Jednokierunkowe sieci wielowarstwowe

Multilayer Perceptron (MLP)

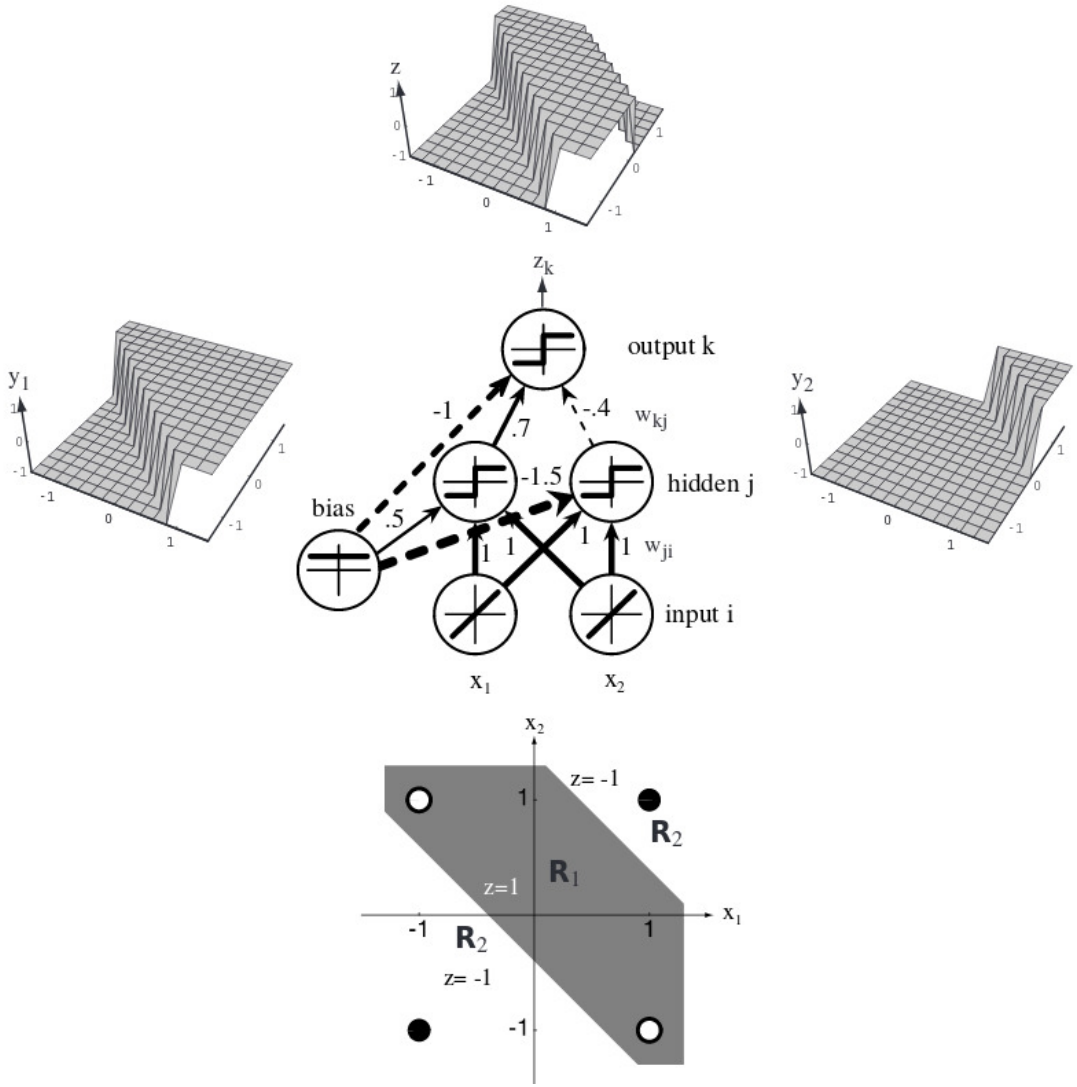
- Perceptrony jednowarstwowe potrafią rozwiązywać problemy separowalne liniowo
- Reguła delta dla sieci jednowarstwowej:

$$\Delta w_{ij} = \eta \left(y_j - f_j(\mathbf{x}) \right) f_j'(\mathbf{x}) x_i$$

  gdzie w_{ij} waga pomiędzy i-tym wejściem a j-tym wyjściem
- Problemy bardziej złożone (np. XOR) wymagają dodania kolejnej warstwy (nieliniowej) przetwarzania
- Każda kolejna warstwa tworzy nowy obraz danych, który stanowi wejście dla kolejnych warstw

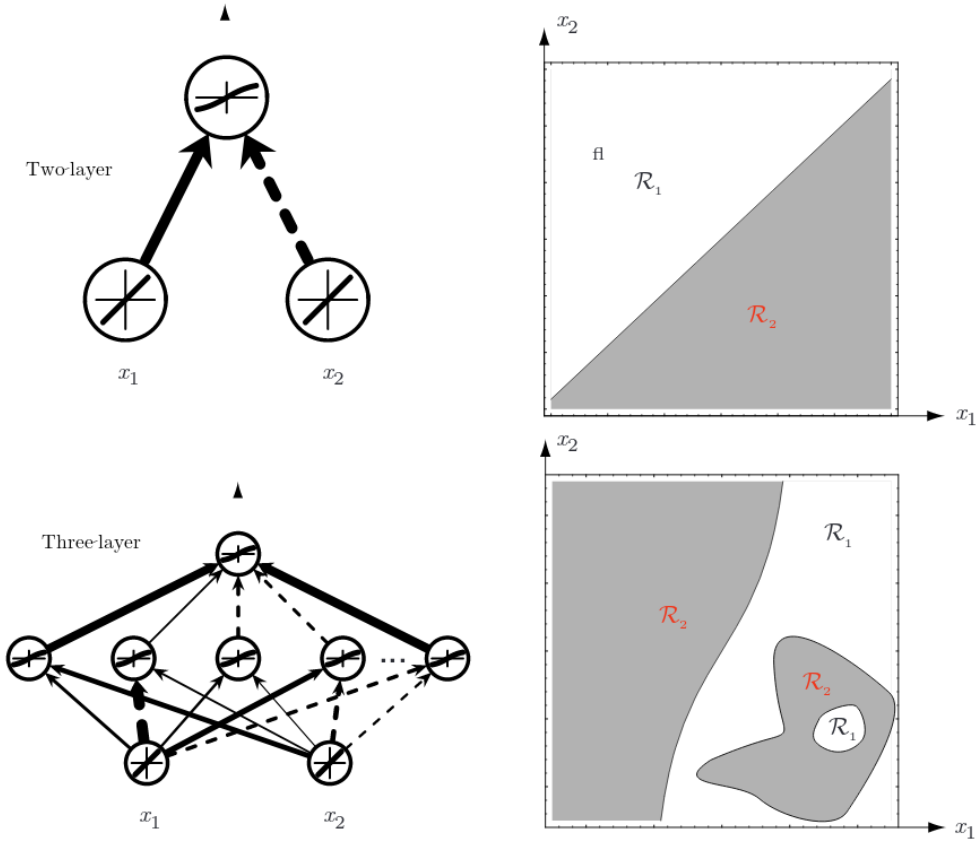
$$\mathbf{y} = f(\mathbf{x}) = f^{(M)} \left(f^{(M-1)} \left(\dots f^{(1)}(\mathbf{x}) \right) \right)$$

Jeszcze raz XOR



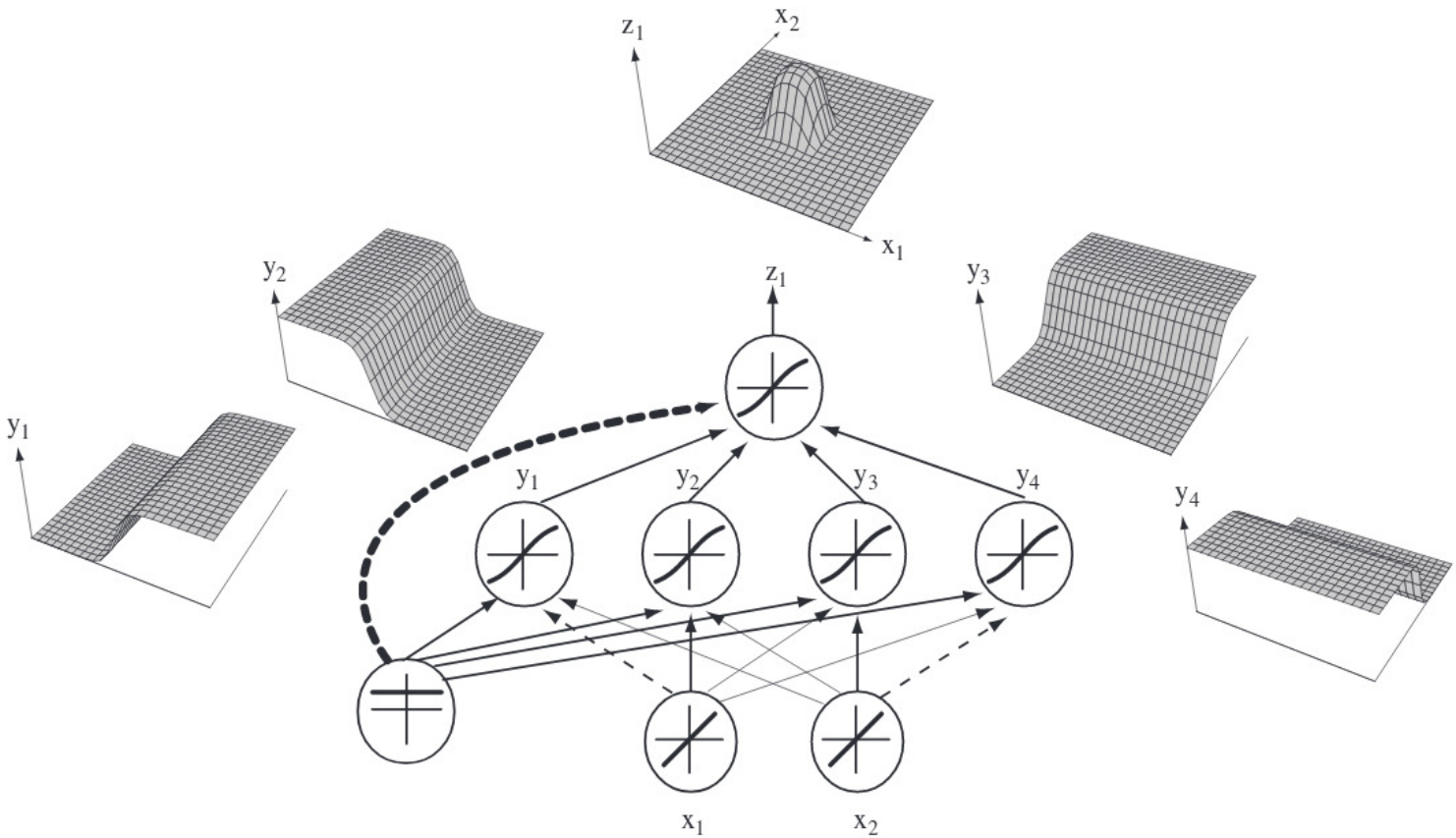
Rys: Duda, Hart, Pattern Classification

Warstwa ukryta a granice decyzji



Rys: Duda, Hart, Pattern Classification

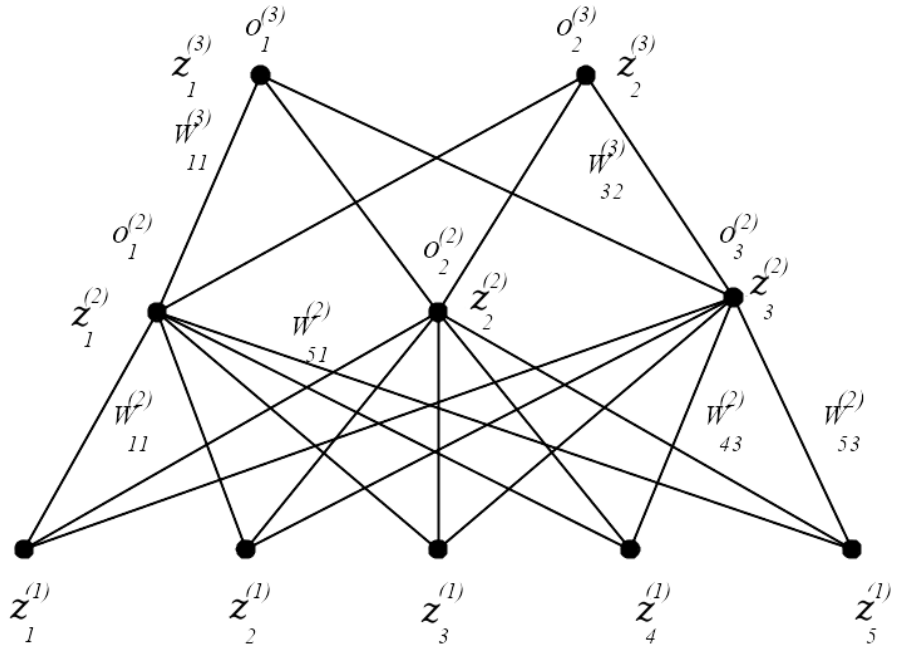
Przykład: sieć MLP 2-4-2



Rys: Duda, Hart, Pattern Classification

Perceptron 3-warstwowy (typowa sieć MLP)

- Warstwy: wejściowa, ukryta, wyjściowa

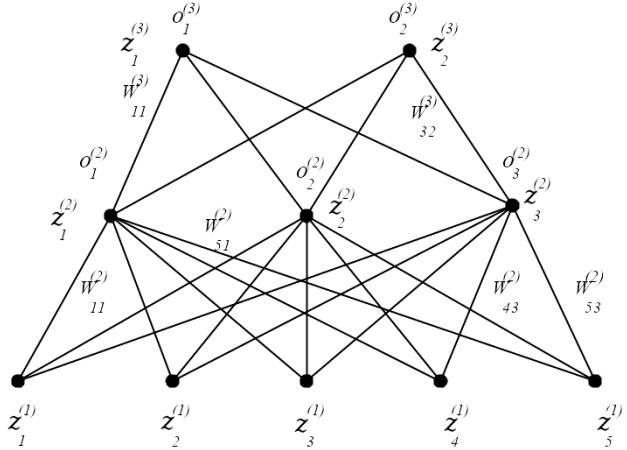


- Warstwy w pełni połączone (wagi neuronów w warstwie K tworzą macierz $\mathbf{W}^{(K)}$).
- Sygnał wyjściowy z warstwy K:

$$\mathbf{o}^{(K)} = \sigma\left(\mathbf{W}^{(K)}\mathbf{o}^{(K-1)}\right)$$

- Waga $w_{ij} = 0$ może być interpretowana jako brak połączenia

MLP - oznaczenia



M - liczba warstw ($M = 3$)

$w_{ij}^{(l)}$ - waga połaczenia neuronu i-tego z warstwy $l-1$ z neuronem j-tym z warstwy l

$z_j^{(l)}$ - aktywacja neuronu j w warstwie l

$$z_j^{(l)} = \sum_i w_{ij}^{(l)} o_i^{(l-1)}$$

$o_j^{(l)}$ - sygnał wyjściowy neuronu j należącego do warstwy l

$$o_j^{(l)} = \sigma(z_j^{(l)}) = \sigma\left(\sum_i w_{ij}^{(l)} o_i^{(l-1)}\right)$$

$f_i(\mathbf{x}) = o_i^{(M)}$ funkcja realizowana przez MLP (i-te wyjście sieci)

Algorytm wstecznej propagacji błędu

(1974, 1986)

Miara błędu sieci dla pojedynczego wzorca wejściowego $\mathbf{x}$ i pożądanej odpowiedzi $\mathbf{y} = [y_1, \ldots, y_n]$

$$E(\mathbf{x}; \mathbf{W}) = \frac{1}{2} \sum_{i=1}^{n} (y_i - f_i(\mathbf{x}; \mathbf{W}))^2$$

gdzie n to liczba wyjść

Minimalizacja błędu metodą gradientu prostego:

$$\Delta w = -\eta \frac{\partial E(\mathbf{x}; \mathbf{W})}{\partial w} = \eta \sum_{i=1}^{n} (y_i - f_i(\mathbf{x}; \mathbf{W})) \frac{\partial f_i(\mathbf{x}; \mathbf{W})}{\partial w}$$

Zmiana wag dla warstwy wyjściowej

Dla wagi $w_{jk}^{(M)}$ łączącej wyjście $o_j^{(M-1)}$ z k-tym wyjściem sieci

$$\begin{aligned}
\Delta w_{jk}^{(M)} &= \eta \sum_{i=1}^{n} \left(y_i - f_i(\mathbf{x}; \mathbf{W})\right) \frac{\partial f_i(\mathbf{x}; \mathbf{W})}{\partial w_{jk}^{(M)}} \\
&= \eta \left(y_k - f_k(\mathbf{x}; \mathbf{W})\right) \frac{\partial f_k(\mathbf{x}; \mathbf{W})}{\partial w_{jk}^{(M)}} \\
&= \eta \left(y_k - f_k(\mathbf{x}; \mathbf{W})\right) \frac{\partial \sigma(z_k^{(M)})}{\partial z_k^{(M)}} \frac{\partial z_k^{(M)}}{\partial w_{jk}^{(M)}} \\
&= \eta \left(y_k - f_k(\mathbf{x}; \mathbf{W})\right) \sigma'(z_k^{(M)}) o_j^{(M-1)}
\end{aligned}$$

$$\Delta w_{jk}^{(M)} = \eta \left(y_k - f_k(\mathbf{x}; \mathbf{W}) \right) \sigma'(z_k^{(M)}) o_j^{(M-1)}$$

Niech $\delta_i^{(M)}$ oznacza błąd „lokalny” dla warstwy M

$$\delta_k^{(M)} = \sigma'(z_k^{(M)}) \left(y_k - f_k(\mathbf{x}; \mathbf{W}) \right)$$

Zmiana wag neuronów warstwy wyjściowej:

$$\Delta w_{jk}^{(M)} = \eta \delta_k^{(M)} o_j^{(M-1)}$$

$$\Delta w_{0k}^{(M)} = \eta \delta_k^{(M)}$$

Zmiana wag dla warstwy ukrytej $(M-1)$

Dla wagi $w_{jk}^{(M-1)}$ z warstwy ukrytej $M-1$:

$$\Delta w_{jk}^{(M-1)} = -\eta \frac{\partial E(\mathbf{W})}{\partial w_{jk}^{(M-1)}} = \eta \sum_{i=1}^{n} \left(y_i - f_i(\mathbf{x};\mathbf{W})\right) \frac{\partial f_i(\mathbf{x};\mathbf{W})}{\partial w_{jk}^{(M-1)}}$$

gradient

$$\frac{\partial f_i(\mathbf{x};\mathbf{W})}{\partial w_{jk}^{(M-1)}} = \frac{\partial \sigma(z_i^{(M)})}{\partial z_i^{(M)}} \frac{\partial z_i^{(M)}}{\partial w_{jk}^{(M-1)}} = \sigma'\left(z_i^{(M)}\right) \frac{\partial \left(\sum_l w_{li}^{(M)} o_l^{(M-1)}\right)}{\partial w_{jk}^{(M-1)}}$$

$$= \sigma'\left(z_i^{(M)}\right) w_{ki}^{(M)} \frac{\partial o_k^{(M-1)}}{\partial w_{jk}^{(M-1)}}$$

$$= \sigma'\left(z_i^{(M)}\right) w_{ki}^{(M)} \sigma'\left(z_k^{(M-1)}\right) o_j^{(M-2)}$$

Otrzymujemy:

$$\frac{\partial f_i(\mathbf{x};\mathbf{W})}{\partial w_{jk}^{(M-1)}} = \sigma'\left(z_i^{(M)}\right) w_{ki}^{(M)} \sigma'\left(z_k^{(M-1)}\right) o_j^{(M-2)}$$

stąd zmiana wag dla warstwy $M-1$

$$\begin{aligned}
\Delta w_{jk}^{(M-1)} &= \eta \sum_{i=1}^{n} \left(y_i - f_i(\mathbf{x};\mathbf{W})\right) \frac{\partial f_i(\mathbf{x};\mathbf{W})}{\partial w_{jk}^{(M-1)}} \\
&= \eta \sum_{i=1}^{n} \left(y_i - f_i(\mathbf{x};\mathbf{W})\right) \sigma'\left(z_i^{(M)}\right) w_{ki}^{(M)} \sigma'\left(z_k^{(M-1)}\right) o_j^{(M-2)} \\
&= \eta \sigma'\left(z_k^{(M-1)}\right) \sum_{i=1}^{n} w_{ki}^{(M)} \delta_i^{(M)} o_j^{(M-2)} \\
&= \eta \delta_k^{(M-1)} o_j^{(M-2)}
\end{aligned}$$

$$\Delta w_{0k}^{(M-1)} = \eta \delta_k^{(M-1)} \qquad \text{gdzie} \qquad \delta_k^{(M-1)} = \sigma'\left(z_k^{(M-1)}\right) \sum_{i=1}^{n} w_{ki}^{(M)} \delta_i^{(M)}$$

Struktura wzoru w kolejnych warstwach jest taka sama

Wsteczna propagacja błędu - podsumowanie

Funkcja realizowana przez sieć wielowarstwową

$$f_i(\mathbf{x};\mathbf{W}) = \sigma\left(\sum_j w_{ji}^{(M)}\sigma\left(\sum_k w_{kj}^{(M-1)}\sigma\left(\sum_l w_{lk}^{(M-2)}\ldots\sigma\left(\sum_n w_{nm}^{(2)}x_n\right)\right)\right)\ldots\right)$$

Aktualizacja wag w dowolnej warstwie K - **uogólniona reguła delta**

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

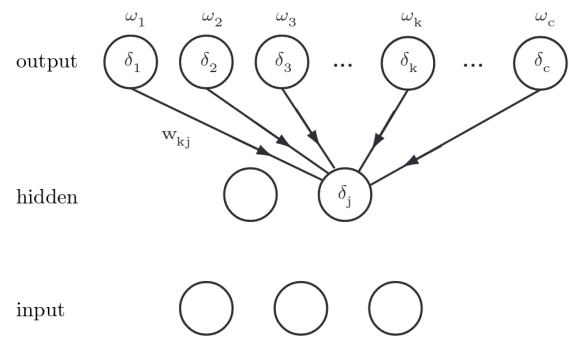
gdzie sygnał wejściowy $o_i^{(1)} = x_i$

Sygnał błędu w warstwie K

$$\delta_i^{(K)} = \sigma'\left(z_i^{(K)}\right)\sum_j \delta_j^{(K+1)}w_{ij}^{(K+1)}$$

sygnał błędu w warstwie wyjściowej :

$$\delta_i^{(M)} = \sigma'(z_i^{(M)})\,(y_i - f_i(\mathbf{x};\mathbf{W}))$$



Algorytm wstecznej propagacji

1. Zainicjuj wagi $\mathbf{W}$ małymi losowymi wartościami
2. Dopóki nie spłoniony warunek stopu wykonuj
   2.1 dla pary uczącej $(\mathbf{x}, \mathbf{y})$ oblicz sygnał uzyskany na kolejnych warstwach (od wejścia do wyjścia sieci)

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

   2.2 oblicz sygnał błędu warstwy wyjściowej

$$\delta_i^{(M)} = \sigma'(z_i^{(M)})\,(y_i - f_i(\mathbf{x}))$$

   2.3 oblicz sygnał błędu wszystkich warstw propagując błąd od warstwy wyjściowej do wejściowej

$$\delta_i^{(K)} = \sigma'\left(z_i^{(K)}\right) \sum_j \delta_j^{(K+1)} w_{ij}^{(K+1)}$$

   2.4 aktualizuj wagi

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

Algorytm wstecznej propagacji można uogólnić dla:

- sieci o innej strukturze połączeń: np. dodatkowe połączenia pomiędzy wejściem lub wyjściem, lub innymi warstwami ukrytymi. Dla sieci rekurencyjnych odpowiednikiem jest BPTT (wsteczna propagacja w czasie)
- dowolną liczbę warstw
- różne nieliniowości w warstwach
- różne funkcje aktywacji dla poszczególnych neuronów
- różne stałe uczenia dla każdej warstwy lub węzła
- inne funkcje kosztu, nie tylko MSE, np. Cross Entropy

Warianty treningu

Dla zbioru zawierającego N przypadków i sieci zawierającej M wyjść

$$\hat{E} = \frac{1}{N}\sum_{i=1}^{N} E(\mathbf{x}_i) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}(y_{ij} - f_j(\mathbf{x}_i))^2$$

$$\Delta\hat{w} = \sum_{i=1}^{N} \Delta w_i$$

Batch gradient descent

- uśredniony błąd całego zbioru treningowego
- niepraktyczne dla dużych danych
- stabilny trening
- nie zależy od kolejności prezentacji wzorców

Warianty treningu

SGD - stochastic gradient descent (on-line)

- aktualizacja dla pojedynczego przypadku treningowego ($N = 1$)
- duża szybkość działania ale i duża wariancja,
- istotna jest kolejność prezentacji przypadków

MiniBatch gradient descent

- średnia z n losowych przypadków (*mini-batch*), trening on-line,
- często utożsamiany z SGD
- mniejsza wariancja niż SGD
- operacje na macierzach, łatwiejsze zrównoleglenie GPU/CPU
- wielkość paczki można dobrać tak aby wykorzystać maksymalnie zasoby (pamięć) zwiększając szybkość przetwarzania

Własności MLP

- **Uniwersalny aproksymator:** sieć MLP z jedną warstwą ukrytą jest w stanie aproksymować dowolną funkcję ciągłą z dowolną dokładnością.
  Dwie warstwy ukryte rozszerzają możliwości na funkcje nieciągłe.
  Klasyfikator potrafi zrealizować dowolne granice decyzji (obszary nie muszą być wypukłe ani połączone)
- Neurony ukryte: transformacja nieliniowa do przestrzeni odwzorowań, tworząca nowe cechy za pomocą nieliniowych kombinacji
- Wiele zastosowań: klasyfikacja, wielowymiarowa regresja

Problemy I

- Optymalizacja nieliniowych funkcji zawsze sprawia problemy, tu mamy złożenie (czasem wielu) funkcji nieliniowych
- Dobór architektury sieci: Ile węzłów w warstwie ? Jakie funkcje aktywacji? Sieci ontogeniczne (rozrastające się) dostosowujące rozmiar do złożoności problemu
- Przeuczenie i generalizacja - zbyt duża liczba optymalizowanych parametrów powoduje przeuczenie, zbyt mała - może generować zbyt proste rozwiązania
- Regularyzacja - metody ograniczenia zjawiska przeuczenia, np. modyfikacje funkcji kosztu, ograniczenie liczby parametrów
- Inicjalizacja parametrów - szybkość treningu i wynik zależy od punktu startowego
- Minima lokalne i plateau, wąskie „rynny” - np. wielokrotny start
- Wpływ nowych wzorców na już nauczone – zapominanie

Problemy II

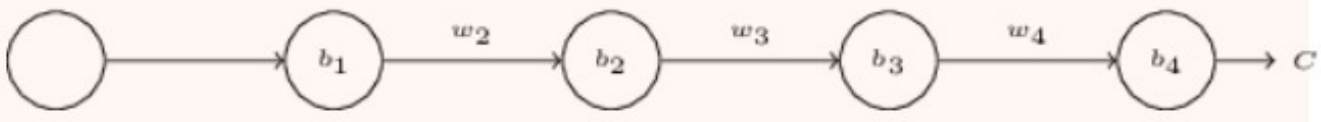
- Dobór stałej uczenia
- Znikający gradient, eksplodujący gradient
- Przygotowanie danych uczących: normalizacja, standaryzacja, kodowanie wyjść, ...
- Ocena modelu: zbiory walidacyjne, testowe, kroswalidacja

Znikający gradient

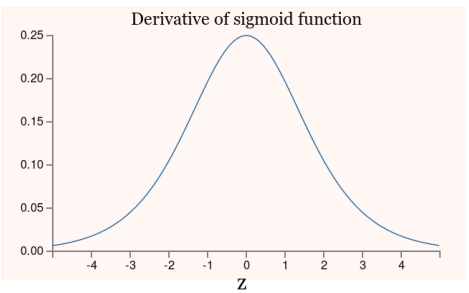
Problem **zanikającego gradientu** (ang. *vanishing gradient*) - zjawisko, w którym szybkość uczenia (zmiany wag) wsteczną propagacją drastycznie maleje w głębszych warstwach sieci - sygnał błędu zanika podczas propagacji do kolejnych warstw.

Przykład: sieć 1 x 1 x 1 x 1

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



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



Nielsen, 2016

Czynniki wpływające na zanikanie gradientu:

- gdy wagi lub pochodne przyjmują wartości mniejsze od 1, ich iloczyn maleje wykładniczo wraz z głębokością sieci
- funkcje aktywacji z nasyceniem: funkcje sigmoidalna i tanh mają małe pochodne w obszarach nasycenia
- nieprawidłowa inicjalizacja wag może prowadzić do mnożenia przez wartości < 1
- duża głębokość architektury sieci, problem zanikania gradientu nasila się wraz z liczbą warstw

Konsekwencje dla uczenia sieci

- brak aktualizacji wag w początkowych warstwach w czasie uczenia
- powolna zbieżność lub całkowite zatrzymanie procesu uczenia

Problem niestabilnego gradientu

- gradient w niższych warstwach zależy od wartości wag, aktywacji i gradientów w warstwach wyższych
- **eksplodujący gradient**
  gdy $w_i >> 1$ oraz $z_i \approx 0$ wówczas $\left|w_i \sigma'(z_i)\right| > 1$, 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, RPROP, Quckprop,
  - Adam, AdaGrad, ...
- Metody regularyzacji:
  - momentum, L_1, L_2 (weight decay), ...

Funkcja kosztu MSE

Błąd średniokwadratowy (*mean squared error)*

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

- średnia kwadratów różnic między wartościami prawdziwymi a przewidywanymi
- stosowana głównie w problemach regresyjnych
- 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 (wartość przewidywana przez model),
Θ - parametry sieci (wagi),
y - oczekiwane wyjścia sieci

Funkcja kosztu Cross Entropy

Entropia krzyżowa (*cross entropy*)

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

- mierzy dysproporcje miedzy rozkładem prawdziwym y i przewidywanym $\hat{y} \in (0, 1)$
- stosowana w problemach klasyfikacji wieloklasowej, gdzie klasy kodowane są w reprezentacji binarnej (*one-hot encoding*)
- gdy $y \approx \hat{y}$ to funkcja kosztu bliska zeru
- $J > 0$
- gradient funkcji kosztu

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

☞ Wizualizacja funkcji CE

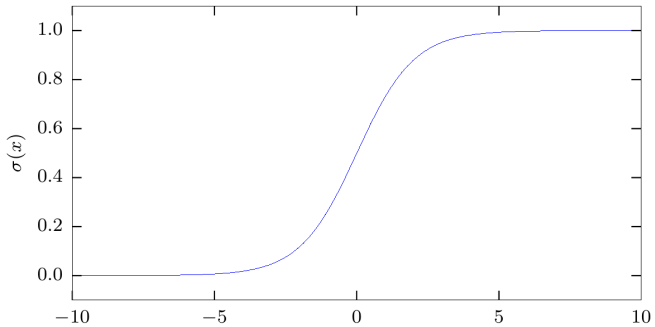
Funkcja liniowa w warstwie wyjściowej

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

- nieograniczone wartości $(-\infty, +\infty)$
- jednostki 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), modelowanie rozkładów gusowskich

Funkcja logistyczna w warstwie wyjściowej

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



- wartości ograniczone $(0, 1)$
- nasycają się, aktywne uczenie tylko w okolicach $x \approx 0$
- zastosowanie: klasyfikacja binarna (2 klasy), modelowanie rozkładu Bernoullego, klasyfikacja wieloetykietowa (*multi-label classification*), gdzie na każdym wyjściu oczekiwana jest wartości binarna (0 lub 1)
- **binarna funkcja kosztu Cross-Entropy**, na każdym wyjściu oczekujemy stanu 0 lub 1

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

Funkcja sigmoidalna w warstwie wyjściowej

Podczas minimalizacji MSE, gdy funkcja sigmoidalna σ neuronu wyjściowego się nasyca to sygnał błędu zanika, także w sytuacji niepoprawnej odpowiedzi sieci, gdy błąd $|\hat{y} - y|$ jest duży

$$\frac{\partial \hat{y}}{\partial \Theta} = \frac{\partial \sigma(z)}{\partial \Theta} = \sigma'(z)\frac{\partial z}{\partial \Theta} \approx 0$$

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

Problem znika przy minimalizacji funkcji kosztu entropii krzyżowej

$$\frac{\partial J_{CE}}{\partial \Theta} = -\sum\left(\frac{y}{\hat{y}}\frac{\partial \hat{y}}{\partial \Theta} + \frac{1-y}{1-\hat{y}}\frac{\partial(1-\hat{y})}{\partial \Theta}\right) = \sum(\hat{y} - y)\frac{\partial z}{\partial \Theta}$$

gdzie z jest liniową aktywacją (*logit*) neuronu wyjściowego $\hat{y} = \sigma(z)$, pochodna f. sigmoidalnej $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Softmax w warstwie wyjściowej

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

- wartości $(0, 1)$ unormowane

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

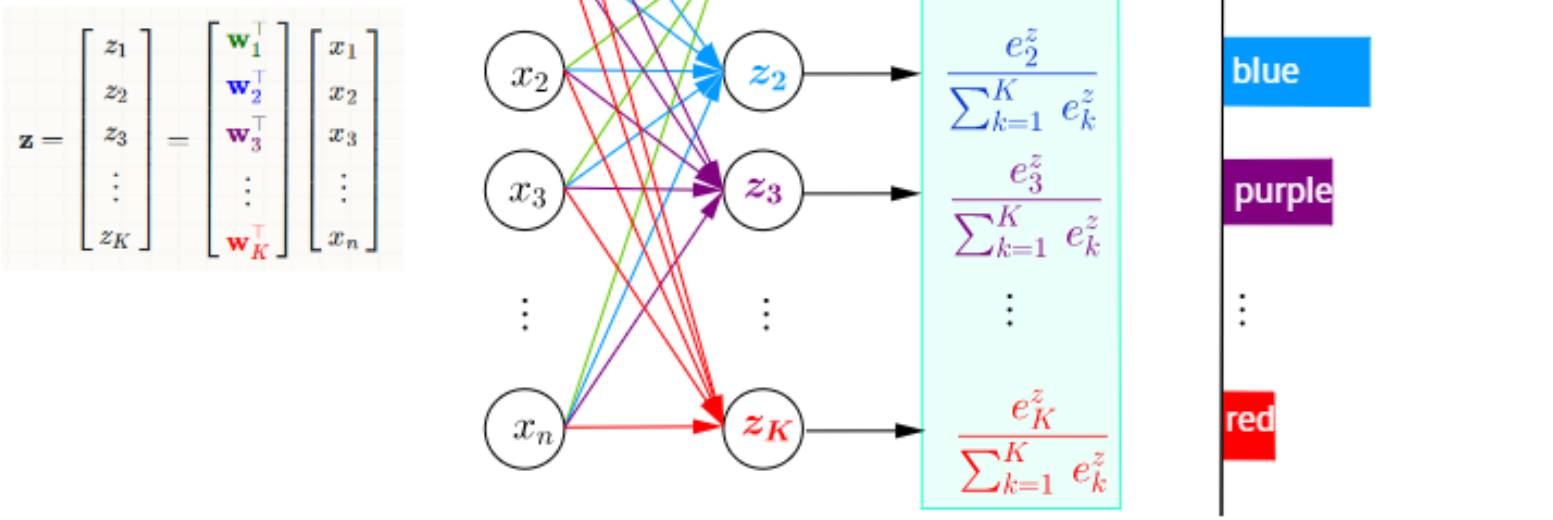
- zastosowanie: klasyfikacja n rozłącznych klas (*multi-class classification*), gdzie każde wyjście sieci związane jest z jedną klasą, etykiety klas kodowane są w postaci wektora binarnego (*one hot*)

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

- wartości wyjściowe sieci reprezentują rozkład prawdopodobieństwa przynależności do poszczególnych klas

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

Multi-Class Classification with NN and SoftMax Function



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

- gradient

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

- gradient funkcji kosztu entropii krzyżowej, gdy na wyjściu sieci softmax $\hat{y}_i = softmax(\mathbf{z})_i$

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

- wzór sie upraszcza eliminując problem zanikania sygnału błędu dla nasyconych wyjść, który występował dla funkcji kosztu MSE i wyjść sigmoidalnych
- w praktyce, podczas uczenia obliczanie logarytmu i softmax nie jest konieczne, wystarczy wartość *logit* $\mathbf{z}$

- ważne są różnice wartości pomiędzy wyjściami a nie amplitudy poszczególnych wyjść

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

- minimalizacja kosztu CE powoduje zwiększenie wartości na jednym z wyjść z_i i zmniejszenie wszystkich pozostałych $-\log \sum e^{z_k}$

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

- ponieważ

$$\log \sum_k e^{z_k} \approx \max_i z_i$$

  to dla poprawnej odpowiedzi $z_i \approx \max z_i$, błąd znika. Karane są najbardziej aktywne niepoprawne predykcje
- z punktu widzenia neurobiologicznego softmax może być widziany jako realizacja mechanizmów hamowania występującego w grupach neuronów w korze, podobnie do mechanizmu zwycięzca bierze wszystko (*winner takes all*)

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

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

Gooffellow, Deep Learning, 2016 [?]

Dobór neuronów ukrytych

- wybór funkcji realizowanych przez warstwy ukryte jest kluczowy dla przebiegu procesu uczenia
- funkcje liniowe - wiele warstw liniowych sprowadza się do pojedynczej transformacji liniowej

$$f(\mathbf{x}) = \mathbf{W}_n \mathbf{W}_{n-1} \ldots \mathbf{W}_1 \mathbf{W}_1 \mathbf{x} = \mathbf{A}\mathbf{x}$$

- funkcje ograniczone: sigmoidalne i tangens hiperboliczny powodują problemy z niestabilnym gradientem, uczenie może utknąć, gdy funkcja się nasyci
- obecnie najpopularniejszym typem aktywacji w sieciach jest ReLU

Demonstracje

- ☞ TensorFlow playground
- ☞ ConvNetJS - Deep Learning in your browser