



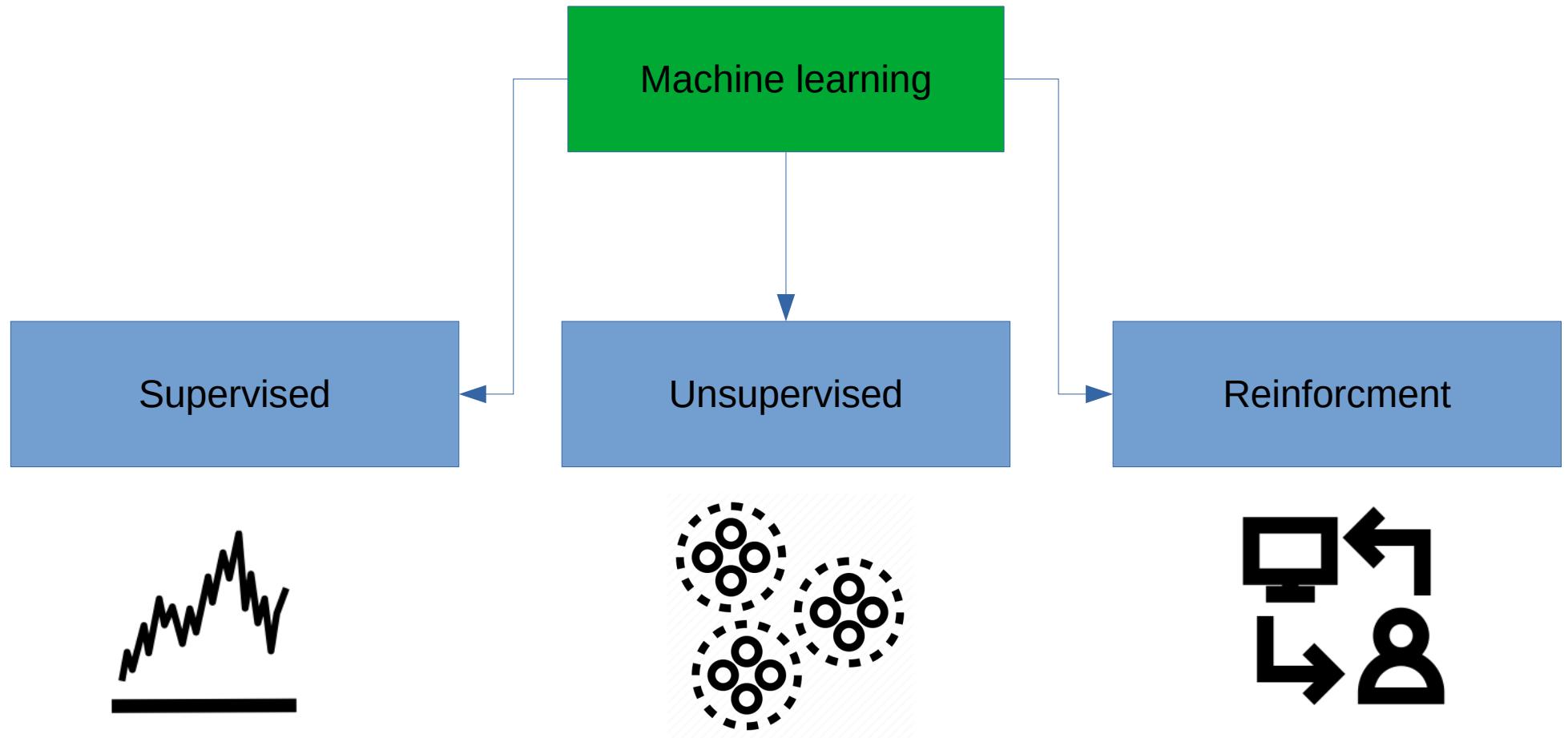
# **Generowanie muzyki za pomocą głębokich sieci neuronowych**

Łukasz Ogan

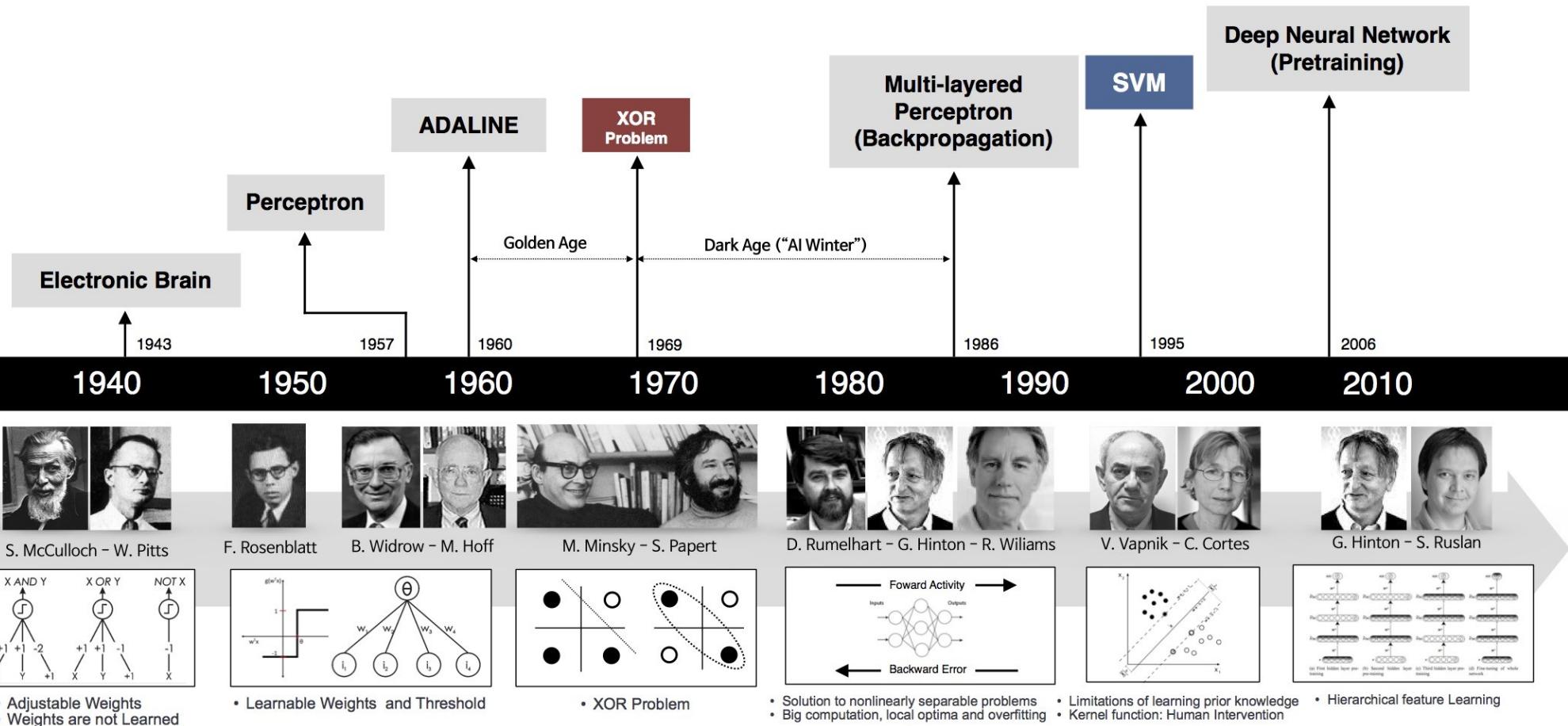
34 spotkanie Bydgoszcz JUG

# Plan

1. Boom na sieci neuronowe
2. Deep Learning w muzyce
3. Perceptron – sztuczny neuron
4. Funkcje aktywacji
5. Zdolność uczenia
6. Wielowarstwowe sieci neuronowe
7. Sekwencje
8. Rekurencyjne sieci neuronowe
9. Sieci z pamięcią
10. Reprezentacja utworów muzycznych
11. Generowanie muzyki z wykorzystaniem modelu znakowego
12. Gdzie trenować sieci?



# Boom na sieci neuronowe



# Boom na sieci neuronowe

1. Dostęp do dużych zbiorów danych
2. Wsparcie sprzętowe – karty graficzne
3. Możliwość trenowania w chmurze
4. Obliczenia równoległe
5. Software – biblioteki pozwalające na szybkie stworzenie modeli (Keras, Tensorflow)



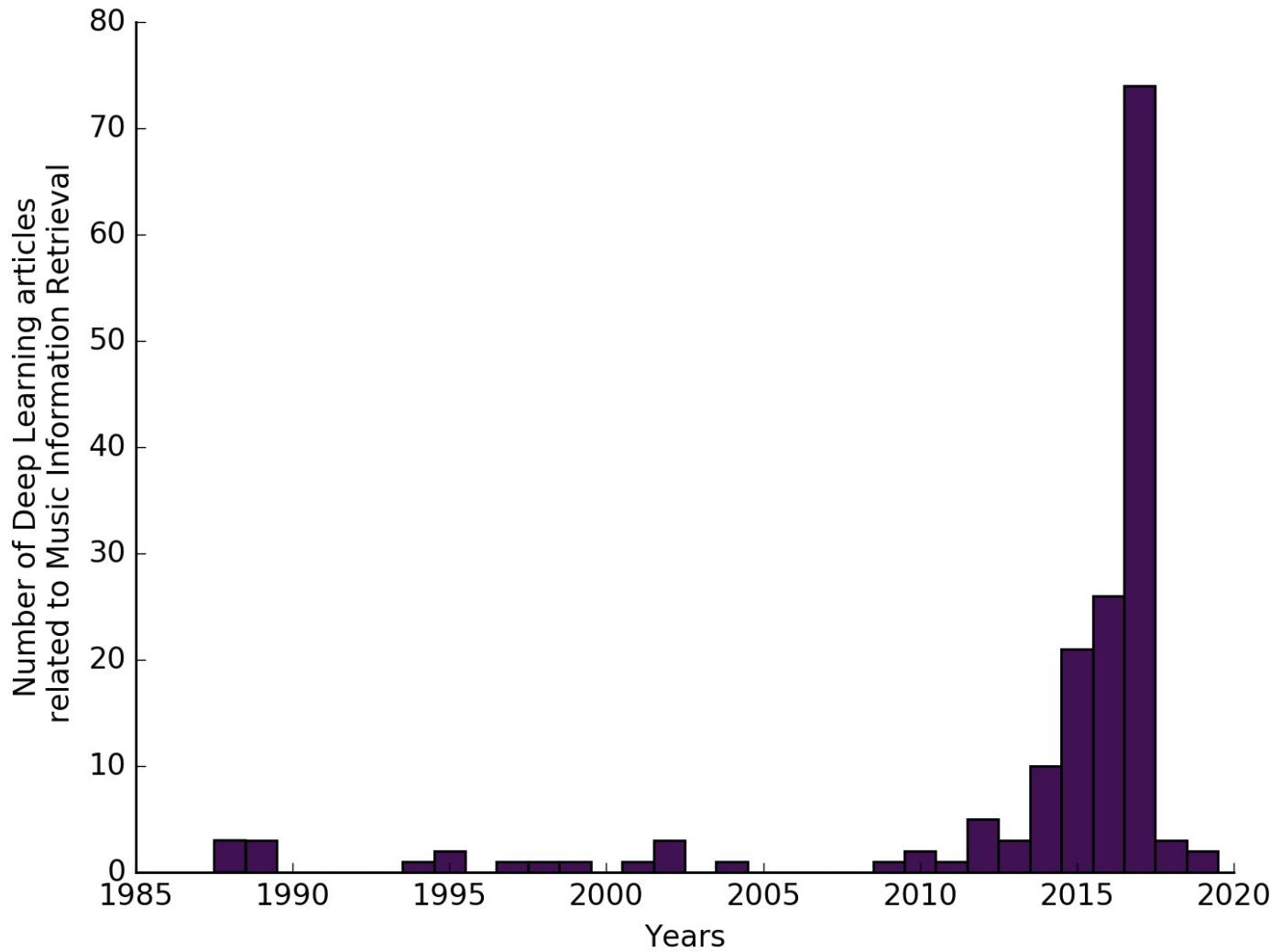
colab





# Aiva Technologies





Źródło: <https://github.com/ybayle/awesome-deep-learning-music/blob/master/dl4m.bib>

# Deep Learning Techniques for Music Generation – *A Survey*

Jean-Pierre Briot<sup>\*,1</sup>, Gaëtan Hadjeres<sup>†</sup> and François-David Pachet<sup>‡</sup>

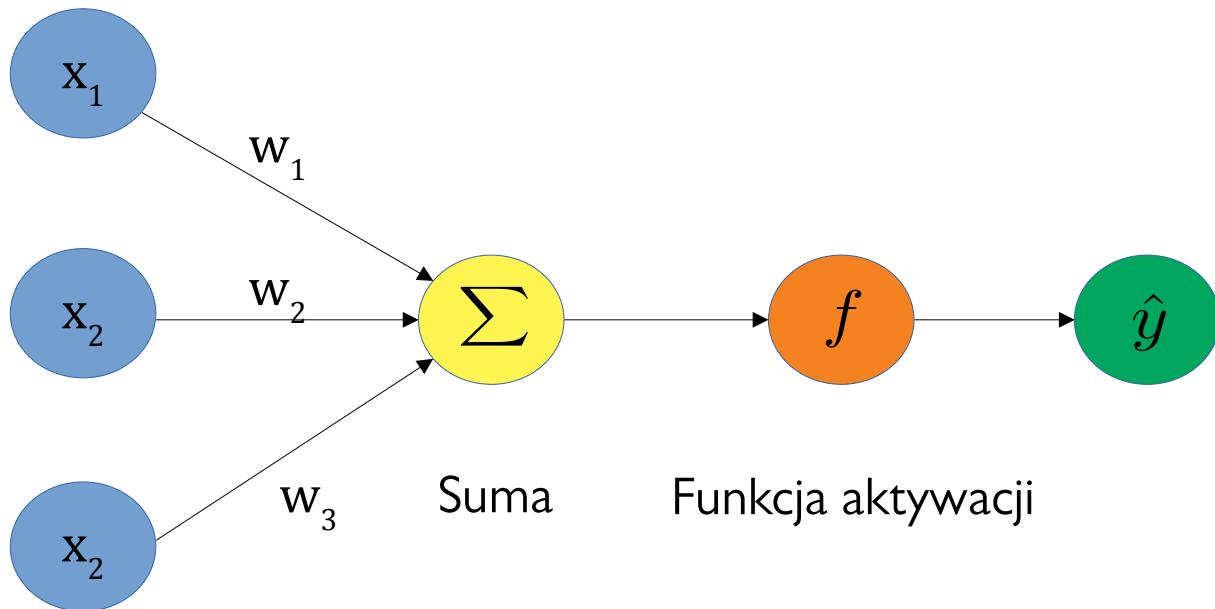
<sup>\*</sup> Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

<sup>†</sup> Sony Computer Science Laboratories, CSL-Paris, F-75005 Paris, France

<sup>‡</sup> Spotify Creator Technology Research Lab, CTRL, F-75008 Paris, France

This paper is a survey and an analysis of different ways of using deep learning (deep artificial neural networks) to generate musical content. We propose a methodology based on five dimensions for our analysis:

# Sztuczny neuron



Wejście

Wagi

Suma

Funkcja aktywacji

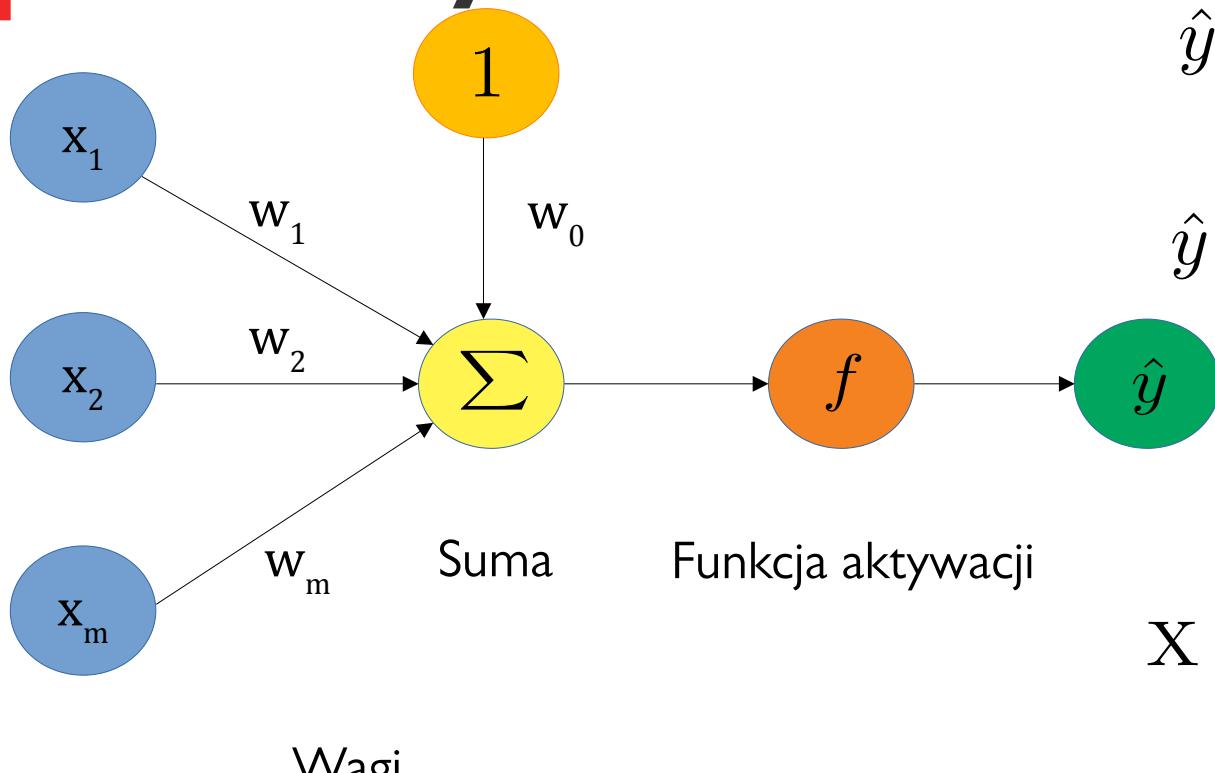
Funkcja aktywacji

$$\hat{y} = g\left(\sum_{i=0}^m x_i w_i\right)$$

Wyjście

Suma wejść

# Sztuczny neuron



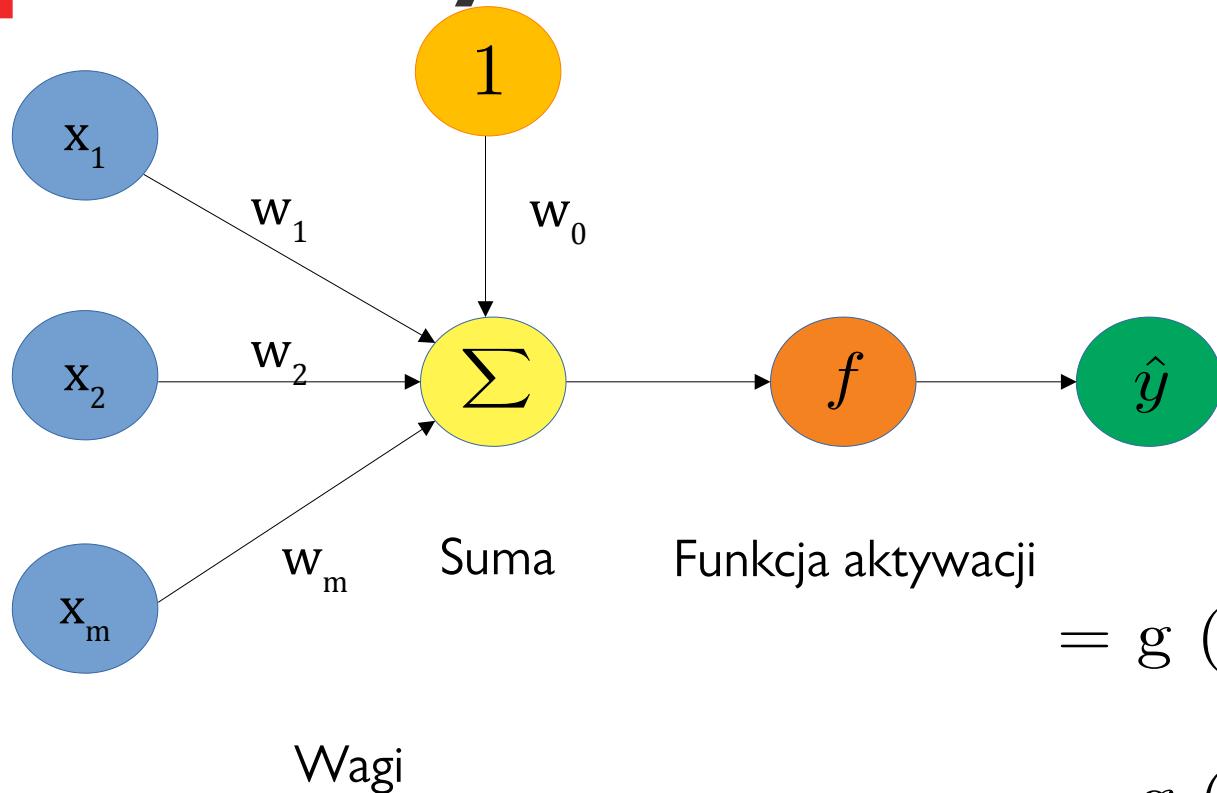
$$\hat{y} = g \left( w_0 + \sum_{i=0}^m x_i w_i \right)$$

$$\hat{y} = g \left( w_0 + \mathbf{X}^T \mathbf{W} \right)$$

$$\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

$$\mathbf{X}^T = [x_1 \quad \dots \quad x_m]$$

# Sztuczny neuron



Wejście

Wagi

Suma

Funkcja aktywacji

$$\hat{y} = g \left( w_0 + \mathbf{X}^T \mathbf{W} \right)$$

$$\mathbf{W} = \begin{bmatrix} 5 \\ -4 \end{bmatrix}$$

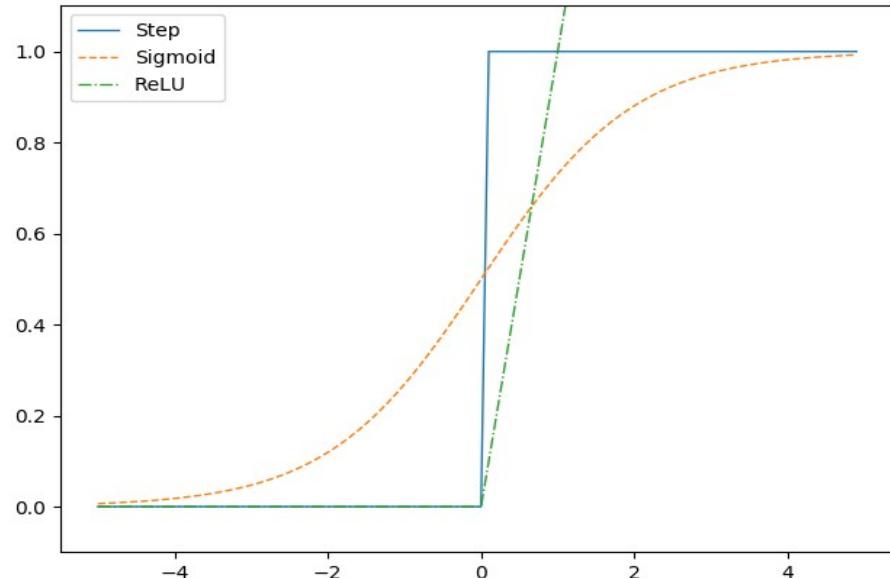
$$\begin{aligned} &= g \left( 1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 5 \\ -4 \end{bmatrix} \right) \\ &= g \left( 1 + 5x_1 - 4x_2 \right) \end{aligned}$$

Równanie prostej

# Funkcja aktywacji

$$\hat{y} = g\left(\sum_{i=0}^m x_i w_i\right)$$

↑  
Funkcja aktywacji



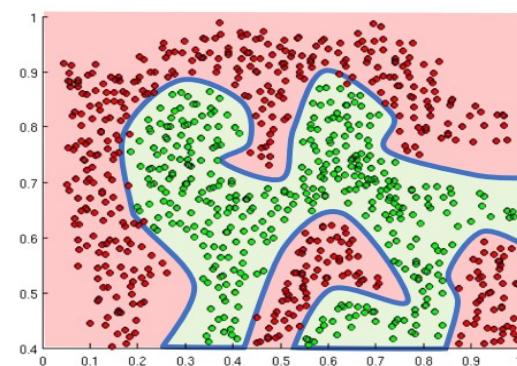
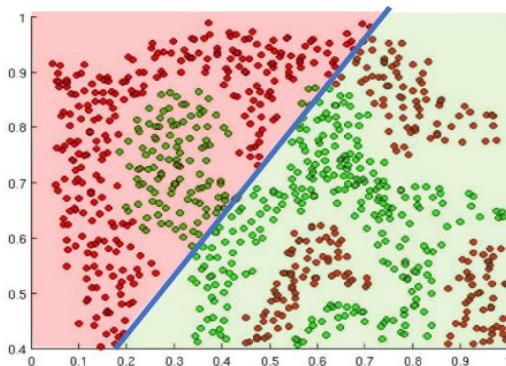
Wybór funkcji aktywacji zależy od rodzaju problemu jaki stawiamy przed siecią do rozwiązania. Dla sieci wielowarstwowych najczęściej stosowane są funkcje nieliniowe, gdyż neurony o takich charakterystykach wykazują największe zdolności do nauki. Umożliwia to otrzymanie na wyjściu sieci informacji ciągłej, a nie tylko postaci: TAK - NIE.

# Funkcja aktywacji

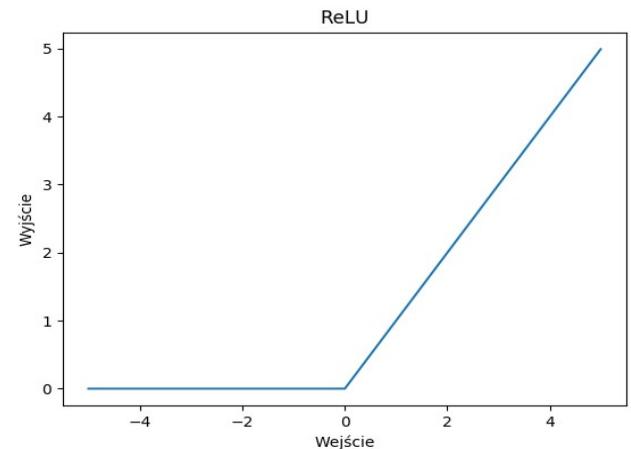
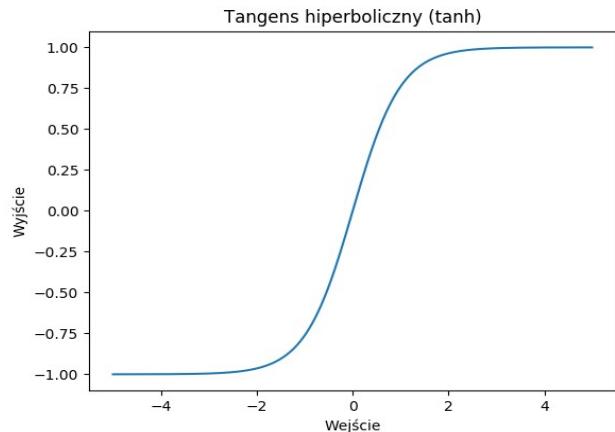
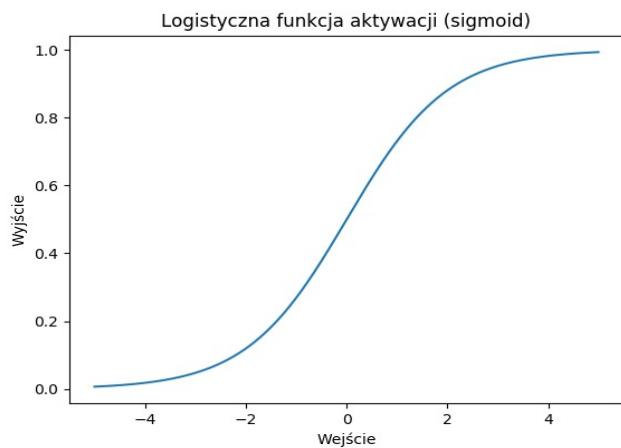
$$\hat{y} = g\left(\sum_{i=0}^m x_i w_i\right)$$

Funkcja aktywacji

Nieliniowe funkcje aktywacji pozwalają na pracę z danymi, które nie są liniowo separowalne.



# Funkcje aktywacji

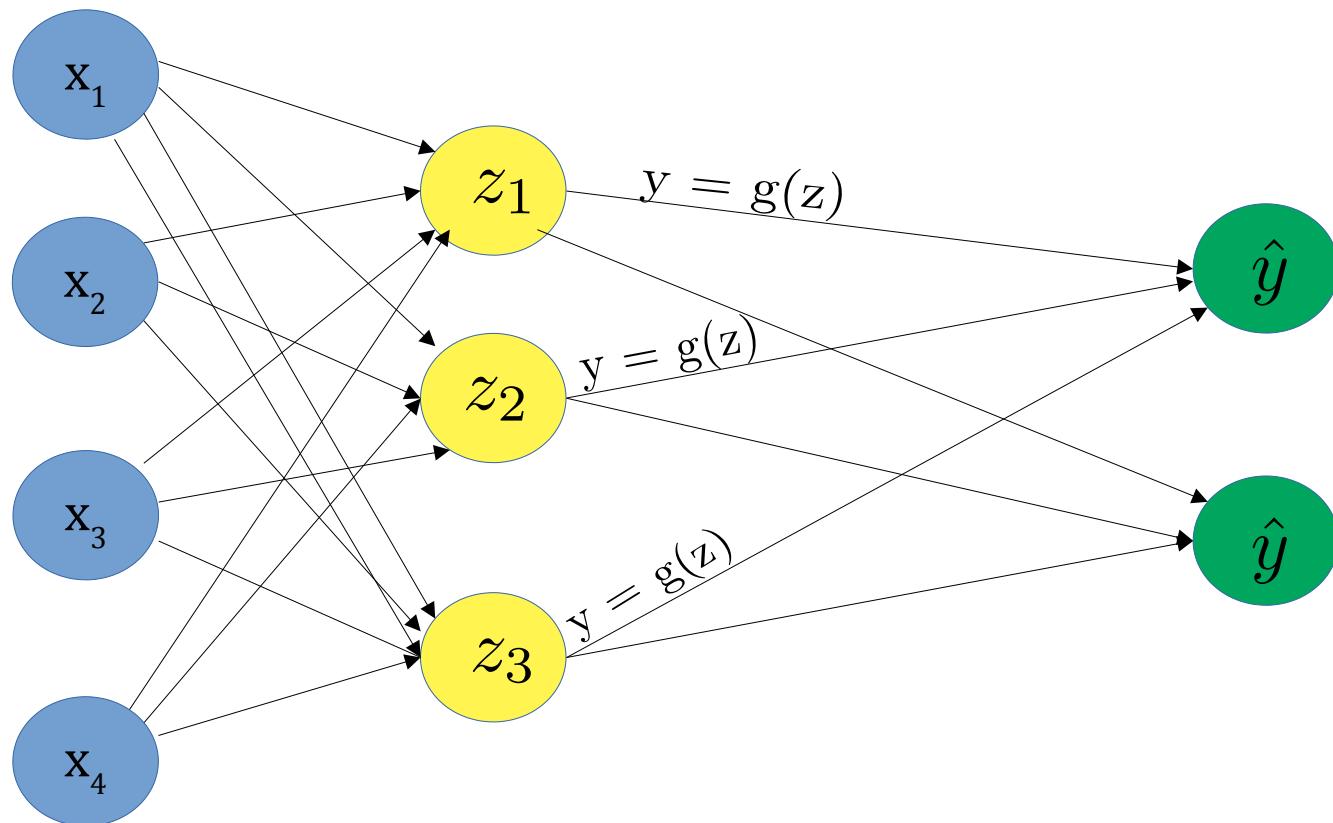


$$f(x) = \frac{1}{1 + e^{-x}}$$

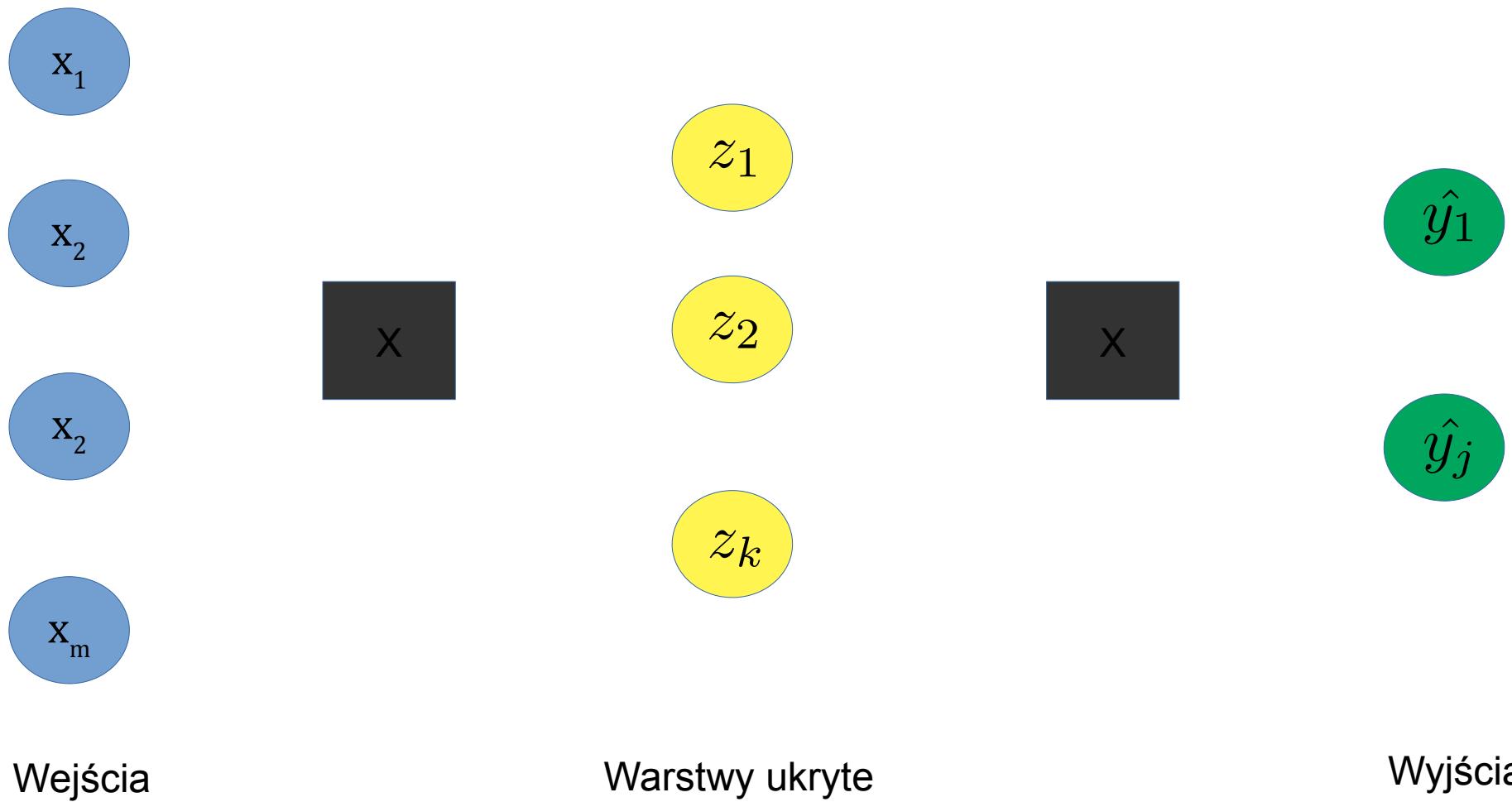
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$f(x) = \max(0, x)$$

# Sieci wielowarstwowe

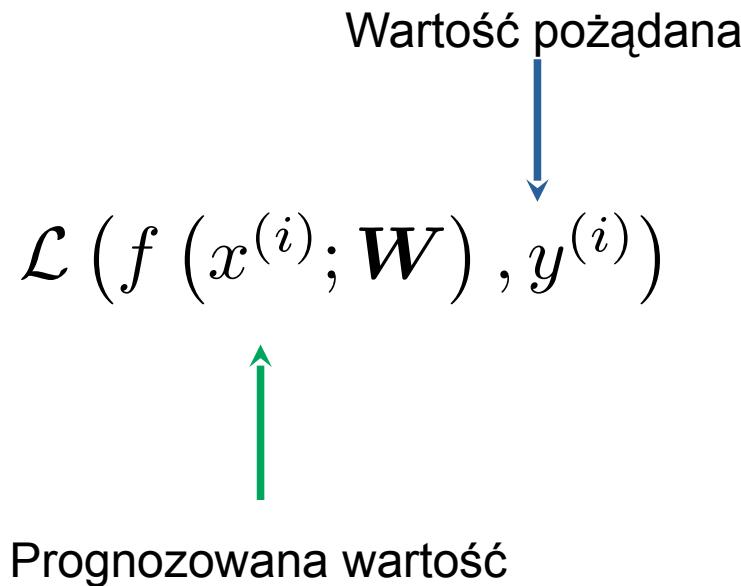


# Sieci wielowarstwowe



# Uczenie sieci

Uczenie sieci polega na minimalizowaniu błędu pomiędzy wartościami oczekiwanyymi z sieci, a wyjściowymi.



# Optymalizacja funkcji kosztu

Należy znaleźć takie wagi, które zminimalizują funkcję kosztu.

Funkcję kosztu wybieramy w zależności od problemu:

**Regresja** – metoda najmniejszych kwadratów

**Klasyfikacja binarna (2 klasy)** – funkcja krosentropii binarnej

**Klasyfikacja wieloklasowa (> 2 klasy)** – funkcja krosentropi kategorycznej

# Funkcja kosztu

Błąd średniokwadratowy (**Mean squared error**)

$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \left( y^{(i)} - f(x^{(i)}; \mathbf{W}) \right)^2$$

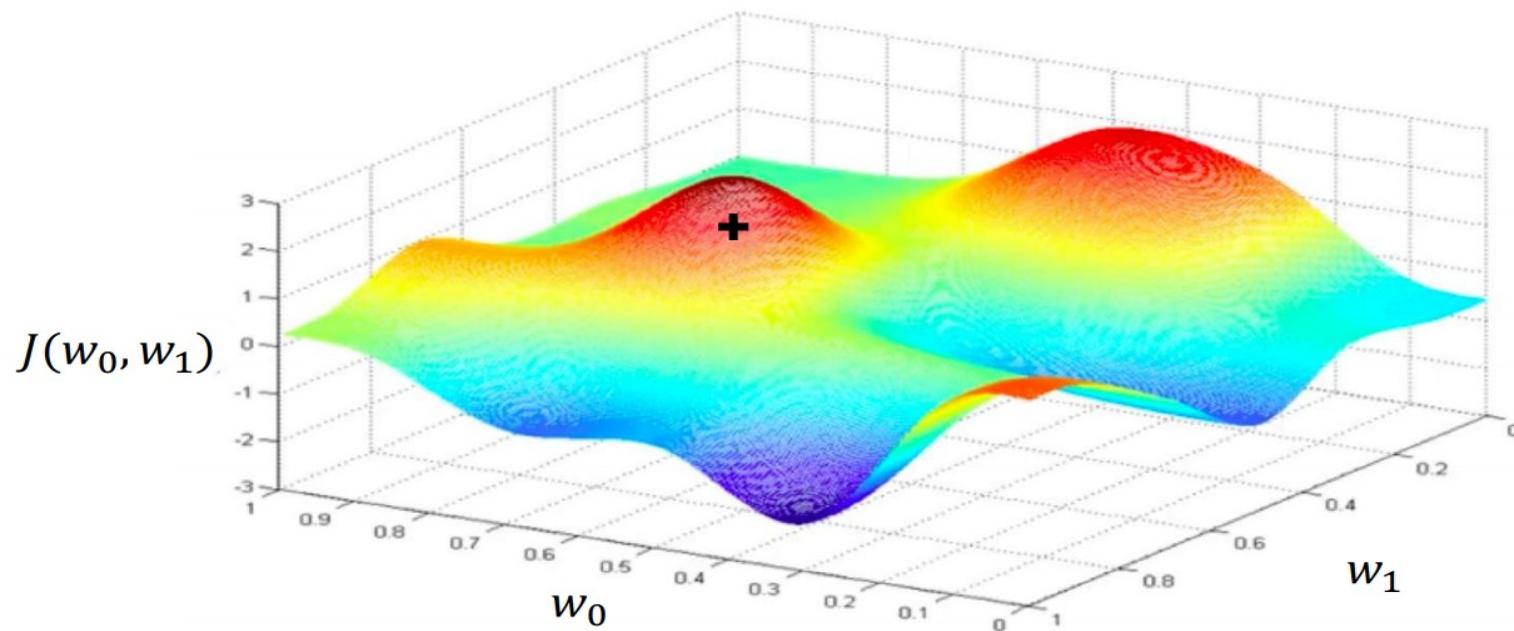
↑                              ↑  
Wartość pożądana          Prognozowana wartość

Entropia krzyżowa (**Cross entropy**)

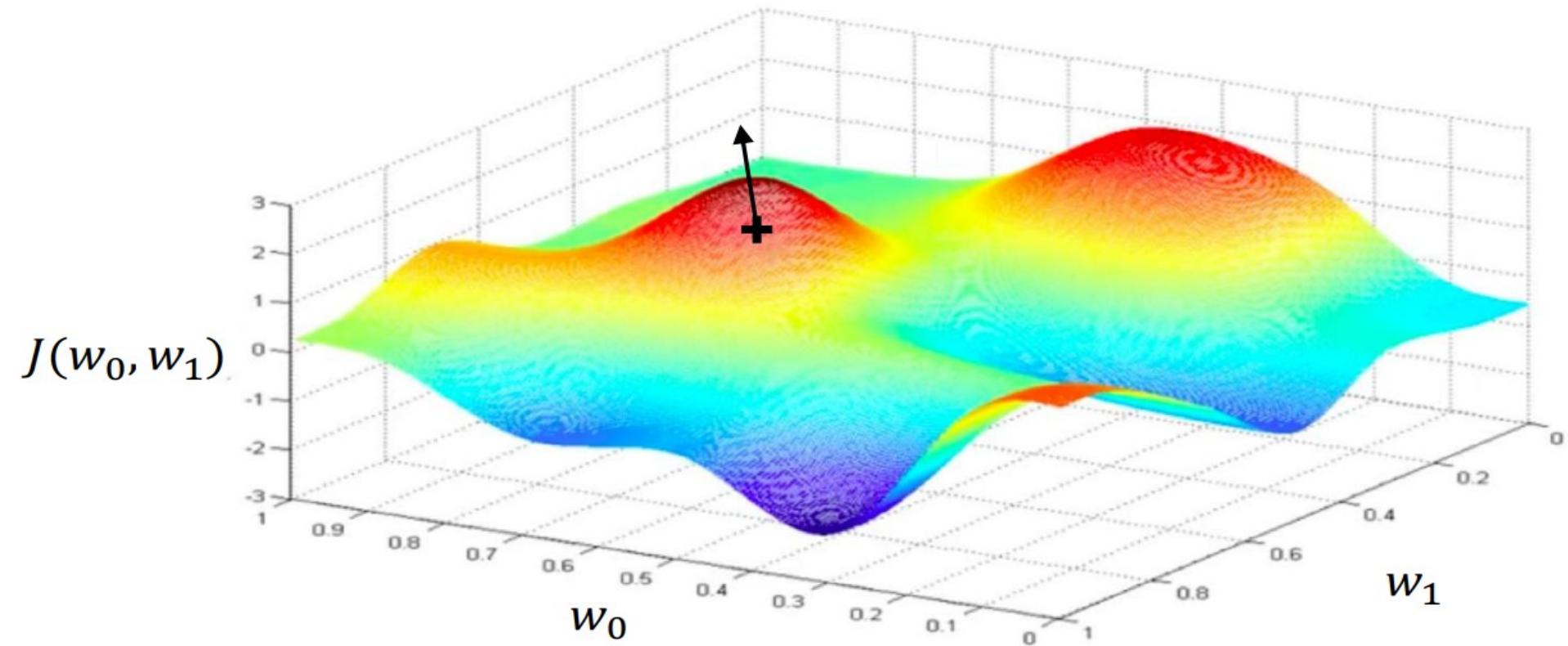
$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n y^{(i)} \log(f(x^{(i)}; \mathbf{W})) + (1 - y^{(i)}) \log(1 - f(x^{(i)}; \mathbf{W}))$$

↑                              ↑                              ↑                              ↑  
Wartość pożądana          Prognozowana wartość

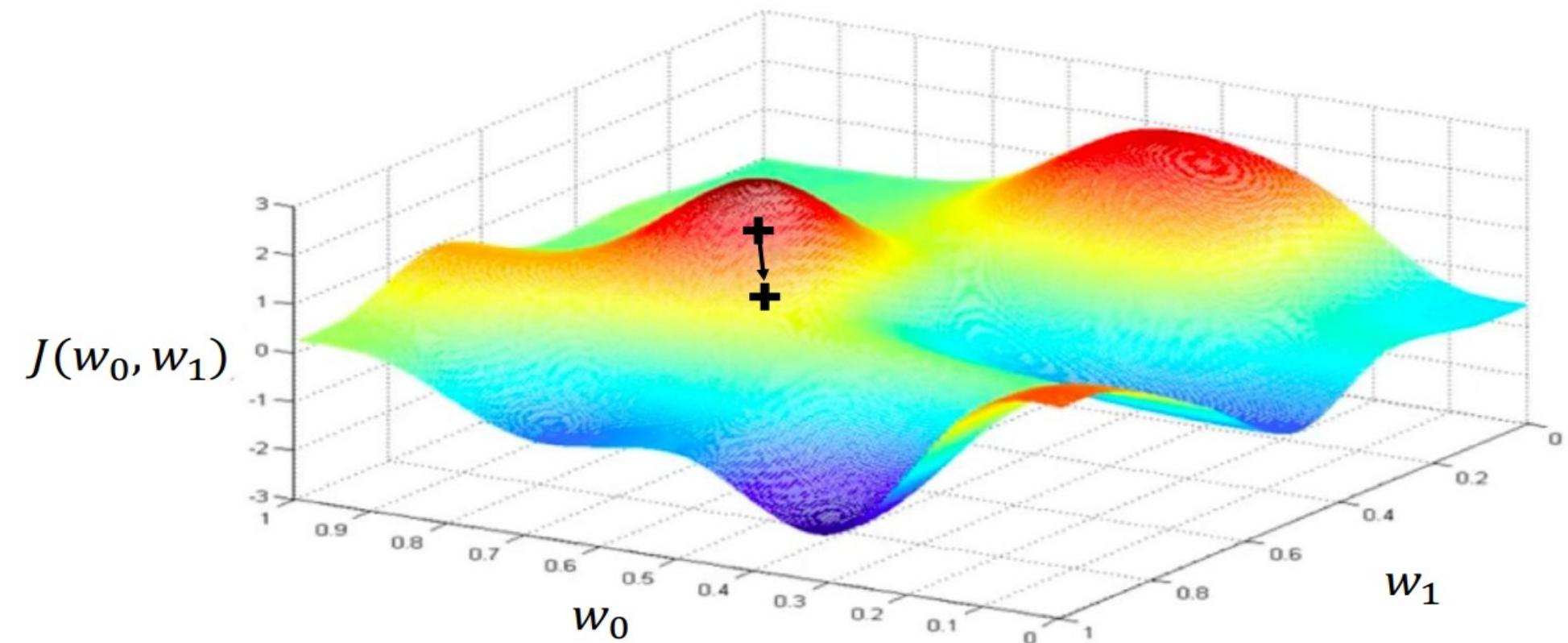
# Optymalizacja funkcji kosztu



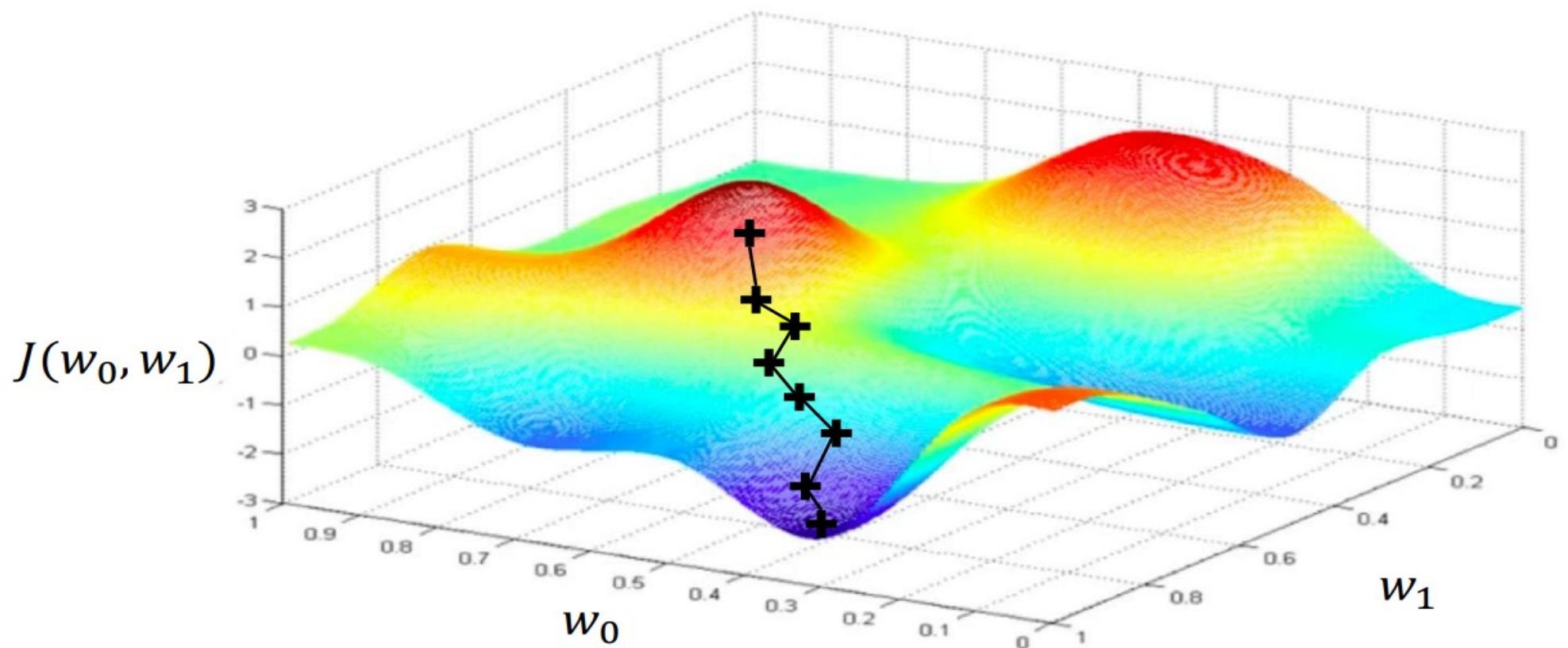
# Optymalizacja funkcji kosztu



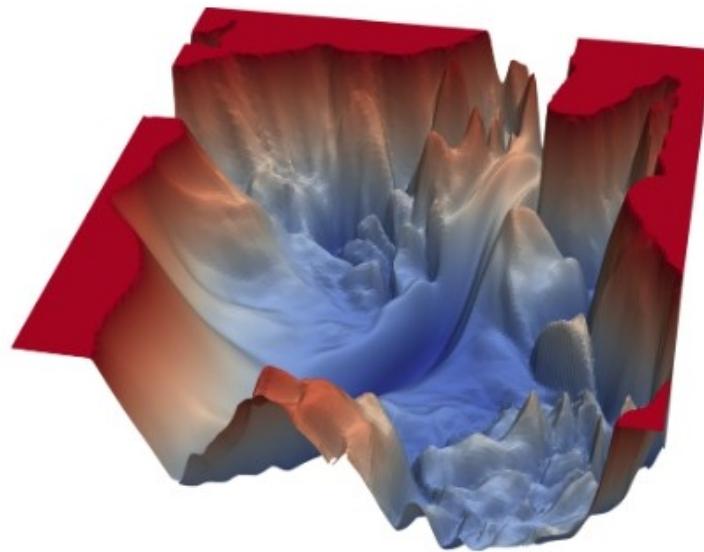
# Optymalizacja funkcji kosztu



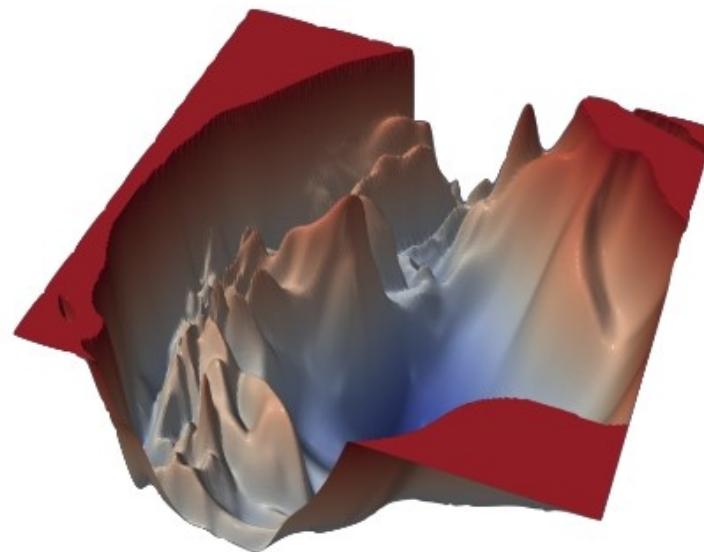
# Optymalizacja funkcji kosztu



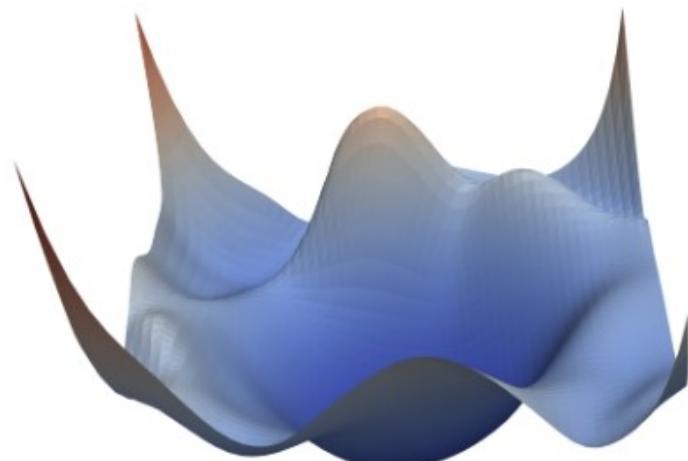
**VGG-56**



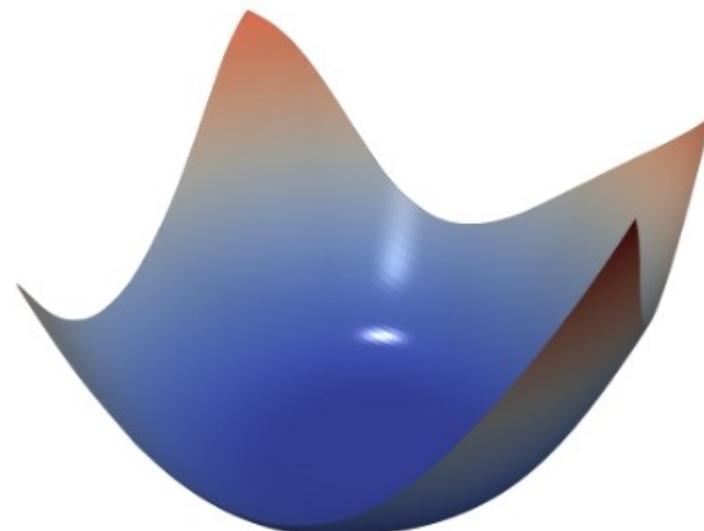
**VGG-110**



**Renset-56**



**Densenet-121**



# Tensorflow Playground



Epoch  
000,000

Learning rate  
0.01

Activation  
Tanh

Regularization  
None

Regularization rate  
0

Problem type  
Classification

## DATA

Which dataset do you want to use?



Ratio of training to test data: 50%

Noise: 0

Batch size: 10

**REGENERATE**

## FEATURES

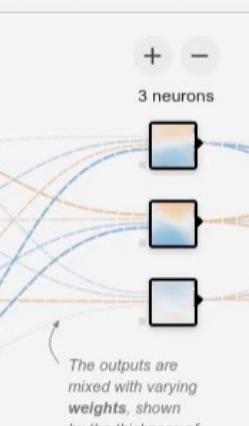
Which properties do you want to feed in?

$x_1$      $x_2$      $x_1^2$      $x_2^2$      $x_1 x_2$

$\sin(x_1)$

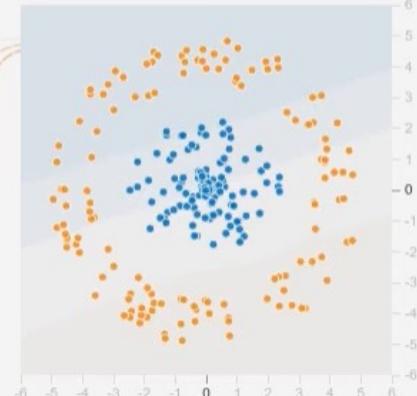
$\sin(x_2)$

## 3 HIDDEN LAYERS



## OUTPUT

Test loss 0.507  
Training loss 0.504



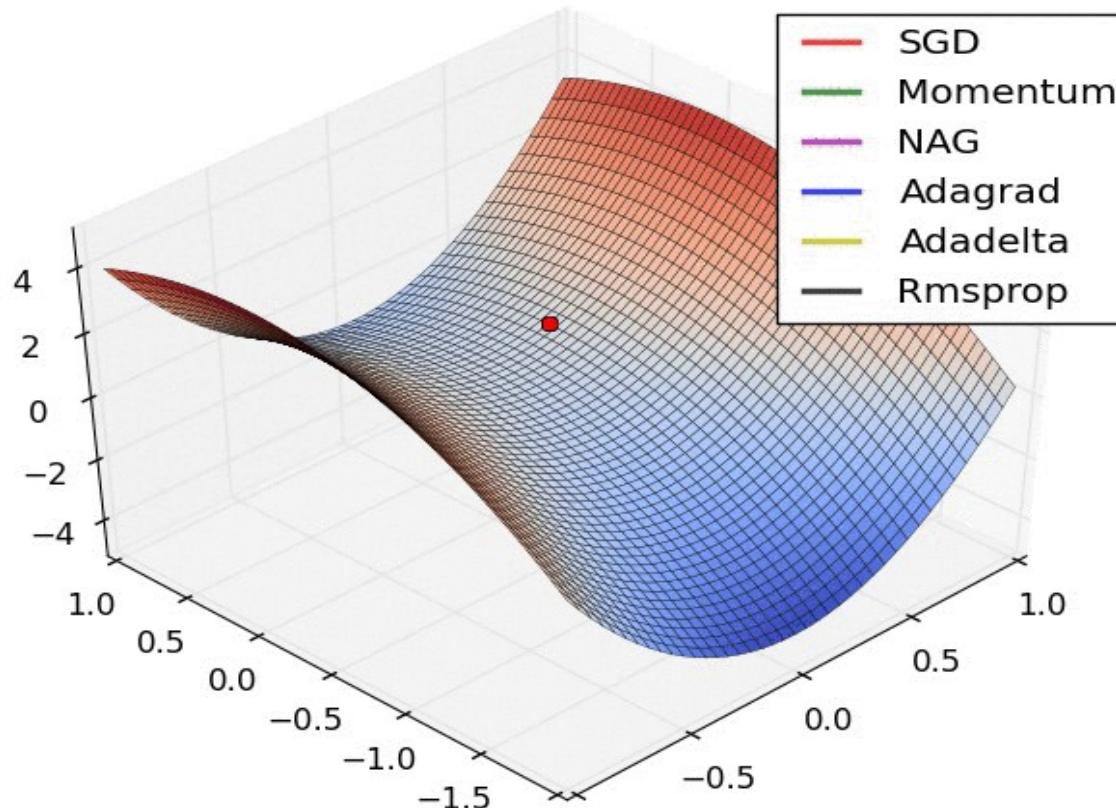
Colors show data, neuron and weight values.  
 Show test data    Discretize output

# Metoda gradientu prostego

## Schemat postępowania

1. Inicjalizacja wag
2. Wykonuj dopóki nie uzyskasz zbieżności
3.       Oblicz gradient
4.       Zaktualizuj wyniki
5. Zwróć wagi

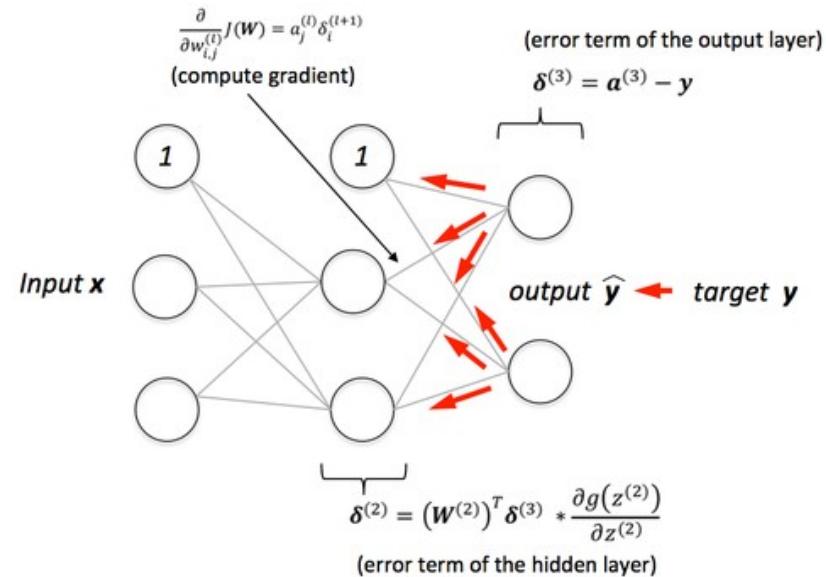
# Optymalizatory



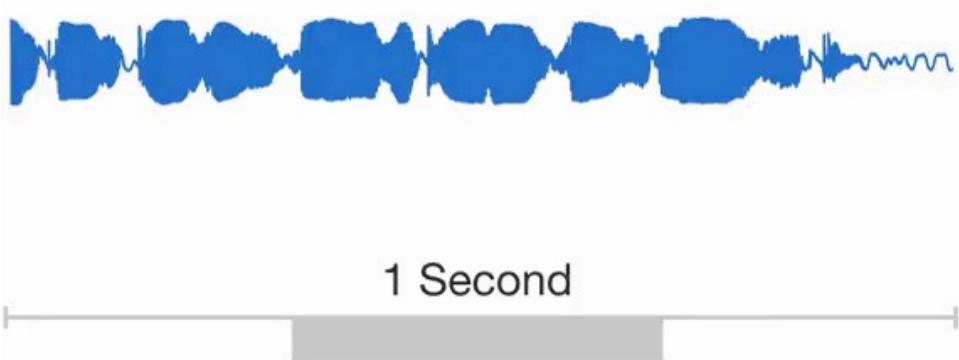
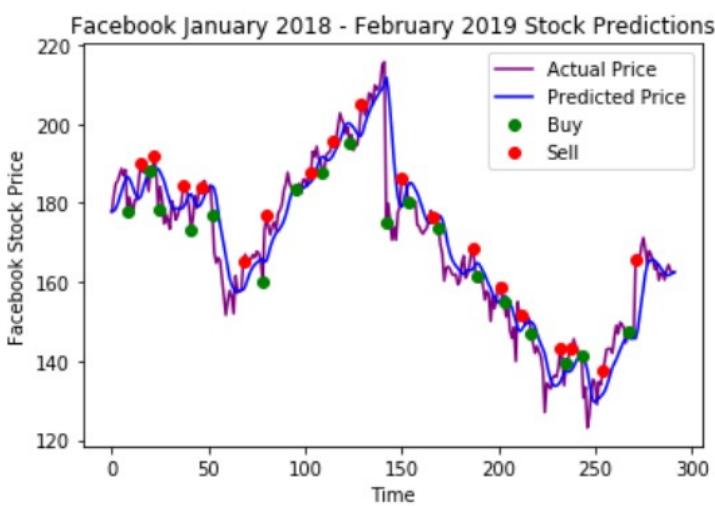
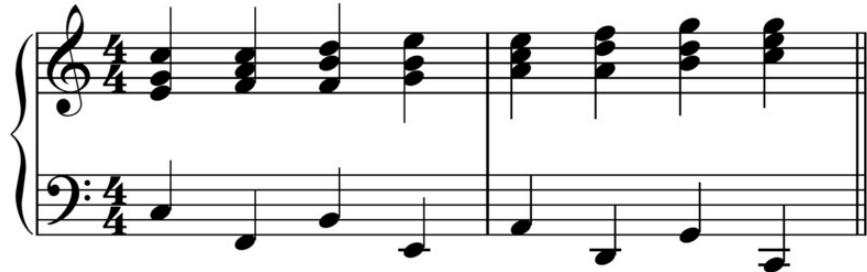
# Wsteczna propagacja

## Schemat postępowania

1. Ustalenie topologii sieci
2. Inicjalizacja wag
3. Dla danego wektora z zestawu uczącego oblicz odpowiedź sieci
4. Dla każdego neuronu wyjściowego oblicz błąd oparty na różnicy
5. Propagacja błędu wstecz do wcześniejszych warstw sieci
6. Zmodyfikuj wagi na podstawie wartości błędu
7. Zatrzymaj się, gdy średni błąd na danych z zestawu uczącego przestanie maleć



# Sekwencje



# Sekwencje

Dziś jest ładna pogoda więc pojadę nad jezioro

Iteracja po wyrazach oknem o stałym rozmiarze (2 słowa)

Dziś jest ładna pogoda więc pojadę nad \_\_\_\_\_

```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')
X = [["dziś"], ["jest"], ["ładna"], ["pogoda"],
      ["więc"], ["pojade"], ["nad"], ["jezioro"]]
enc.fit(X)
enc.transform(X).toarray()
```

```
array([[1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.]])
```

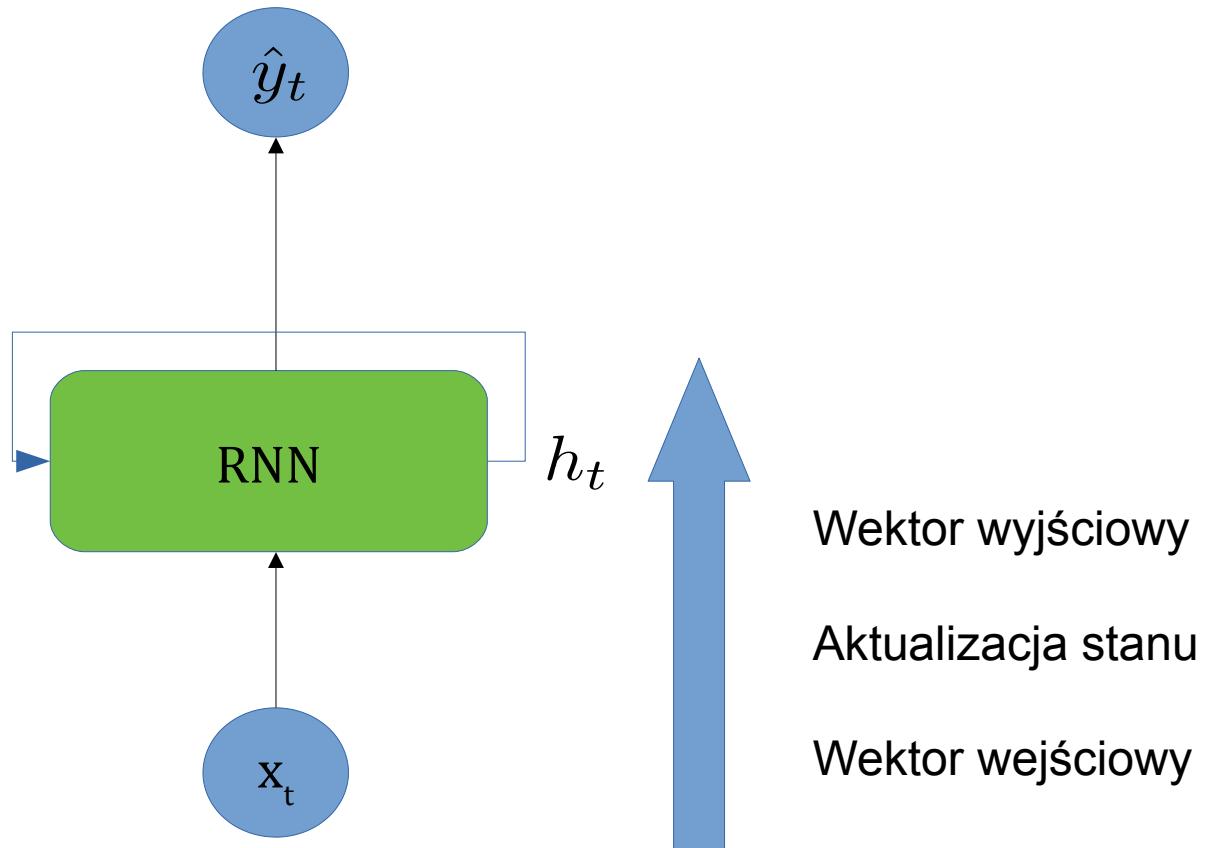
# Rekurencyjna sieć neuronowa

$$\underline{h_t} = f_W (\underline{h_{t-1}}, \underline{x_t})$$

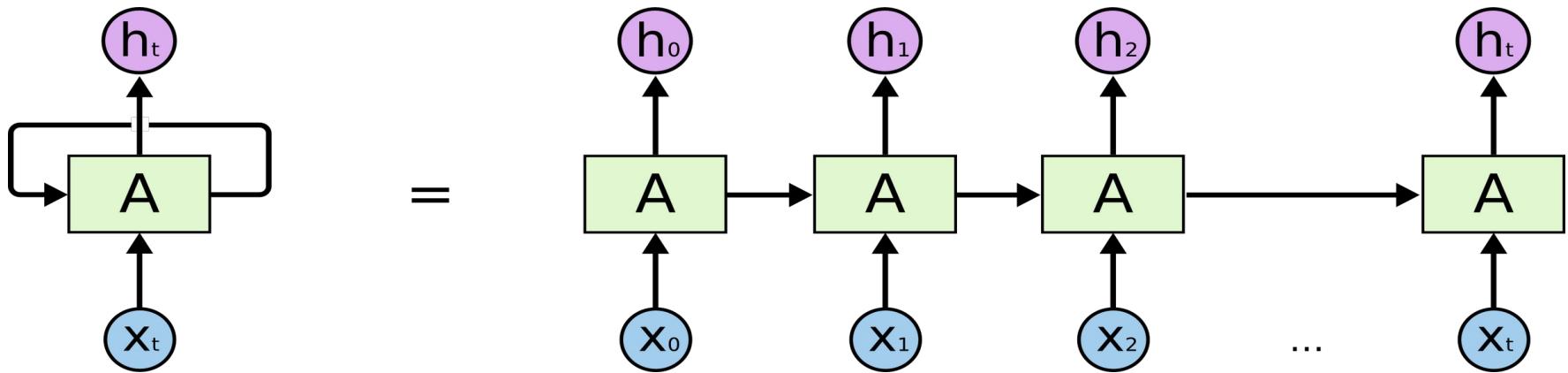
Stan komórki

Poprzedni stan

Wektor wejściowy w czasie t



# Rekurencyjna sieć neuronowa



# Problemy z RNN

1. Podczas uczenia sieci długie sekwencje generują problem zanikania gradientów (*Vanishing gradient problem*)

- występowanie liczb  $< 1$
- mnożenie bardzo małych liczb

2. Rozwiązanie: zastosowanie komórek z bramkami

- LSTM – Long Short Term Memory
- GRU – Gated Recurrent Unit

# Komórki z bramkami (gated cell)

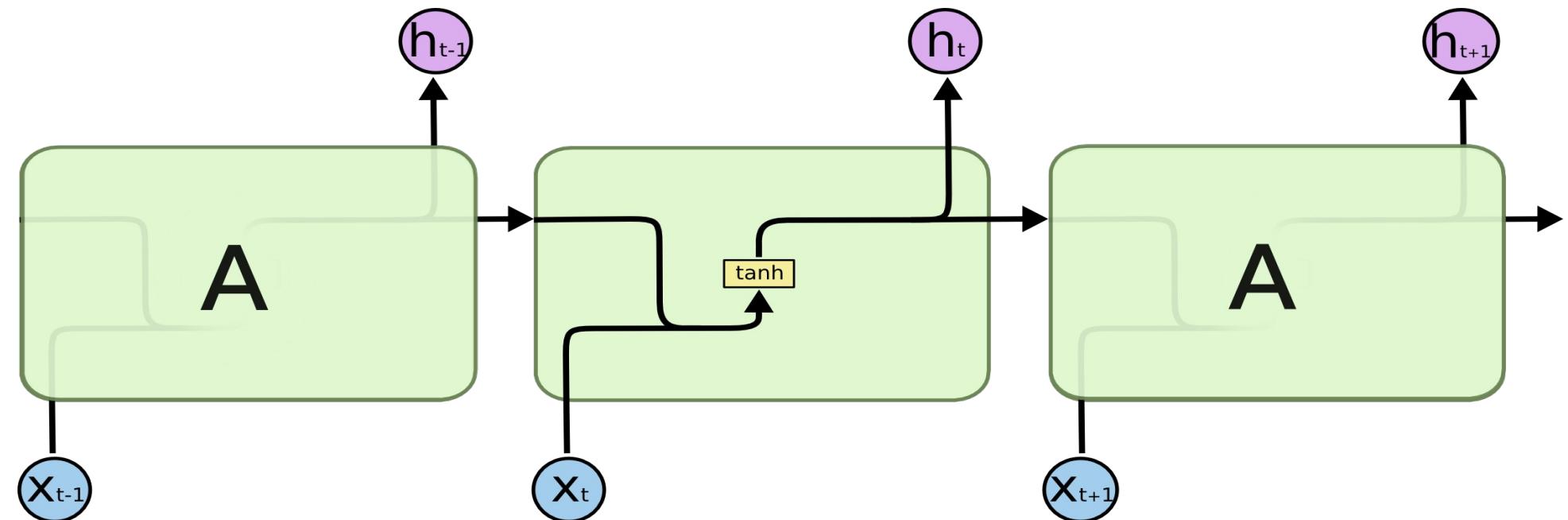
Sieci z komórkami LSTM (Long Short Term Memory) bazującymi na bramkach pozwalają na śledzenie informacji przez wiele kroków czasowych.

Zaproponowane w 1997 przez Seppa Hochreitera i Jürgena Schmidhubera, ulepszana na przestrzeni lat między innymi przez: Alexa Graves, Hasim Sak, Wojciecha Zarembę.

Komórki LSTM wykrywają długoterminowe zależności w sekwencjach.

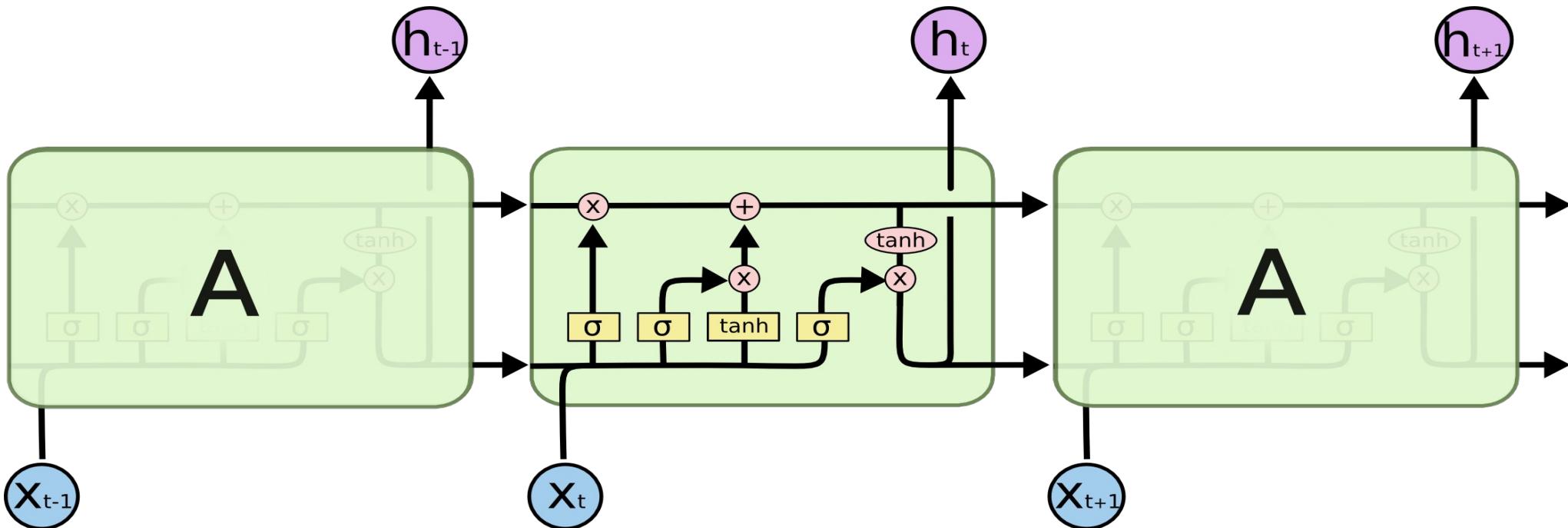


# Schemat sieci RNN



Źródło: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

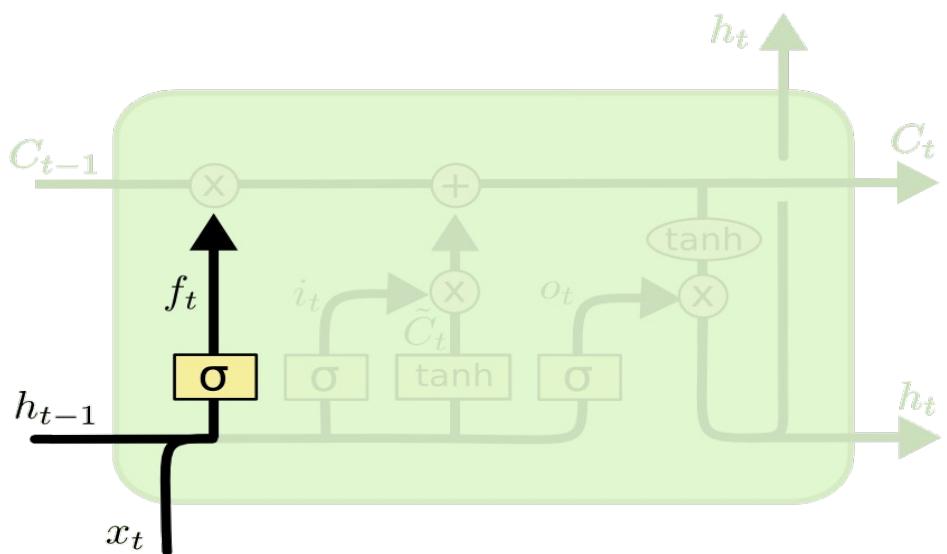
# Komórka LSTM



Źródło: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Bramki w komórce LSTM pozwalają zarządzać przepływem informacji w sieci

# LSTM – forget gate

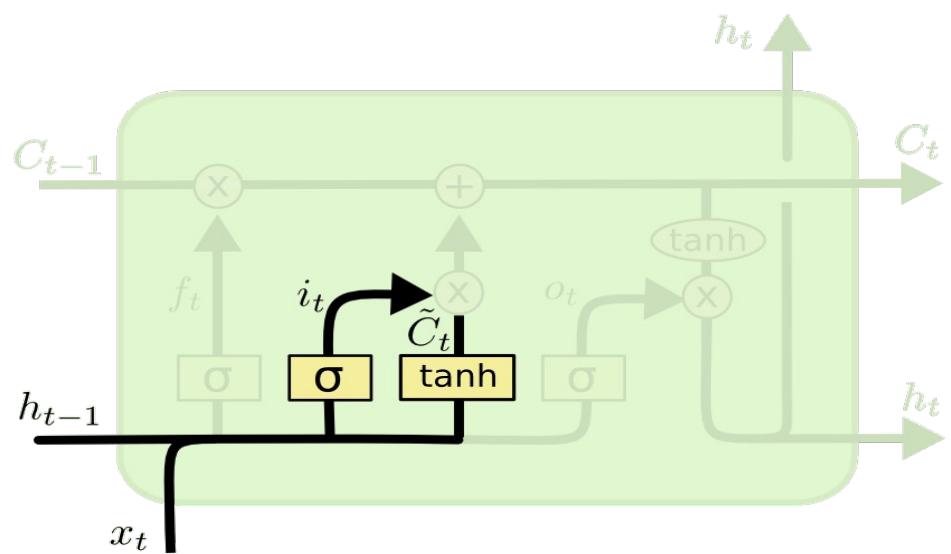


$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Źródło: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Określa, które składniki pamięci długoterminowej powinno zostać usunięte

# LSTM – input gate



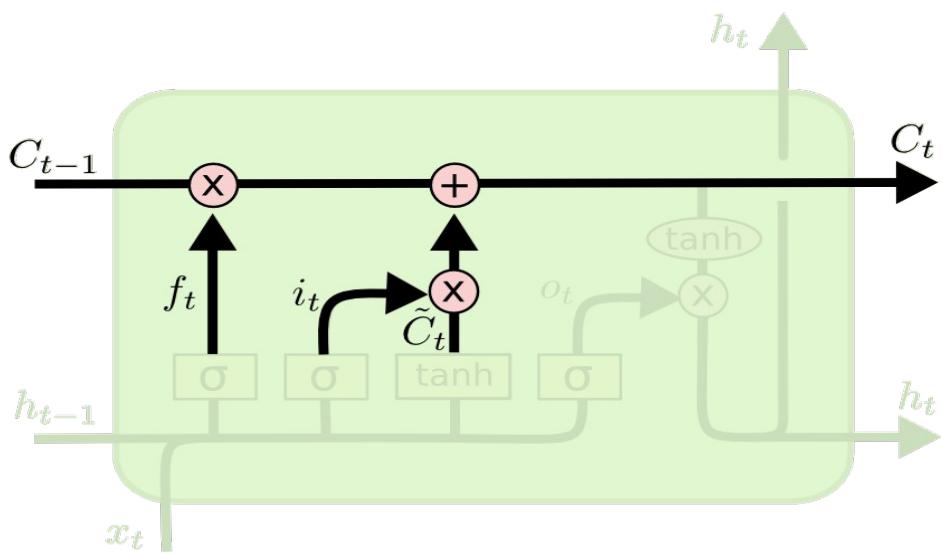
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Źródło: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Decyduje o dodaniu nowych informacji do starego stanu komórki

# LSTM – aktualizacja stanu komórki

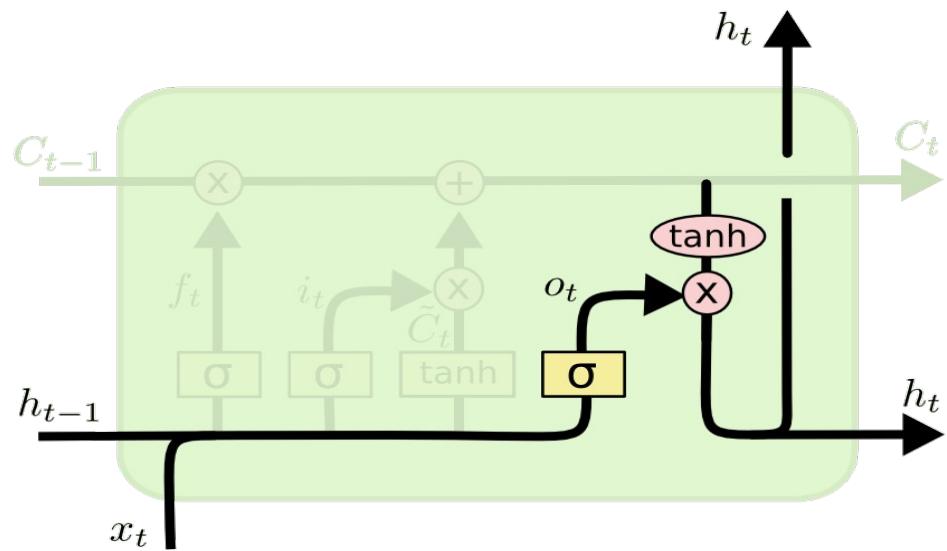


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Źródło: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Zaktualizowanie informacji z bramki zapomnienia, dodanie nowych wartości do stanu długoterminowego

# LSTM – output gate

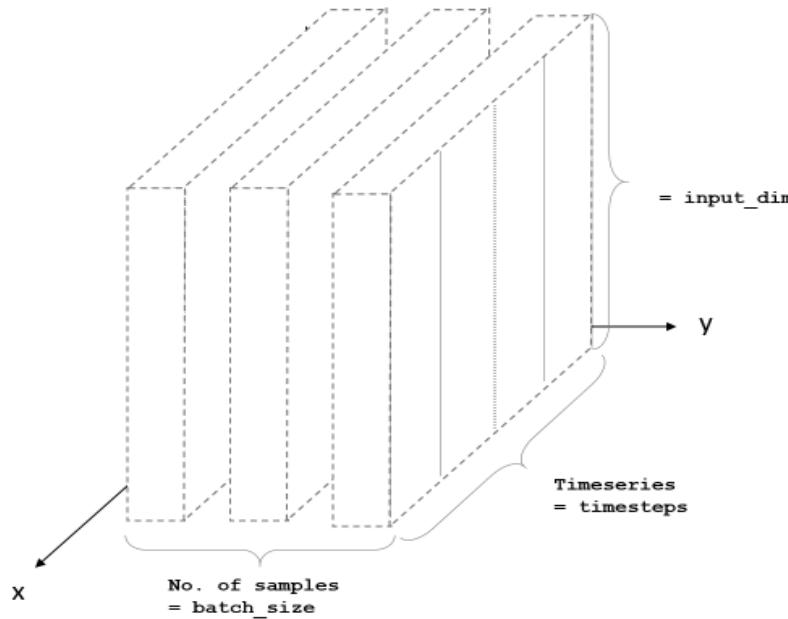


$$o_t = \sigma (W_o [ h_{t-1}, x_t ] + b_o)$$
$$h_t = o_t * \tanh (C_t)$$

Źródło: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Określa, które składowe stanu długoterminowego powinny być wysłane na wyjście w danym kroku czasowym

# LSTM – przygotowanie danych



Tablica 3D:

- Pierwszy wymiar: całkowita liczba próbek
- Drugi wymiar: liczba kroków czasowych
- Trzeci wymiar: długość wektora w jednej sekwencji wejściowej

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
n_steps = 3
X, y = split_sequence(raw_seq, n_steps)

for i in range(len(y)):
    print("{} => {}".format(X[i], y[i]))

[10 20 30] => 40
[20 30 40] => 50
[30 40 50] => 60
[40 50 60] => 70
[50 60 70] => 80
[60 70 80] => 90

n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
# definiujemy model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# karmimy model danymi
model.fit(X, y, epochs=200, verbose=1)
x_input = array([15, 22, 40])
x_input = x_input.reshape((1, n_steps, n_features))
# dokonujemy predykcji
yhat = model.predict(x_input, verbose=0)

print(yhat)
[[47.733196]]
```

# Zapis utworu muzycznego

**Surowy plik audio** – plik przechowujący cyfrowy sygnał audio.

**MIDI** - (Musical Instrument Digital Interface) system służący do zapisu i przekazywania informacji pomiędzy instrumentami.

**ABC** – Notacja pozwalająca zapisać nuty w formie tekstowej

À la Comtesse D'APPONY.

# Nocturne.

F. CHOPIN. Op. 27, N° 2.

Lento sostenuto. ( $\text{♩} = 50$ )

8.

*dolce.*

*legato sempre.*

*R.W.*

*fz*

```
from mido import MidiFile
mid = MidiFile('chopin27.mid')
for i, track in enumerate(mid.tracks):
    print('Track {}: {}'.format(i, track.name))
    for message in track:
        print(message)
```

```
Track 0: Chopin Nocturne Opus 27 Nr. 1
<meta message track_name name='Chopin Nocturne Opus 27 Nr. 1' time=0>
<meta message copyright text='Copyright © 2003 by Bernd Krueger ' time
=0>
<meta message text text='Frederic Chopin' time=0>
<meta message text text='Larghetto' time=0>
<meta message text text='Fertiggestellt am 24.5.2003\n' time=0>
<meta message text text='Update am 26.7.2010\n' time=0>
<meta message text text='Update am 19.2.2014\n' time=0>
<meta message text text='Dauer: 4:52 Minuten\n' time=0>
<meta message smpte_offset frame_rate=25 hours=32 minutes=0 seconds=3
frames=0 sub_frames=0 time=0>
<meta message time_signature numerator=4 denominator=4 clocks_per_clic
k=24 notated_32nd_notes_per_beat=8 time=0>
```

```
note_on channel=0 note=37 velocity=0 time=0
note_on channel=0 note=44 velocity=21 time=0
note_on channel=0 note=44 velocity=0 time=160
note_on channel=0 note=56 velocity=22 time=0
<meta message set_tempo tempo=993213 time=160>
note_on channel=0 note=56 velocity=0 time=0
note_on channel=0 note=49 velocity=23 time=0
note_on channel=0 note=49 velocity=0 time=160
note_on channel=0 note=44 velocity=24 time=0
note_on channel=0 note=44 velocity=0 time=160
note_on channel=0 note=37 velocity=26 time=0
<meta message set_tempo tempo=952381 time=160>
note_on channel=0 note=37 velocity=0 time=0
note_on channel=0 note=44 velocity=27 time=0
note_on channel=0 note=44 velocity=0 time=160
```

# Format MIDI

Octave Number	Notes												
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	
0	0	1	2	3	4	5	6	7	8	9	10	11	
1	12	13	14	15	16	17	18	19	20	21	22	23	
2	24	25	26	27	28	29	30	31	32	33	34	35	
3	36	37	38	39	40	41	42	43	44	45	46	47	
4	48	49	50	51	52	53	54	55	56	57	58	59	
5	60	61	62	63	64	65	66	67	68	69	70	71	
6	72	73	74	75	76	77	78	79	80	81	82	83	
7	84	85	86	87	88	89	90	91	92	93	94	95	
8	96	97	98	99	100	101	102	103	104	105	106	107	
9	108	109	110	111	112	113	114	115	116	117	118	119	
10	120	121	122	123	124	125	126	127					

# Notacja ABC

Notes



X:1  
T:Notes  
M:C  
L:1/4  
K:C  
C, D, E, F, | G, A, B, C | D E F G | A B c d | e f g a | b c' d' e' | f' g' a' b' | ]

# Klasyfikacja i predykcja

**Klasyfikacja** - przypisanie pewnych obiektów do odpowiednich klas na podstawie pewnych cech tych obiektów.

**Predykcja (prognozowanie)** - przewidywanie przyszłych zdarzeń. Wybór, najbardziej prawdopodobnej wartości w nadchodzącym okresie, gdzie podstawę tego wyboru stanowi dotychczasowy przebieg tego zjawiska.

# Zbiory danych

**MIDI** – Preludia Chopina – 130511 nut, 190 – symboli unikalnych (rozmiar alfabetu)

**ABC** – Muzyka Irlandzka – 103664 znaków, 79 – rozmiar alfabetu

ABC datasets - <http://abc.sourceforge.net/>

MIDI datasets - <http://www.piano-midi.de/>

<https://colinraffel.com/projects/lmd/>

Kilkenny Tune



# Przygotowanie danych - MIDI

1. Parsowanie plików midi z wykorzystaniem biblioteki music21.
2. Stworzenie listy nut.
3. Stworzenie zbioru unikalnych nut.
4. Posortowanie nut
5. Przypisanie liczby całkowitej do danej nuty

```
{  
    'A4': 0,  
    'B4': 1,  
    'C4': 2,  
    'C5': 3,  
    'D5': 4,  
    'E4': 5,  
    'E5': 6,  
    'F4': 7,  
    'F5': 8,  
    'G4': 9,  
    ...  
}  
{  
    [5, 7, 9, 0, 9] => 7  
    [7, 9, 0, 9, 7] => 5  
    [9, 0, 9, 7, 5] => 9  
    [0, 9, 7, 5, 9] => 3  
    [9, 7, 5, 9, 3] => 4  
    ...  
}  
7 = G4 = [0, 0, 0, 1, 0, 0, 0, 0, 0]
```

Wyjście z sieci ma rozmiar słownika wejściowego, na każdej pozycji znajduje się prawdopodobieństwo wygenerowanego znaku.

# Przygotowanie danych

[77, 53, 34, 53, 77] => 53

[53, 34, 53, 77, 53] => 77

[34, 53, 77, 53, 77] => 53

[53, 77, 53, 77, 53] => 77

[77, 53, 77, 53, 77] => 77

[53, 77, 53, 77, 77] => 27

[77, 53, 77, 77, 27] => 16

[53, 77, 77, 27, 16] => 7

[77, 77, 27, 16, 7] => 77

[77, 27, 16, 7, 77] => 53

[27, 16, 7, 77, 53] => 34

[16, 7, 77, 53, 34] => 53

[7, 77, 53, 34, 53] => 23

[77, 53, 34, 53, 23] => 23

[53, 34, 53, 23, 23] => 73

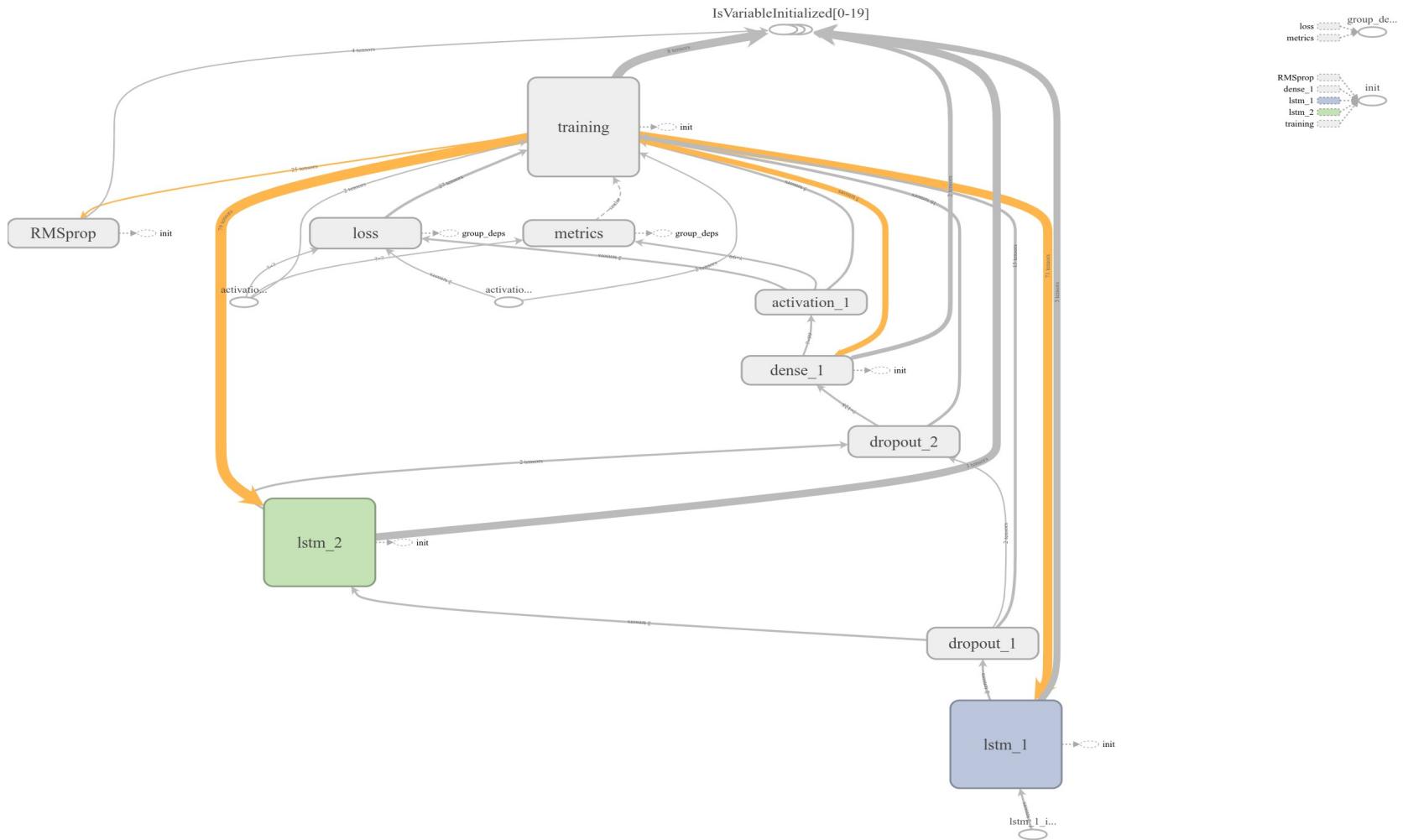
[34, 53, 23, 23, 73] => 23

...

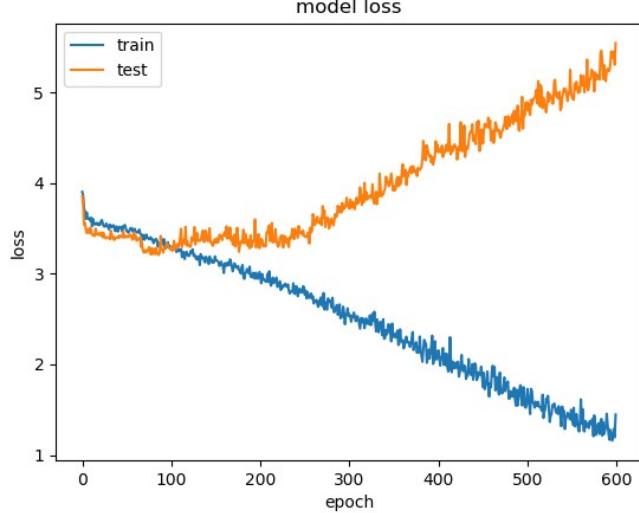
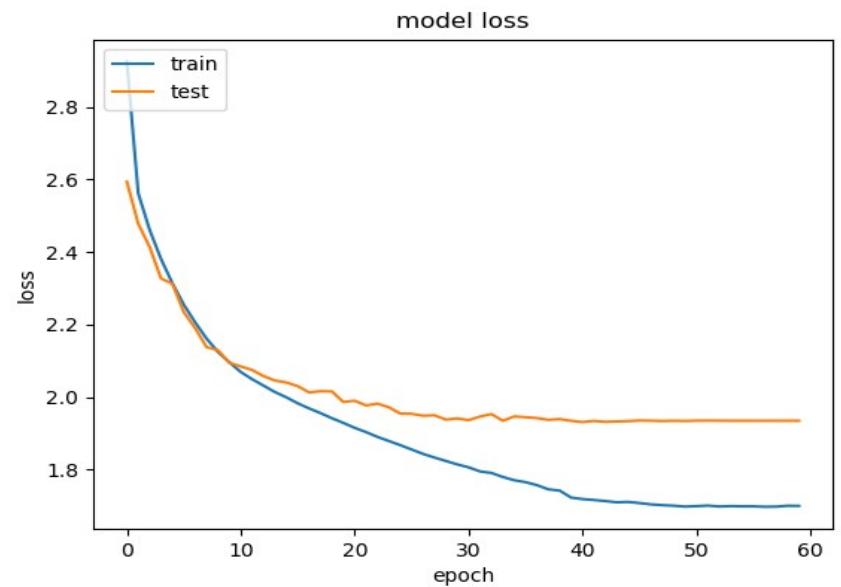
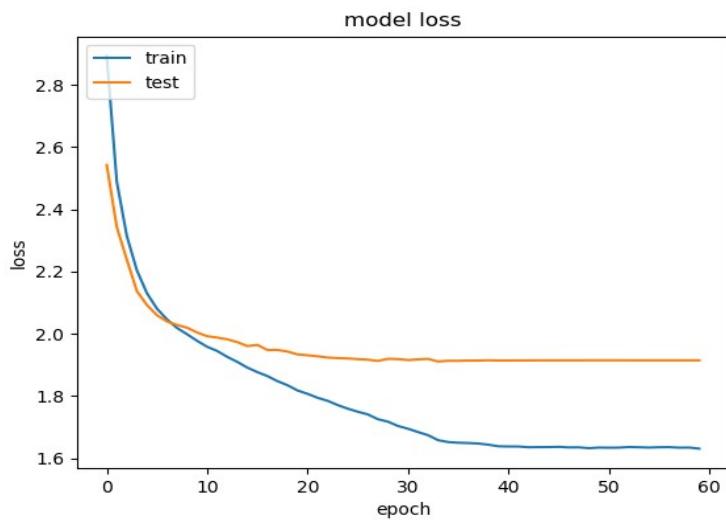
# Model sieci

Layer (type)	Output Shape	Param #
<hr/>		
<hr/>		
lstm_1 (LSTM)	(None, 100, 128)	106496
dropout_1 (Dropout)	(None, 100, 128)	0
lstm_2 (LSTM)	(None, 128)	131584
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 79)	10191
activation_1 (Activation)	(None, 79)	0
<hr/>		
<hr/>		

# Model sieci - TensorBoard



# Wyniki - MIDI

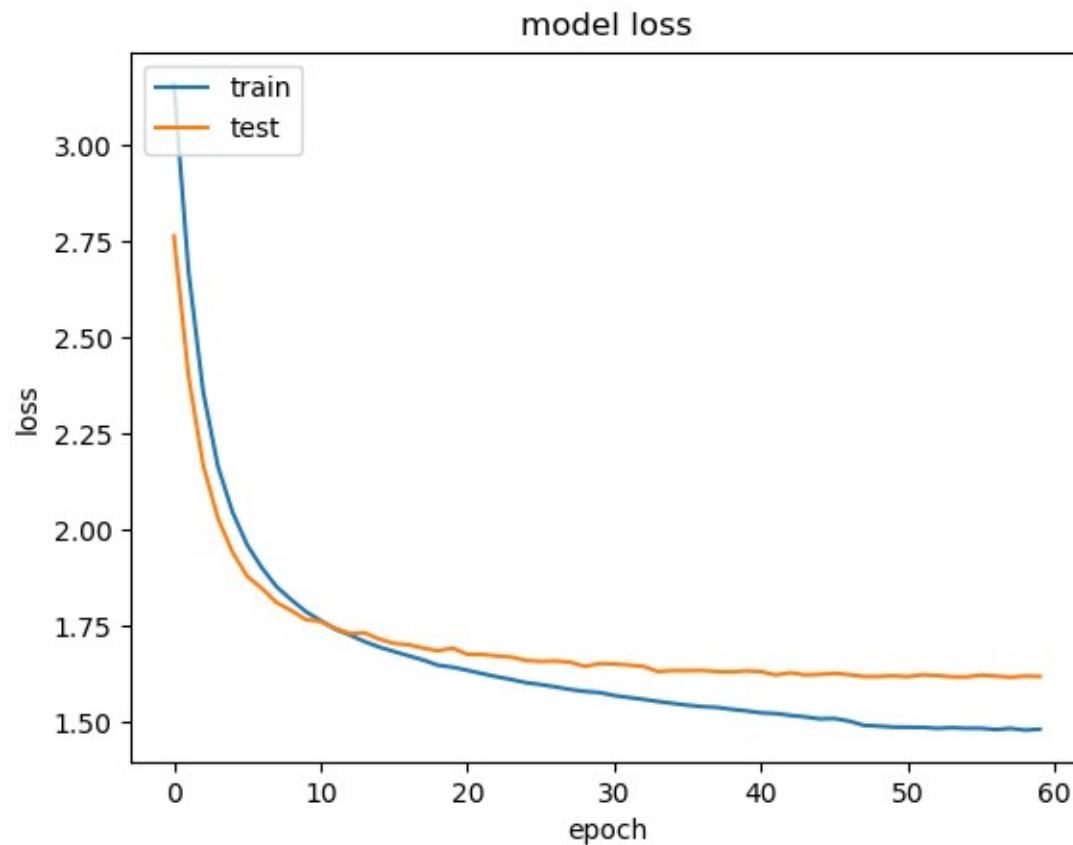


# Przygotowanie danych do sieci - ABC

X:1  
T:Alexander's  
Z: id:dc-hornpipe-1  
M:C|  
L:1/8  
K:D Major  
(3ABC|dAFA DFAd|fdcd FAdf|gfge fefd|(3efe (3dcB A2  
(3ABC|!  
dAFA DFAd|fdcd FAdf|gfge fefd|(3efe dc d2:||!  
AG|FAdA FAdA|GBdB GBdB|Acec Acec|dfaf gecA|!  
FAdA FAdA|GBdB GBdB|Acef fefd|(3efe dc d2:||!

```
{\n    '\n': 0,\n    ' ': 1,\n    '!': 2,\n    '\"': 3,\n    '#': 4,\n    """': 5,\n    '(': 6,\n    ')': 7,\n    ',': 8,\n    '-': 9,\n    '.': 10,\n    '/': 11,\n    '0': 12,\n    '1': 13,\n    '2': 14,\n    '3': 15,\n    '4': 16,\n    '5': 17,\n    '6': 18,\n    '7': 19,\n    ...\n}
```

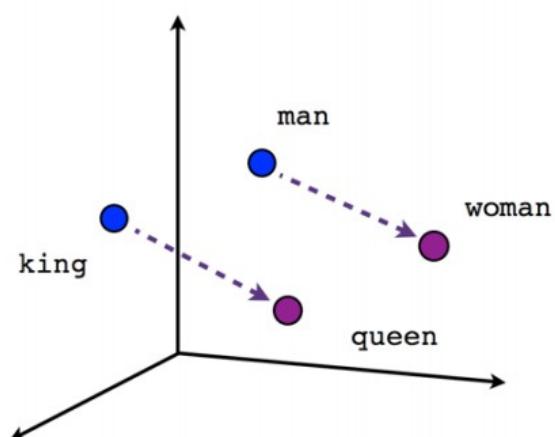
# Wyniki - ABC



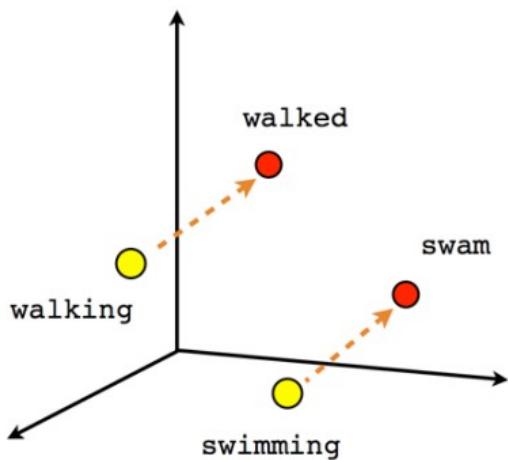
# Doskonalenie modelu

1. Transpozycja utworów celem zwiększenia ilości danych
3. Dodanie długości dźwięków
4. Reprezentacja nut jako Word Embedding
5. Zastosowanie warstwy Attention

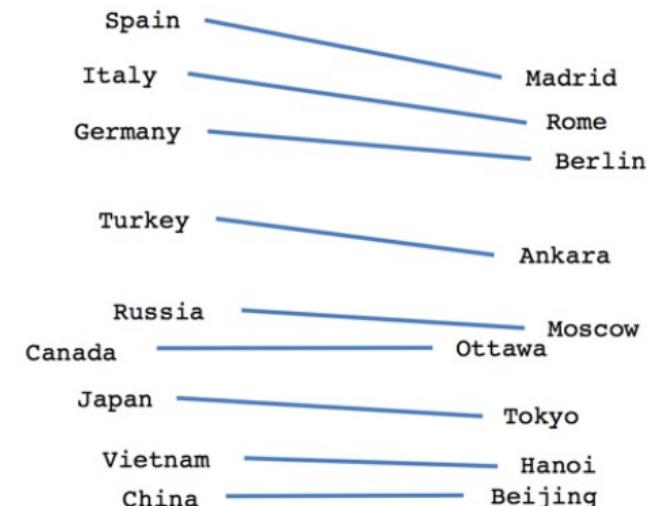
# Word Embedding



Male-Female



Verb tense



Country-Capital

# Word Embedding

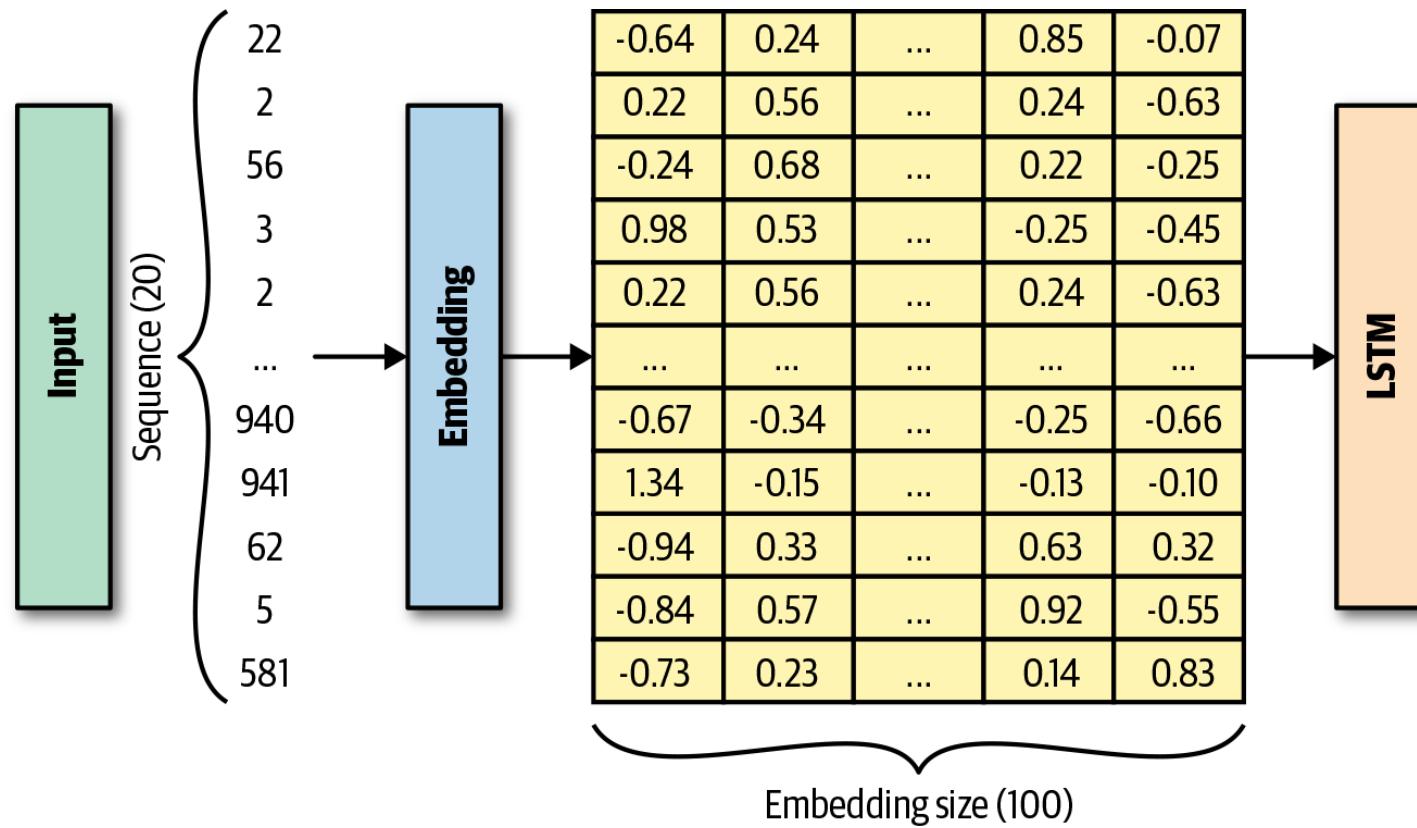
token	Embedding					
1	0.23	0.44	...	0.4	-0.1	0.8
2	0.3	0.5	...	0.32	0.1	0.4
....	...	...	...	...	...	...
2343	0.2	0.45	...	0.22	0.89	0.9
2344	0.1	-0.3	...	0.3	0.1	0.2

X

X - Rozmiar warstwy Embedding

Y – Ilość słów

# Word Embedding



# Attention

---

## Attention Is All You Need

---

**Ashish Vaswani\***

Google Brain

[avaswani@google.com](mailto:avaswani@google.com)

**Noam Shazeer\***

Google Brain

[noam@google.com](mailto:noam@google.com)

**Niki Parmar\***

Google Research

[nikip@google.com](mailto:nikip@google.com)

**Jakob Uszkoreit\***

Google Research

[usz@google.com](mailto:usz@google.com)

**Llion Jones\***

Google Research

[llion@google.com](mailto:llion@google.com)

**Aidan N. Gomez\*** †

University of Toronto

[aidan@cs.toronto.edu](mailto:aidan@cs.toronto.edu)

**Łukasz Kaiser\***

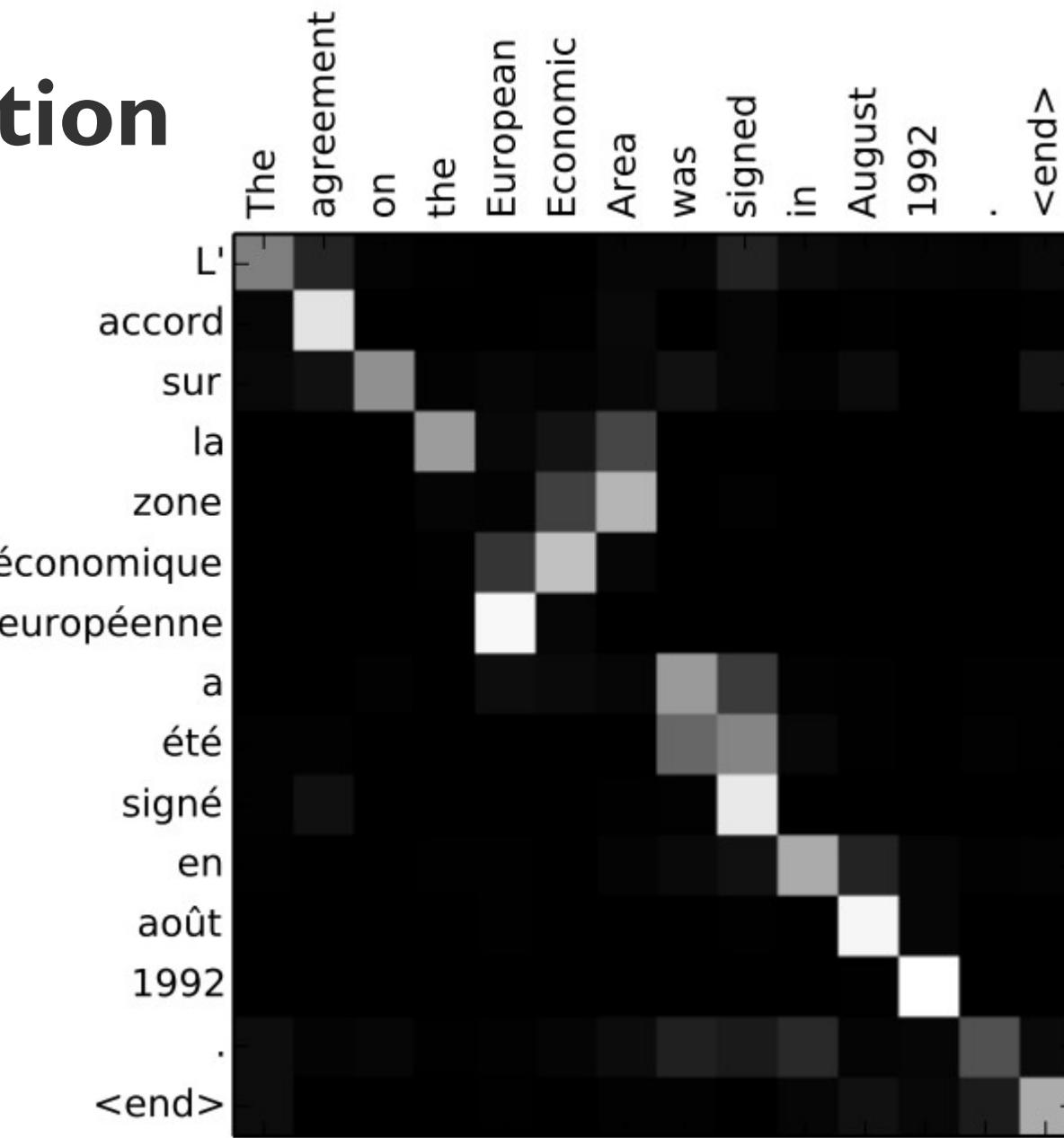
Google Brain

[lukaszkaiser@google.com](mailto:lukaszkaiser@google.com)

**Illia Polosukhin\*** ‡

[illia.polosukhin@gmail.com](mailto:illia.polosukhin@gmail.com)

# Attention



# Suity JS Bacha na Wiolonczelę

- 38 utworów
- transpozycja o 2 oktawy w góre
- 57 626 nut
- 461 symboli unikalnych
- 40 długość sekwencji

A musical score for cello in treble clef, 8/4 time signature, and a tempo of quarter note = 100. The score consists of seven eighth notes followed by a double bar line.

Transpozycja o jeden interwał w góre

A transposed musical score for cello in bass clef, 8/4 time signature, and a tempo of quarter note = 100. The notes are eighth notes, starting with a sharp sign on the first note. The score consists of seven eighth notes followed by a double bar line.

# JS Bach - wyniki

100 epok

Błąd: 0.25

Dopasowanie: 94 %



A continuation of the musical score from the previous measure. The bass clef and 4/4 time signature remain. The key signature is one sharp. Measure number 3 is indicated above the staff. The music consists of a series of eighth and sixteenth note patterns, some with grace notes, separated by rests. Measure endings are marked with brackets and the number 3.

A continuation of the musical score from the previous measures. The bass clef and 4/4 time signature remain. The key signature is one sharp. Measure number 4 is indicated above the staff. The music consists of a series of eighth and sixteenth note patterns, some with grace notes, separated by rests. Measure endings are marked with brackets and the number 3.

# Nottingham dataset

- 1038 utworów
- transpozycja - 2 tonacje
- 597 546 nut
- 1028 symboli unikalnych
- długość sekwencji 50

100 epok

Błąd: 0.20

Dopasowanie: 90 %

# Muzyka klasyczna

- 29 utworów
- transpozycja – wszystkie tonacje
- 377 504 nut
- 32 923 symboli unikalnych
- długość sekwencji - 200

40 epok

Błąd: 1.4

Dopasowanie: 60 %

# Gdzie trenować? - Google Cloud

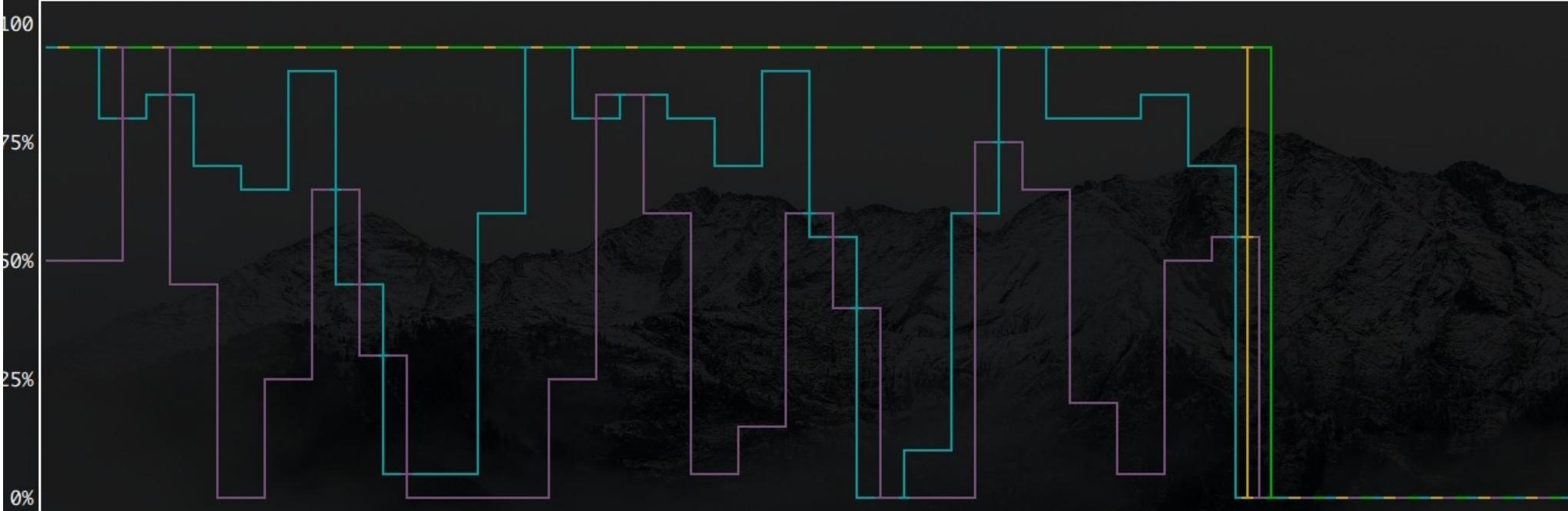
```
./+o+-  
yyyyy- -yyyyyy+  
://++++/-yyyyyyo  
.++ .:/++++/-+.+sss/^  
.:+o: /++++++/:-:/-  
o:+o:++. ``-.-/oo++++/  
.:+o:+o/. `+sssoo+/  
.++/+:+oo+o: `/sssooo.  
/++//+:`oo+o /::--:.  
\+/+o+++`o++o ++///.  
.++.o+++oo+: `/dddhhh.  
.+.o+oo:.` oddhhhh+  
\+.++o+o``-``.:ohdhhhhh+  
.:+o++ `ohhhhhhhhyo++os:  
.o: `syhhhhhhh/.oo++o`  
/osyyyyyyo++ooo++/  
````+oo++o\:  
.oo++.
```

**lukasz@instance-1**  
**OS:** Ubuntu 18.04 bionic  
**Kernel:** x86\_64 Linux 5.0.0-1029-gcp  
**Uptime:** 1d 18h 8m  
**Packages:** 919  
**Shell:** zsh 5.4.2  
**CPU:** Intel Xeon @ 4x 2.3GHz  
**GPU:** Tesla K80, Tesla K80  
**RAM:** 12055MiB / 15030MiB

# Gdzie trenować? - Google Cloud

Device 0 [Tesla K80] PCIe GEN 3@16x RX: N/A TX: N/A  
GPU 875MHz MEM 2505MHz TEMP 73°C FAN N/A% POW 143 / 149 W  
GPU[|||||195%] MEM[11.6G/12.0G]

Device 1 [Tesla K80] PCIe GEN 3@16x RX: N/A TX: N/A  
GPU 875MHz MEM 2505MHz TEMP 73°C FAN N/A% POW 108 / 149 W  
GPU[49%] MEM[11.6G/12.0G]



```

1 [|||||||||] 25.5%] Tasks: 39, 59 thr; 2 running
2 [|||||||] 8.2%] Load average: 0.91 0.99 1.00
3 [|||||||||] 31.5%] Uptime: 04:00:39
4 [|||||||||] 23.6%]
Mem[|||||||||||||] 12.2G/14.7G]
Swp[ 0K/0K]

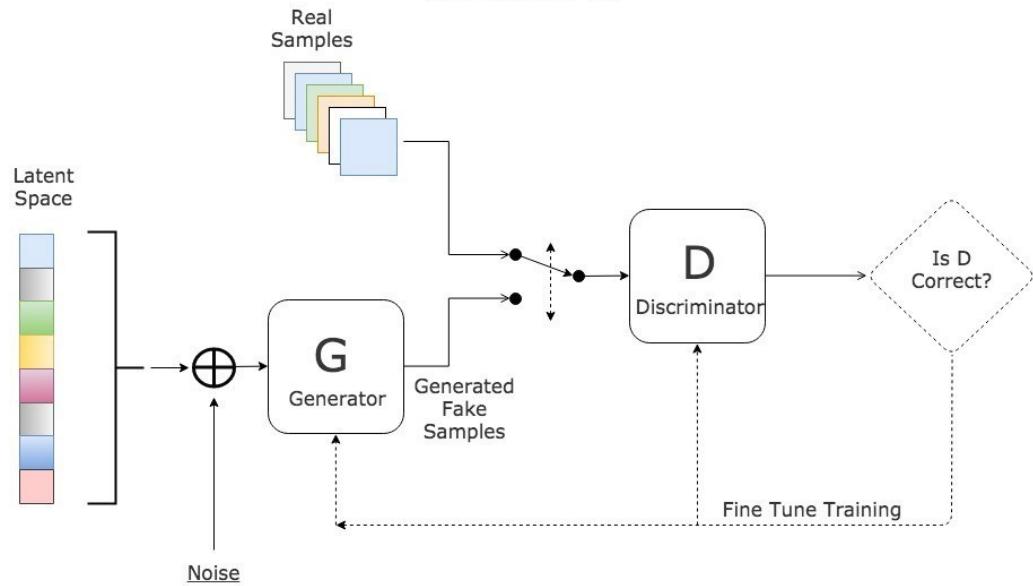
```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
4006	clouptest	20	0	848G	10.7G	1081M	R	92.3	73.1	1h32:05	python3 train.py -data processed/classical-200-seq
4076	clouptest	20	0	848G	10.7G	1081M	S	8.8	73.1	5:09.66	python3 train.py -data processed/classical-200-seq
4062	clouptest	20	0	848G	10.7G	1081M	S	6.7	73.1	4:17.28	python3 train.py -data processed/classical-200-seq
4074	clouptest	20	0	848G	10.7G	1081M	S	6.1	73.1	5:09.48	python3 train.py -data processed/classical-200-seq
4017	clouptest	20	0	848G	10.7G	1081M	S	4.0	73.1	1:18.76	python3 train.py -data processed/classical-200-seq
4075	clouptest	20	0	848G	10.7G	1081M	S	3.4	73.1	5:12.60	python3 train.py -data processed/classical-200-seq
4019	clouptest	20	0	848G	10.7G	1081M	S	3.4	73.1	1:18.30	python3 train.py -data processed/classical-200-seq
4016	clouptest	20	0	848G	10.7G	1081M	S	2.7	73.1	1:18.69	python3 train.py -data processed/classical-200-seq
4017	clouptest	20	0	848G	10.7G	1081M	S	4.0	72.9	1:18.70	python3 train.py -data processed/classical-200-seq
4064	clouptest	20	0	848G	10.7G	1081M	S	3.4	72.9	4:02.68	python3 train.py -data processed/classical-200-seq
4076	clouptest	20	0	848G	10.7G	1081M	S	1.3	72.9	5:09.53	python3 train.py -data processed/classical-200-seq
5148	clouptest	20	0	32968	5036	3644	R	1.3	0.0	0:00.17	htop
4061	clouptest	20	0	848G	10.7G	1081M	S	0.7	72.9	0:30.15	python3 train.py -data processed/classical-200-seq
4063	clouptest	20	0	848G	10.7G	1081M	S	0.0	72.9	0:26.08	python3 train.py -data processed/classical-200-seq
1787	clouptest	20	0	105M	4380	3364	S	0.0	0.0	0:02.31	sshd: clouptestexamples@pts/0
4050	clouptest	20	0	848G	10.7G	1081M	S	0.0	72.9	0:01.65	python3 train.py -data processed/classical-200-seq
4051	clouptest	20	0	848G	10.7G	1081M	S	0.0	72.9	0:01.66	python3 train.py -data processed/classical-200-seq
4048	clouptest	20	0	848G	10.7G	1081M	S	0.0	72.9	0:00.14	python3 train.py -data processed/classical-200-seq

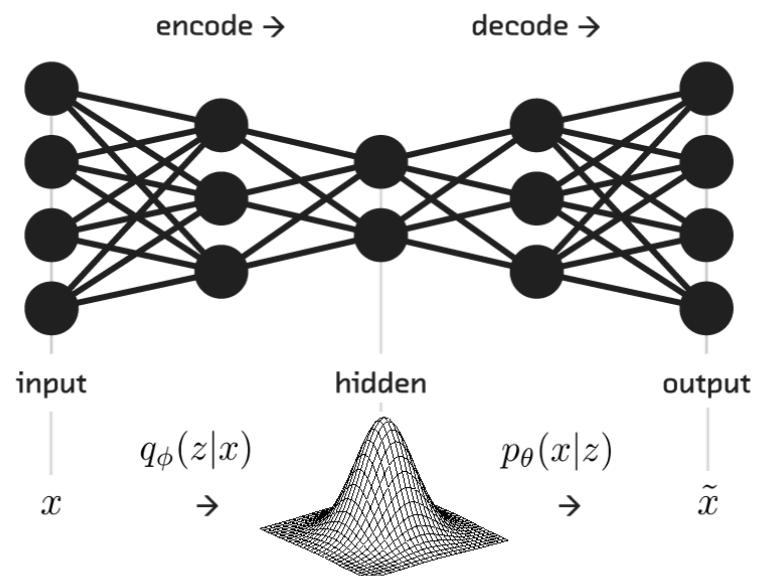
# Tips

1. Zapis wag przez checkpointy
2. Sprawdzenie modelu predykcyjnego na początkowym zestawie wag
3. Użycie TensorBoard do obserwowania działania sieci
4. Rozpoczynanie z mały datasetem
5. Wykorzystanie CUDA przez warstwy cuDNNLSTM

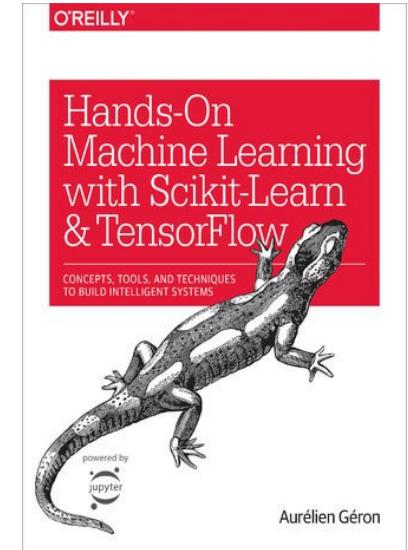
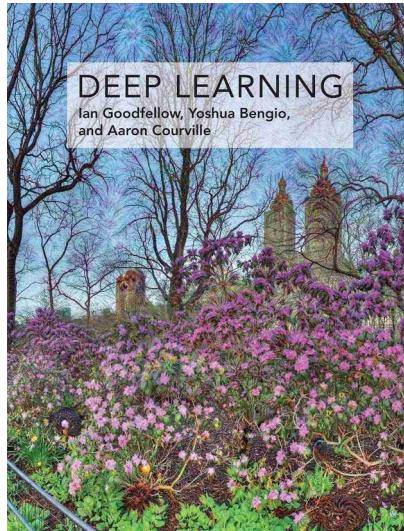
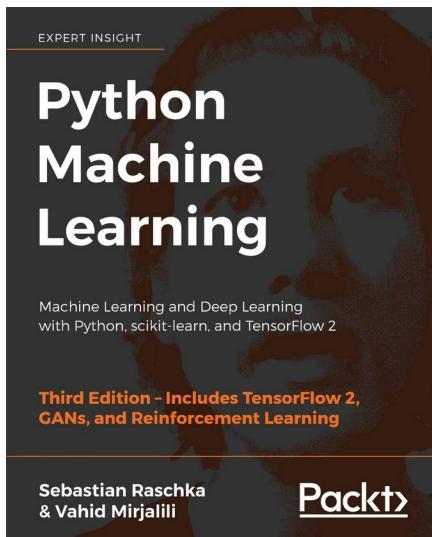
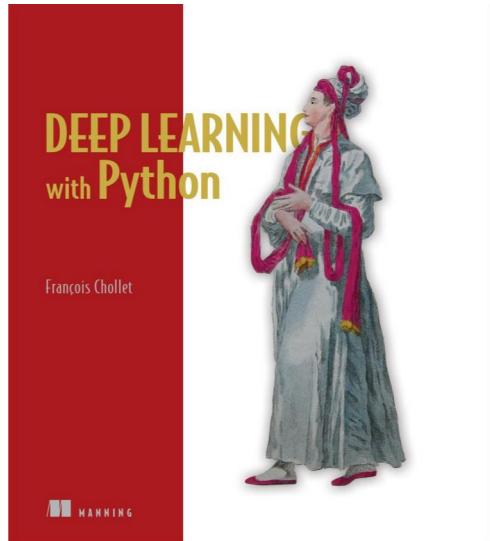
## Generative Adversarial Network



## Variational autoencoder



# Literatura



# Dziękuję za uwagę

<https://blog.lukaszogan.com/informatyka/algorytmiczne-generowanie-muzyki-poczatek-cyku/>

<https://blog.lukaszogan.com/informatyka/ukryte-modele-markowa-w-progresji-jazzowej-ii-v-i/>

<https://blog.lukaszogan.com/informatyka/rekurencyjne-sieci-neuronowe-generuja-muzyke/>