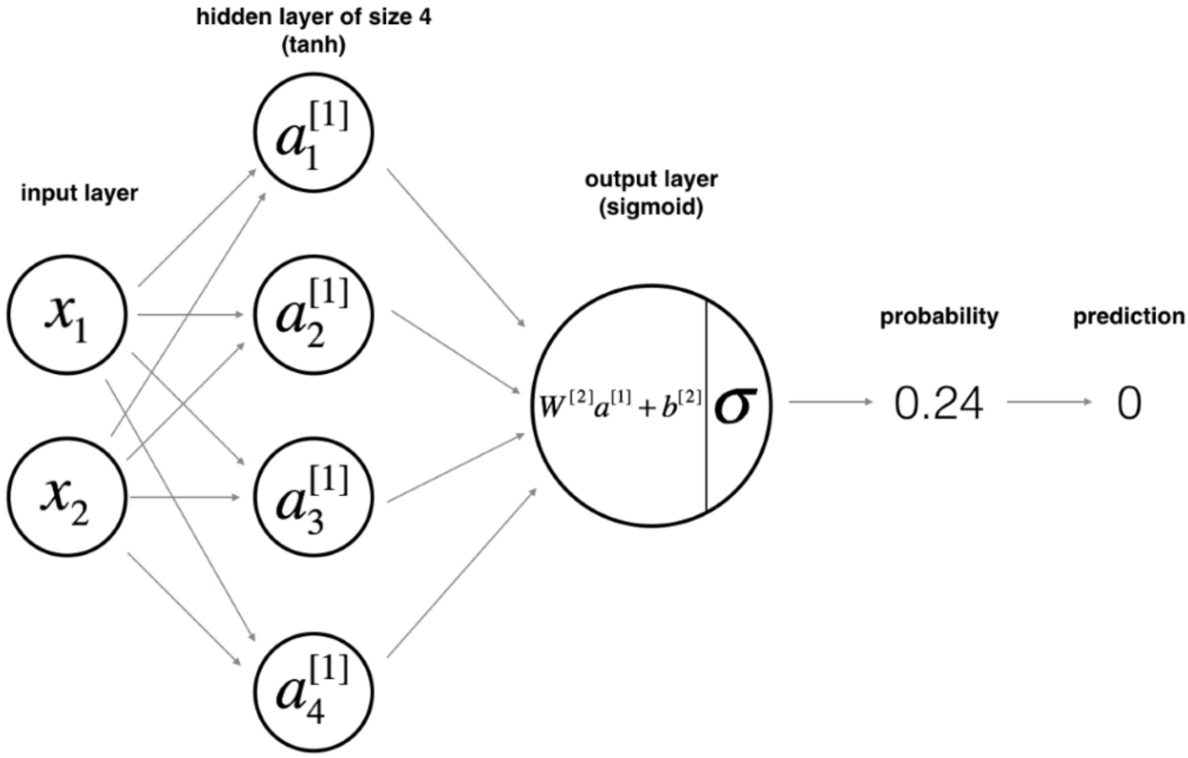
Sieci Neuronowe

- Sieć z pojedynczą warstwą ukrytą

Sieć Neuronowa - złożenie wielu "regresji logistycznych" (*sigmoidów / ang. Sigmoid unit*). Hidden *Layer* oznacza, że prawdziwe wartości dla węzłów wewnętrznych są nieznane w zbiorze treningowym.



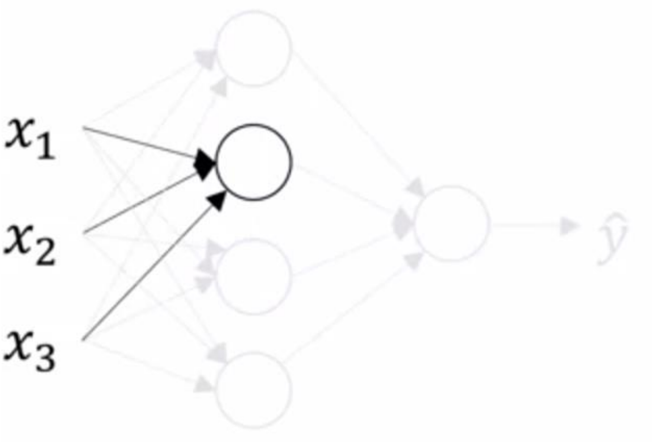
Sieć neuronowa o dwóch warstwach (nie liczymy *input layer*)

$a^{[0]} = x$ - input features (inaczej aktywacje czyli wartości, które warstwy przekazują do kolejnych warstw)

$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$ - aktywacje *hidden layer*

$a^{[2]} = \hat{y}$ - output layer

Wyliczanie wyniku sieci – Forward Propagation



Przykładowa aktywacja pojedynczego węzła (neuronu sieci) składa się z dwóch wyliczeń

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \; a_2^{[1]} = \sigma(z_2^{[1]})$$

$W^{[1]T}$ – macierz o wymiarach (4,3), gdyż mamy 4 neurony oraz 3 wartości wejściowe

$b^{[1]}$ – macierz o wymiarach (4,1)

Wektorowa reprezentacja wyliczenia wartości węzłów w sieci dla pojedynczej warstwy (*forward propagation*):

$$z^{[1]} = \begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}, \quad a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

- Jaka będzie reprezentacja dla warstwy wyjściowej?
- Powyższe obliczenia (w tym dla warstwy wyjściowej) należy wykonać w pętli dla każdej pary uczącej (wynik dla danej $i - tej$ pary oznaczamy $a^{[1](i)}$)

`for i = 1 to m:`

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

Wektoryzacja dla wszystkich par uczących

$$X = [x^{(1)} \quad \dots \quad x^{(m)}] - zbiór\ treningowy$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[2]} + b^{[2]}$$

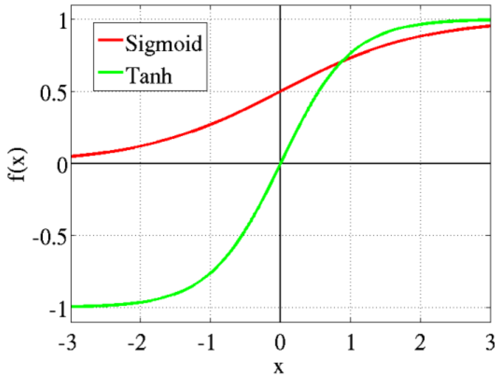
$$A^{[2]} = \sigma(Z^{[2]})$$

Pary uczące

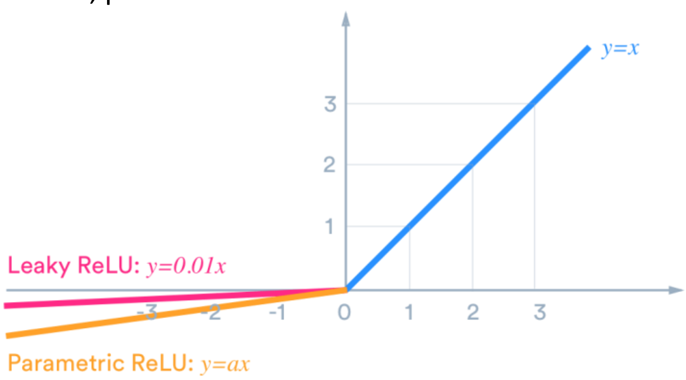
Neurony

Funkcje aktywacji

- Wybór funkcji aktywacji jest jednym z problemów podczas projektowania sieci
- Sigmoid
  - Klasyfikacja binarna
  - Tylko do warstwy wyjściowej
- Tanh
  - Lepszy wybór od sigmoida



- ReLU (*rectified linear unit*)
  - Zazwyczaj używane do warstw ukrytych (default)
  - Przyspiesza uczenie, prostsze do obliczenia



Gradient descent dla sieci neuronowych

Cost Function: $J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$

- Oblicz $\hat{y}$, dla każdej pary ze zbioru uczącego
- Następnie w pętli:
  - Oblicz pochodne z Funkcji Kosztu (*Cost Function*) względem wszystkich parametrów:
    - $dW^{[1]} = \frac{dJ}{W^{[1]}},\ db^{[1]} = \frac{dJ}{b^{[1]}},\ dW^{[2]} = \frac{dJ}{W^{[2]}},\ db^{[2]} = \frac{dJ}{b^{[2]}}$
  - Zaktualizuj parametry:
    - $W^{[1]} = W^{[1]} - \alpha\, dW^{[1]}$
    - $b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$
    - $W^{[2]} = W^{[2]} - \alpha\, dW^{[2]}$
    - $b^{[2]} = b^{[2]} - \alpha\, db^{[2]}$

<u>Skąd wziąć pochodne funkcji kosztu?</u>

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

Andrew Ng

https://www.coursera.org/learn/neural-networks-deep-learning/lecture/Wh8NI/gradient-descent-for-neural-networks

https://www.coursera.org/learn/neural-networks-deep-learning/lecture/6dDj7/backpropagation-intuition-optional

Zadanie

Dla zbioru danych użytego w przykładzie PlanarClassification zaimplementować sieć neuronową przy uzyciu framework'a **PyTorch**.

- Zaimplementować strukturę analogiczną jak w przykładzie
- Wykonać porównania wyników oraz czasów uczenia dla:
  - Różnej liczby neuronów w warstwie ukrytej
  - Różnych funkcji aktywacji (*sigmoid*, *tanh*, *ReLU*)

- (dla chętnych) Użyć innego zbioru danych do klasyfikacji binarnej