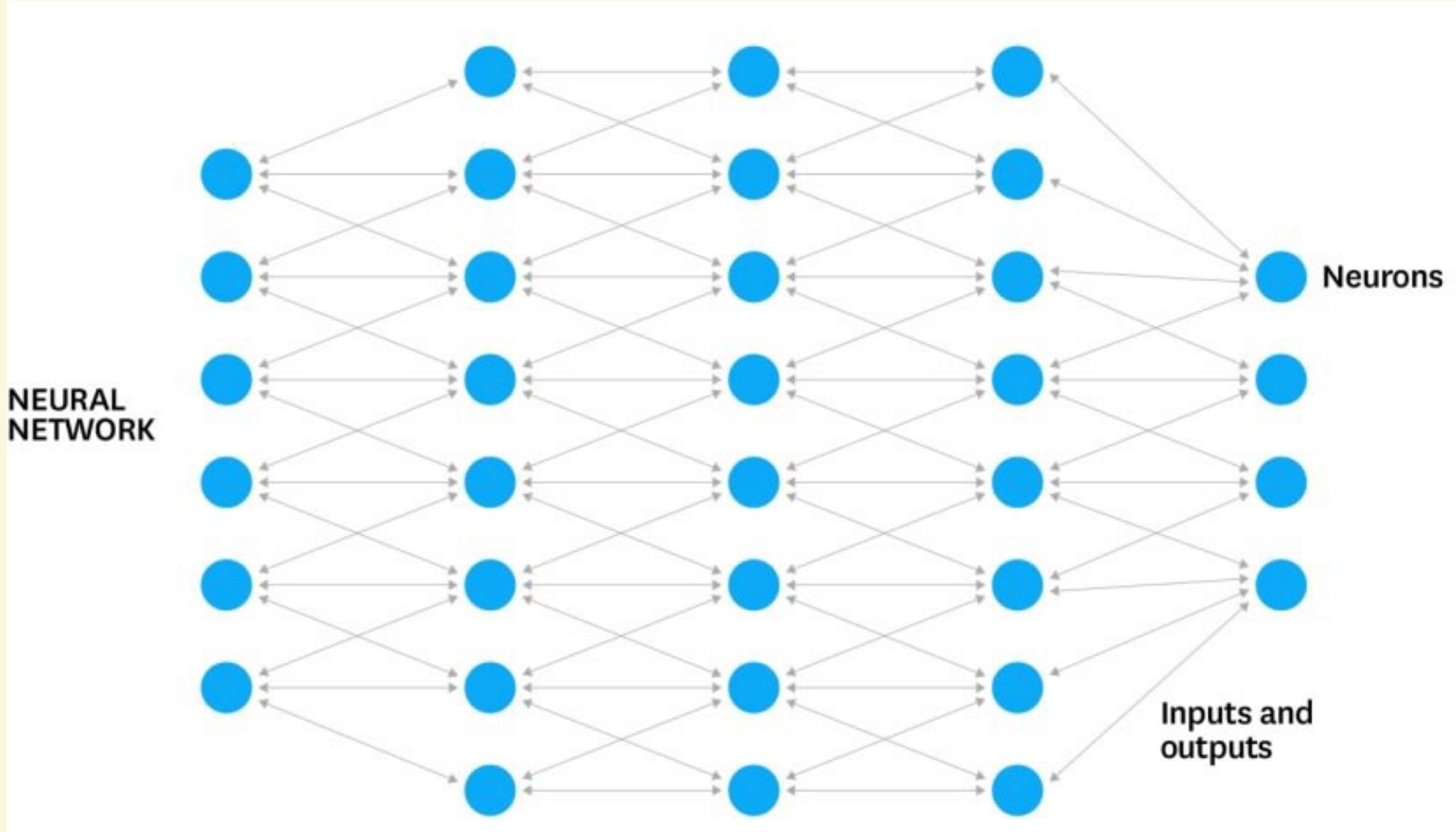
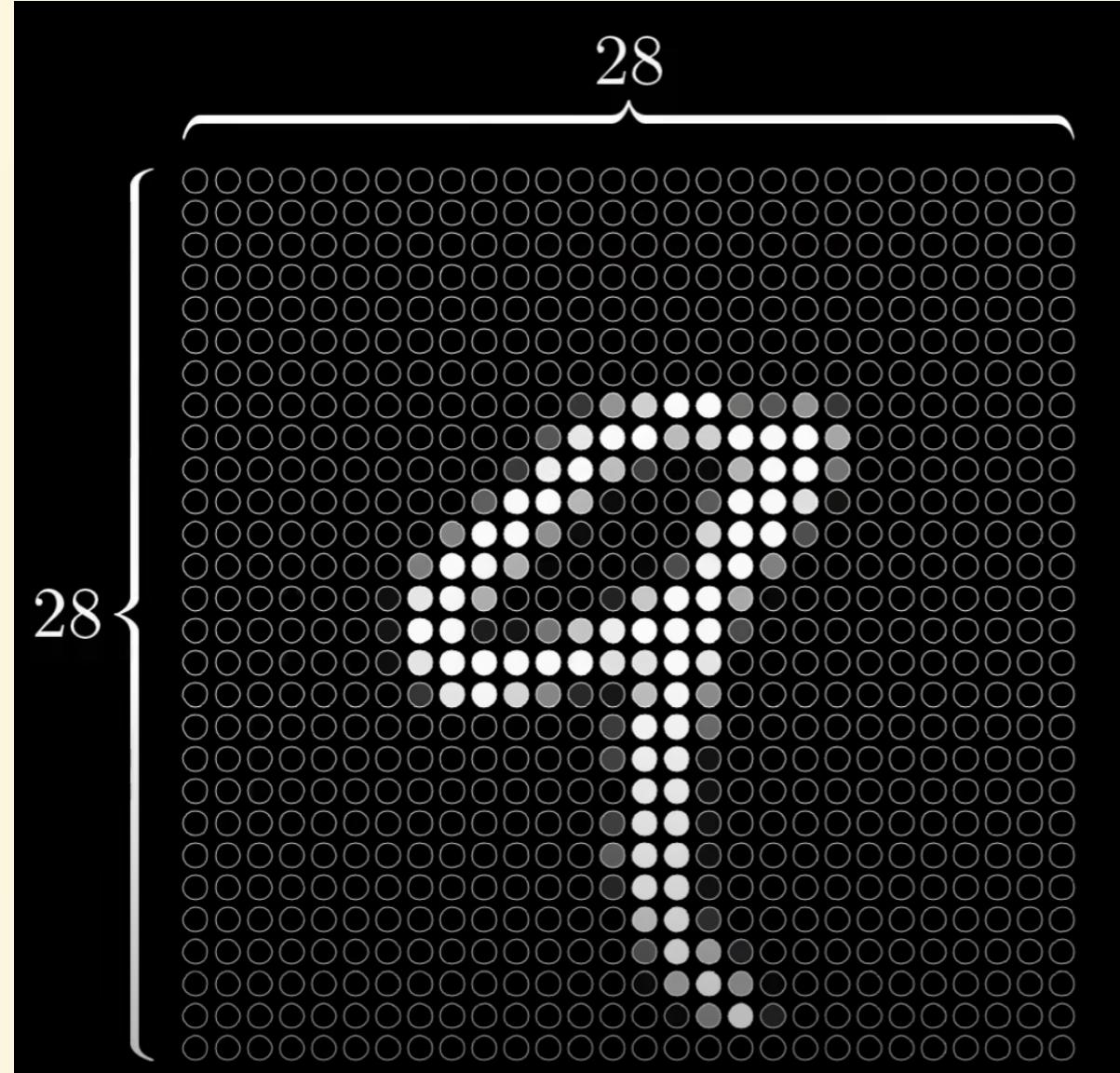


Deep Learning

Algorytmy wykorzystujące sieci neuronowe składające się z więcej niż trzech warstw. Znajdują swoje zastosowanie w między innymi rozpoznawaniu obrazów, mowy, procesowaniu języka naturalnego.



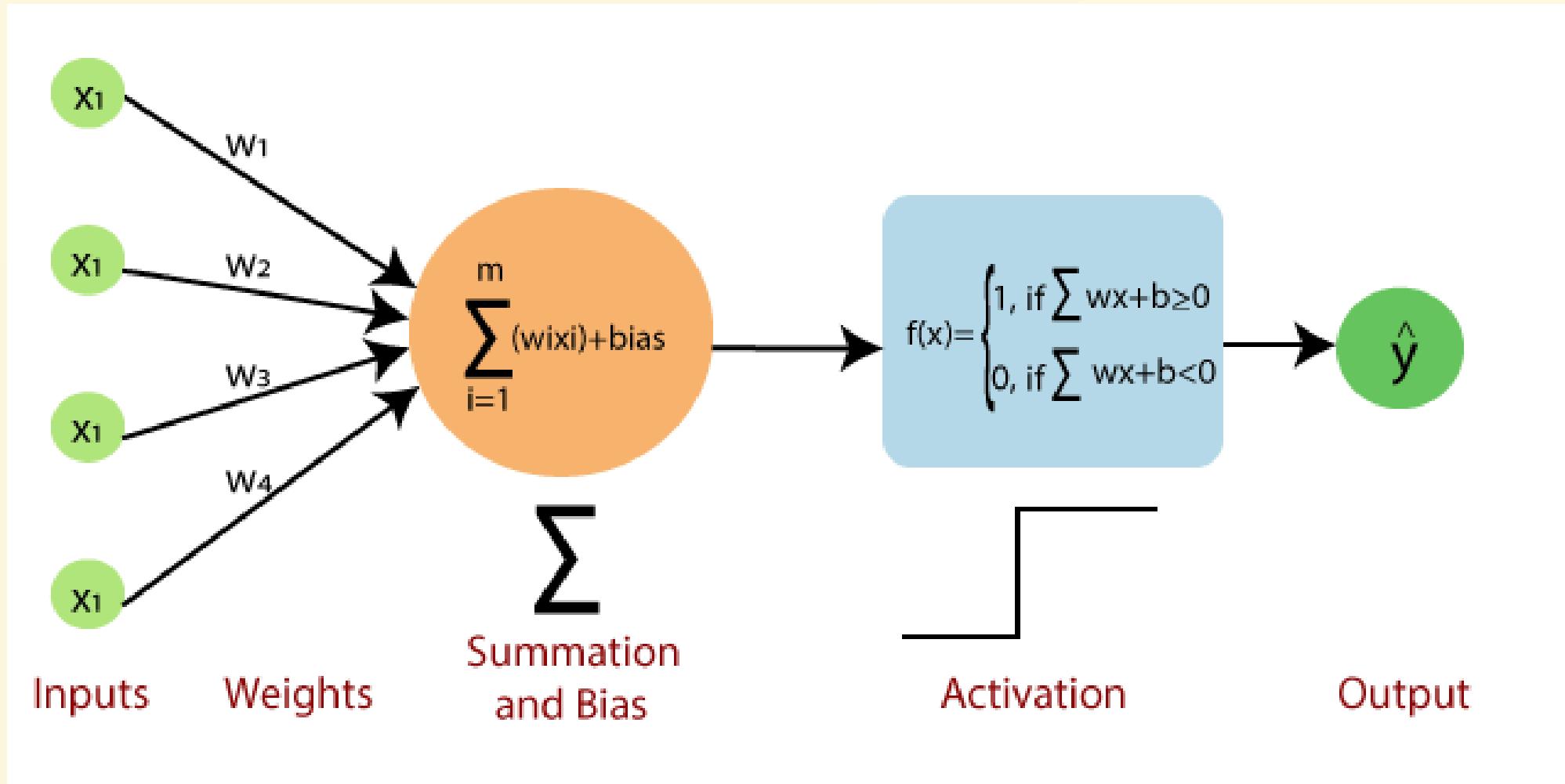
Sieci deep learningowe, mogą mieć wiele ukrytych warstw. Można sobie wyobrazić, że dane wejściowe są w najprostszym poziomie reprezentacji, a następne warstwy zwiększą poziomy abstrakcji, aż w warstwie wyjściowej mamy najwyższy poziom abstrakcji, którym jest odpowiedź na zadane pytanie.



$$\begin{aligned} \boxed{\text{O}} &= \boxed{\text{I}} + \boxed{\text{J}} + \boxed{\text{K}} + \boxed{\text{L}} + \boxed{\text{M}} \\ \boxed{\text{I}} &= \boxed{\text{N}} + \boxed{\text{O}} + \boxed{\text{P}} \end{aligned}$$

$$\begin{array}{c} \text{9} \\ = \\ \text{9} + \text{l} \end{array}$$
$$\begin{array}{c} \text{8} \\ = \\ \text{8} + \text{o} \end{array}$$
$$\begin{array}{c} \text{4} \\ = \\ \text{4} + \text{l} + \text{f} + \text{h} \end{array}$$

Działanie neuronu



Neuron przechowuje w sobie wartość zwaną aktywacją, która może przyjąć wartość $< 0, 1 >$. Wartość ta jest obliczana następująco:

- Bierzemy aktywacje z poprzedniej warstwy: a_1, a_2, \dots, a_n
- Bierzemy wagi związane z tymi aktywacjami: w_1, w_2, \dots, w_n
- Przenożamy wagi przez aktywacje oraz sumujemy je
 $\sum_{i=1}^n w_i x_i$
- Dodajemy bias, czyli to jak łatwo lub ciężko aktywować neuron
$$z = \sum_{i=1}^n w_i x_i + b$$
- Abytrzymać wartość z przedziału $< 0, 1 >$ przepuszczamy z przez funkcję aktywacji

Neuron przechowuje w sobie wartość $< 0, 1 >$, ale można myśleć o nim jako funkcji, Gjjktóra przyjmuje n wartości i dla nich zwraca wartość $< 0, 1 >$

Wiele neuronów

$L = 0$ $L = 1$

$$o_0^{(1)} = o_0^{(0)} w_{00} + o_1^{(0)} w_{01} + o_2^{(0)} w_{02} + b_0$$
$$o_1^{(1)} = o_0^{(0)} w_{10} + o_1^{(0)} w_{11} + o_2^{(0)} w_{12} + b_1$$
$$G \begin{pmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \end{pmatrix} \begin{pmatrix} o_0^{(0)} \\ o_1^{(0)} \\ o_2^{(0)} \end{pmatrix} + \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} o_0^{(1)} \\ o_1^{(1)} \end{pmatrix}$$

$$f_{\text{activate}} \left(\begin{pmatrix} w_{0,0}^{(L)} & w_{0,1}^{(L)} & \dots & w_{0,n}^{(L)} \\ w_{1,0}^{(L)} & w_{1,1}^{(L)} & \dots & w_{1,n}^{(L)} \\ \dots & \dots & \dots & \dots \\ w_{k,0}^{(L)} & w_{k,1}^{(L)} & \dots & w_{k,n}^{(L)} \end{pmatrix} \begin{pmatrix} a_0^{(L-1)} \\ a_1^{(L-1)} \\ \dots \\ a_n^{(L-1)} \end{pmatrix} + \begin{pmatrix} b_0^{(L)} \\ b_1^{(L)} \\ \dots \\ b_k^{(L)} \end{pmatrix} \right) = \begin{pmatrix} a_0^{(L)} \\ a_1^{(L)} \\ \dots \\ a_k^{(L)} \end{pmatrix}$$

n - liczba neuronów w warstwie poprzedniej

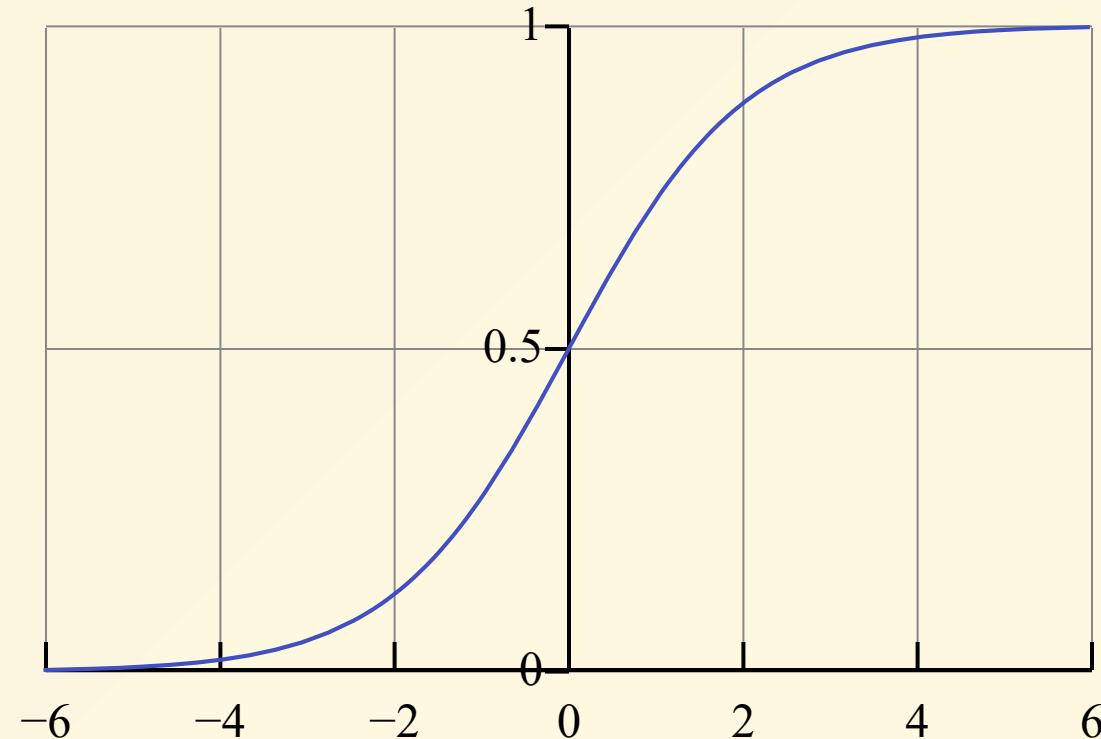
k - liczba neuronów w warstwie następnej

```
z = numpy.dot(weights[L], activations[L-1]) + biases[L]
current_activations = activation_func(z)
activations.append(current_activations)
```

Funckje aktywacji

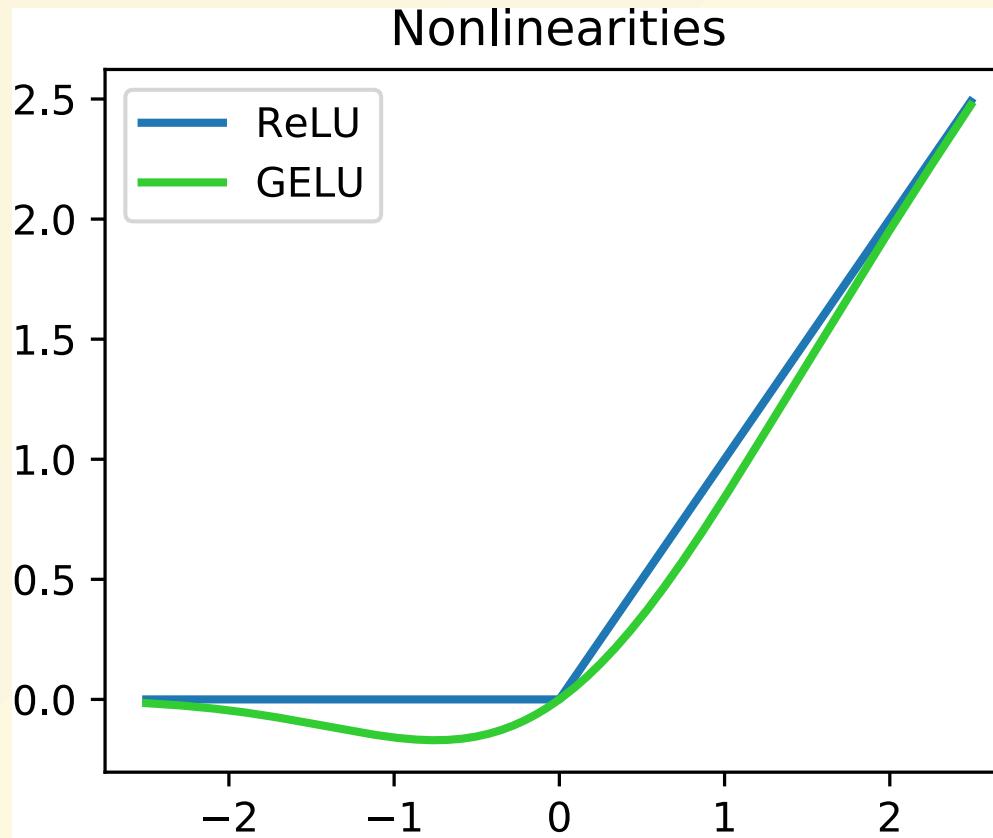
Sigmoid

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



RELU

Rectified Linear Unit $f(x) = \max(0, x)$



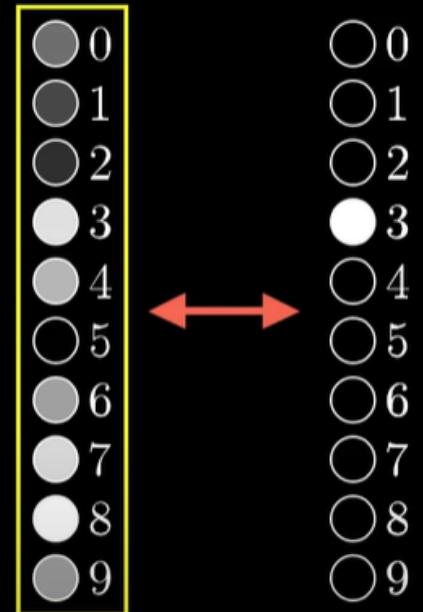
Funkcja kosztu

Cost of $\boxed{3}$

3.37

$$\left\{ \begin{array}{l} 0.1863 \leftarrow (0.43 - 0.00)^2 + \\ 0.0809 \leftarrow (0.28 - 0.00)^2 + \\ 0.0357 \leftarrow (0.19 - 0.00)^2 + \\ 0.0138 \leftarrow (0.88 - 1.00)^2 + \\ 0.5242 \leftarrow (0.72 - 0.00)^2 + \\ 0.0001 \leftarrow (0.01 - 0.00)^2 + \\ 0.4079 \leftarrow (0.64 - 0.00)^2 + \\ 0.7388 \leftarrow (0.86 - 0.00)^2 + \\ 0.9817 \leftarrow (0.99 - 0.00)^2 + \\ 0.3998 \leftarrow (0.63 - 0.00)^2 \end{array} \right.$$

What's the “cost”
of this difference?

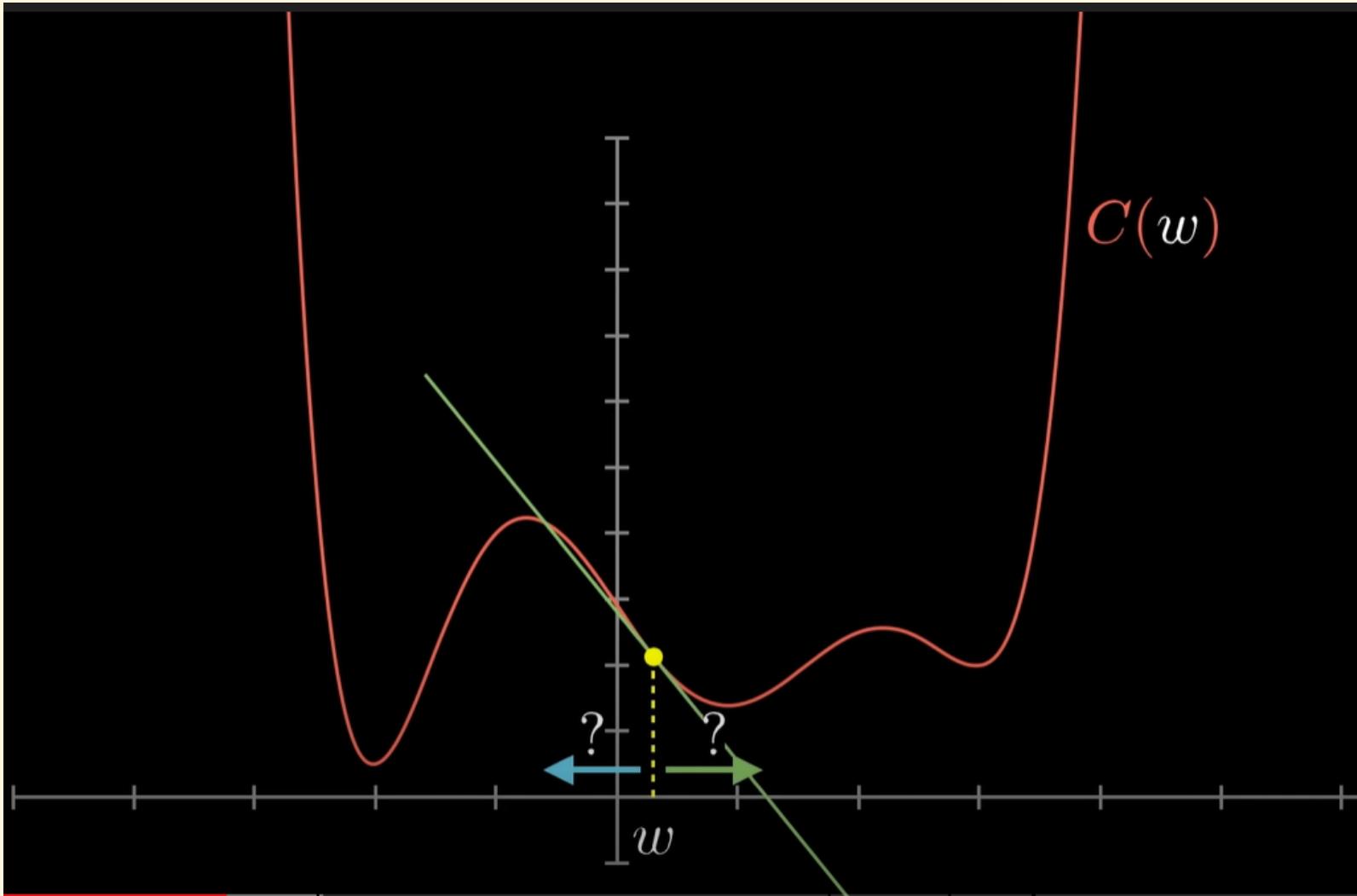


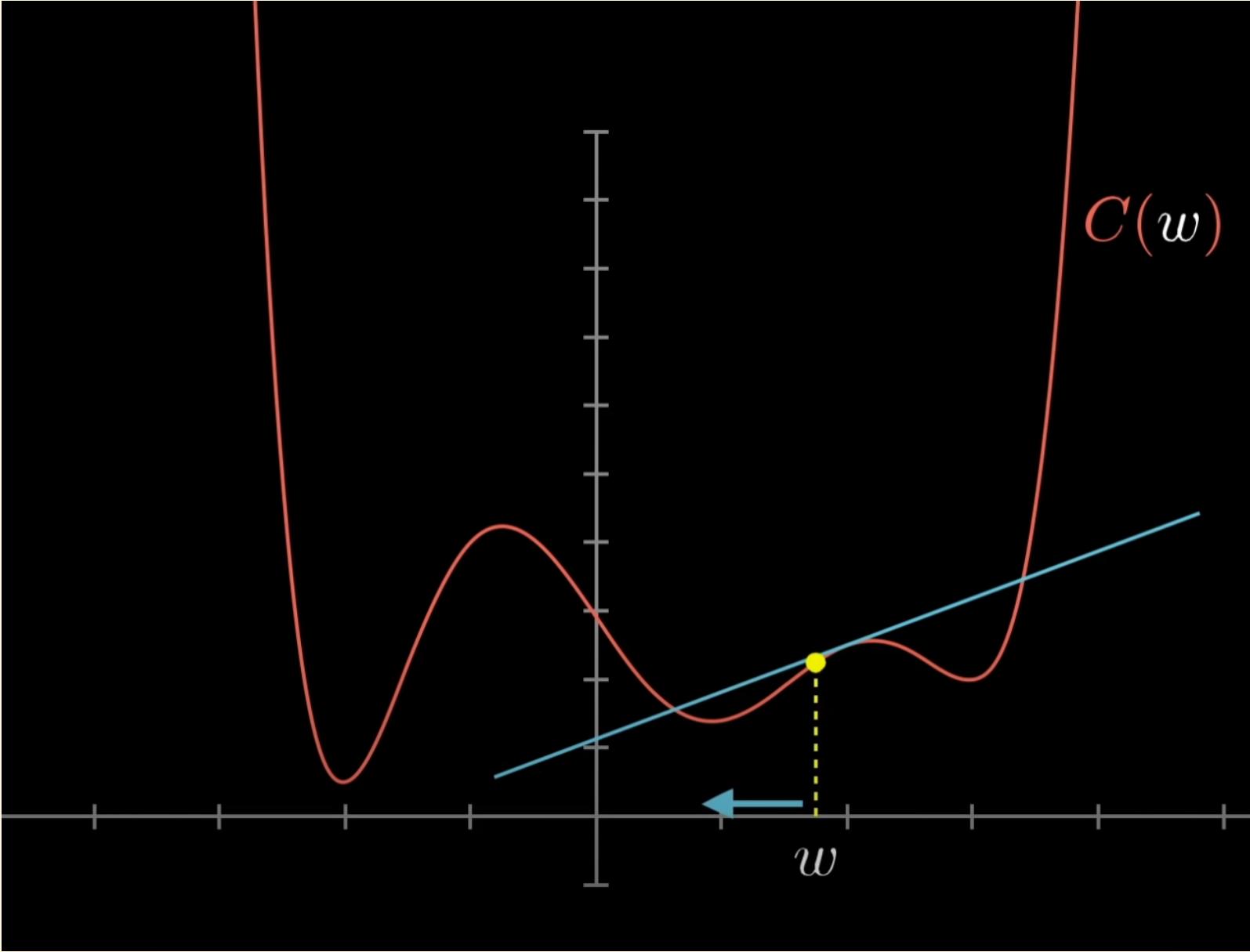
Utter trash

$$C = \sum_{i=0}^{n-1} (a_i - y_i)^2$$

Jeżeli wartość kosztu jest duża, oznacza to, że sieć dla podanych wejść nie jest w stanie zwrócić poprawnej odpowiedzi. Aby nauczyć sieć odpowiadać poprawnie dla danych wejściowych trzeba skonfigurować odpowiednio jej wagi i biasy. Konfiguracja ta odbywa się za pomocą minimalizacji funkcji kosztu. Funkcja kosztu przyjmuje jako parametry wagi i biasy, a zwraca koszt.

Pochodna



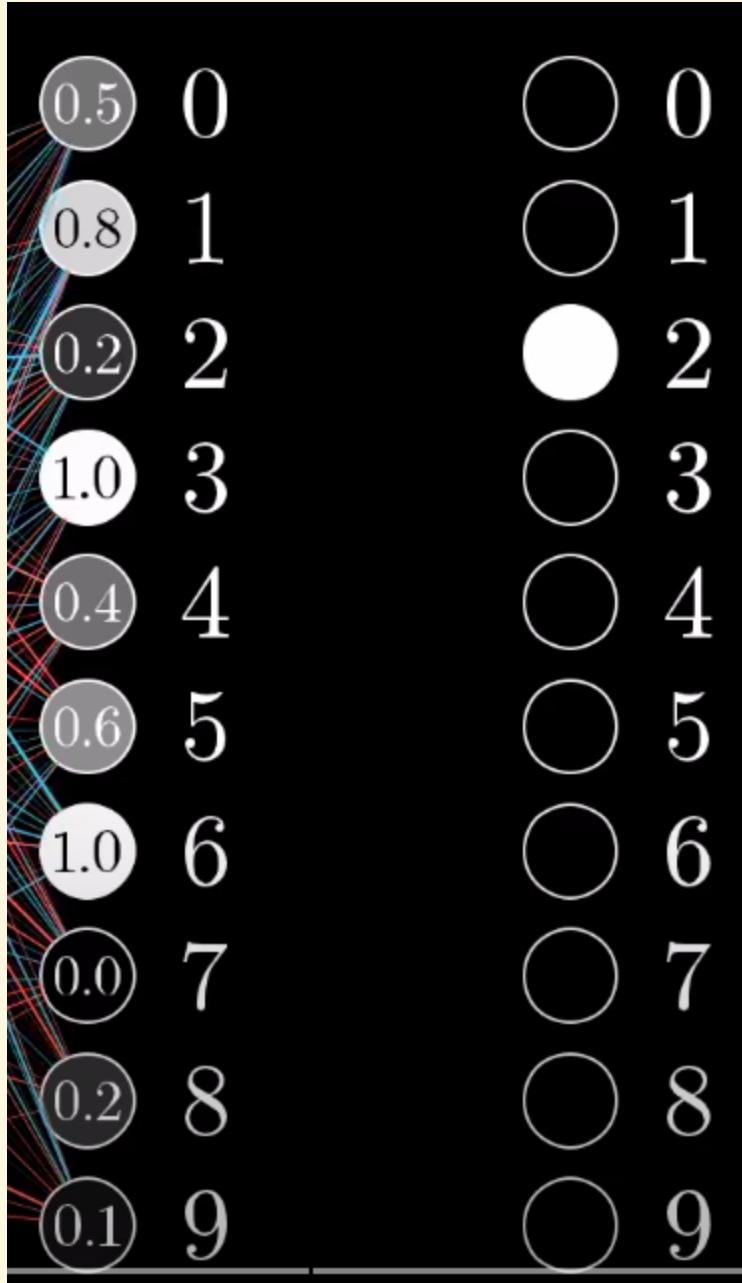


Minimalizacja funkcji jednej zmiennej

1. liczymy pochodną tej funkcji
2. przyjmujemy losową pozycję (wybieramy jakiegoś x)
3. liczymy nachylenie funkcji poprzez podstawienie wartości x do pochodnej
4. jeśli wartość jest dodatnia to musimy przesunąć x w lewo
5. jeśli wartość jest ujemna to musimy przesunąć x w prawo
6. powtarzamy punkty 3-5 do momentu aż nachylenie będzie odpowiednio małe (można sobie założyć jakąś zadawalającą wartość)

Backpropagation

Celem sieci deep learningowej jest osiągnięcie jak najmniejszego kosztu dla danych testowych. Może to osiągnąć za pomocą konfiguracji swoich parametrów, czyli wag i biasów. W algorytmie backpropagation liczymy gradient funkcji kosztu, który powie nam jak dane parametry wpływają na sieć - będziemy wtedy wiedzieli jak je zmodyfikować.



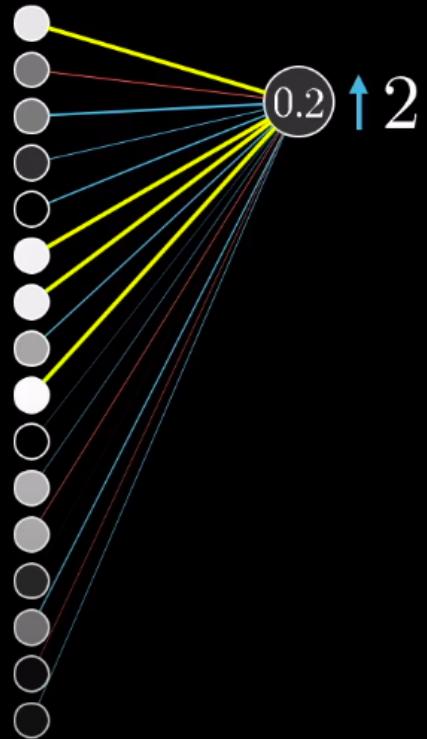


$$_{(0.2)} = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Increase b

Increase w_i

Change a_i



							...	Average over all training data
w_0	-0.08	+0.02	-0.02	+0.11	-0.05	-0.14	...	→ -0.08
w_1	-0.11	+0.11	+0.07	+0.02	+0.09	+0.05	...	→ +0.12
w_2	-0.07	-0.04	-0.01	+0.02	+0.13	-0.15	...	→ -0.06
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$w_{13,001}$	+0.13	+0.08	-0.06	-0.09	-0.02	+0.04	...	→ +0.04

Aby policzyć gradient funkcji kosztu musimy policzyć jak bardzo wpływają na nią wagi i biasy. Jak to zrobić?

$$C(w_1, b_1, \textcolor{violet}{w}_2, b_2, \textcolor{teal}{w}_3, b_3)$$



$$\frac{\partial C}{\partial w^{(L)}}$$

$$\frac{\partial C}{\partial b^{(L)}}$$

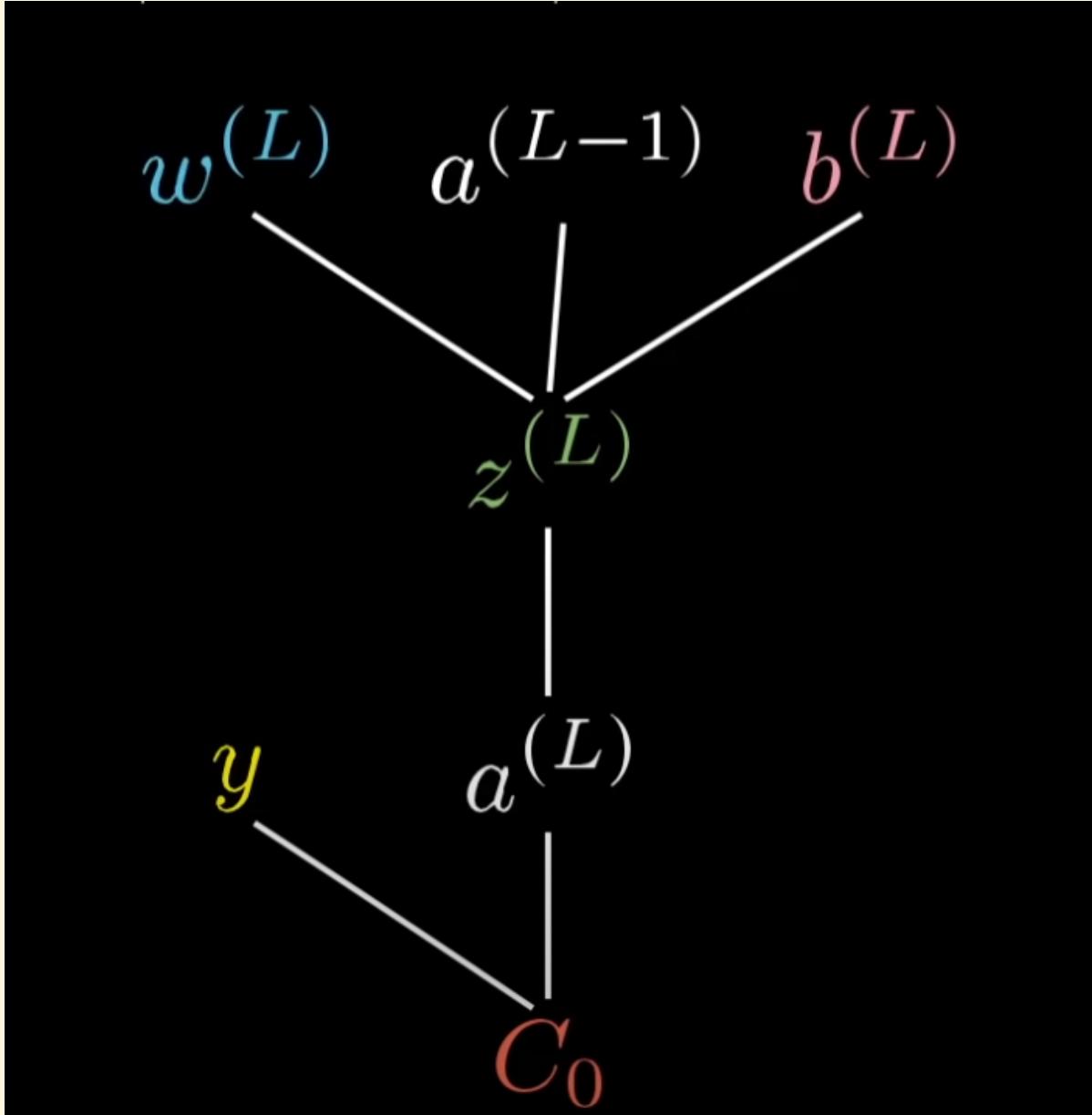
$$\frac{\partial C}{\partial a^{(L-1)}}$$

$$C = (a^{(3)} - y)^2$$

$$a^{(3)} = f_{activate}(w^{(3)}a^{(2)} + b^{(3)})$$

$$z^{(3)} = w^{(3)}a^{(2)} + b^{(3)}$$

$$a^{(3)} = f_{activate}(z^{(3)})$$



Chain rule

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C^{(L)}}{\partial a^{(L)}}$$

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C^{(L)}}{\partial a^{(L)}}$$

$$\frac{\partial C}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C^{(L)}}{\partial a^{(L)}}$$

$$C = (a^{(3)} - y)^2 = a^{2(3)} - 2a^{(3)}y + y^2$$

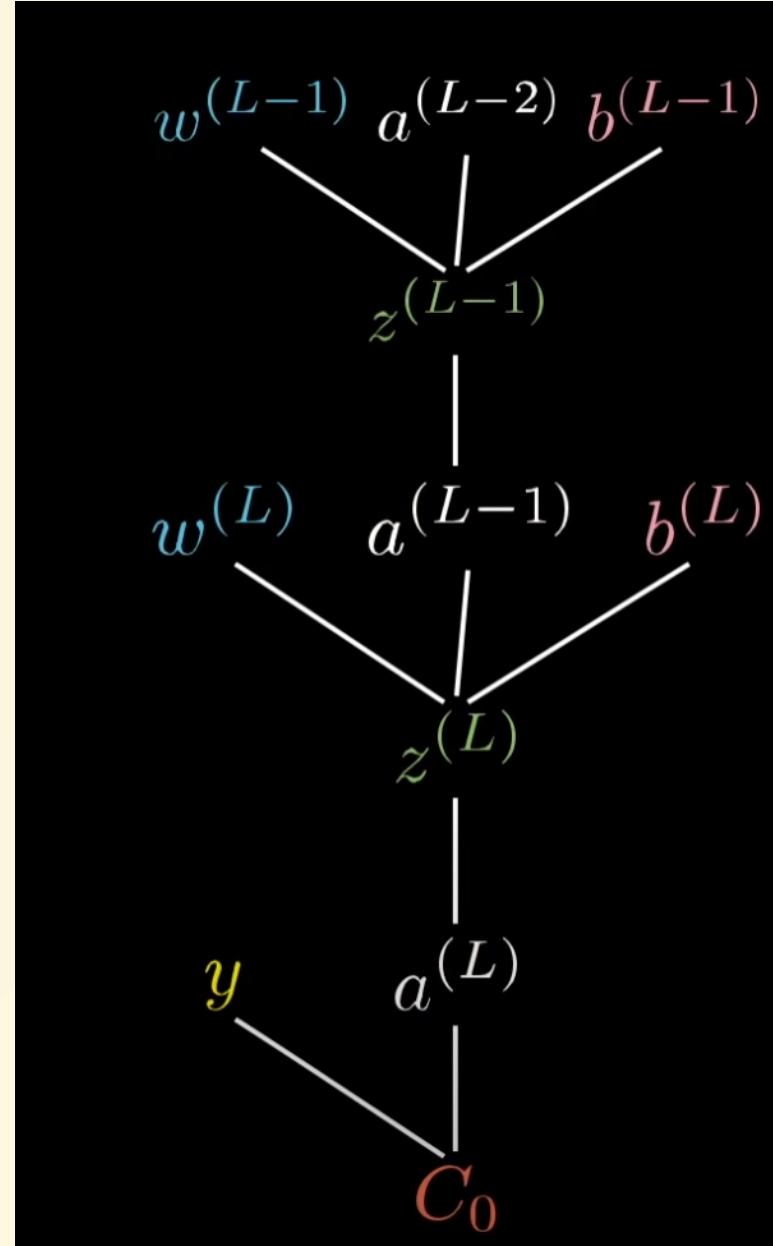
$$z^{(3)} = w^{(3)}a^{(2)} + b^{(2)}$$

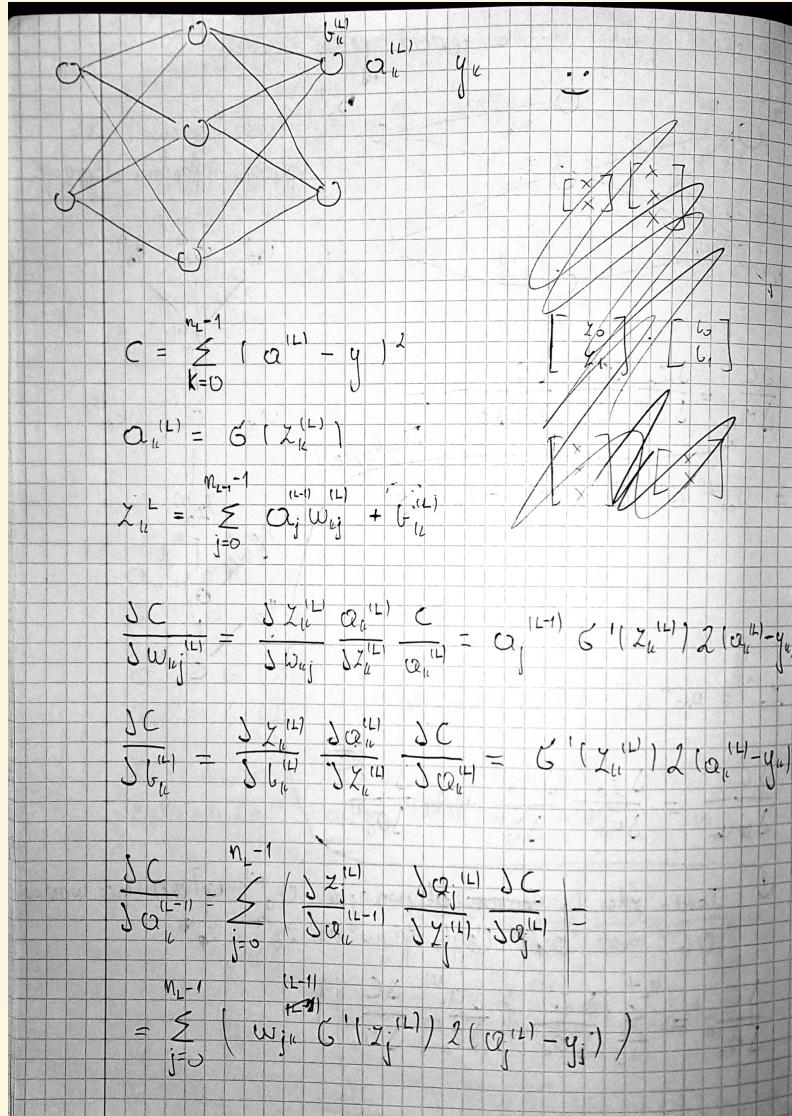
$$a^{(3)} = f_{activate}(z^{(3)})$$

$$\frac{\partial C}{\partial w^{(3)}} = \frac{\partial z^{(3)}}{\partial w^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} \frac{\partial C^{(3)}}{\partial a^{(3)}} = a^{(2)} f'_{activate}(z^{(3)}) 2(a^{(3)} - y)$$

$$\frac{\partial C}{\partial b^{(3)}} = \frac{\partial z^{(3)}}{\partial b^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} \frac{\partial C^{(3)}}{\partial a^{(3)}} = f'_{activate}(z^{(3)}) 2(a^{(3)} - y)$$

$$\frac{\partial C}{\partial a^{(2)}} = \frac{\partial z^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} \frac{\partial C^{(3)}}{\partial a^{(3)}} = w^{(3)} f'_{activate}(z^{(3)}) 2(a^{(3)} - y)$$





Źródło

Większość obrazów wykorzystanych w prezentacji to screenshoty z filmów stworzonych przez Granta Sandersona - 3Blue1Brown.

[https://www.youtube.com/watch?
v=aircAruvnKk&list=PLZHQBOWTQDNU6R1_67000Dx_ZCJB-3pi](https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQBOWTQDNU6R1_67000Dx_ZCJB-3pi)

<https://www.3blue1brown.com/topics/neural-networks>

Link do prezentacji

<https://github.com/Shady6/Prezentacja-Deep-Learning>