

Głębokie uczenie

Marek Grochowski

Plan

1. Głębokie sieci neuronowe (DNN)
2. Sieci splotowe (CNN)
3. Sieci rekurencyjne (RNN) i uczenie sekwencji
4. Autoenkodery, wykrywanie cech, DBN

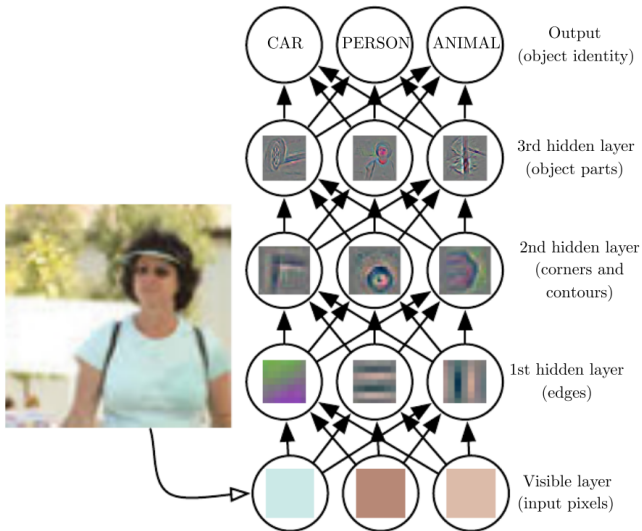
Literatura

- [1] Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*. MIT Press, 2016,
<http://www.deeplearningbook.org>
- [2] Michael Nielsen, *Neural Networks and Deep Learning*,
<http://neuralnetworksanddeeplearning.com/>
- [3] Denny Britz, *Deep Learning Glossary*,
<http://www.wildml.com/deep-learning-glossary/>

Głębokie 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 reprezentując najprostszych cech z sygnału wejściowego, wyższe warstwy - generują bardziej ogólne koncepty bazując na relacjach z poprzednich warstw.

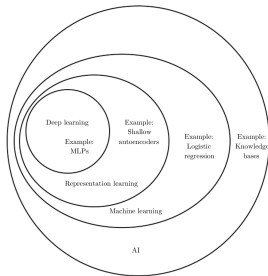
Hierarchia reprezentacji



Gooffellow, 2016 [1]

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.



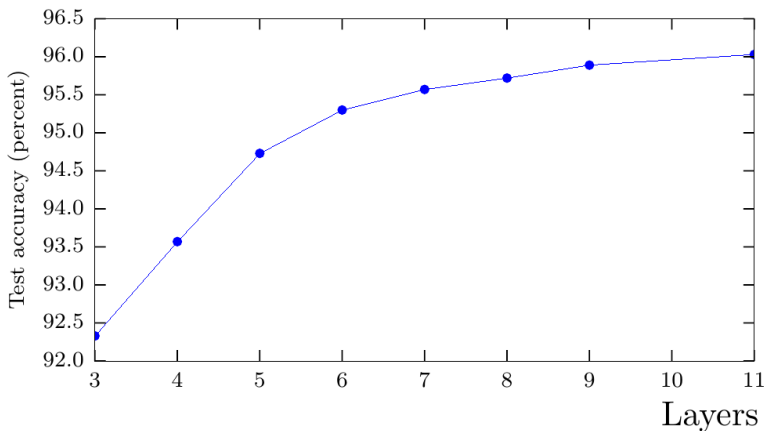
Głębokie vs. płytkie uczenie

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

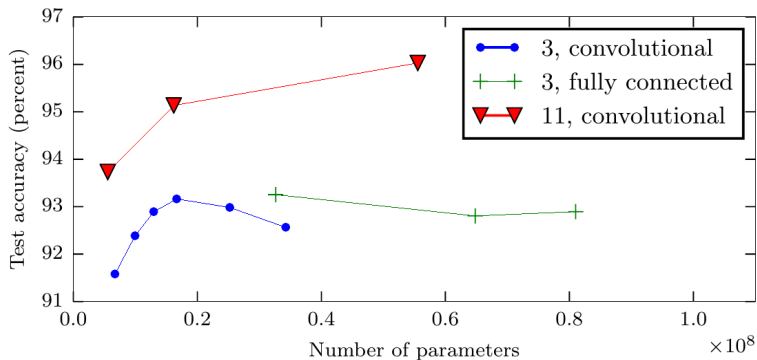
- **Uniwersalny aproksymator:** sieć MLP z jedną warstwą ukrytą jest w stanie aproksymować dowolną funkcję z dowolną dokładnością
- Deep learning też nie taki prosty, problemy nie tylko z treningiem ale z interpretacją działania
- Po co więc tworzyć głębsze sieci?
- „Płytkie” modele wymagają o wiele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

Głębokość a poprawność klasyfikacji

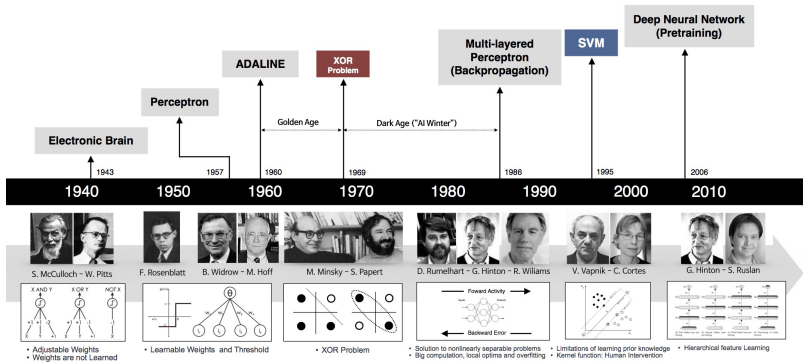


Goofellow, 2016 [1]

Głębokość ma znaczenie



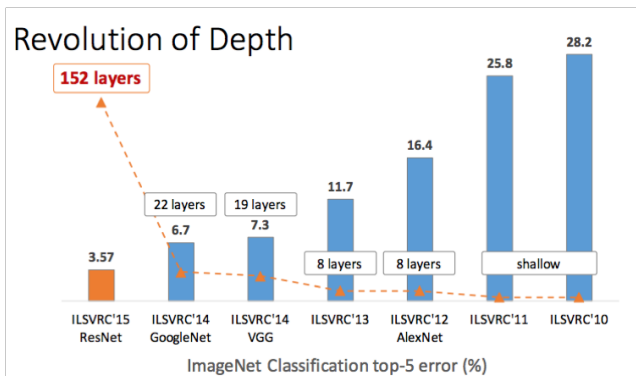
Goofellow, 2016 [1]



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

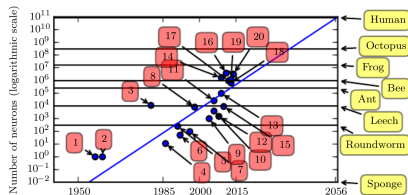
State of art results

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

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 rozmawiające z AI (Google Duplex)
- 🖐️ 30 amazing applications of deep learning

DNN

Głębokie MLP

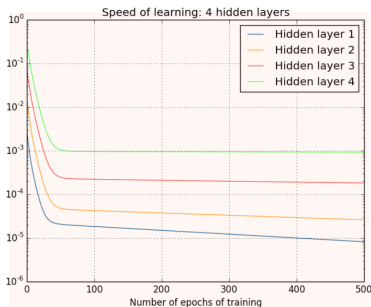
Problemy związane z uczeniem (głębokich) sieci

- **zanikający gradient** - błąd propagowany od warstwy wyjściowej zanika przy niższych warstwach
- funkcje ograniczone (sigmoidalna, tangens hiperboliczny) „nasycają się” i posiadają niezerowy gradient tylko 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 → długi czas uczenia i duże wymagania dotyczące pamięci

Zanikający gradient

Przykład:

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

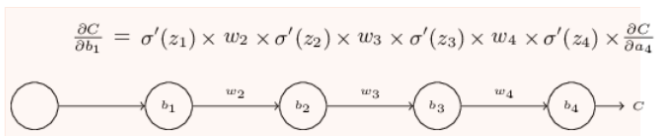


Szybkość uczenia $\|\cdot\|$ drastycznie maleje w głębszych warstwach

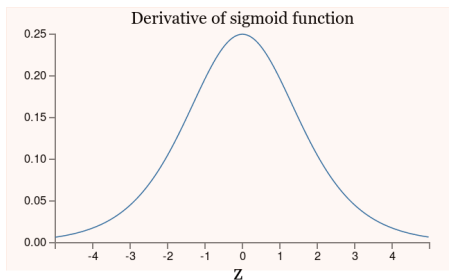
Nielsen, 2016 [2]

Znikający gradient

Przykład: sieć 1 x 1 x 1 x 1



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



Problem niestabilnego gradientu

- gradient w niższych warstwach zależy od wartości wag, aktywacji i gradientów w warstwach wyższych
- **eksplodujący gradient**
gdy $w_i \gg 1$ 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 normalization, 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 między rozkładami y i $\hat{y} \in (0, 1)$
- gdy $y \approx \hat{y}$ to funkcja kosztu bliska zeru
- $J > 0$

Gradient funkcji kosztu:

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

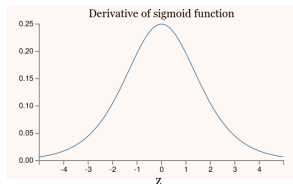
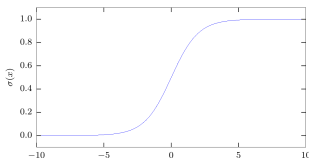
Funkcja liniowa w warstwie wyjściowej

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

- 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 zanika 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

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

- wartości $(0, 1)$ unormowane

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

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

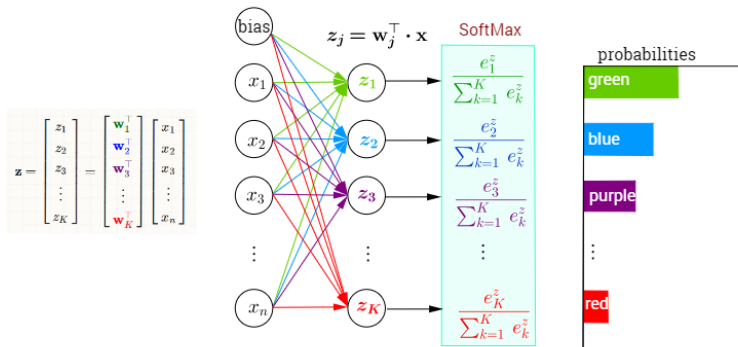
$$[0, \dots, 0, 1, 0, \dots, 0]$$

- wartości wyjściowe sieci reprezentują rozkład prawdopodobieństw zmiennej wyjściowej

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

Softmax w warstwie wyjściowej

Multi-Class Classification with NN and SoftMax Function



Źródło:

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

- gradient

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

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

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

- obliczenia się upraszczają i znika problem zanikania błędu dla nasyconych wyjść, który występował dla funkcji kosztu MSE

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

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

- ważne są różnice pomiędzy wyjściami a nie amplitudy poszczególnych wyjść

$$\text{softmax}(x) = \text{softmax}(x + c)$$

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

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

- 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

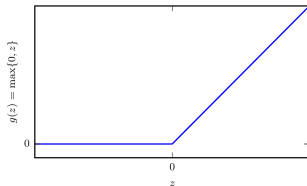
Output Type	Output Distribution	Output Layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)
Continuous	Mixture of Gaussian	Mixture Density	Cross-entropy
Continuous	Arbitrary	See part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

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)

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



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

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

Sparse representaion

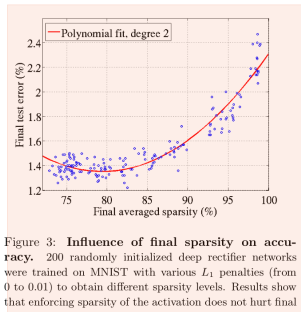
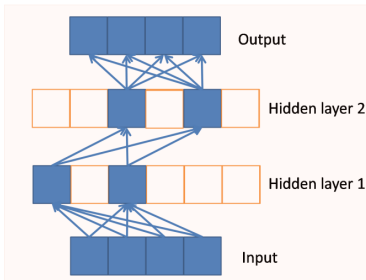


Figure 3: **Influence of final sparsity on accuracy.** 200 randomly initialized deep rectifier networks were trained on MNIST with various L_1 penalties (from 0 to 0.01) to obtain different sparsity levels. Results show that enforcing sparsity of the activation does not hurt final performance until around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

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 osiągną 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 $z < 0$

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

- **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

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

Maxout

Zwraca wyjście najaktywniejszego neuronu z grupy k neuronów w warstwie

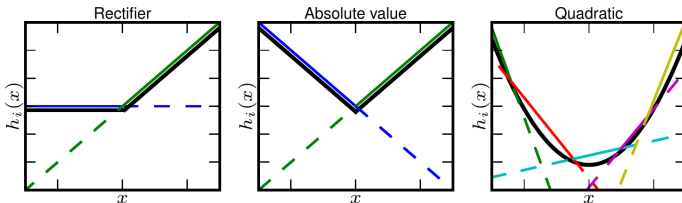
$$g_i(\mathbf{z}) = \max_{j \in G_i} z_j$$

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

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

Maxout

- realizuje funkcję wypukłą fragmentami liniowa złożoną z k fragmentów



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

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

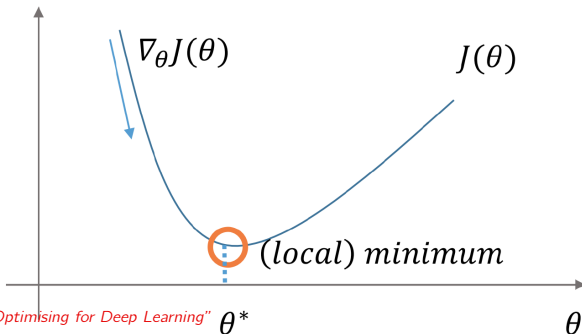
Optymalizacja spadkiem gradientu

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

$J(\Theta)$ 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 zbioru 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

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

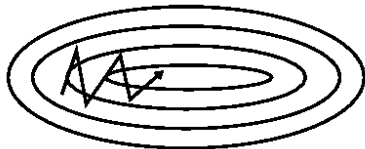
$$\Theta_t = \Theta_{t-1} - v_t$$

- 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 $o(1/k^2)$. 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)

Adaptive Moment Estimation (Adam)

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

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

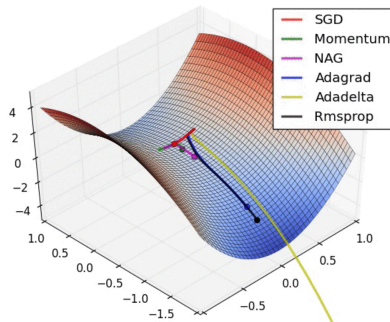
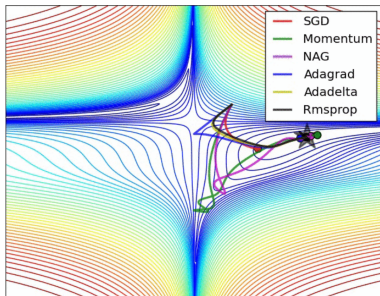
- poprawka na odchylenie w stronę 0 (początkowe wartości)

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

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

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

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

CNN

Convolutional Neural Networks

CNN

- **Convolutional Neural Networks (CNN, ConvNet)**
wariant MLP inspirowany biologicznie, gdzie mnożenie macierzy wag i sygnału wejściowego zastąpione jest operacją splotu
- rzadka reprezentacja, współdzielone wagi, pooling
- zdolne do generalizacji sygnału posiadającego relacje przestrzenne, odporne na przestrzenne transformacje sygnału (skalowanie, obrót, przesunięcie):
 - 1D sygnały czasowe
 - 2D obrazy
 - 3D fMRI, video, obrazy RGB
 - sygnały wielokanałowe

Splot

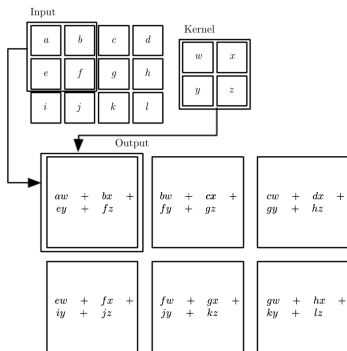
$$(x \star w)(t) = \int x(a)w(t - a)da$$

- wynikiem jest funkcja, np. uśredniona wartość $x(x)$ względem wszystkich pozycji $w(a)$ jeśli w spełnia wymagania gęstości prawdopodobieństwa
- W terminologii CNN:
 x - sygnał wejściowy
 w - kernel (filtr), wagi połączeń neuronu
wartości wyjściowe tworzą mapę cech (*feature map*)

$$s(i) = \sum_m x(m)w(i - m)$$

- w sieciach CNN w niezerowe tylko w organicznym obszarze (pole recepcyjne)

Splot 2D



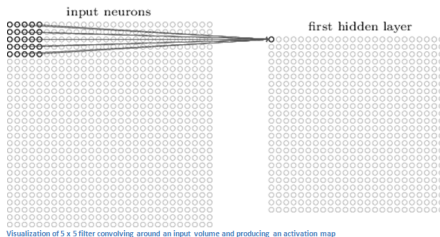
$$S_{ij} = \sum_l \sum_m w_{lm} x_{i+l, j+m}$$

Filtry graficzne 2D – przykłady

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

Pola recepcyjne i mapy cech

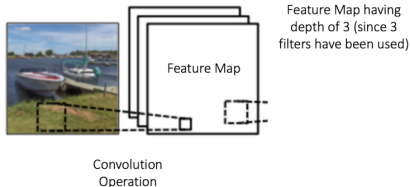


- **Pole recepcyjne** - obszar „widziany” przez neuron (rozmiar filtra)
- wyjście neuronu: spłot sygnału i liniowego filtra, ewentualny wyraz wolny (bias) i nieliniowa funkcja wyjściowa (ReLU, tanh), generują **mapę cech** (*feature map*)

$$s_{ij} = \sigma \left(b + \sum_l \sum_m w_{lm} x_{i+l, j+m} \right)$$

Pola recepcyjne i mapy cech

- Warstwa zawierająca N neuronów (filtrów) tworzy N map (objętość, tensor n wymiarowy)

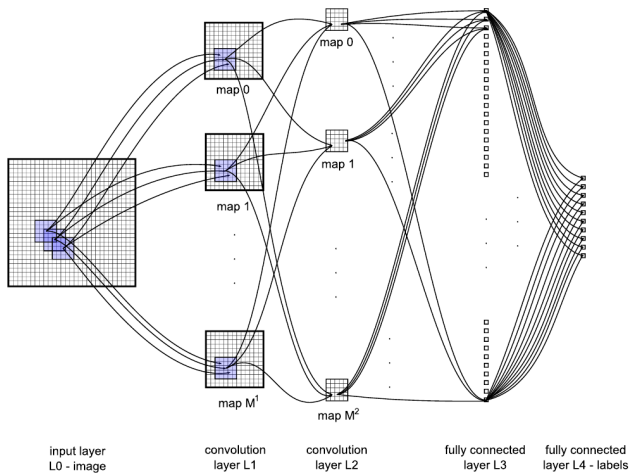


- Przy przetwarzaniu obrazów sygnał wejściowy to tensor 3D (kanał, szerokość, wysokość), w zastosowaniu rozszerzony do 4D o wymiar związany z rozmiarem mini-batcha

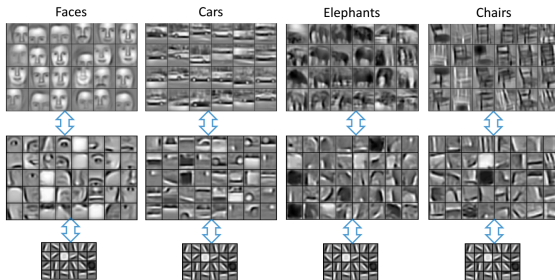
$$s(i, j, k) = \sum_{lmn} x(l, j + m, k + n)w(i, l, m, n)$$

- tensor wyjściowy warstwy zawierającej N filtrów (N, M_x, M_y) , gdzie M_x, M_y - rozmiary mapy wyjściowej

Typowa architektura CNN



Przykłady filtrów



Rozmiar mapy cech

- Warstwa zawierająca N neuronów (filtrów) tworzy N map (objętość, tensor n wymiarowy)
- Rozmiar mapy wyjściowej (M_x, M_y) zależy od:

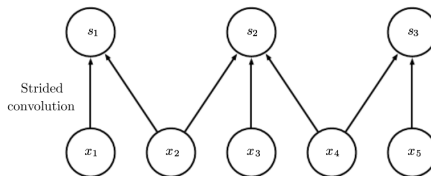
$$M_i^n = \frac{M_i^{n-1} - K_i^n + 2P_i^n}{S_i^n} + 1$$

- rozmiar filtra (K_x, k_y) (szerokość i długość)
- przesunięcie (*stride*) (S_x, S_y) w każdym z wymiarów, np. spłot 2D

$$s(i, j, k) = \sum_{lmn} x(l, j \times S + m, k \times S + n) w(i, l, m, n)$$

- sposób uwzględnienia wartości brzegowych (*zero padding*), P_i
 - wielkość rozszerzenia brzegu w wymiarze i

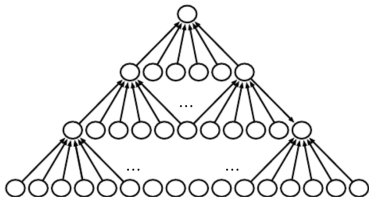
Stride



- redukcja wymiaru - zmniejszenie wymogów obliczeniowych i pamięciowych
- zmniejszenie rozdzielczości sygnału
- kosztem mniej dokładnej reprezentacji

Zero padding

valid zero padding - rozmiar kolejnych map maleje w kolejnych warstwach



- ogranicza to możliwość budowania głębokich sieci i wymusza stosowanie małych filtrów
- sygnał wejściowy na brzegach ma mniejszy wpływ na sygnał wyjściowy

Zero padding

same zero padding - brzegi wypełnione dodatkowymi wartościami (zerami) aby zapewniać odtworzenie wymiaru sygnału wejściowego



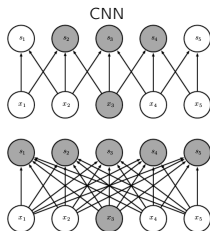
- pozwala budować bardzo głębokie sieci o dowolnych wielkościach filtrów $M_i^n = \lceil \frac{M_i^{n-1}}{S_i} \rceil$
- sygnał wejściowy na brzegach ma mniejszy wpływ na sygnał wyjściowy

Właściwości CNN

- **rzadka reprezentacja** - rozmiar filtra jest dużo mniejszy od rozmiaru sygnału wejściowego, pojedyncze wejście oddziałuje tylko na grupę neuronów
- **współdzielenie parametrów** - warstwa splotowa może być widziana jako w pełni połączona warstwa ze współdzielonymi wagami (dużo mniejsza liczba parametrów w stosunku do MLP)
- **równoważność względem przesunięcia** sygnału - ta sama cecha znajdująca się w różnych miejscach obrazu będzie aktywowała ten sam filtr
- możliwość użycia sygnału wejściowego o zmiennym rozmiarze - większy obraz wejściowy wygeneruje większe mapy, w przypadku MLP zwiększenie rozmiaru wektora wejściowego wymaga rozbudowy architektury

Rzadka reprezentacja

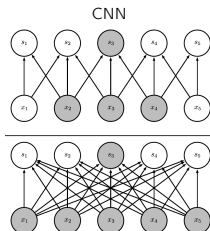
pojedyncze wejście aktywuje tylko grupę neuronów w małym obszarze



MLP

MLP: mnożenie macierzy $O(m \times n)$ parametrów
CNN: warstwa splotowa $O(k \times n)$, gdzie $k \ll m$

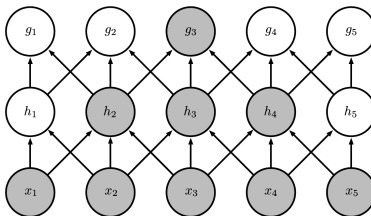
wyjście neuronu zależne od małego obszaru sygnału wejściowego



MLP

Efektywne pole recepcyjne

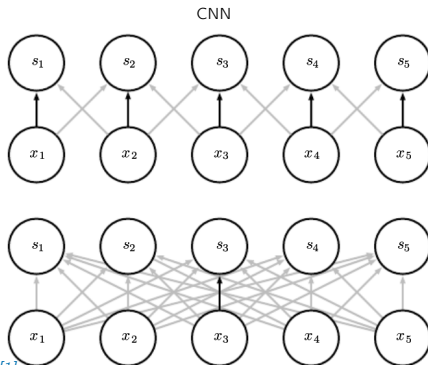
- **Efektywne pole recepcyjne** - obszar sygnału wejściowego pokryty przez neurony w wyższych warstwach, rośnie z głębokością



- stride, pooling i dilation (rozrzedzony splot) dodatkowo zwiększają efektywne pole recepcyjne
- pomimo rzadkich połączeń sieć jest w stanie w ten sposób modelować złożone zależności

Współdzielenie wag

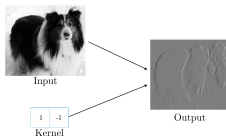
- ta sama waga jest używana przy przetwarzaniu każdego punktu wejściowego
- pojedynczy filtr pozwala wykryć tę samą cechę w różnych położeniach obrazu wejściowego (*equivariance to translation*)



Źródło grafiki: Goofellow, 2016 [1]

Rzadkie połączenia i współdzielenie wag

Przykład: wykrywanie krawędzi dla obrazu $n \times n$ w poziomie



- MLP: mnożenie pełnej macierzy $n^2 \times n^2 = n^4$
- CNN: splot filtrem 1×2 (2 wagi) wymaga $n^2 \times 3$ operacji (2 mnożenia i dodawanie)
- splot drastycznie bardziej wydajny w mapowaniu powtarzających się relacji w małych, lokalnych rejonach sygnału wejściowego

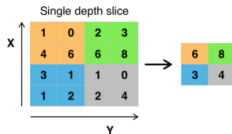
Źródło grafiki: Goofellow, 2016 [1]

Równoważność przy przesunięciu

- przesunięcie sygnału wejściowego powoduje identyczne przesunięcie sygnału wyjściowego na mapie cech
- własność pożądana w rozpoznawaniu obrazów, gdzie lokalne cechy (np. krawędzie) mogą wystąpić w każdym miejscu na obrazie
- współdzielenie wag dla całego wejścia nie zawsze jest pożądane (różne filtry w różnych obszarach obrazu), np. rozpoznawanie twarzy ze zdjęć paszportowych, posiadają charakterystyczne cechy występujące wyłącznie w określonych rejonach sygnału wejściowego
- spłot nie jest niezmienniczy względem zmiany skali lub obrotu obrazu, osiąga się to poprzez rozszerzenie zbioru treningowego o przypadki zdeformowane (szum, skalowanie, obrót, zmiana kontrastu, etc..)

Pooling

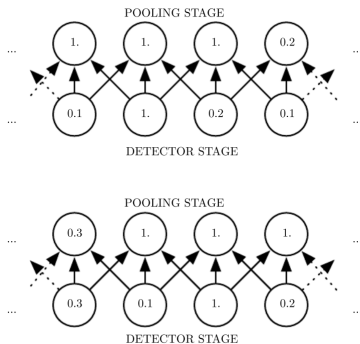
- typowa warstwa w sieciach CNN:
 - liniowa aktywacja (splot)
 - detekcja (nieliniowość np. ReLU)
 - funkcja redukcji (pooling) - „uogólnienie” wartości sąsiadujących wyjść
- **Max pooling** - maksimum z pewnego podobszaru (*winner takes all*)



- redukcja wymiarowości - przesunięcie filtra typowo równe jego wielkości (maksimum z rozłącznych obszarów)
- inne podejścia: *avg. pooling* (średnia z sąsiedztwa), *norm pooling* (norma z sąsiadujących wyjść), ważona średnia od centrum, niektóre architektury rezygnują z tej warstwy

Pooling względem obrazu wejściowego

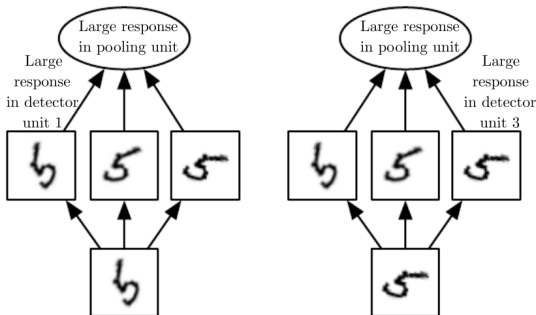
Pooling zapewnia niezmienniczość względem drobnego przesunięcia obrazu wejściowego



Źródło grafiki: Gooffellow, 2016 [1]

Pooling względem map cech

Redukcja względem wyjść różnych splotów umożliwia wprowadzenie do modelu niezmienniczości względem pewnych transformacji sygnału wejściowego (np. obrotu)



Źródło grafiki: Gooffellow, 2016 [1]

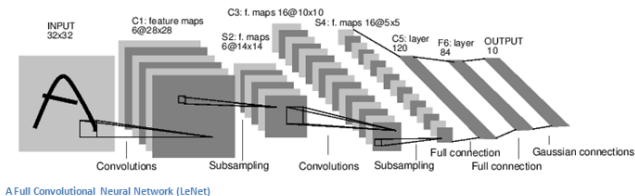
LeNet5 (LeCun 1998)

klasyfikacja cyfr pisanych ręcznie
(odczytywanie czeków)

👉 MNIST

trening: 60k cyfr, 250 osób,

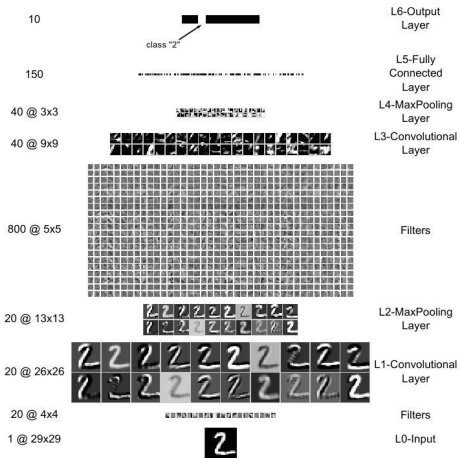
test: 10k cyfr



👉 LeNet-5, convolutional neural networks

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. *Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998*

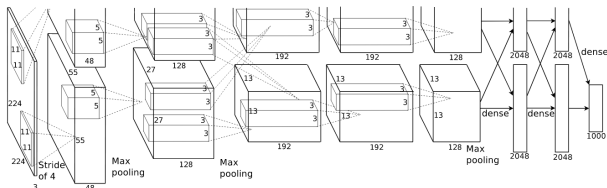
Przykład wizualizacji aktywacji sieci



👉 Wizualizacja sieci splotowej

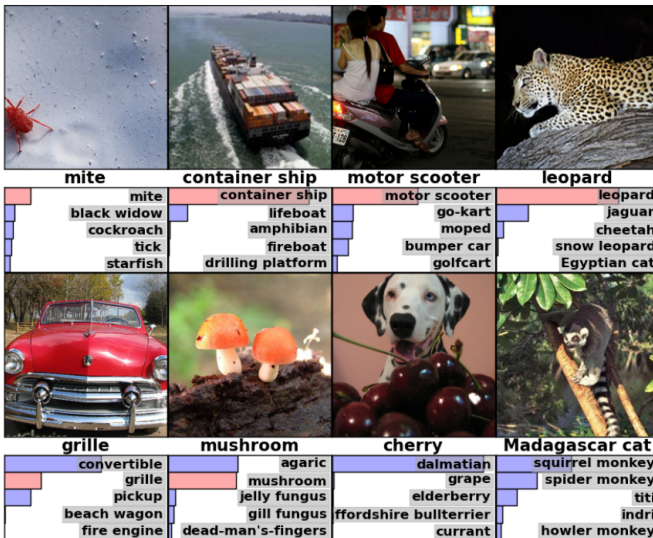
AlexNet (Krizhevsky, 2012)

- Zwycięzca 🖱️ ImageNet w 2012 z poprawnością 15.3% (poprawa z 26%)



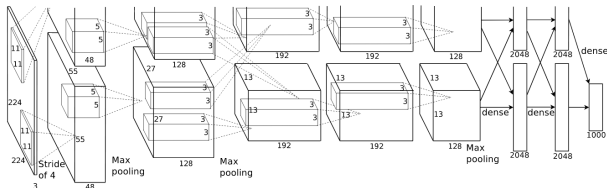
- głębsza od LeNet5 i więcej filtrów na warstwę (11x11, 5x5, 3x3)
- sekwencje warstw splotowych
- dropout
- 6 dni treningu (2x NVIDIA GTX 580 3GB GPUs).

AlexNet



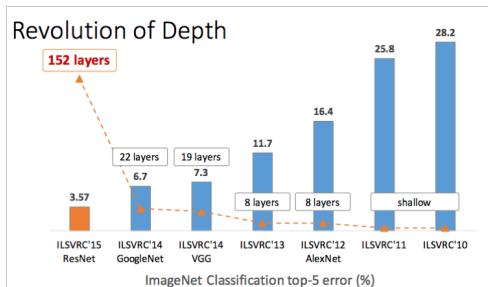
AlexNet (Krizhevsky, 2012)

- Zwycięzca 🖱️ ImageNet w 2012 z poprawnością 15.3% (poprawa z 26%)



- głębsza od LeNet5 i więcej filtrów na warstwę (11x11, 5x5, 3x3)
- sekwencje warstw splotowych
- dropout
- 6 dni treningu (2x NVIDIA GTX 580 3GB GPUs).

ImageNet



- ZFNet(2013) - podobny do AlexNet
- GoogleNet (Inception) - małe sploty w celu zredukowania liczby parametrów
- 🖱️ CNNs Architectures used on ImageNet

VGGNet (Simonyan and Zisserman, 2014)

- dużo małych filtrów 3x3
- wielokrotne warstwy splotowe dodają nieliniowość oraz zwiększają efektywne pole recepcyjne
- stride=1, brak utraty informacji
- 5 warstw max pool
- od 11 do 19 warstw
- 138M parametrów
- trening 2-3 tygodnie (4x GPU).

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

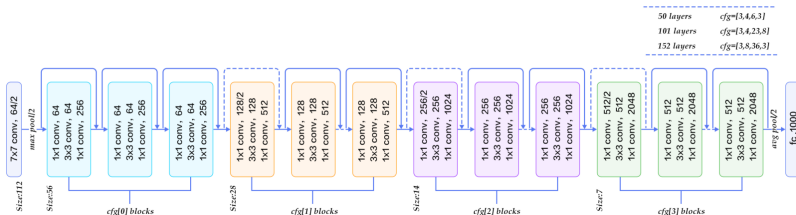
FC-4096

FC-1000

softmax

K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv technical report, 2014

ResNet (Residual Neural Network, Kaiming He et. al 2015)



- wiele bloków zawierających sploty 1x1, 3x3, 1x1
- przejścia „skrótowe” pomagają uczyć bardzo głębokie sieci (152 warstwy)
- batch normalization

RNN

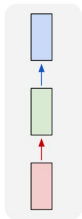
Sieci rekurencyjne i
modelowanie sekwencji

Modelowanie sekwencji

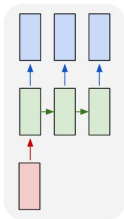
- Dane sekwencyjne lub zależne od czasu: sygnał audio, tekst, ruch obiektów, EEG, ...
- Modelowanie sekwencji: generowanie tekstu (chaterboty), generowanie muzyki, etykietowanie obrazów, sterowanie ruchem robota, ...
- Przetwarzanie sekwencji: maszynowe tłumaczenie, rozpoznawanie mowy, rozpoznawanie pisma, przetwarzanie strumieni wideo, ...
- Klasyfikacja sygnałów czasowych, wykrywanie wzorców w sekwencjach, predykcja (np. przewidywanie notowań giełdowych)

Modelowanie sekwencji

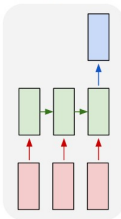
one to one



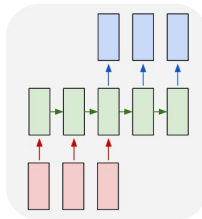
one to many



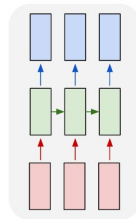
many to one



many to many



many to many



- sieci jednokierunkowe: wejście i wyjście o stałym rozmiarze
- generowanie sekwencji dla pojedynczego wejścia: opis obrazów
- sekwencyjne wejście: np. klasyfikacja tekstów
- wejście i wyjście sekwencyjne:
 - bez dopasowania ramek, np. tłumaczenie maszynowe
 - z dopasowaniem ramek (*alignment*), np: etykietowanie ramek

Modelowanie sekwencji

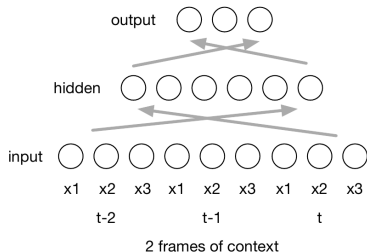
- Model sekwencji: inwariantność względem czasu, aktualny stan zależy od przeszłości
- MLP: dodanie **stałego kontekstu** do wejścia: np. 2 ramki z lewej i prawej (poprzednie i przyszłe wektory wejściowe).

$$[\mathbf{x}^{t-2}, \mathbf{x}^{t-1}, \mathbf{x}^t, \mathbf{x}^{t+1}, \mathbf{x}^{t+2}]$$

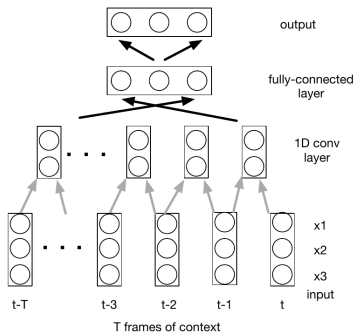
- CNN: splot względem wymiaru czasu, stały kontekst odpowiada wielkości efektywnego pola recepcyjnego. TDNN (*time-delay neural network*) model ze splotem 1D w domenie czasu.

Uczenie sekwencji z MLP i CNN

MLP



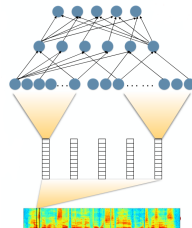
TDNN



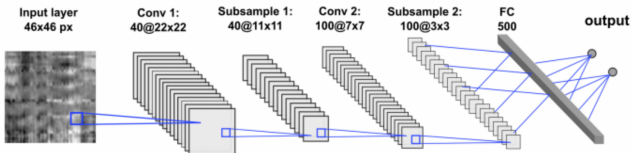
Źródło grafik: Steve Renals, Machine Learning Practical — MLP Lecture 9

Przykłady

Model akustyczny, rozpoznawanie fonemów z nagrań audio



Analiza sygnałów EEG, CNN + widma EEG

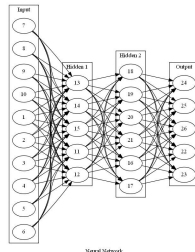


Nurse, E., Mashford, B. S., Yepes, A. J., Kiral-Kornek, I., Harrer, S., & Freestone, D. R. (2016). Decoding EEG and LFP Signals using Deep Learning: Heading TrueNorth

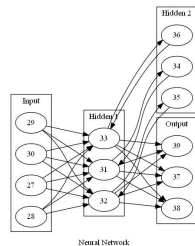
RNN

- **Sieci jednokierunkowe** używają jedynie kontekstu o skończonej długości, nie są w stanie wykryć zależności w dłuższych okresach czasu (*long-term*)
- **Sieci rekurencyjne (RNN)** posiadają połączenia do wcześniejszych warstw, wyjście w chwili t zależy od stanu z czasu $t - 1$

Sieć jednokierunkowa



Sieć rekurencyjna Elmana



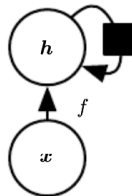
Stan ukryty jednostek RNN

$$\mathbf{h}^t = f(\mathbf{h}^{t-1}, \mathbf{x}^t; \theta)$$

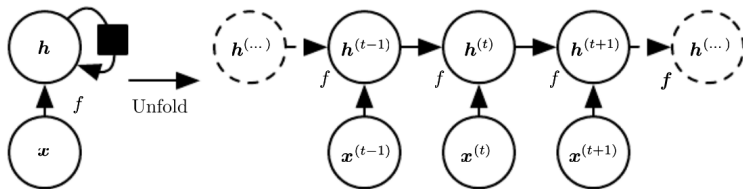
- neurony z połączeniami rekurencyjnymi działają jak pamięć - potencjalnie są w stanie modelować relacje występujące z dowolnie długim odstępem czasowym (nieskończony kontekst)

$$\mathbf{h}^t = g(\mathbf{x}^t, \mathbf{x}^{t-1}, \dots, \mathbf{x}^1)$$

- wektor $\mathbf{h}^t$ tworzy skompresowaną reprezentację sekwencji, mapuje dowolnie długą sekwencję na wektor o skończonej długości
- stan początkowy $\mathbf{h}^0$ może być interpretowany jako dodatkowy sygnał wejściowy, typowo inicjowany zerami

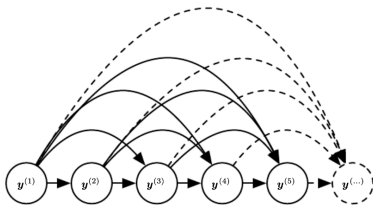


Rozwinięcie rekurencji w czasie

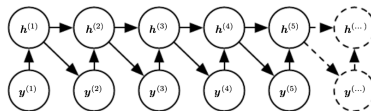


Efektywność kodowania

- Sieć RNN z rekurencją w warstwach ukrytych spełnia wymagania **uniwersalnej maszyny Turinga**
- Ilość parametrów opisujących stan **h** ma wpływ na wielkość pamięci ale nie musi zależeć od długości sekwencji

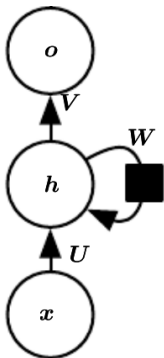


pełny model sekwencji, ilość parametrów $O(k^2)$



RNN, stan obecny zależy pośrednio od stanów z przeszłości, ilość parametrów stała względem dł. sekwencji $O(1)$

Sieć z jedną warstwą rekurencyjną

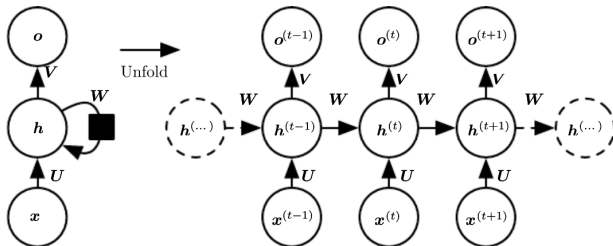


$$o_k^t = \text{softmax} \left(\sum_r v_{kr} h_r^t + b_k \right)$$

$$h_i^t = \sigma \left(\sum_j u_{ij} x_j^t + \sum_r w_{ir} h_r^{t-1} + b_j \right)$$

Źródło grafiki: Goodfellow, Deep Learning, 2016

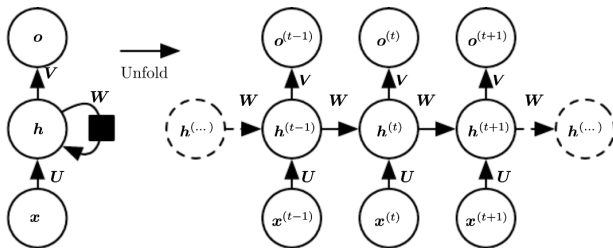
Wsteczna propagacja w czasie



- Rozwinięta sieć odpowiada głębokiej sieci jednokierunkowej ze współdzielonymi wagami **W**, **V**, **U**
- Trening za pomocą wstecznej propagacji odbywa się analogicznie jak w jednokierunkowych sieciach:
BPTT backpropagation through time

Źródło grafiki: Goodfellow, Deep Learning, 2016

Wsteczna propagacja w czasie



Funkcja kosztu dla całej sekwencji:

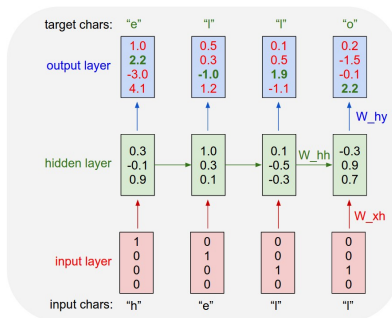
$$L(\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^\tau\}, \{\mathbf{y}^1, \mathbf{y}^2, \dots, \mathbf{y}^\tau\}) = \frac{1}{\tau} \sum_t L^t$$

L^t - koszt w kroku t (np. cross-entropy) zależny od $\mathbf{x}^1, \dots, \mathbf{x}^t$

Źródło grafiki: Goodfellow, Deep Learning, 2016

Przykład: CharRNN

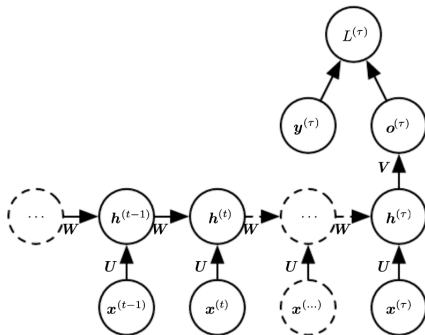
- model języka oparty na znakach (alfabet: h, e, l, o)
- RNN modeluje rozkład prawdopodobieństwa wystąpienia następnego znaku w sekwencji



Wsteczna propagacja w czasie

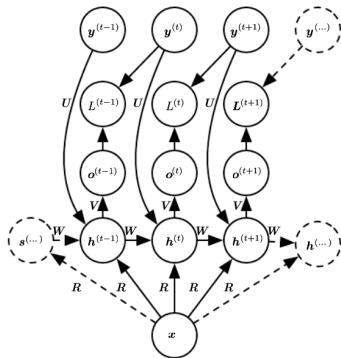
- Rozwinięta w czasie sieć tworzy głęboką sieć neuronową ze wszystkimi tego konsekwencjami (zanikający gradient dla relacji odległych w czasie, rzadziej - wybuchający gradient)
- Obliczenia dla długich sekwencji są kosztowne
- **Truncated BPTT** - ograniczenie długości sekwencji do stałej wartości (np. 20)

Pojedyncze wyjście na końcu sekwencji



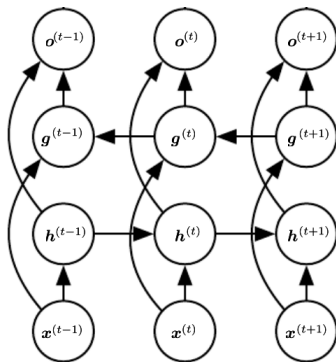
Sieć produkuje wyjście po obejrzeniu całej sekwencji, np. klasyfikacja wpisów w portalach społecznościowych pod względem emocji

Pojedyncze wejście generujące sekwencję



- Sieć produkuje sekwencję wyjść przy prezentacji stałego wzorca (np. etykietowanie zdjęć).
- Sygnał wejściowy pełni rolę kontekstu

Dwukierunkowa sieć RNN



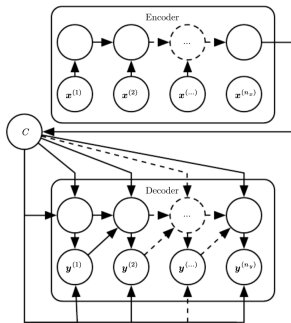
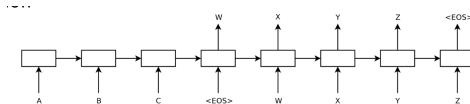
Sieci dwukierunkowe RNN łączą warstwy z rekurencją zależną od sygnału przeszłego **h** z rekurencją zależną od przyszłego sygnału **g**

Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." Signal Processing, 1997

Dwukierunkowa sieć RNN

- informacja z przyszłości może zawierać istotną informację, np. dźwięki kolejnych fonemów (koartykulacja), znaczenie słów w zdaniu
- wyjścia sieci w dowolnym momencie zależą od całej sekwencji wejściowej
- zastosowanie: rozpoznawanie mowy (Graves, 2008, 2009), bioinformaryka (Baldi, 1999), rozpoznawanie pisma (Liwicki, Marcus, 2007)
- dla obrazów 2D możliwe zastosowanie sieci 4 kierunkowych, co pozwala modelować odległe relacje, jednak nie są one tak wydajne w tych zastosowaniach jak CNN

Model Encoder-Decoder



Sutskever et al (2014). "Sequence to Sequence Learning with Neural Networks", NIPS.

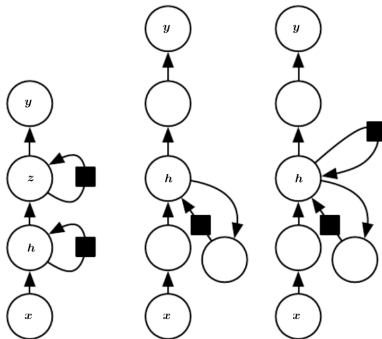
K Cho et al (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", EMNLP.

Model Encoder-Decoder

seq2seq (Sutskever, 2014) to dwie sieci RNN uczone jednocześnie:

- **enkoder** (reader, input) przetwarza sekwencję wejściową generując kontekst C
- **dekoder** (writer, output) zależnie od kontekstu C generuje sekwencję wyjściową
- wielkość kontekstu C może ograniczyć zdolność sieci do reprezentacji dalekich w czasie relacji
- dodanie uwagi (*attention model*, Bahdanan, 2015) - uwzględnienie w C więcej informacji z enkodera, które skalują wyjścia dekodera

Deep RNN

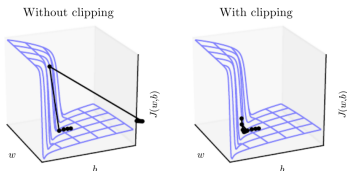


- warstwowe RNN - kolejne warstwy tworzą hierarchię reprezentacji (Graves, 2013)

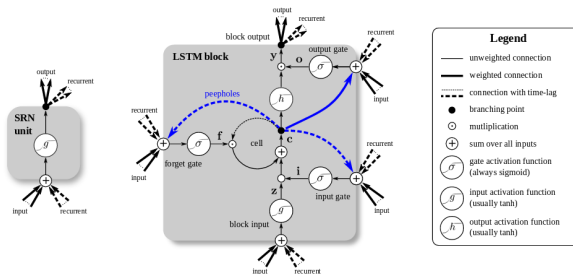
Deep RNN

Problemy uczenia sieci RNN:

- znikający gradient dla odległych w czasie relacji (zanika wykładniczo z czasem), rozwiązaniem LSTM
- niestabilność uczenia - eksplodujący gradient, rozwiązanie: przycinanie gradientu i regularyzacja
- **gradient clipping** - ograniczenie wartości gradientu dla wszystkich wymiarów w mini-batchu, może zmienić kierunek spadku gradientu
- **norm clipping**: jeśli $\|g\| > v$ to $g \leftarrow \frac{gv}{\|g\|}$



Long Short-Term Memory



$$\begin{aligned}
 \mathbf{z}^t &= g(\mathbf{W}_z \mathbf{X}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z) \\
 \mathbf{i}^t &= \sigma(\mathbf{W}_i \mathbf{X}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i) \\
 \mathbf{f}^t &= \sigma(\mathbf{W}_f \mathbf{X}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f) \\
 \mathbf{c}^t &= \mathbf{i}^t \odot \mathbf{z}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1} \\
 \mathbf{o}^t &= \sigma(\mathbf{W}_o \mathbf{X}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o) \\
 \mathbf{y}^t &= \mathbf{o}^t \odot h(\mathbf{c}^t)
 \end{aligned}$$

Sepp Hochreiter, Jürgen Schmidhuber, "Long short-term memory", 1997

LSTM

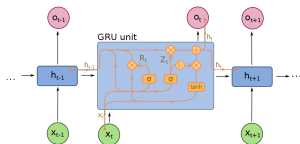
- LSTM niweluje wpływ zanikania gradientu, dzięki czemu pozwala mapować odległe w czasie relacje
- Wprowadza dodatkową wewnętrzną jednostkę stanu $\mathbf{c}$
- Stan $\mathbf{c}^t$ zależy w sposób liniowy od stanu poprzedniego $\mathbf{c}^{t-1}$ (liniowa rekurencja), więc nawet przy bardzo długich rekurencjach gradient ma szansę nie zaniknąć (*long-term memory*)
- Bramki wejściowa, wyjściowa i bramka zapominania sterują zapamiętywanymi informacjami, zależą od sygnału wejściowego i stanu poprzedniego
- Funkcja różniczkowalna - można uczyć metodami spadku gradientu i wsteczną propagacją

LSTM

- Bramka wejściowa ma wpływ na to jaki sygnał zostanie zapisany w jednostce stanu $\mathbf{c}$
- Bramka zapominania (*forget gate*) steruje wpływem poprzedniego stanu $\mathbf{c}^{t-1}$ na obecny stan $\mathbf{c}^t$
- bramka wejściowa i zapominania wpływają na to co umieszczane jest i co jest usuwane ze stanu
- Bramka wyjściowa - steruje ilością informacji przepływającą ze stanu wewnętrznego do wyjścia jednostki, pozwala np. zachować na później informacje, które w tym momencie nie są istotne

Gated Recurrent Unit

- GRU (Kyunghyun Cho et al., 2014) brak jedynki wewnętrznej c i bramki wyjściowej
- bramka resetu R_t i bramka aktualizacji Z_t
- mniej parametrów wzgl. LSTM, porównywalna jakość wyników dla wielu problemów



$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h (r_t \circ h_{t-1}) + b_h)$$

Źródło: https://en.wikipedia.org/wiki/Recurrent_neural_network

Connectionist Temporal Classification

- metoda treningu sekwencji, gdy nie ma dostępnego dopasowania ramek (*aligement*), długość sekwencji wejściowej jest dłuższa od sekwencji wyjściowej, np. transkrypcja tekstu na podstawie nagrań audio
- wyjście softmax uzupełnione o dodatkową etykietę 'blank'
 $\mathcal{L}' = \mathcal{L} \cup \{blank\}$ generuje rozkład prawdopodobieństw zaobserwowania sekwencji etykiet (ścieżki) $\mathbf{y}'$ o długości T

$$P(\mathbf{y}'|\mathbf{x}) = \prod_{t=1}^T P(y'_t|\mathbf{x})$$

Connectionist Temporal Classification

- CTC poszukuje sekwencji etykiet o największej wiarygodności $P(\mathbf{y}|\mathbf{x})$ marginalizując po wszystkich możliwych ścieżkach $\mathbf{y}'$ z blankiem

$$p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{y}' \in \mathcal{M}^{-1}(\mathbf{y})} P(\mathbf{y}'|\mathbf{x})$$

gdzie $\mathcal{M}$ mapuje sekwencję etykiet o długości T do sekwencji krótszej poprzez usunięcie wszystkich blanków i powtarzających się etykiet

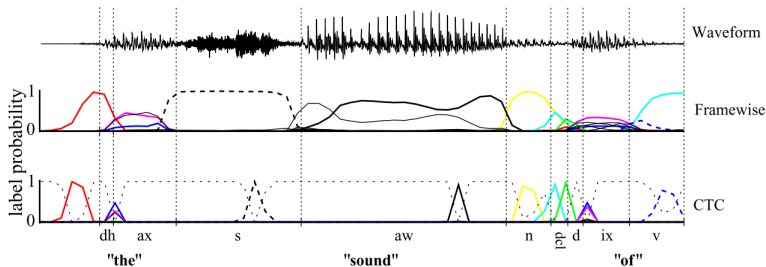
$$\mathcal{M}(a - ab -) = \mathcal{M}(-aa - -abb) = aab$$

- funkcja kosztu

$$L = - \sum \log P(\mathbf{y}|\mathbf{x})$$

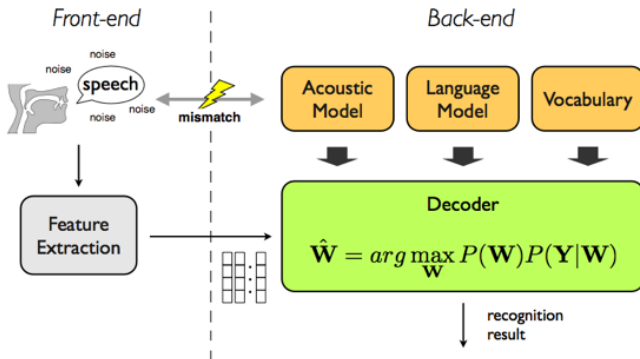
Graves, A., Fernández, S., Gomez, F., Schmidhuber, J., Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. 2006.

Predyckj CTC



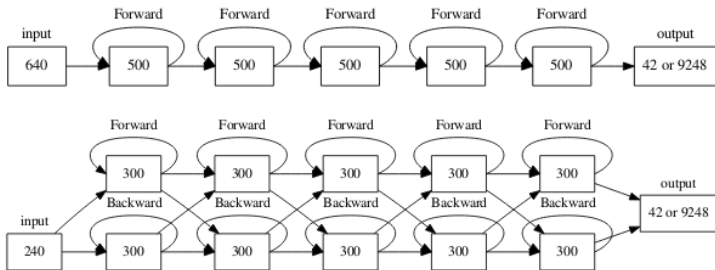
- CTC generuje sekwencję pików etykiet (fonemów) przedzielonych etykietą 'blank'
- w sieci z kosztem CE przesunięcie granic fonemu względem poprawnego dopasowania owocuje błędem nawet gdy fonem jest poprawnie przewidziany
- wyjście CTC może być wykorzystane bezpośrednio (bez konieczności dalszego przetwarzania) do transkrypcji

Automatyczne rozpoznawanie mowy



Źródło grafiki: <https://www.esat.kuleuven.be/psi/spraak/theses/08-09-en/MDT.php>

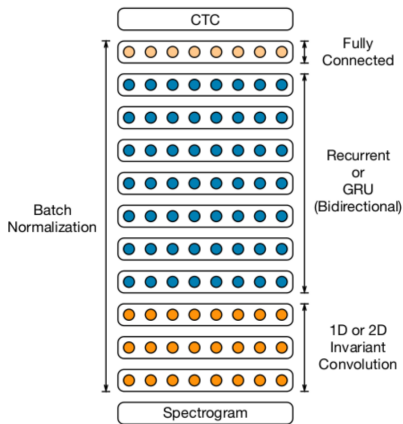
Głęboki model akustyczny



- input: MFCC + context
- output: phonemes or CD states
- node type: LSTM

Sak, Haşim, et al. "Fast and accurate recurrent neural network acoustic models for speech recognition." *arXiv preprint arXiv:1507.06947* (2015).

Deep Speech 2



Read Speech			
Test set	DS1	DS2	Human
WSJ eval'92	4.94	3.60	5.03
WSJ eval'93	6.94	4.98	8.08
LibriSpeech test-clean	7.89	5.33	5.83
LibriSpeech test-other	21.74	13.25	12.69

Amodei, Dario, et al. "Deep speech 2: End-to-end speech recognition in english and mandarin."(2015).

Analiza sygnałów EEG

„Film” EEG - sekwencja obrazów spektralnych: CNN + LSTM

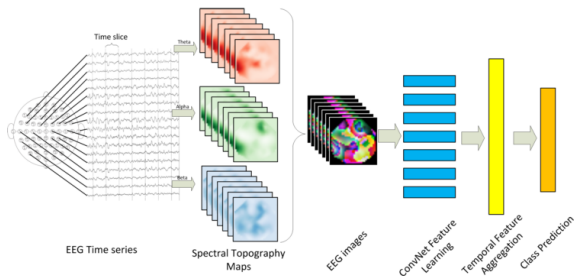


Figure 2: Overview of our approach: (1) EEG time series from multiple locations are acquired; (2) spectral power within three prominent frequency bands is extracted for each location and used to form topographical maps for each time frame (image); (3) sequence of topographical maps are combined to form a sequence of 3-channel images which are fed into a recurrent-convolutional network for representation learning and classification.

Bashivan, P., Rish, I., Yeasin, M., & Codella, N. (2015). Learning Representations from EEG with Deep Recurrent-Convolutional Neural Networks.

Analiza sygnałów EEG

- dane zachowują relacje przestrzenne, czasowe i widmo mocy EEG
- sieci CNN wydobywają inf. przestrzenne (położenie elektrod) i widmo (kanały)
- sieć RNN - relacje czasowe pomiędzy ramkami
- klasyfikacja pojemności pamięci roboczej (4 klasy): błąd SVN 15.3% → CNN+LSTM 8.9%

Gromadzenie informacji czasowej

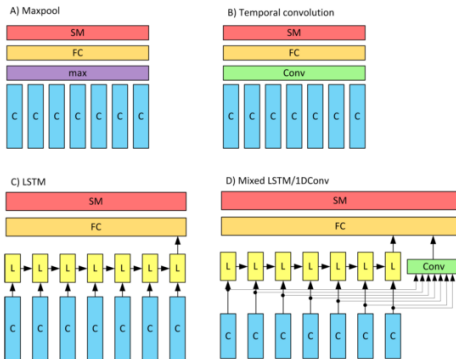
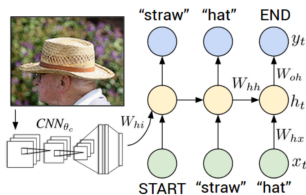
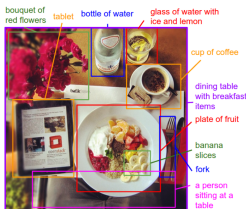


Figure 3: Different multi-frame architectures; the notation here is as follow. C: 7-layer ConvNet; max: maxpool layer across time frame features; FC: fully-connected layer; SM: softmax layer; Conv: 1-D convolution layer across time frames; L: LSTM layer.

Bashivan, P., Rish, I., Yeasin, M., Codella, N. (2015). Learning Representations from EEG with Deep Recurrent-Convolutional Neural Networks.

NeuralTalk - generator opisów zdjęć



Multimodal Recurrent Neural Network 🖱️ NeuralTalk2

Karpathy A., Fei-Fei L. „Deep Visual-Semantic Alignments for Generating Image Descriptions”, 2015

Uczenie nienadzorowane

Autoenkodery i RBM

Uczenie nienadzorowane

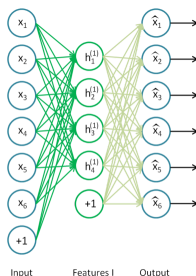
- Uczenie **nadzorowane** (*supervised*) - dane wejściowe **X** posiadają spodziewane wyjście (etykiety) (**X, Y**)
- Uczenie **nienadzorowane** (*unsupervised*) używa wyłącznie danych wejściowych **X**. Dane bez etykiet są powszechnie występujące.
 - trening automatycznie wykrywa istotne cechy w danych,
 - modeluje rozkład pr. danych
- Modele sieci:
 - autoenkodery (AE)
 - Restricted Boltzman Machines (RBM)
 - rzadkie kodowanie, sieci dekonwolucyjne
- Zastosowanie:
 - kodowanie sygnału, wykrywanie istotnych cech, usuwanie szumu i rekonstrukcja sygnału
 - inicjowanie głębokich sieci DNN
 - generowanie sygnałów

Autoenkodery

Encoder:

$$\mathbf{h}(\mathbf{x}) = f(\mathbf{x})$$

$$= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$



Decoder:

$$\hat{\mathbf{x}} = g(\mathbf{h})$$

$$= \sigma(\mathbf{W}'\mathbf{h}(\mathbf{x}) + \mathbf{b}')$$

- **Autoencoder** - sieć jednokierunkowa, której celem jest rekonstrukcja sygnału wejściowego
- warstwa ukryta **h**: *code*, detektor cech, skompresowana reprezentacja danych
- często $\mathbf{W}' = \mathbf{W}^T$ współdzielone wagi

Źródło grafiki: <http://ufldl.stanford.edu>

Koszt rekonstrukcji

Funkcja rekonstrukcji $L(\mathbf{x}, \hat{\mathbf{x}})$

- MSE dla danych wejściowych rzeczywistych

$$L(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{2} \sum_k (\hat{x}_k - x_k)^2$$

– liniowa aktywacja warstwy wyjściowej $\hat{\mathbf{x}} = \mathbf{W}'\mathbf{h} + \mathbf{b}'$

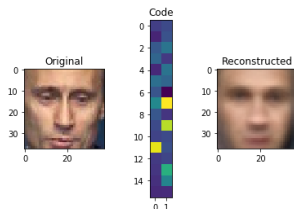
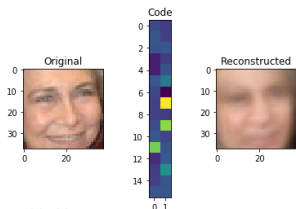
- Cross Entropy dla danych wejściowych binarnych

$$L(\mathbf{x}, \hat{\mathbf{x}}) = - \sum x_k \log \hat{x}_k + (1 - x_k) \log (1 - \hat{x}_k)$$

- trening wsteczną propagacją błędu

Undercomplete AE

- **Undercomplete AE** gdy warstwa **h** mniejsza od rozmiaru wejścia,
- kompresja sygnału do rozmiaru **h**
- nie generalizuje, jest w stanie kodować wyłącznie rozkład danych treningowych
- dla liniowego dekodera jest równoważny z PCA
- Przykład: input-output: 32x32 , hidden (code size): 32



Regularyzowany AE

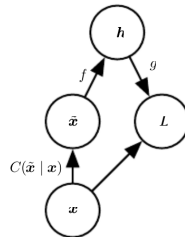
- wielkość $\mathbf{h}$ decyduje o pojemności ale za duża pojemność sprawia, że AE nie jest w stanie wyekstrahować wartościowej informacji o rozkładzie sygnału wejściowego (wejścia mogą być kopiowane na wyjście)
- *Overcomplete* AE gdy rozmiar $\mathbf{h}$ większy od $\mathbf{x}$,
 - brak kompresji
 - nie gwarantują uzyskania wartościowych reprezentacji bez dodania odpowiedniej regularyzacji wymuszającej odpowiednią reprezentację w $\mathbf{h}$ (np. rzadkość)
- **Sparse autoencoder** z karą za nie rzadką reprezentację

$$L(\mathbf{x}, \hat{\mathbf{x}}) + \lambda \sum |h_i|$$

- można też użyć ReLU (Glorat, 2011)

Denoise autoencoder (DAE)

- **Denoise autoencoder (DAE)** usuwa szum (lub inne zniekształcenia) sygnału wejściowego.
- Sygnał wejściowy jest kopią $\mathbf{x}$ z dodanym szumem (lub innymi deformacjami) a AE uczy się odtwarzać oryginalny sygnał bez szumu



Contractive AE

- **Contractive autoencoders (CAE)**

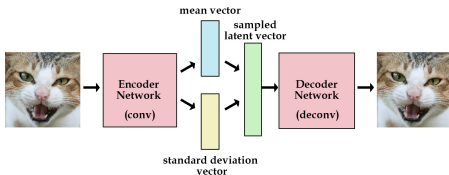
$$L(\mathbf{x}, \hat{\mathbf{x}}) + \lambda \|\nabla_{\mathbf{x}} \mathbf{h}(\mathbf{x})\|_F^2$$

- kara za dużą zmienność reprezentacji przy zmianie sygnału (norma Frobeniusa)

$$\|\nabla_{\mathbf{x}} \mathbf{h}(\mathbf{x})\|_F^2 = \sum_i \sum_j \left(\frac{\partial h_i(x)}{\partial x_j} \right)^2$$

Autoenkodery c.d.

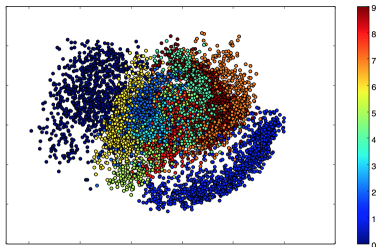
- **Variational autoencoders (VAE)** dodaje człon regularyzujący wymuszający odpowiedni rozkład sygnału w warstwie kodującej



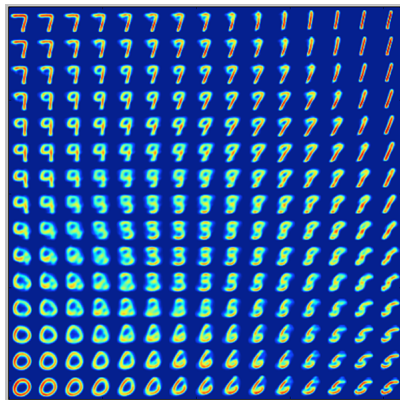
- Convolutional AE, dekodery jest symetryczną do enkodera siecią dekonwolucyjną
- sequence-to-sequence AU (z użyciem np. LSTM)

VAE na MNIST

wizualizacja danych
treningowych

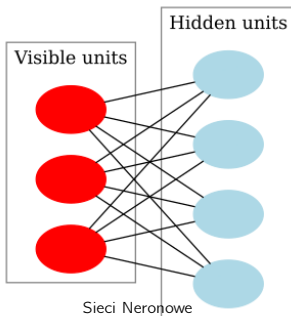


generowanie cyfr



Restricted Boltzman Machine

- **Restricted Boltzman Machines (RBM)** (Hinton 2000), wcześniej Harmonium (P. Smolensky, 1986), to szczególny przypadek Maszyny Boltzmana
- stochastyczna sieć neuronowa modelująca rozkład prawdopodobieństw wejść
- *restrictive* - brak połączeń wewnątrz warstwy
- algorytm uczenia *contrastive divergence* (Hinton)



Źródło grafiki: Wikipedia

Budowa RBM

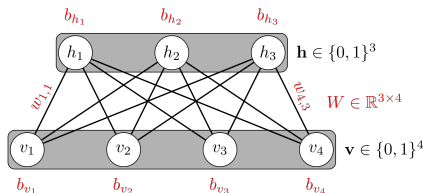
- binarne jednostki umieszczone w dwóch warstwach: widzialnej $\mathbf{v}$ i ukrytej $\mathbf{h}$
- sygnał wejściowy podawany do jednostki widzialnej $\mathbf{v} = \mathbf{x}$
- energia układu

$$E(\mathbf{x}, \mathbf{h}) = - \sum a_i x_i - \sum b_i h_i - \sum \sum x_i w_{ij} h_j$$

gdzie w_{ij} to wagi połączeń między warstwami, a_i i b_i to wyrazy wolne związane z warstwą widzialną i ukrytą

- prawdopodobieństwo rozkładu

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{x}, \mathbf{h})}$$



src: <https://kwonkyo.wordpress.com/>

RBM

- prawdopodobieństwo marginalne

$$P(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})}$$

- brak połączeń wewnątrz warstw, więc stany w warstwach są niezależne

$$P(\mathbf{h}|\mathbf{x}) = \prod_{j=1}^n P(h_j|\mathbf{x}), \quad P(\mathbf{x}|\mathbf{h}) = \prod_{i=1}^m P(x_i|\mathbf{h})$$

$$P(h_i = 1|\mathbf{x}) = \sigma(\mathbf{w}_i\mathbf{x} + b), \quad P(x_k = 1|\mathbf{h}) = \sigma(\mathbf{h}^T \mathbf{w}_k + c)$$

Contrastive divergence CD

- maksymalizacja prawdopodobieństwa $P(\mathbf{x})$

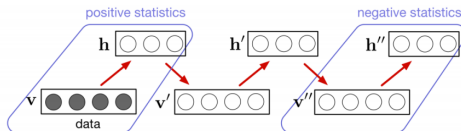
$$L = \frac{1}{N} \sum \log P(\mathbf{x})$$

- *Contrastive divergence* (Hinton) metoda przybliżona korzystająca z samplowania Gibbsa
 - dla każdego $\mathbf{x}$ wejściowego wygeneruj $\mathbf{x}'$ z k krotnego samplowania Gibbsa
 - aktualizacja

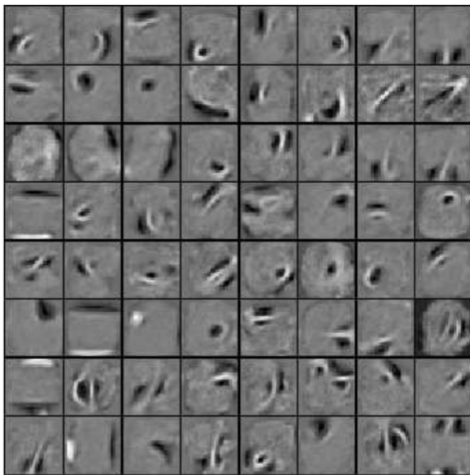
$$\Delta \mathbf{W} = \epsilon(\mathbf{x}\mathbf{h}^T - \mathbf{x}'\mathbf{h}'^T)$$

$$\Delta \mathbf{a} = \epsilon(\mathbf{x} - \mathbf{x}'), \quad \Delta \mathbf{b} = \epsilon(\mathbf{h} - \mathbf{h}')$$

- w praktyce $k = 1$ daje dobre cechy

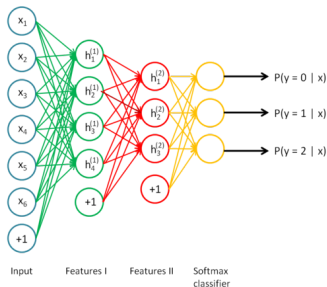


Filtry RBM na MNIST



Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

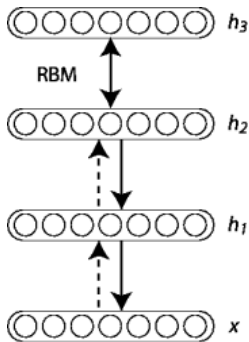
Stacked autoencoders



- Sygnał na wyjściu warstwy ukrytej jest wejściem kolejnego autoenkodera
- Hierarchia reprezentacji: pierwsza warstwa koduje najprostsze warstwy, kolejne bardziej ogólne relacje, itd.
- Podobna architektura: wielowarstwowy RBM to Deep Belief Networks

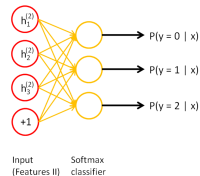
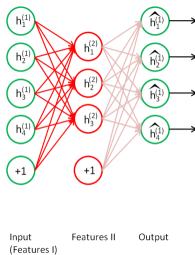
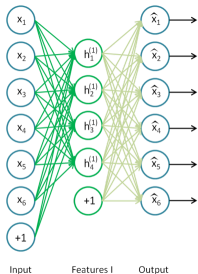
Deep Belief Networks (DBN)

- DBN (Teh, Hinton 2006)
wielowarstwowy RBM,
- uczy się głębokiej, hierarchicznej reprezentacji danych, każda kolejna warstwa analizuje sygnał widoczny w warstwie ukrytej poprzedniego RBM
- uczenie zachłanne: kolejne warstwy dodawane i uczone pojedynczo
- pierwszy efektywny model głęboki.
- to nie jest sieć jednokierunkowa



src: <http://deeplearning.net/tutorial/DBN.html>

Inicjalizacja sieci DNN



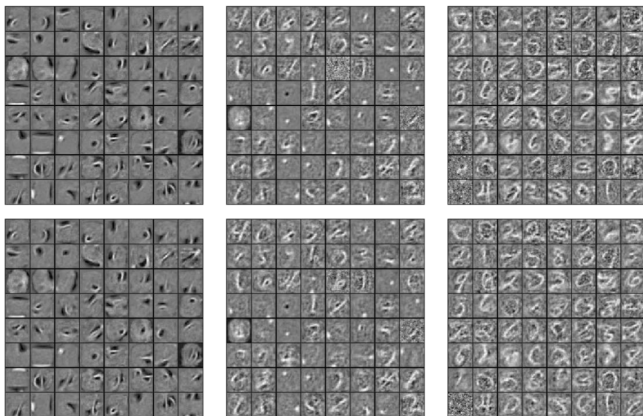
1. Greedy layer-wise pre-training:

- trening nienadzorowany autoenkodera wsteczną propagacją
- usuwamy warstwę wyjściową (dekoder) i uczymy kolejny AE, którego wejściem jest sygnał z warstwy ukrytej poprzedniego AE przy niezmiennych wartościach wag w poprzednich warstwach
- powtarzamy procedurę dla każdej kolejnej warstwy

2. fine tuning: dodajemy w pełni połączoną warstwę wyjściową (lub kilka warstw) i uczymy całą sieć w sposób nadzorowany

Filtry DBN na MNIST

pre-training



Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

fine tune

Wyniki MNIST, Bengio 2007

	train.	valid.	test
DBN, unsupervised pre-training	0%	1.3%	1.4%
Deep net, auto-associator pre-training	0%	1.4%	1.4%
Deep net, supervised pre-training	0%	1.75%	2.0%
Deep net, no pre-training	.004%	2.1%	2.4%
Shallow net, no pre-training	.004%	1.8%	1.9%

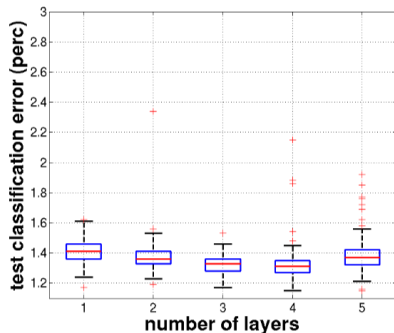
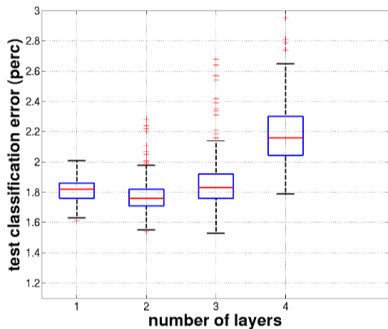
Table 2: *Classification error on MNIST training, validation, and test sets, with the best hyper-parameters according to validation error, with and without pre-training, using purely supervised or purely unsupervised pre-training.*

- architektura: 784 wejść, 10 wyjść, 3 warstwy ukryte
- na MNIST 0.1% statystycznie istotna różnica

Bengio, et al., Greedy layer-wise training of deep networks, 2007.

Pre-training DNN

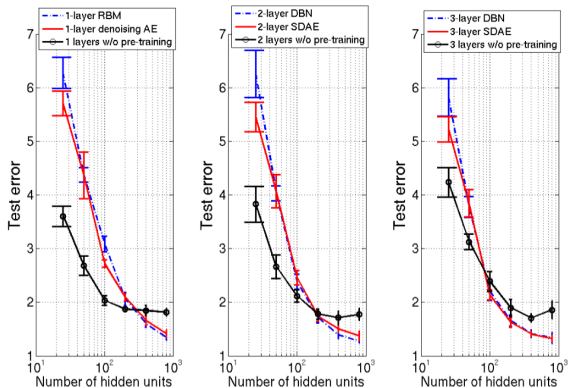
Pre-trening poprawia generalizację, działa jak regularyzacja



Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

Pre-training DNN

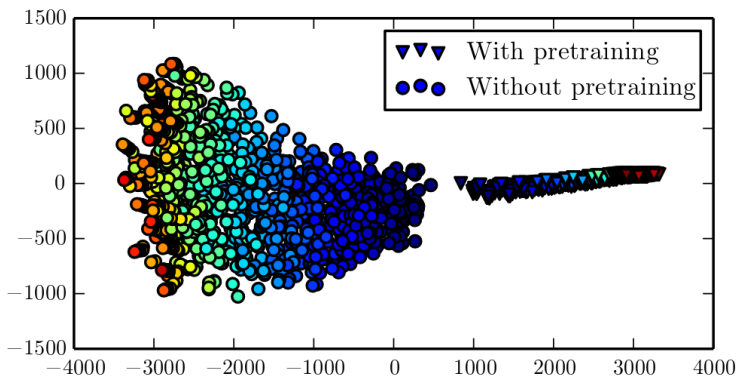
Wartość pre-treningu wzrasta z liczbą warstw



Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

Pre-training DNN

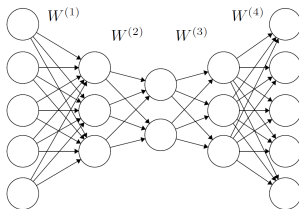
Pre-training ma wpływ na trajektorię uczenia



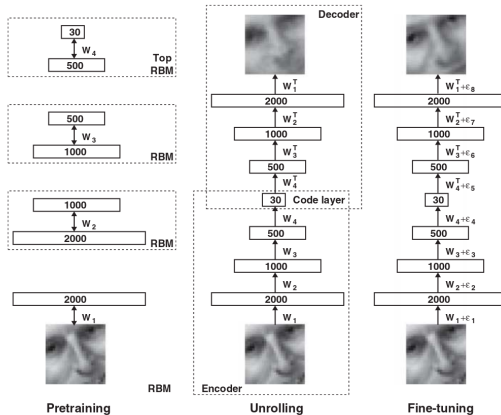
Goodfellow, 2016

Głębokie autoenkodery

- dodatkowa warstwa ukryta zwiększa możliwości enkodera w odwzorowaniu kodu (uniwersalny aproksymator jest w stanie nauczyć się dowolnego kodowania)
- dodanie warstw pozwala zmniejszyć złożoność reprezentacji trudnych problemów
- głębokie AE pozwalają uzyskać większą kompresję (Hinton 2006)
- niekompletne głębokie AE - żadna warstwa ukryta nie powinna być mniejsza niż warstwa kodująca



Pretraining dla głębokich AE

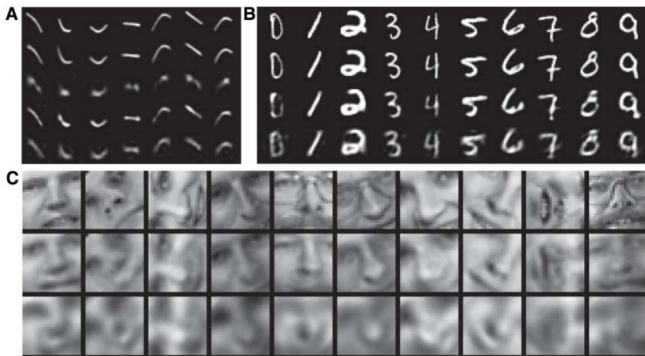


- RBM używany do inicjacji kodera i dekodera

G. Hinton, R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science 2006

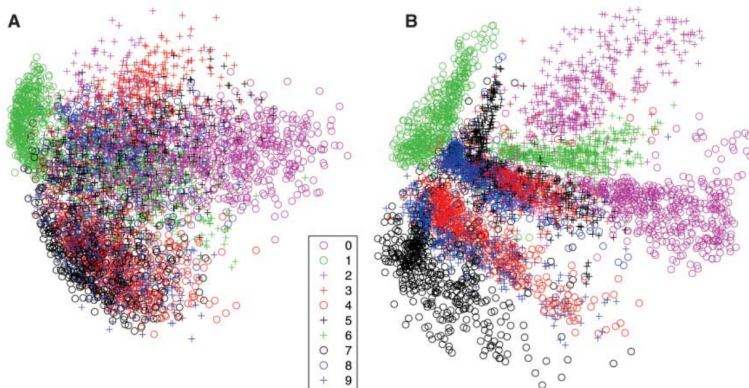
Komresja obrazów: Deep AE vs. PCA

Fig. 2. (A) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by "logistic PCA" (8) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error per image for the last four rows is 1.44, 7.64, 2.45, 5.90. (B) Top to bottom: A random test image from each class; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional logistic PCA and standard PCA. The average squared errors for the last three rows are 3.00, 8.01, and 13.87. (C) Top to bottom: Random samples from the test data set; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.



Projekcja do 2D (dwa neurony kodujące)

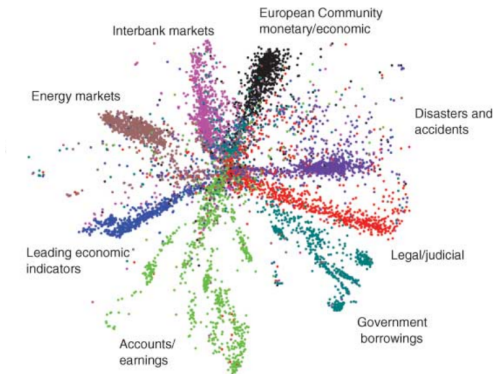
MNIST: PCA vs. Deep AE (784-1000-500-250-2)



G. Hinton, R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science 2006

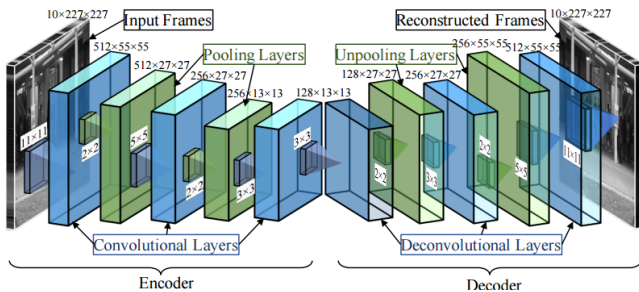
Projekcja do 2D (dwa neurony kodujące)

Wizualizacja dokumentów: architektura 2000-500-250-125-2



G. Hinton, R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science 2006

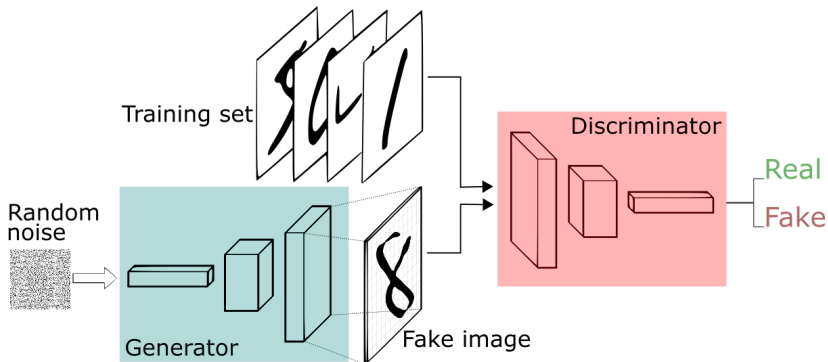
Splotowe autoenkodery



Generative Adversal Network (GAN)

- GAN (Goodfellow 2014) - dwie sieci (generująca i oceniająca) rywalizujące ze sobą w grze o sumie zerowej
- zastosowanie: generowanie zdjęć o realistycznych cechach
- sieć oceniająca (dyskryminująca, np. CNN) stara się odróżnić prawdziwy sygnał od wygenerowanego przez sieć generującą
- sieć generująca (np. sieć dekonwolucyjna) tworzy sygnał z pewnego rozkładu starając się „oszukać” sieć oceniającą, dąży do maksymalizacji błędu dyskryminacji
- obie sieci uczone wsteczną propagacją, sieć generująca tworzy coraz bardziej realistyczne obrazy, sieć oceniająca specjalizuje się w rozróżnianiu coraz subtelniejszych różnic pomiędzy obrazami prawdziwymi i wygenerowanymi

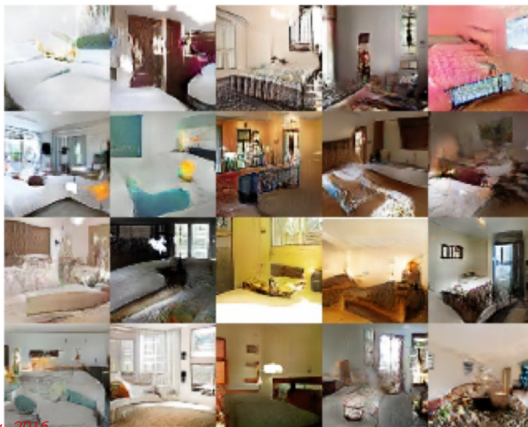
GAN



src: <https://deeplearning4j.org/generative-adversarial-network>

Przykłady

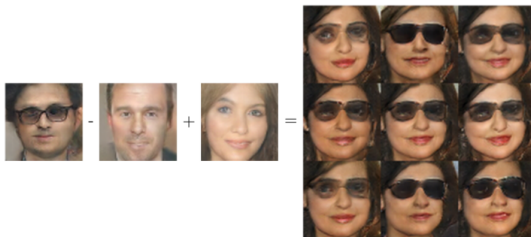
DCGAN (Denton et al., 2015) generowanie obrazów wysokiej rozdzielczości



Goodfellow, Deep Learning, 2016

Ukryta reprezentacja pojęć

- utworzona ukryta reprezentacja koduje pojęcia
- arytmetyczne operacje na wektorach kodujących odpowiadają relacjom semantycznym pomiędzy pojęciami





👉 AI i uczenie maszynowe przewidują wyniki MS 2018