Uczenie nienadzorowane

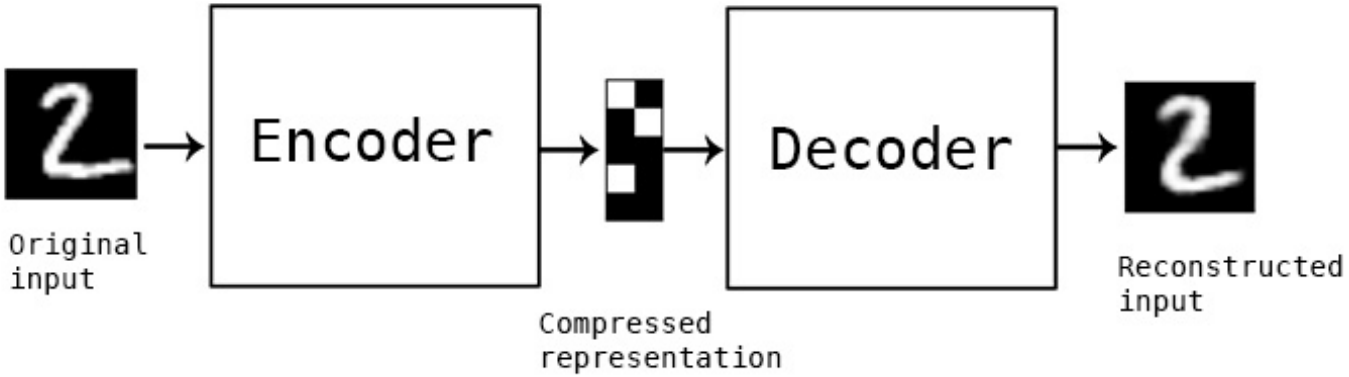
Autoenkodery

Modele generatywne

Uczenie nienadzorowane

- Uczenie **nadzorowane** (*supervised*) - dane wejściowe $\mathbf{X}$ posiadają spodziewane wyjście (etykiety) $(\mathbf{X}, \mathbf{Y})$
- Uczenie **nienadzorowane** (*unsupervised*) używa wyłącznie danych wejściowych $\mathbf{X}$. Dane bez etykiet są powszechnie występujące.
  - trening automatycznie wykrywa istotne cechy w danych,
  - modeluje rozkład prawdopodobieństwa danych
- Modele sieci:
  - autoenkodery (AE), splotowe AE (CAE), Variational AE (VAE)
  - Restricted Boltzman Machines (RBM)
  - Generative Adversarial Networks (GAN)
- Zastosowanie:
  - kodowanie sygnału, wykrywanie istotnych cech, usuwanie szumu i rekonstrukcja sygnału, głęboka klaseryzacja
  - generowanie sygnałów, transfer stylu
  - inicjowanie głębokich sieci DNN (stacked autoencoders)

Autoenkodery



- autoenkodery uczą się odtwarzać sygnał wejściowy
- warstwy pośrednie kodują specyficzne cechy danych
- kompresja stratna - wymagamy aby model nie był jednostkowym odwzorowaniem
- AE uczą się reprezentacji danych (ekstrakcja cech), modelują rozkład prawdopodobieństwa sygnału
- koder i dekoder mogą być sieciami neuronowymi, całość uczona wsteczną propagacją błędu

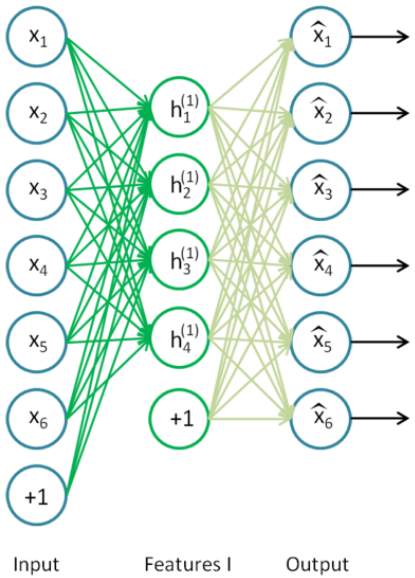
Rys: ☞ Building Autoencoders in Keras

Autoenkodery

Encoder:

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

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



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 $\mathbf{h}$: koduje sygnał wejściowy, detektor cech, skompresowana reprezentacja danych
- często wymaga się aby $\mathbf{W}' = \mathbf{W}^T$, współdzielone wagi kodera i dekodera

Źródło grafiki: `http://ufldl.stanford.edu`

Koszt rekonstrukcji

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

- MSE dla danych wejściowych rzeczywistych

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

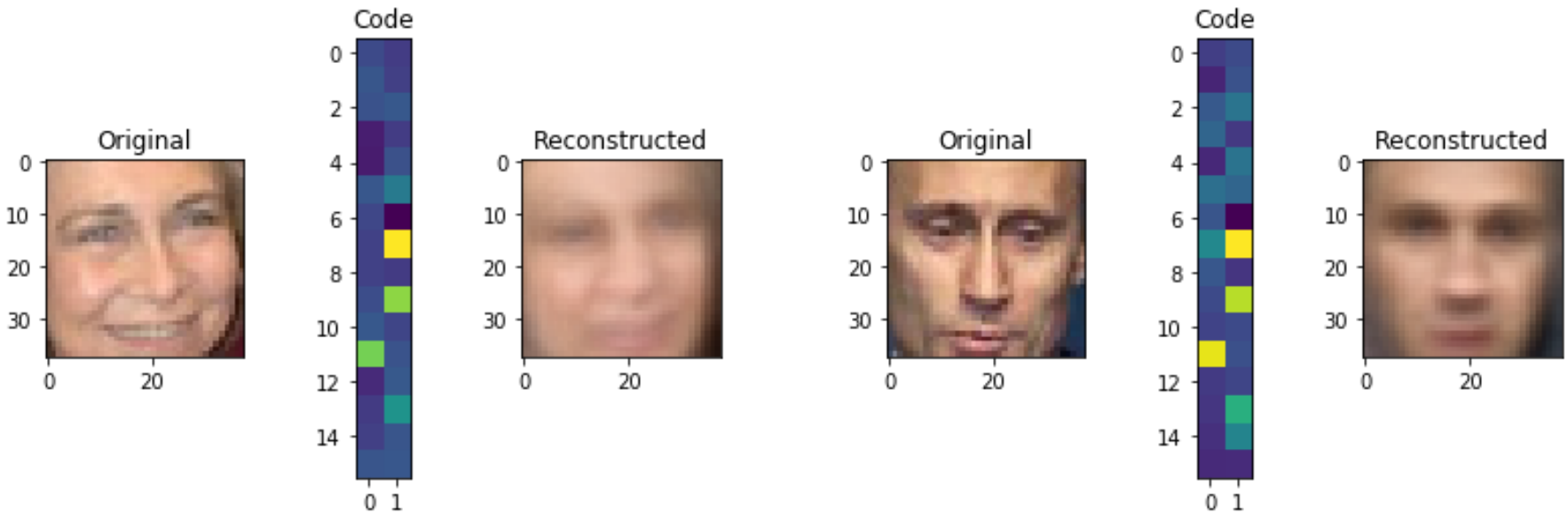
  - liniowa aktywacja warstwy wyjściowej $\hat{\mathbf{x}} = \mathbf{W}'\mathbf{h} + \mathbf{b}'$
- Cross Entropy dla danych binarnych (lub z zakresu $[0, 1]$)

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

- trening wsteczną propagacją błędu
- ograniczenia zapobiegające tworzeniu odwzorowania jednostkowego: niekompletny AE, regularyzowany AE, kurczliwy AE, ...

Niekompletny autokoder

- **Niekompletny AE** (*undercomplete*) - rozmiar warstwy **h** mniejszy od rozmiaru wejścia,
- kompresja sygnału do rozmiaru **h**
- dla liniowego dekodera wynik jest równoważny z PCA, wagi neuronów tworzą składowe główne maksymalizujące wariancję
- Przykład: input-output: 32x32 (1024 piksele) ,
  hidden (code size): 32



Źródło grafiki: ☞ Manash's blog

Regularyzowany AE

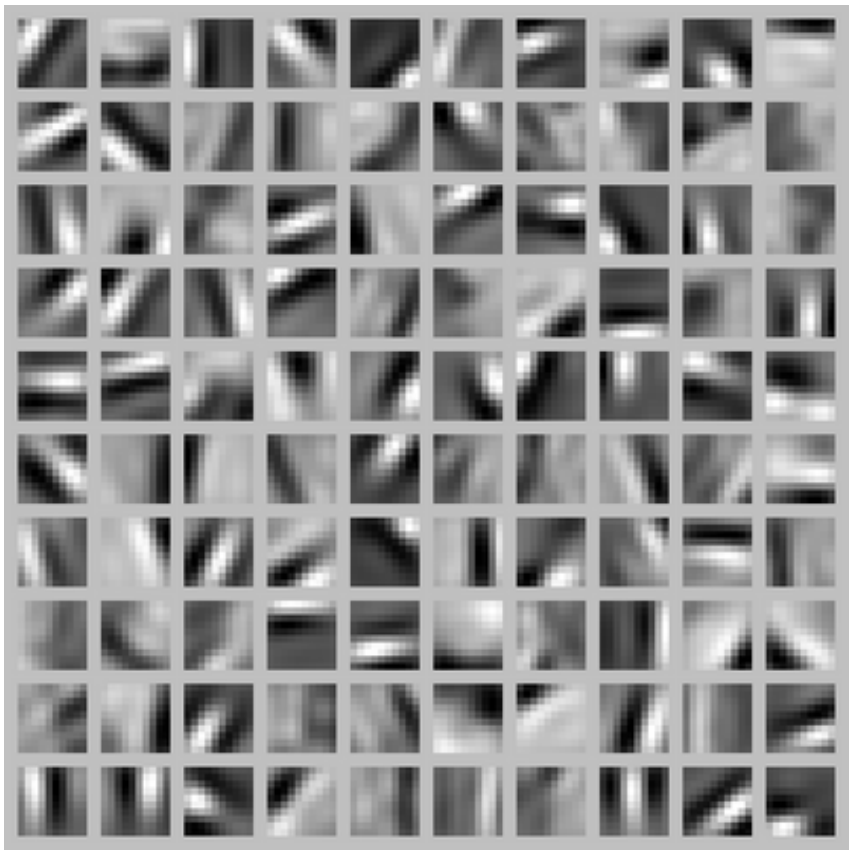
- wielkość **h** decyduje o pojemności sieci i jakości odwzorowania. Dla niekompletnych AE jest ograniczenie zwiększania rozmiaru warstwy ukrytej
- **Nadkompletny** AE (*overcomplete*) gdy rozmiar **h** większy od rozmiaru sygnału wejsciowego **x**,
  - brak kompresji
  - nie gwarantują uzyskania wartościowych reprezentacji bez dodania odpowiedniej regularyzacji wymuszającej odpowiednią reprezentację w **h** (np. rzadkość)
- **Rzadki autoencoder** z czynnikiem kary za brak rzadkiej reprezentacji

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

  - w celu uzyskania rzadkich aktywacji można też użyć ReLU (Glorat, 2011)

Przykład

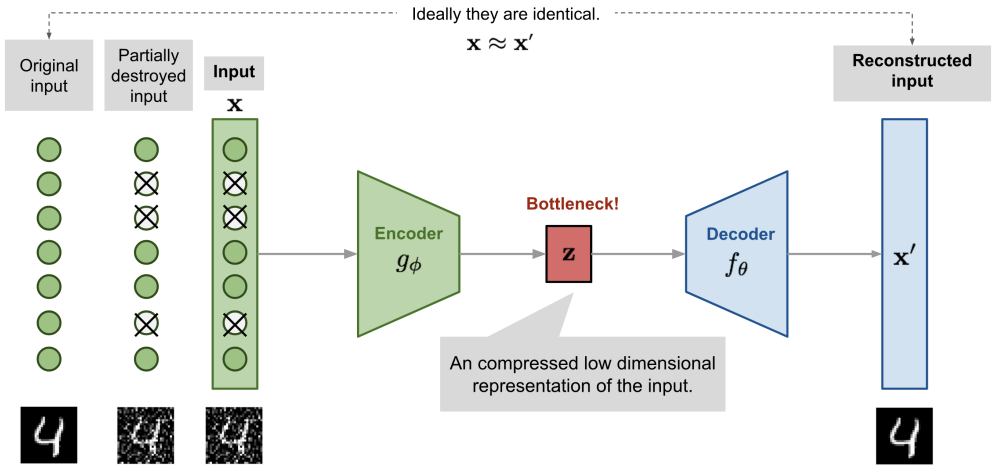
Cechy uzyskane w warstwie ukrytej przez autoenkoder rzadki (regularyzacja L_1) trenowany na naturalnych obrazach (bez etykiet!)



Oja, E. (1982). A simplified neuron model as a principal component analyzer. Journal of mathematical biology, 15(3), 267–73.

Autoenkoder z odszumianiem (DAE)

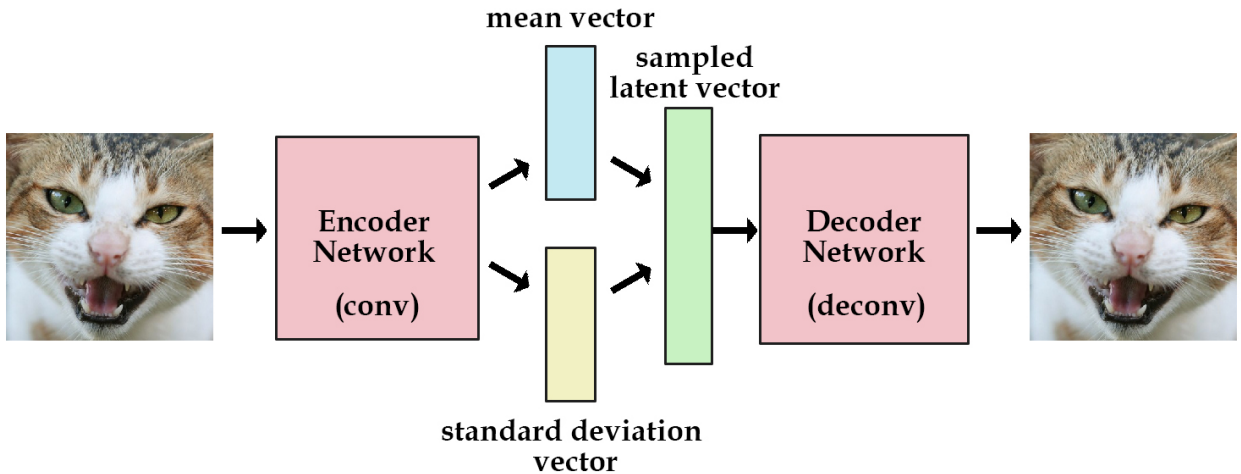
- **Denoise autoencoder (DAE)** usuwa szum (lub inne zniekształcenia) sygnału wejściowego.
- Sygnał wejściowy $\tilde{\mathbf{x}}$ jest zniekształconą kopią $\mathbf{x}$
- AE uczy się odtwarzać oryginalny sygnał bez deformacji



src: ☞ *Building Autoencoders in Keras*
☞ *From Autoencoder to Beta-VAE*

Wariacyjny AE

- **Variational autoencoders (VAE)** jawnie modeluje rozkład gęstości danych, kodowana reprezentacja to parametry funkcji gęstości, np. σ_i, μ_i rozkładu normalnego $N(\mu_i, \sigma_i)$



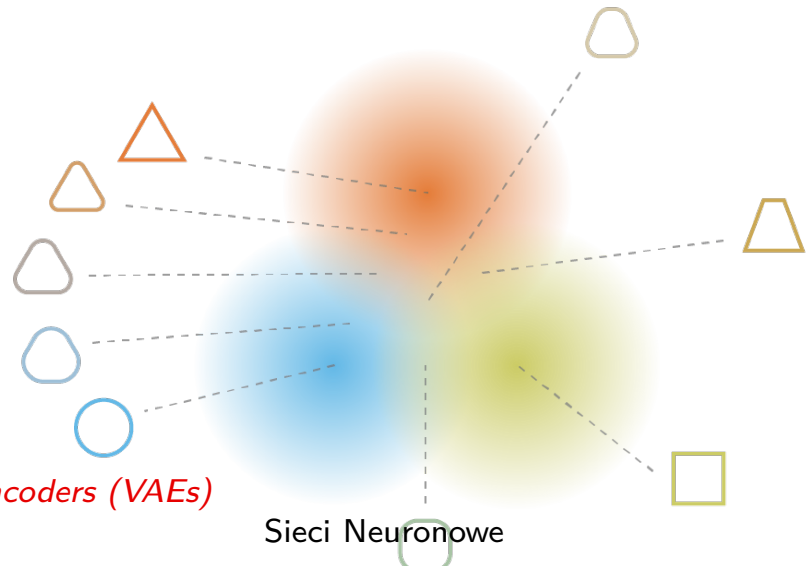
- sygnał wejściowy dekodera jest próbkowany z rozkładu określonego przez parametry warstwy kodującej
- model generatywny - umożliwia próbkowanie danych wyjściowych, tworzenie nowych sygnałów o charakterystyce zbliżonej do danych treningowych

Kullback–Leibler divergence

- czynnik regularyzujący, dodawany do funkcji kosztu (np. MSE), wymuszający odpowiedni rozkład danych w warstwie kodującej (KL divergence), np. preferujący rozkład $N(0, 1)$

$$\sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1$$

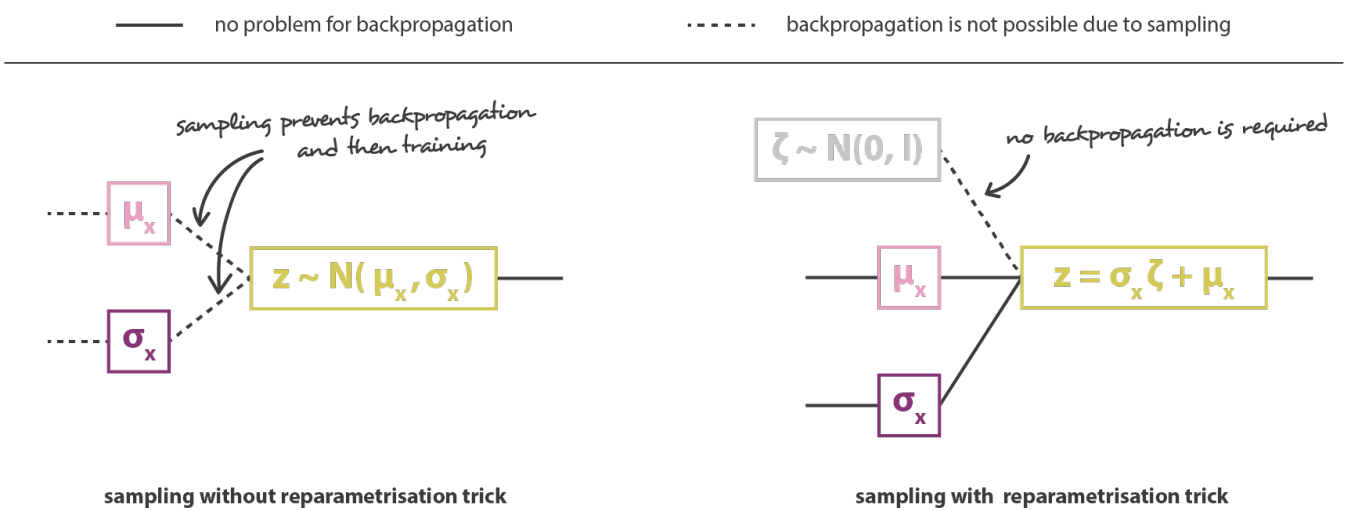
- osiąga minimum dla $\sigma_i = 1, \mu_i = 0$
- wymusza rozkład zakodowanych danych w centrum przestrzeni ukrytej, minimalizacja kosztu rekonstrukcji konkuruje z regularyzajcą próbując rozdzielić grupy różnych danych



☞ *Understanding Variational Autoencoders (VAEs)*

Representation trick (stochastyczne BP)

W jaki sposób zrealizować wsteczną propagację błędu?

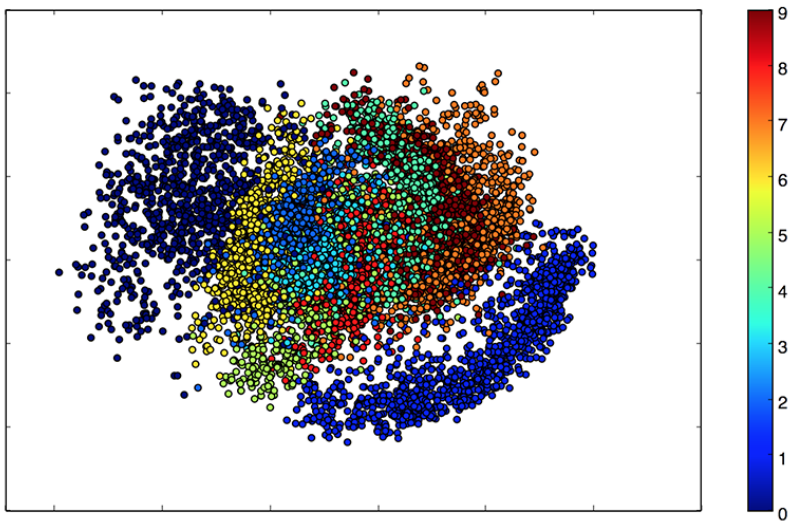


$$\mathbf{z} = \mu_{\mathbf{x}} + \sigma_{\mathbf{x}}\xi$$

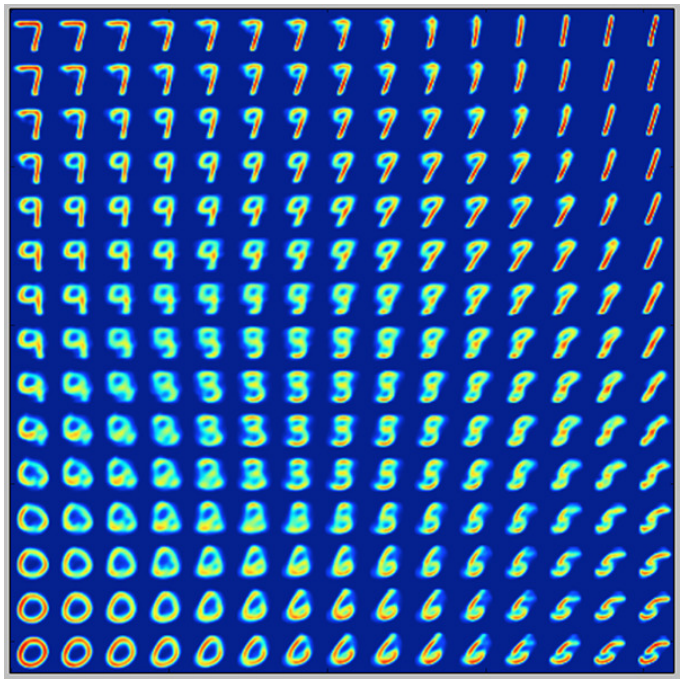
Próbka $\xi \sim \mathcal{N}(0, 1)$ stanowi kolejne wejście sieci neuronowej jednak nie posiadające parametrów optymalizacyjnych (nie propagujemy błędu tym wejściem).
Całą sieć można więc trenować wsteczną propagacją.

VAE na MNIST

wizualizacja danych treningowych w 2 wymiarowej warstwie ukrytej kodera



generowanie cyfr dla siatki 15x15 wartości 2 wymiarowej warstwy ukrytej dekodera

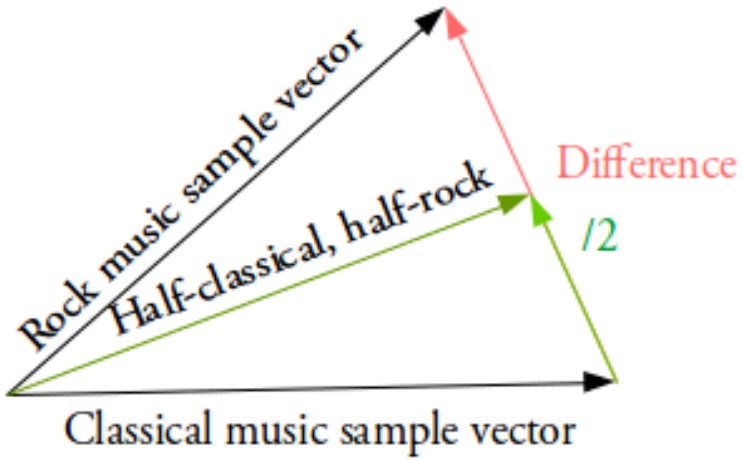


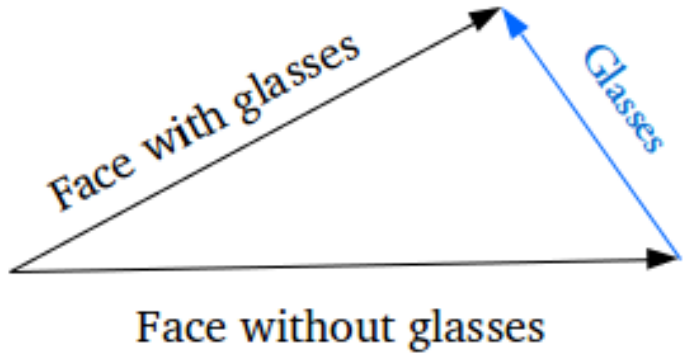
Architektura: Input-512-2-(μ, σ)-2-512-Output

src: ☞ *Building Autoencoders in Keras*

Interpolowanie i generowanie pojęć

Arytmetyka w przestrzeni zmiennych utajonych





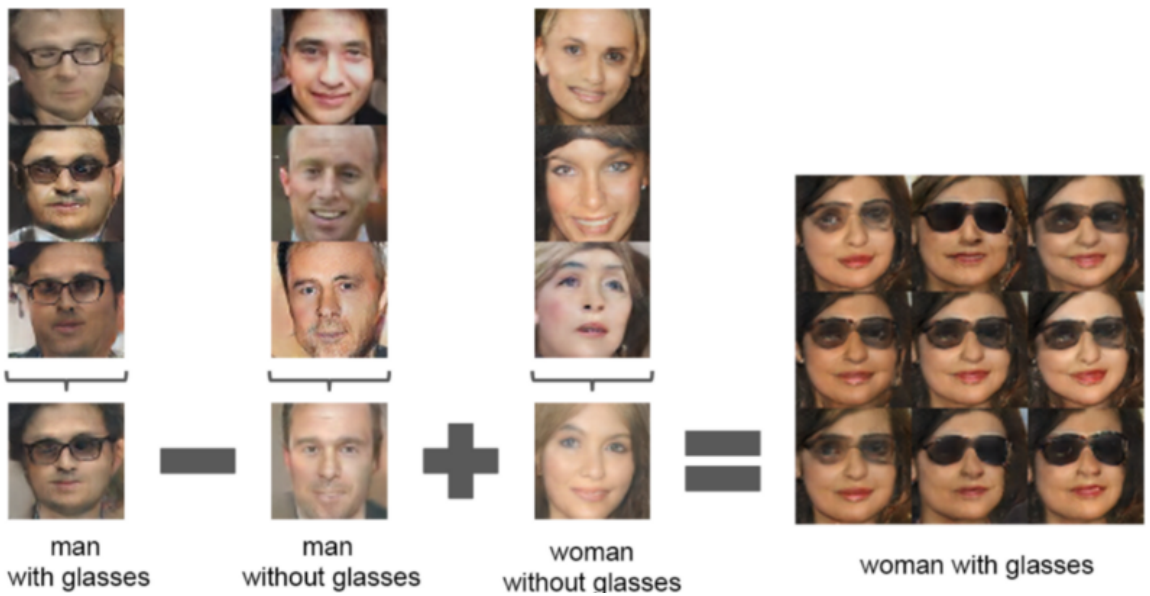
Środek między wektorami μ kodującymi pojęcia

Róznica pomiędzy pojęciami może być dodana do innego pojęcia

☞ *Irhum Shafkat, Intuitively Understanding Variational Autoencoders*

Ukryta reprezentacja pojęć

- wyuczona reprezentacja danych koduje pojęcia
- arytmetyczne operacja na wektorach kodujących odpowiadają relacjom semantycznym pomiędzy pojęciami



Przykład: generowanie twarzy

Wejście

Rekonstrukcja

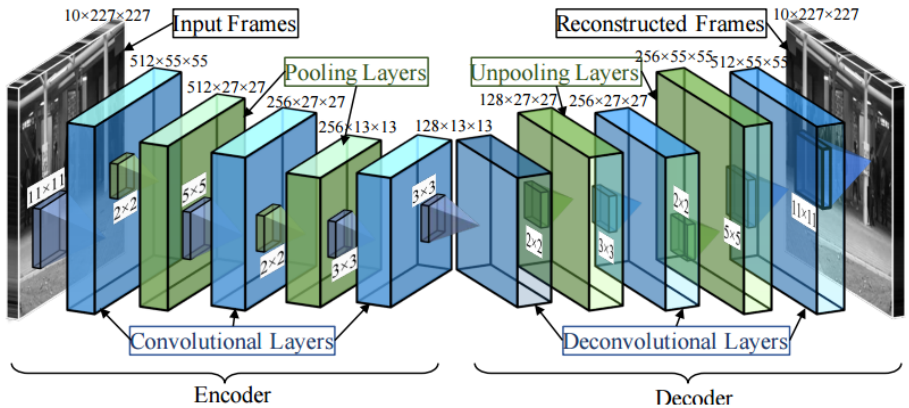
Losowe

☞ Deep Feature Consistent Variational Autoencoder

☞ *Variational auto-encoder trained on celebA*

Inne typy autoenkoderów

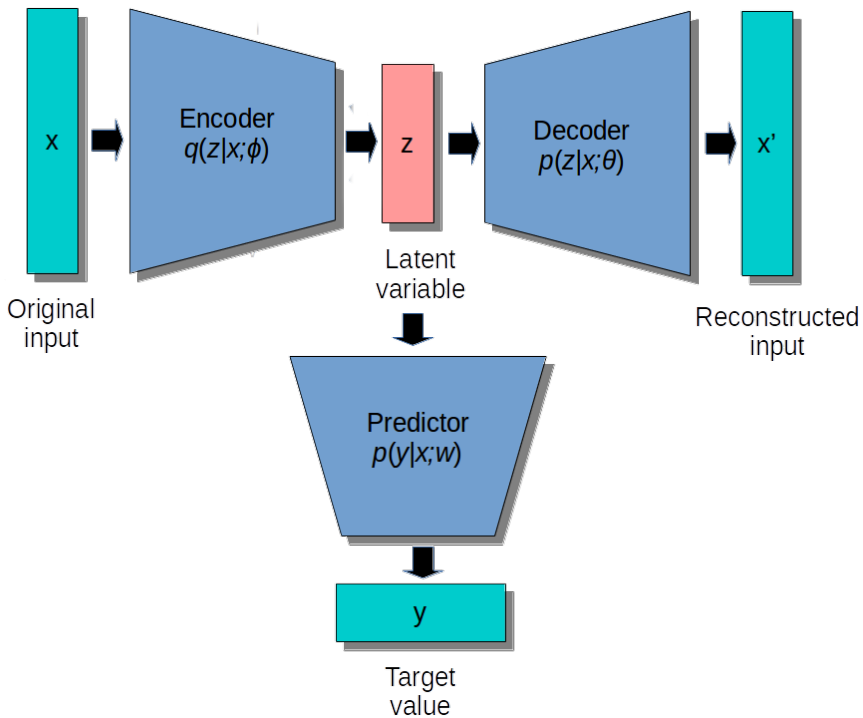
- Convolutional AE (CAE), dekoder jest symetryczną względem enkodera siecią dekonwolucyną



- seq2seq: model RNN endoder-decoder do modelowania sekwencji
- głębokie autoenkodery (DAE) - więcej warstw sprzyja kompresji danych o złożonych rozkładach
- stacked autoencoders (SAE) - tworzenie głębokich sieci z uczonych sekwencyjnie autoenkoderów

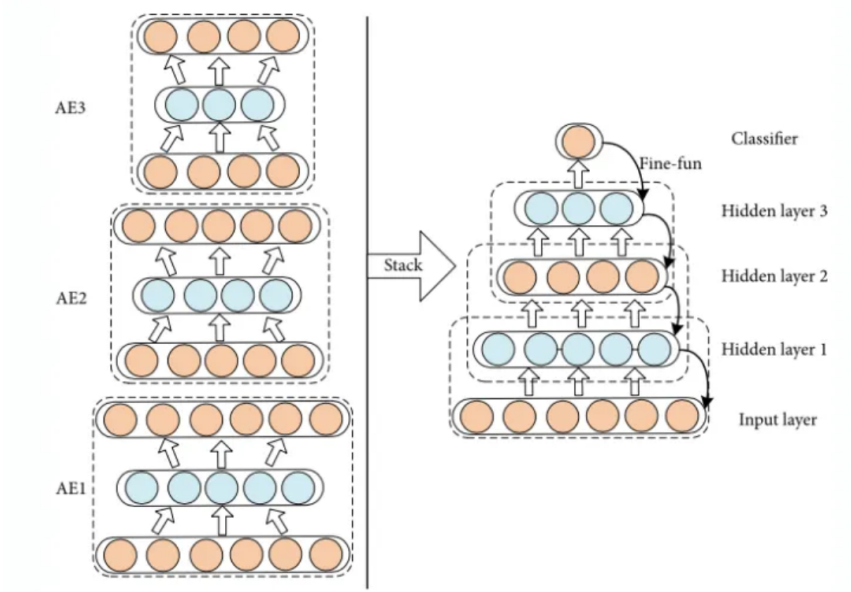
Uczenie pół-nadzorowane (samouczenie)

Ucznie pół-nadzorowane (**semi-supervised**) lub samouczenie (**self-taught**): najpierw trenujemy AE na dużym zbiorze danych w sposób nienadzorowany, następnie reprezentacja danych uzyskana z kodera wykorzystywana jest do uczenia płytkiego klasyfikatora na małym zbiorze z etykietami



Guo et al., 2017, Deep convolutional autoencoder

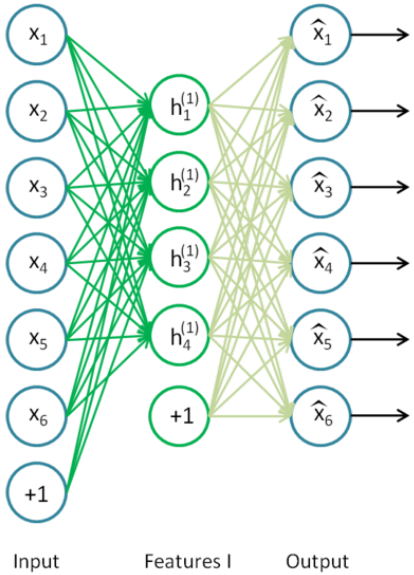
Stacked autoencoders

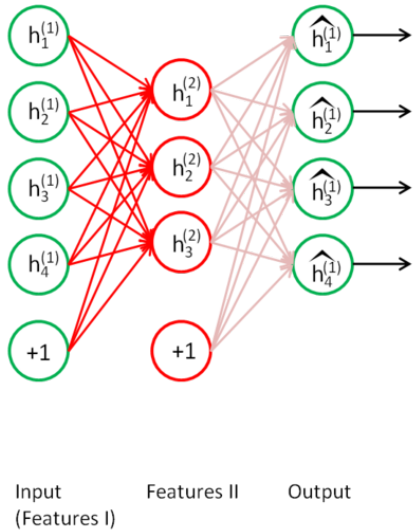


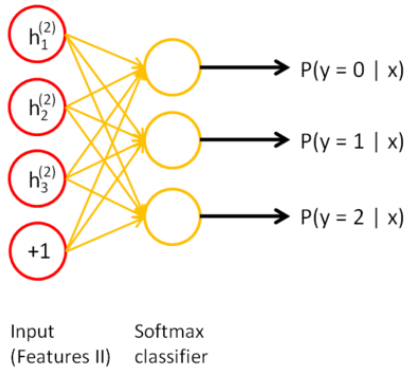
- *Stacked Autoencoder (SAE)*: kilku warstw autoenkoderów ułożonych jeden na drugim
- każda warstwa autoenkodera jest trenowana osobno, najczęściej w trybie uczenia nienadzorowanego, aby nauczyć się coraz bardziej abstrakcyjnych reprezentacji danych

☞ *Deep Clustering with Convolutional Autoencoder*

Inicjalizacja głębokich sieci za pomocą SAE



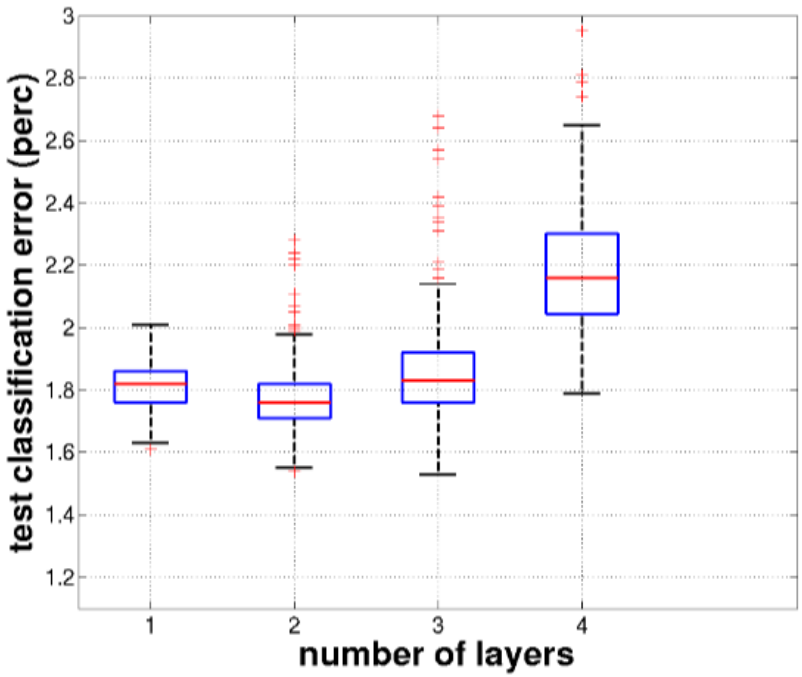


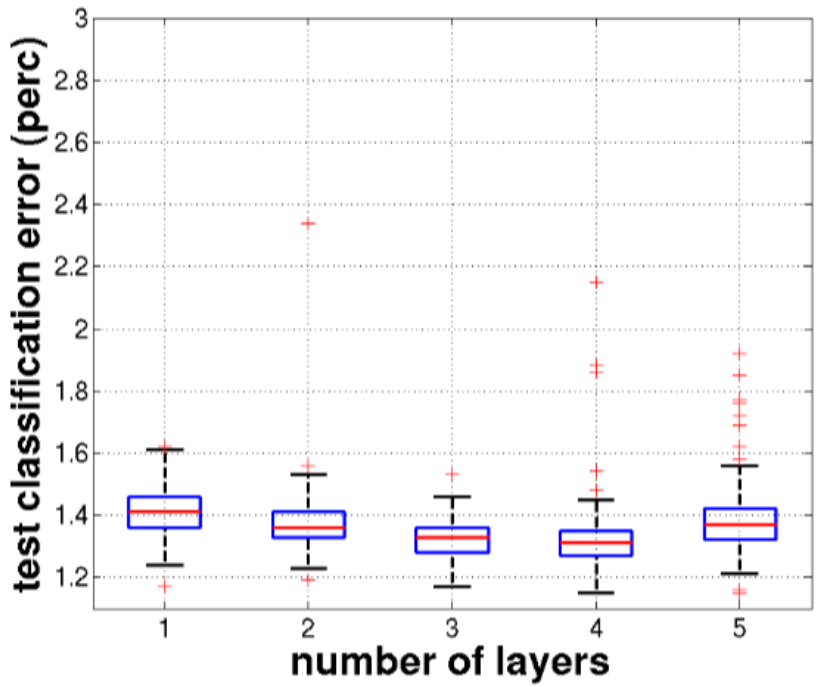


1. *Greedy layer-wise pre-training*:
   - trening nienadzorowany autoenkodera wsteczną propagacją
   - usuwamy warstwę wyjściową (dekoder) i uczymy kolejny AE, którego wejściem jest sygnał z warstwy ukrytej poprzedniego AE przy niezmiennych wartościach wag w poprzednich warstwach
   - powtarzamy procedurę dla każdej kolejnej warstwy
2. *fine tuning*: dodajemy w pełni połączoną warstwę wyjściową (lub kilka warstw) i uczymy całą sieć w sposób nadzorowany

Pre-trening DNN

Pre-trening poprawia generalizację, działa jak regularyzacja

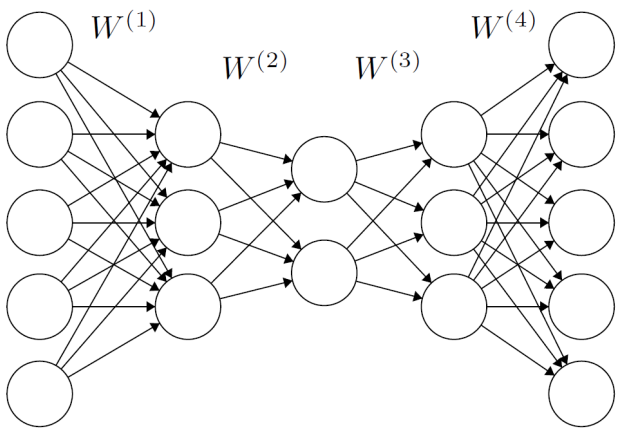




Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

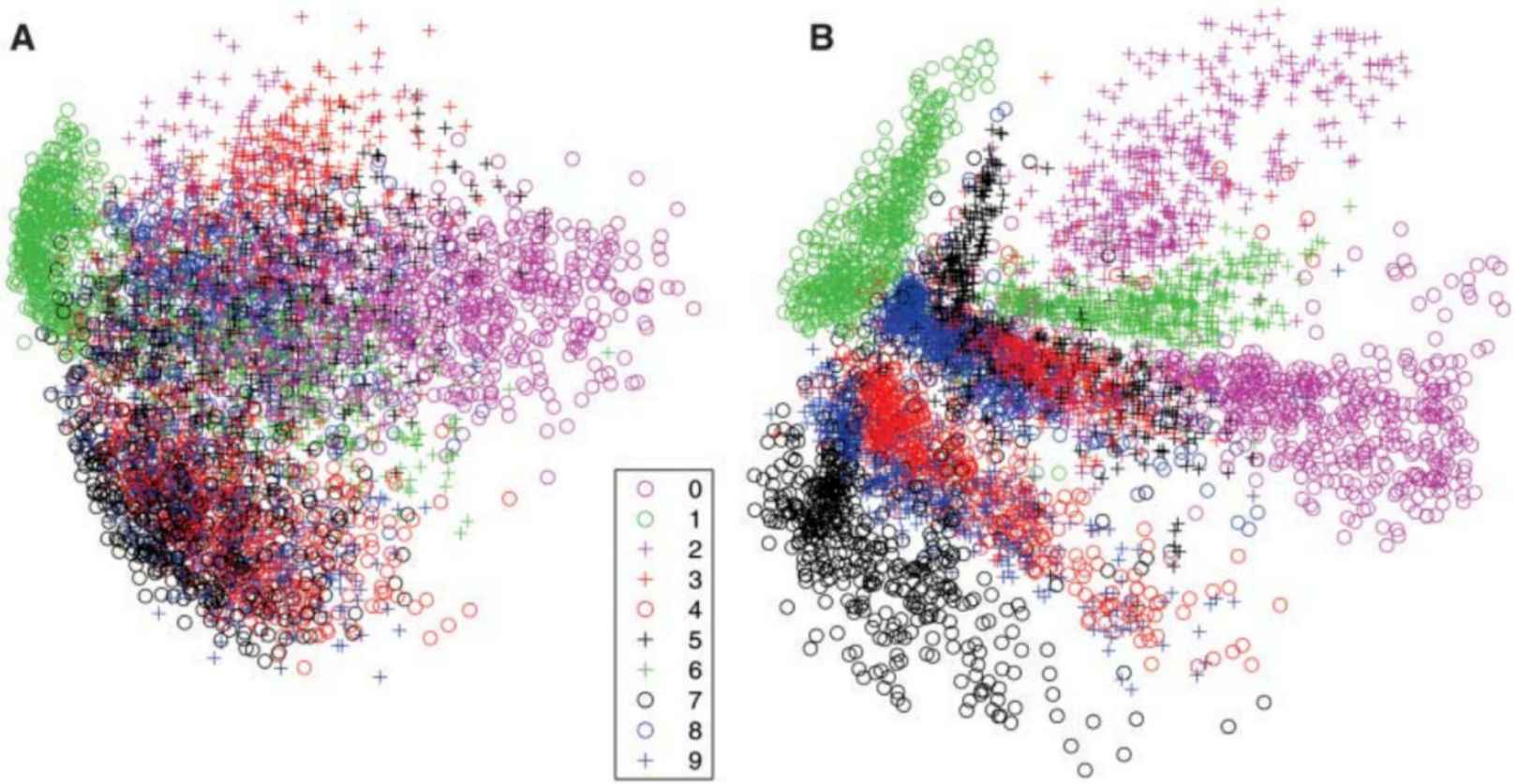
Głębokie autoenkodery

- 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



Projekcja do 2D (dwa neurony kodujące)

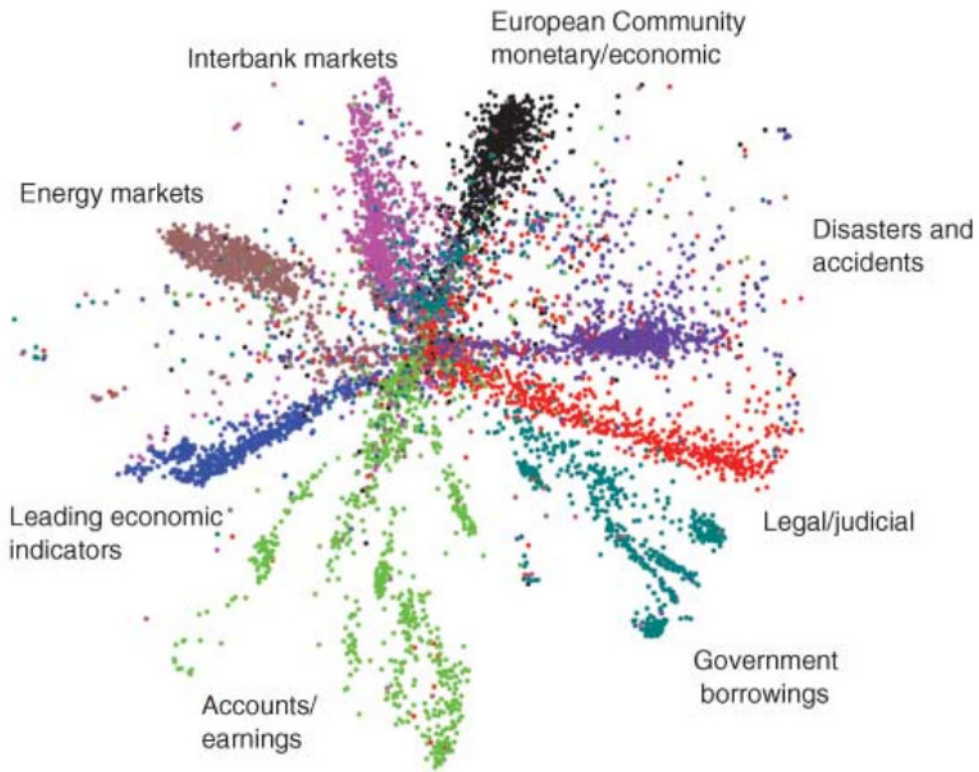
MNIST: PCA vs. Deep AE (784-1000-500-250-2)



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

Projekcja do 2D (dwa neurony kodujące)

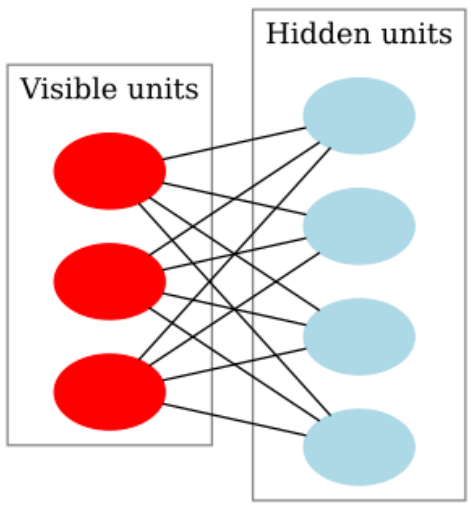
Wizualizacja dokumentów: architektura 2000-500-250-125-2



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

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 nad zbiorem swoich wejść
- jest ograniczona (*restrictive*), bo brak połączeń w obrębie jednej warstwy
- algorytm uczenia: **dywergencja kontrastowa** (*contrastive divergence*)



Źródło grafiki: Wikipedia

Budowa RBM

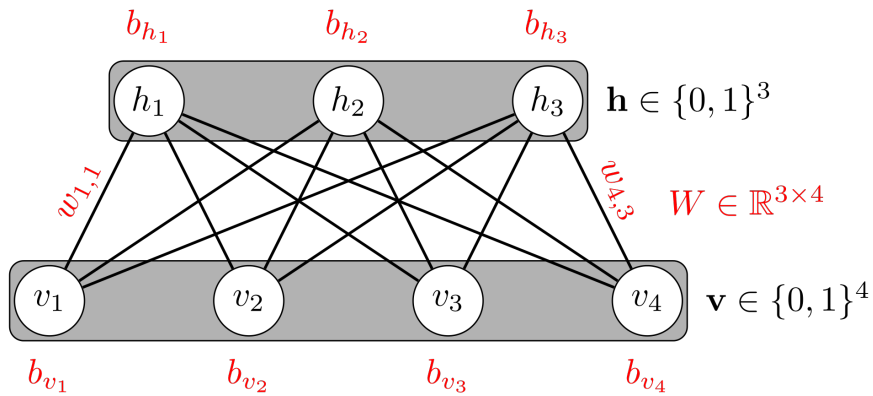
- binarne jednostki umieszczone w dwóch warstwach: widzialnej **v** i ukrytej **h**
- sygnał wejściowy podawany do jednostki widzialnej $\mathbf{v} = \mathbf{x}$
- energia układu

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

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

- rozkład brzegowy

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

- brak połączeń wewnątrz warstw, więc stany w warstwach są niezależne. Prawdopodobieństwo aktywacji:

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

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

- model generuje próbki widocznych danych przez iteracyjne próbkowanie jednostek ukrytych i widocznych (*Gibbs sampling*)

Contrastive divergence CD

- cel treningu: maksymalizacja prawdopodobieństwa rozkładu danych $P(\mathbf{x})$

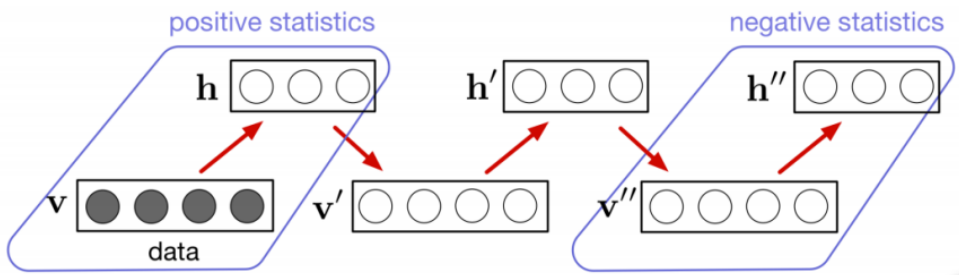
$$L = \frac{1}{N} \sum \log P(\mathbf{x})$$

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

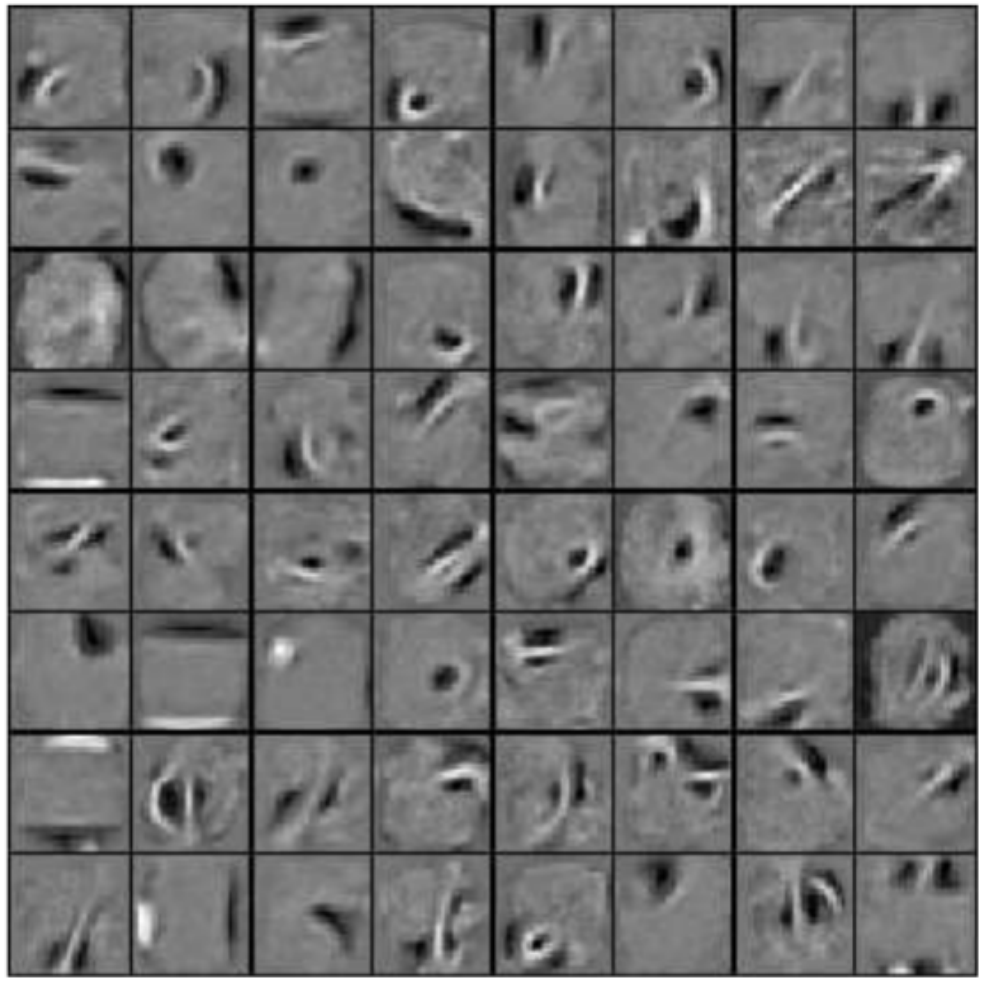
$$\Delta \mathbf{W} = \epsilon(\mathbf{x}\mathbf{h}^T - \mathbf{x}'\mathbf{h}'^T)$$

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

- RBM uczy się przybliżać rozkład danych, minimalizując różnicę między obserwowanymi a rekonstruowanymi próbkami
- w praktyce $k = 1$ (pojedyncze próbkowanie) daje dobre rezultaty



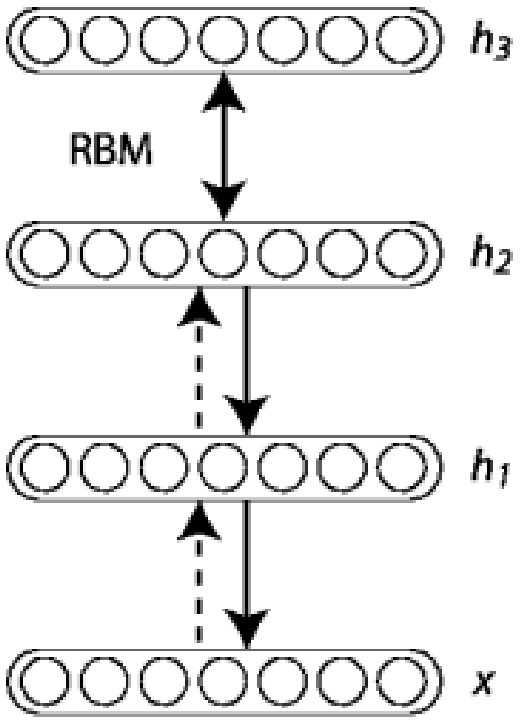
Filtry RBM na MNIST



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

Deep Belief Networks (DBN)

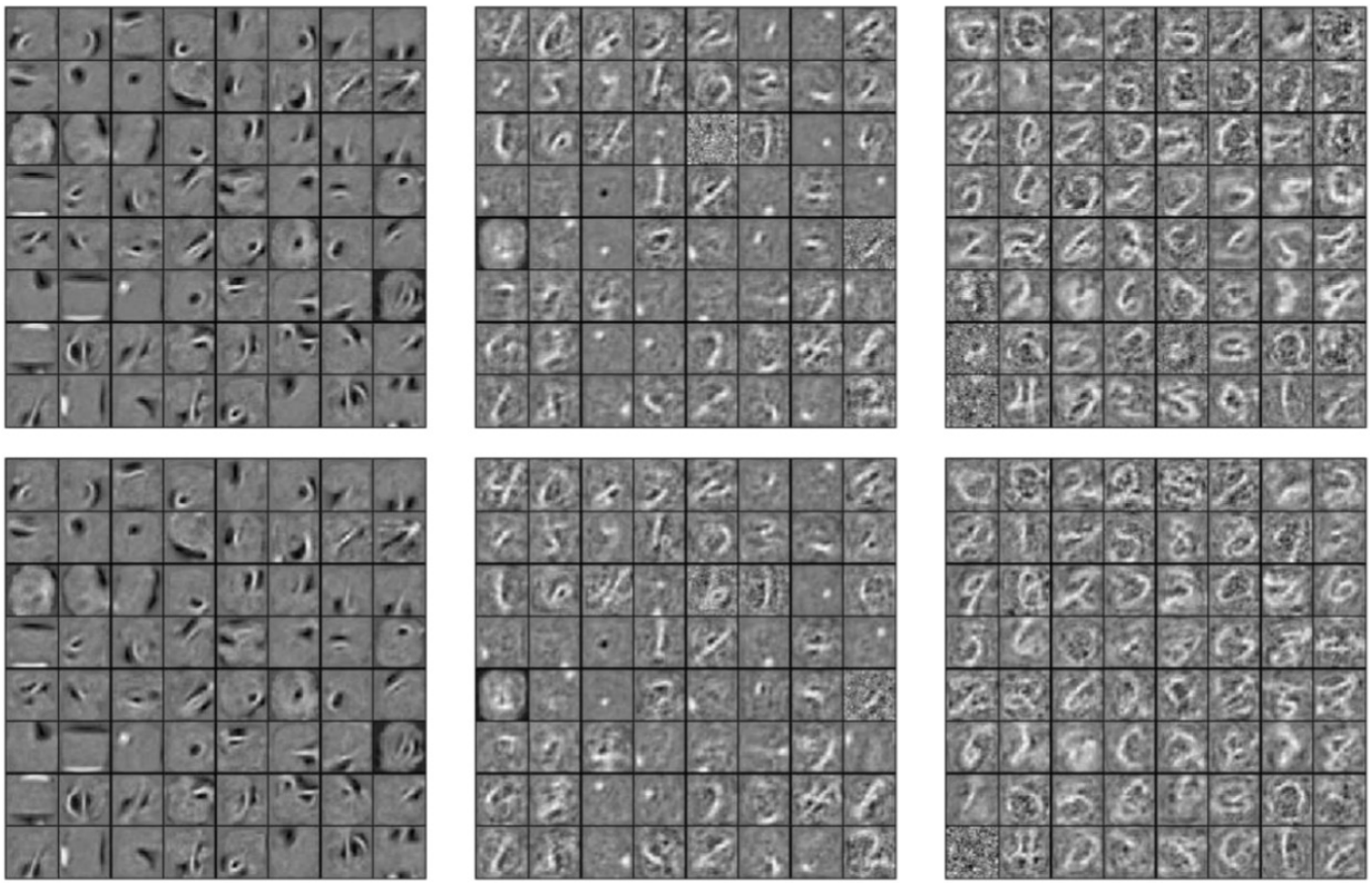
- DBN (Teh, Hinton 2006) wielowarstwowa sieć złożona z nałożonych na siebie RBM
- uczy się głębokiej, hierarchicznej reprezentacji danych, każda kolejna warstwa analizuje sygnał widoczny w warstwie ukrytej poprzedniego RBM
- uczenie zachłanne: kolejne warstwy dodawane i uczone pojedynczo
- DBN to pierwszy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



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

Filtry DBN na MNIST

pre-training

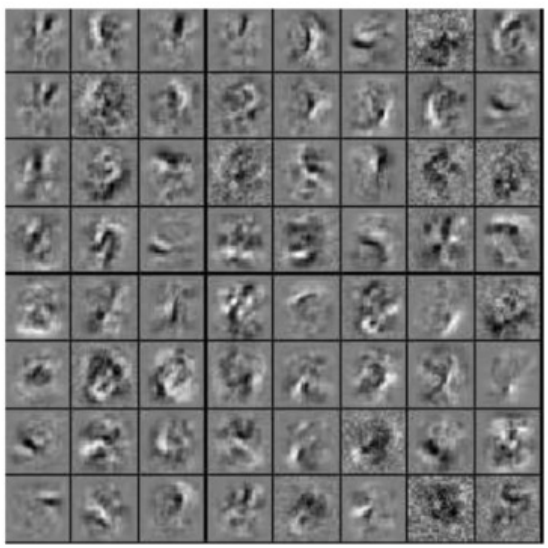
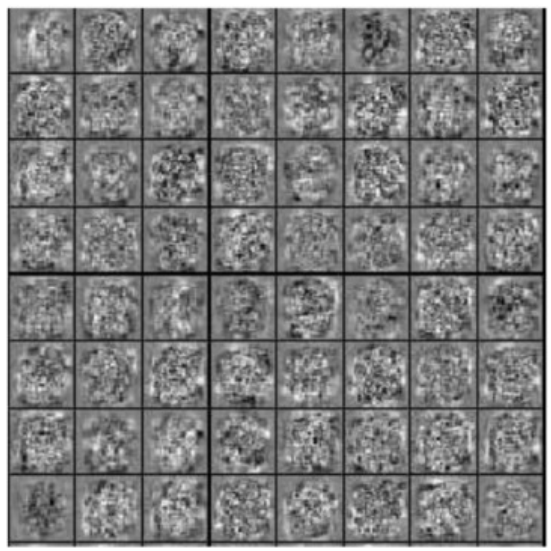
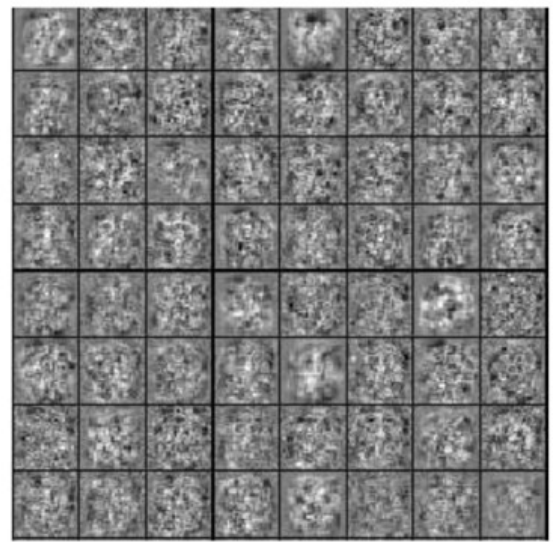


fine tune

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

Filtry DBN na MNIST

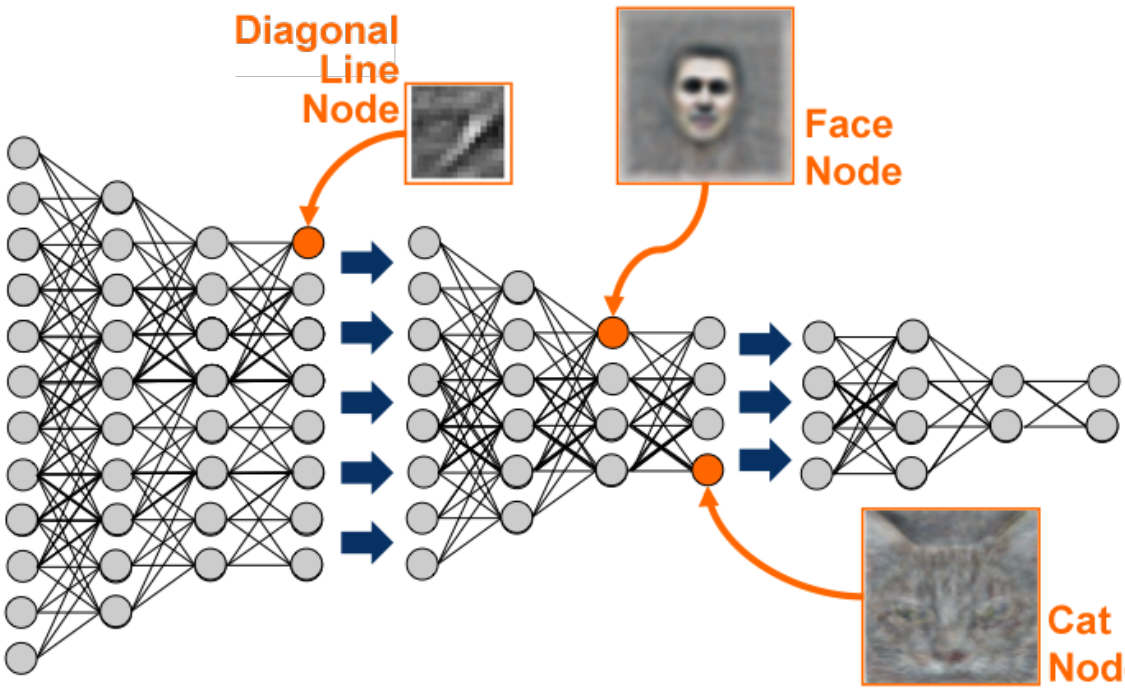
without pre-training

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

Czym jest kot?

Sieć DBN trenowana na obrazach 200x200 z 10M losowych nagrań wideo z YouTube. Dane bez etykiet - nie było żadnego celu treningu. Trening rozproszony na 1000 maszynach (16K CPU) przez 3 dni.

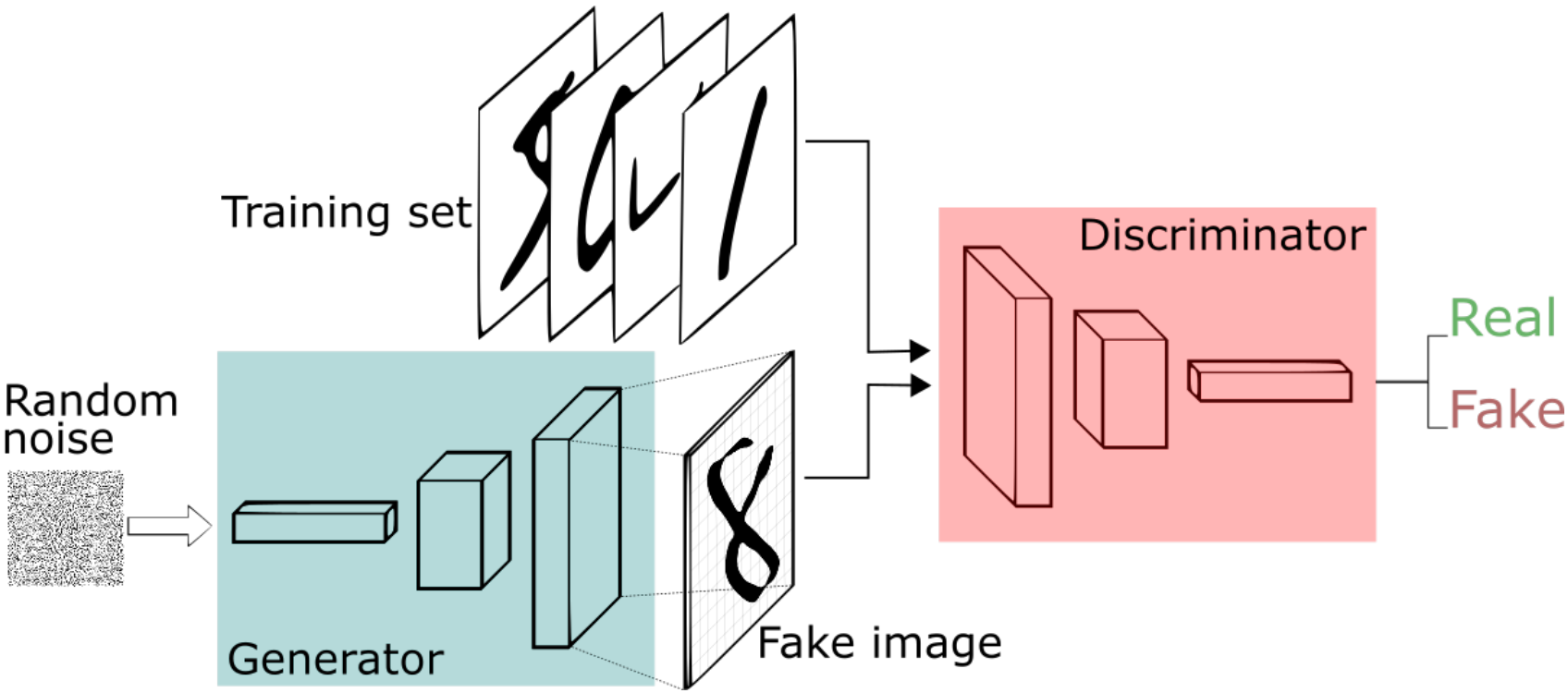


Pojedyncze neurony reagują na prezentację twarzy lub kota.

Le, 2013, Deep Belief Network

Generative Adversal Network (GAN)

Goodfelow 2014



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

Generative Adversal Network (GAN)

- GAN (Goodfelow 2014) - dwie sieci (generująca i oceniająca) rywalizujące ze sobą w grze o sumie zerowej
- model GAN uczy się generować dane o właściwościach zbliżonych do zbioru treningowego, np. 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, trening 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

Trening GAN

Funkcja kosztu dyskryminatora - binarna entropia krzyżowa (spodziewana odpowiedź 1 dyskryminatora dla obrazu prawdziwego)

$$L_D = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\mathbf{x})} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_z(\mathbf{z})} \log(1 - D(G(\boldsymbol{z})))$$

gdzie $p_{data}(\mathbf{x})$ rozkład danych prawdziwych, $p_z(\mathbf{z})$ rozkład danych generowanych

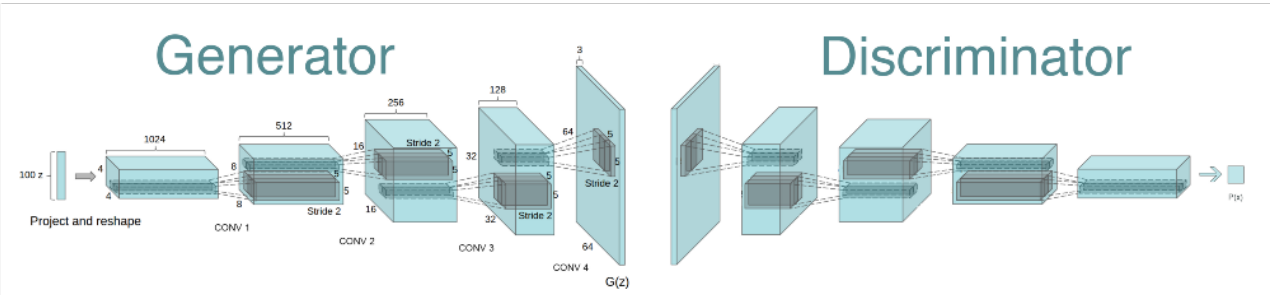
Funkcja kosztu generatora

$$L_G = -\frac{1}{2}\mathbb{E}_z \log D(G(\boldsymbol{z}))$$

maksymalizuje prawdopodobieństwo popełnienia błędu przez dyskryminator (spodziewana odpowiedź 1 dyskryminatora dla obrazu fałszywego)

DCGAN - Deep Conv GAN

Generowanie obrazów za pomocą splotowej sieci GAN



Radford, A., et al. (2015) Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

SRGAN - super resolution GAN

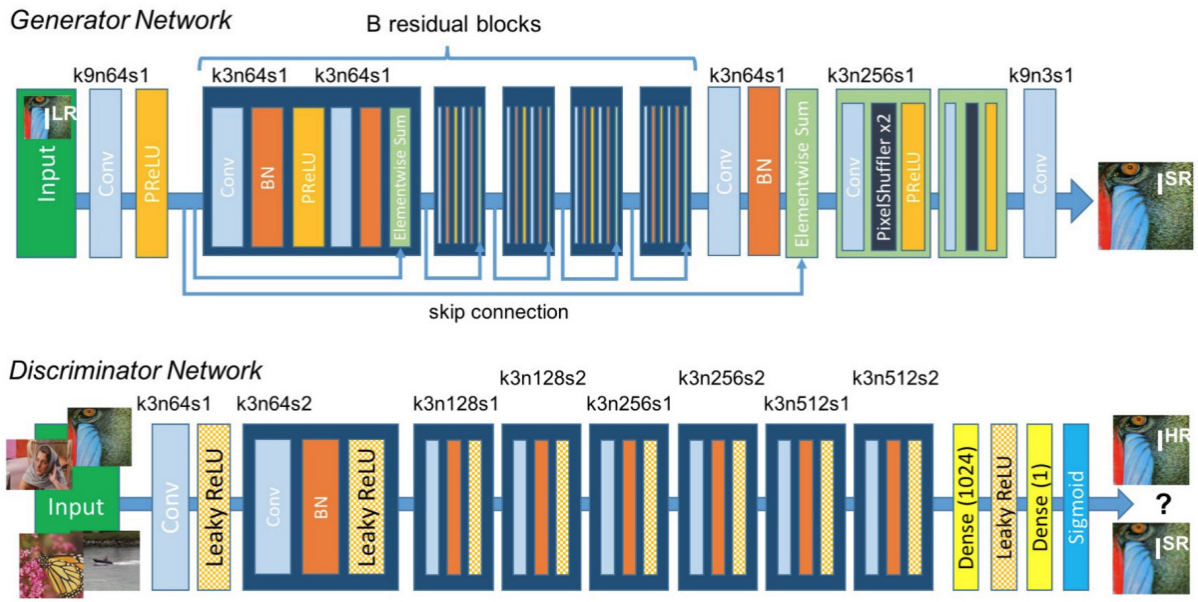
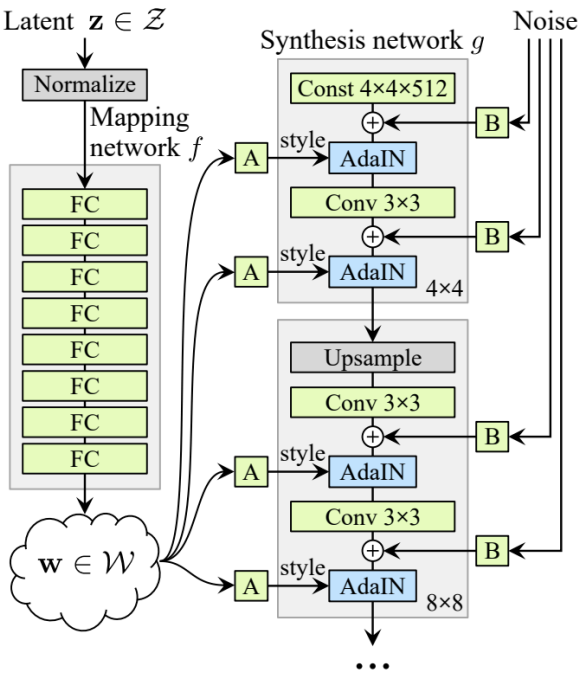


Figure 4: Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer.

Rys: ☞ Understanding and building Generative Adversarial Networks(GANs)

StyleGAN

Generator



Demo: ☞ Generator twarzy

T. Karras, S. Laine, T. Aila (2019) A Style-Based Generator Architecture for Generative Adversarial Networks

Inne zastosowania

- rekonstrukcja obrazów 3D ze zdjęć
- generowanie tekstu, synteza mowy, ...
- modyfikacje obrazów: zmiana twarzy, mimiki, fryzury, postarzanie osób na zdjęciach
- transfer stylu
- generowanie scen w grach wideo, zwiększanie rozdzielczości tekstur
- bezpieczeństwo: podnoszenie skuteczności algorytmów w wykrywaniu ataków cybenetycznych, wykrywanie fałszerstw, wykrywanie anomalii
- generowanie dane na potrzeby uczenia maszynowego
- generowanie obrazów z tekstu, generowanie nagrań wideo
- ☞ Curated list of awesome GAN applications and demo