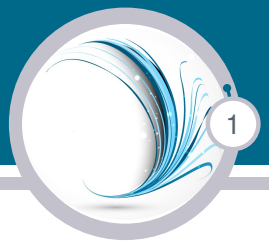
Katedra Sztucznej Inteligencji i Matematyki Stosowanej
WI ZUT Szczecin

Sztuczna inteligencja i maszynowe uczenie w systemach interaktywnych

Joanna Kolodziejczyk
jkolodziejczyk@zut.edu.pl

December 1, 2021

Plan wykładu



Głębokie sieci neuronowe
- Perceptrony wielowarstwowe

Deep Learning: Auto-Encoders

Głębokie sieci neuronowe

Definicja

Głębokie sieci neuronowe (Deep Neural Network) są nowym trendem w sztucznej inteligencji. Wydajność DNN wynika ze zdolności do ekstrakcji cech z dużej liczby surowych danych sensorycznych przy zastosowaniu statystyki, probabilistyki i algebry liniowej.

Przykładowe zastosowania

- rozpoznawanie mowy
- rozpoznawanie obrazu
- uczenie się strategii w skomplikowanych grach
- aplikacje w autonomicznych samochodach
- wykrywanie jednostek chorobowych z różnych form obrazowań

Deep Learning: Motywacja

Twierdzenie o aproksymacji uniwersalnej

Każda funkcja ciągła na dowolnie zwartej podprzestrzeni $\mathbb{R}^n$ może być dowolnie dobrze przybliżona za pomocą trójwarstwowego perceptronu.

To twierdzenie jest często przywoływane jako (rzekomo!) oznaczające, że:

- możemy ograniczyć się do wielowarstwowych perceptronów z tylko jedną lub dwoma warstwami ukrytymi ,
- nie ma rzeczywistej potrzeby tworzenia wielowarstwowych perceptronów z większą liczbą warstw ukrytych.

Jednakże:

- Twierdzenie nie mówi nic o liczbie ukrytych neuronów, które mogą być potrzebne do osiągnięcia pożądanej dokładności przybliżenia.
- W zależności od funkcji do przybliżenia, może być potrzebna bardzo duża liczba neuronów.
- Dopuszczenie większej liczby warstw ukrytych może pozwolić na osiągnięcie tej samej jakości przybliżenia przy znacznie mniejszej liczbie neuronów.

Deep Learning: przykład - funkcja parzystości

Funkcja parzystości

Bardzo prostym i powszechnie stosowanym przykładem jest n-bitowa funkcja parzystości: Wyjście wynosi 1 jeśli liczba wejść jest nieparzysta; w przeciwnym wypadku wyjście jest 0.

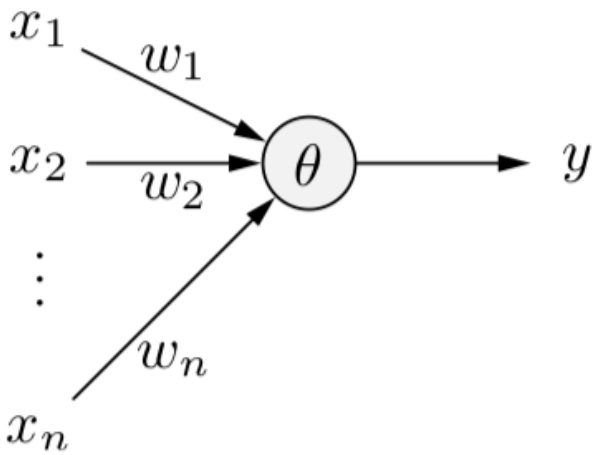
Definicja

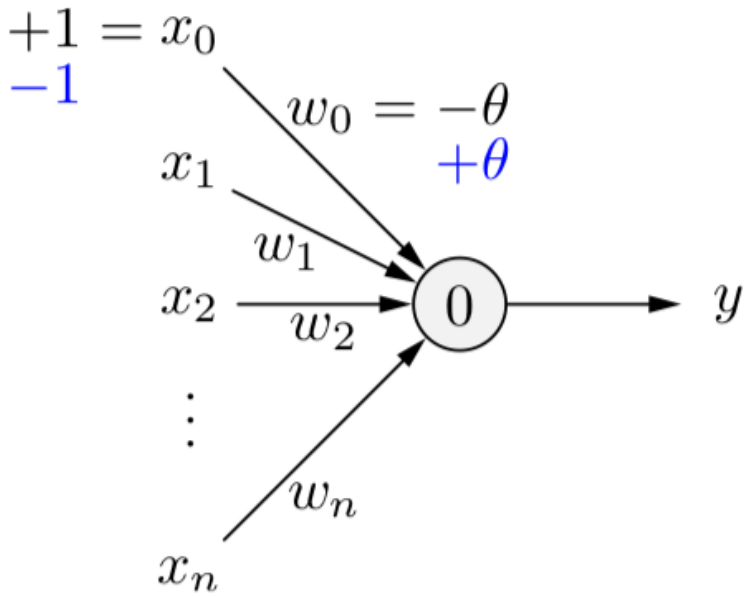
Jest to Boolowska funkcja $f : \{0,1\}^n \to \{0,1\}$, o własnościach: $f(x) = 1$ wtedy i tylko wtedy, gdy liczba jedynek w wektorze $x \in \{0,1\}^n$ jest nieparzysta. Innymi słowy, f jest zdefiniowane jako:

$$f(x) = x_1 \oplus x_2 \oplus \cdots \oplus x_n,$$

gdzie $\oplus$ oznacza ExOR.

Neuron i wartość progowa - przypomnienie



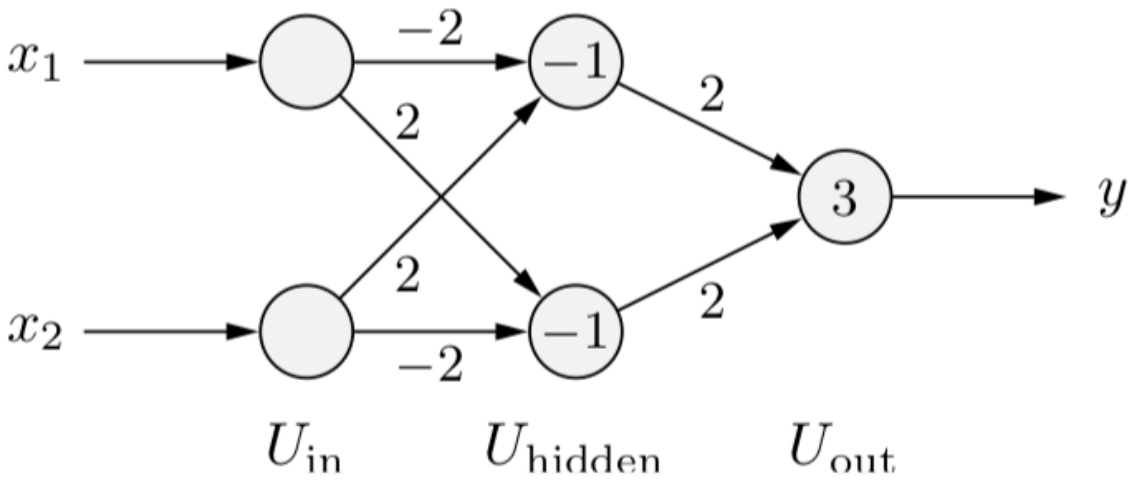


$$\sum_{i=1}^{n} w_i x_i \geq \theta$$

$$\sum_{i=1}^{n} w_i x_i - \theta \geq 0$$

Równoważność - przypomnienie

Przykład sieci rozwiązującej problem równoważności ($x_1 \leftrightarrow x_2$)



x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1

$$\mathbf{W}_1 = \begin{pmatrix} -2 & 2 \\ 2 & -2 \end{pmatrix}$$

i $\quad \mathbf{W}_2 = \begin{pmatrix} 2 & 2 \end{pmatrix}$

Deep Learning: Przykład - funkcja parzystości

- F. parzystości może być łatwo reprezentowana przez wielowarstwowy perceptron z tylko jedną warstwą ukrytą.
- Dopuszczenie większej liczby warstw ukrytych może pozwolić na osiągnięcie tej samej jakości przybliżenia przy znacznie mniejszej liczbie neuronów.
- Jednak rozwiązanie ma 2^{n-1} ukrytych neuronów bo: dysjunkcyjna postać normalna jest sumą 2^{n-1} koniunkcji, które reprezentują 2^{n-1} kombinacji z parzystym zestawem bitów.
- Liczba ukrytych neuronów rośnie wykładniczo wraz z liczbą wejść.
- Jeśli jednak dopuszczalna jest większa liczba ukrytych warstw, możliwy jest liniowy wzrost:
  - Zacznij od równoważności ($x_1 \leftrightarrow x_2$) dwóch wejść
  - Kontynuuj łańcuch ExOR, z których każdy jest łączony z kolejną warstwą.
  - Taka sieć potrzebuje w sumie $n + 3(n-1) = 4n - 3$ neuronów (gdzie n -liczba neuronów wejściowych, $3(n-1) - 1$ neuronów ukrytych, 1 neuron w warstwie wyjściowej.

Reprezentowanie Funkcji Logicznych MLP

Algorytm: Niech $y = f(x_1, \ldots, x_n)$ będzie funkcją boole'owską n zmiennych.

1. Przedstawić daną funkcję $f(x_1, \ldots, x_n)$ w dysjunkcyjnej postaci normalnej. To znaczy, wszystkie C_j są iloczynami n literałów, to jest $C_j = l_{j1} \wedge \ldots \wedge l_{jn}$ z $l_{ji} = x_i$ (literałami pozytywnymi) lub $l_{ji} = \neg x_i$ (literałami negatywnymi).
2. Stworzyć neuron dla każdej koniunkcji C_j dysjunkcyjnej postaci normalnej (mający n wejść - jedno wejście dla każdej zmiennej), gdzie

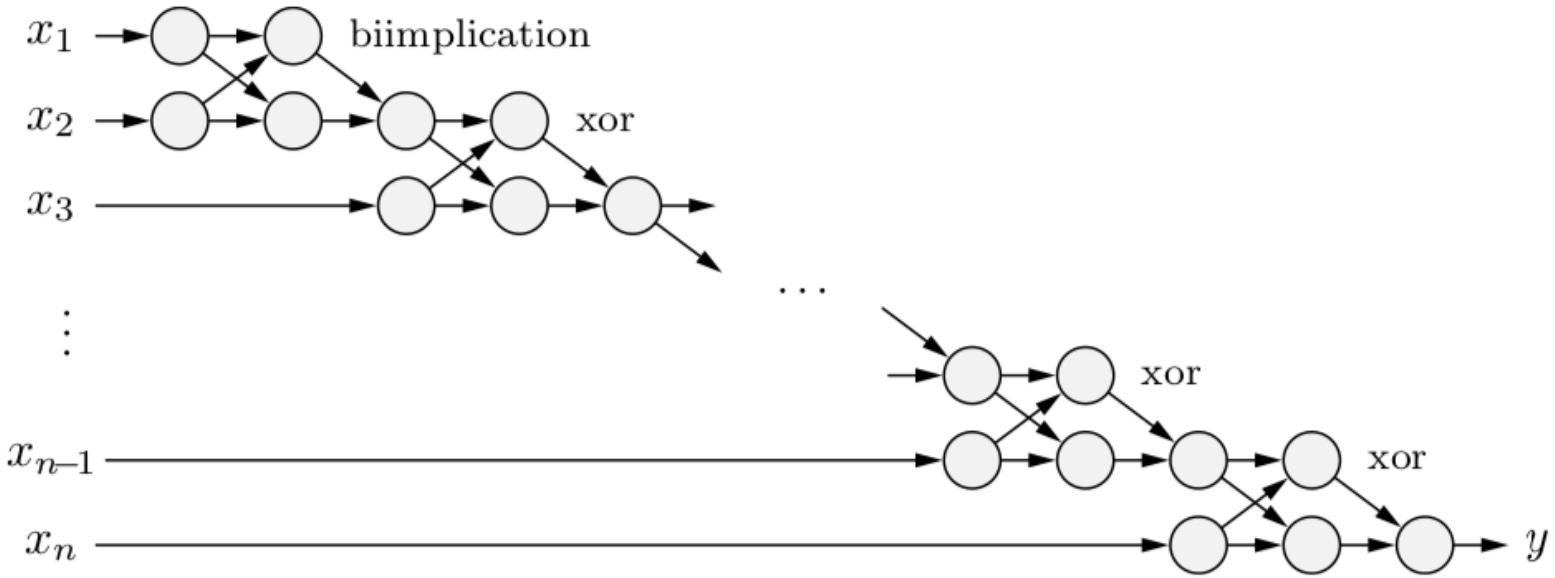
$$w_{ji} = \begin{cases} 2, \text{ if } l_{ji} = x_i, \\ -2, \text{ if } l_{ji} = \neg x_i \end{cases} \quad \text{and} \quad \theta_j = n - 1 + \frac{1}{2}\sum_{i=1}^{n} w_{ji}$$

3. Utworzyć neuron wyjściowy (posiadający m wejść - jedno wejście dla każdego neuronu, który został utworzony w kroku 2), gdzie

$$w_{(n+1)k} = 2, \quad k = 1, \ldots, m, \quad \text{and} \quad \theta_{n+1} = 1$$

Aby zapewnić wartość progową jako całkowitą przyjęto wagi +/- 2 a nie +/-1

Głębokie uczenie: n-bitowa funkcja



- Implementacja funkcji parzystości n-bitowego łańcuchem równoważności i $n-2$ podsieci ExOR.
- Należy pamiętać, że konstrukcja nie jest ściśle warstwowa, ale może być uzupełniona. Przy takiej implementacji, liczba neuronów wzrasta do $n(n+1)-1$.

Głębokie uczenie: motywacja

- W przypadku funkcji separowalnych liniowo
  - Tylko kilka *n*-wymiarowych funkcji Boolowskich jest liniowo rozdzielnych (patrz następny slajd).
  - Tylko kilka *n*-wymiarowych funkcji Boolowskich wymaga kilku ukrytych neuronów w pojedynczej warstwie.
- Załóżmy, że *n*-wymiarowa funkcja Boolowska ma k_0 kombinacji wejść z wyjściem 0 oraz ma k_1 kombinacji wejść z wyjściem 1 czyli: $k_0 + k_1 = 2^n$
- Wymagane jest co najmniej $2^{\min\{k_0,k_1\}}$ neuronów ukrytych dla dysjunkcyjnej formy normalnej $k_1 \leq k_0$ i koniunkcyjnej formy normalnej $k_1 > k_0$.
- Ponieważ istnieje $\begin{pmatrix} 2^n \\ k_0 \end{pmatrix}$ możliwych funkcji z k_0 kombinacjami wejść mapowanymi na 0 i k_1 zmapowanych na 1, wiele funkcji wymaga znacznej liczby neuronów ukrytych .

Przypomnienie: Ograniczenia przetwornika logicznego z progiem

Całkowita liczba i liczba funkcji boole'owskich separowalnych liniowo

wejście	funkcje Boolowskie	funkcje separowalne liniowo
1	4	4
2	16	14
3	256	104
4	65,536	1,882
5	4,294,967,296	94,572
6	18,446,744,073,709,551,616	15,028,134
n	$2^{(2^n)}$	nie jest znana ogólna zależność

- Dla wielu wejść przetwornik logiczny ma bardzo małą wydajność.
- Sieci przetworników są potrzebne do przezwyciężenia ograniczenia wydajnościowego.

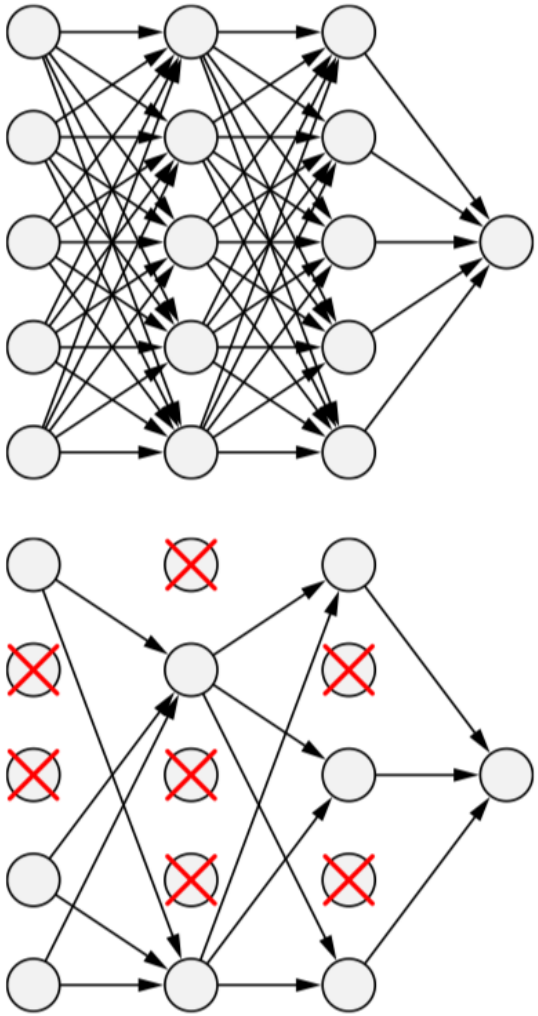
Głębokie uczenie: motywacja

- W praktyce problem wydajności nie jest tak dotkliwy gdyż zbiory danych treningowych są z reguły ograniczone pod względem wielkości.
- Kompletne dane treningowe (wszystkie kombinace) dla *n*-wymiarowej funkcji binarnej mają 2^n próbek. Zbiory danych dla problemów praktycznych zwykle zawierają znacznie mniej przypadków. Prowadzi to do wielu konfiguracji wejściowych, dla których nie jest podane żadne pożądane wyjście; swoboda w przypisywaniu do nich wyjść pozwala na prostszą reprezentację.
- Niemniej jednak, stosowanie więcej niż jednej warstwy ukrytej daje, w wielu przypadkach, szansę zmniejszenia liczby potrzebnych neuronów.
- Na tych własnościach koncentrują się głębokie uczenie. (Głębokość oznacza długość najdłuższej ścieżki w grafie jakim jest sieć neuronowa).
- W przypadku perceptronów wielowarstwowych (najdłuższa ścieżka: liczba warstw ukrytych plus jeden), głębokie modele rozpoczynają się od więcej niż jednej ukrytej warstwy.
- W przypadku 10 lub więcej ukrytych warstw, czasami mówi się o bardzo głębokim uczeniu się.

Deep Learning: Główne problemy

- Wielowarstwowe głębokie perceptrony cierpią na dwa główne problemy: przetrenowanie i zanikający gradient.
- Przetrenowanie wynika przede wszystkim ze zwiększonej liczby możliwych do dostosowania parametrów.
- Zmniejszające się wagi zapobiegają powstawaniu dużych wag i tym samym zbyt precyzyjnego dopasowania.
- Rzadka (sparse) sieć, to taka, gdzie nie ma wszystkich połączeń. Ich stosowanie zapobiega przetrenowaniu:
  - liczba neuronów w warstwach ukrytych jest ograniczona
  - (średnio) tylko kilka neuronów w ukrytych warstwach powinno być aktywna.
    Można to osiągnąć poprzez dodanie do funkcji błędu składowej zwanej regularyzacją (porównuje zaobserwowaną liczbę aktywnych neuronów z pożądaną liczbą i popycha dopasowanie w kierunku, który stara się dopasować do tej liczby).
- Ponadto, można zastosować metodę uczenia zwaną dropout: niektóre jednostki podczas treningu są losowo pomijane na wejściu/warstwie ukrytej.

Deep Learning: Dropout



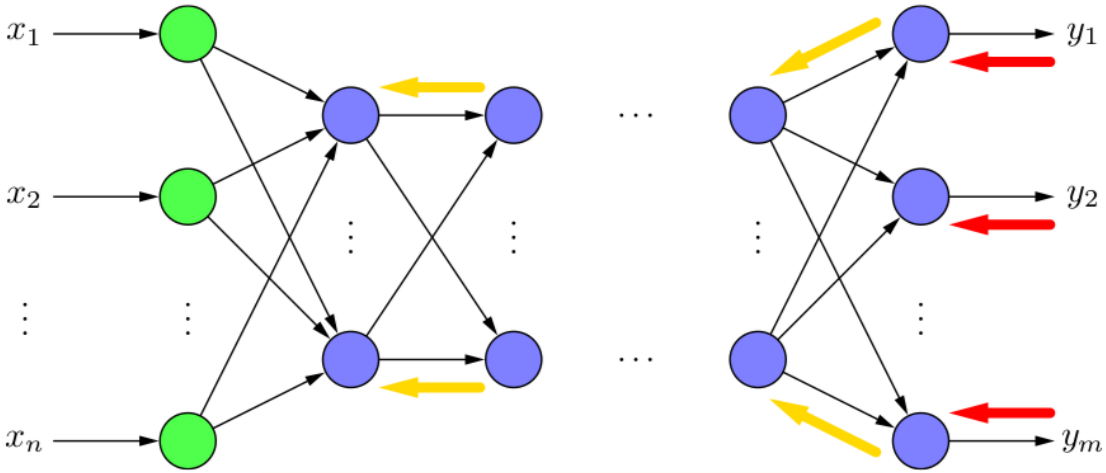
- Pożądana cecha: Odporność na uszkodzenia neuronów.
- Podczas trenowania:
  - Używaj tylko p% neuronów (np. $p = 50\%$ połowa neuronów).
  - Wybieraj losowo neurony do odrzucenia.
- Podczas wykonania:
  - Użyj wszystkich neuronów.
  - Pomnóż wszystkie wagi przez p%.
- Wynik działania:
  - Bardziej efektywna reprezentacja.
  - Lepsza generalizacja.

Powtórka: wsteczna propagacja błędów

$\forall u \in U_{\text{in}}$:
$\text{out}_u^{(l)} = \text{ext}_u^{(l)}$

forward propagation:

$\forall u \in U_{\text{hidden}} \cup U_{\text{out}}$:
$\text{out}_u^{(l)} = \left(1 + \exp\left(-\sum_{p \in \text{pred}(u)} w_{up}\, \text{out}_p^{(l)}\right)\right)^{-1}$



- logistic activation function
- implicit bias value

error factor:

backward propagation:

$\forall u \in U_{\text{hidden}}$:
$\delta_u^{(l)} = \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su}\right) \lambda_u^{(l)}$

$\forall u \in U_{\text{out}}$:
$\delta_u^{(l)} = \left(o_u^{(l)} - \text{out}_u^{(l)}\right) \lambda_u^{(l)}$

activation derivative:

$\lambda_u^{(l)} = \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)}\right)$

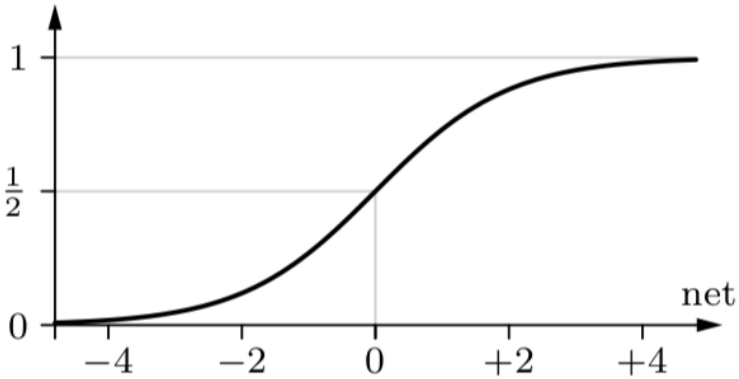
weight change:

$\Delta w_{up}^{(l)} = \eta\, \delta_u^{(l)}\, \text{out}_p^{(l)}$

Deep Learning: Znikający gradient

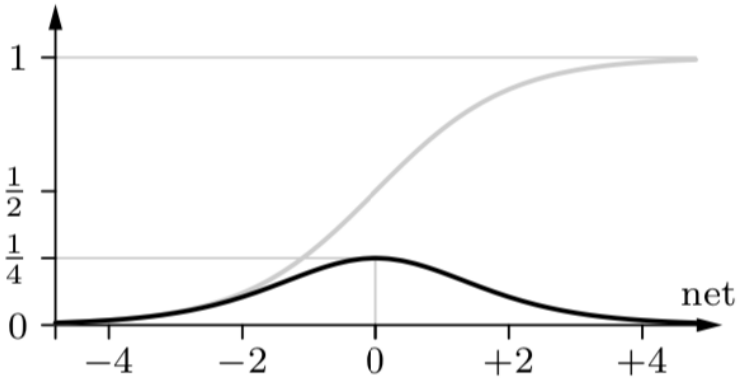
logistic activation function:

$$f_{\text{act}}(\text{net}_u^{(l)}) = \frac{1}{1 + e^{-\text{net}_u^{(l)}}}$$



derivative of logistic function:

$$f'_{\text{act}}(\text{net}_u^{(l)}) = f_{\text{act}}(\text{net}_u^{(l)}) \cdot (1 - f_{\text{act}}(\text{net}_u^{(l)}))$$



- W przypadku zastosowania funkcji aktywacji logistycznej, zmiany wag są proporcjonalne do $\lambda_u^{(l)} = \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)}\right)$
- Współczynnik jest propagowany wstecz, ale nie może być większy niż $\frac{1}{4}$ (rys. po prawej).
- Gradient ma tendencję do zaniku, jeśli przejdzie przez wiele warstw. Nauka w początkowych warstwach ukrytych może być bardzo wolna.

Deep Learning: Znikający gradient

- Załóżmy, że niewielki gradient może być zwalczany przez duże wagi.

$$\delta_u^{(l)} = \left(\sum_{s \in \mathrm{succ}(u)} \delta_s^{(l)} w_{su} \right) \lambda_u^{(l)}$$

- Jednak duża waga, ponieważ również wchodzi w skład funkcji aktywacji, prowadzi do nasycenia funkcja aktywacji.
- Tak więc wartość (bezwzględna) pochodnej jest mniejsza, im większa wartość (bezwzględna) wag.
- Co więcej, wagi połączeń są zazwyczaj inicjowane wartością losową z przedziału od -1 do +1.
  Szczególnie wpływa to na początkowy etap trenowania: zarówno gradient, jak i waga są iloczynem wartości mniejszych niż 1.
- Teoretycznie, mogą wystąpić wyjątki. Jednak w praktyce są one rzadkie i zwykle obserwuje się zanikanie gradientu.

Deep Learning: Znikający gradient

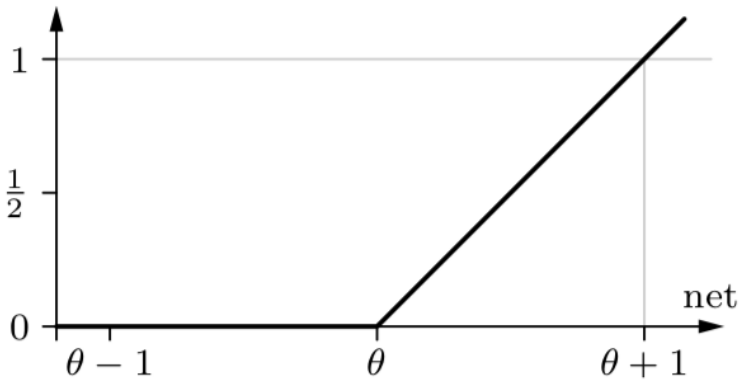
Alternatywny sposób na zrozumienie efektu zanikania gradientu:

- Logistyczna funkcja aktywacji jest funkcją malejącą:
  dla dwóch dowolnych argumentów x i y, $x \neq y$, spełniona jest zależność: $|f_{\text{act}}(x) - f_{\text{act}}(y)| < |x - y|$
  (Wynika z faktu, że jej pochodna jest zawsze < 1; w rzeczywistości $\leq \frac{1}{4}$.)
- Jeśli kilka funkcji logistycznych jest połączonych łańcuchem, spadki wartości łączą się i dają jeszcze silniejszy spadek zakresu wejściowego.
- W konsekwencji, dość duża zmiana wartości wejściowych spowoduje tylko niewielką zmianę wartości wyjściowych, a im bardziej, tym więcej funkcji logistycznych jest połączonych w łańcuch zależności.
- Dlatego też funkcja mapująca wejścia wielowarstwowego perceptronu na jego wyjścia zwykle staje się tym bardziej płaska, im więcej warstw ma wielowarstwowy perceptron.
- W konsekwencji gradient w pierwszej warstwie ukrytej (gdzie przetwarzane są wejścia) staje się coraz mniejszy.

Deep Learning: Różne funkcje aktywacji

rectified maximum/ramp function:

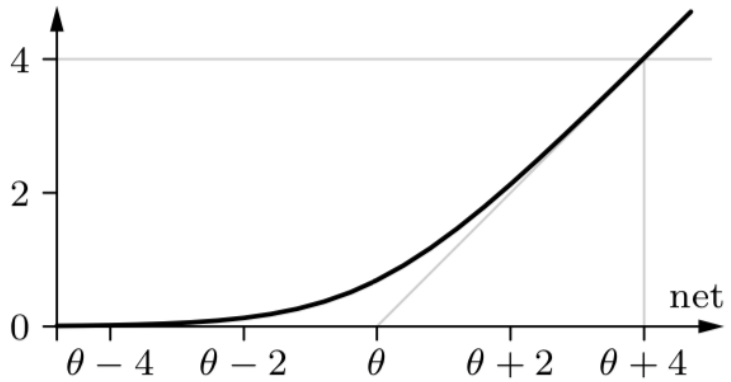
$f_{\text{act}}(\text{net}, \theta) = \max\{0, \text{net} - \theta\}$



softplus function:

Note the scale!

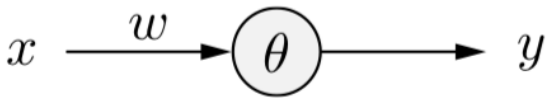
$f_{\text{act}}(\text{net}, \theta) = \ln(1 + e^{\text{net} - \theta})$



- Problem zanikającego gradientu może być rozwiązany za pomocą innych funkcji aktywacyjnych.
- Taką funkcją jest np. rectified linear units (ReLUs).
- Rectified maximum/ramp function
  Zalety: proste obliczenie, prosta pochodna, zera upraszczają uczenie.
  Wady: brak uczenia się $\leq \theta$, raczej nieelegancka, nieróżniczkowalna

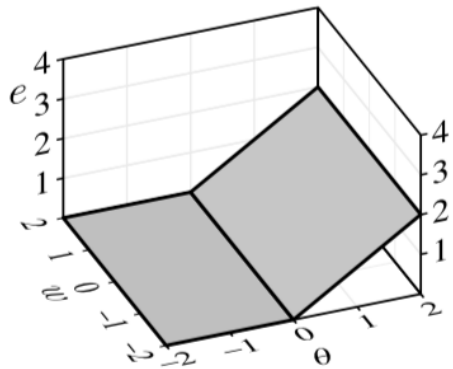
Neuron negator oparty na ReLU

Jednowejściowy przetwornik logiczny reagujący jak funkcji $\neg$.

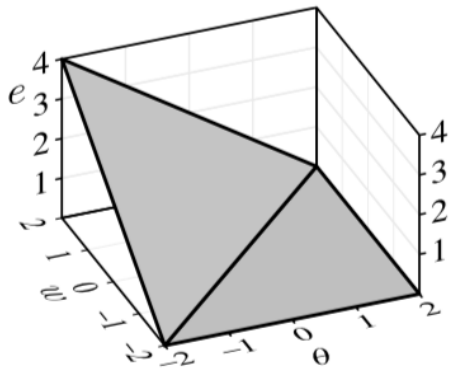


x	y
0	1
1	0

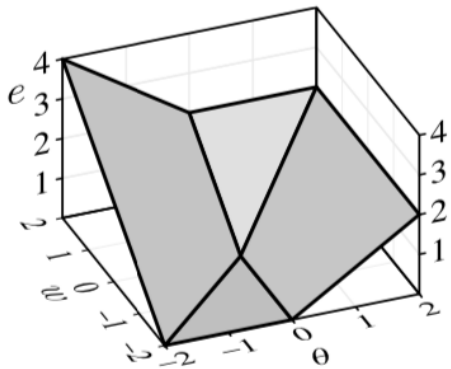
Zmodyfikowany błąd jako funkcja wagi i progu:



error for $x = 0$



error for $x = 1$



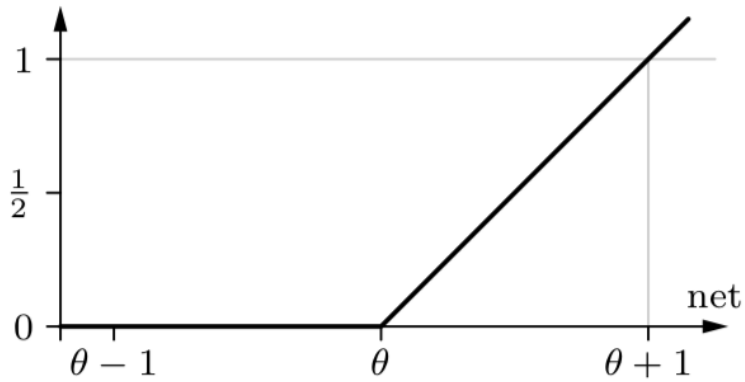
sum of errors

Funkcja Rectified maximum/ramp powoduje powstanie powyższej płaszczyzny błędu.

Deep Learning: Różne funkcje aktywacji

rectified maximum/ramp function:

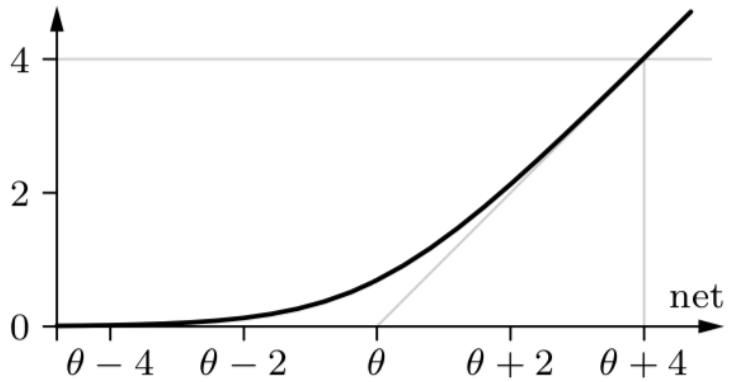
$f_{\text{act}}(\text{net}, \theta) = \max\{0, \text{net} - \theta\}$



softplus function:

Note the scale!

$f_{\text{act}}(\text{net}, \theta) = \ln(1 + e^{\text{net} - \theta})$

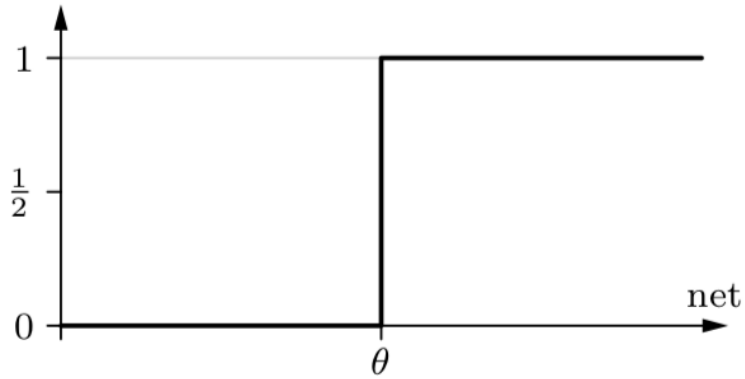


- Funkcja softplus: różniczkowalna i bardziej złożona
- inne alternatywy:
  - Leaky ReLU:
    $f(x) = \begin{cases} x & \text{if } x > 0 \\ \nu x & \text{otherwise} \end{cases}$ (adds learning $\leq \theta; \nu \approx 0.01$)
  - Noisy ReLU:
    $f(x) = \max\{0, x + \mathcal{N}(0, \sigma(x))\}$ (adds Gaussian noise)

Deep Learning: Różne funkcje aktywacji

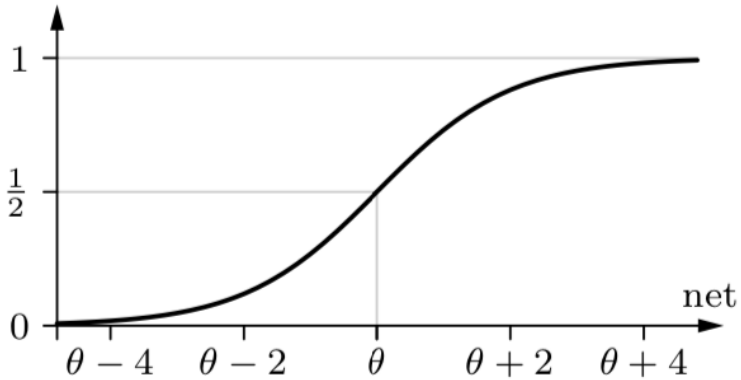
derivative of ramp function:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{if net} \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$



derivative of softplus function:

$$f_{\text{act}}(\text{net}, \theta) = \frac{1}{1 + e^{-(\text{net} - \theta)}}$$

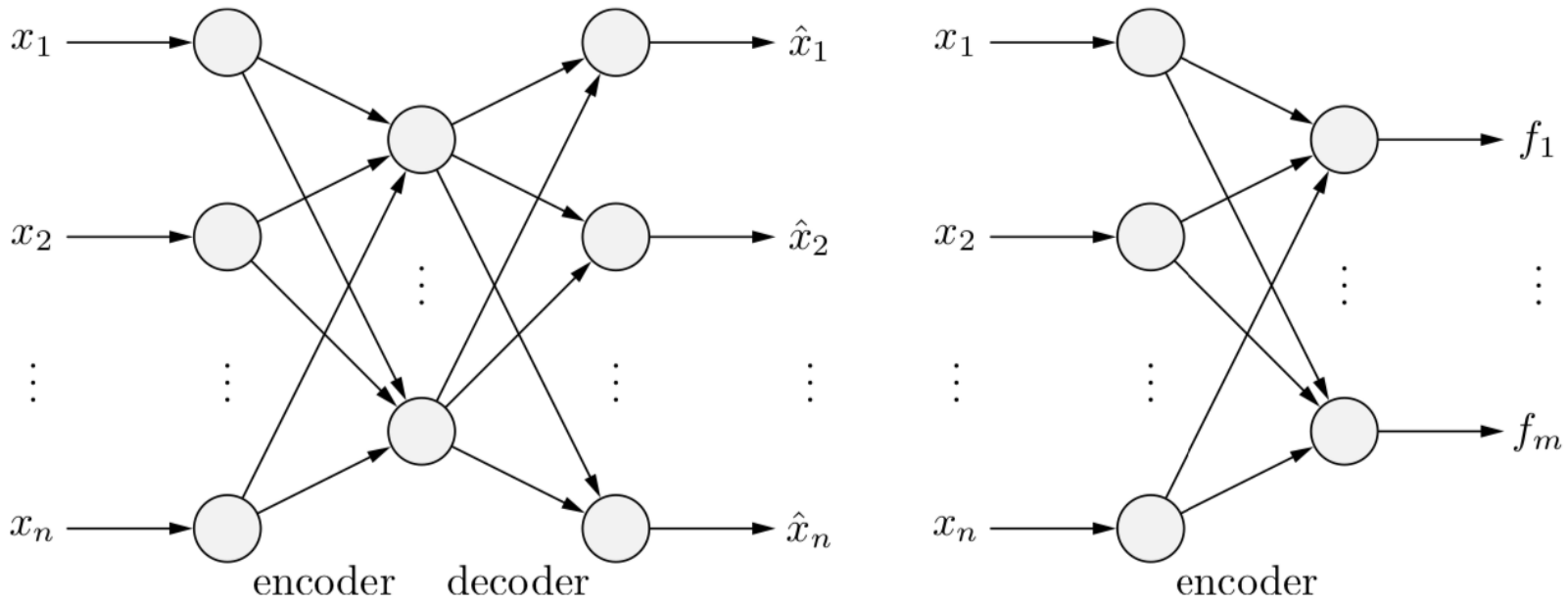


- Pochodna ReLU jest funkcją skoku jednostkowego.
- Pochodną funkcji softplus jest funkcja logistyczna.
- Czynnik wynikający z pochodnych tych funkcji aktywacji może być znacznie większy. Większy gradient w pierwszych warstwach sieci –> (znacznie) szybszy trening.
- Pomaga również szybszy sprzęt –> implementacje na układach GPU itp.

Deep Learning: Auto-Encoders

- Przy braku pożądanych wartości dla neuronów pierwszej warstwy (warstw) sieci z przetwornikami logicznymi (neuronami) nie mogą być trenowane
  - problem ten można zazwyczaj rozwiązać stosując kilka różnych funkcji wyliczanych przez neurony pierwszej warstwy (rozwiązanie niejednorodne).
- Chociaż można trenować MLP, nawet z wieloma warstwami ukrytymi, to ten sam problem nadal wpływa na skuteczność i efektywność uczenia.
- Alternatywa: Zbuduj sieć warstwa po warstwie, w każdym kroku trenuj tylko nowo dodaną warstwę.
  Popularne: Zbuduj sieć jako stos autenkoderów.
- Auto-enkoder to trójwarstwowy perceptron, który mapuje dane wejściowe w aproksymację tych dane.
  - Warstwa ukryta tworzy enkoder w formie pewnej reprezentacji wewnętrznej.
  - Warstwa wyjściowa tworzy koder, który (w przybliżeniu) odtwarza dane wejściowe.

Deep Learning: Auto-Encoders



- Auto-enkoder/dekoder (po lewej), z którego później używana jest tylko część enkodera (po prawej).
- x_i jest zadanym wejściem, a $\hat{x}_i$ jest dotworzonym wejściem a f_i to tworzone cechy. Błąd to $e = \sum_{i=1}^{n} (\hat{x}_i - x_i)^2$.
- Uczenie prowadzone jest z wykorzystaniem wstecznej propagacji błędów.

Deep Learning: Auto-Encoders

- Uzasadnienie uczenia autoenkodera: oczekuje się, że warstwa ukryta ma wytwarzać cechy/atrybuty.
- Oczekiwanie: Funkcje przechwytują informacje zawarte na wejściu w postaci skompresowanej (enkoder), dzięki czemu wejście może być z niego poprawie odtworzone (dekoder).
- Uwaga: zakłada to domyślnie, że funkcje, które są dobrze dopasowane do reprezentacji wejść w sposób skompresowany, są również przydatne do przewidywania niektórych pożądanych wyjść. Doświadczenie pokazuje, że to założenie jest często uzasadnione.
- Główne problemy: Ile jednostek należy wybrać dla warstwy ukrytej? Jak należy traktować tę warstwę podczas uczenia?
- Jeśli jest tyle (lub nawet więcej) jednostek ukrytych, ile jest wejść, prawdopodobnie sygnał przejdzie jedynie przez wejścia do warstwy wyjściowej.
- Oczywistym zdaje się rozwiązanie, by stosować nieliczne autoenkodery: Powinno być (znacznie) mniej ukrytych neuronów niż wejść.

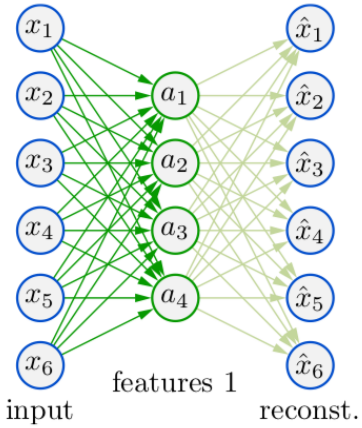
Deep Learning: Sparse and Denoising Auto-Encoders

- Mała liczba ukrytych neuronów zmusza autoenkoder do nauczenia się odpowiednich cech (ponieważ nie jest możliwe proste przejście bez przekształcenia przez warstwę).
- Ile ukrytych neuronów?
  Walidacja krzyżowa nie musi być dobrym podejściem!
- Alternatywa dla kilku ukrytych neuronów: sparse activation scheme
  Liczba aktywnych neuronów w warstwie ukrytej jest ograniczona do małej liczby.
  Może być wymuszona przez dodanie członu z regularyzacją do funkcji błędu, który karze większą liczbę aktywnych neuronów ukrytych lub przez dezaktywację wszystkich oprócz kilku neuronów o najwyższym poziomie aktywacji.
- Trzecie podejście polega na dodaniu szumu (czyli przypadkowych zmian) do wejścia (ale nie do kopii wejścia używanego do oceny błędu rekonstrukcji): denoising auto-enkoderów.
  Oczekuje się, że trenowany autokoder będzie mapował wejście z szumem na (kopię) wejście bez szumu, zapobiegając w ten sposób przechodzeniu przez nie bez modyfikacji).

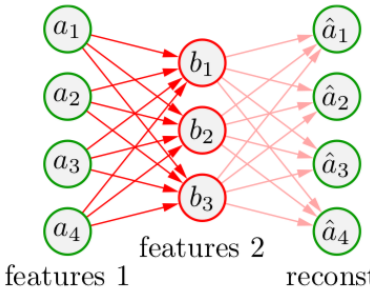
Deep Learning: Stacked Auto-Encoders

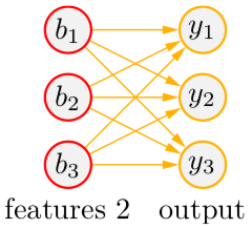
- Popularnym sposobem inicjowania/przygotowania do treningu Stacked Auto-Encoders to greedy layer-wise training approach.
- W pierwszym kroku, (sparse) auto-encoder jest trenowany z surowych danych wejściowych. Ukryta warstwa wykrywa kluczowe cechy przydatne do rekonstrukcji.
- Podstawowy zestaw danych o cechach jest uzyskiwany poprzez propagację surowych cech wejściowych aż do warstwy ukrytej i zapisywanie aktywacji neuronów ukrytych.
- W kroku drugim, (sparse) auto-encoder jest uczony uzyskanym zestawem cech. Warstwa ukryta konstruuje wtórne cechy przydatne do rekonstrukcji.
- Zbiór danych o cechach wtórnych uzyskuje się poprzez propagację cech podstawowych aż do warstwy ukrytej i zapisywanie aktywacji neuronów ukrytych.
- Proces ten jest powtarzany tyle razy, ile ma powstać ukrytych warstw.
- Na koniec części enkoderów są układane w stos, a powstała sieć jest dostrajana za pomocą wstecznej błędów.

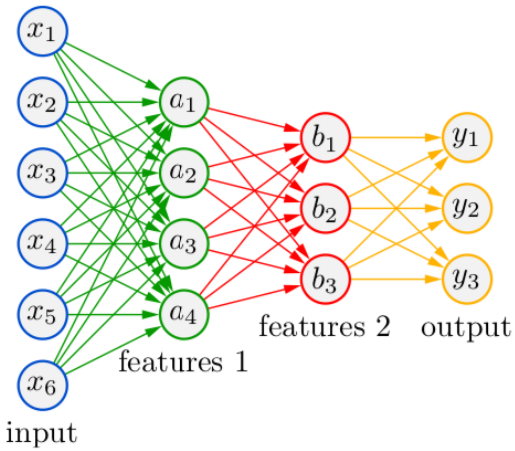
Deep Learning: Auto-Encoders



Layer-wise training of auto-encoders:







1. Trenowanie autoencodera na podstawie danych surowych. Uzyskanie podstawowego zestawu cech.
2. Trenowanie autoencodera dla cech pierwotnych. Uzyskanie wtórnego zestawu cech.
3. Klasyfikator, regresor dla wyjść jest trenowany z warstwy cech wtórnych.
4. Złożenie warstw sieci wynikowej.

Do czego są dobre autokody?

1. Są one rzadko używane w zastosowaniach praktycznych.
2. Jednym z praktycznych zastosowań jest odszumianie oraz
3. redukcja wymiarowości dla wizualizacji danych. Dzięki odpowiednim ograniczeniom wymiarowości i skąpych rozmiarów, autokodery mogą nauczyć się projekcji danych, które są bardziej interesujące niż PCA lub inne podstawowe techniki.

Do przeczytania:
`https://blog.keras.io/building-autoencoders-in-keras.html`

Dziękuję za uwagę
Czas na pytania ????