [?]

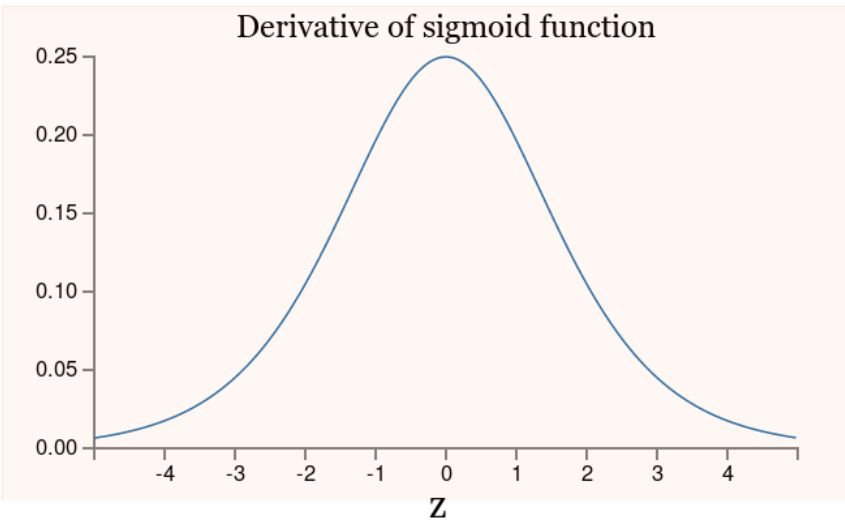
Znikający gradient

Przykład: sieć 1 x 1 x 1 x 1

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



gdy $|w_j| < 1$ wówczas $|w_j \sigma'(z_j)| < \frac{1}{4}$



Nielsen, 2016 [?]

Problem niestabilnego gradientu

- gradient w niższych warstwach zależy od wartości wag, aktywacji i gradientów w warstwach wyższych
- **eksplodujący gradient**
  gdy $w_i >> 1$ oraz $z_i \approx 0$ wówczas $|w_i \sigma'(z_i)| > 1$, może to owocować bardzo dużymi gradientami w początkowych warstwach
- znikający i eksplodujący gradient to przypadki szczególne **problemu niestabilnego gradientu** -> wartości gradientów w poszczególnych warstwach mogą znacznie się różnić, warstwy uczą się z różnym tempem

Specyfikacja sieci i metody treningu

- Specyfikacja modelu:
  - specyfikacja architektury
  - typ neuronów ukrytych: liniowe, sigmoidalne, ReLU, ...
  - typ neuronów wyjściowych: liniowe, sigmoidalne, softmax, ...
- Funkcja kosztu:
  - mean squared error (MSE),
  - cross-entropy (CE),
- Optymalizacja metodami spadku gradientu lub innymi:
  - SGD,
  - Adam, AdaGrad, ...
- Metody regularyzacji:
  - momentum, dropout, L_1, L_2, batch normaliation, gradient clipping, data augmentation....

Funkcja kosztu MSE

Błąd średniokwadratowy (*Mean Squared Error)*

$$J(\Theta) = \frac{1}{N} \sum (\hat{y} - y)^2$$

Gradient funkcji kosztu:

$$\frac{\partial J}{\partial \Theta} = \frac{2}{N} \sum (\hat{y} - y) \frac{\partial \hat{y}}{\partial \Theta}$$

gdzie
$\hat{y}$ - wyjścia sieci,
Θ - parametry sieci,
y - oczekiwane wyjścia

Funkcja kosztu Cross Entropy

Cross entropy

$$J(\Theta) = -\sum y \log \hat{y}$$

- miara odległości miedzy rozkładami y i $\hat{y} \in (0, 1)$
- gdy $y \approx \hat{y}$ to funkcja kosztu bliska zeru
- $J > 0$

Gradient funkcji kosztu:

$$\frac{\partial J}{\partial \Theta} = -\sum y \frac{1}{\hat{y}} \frac{\partial \hat{y}}{\partial \Theta}$$

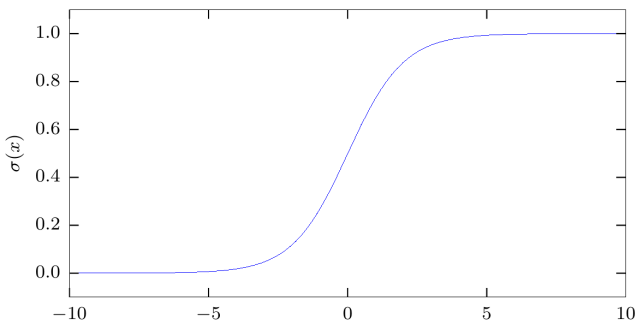
Funkcja liniowa w warstwie wyjściowej

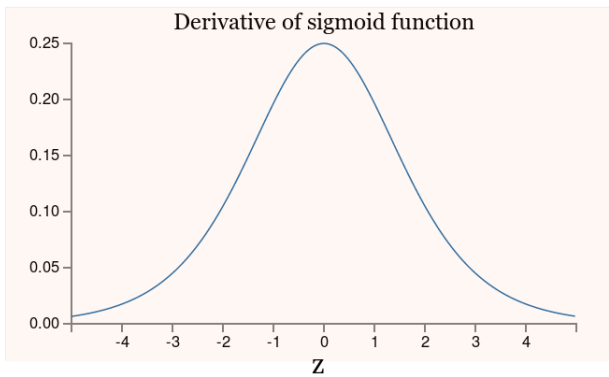
$$z = \mathbf{w}^T \mathbf{h} + b$$

- nieograniczone $(-\infty, +\infty)$, nie nasycają się ale bardzo duże wartości aktywacji również mogą przysporzyć problemów w trakcie uczenia
- stabilniejsze w optymalizacji metodami gradientowymi od innych typowych funkcji nieliniowych stosowanych w sieciach
- zastosowanie wraz funkcją kosztu MSE do problemów regresji (dane wyjściowe ciągłe)

Funkcja logistyczna w warstwie wyjściowej

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$





- ograniczone $(0, 1)$, nasycają się, aktywne uczenie tylko w okolicach $x \approx 0$
- zastosowanie: klasyfikacja 2 klas, modelowanie rozkładu Bernoullego, binarne wartości wyjściowe

Funkcja logistyczna w warstwie wyjściowej

- podczas minimalizacji MSE błąd zanika, gdy funkcja się nasyca (także dla niepoprawnej odpowiedzi) $\frac{\partial \hat{y}}{\partial \Theta} \approx 0$

$$\frac{\partial J_{MSE}}{\partial \Theta} = \frac{2}{N} \sum (\hat{y} - y) \frac{\partial \hat{y}}{\partial \Theta}$$

- problem znika przy zastosowaniu kosztu CE (na wyjściu oczekujemy 2 stanów: 0 lub 1)

$$J_{CE} = -\sum [y \ln \hat{y} + (1 - y) \ln(1 - \hat{y}))]$$

$$\frac{\partial J_{CE}}{\partial \Theta} = \sum (\hat{y} - y) \frac{\partial z}{\partial \Theta}$$

Softmax w warstwie wyjściowej

$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- wartości $(0, 1)$ unormowane

$$\sum_i softmax(\mathbf{z})_i = 1$$

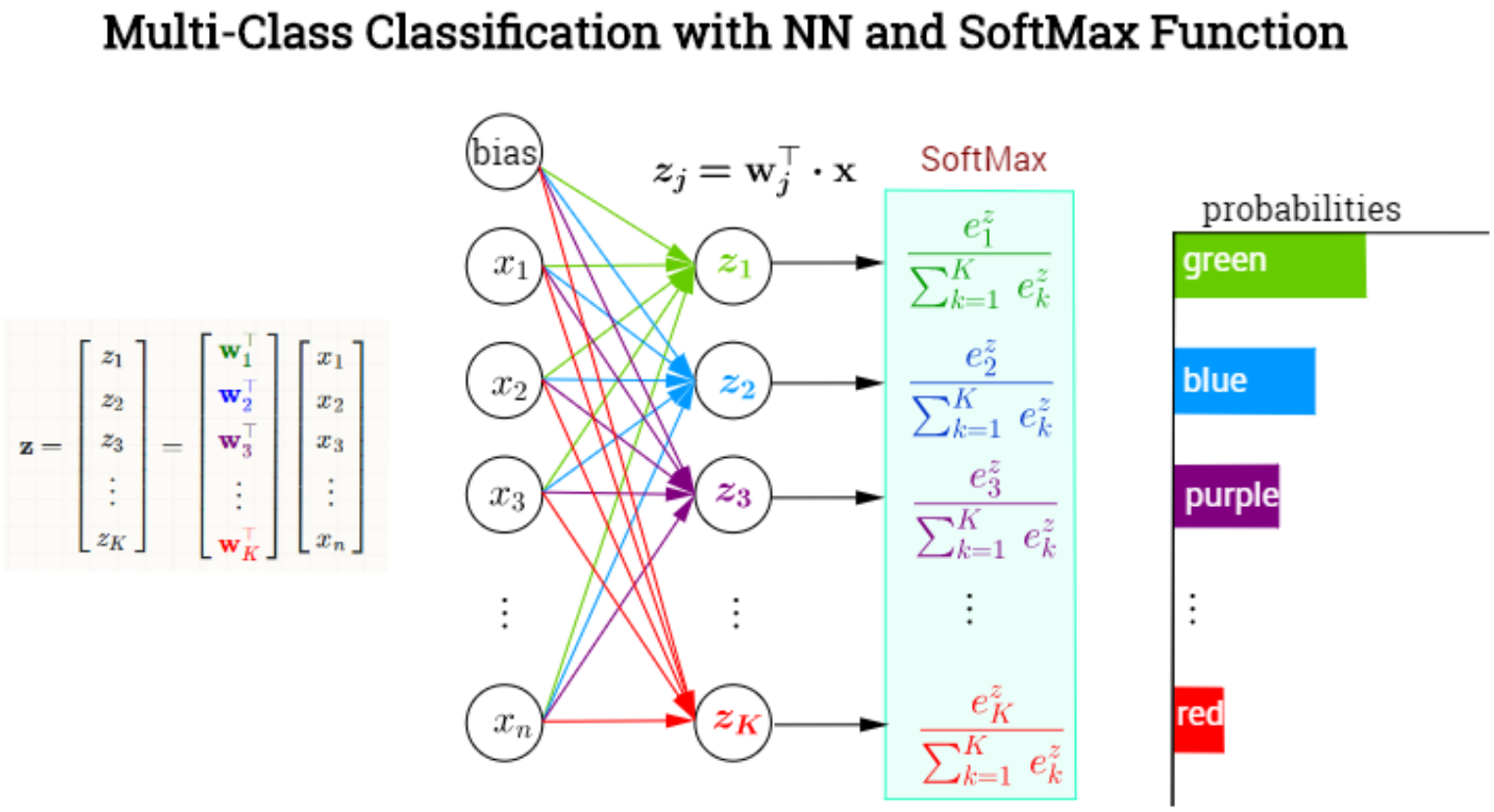
- Zastosowanie: klasyfikacja n rozłącznych klas, każde wyjście sieci związane jest z jedną klasą, wartości kodowane w postaci wektora binarnego *one hot*

$$[0, \ldots, 0, 1, 0, \ldots, 0]$$

- wartości wyjściowe sieci reprezentują rozkład prawdopodobieństw zmiennej wyjściowej

$$\hat{y}_i = P(y = i|X), \qquad i = 1, \ldots, n$$

Softmax w warstwie wyjściowej



Źródło:
stats.stackexchange.com/questions/265905/derivative-of-softmax-with-respect-to-weights

- gradient

$$\frac{\partial softmax(\mathbf{z})_i}{\partial z_j} = softmax(\mathbf{z})_i(\delta_{ij} - softmax(\mathbf{z})_i)$$

- gradient funkcji kosztu CE gdy $\hat{y}_i = softmax(\mathbf{z})_i$

$$\frac{\partial J_{CE}}{\partial \Theta} = \sum(\hat{y} - y)\frac{\partial z}{\partial \Theta}$$

- obliczenia się upraszczają i znika problem zanikania błedu dla nasyconych wyjść, który wystepował dla funkcji koszty MSE

- dobrze współgra z treningiem *cross entropy*, błąd nie znika nawet dla dużych wartości aktywacji z_i

$$\log softmax(\mathbf{z})_i = z_i - \log \sum_j e^{z_i}$$

- ważne są równice pomiędzy wyjściami a nie amplitudy poszczególnych wyjść

$$softmax(x) = softmax(x + c)$$

- minimalizacja kosztu CE powoduje zwiększenie wartości z_i na jednym z wyjść i zmniejszenie wszystkich pozostałych

$$\log \sum_j e^{z_i} \approx \max_i z_i$$

- z punktu widzenia neurobiologicznego softmax może być widziany jako realizacja mechanizmów hamowania występującego w grupach neuronów w korze, podobnie do mechanizmu *winner takes all*

Dobór funkcji kosztu uzależniony jest od rozkładu wartości wyjściowych i typu neuronów wyjściowych

Output Type	Output Distribution	Output Layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)
Continuous	Mixture of Gaussian	Mixture Density	Cross-entropy
Continuous	Arbitrary	See part III: GAN, VAE, FVBN	Various

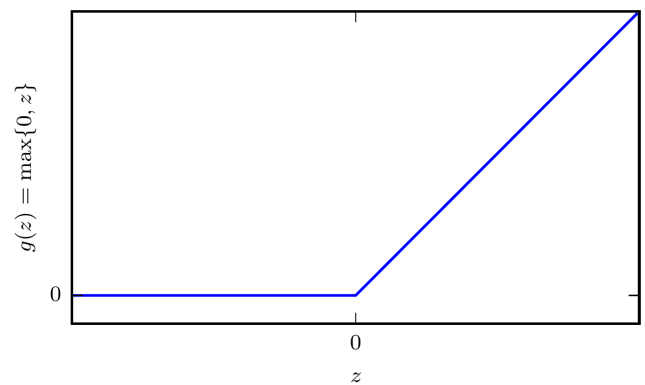
Gooffellow, 2016 [?]

Dobór neuronów ukrytych

- wybór funkcji realizowanych przez warstwy ukryte kluczowy dla przebiegu procesu uczenia
- funkcje liniowe - wiele warstw liniowych sprowadza się do pojedynczej transformacji liniowej
- funkcje ograniczone: sigmoidalne i tangens hiperboliczny powodują problemy z niestabilnym gradientem, uczenie może utknąć gdy funkcja się nasyci
- obecnie najpopularniejszym typem aktywacji w sieciach DNN jest ReLU
- inne typy neuronów: modyfikacje ReLU, Maxout, ...

Rectified linear unit (ReLU)

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



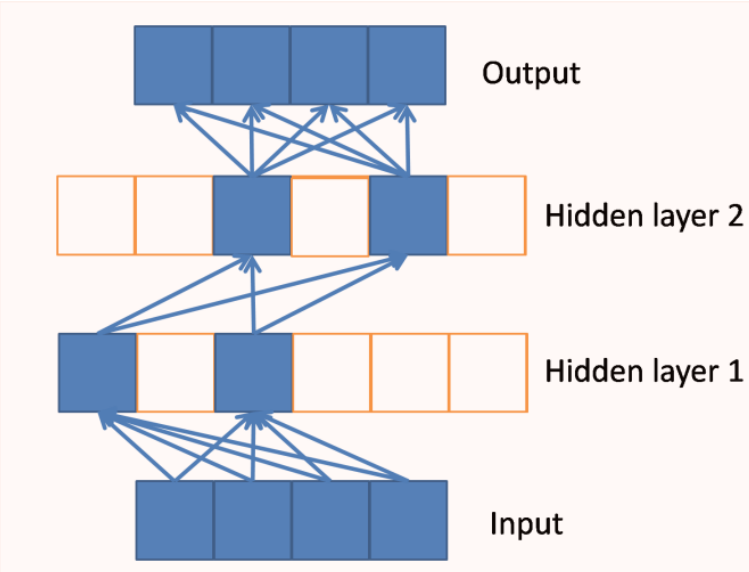
X. Glorot, A. Bordes and Y. Bengio (2011). Deep sparse rectifier neural networks

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 (Galrot, 2011)
- rzadka reprezentacja (*sparse representation*)

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

Sparse representaion



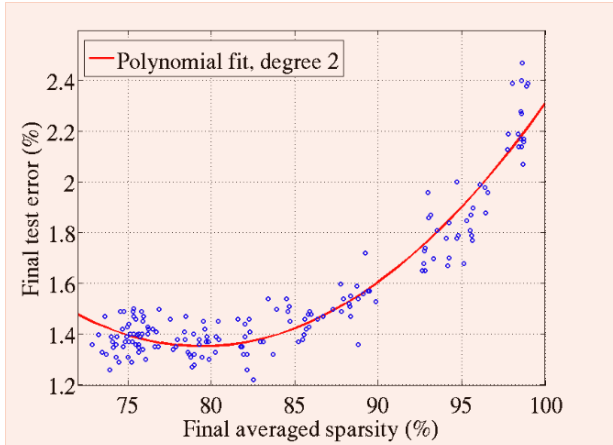


Figure 3: **Influence of final sparsity on accuracy.** 200 randomly initialized deep rectifier networks were trained on MNIST with various L_1 penalties (from 0 to 0.01) to obtain different sparsity levels. Results show that enforcing sparsity of the activation does not hurt final performance until around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

Rzadka reprezentacja dodaje redundancje do modelu, zmiana wywołana przez dany wektor treningowy wprowadza zmiany tylko w podzbiorze neuronów

ReLU networks results

Neuron	MNIST	CIFAR10	NISTP	NORB
With **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%
Without **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%

Garlot, Deep Sparse Rectifier Neural Networks, 2011

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 osiągają stan permanentnego braku aktywacji $g(z) = 0$ i ich wagi nie są modyfikowane w trakcie uczenia

Uogólnienia ReLU

- **Absolute ReLU** (Jarret, 2009), $g(z) = |z|$
- **Leaky ReLU** (Maas, 2013), wprowadza małą ($\alpha = 0.01$) wartość aktywacji dla $z < 0$

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

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

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

Maxout

Zwraca wyjście najaktywniejszego neuronu z grupy k neuronów w warstwie

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

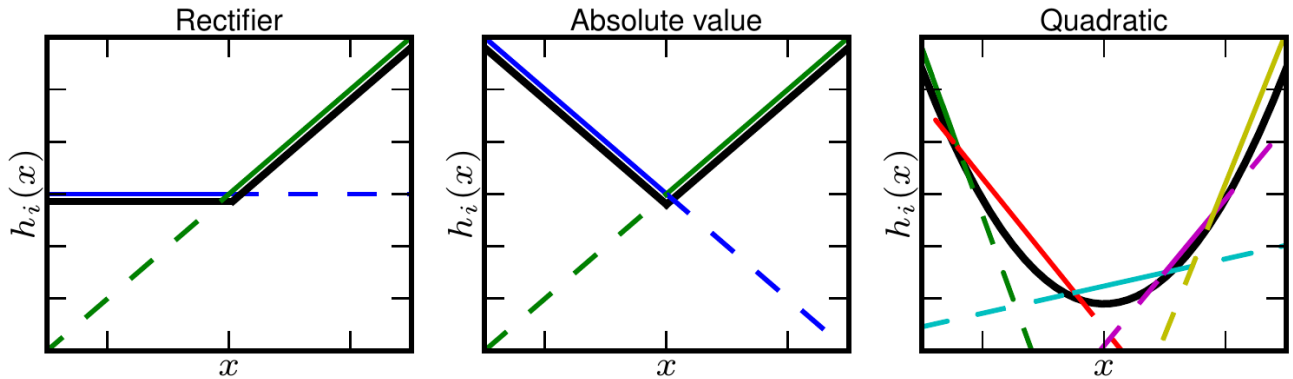
gdzie G_i to zbiór indeksów i-tej grupy $G_i = [(k-1)i + 1, \ldots, ki]$

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

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

Maxout

- realizuje funkcję wypukła fragmentami liniowa złożoną z k fragmentów



- ReLU i PReLU to szczególne przypadki Maxout
- sieć z nauronami maxout jest uniwersalnym aproksymatorem
- zależy od grypy wag, więc wprowadza do modelu redundancję, która pozwala niwelować fenomen „katastrofalnego zapominania”, gdy sieć zapomina wyuczone wcześniej wzorce

Maxout netowrk on MNIST

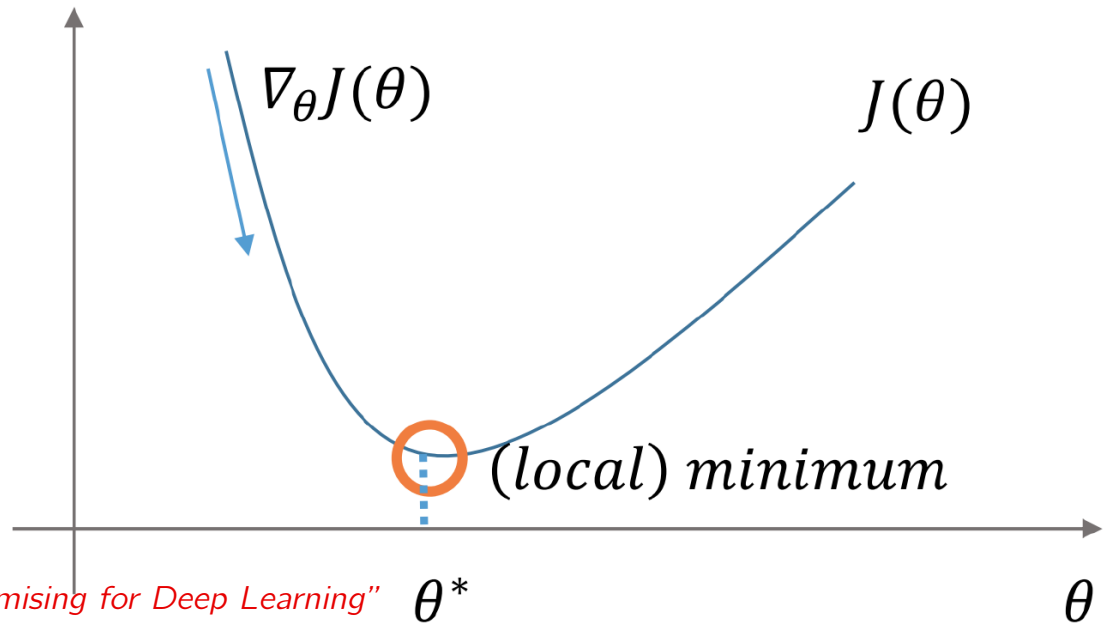
Method	Test error
Rectifier MLP + dropout (Srivastava, 2013)	1.05%
DBM (Salakhutdinov & Hinton, 2009)	0.95%
Maxout MLP + dropout	**0.94%**
MP-DBM (Goodfellow et al., 2013)	0.91%
Deep Convex Network (Yu & Deng, 2011)	0.83%
Manifold Tangent Classifier (Rifai et al., 2011)	0.81%
DBM + dropout (Hinton et al., 2012)	0.79%

Goodfellow, Maxout Networks, et al., 2013

Optymalizacja spadkiem gradientu

$$\Theta \leftarrow \Theta - \eta \nabla_{\Theta} J(\Theta)$$

$J(\Theta)$ funkcja kosztu,
Θ parametry modelu,
η stała uczenia



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

Optymalizacja spadkiem gradientu

$$J(\Theta) = \frac{1}{N} \sum_{\mathbf{x}} L(f(\mathbf{x}), y)$$

- **Batch gradient descent** - gradient dla całego zbiioru danych, nie praktyczne dla dużych danych
- **SGD** - stochastic gradient descent (on-line) - aktualizacja dla pojedynczego przypadku treningowego ($N = 1$), duża szybkość działania ale i duża wariancja
- **MiniBatch** gradient descent (często utożsamiany z SGD) - średnia z n losowych przypadków (*mini-batch*), trening on-line, mniejsza wariancja, możliwe operacje na macierzach, łatwiejsze zrównoleglenie GPU/CPU

Modyfikacje: momentum, Nestereov momentum, AdaGrad, AdaDelta, RMSProp, Adam, ..

SGD z momentem

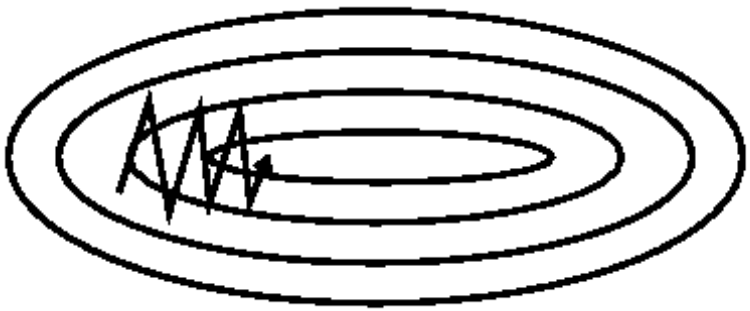
prędkość uczenia zależna od poprzednich wartości gradientów

$$v_t = \gamma v_{t-1} + \eta \nabla_\Theta J(\Theta)$$

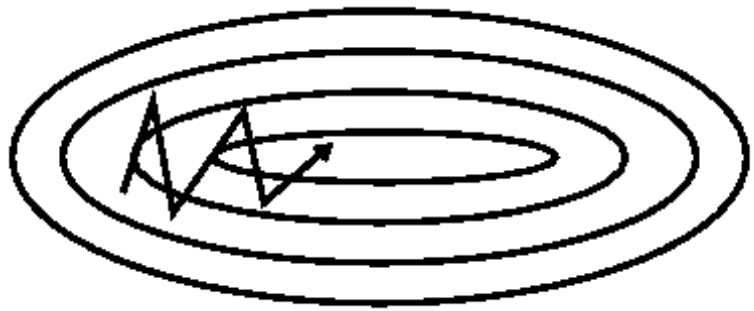
$$\Theta_t = \Theta_{t-1} - v_t$$

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

SGD z momentem



SGD bez momentu



SGD z momentem

Nesterov accelerated gradient (NAG)

$$v_t = \gamma v_{t-1} + \eta \nabla_\Theta J(\Theta - \gamma v_{t-1})$$

$$\Theta_t = \Theta_{t-1} - v_t$$

- gradient liczony dla przybliżonej przyszłej pozycji $\Theta - \gamma v_{t-1}$
- 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/k2). Doklady ANSSSR (translated as Soviet.Math.Docl.), vol. 269, pp. 543– 547.

Adagrad (Duchi, 2011)

$$\Theta_t = \Theta_{t-1} - \frac{\eta}{\sqrt{\sum_{t'<t} g_{t'}^2}} g_t$$

gdzie g_t to gradient w chwili t

$$g_t = \nabla J_\Theta(\Theta_t)$$

- **efektywna stała uczenia** ustalana dla każdego optymalizowanego parametru, zależna od wszystkich poprzednich wartości gradientów g_t
- domyślna wartość $\eta = 0.01$, zazwyczaj nie wymaga zmiany
- wada: mianownik może szybko przybrać duże wartości, co powoduje zanik uczenia (problem ten niwelują algorytmy Adadelta, RMSProp, Adam)

Duchi, J., Hazan, E., Singer, Y. (2011) „Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", JMLR

Adaptive Moment Estimation (Adam)

- wyznaczane na bieżąco estymaty pierwszego (średnia) i drugiego (wariancja) momentu gradientów

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

- poprawka na odchylenie w stronę 0 (początkowe wartości)

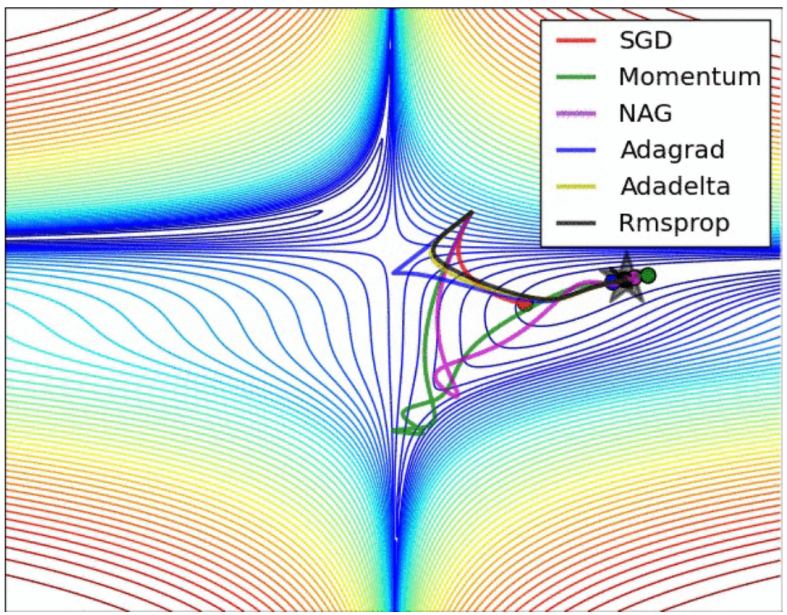
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

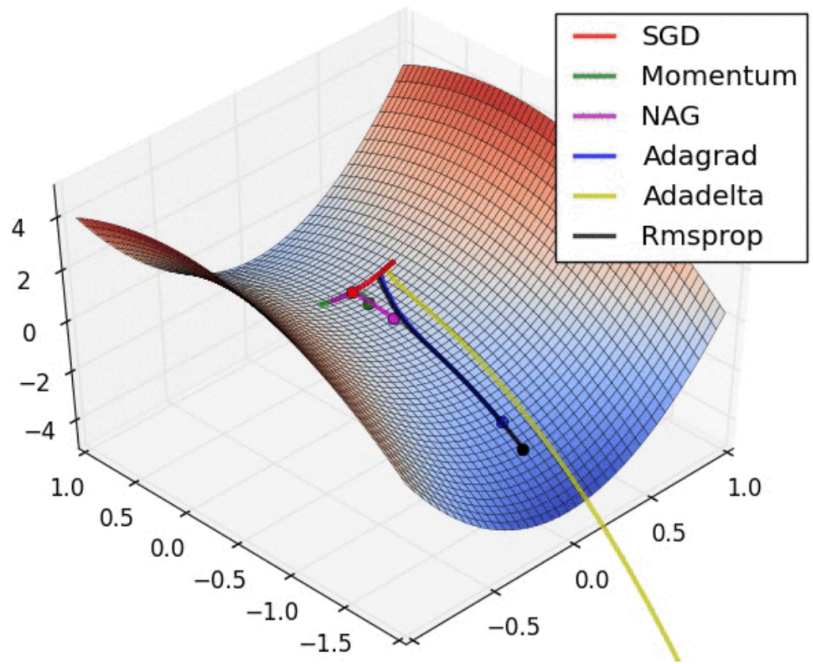
$$\Theta_t = \Theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t}} \hat{m}_t$$

- typowo $\beta_1 = 0.9$, $\beta_2 = 0.999$ współczynniki zanikania, $\eta = 0.01$

Kingma, D. P., Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13

Porównanie





☞ An overview of gradient descent optimization algorithms

Podsumowanie

- DNN dla danych bez struktury (przestrzennej, czasowej)
- w większości zastosowań ReLU sprawdza się bardzo dobrze
- softmax + Cross Entropy
- Adam w wielu przypadkach lepszy od innych algorytmów z domyślnymi parametrami η, β_1, β_2
- SGD z zanikającą wykładniczo wartością stałej uczenia może dorównać algorytmom takim jak Adam