Uczenie sieci neuronowych

Mariusz Zalewski

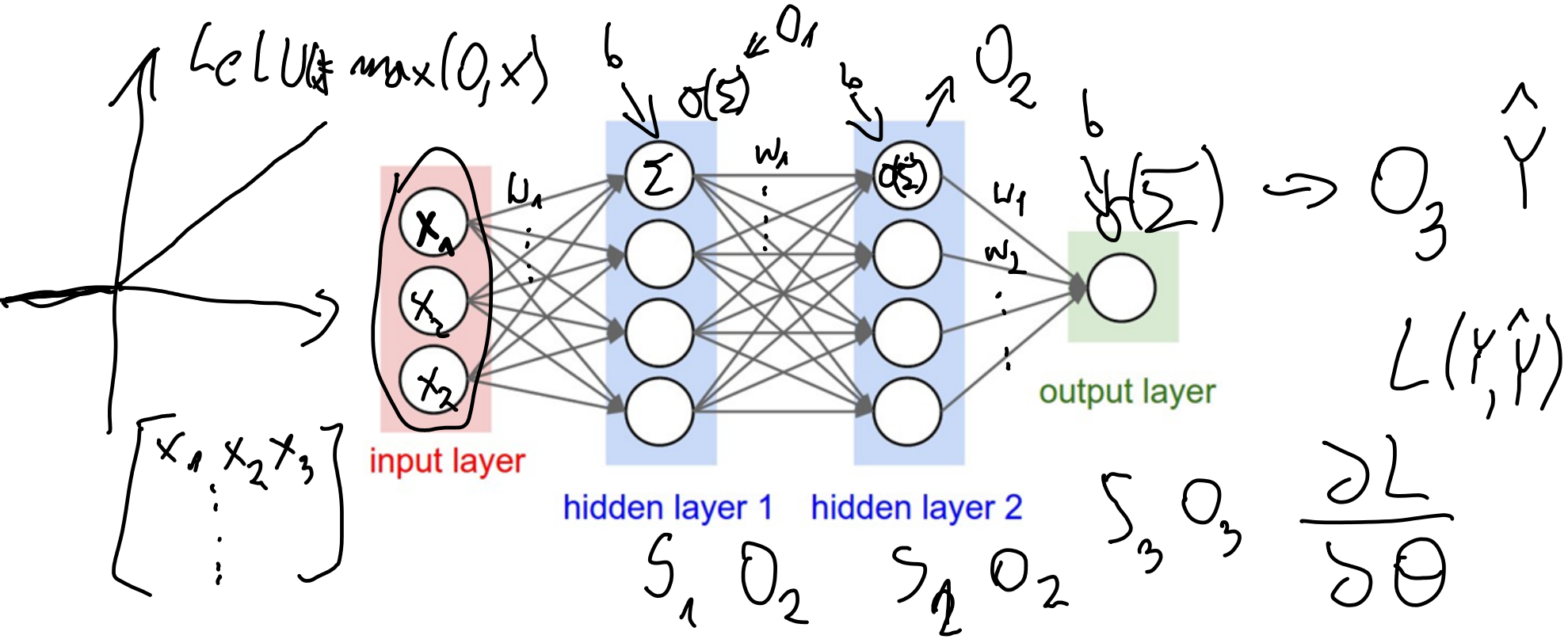
Uniwersytet Wrocławski
Wydział Matematyki i Informatyki

14 kwietnia 2021

Plan prezentacji

- Uczenie sieci neuronowych
  - Idea
  - Reguła łańcuchowa
  - Automatyczne różniczkowanie
  - Przykład – uczenie sieci rozwiązującej problem XOR
  - Weight decay – regularyzacja L_2
- Stochastic gradient descent (SGD)
- Momentum i przyspieszenie Nesterova

Uczenie sieci neuronowych – idea



1. $\theta^{(0)}$ - parametry startowe, przeważnie losowe (np z $N(0,\ 0.5)$)
2. Forward pass
3. Backward pass
4. $\theta^{(i+1)} = \theta^{(i)} - \alpha \frac{\partial L}{\partial \theta^{(i)}}$, α – learning rate (długość kroku)

Uczenie sieci neuronowych – reguła łańcuchowa

Trudności? Duża złożoność
Rozwiązanie – **reguła łańcuchowa**

Twierdzenie (reguła łańcuchowa)

Jeśli funkcja $f : \mathbb{R} \to \mathbb{R}$ *jest zdefiniowana jako* $f(x) = (f_1 \circ ... \circ f_n)(x)$, *to jej pochodna ma postać* $f'(x) = \prod_{i=1}^{n}(f_i' \circ f_{i+1} \circ ... \circ f_n)(x)$.

Przykład

$$(f(g(x)))' = f'(g(x)) \cdot g'(x) = (f' \circ g)(x) \cdot g'(x)$$

Przykład (Notacja Leibniza)

W notacji Leibniza reguła łańcuchowa przypomina skracanie ułamków:

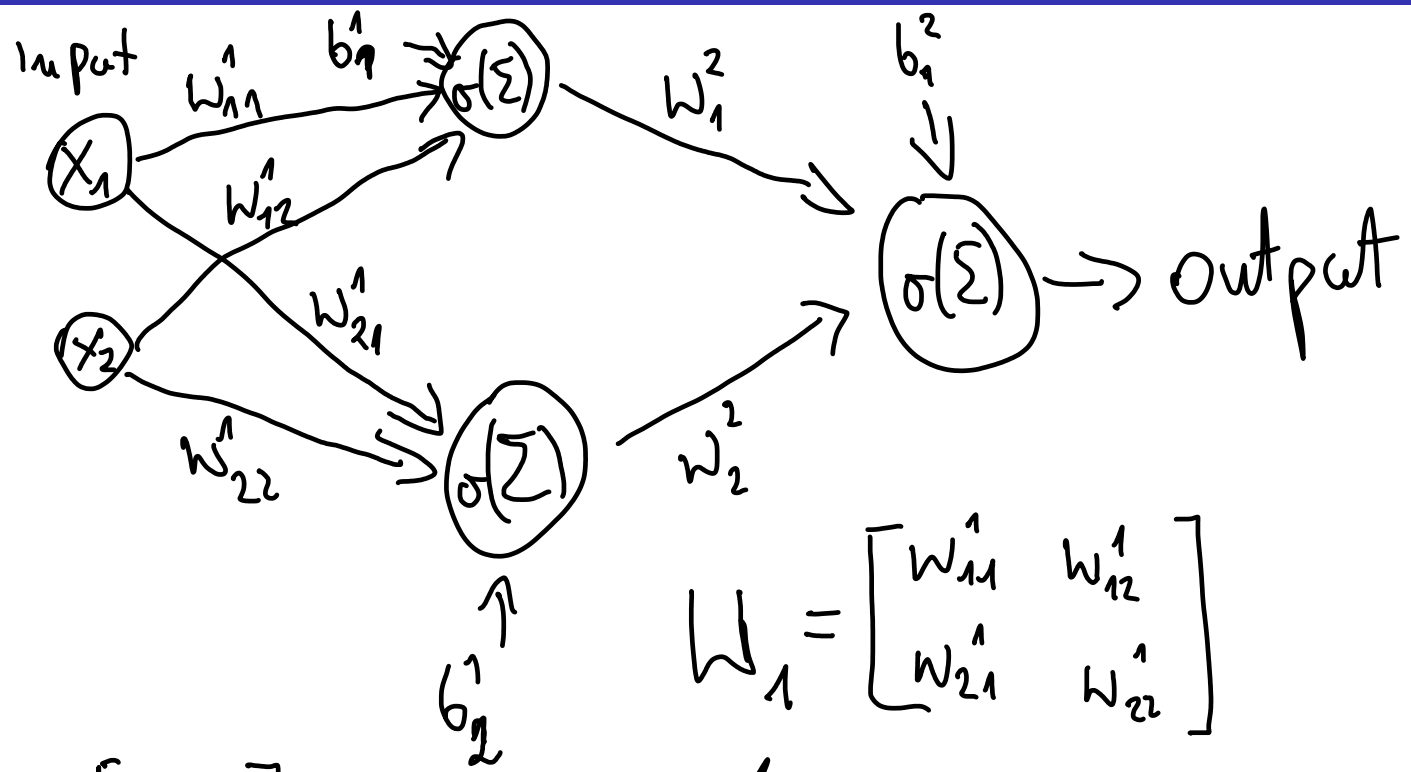
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$$

Uczenie sieci neuronowych – automatyczne różniczkowanie

Automatyczne różniczkowanie:

1. ***Forward pass*** – zapamiętujemy output'y każdej warstwy sieci.
2. ***Backward pass*** – obliczamy kolejne pochodne funkcji straty za pomocą *reguły łańcuchowej* oraz z wykorzystaniem zapamiętanych output'ów, używając postaci macierzowej wag oraz postaci wektorowej obciążeń dla każdej warstwy sieci.

Uczenie sieci neuronowych – przykład XOR



$$W_1 = \begin{bmatrix} w^1_{11} & w^1_{12} \\ w^1_{21} & w^1_{22} \end{bmatrix}$$

$$S_2 = O_1 W_2^T + b_2$$

$$O_2 = \sigma(S_2)$$

$$O_2 = \hat{Y}$$

$$X = \begin{bmatrix} x_{11} & x_{12} \\ \vdots & \\ x_{41} & x_{42} \end{bmatrix}$$

$$b^1 = \begin{bmatrix} b^1_1, b^1_2 \\ b^1_1, b^1_2 \\ \vdots \\ \vdots \end{bmatrix}$$

$$S_1 = X W_1^T + b^1$$

$$O_1 = \sigma(S_1)$$

$$L(x, y) = -y \log(O_2(x)) - (1-y) \log(1 - O_2(x))$$

$$\frac{\partial L}{\partial S_2} = O_2 - Y \qquad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

$$\frac{\partial L}{\partial O_1} = \frac{\partial L}{\partial S_2} \cdot \frac{\partial S_2}{\partial O_1} = (O_2 - Y) W_2$$

$$\frac{\partial L}{\partial S_1} = \frac{\partial L}{\partial O_1} \cdot \frac{\partial O_1}{\partial S_1} = (O_2 - Y) W_2 \odot \underbrace{\sigma(S_1)}_{O_1} (1 - \underbrace{\sigma(S_1)}_{O_1})$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial S_2} \cdot \frac{\partial S_2}{\partial b_2} = (O_2 - Y) \cdot 1 \mid$$

$$\frac{\partial L}{\partial W_2} = \left(\frac{\partial L}{\partial S_2}\right)^T \frac{\partial S_2}{\partial W_2} = (O_2 - Y)\, O_1$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial S_1} \frac{\partial S_1}{\partial b_1} = (O_2 - Y) W_2 \odot O_1 \odot (1 - O_1) \mid$$

$$\frac{\partial L}{\partial W_1} = \left(\frac{\partial L}{\partial S_1}\right)^T \frac{\partial S_1}{\partial W_1} = (O_2 - Y) W_2 \odot O_1 \odot (1 - O_1)\, X$$

Uczenie sieci neuronowych – weight decay

Podczas treningu sieci neuronowej może zajść zjawisko *przeuczenia*, tzn sieć może zacząć dopasowywać się do szumu. Aby tego uniknąć, stosuje się tak zwane metody regularyzacji. Jedną z podstawowych, szeroko używanych metod regularyzacji w uczeniu sieci neuronowych jest metoda ***weight decay***. Polega ona na "karaniu" poprzez dodawanie do funkcji straty kwadratu normy L_2 parametrów sieci, przemnożonej przez parametr λ.

$$\underbrace{L(x, y; \Theta) + \frac{\lambda}{2}||\Theta||^2}$$

Stochastic gradient descent (SGD)

1. $\theta^{(0)}$ - losujemy parametry startowe (np z $N(0,\ 0.5)$)
2. Permutacja zbioru treningowego - tasowanie (shuffling)
3. Dzielimy dane na podzbiory (minibatches)
4. Na każdym podzbiorze wykonujemy jedną iterację GD (forward i backward pass)

Kroki 2–4 stanowią jedną epokę. W procesie uczenia sieci wykonywanych jest wiele epok.

Jeden podzbiór może być nawet jedną obserwacją.

SGD działa szybciej na dużych zbiorach danych.

Większy szum, który jest powodowany przez zmniejszenie zbioru danych, jest bardzo ważny. Pomaga on lepiej dopasować model do nieznanych danych.

SGD – dlaczego tak można?

Chcemy minimalizować wartość oczekiwaną straty na rozkładzie pełnych danych:

$$\mathbb{E}_{x,y}\Big[\ell(F(x,\theta),y)\Big]$$

W tym celu liczymy pochodną:

$$\frac{\partial \mathbb{E}_{x,y}\Big[\ell(F(x,\theta),y)\Big]}{\partial\theta} = \mathbb{E}_{x,y}\left[\frac{\partial \ell(F(x,\theta),y)}{\partial\theta}\right]$$

Możemy ją aproksymować w następujący sposób:

$$\mathbb{E}_{x,y}\left[\frac{\partial \ell(F(x,\theta),y)}{\partial\theta}\right] \approx \frac{1}{N_B}\sum_{x,y\in X_B,Y_B}\frac{\partial \ell(F(x,\theta),y)}{\partial\theta} = \frac{\partial L(X_B,Y_B;\theta)}{\partial\theta}$$

Zatem:

$$\frac{\partial \mathbb{E}_{x,y}\Big[\ell(F(x,\theta),y)\Big]}{\partial\theta} = \frac{\partial L(X_B,Y_B;\theta)}{\partial\theta} + \varepsilon_{X_B,Y_B}$$

SGD – learning rate

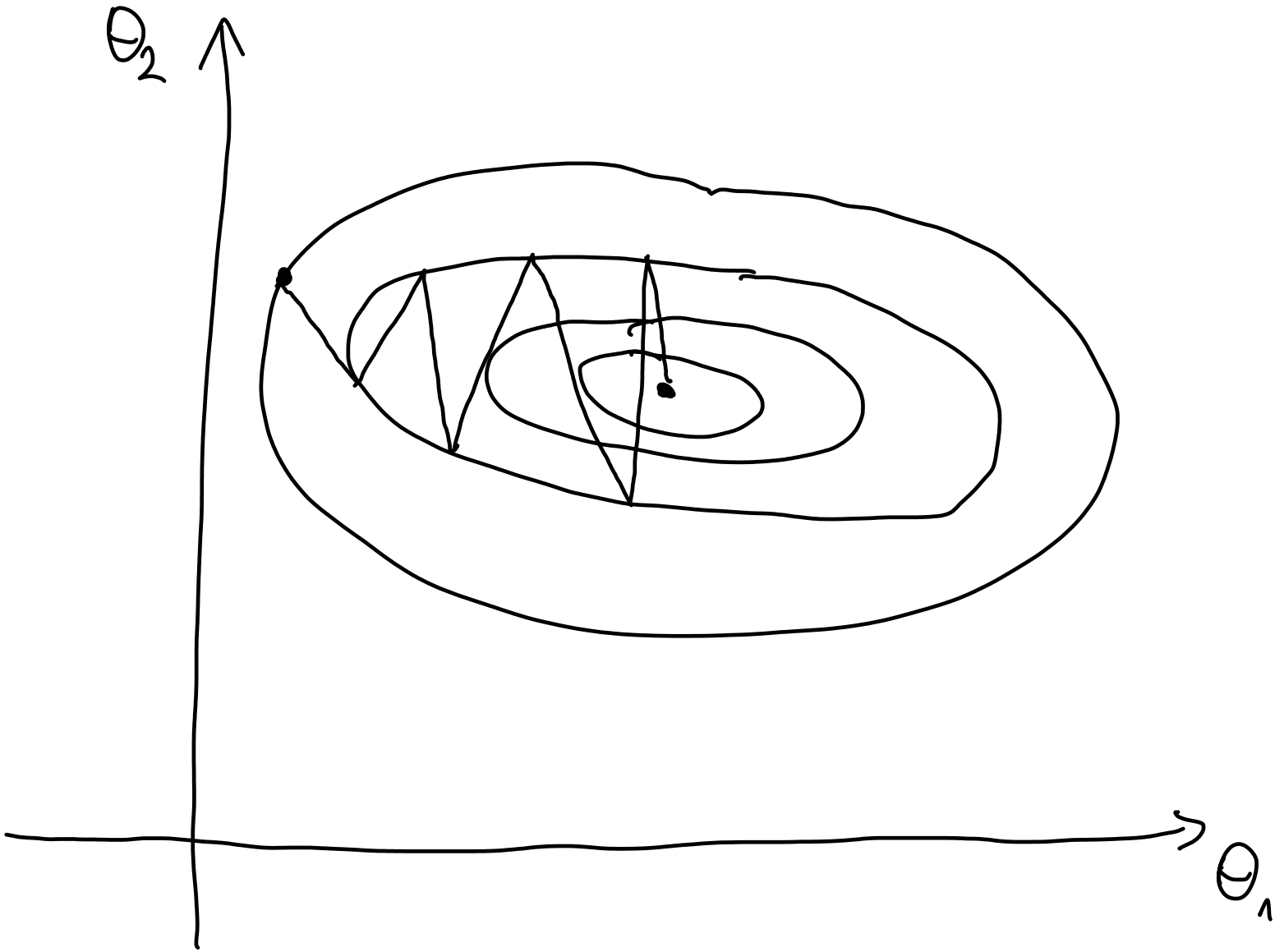
Długość kroku w SGD powinna maleć w kolejnych epokach.

$$\sum_t \alpha_t = \infty$$

$$\sum_t \alpha_t^2 < \infty$$

$$\frac{\alpha_0}{\text{num\_epoch}}$$

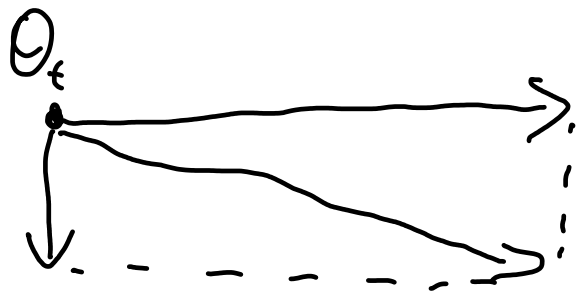
Przyspieszanie w dążeniu do minimum funkcji straty



Momentum (Polyak, 1964)

$$V^{(t)} = \mu_t V^{(t-1)} - \alpha_t \frac{\partial L}{\partial \theta}$$

$$\theta^{(t)} = \theta^{(t-1)} + V^{(t)}$$



Przyspieszenie Nesterova (1983)

$$V^{(t)} = \mu_t V^{(t-1)} - \alpha_t \frac{\partial L}{\partial(\theta + \mu_t V^{(t-1)})}$$

$$\theta^{(t)} = \theta^{(t-1)} + V^{(t)}$$