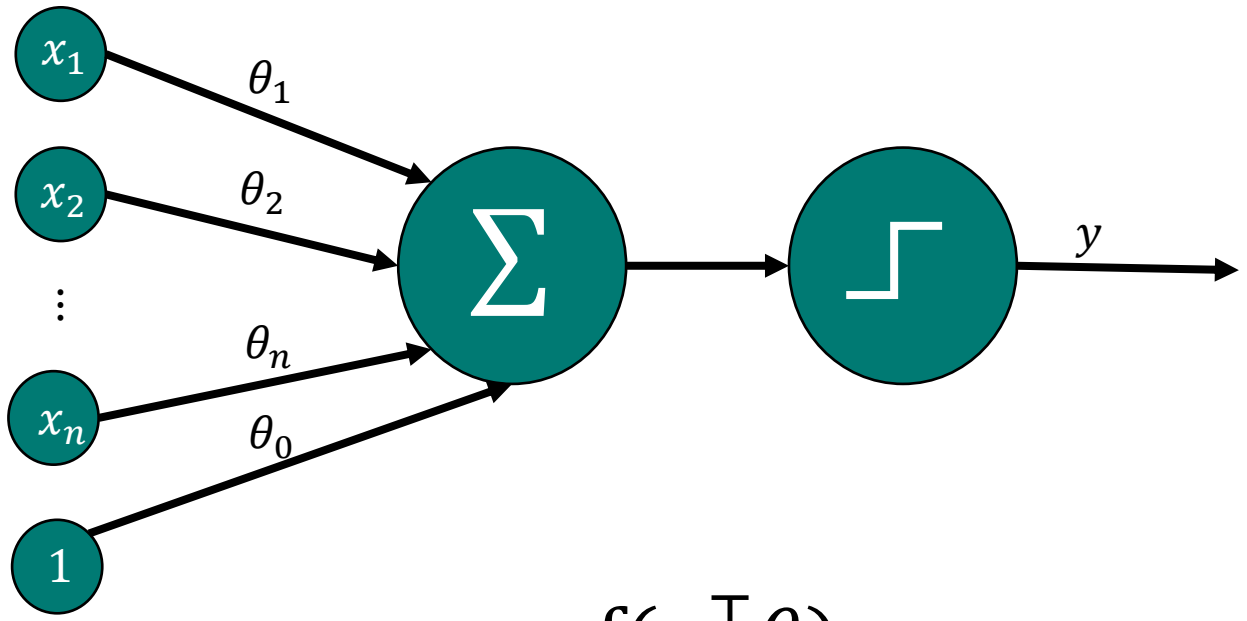
Matematyczne podstawy deep learningu

Wprowadzenie

- Perceptron (McCulloch, Pitts, 1943; Rosenblatt 1957):

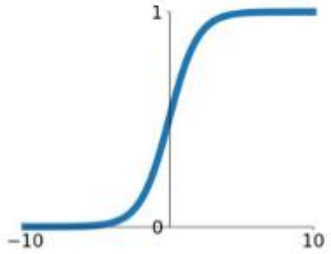


$$y = \mathrm{f}(x^{\top}\theta)$$

Funkcje aktywacji

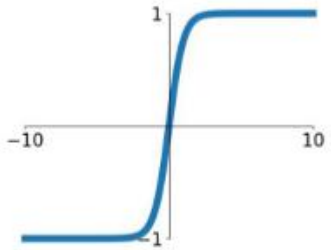
Sigmoid

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



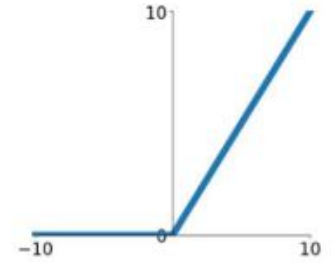
tanh

$$\tanh(x)$$



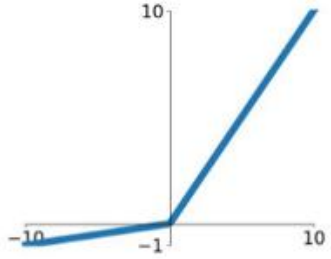
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

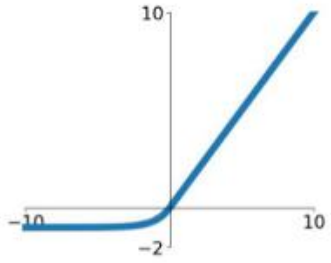


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

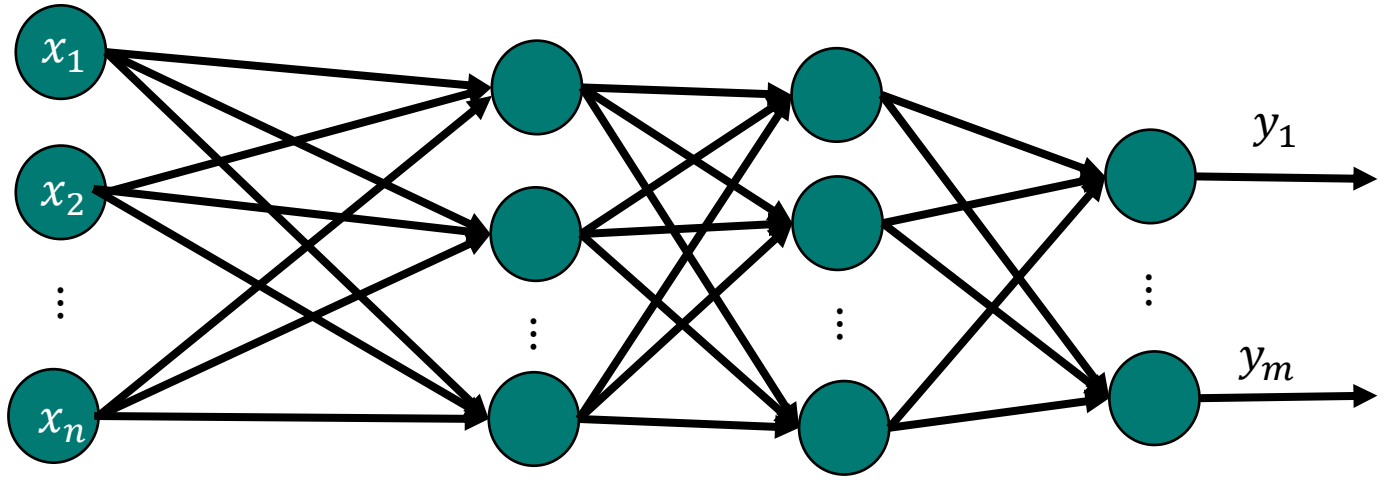
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Źródło: https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

Wprowadzenie

- Dużo bardziej interesujące od perceptronów są sieci z warstwami ukrytymi:

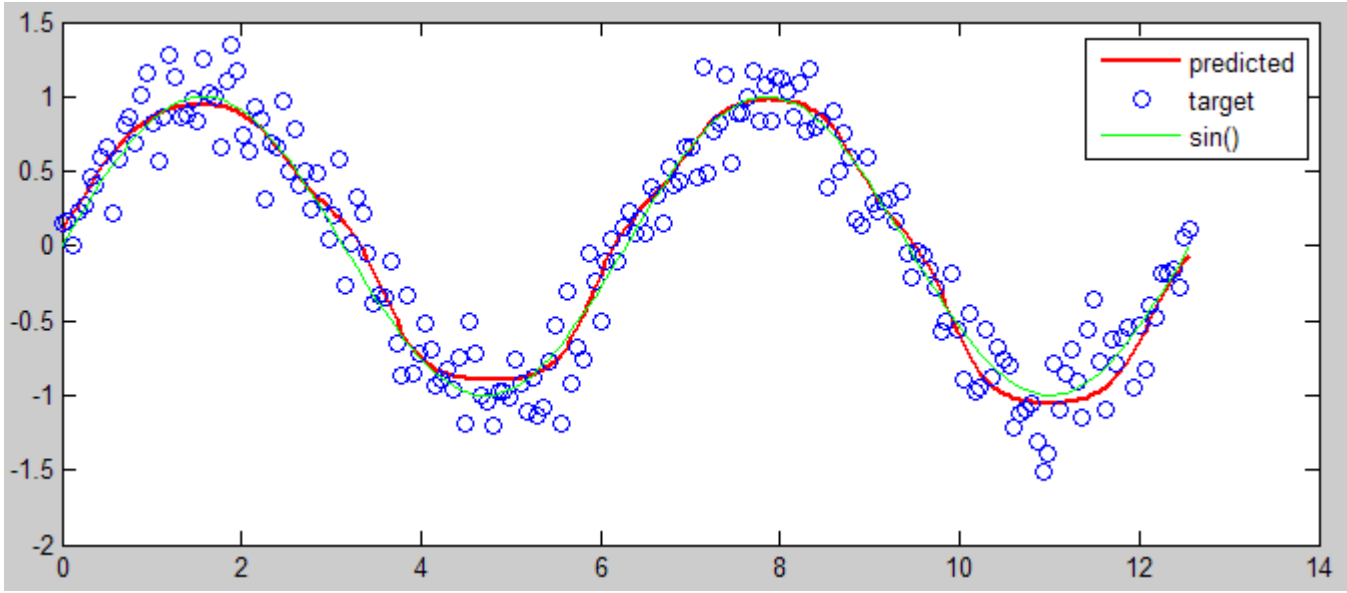


Wprowadzenie

- Problem uczenia sieci neuronowych możemy de facto sprowadzić do problemu aproksymacji pewnej funkcji $\phi(x)$ za pomocą złożenia funkcji: $\hat{\phi}(x;\theta) = \mathrm{f}^{(\mathrm{L})}(\ldots f^{(2)}\left(f^{(1)}\left(x,\theta^{(1)}\right),\theta^{(2)}\right),\theta^{(L)})$, gdzie $f^{(i)}$ oznacza **funkcję aktywacji** na i-tej warstwie, a $\theta^{(i)}$ wektor wag na i-tej warstwie.

Wprowadzenie

- Problem uczenia sieci neuronowych możemy de facto sprowadzić do problemu aproksymacji pewnej funkcji $\phi(x)$ za pomocą złożenia funkcji: $\hat{\phi}(x;\theta) = \mathrm{f}^{(\mathrm{L})}(\ldots f^{(2)}\left(f^{(1)}\left(x,\theta^{(1)}\right),\theta^{(2)}\right),\theta^{(L)})$, gdzie $f^{(i)}$ oznacza **funkcję aktywacji** na i-tej warstwie, a $\theta^{(i)}$ wektor wag na i-tej warstwie.



Źródło: https://stackoverflow.com/questions/1565115/approximating-function-with-neural-network

Wprowadzenie

- O ile funkcje aktywacji f musimy zadać z góry (są one **hiperparametrami** modelu), tak wagi θ są parametrami na które mamy wpływ (możemy je uczyć).
- Celem jest efektywna aproksymacja funkcji ϕ za pomocą aproksymaty$\hat{\phi}$.
- Rozwiązaniem tego problemu jest zoptymalizowanie wag θ tak aby funkcja $\hat{\phi}$ była jak najbliższa rzeczywistej funkcji ϕ.
- Nie możemy jednak zrobić tego wprost (nie znamy przebiegu funkcji ϕ, mamy tylko jej realizację w postaci zebranych danych) – optymalizacja w procesie uczenia sieci neuronowych musi odbywać się w sposób niejawny za pomocą zdefiniowanej **funkcji kosztu** $\mathrm{J}(\theta)$.

Optymalizacja sieci neuronowej

- Funkcje kosztu $\mathrm{J}(\theta)$ definiujemy zazwyczaj jako przeciętną wartość **funkcji straty** L:

$$\mathrm{J}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\, \mathrm{L}(\hat{\phi}(x;\theta), y)$$

  gdzie $\mathcal{D}$ to rozkład zmiennych x i y.

- Naszym celem jest **minimalizacja** funkcji $\mathrm{J}(\theta)$, czyli znalezienie takiego wektora wag θ dla którego **błąd aproksymacji** będzie możliwie najniższy.

Optymalizacja sieci neuronowej

- Funkcje kosztu $\mathrm{J}(\theta)$ definiujemy zazwyczaj jako przeciętną wartość **funkcji straty** L:

$$\mathrm{J}(\theta) = \mathbb{E}_{(x,y)\sim\hat{\mathcal{D}}}\,\mathrm{L}(\hat{\phi}(x;\theta), y)$$

  gdzie $\hat{\mathcal{D}}$ to rozkład empiryczny zmiennych x i y.

- Naszym celem jest **minimalizacja** funkcji $\mathrm{J}(\theta)$, czyli znalezienie takiego wektora wag θ dla którego zarówno **błąd aproksymacji** jak i **błąd estymacji** będą możliwie najniższe.

Definicja uczenia maszynowego

Pierwotna definicja uczenia statystycznego (Vapnik, 1999):

Dla zadanej klasy funkcji $\mathcal{F} = \{\alpha \in \Lambda: \hat{f}(x, \alpha)\}$, procesu generującego dane $\mathcal{D} = (X, Y)$ oraz funkcji straty $L(Y, \hat{Y})$ należy rozwiązać problem:

$$\hat{\alpha} = \underset{\alpha \in \Lambda}{\operatorname{argmin}} \left(\mathbb{E} \left(L\left(\hat{f}(x; \alpha), y \right) \right) \right)$$

Na podstawie próby $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ niezależnych i mających jednakowe rozkłady zmiennych losowanych z $\mathcal{D}$.

Optymalizacja sieci neuronowej

Table 1: List of losses analysed in this paper. $\mathbf{y}$ is true label as one-hot encoding, $\hat{\mathbf{y}}$ is true label as +1/-1 encoding, $\mathbf{o}$ is the output of the last layer of the network, $.^{(j)}$ denotes jth dimension of a given vector, and $\sigma(\cdot)$ denotes probability estimate.

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y}-\mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y}-\mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y}-\sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y}-\sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \|\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\|$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
$\log^2$	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2^2 + \|\mathbf{y}\|_2^2 - \sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log \frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2 \|\mathbf{y}\|_2}$

Źródło: (Janocha, Czarnecki 2017): https://arxiv.org/pdf/1702.05659.pdf

Optymalizacja sieci neuronowej

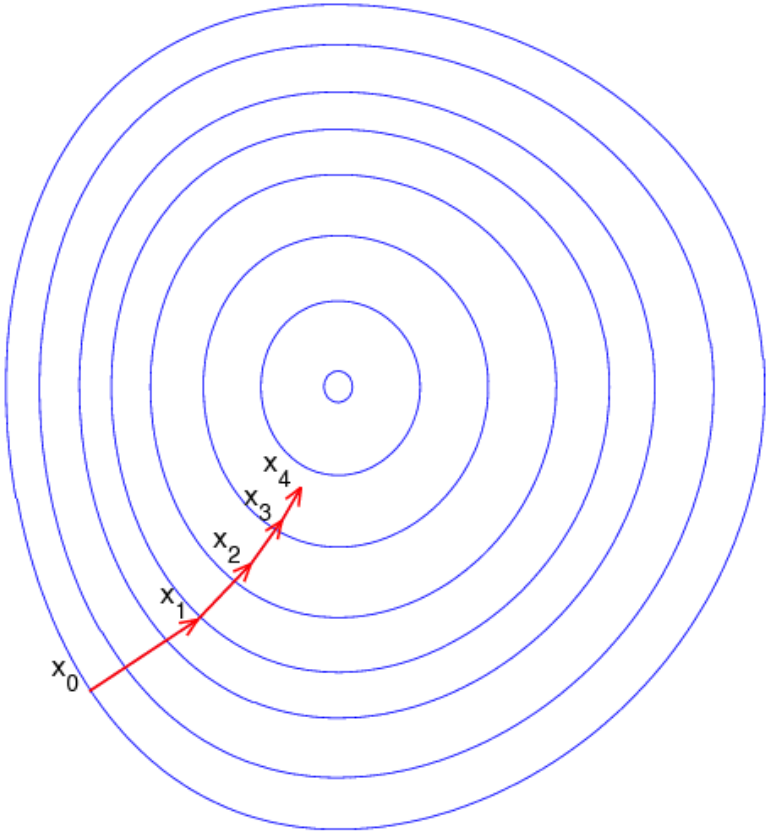
- Zazwyczaj nie robimy tego w sposób **deterministyczny** – optymalizacja funkcji kosztu dla każdej obserwacji ze zbioru testowego byłaby nieefektywna i powolna. Efektywniejszym rozwiązaniem jest **optymalizacja stochastyczna**.

Optymalizacja sieci neuronowej

- Najczęściej wykorzystywanym algorytmem optymalizacji uczenia sieci neuronowej są algorytmy z rodziny stochastycznych algorytmów gradientowych.

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while



Optymalizacja sieci neuronowej

- W przypadku ich wykorzystywania kluczowe jest wybranie odpowiedniej wartości **stopy uczenia (*learning rate*) η**
- Zazwyczaj nie przyjmuje się jej na stałym poziomie dla wszystkich iteracji algorytmu, tylko maleje ona z czasem.

Optymalizacja sieci neuronowej

- Warunki gwarantujące zbieżność algorytmu (Robbinsa-Munro):

$$\sum_{i=1}^{k} \eta_i = \infty$$

$$\sum_{i=1}^{k} \eta_i^2 < \infty$$

Błąd generalizacji

- Dla funkcji straty:

$$\mathrm{J}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\, \mathrm{L}(\hat{\phi}(x;\theta), y)$$

- I miary empirycznego ryzyka postaci:

$$\hat{J}\,(\theta) = \frac{1}{k}\sum_{i=1}^{k} \mathrm{L}(\hat{\phi}(x_i;\theta), y_i)$$

- Błąd generalizacji definiujemy jako:

$$\hat{J}\,(\theta)\; - \mathrm{J}(\theta)$$

Twierdzenie Hardta

Hardt M., Recht B., Singer Y. *Train faster, generalize better: Stability of stochastic gradient descent,* Mathematics of Control, Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, s. 1225-1234, 2016

- Zgodnie z twierdzeniem każdy model uczony za pomocą stochastycznej metody gradientów jest w stanie osiągnąć pomijalnie błąd generalizacji za pomocą dostatecznie dużej liczby kroków.

Twierdzenie Hardta

- Zdefiniujmy $A(S)$ jako algorytm generujący modele na próbce losowej $S = (z_1, z_2, \dots, z_n)$ generowanej $\mathcal{D}$. Wtedy górną granicę błędu generalizacji możemy przedstawić jako:

$$\mathbb{E}_{\mathrm{S,A}}\left[\hat{J}\left(A(S)\right) - \mathrm{J}(A(S))\right] \leq \epsilon_s(A, n)$$

Twierdzenie Hardta

Twierdzenie :

Niech $\forall y\ \phi(\cdot, y)$ będzie funkcją wypukłą spełniającą warunek L Lipschitza i niech $\nabla\phi$ spełnia warunek β Lipschitza. Przyjmijmy, że uruchomiliśmy algorytm stochastycznego gradientu T razy z krokiem $\eta_t \leq 2/\beta$. Wtedy:

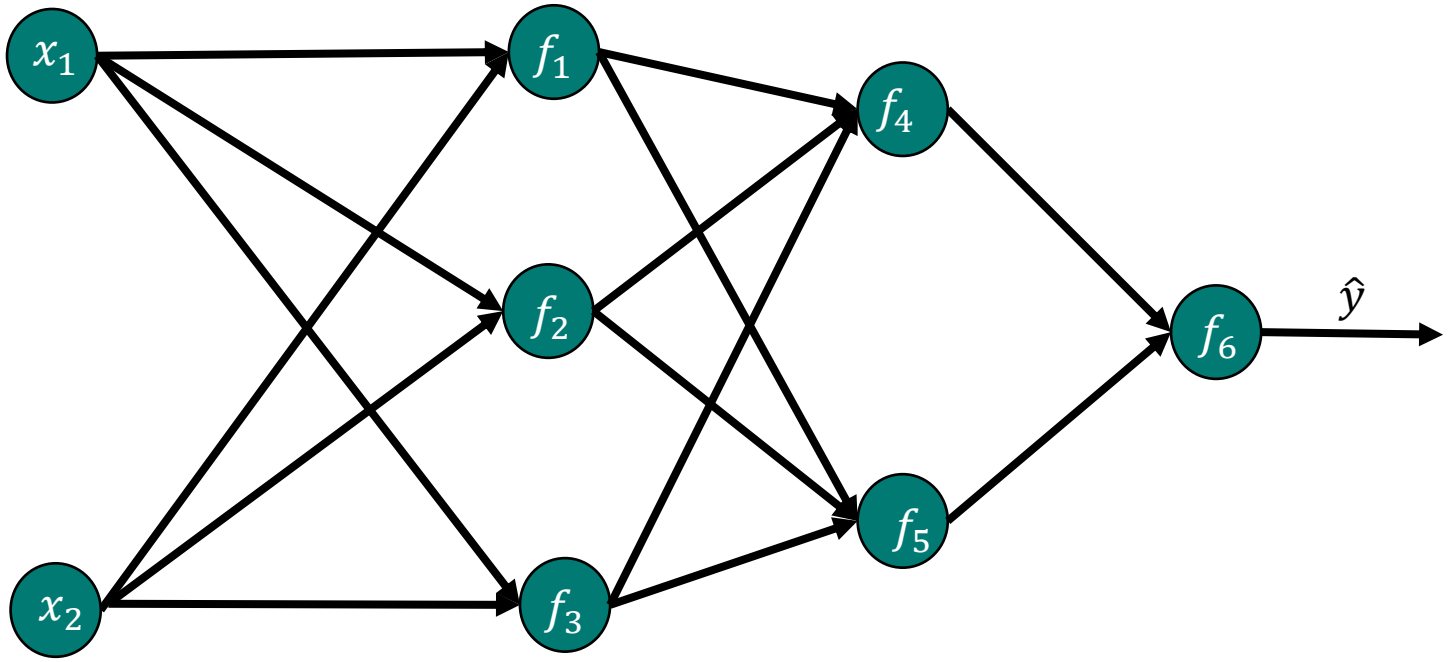
$$\epsilon_s = \frac{2L^2}{n} \sum_{t=1}^{T} \eta_t$$

Optymalizacja sieci neuronowej

- Algorytmem wykorzystywanym do optymalizacji jest zazwyczaj algorytm **propagacji wstecznej** (*backpropagation*).

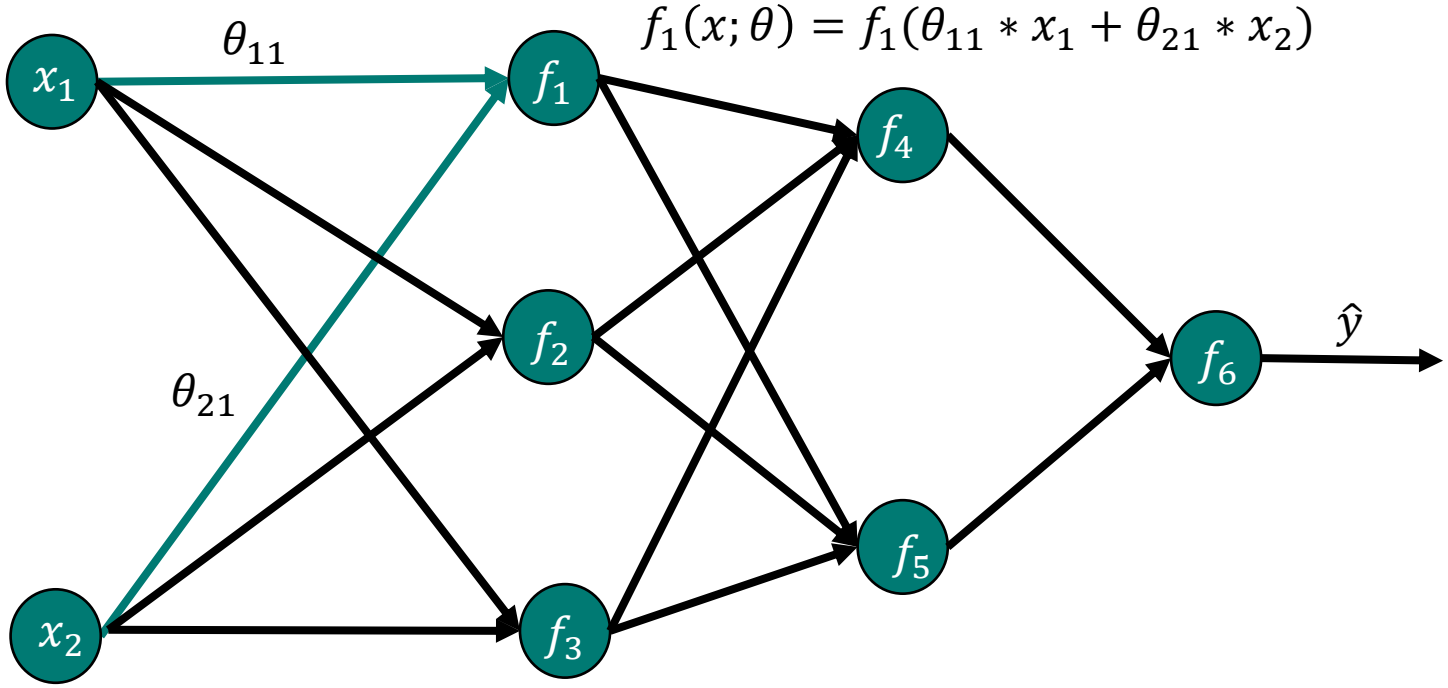
Optymalizacja sieci neuronowej

- Proces propagacji danych wewnątrz sieci neuronowej jest intuicyjny:



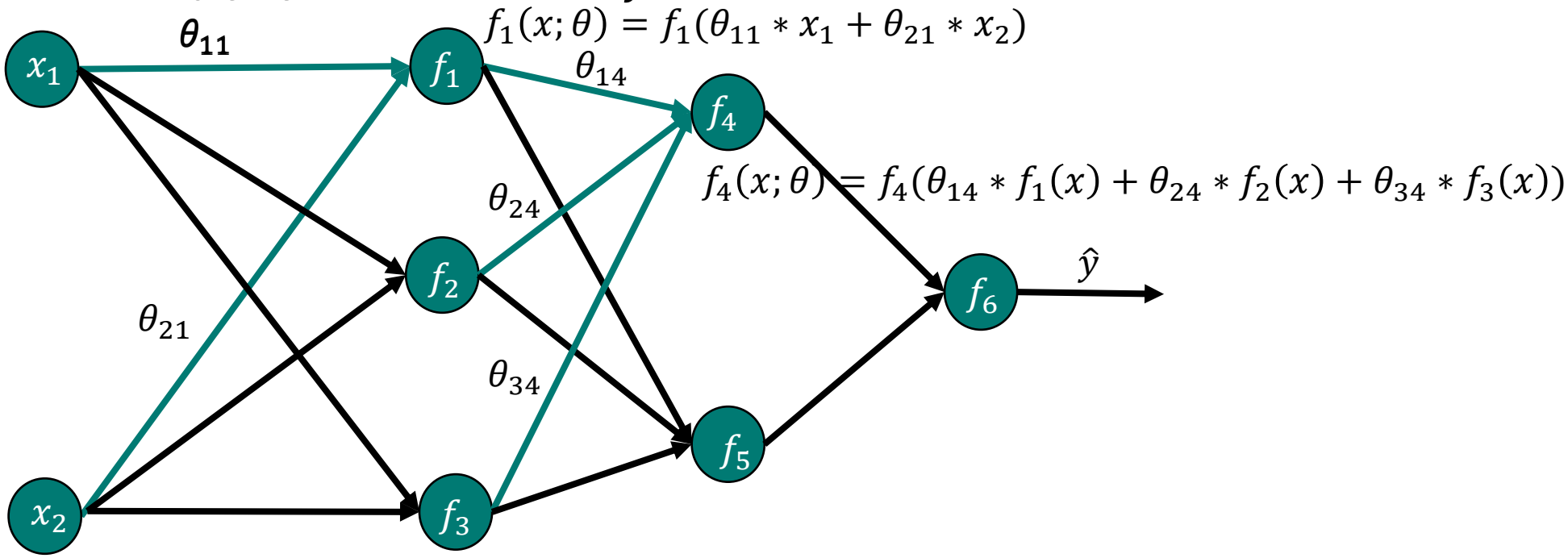
Optymalizacja sieci neuronowej

- Wartość każdego neuronu wewnątrz sieci jest kombinacją liniową danych z poprzedniej warstwy przetworzonych przed odpowiednią funkcję wprowadzającą nieliniowość f:



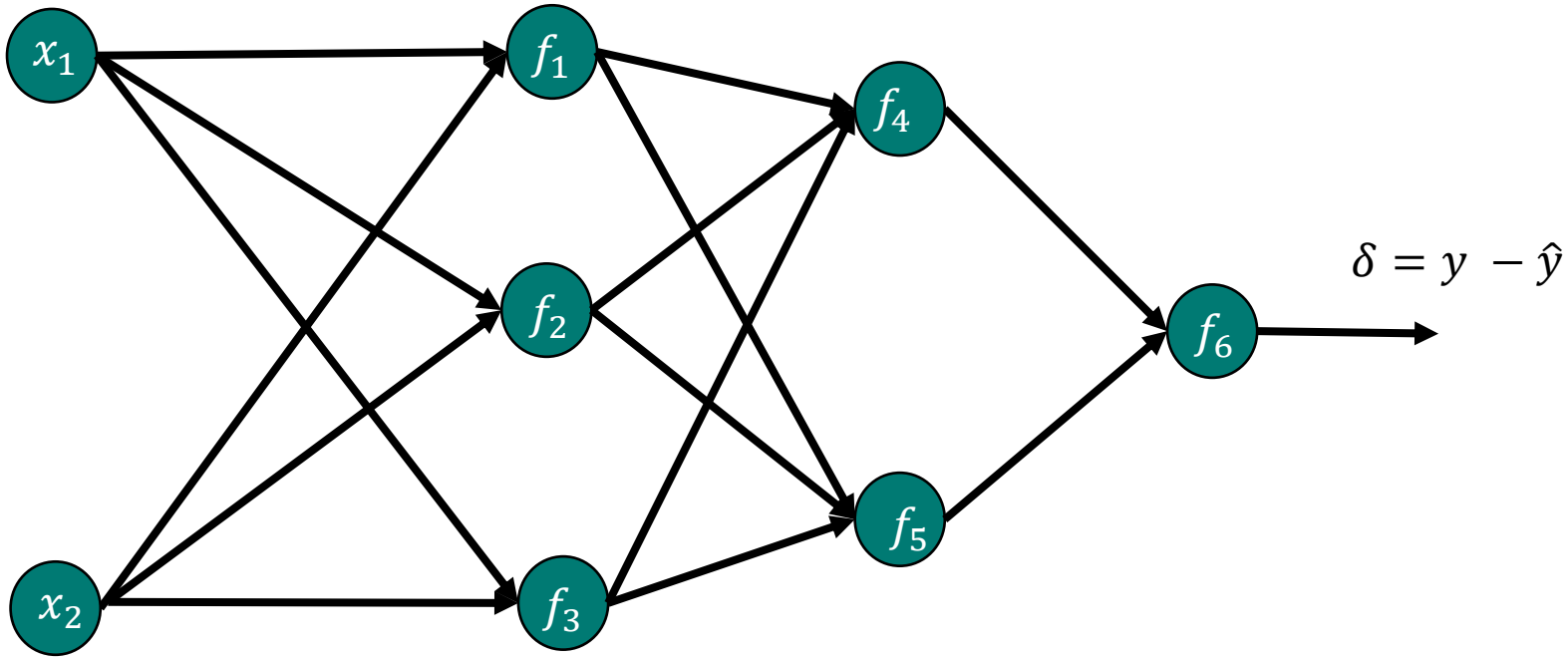
Optymalizacja sieci neuronowej

- Wartość każdego neuronu wewnątrz sieci jest kombinacją liniową danych z poprzedniej warstwy przetworzonych przed odpowiednią funkcję wprowadzającą nieliniowość f:



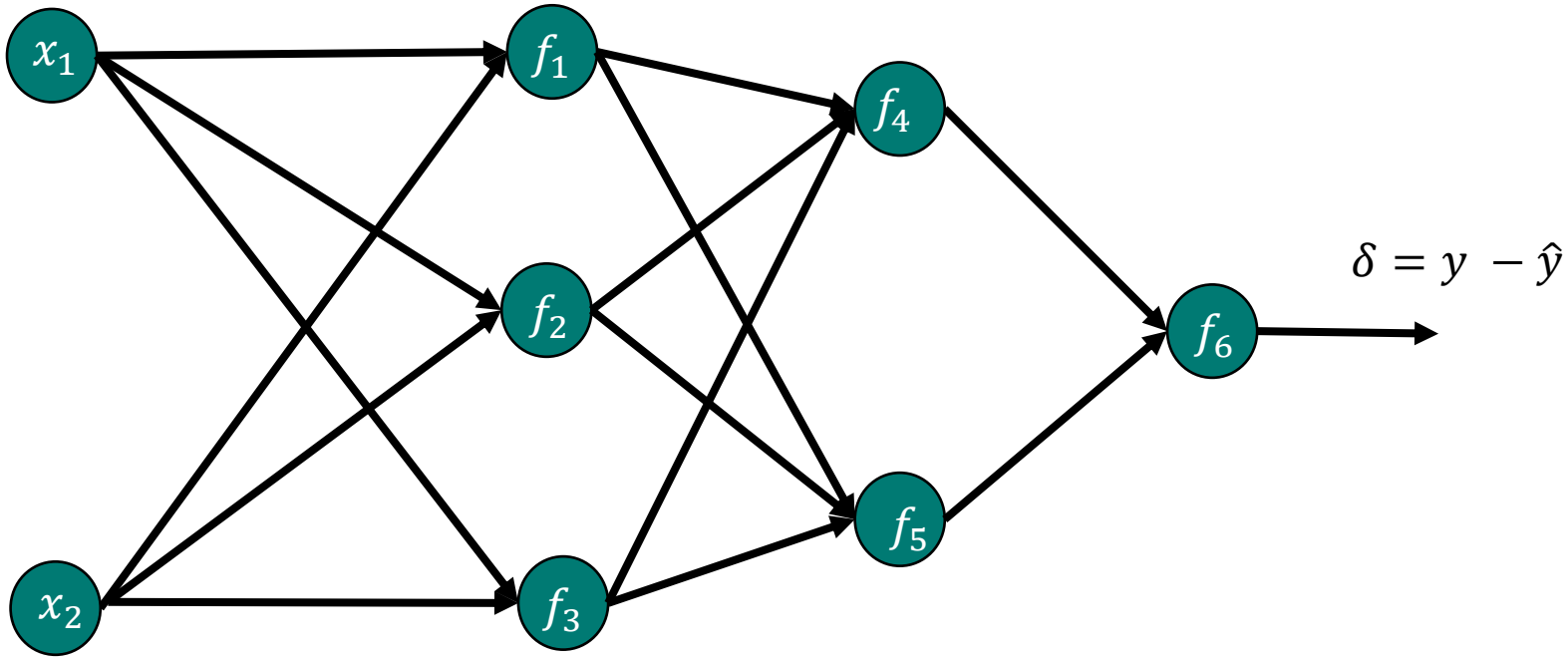
Optymalizacja sieci neuronowej

- Nietrywialne jest jednak uczenie sieci. Jesteśmy w stanie wyznaczyć błąd na wyjściowej warstwie:



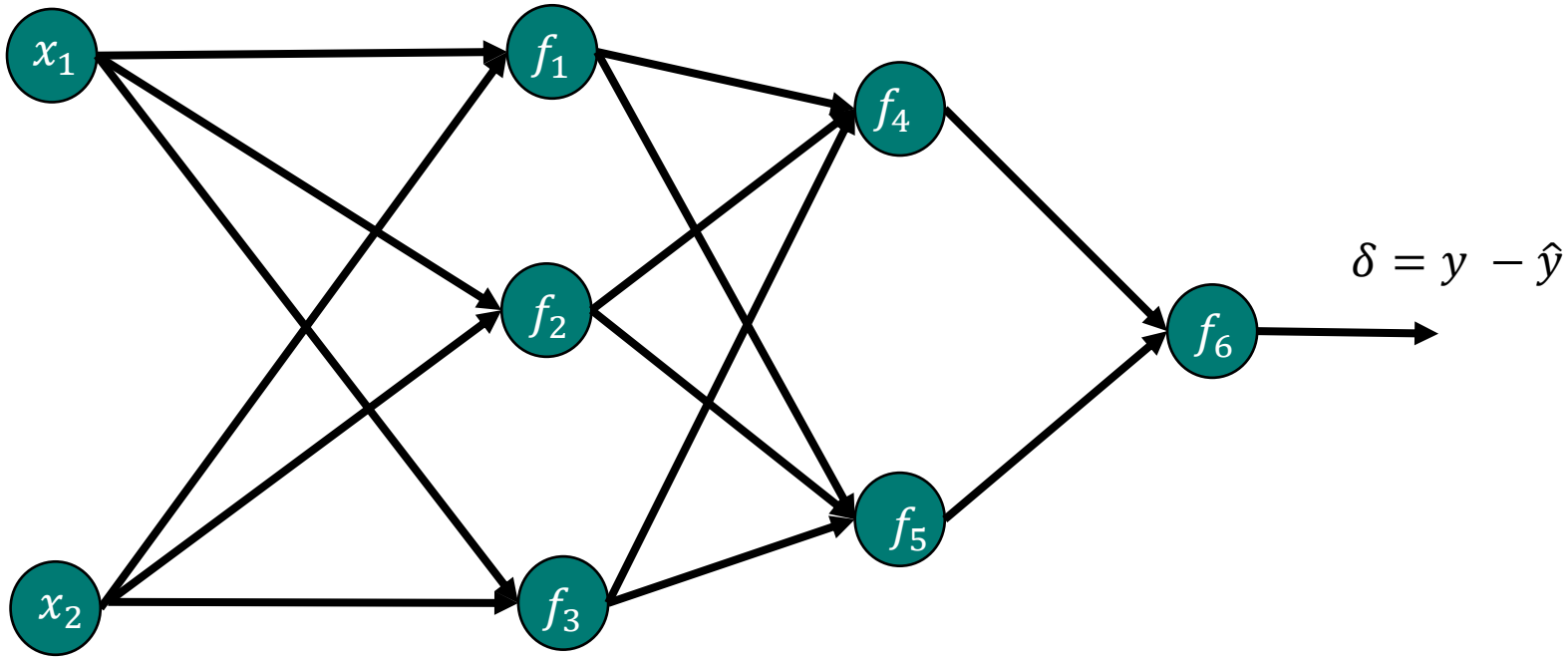
Optymalizacja sieci neuronowej

- Jak jednak wyznaczyć błąd dla warstw ukrytych?



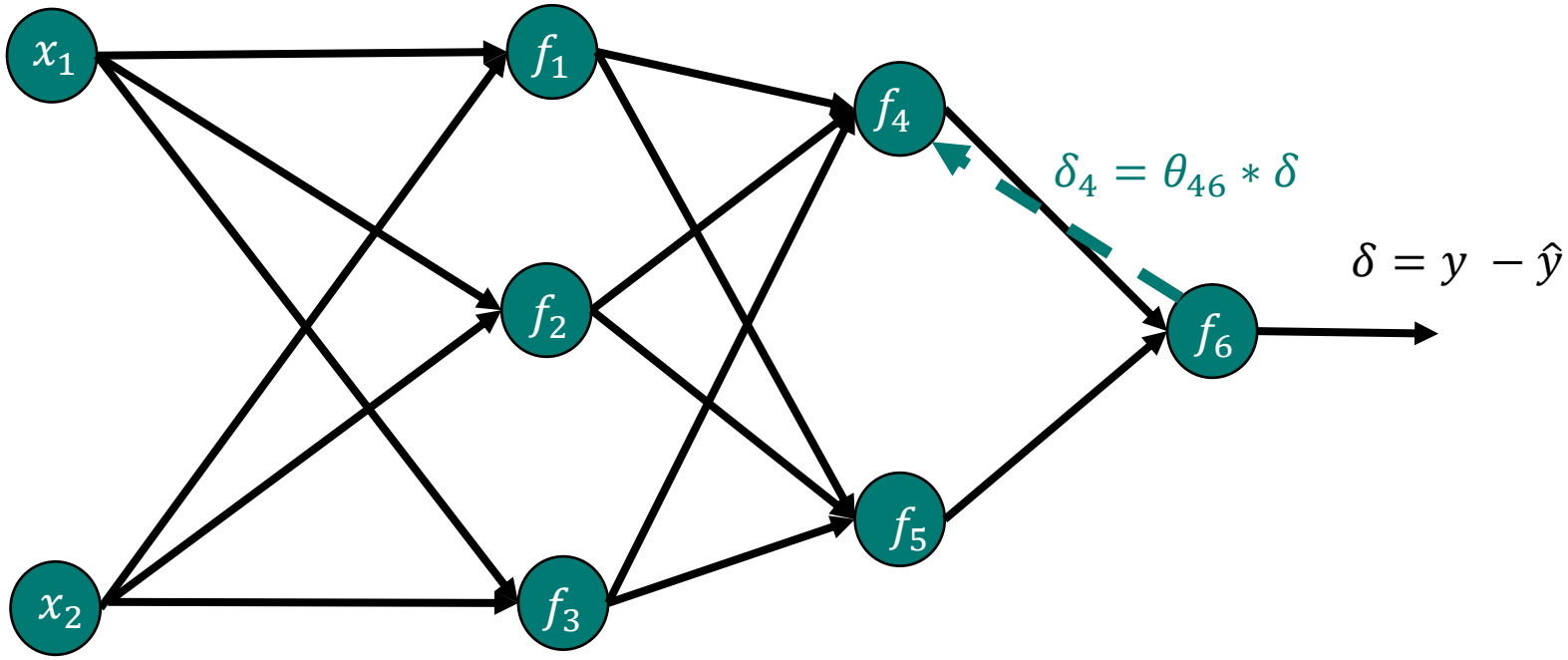
Optymalizacja sieci neuronowej

- Rozwiązaniem tego problemu jest wykorzystanie algorytmu propagacji wstecznej.



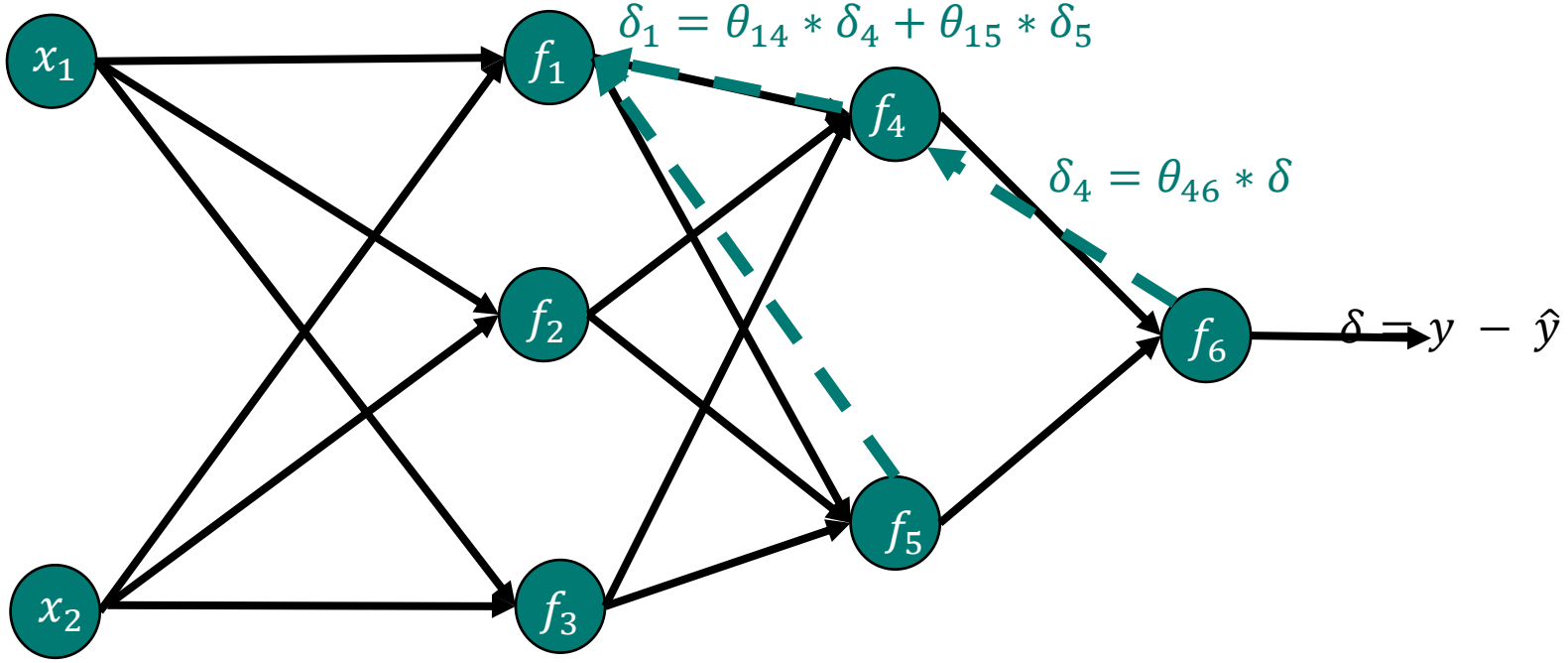
Optymalizacja sieci neuronowej

- Okazuje się, że błąd otrzymany na wyjściu można w efektywny sposób wykorzystać do wyznaczenia błędu na pozostałych warstwach. „Odwracając” ścieżkę jaką przechodzimy w sieci:



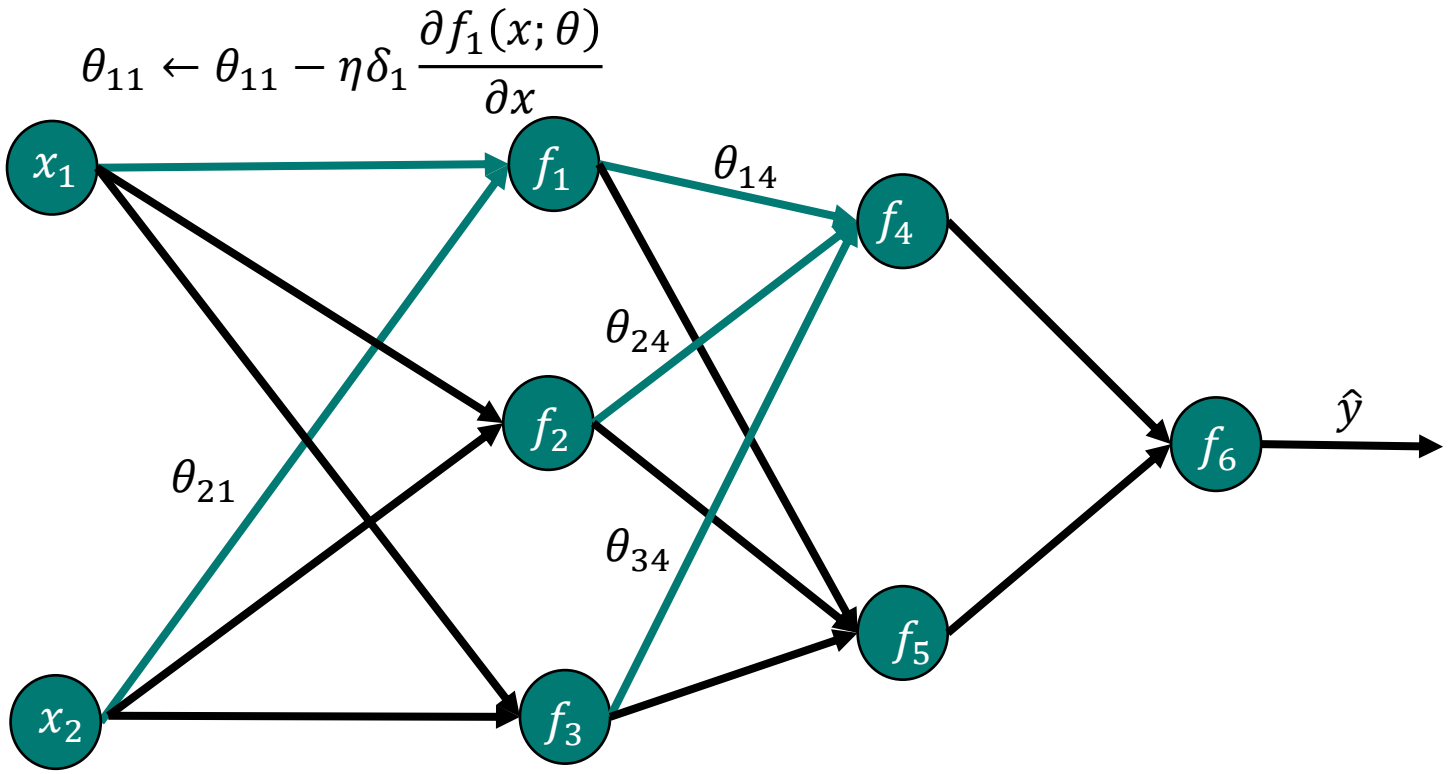
Optymalizacja sieci neuronowej

- Okazuje się, że błąd otrzymany na wyjściu można w efektywny sposób wykorzystać do wyznaczenia błędu na pozostałych warstwach. „Odwracając” ścieżkę jaką przechodzimy w sieci:



Optymalizacja sieci neuronowej

- Do modyfikacji wag wykorzystujemy algorytm gradientowy:



Sieć neuronowa jako aproksymata

- Dlaczego jednak sieć neuronowa jest efektywniejsza niż inne algorytmy aproksymacyjne?
- Okazuje się, że za pomocą jednokierunkowej sieci neuronowej z jedną warstwą ukrytą i sigmoidalną funkcją aktywacji $\sigma(x)$:

$$\sigma(t) \to \begin{cases} 1\ gdy\ \ x \to\ +\infty \\ 0\ gdy\ x \to -\infty \end{cases}$$

jesteśmy w stanie aproksymować dowolną funkcję $f(x)$ z zadaną dokładnością ϵ.

Twierdzenie Cybenki (twierdzenie o uniwersalnym aproksymatorze).

- **Cybenko G., *Approximation by superpositions of a sigmoidal function*, Mathematics of Control, Signals, and Systems, 2, s. 303-314, 1989**
- Zgodnie z twierdzeniem Cybenki okazuje się, że możliwa jest aproksymacja funkcji d-wymiarowej zmiennej rzeczywistej $x \in \mathbb{R}^d$ z zadaną dokładnością za pomocą skończonej liniowej kombinacji postaci:

$$G(x) = \sum_{i=1}^{n} \alpha_i \sigma(w_i^T x + b_i)$$

gdzie $w_i \in \mathbb{R}^d$ i $\alpha_i, b_i \in \mathbb{R}$ są stałe.

- Wystarczy, że powyższa sieć neuronowa zawiera dostatecznie dużą liczbę neuronów na warstwie ukrytej.

Twierdzenie Cybenki

- Aby móc formalnie wyprowadzić to twierdzenie zdefiniujmy kilka potrzebnych pojęć. Niech:
  - $\boldsymbol{I_n}$ oznacza n-wymiarowy hipersześcian jednostkowy $[0,1]^n$
  - $\mathbf{C}(\boldsymbol{I_n})$ przestrzeń ciągłych funkcji na $\boldsymbol{I_n}$
  - $||\boldsymbol{f}||$ oznacza normę supremum $\boldsymbol{f} \in \mathbf{C}(\boldsymbol{I_n})$
  - $\boldsymbol{\mathcal{M}}(\boldsymbol{I_n})$jest przestrzenią skończonych, borelowskich regularnych miar znakowanych na $\boldsymbol{I_n}$

Twierdzenie Cybenki

- Zdefiniujmy pojęcie **funkcji dyskryminującej**:
  - Funkcja σ jest **dyskryminująca** jeżeli dla miary $\mu \in \mathcal{M}(I_n)$

$$\int_{I_n} \sigma(w^T x + b) d\mu(X) = 0 \qquad \forall\, w \in \mathbb{R}^n \; i \; b \in \mathbb{R}$$

oznacza, że $\mu = 0$

Twierdzenie Cybenki

Twierdzenie 1:

Niech $\boldsymbol{\sigma}$ będzie dowolną ciągłą funkcją dyskryminującą. Wtedy skończona suma postaci:

$$G(X) = \sum_{i=1}^{n} a_i \sigma(w_i^T x + b_i)$$

jest zbiorem gęstym w $C(I_n)$.

Innymi słowy, dla dowolnego $f \in C(I_n)$ i $\varepsilon > O$ istnieje taka suma $G(X)$, dla której:

$$\|G(X) - f(X)\|_\infty < \varepsilon$$

Twierdzenie Cybenki

Twierdzenie 2:

Niech $\boldsymbol{\sigma}$ będzie dowolną ciągłą funkcją sigmoidalną. Wtedy skończona suma postaci:

$$G(X) = \sum_{i=1}^{n} a_i \sigma(w_i^T x + b_i)$$

jest zbiorem gęstym w $C(I_n)$.

Innymi słowy, dla dowolnego $f \in C(I_n)$ i $\varepsilon > O$ istnieje taka suma $G(X)$, dla której:

$$\|G(X) - f(X)\|_\infty < \varepsilon$$

Twierdzenie Cybenki

Twierdzenie 3:

Niech $\boldsymbol{\sigma}$ będzie dowolną ciągłą funkcją sigmoidalną. Niech $\boldsymbol{f}$ będzie funkcją decyzyjną dla dowolnego mierzalnego fragmentu I_n. Wtedy dla każdego $\epsilon > 0$ istnieje skończona suma postaci:

$$G(X) = \sum_{i=1}^{n} a_i \sigma(w_i^T x + b_i)$$

i zbiór $\boldsymbol{D} \subset \boldsymbol{I_m}$ taki, że miara Lebesque'a $m(D) \geq 1 - \epsilon$ oraz:

$$\|G(X) - f(X)\|_\infty < \varepsilon \text{ dla } x \in D$$

Twierdzenie Cybenki

- Twierdzenie Cybenki swoją treścią przypomina klasyczny wynik teorii aproksymacji:

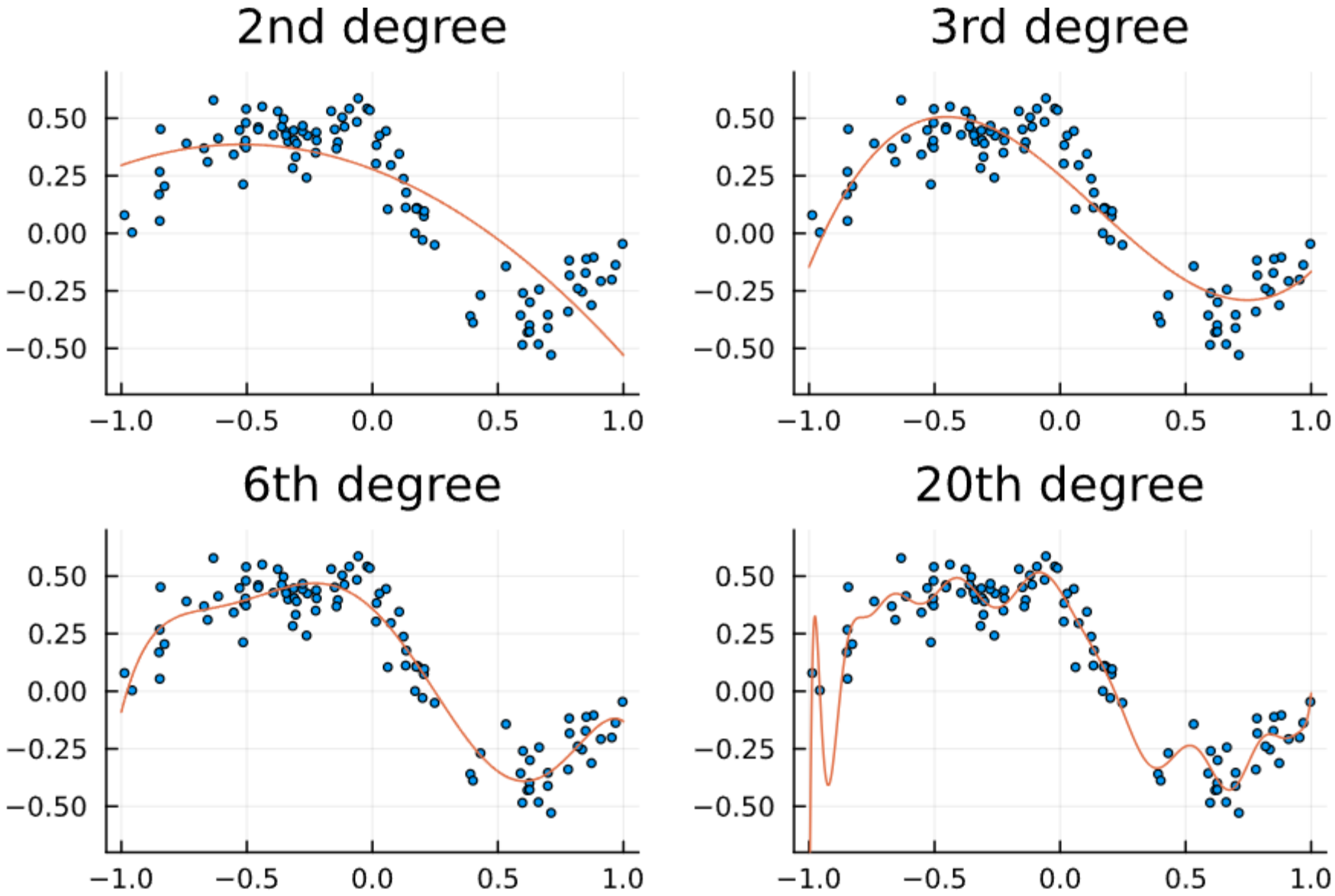
Twierdzenie Stone'a-Weierstrassa:

Jeżeli funkcja $f(x)$ jest określona i ciągła na skończonym przedziale $[a, b]$, to dla każdego $\epsilon > 0$ istnieje takie n i wielomian $\mathrm{W}_n(x)$ stopnia n, dla którego zachodzi nierówność:

$$|f(x) - W_n(x)| < \epsilon$$

na przedziale $[a, b]$.

Twierdzenie Cybenki



Sieć neuronowa jako aproksymata

- Twierdzenie Cybenki dotyczy jedynie specyficznego przypadku funkcji sigmoidalnej. Czy dla innych funkcji aktywacji aproksymacja jest podobnie efektywna?

Twierdzenie Hornika

- **Hornik K., *Approximation Capabilities of Multilayer Feedforward Networks* , Neural Networks, Vol. 4, pp. 251-257, 1991**
- Twierdzenie Hornika rozszerza wyniki otrzymane przez Cybenkę na znacznie szerszą przestrzeń funkcji σ, pokazując de facto że to sama w sobie architektura sieci neuronowej a nie wybór funkcji aktywacji wpływa na zdolność sieci do bycia aproksymatorem.

Sieć neuronowa jako aproksymata

- Oba wymienione twierdzenia pokazują siłę sieci neuronowych jako narzędzi aproksymacyjnych.
- Ale żadne z nich nie pokazuje jak powinna wyglądać sieć zdolna do efektywnej aproksymacji, jaka powinna być liczba neuronów na warstwie ukrytej, etc.
- Istnieje jednak cały szereg twierdzeń wykazujących rząd wielkości błędu oszacowania sieci w zależności od liczby neuronów i jej głębokości.

Twierdzenie Barrona

- **Barron A.E., Universal approximation bounds for superpositions of a sigmoidal function, IEEE Trans. on Information Theory, 39, s. 930-945, 1993**
- Zajmijmy się sytuacją w której naszym celem jest aproksymacja pewnej funkcji $f(X)$ za pomocą znanej już nam sieci neuronowej postaci: $G(X) = \sum_{i=1}^{n} a_i \sigma(w_i^T x + b_i)$
- Naszym celem jest osacowanie koniecznej liczby neuronów na warstwie ukrytej.

Twierdzenie Barrona

- Jedną z podstawowych miar odległości jest **MISE** *(mean integrated squared error)* definiujemy ją jako:

$$MISE = E\left|\left|G - f\right|\right|_2^2 = E[\int \left(G(X) - f(x)\right)^2 dx]$$

Twierdzenie Barrona

- Rozpatrujemy klasę funkcji f: $\mathbb{R}^d \longrightarrow \mathbb{R}$ i $f \in \Gamma_C$ dla których istnieje transformata Fouriera postaci:

$$f(x) = \int_{\mathbb{R}^d} e^{i\omega \cdot x} \hat{f}(\omega) d\omega$$

gdzie $\hat{f}(\omega)$ jest funkcją zespoloną dla której wyrażenie $\omega \hat{f}(\omega)$ jest różniczkowalne. Zdefiniujmy:

$$C_f = \int_{\mathbb{R}^d} \hat{f}(\omega) |\omega| d\omega$$

gdzie $|\omega| = (\omega \cdot \omega)^{1/2}$, a Γ_C jest zbiorem funkcji f takich, że: $\mathrm{C}_f \leq C$ i $0 < C$.

Twierdzenie Barrona

Twierdzenie:

Dla dowolnej funkcji $f \in \Gamma_c$, dowolnej sigmoidalnej funkcji $\boldsymbol{\sigma}$, dowolnej miary prawdopodobieństwa μ rozpiętej na kuli $B_r = \{x: |x| \leq r\}$ i $n \geq 1$ istnieje taka liniowa kombinacja n funkcji sigmoidalnych postaci $G(X) = \sum_{i=1}^{n} a_i \sigma\left(w_i^T x + b_i\right) + a_0$, że:

$$\int \left(G(X) - f(x)\right)^2 \mu(dx) \leq \frac{\left(2rC_f\right)^2}{n}$$

przy czym współczynniki wewnątrz funkcji $G(X)$ muszą spełnić 2 warunki:

1. $\sum_{i=1}^{n} |w_i| \leq 2\mathrm{rC}$
2. $a_0 = f(0)$

Twierdzenie Barrona

- Błąd aproksymacji jest rzędu co najwyżej $\mathrm{O}\left(\frac{1}{n}\right)$ a błąd estymatora opartego na jednowarstwowej sieci neuronowej:

$$\mathrm{O}\left(\frac{1}{n}\right) + \mathrm{O}\left(\frac{nd}{M}\ln M\right)$$

gdzie M oznacza wielkość próby losowej.

  - Widzimy też, że błąd aproksymacji zależy jedynie od liczby neuronów na warstwie ukrytej.
  - Oznacza to, że sieć neuronowa jest modelem, stosując który możemy uniknąć **przekleństwa wymiarowości** (*Curse of Dimensionality*).

Twierdzenie Barrona

- Jednak nadal problemem może być liczba neuronów na warstwie ukrytej konieczna do wygenerowania efektywnie aproksymującej sieci neuronowej.
- W najgorszym przypadku odpowiadać ona może wszystkim możliwym kombinacjom danych wejściowych 2^d.

Twierdzenie Barrona

- Okazuje się jednak, że efektywną metodą przeciwdziałania tego typu problemom jest zastosowanie więcej niż jednej ukrytej warstwy w swojej sieci.
- W wielu przypadkach budowa głębszego modelu pozwala na redukcję liczby neuronów potrzebnych do odpowiedniego reprezentowania aproksymowanej funkcji.
- O rzędzie wielkości błędu aproksymacji secie z wieloma warstwami ukrytymi mówią kolejne twierdzenia, które omówimy.

Twierdzenie Telgarskiego

- **Telgarsky M., Representation Benefits of Deep Feedforward Networks**
- Na bardzo prostym przykładzie Telgarsky pokazuje jak działa taki mechanizm.

Twierdzenie Telgarskiego

- Funkcję postaci:

$$m(x) = \begin{cases} 2x \ gdy \ x \in [0,0.5] \\ 2(1-x) \ gdy \ x \in [0.5,1] \end{cases}$$

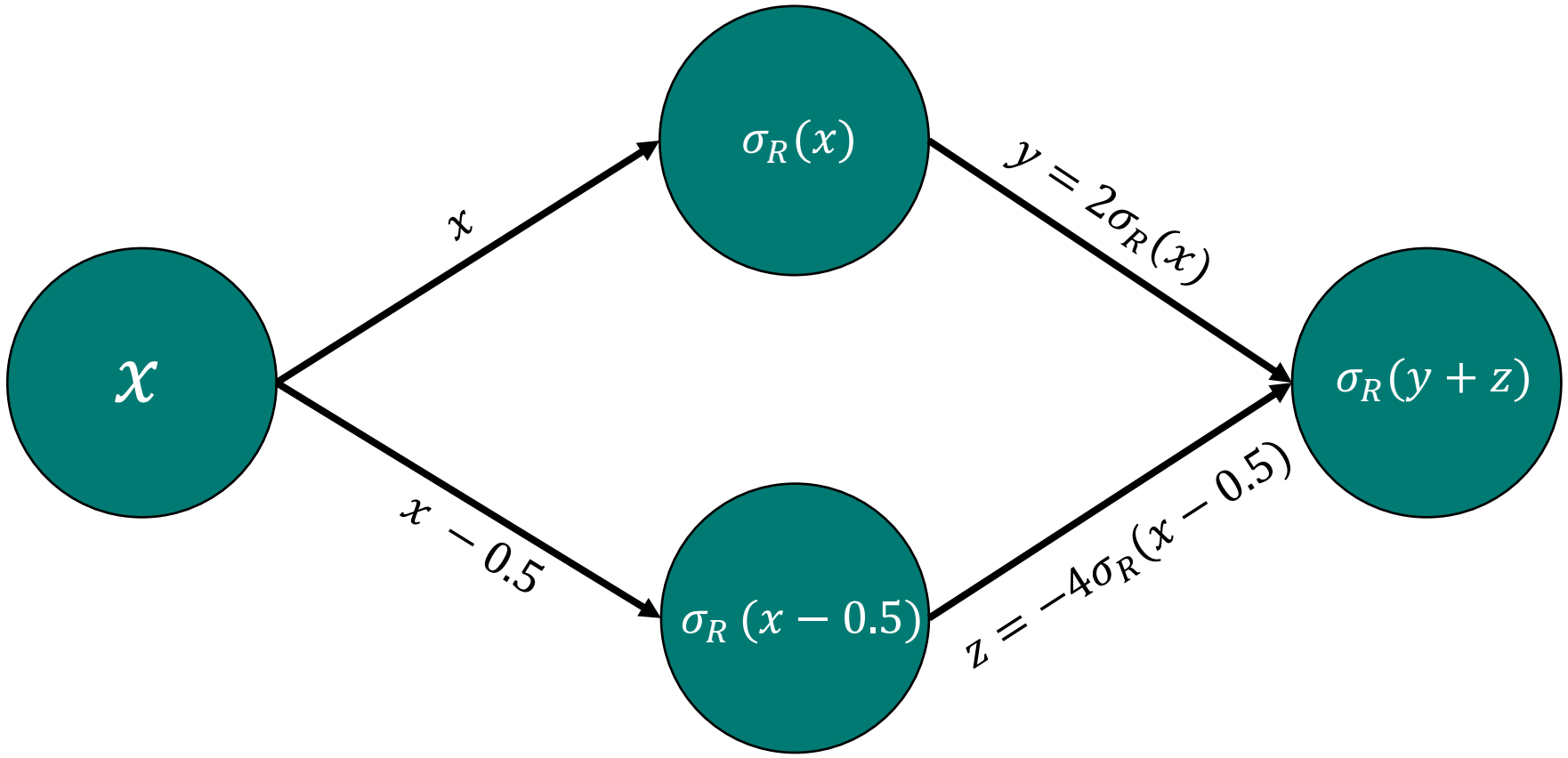
- Jesteśmy w stanie wyaprosymować ze 100% dokładnością za pomocą sieci neuronowej:

$$f(x) = \sigma_R(2\sigma_R(x) - 4\sigma_R(x - 0.5))$$

Gdzie σ_R to funkcja aktywacji ReLU: $\sigma_R(x) = \max(0, x)$

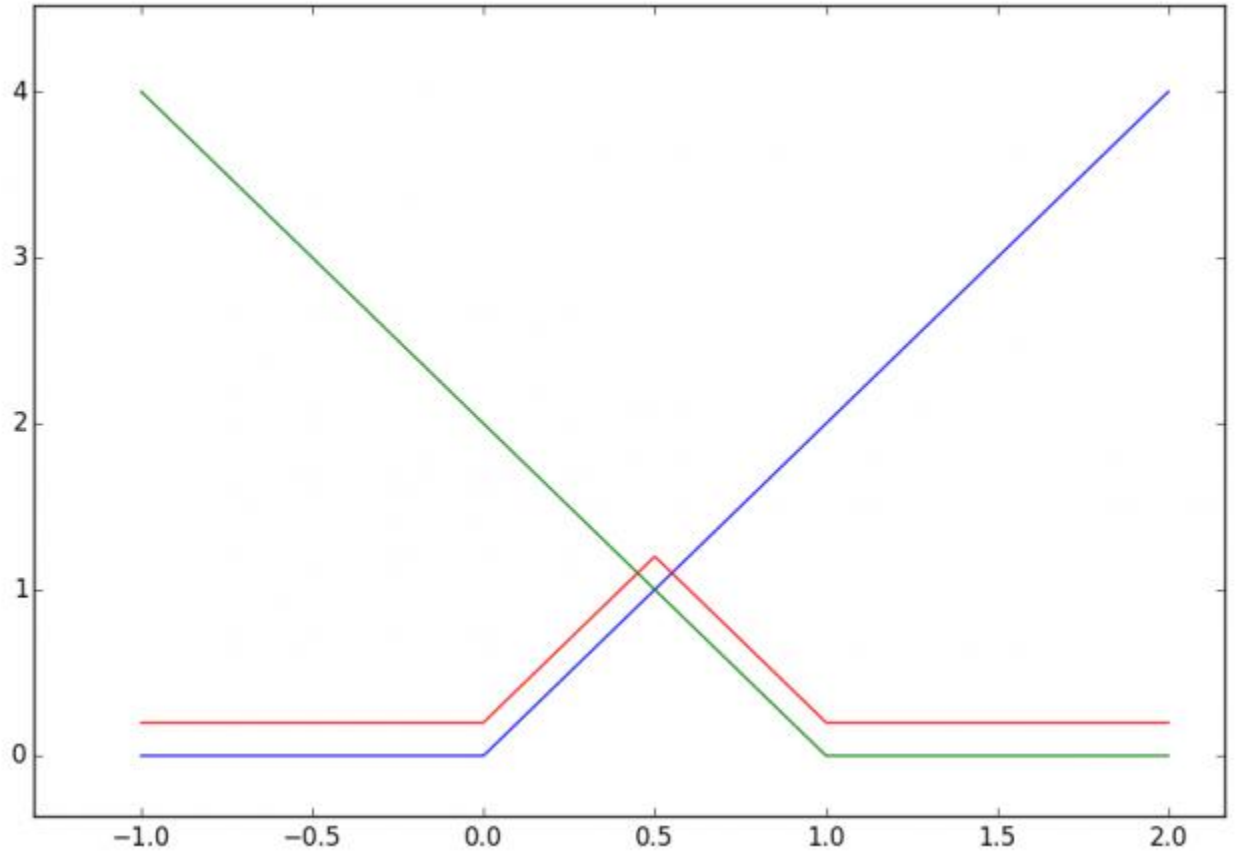
Twierdzenie Telgarskiego

- Taka sieć neuronowa ma postać:



Twierdzenie Telgarskiego

- A jej funkcje aktywacji wyglądają następująco:



Źródło: http://elmos.scripts.mit.edu/mathofdeeplearning/2017/04/09/mathematics-of-deep-learning-lecture-2/

Twierdzenie Telgarskiego

- Przy każdym kolejnym złożeniu funkcji $m\left(\ldots\left(m(x)\right)\right) = m^{(k)}(x)$ dla $k = 1,2,\ldots$ dwukrotnie wzrasta ilość jej „zębów”.
- Każdy nowy trójkąt ma o połowę mniejszą szerokość i taką samą wysokość.
- Funkcja $m^{(k)}(x)$ ma dokładnie 2^k przedziałów liniowych.
- Do jej aproksymacji ze 100% dokładnością możemy wykorzystać sieć neuronową o strukturze podanej wcześniej i dokładnie $3k + 1$ neuronach.

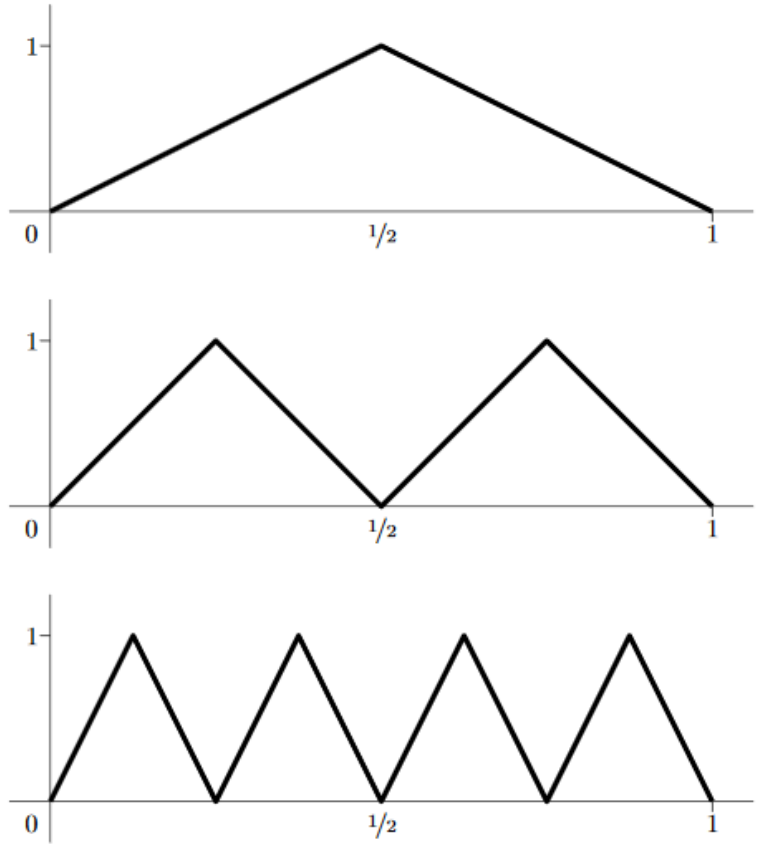


Figure 2: f_{m}, f_{m}^2, and f_{m}^3.

Twierdzenie Telgarskiego

Twierdzenie:

Niech $x_i = i/2^k$, $y_i = m^{(k)}(x_i)$ i $y_i \in \{0,1\}$. Dla zadanej funkcji F i zbioru punktów (x_i, y_i) zdefiniujmy błąd klasyfikacji jako:

$$R(F) = \frac{1}{n}\sum_i \mathbf{1}\{\tilde{F}(x) \neq y_i\},$$

gdzie $\tilde{F}(x) = \mathbf{1}\{F(x_i) \geq 1/2\}$ to funkcja klasyfikująca. Dla prezentowanej wcześniej sieci neuronowej $f(x)$ o 2^k warstwach ukrytych i 2 dwóch neuronach na każdej z warstw błąd klasyfikacji jest zawsze zerowy:

$$R(f) = 0$$

Dla dowolnej sieci neuronowej $g(x)$ o l warstwach i szerokości każdej z warstw $w < 2^{(n-k)/l\,-1}$ dolne oszacowanie błędu wynosi:

$$R(g) > \frac{1}{2} - \frac{1}{3 * 2^{k-1}}$$

Twierdzenie Montufara

- **Montúfar G.F., Pascanu R., Cho K., and Bengio, Y., On the number of linear regions of deep neural networks, in: NIPS'2014, 2014**
- Twierdzenie Montufara omawia bardziej ogólny przypadek, sieć neuronowa z przedziałami liniową funkcją aktywacji (np. ReLU) może aproksymować dowolną funkcje z dokładnością rosnącą wykładniczo wraz ze wzrostem głębokości sieci ℓ.

Twierdzenie Montufara

- Dzieje się tak dlatego, że głęboka sieć neuronowa jest zdolna do identyfikowania obszarów funkcji posiadających taki sam przebieg i mapowania ich do tego samego neuronu na kolejnej warstwie. Formalnie:

Definition 1. A map F *identifies* two neighborhoods S and T of its input domain if it maps them to a common subset $F(S) = F(T)$ of its output domain. In this case we also say that S and T are *identified* by F.

Twierdzenie Montufara

- Intuicyjnie można to porównać do składania funkcji narysowanej na kartce papieru wzdłuż jej osi symetrii – co ważne symetria może dotyczyć jedynie pewnych obszarów funkcji a nie jej całości.

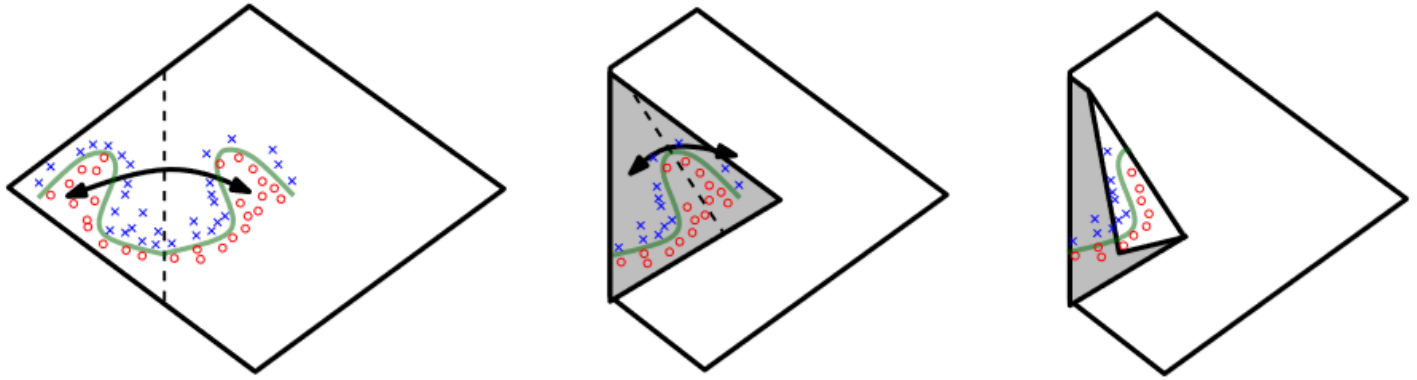


Figure 3: Space folding of 2-D space in a non-trivial way. Note how the folding can potentially identify symmetries in the boundary that it needs to learn.

Twierdzenie Montufara

- Oznacza to, że głęboka sieć neuronowa z wektorem danych wejściowych o długości d, głębokością ℓi n neuronami na warstwie ukrytej może wyaproksymować następującą liczbę liniowych regionów:

$$O\left(\binom{n}{d}^{d(\ell-1)} n^d\right)$$

Twierdzenie Montufara

- W rezultacie efektywność aproksymacji rośnie wraz z liczbą warstw ukrytych, różnica jest szczególnie widoczna w porównaniu z siecią z tylko jedną warstwą ukrytą.
- Sieć z jedną warstwą ukrytą musi wykorzystać pewną ilość neuronów do aproksymacji każdego fragmentu funkcji – nawet takich, których przebieg jest dokładnie taki sam (co do osi symetrii).

Twierdzenie Montufara

- W przypadku sieci głębokich symetryczne fragmenty są mapowane do tego samego neuronu na kolejnej warstwie.
- Dzięki temu każdą kolejna warstwa aproksymuje mniejszy kawałek funkcji z większą precyzją.
- Prowadzi to też do ważnego wniosku na temat aproksymowanej funkcji – im bardziej jest ona symetryczna tym większa jest dokładność jaką możemy uzyskać za pomocą naszego modelu.

Dodatkowa praca domowa

- Wykaż, że algorytm propagacji wstecznej poprawnie wyznacza błąd na każdym z neuronów sieci neuronowej (**5 punktów**).