



POLITECHNIKA
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Uczenie głębokie

Wykład 1: Uczenie głębokie i sieci splotowe - wprowadzenie

Jacek Rumiński
Katedra Inżynierii Biomedycznej, Wydział ETI



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu:

- **Uczenie głębokie - wprowadzenie do przedmiotu i tematu**
- Klasy głębokich sieci neuronowych
- Operacja splotu 1D i sieci neuronowe
- Podsumowanie

Wprowadzenie - omówienie przedmiotu

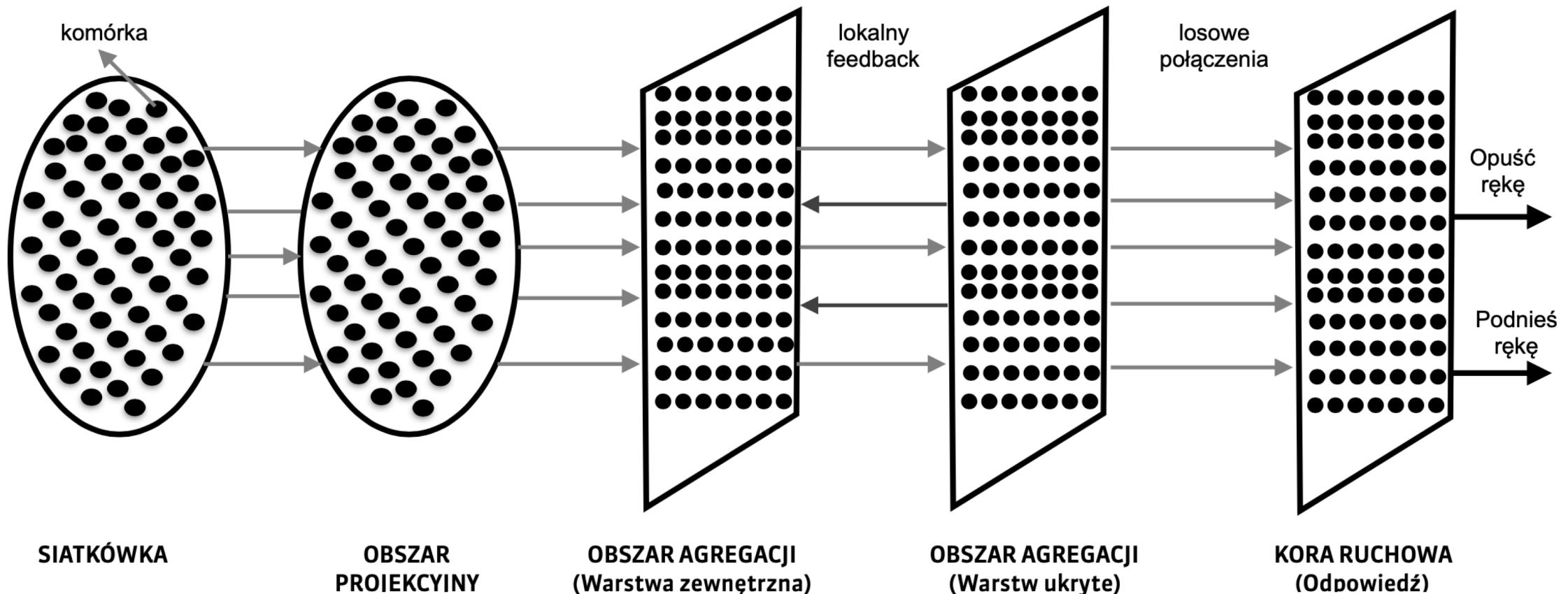
- Treści przedmiotu
- Organizacja przedmiotu
- Warunku zaliczenia
- Materiały do wykładu – slajdy oraz notatniki
- Literatura i materiały uzupełniające (m.in. Książki, kursy S. Raschka, A. Canziani, dokumentacja TF i PyTorch, <https://paperswithcode.com/sota>, itp.)

Uczenie głębokie (ang. deep learning) to:

- rodzaj uczenia maszynowego wykorzystujący modele złożone z wielu warstw (sztucznych sieci neuronowych),
- „*Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers.*” [IBM, <https://www.ibm.com/cloud/learn/deep-learning>, dostęp 2021]
- „*Deep Learning is a machine learning technique that constructs artificial neural networks to mimic the structure and function of the human brain. In practice, deep learning, also known as deep structured learning or hierarchical learning, uses a large number hidden layers -typically more than 6 but often much higher - of nonlinear processing to extract features from data and transform the data into different levels of abstraction (representations).*” [DeepAI, <https://deeppai.org/machine-learning-glossary-and-terms/deep-learning>, dostęp 2021]
- Analiza pojęcia „deep learning” w [W.J.Zhang i inni, On Definition of Deep Learning, <https://doi.org/10.23919/WAC.2018.8430387>]

Sztuczne sieci neuronowe - trochę historii

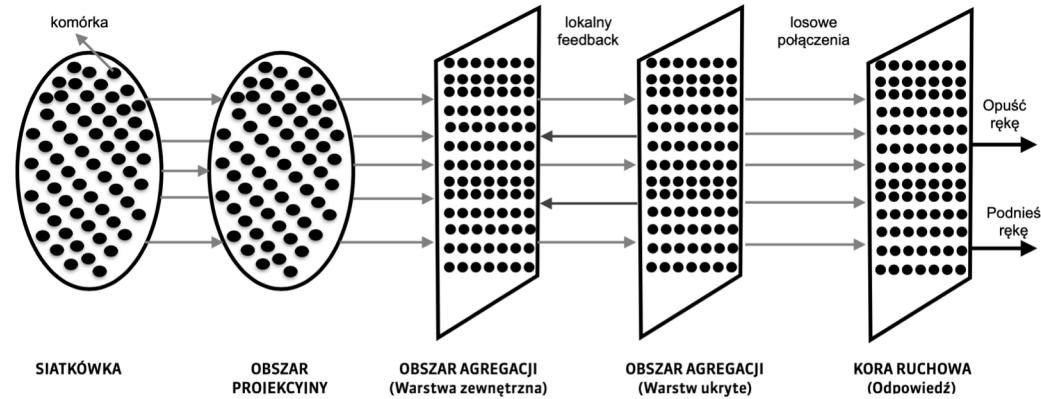
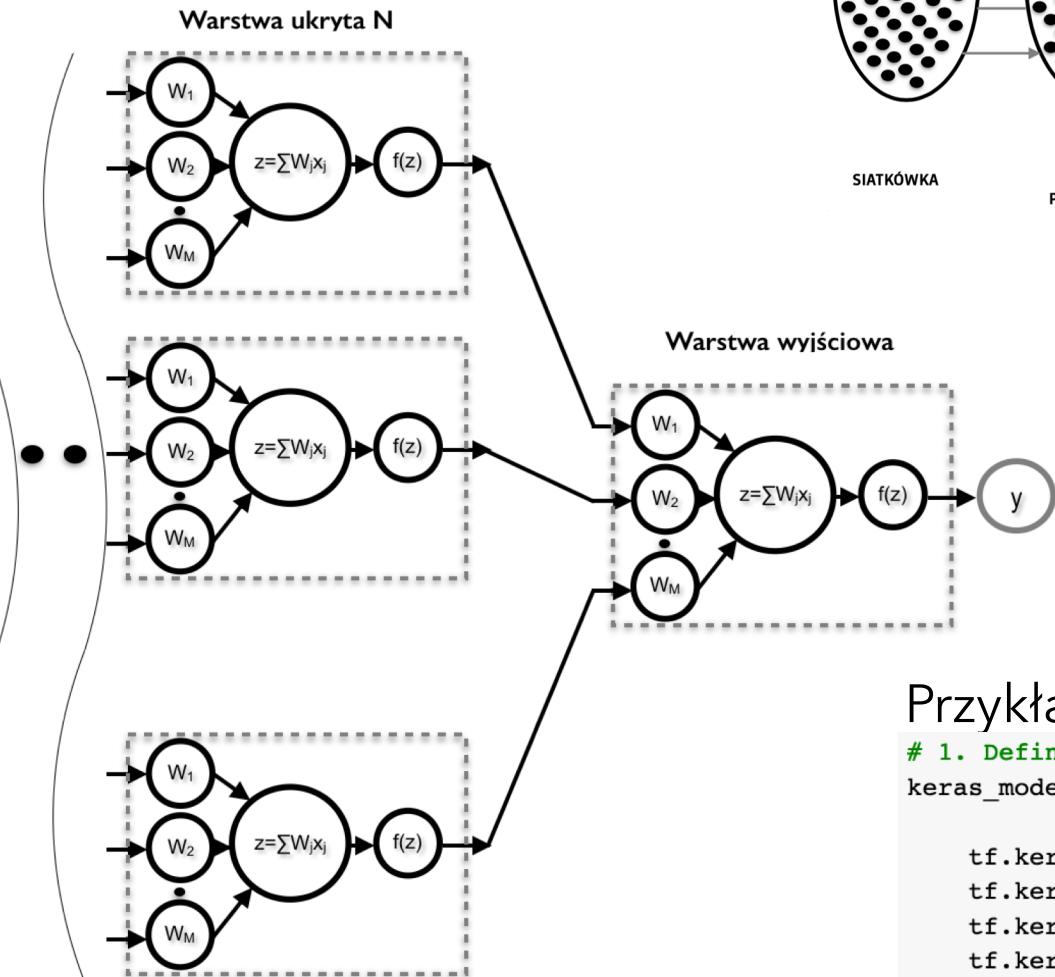
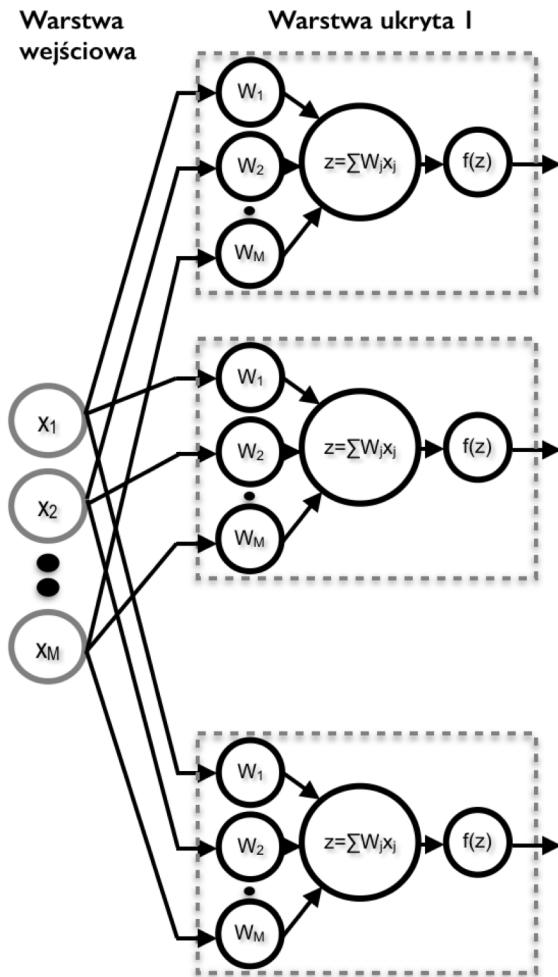
Ogólna postać perceptronu Rosenblatta wykorzystująca WIELE WARSTW komórek.



Inspirowane: F. Rosenblatt, "Perceptron Simulation Experiments," in Proceedings of the IRE, vol. 48, no. 3, pp. 301-309, March 1960.

Sztuczne sieci neuronowe

Muli Layer Perceptron



Przykładowy model w TensorFlow:

```
# 1. Define and use the Model
keras_model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(784, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(no_of_classes, activation='softmax')
])
```

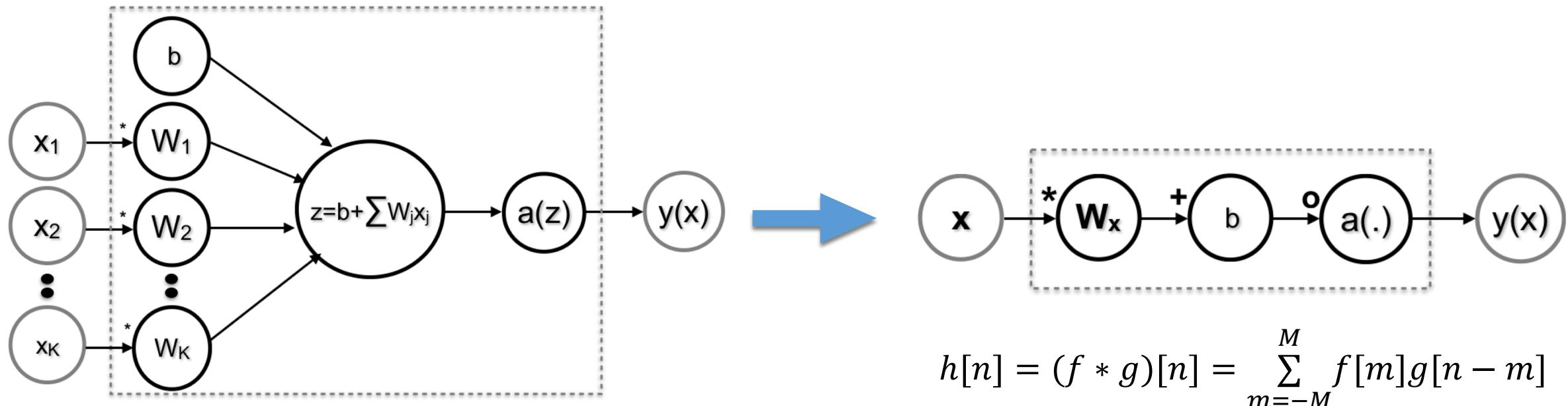
Plan wykładu:

- Uczenie głębokie – wprowadzenie do przedmiotu i tematu
- **Klasy głębokich sieci neuronowych**
- Operacja splotu 1D i sieci neuronowe
- Podsumowanie

Sieci splotowe (ang. Convolutional Neural Networks)

Klasa sieci neuronowych, w których węzły realizują (w przybliżeniu) operację splotu w celu wyodrębnienia z danych wejściowych nowej reprezentacji danych (cech) tak, aby uzyskać optymalne rozwiązanie określonego problemu uczenia maszynowego (np. klasyfikacja, regresja, itp.).

W trakcie treningu poszukiwane są wagi (W , b) sygnału (funkcji), która jest splatana z danymi wejściowymi.

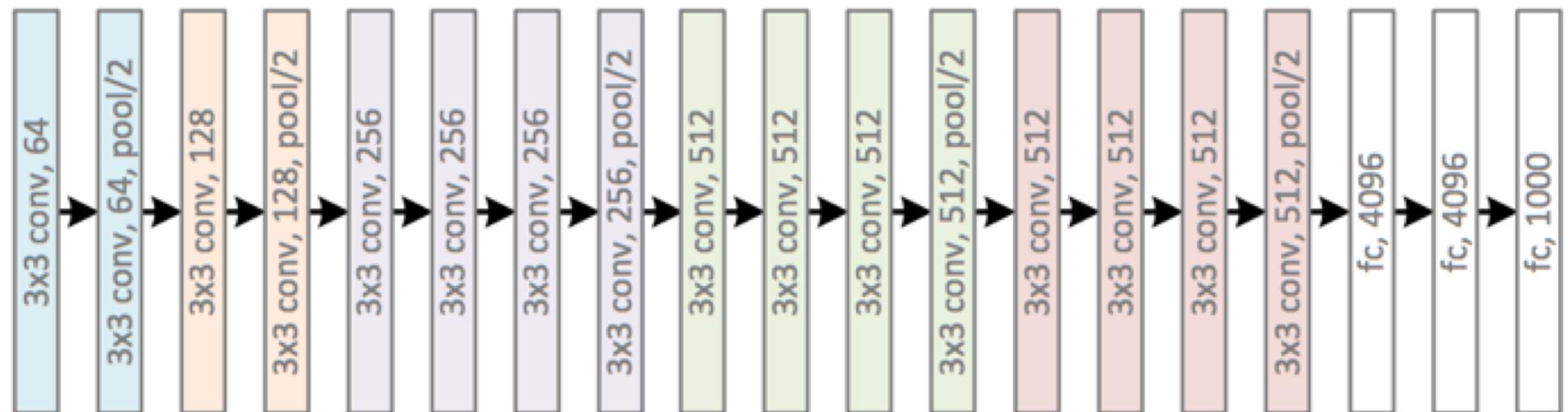


Sieci splotowe (ang. Convolutional Neural Networks)

W praktyce stosuje się wielokrotnie operację splotu, co prowadzi do wykorzystania wielu warstw ukrytych (model głęboki) realizujących wyodrębnianie cech w kierunku nowej reprezentacji.

Ostatecznie, nowa reprezentacja danych jest często sprowadzana wektora 1D, którego dane podawane są do sieci np. typu MLP (modele typu fully connected) prowadząc do uzyskania końcowego wyniku zgodnie z danym celem uczenia maszynowego.

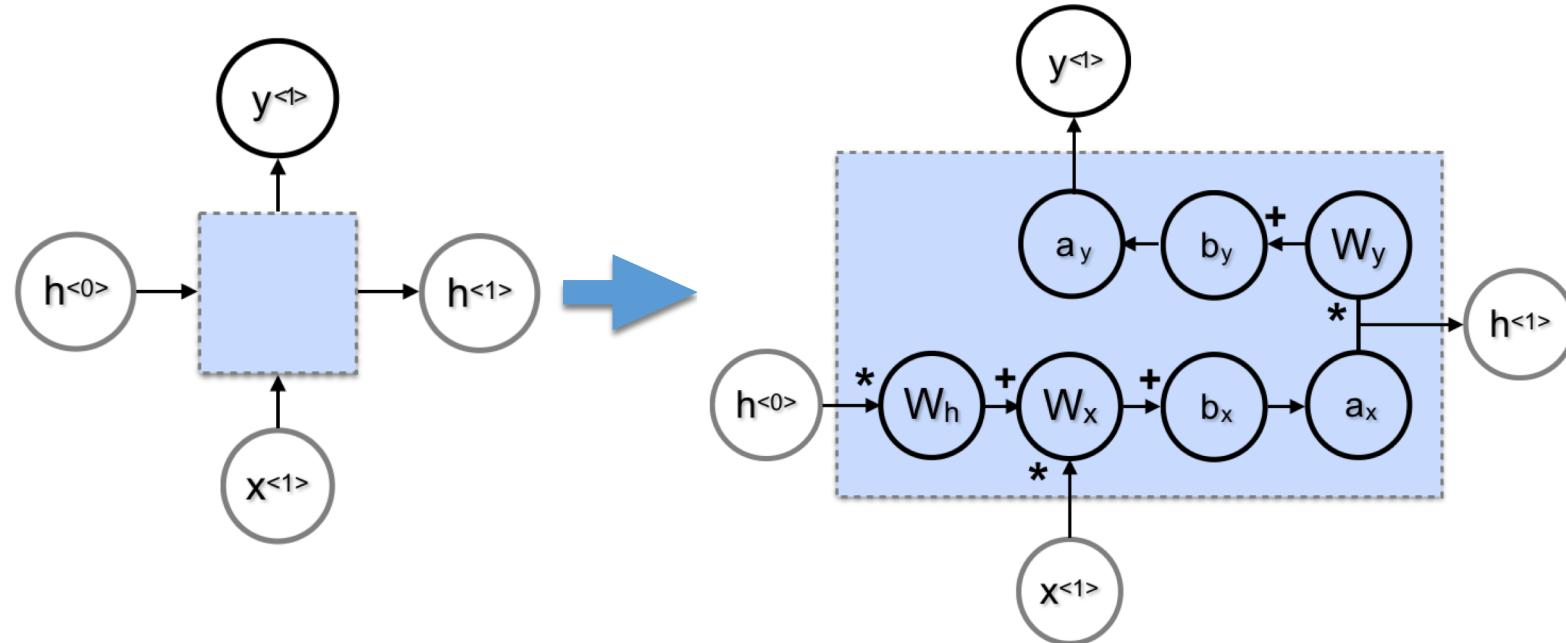
VGG 2014



Sieci rekurencyjne (ang. RNN - Recurrent Neural Network)

Klasa sieci neuronowych, w ramach których połączenia pomiędzy węzłami tworzą graf skierowany dla sekwencji kolejnych próbek danych (w czasie).

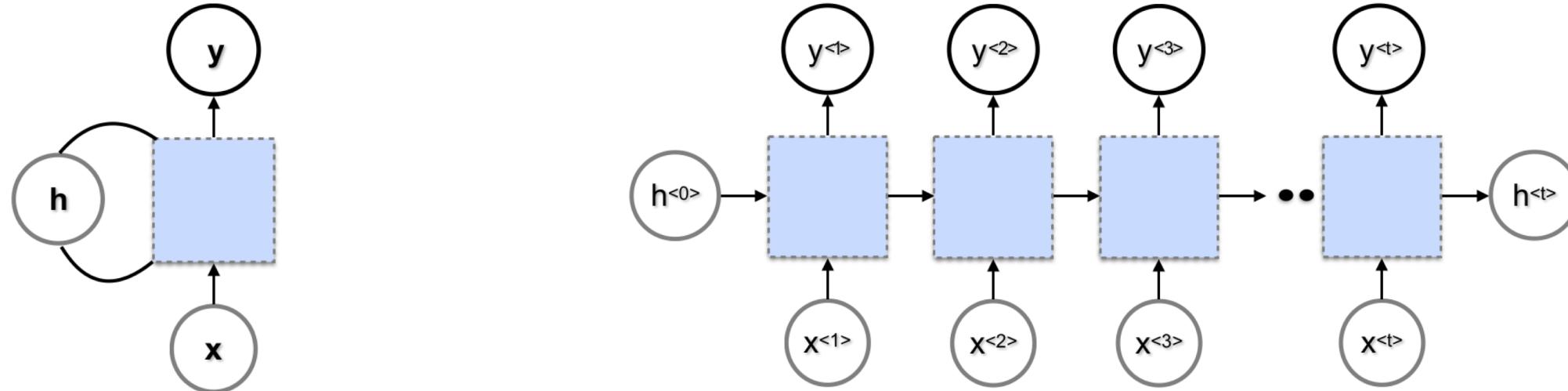
W RNN stan ukryty poprzedniego węzła sieci (lub wyjście) wpływa na wejście kolejnego węzła sieci.



$$\begin{aligned} h^{<1>} &= a_x(W_h h^{<0>} + W_x x^{<1>} + b_x) \\ y^{<1>} &= a_y(W_y h^{<1>} + b_y) \end{aligned}$$

Sieci rekurencyjne (ang. RNN – Recurrent Neural Network)

W rekurencyjnych sieciach neuronowych stosowane są połączenia rekurencyjne dla kolejnych próbek w czasie, co można zaprezentować (po swoistym „rozwinięciu” w czasie ang. unfold) jako sekwencję warstw.



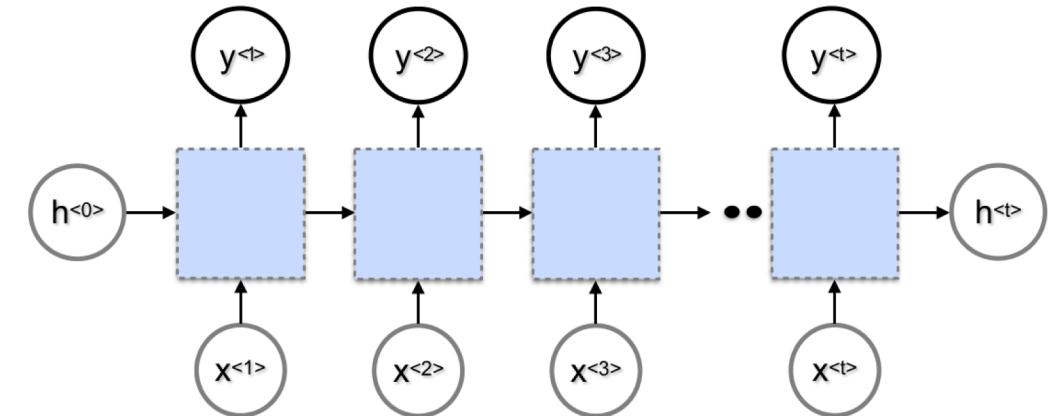
Taka budowa sieci wskazuje na podstawowy rodzaj danych jaki wykorzystywany jest w RNN: sekwencje danych (np. sygnał EKG, zdanie, utwór muzyczny, itp.).

Sieci rekurencyjne (ang. RNN - Recurrent Neural Network)

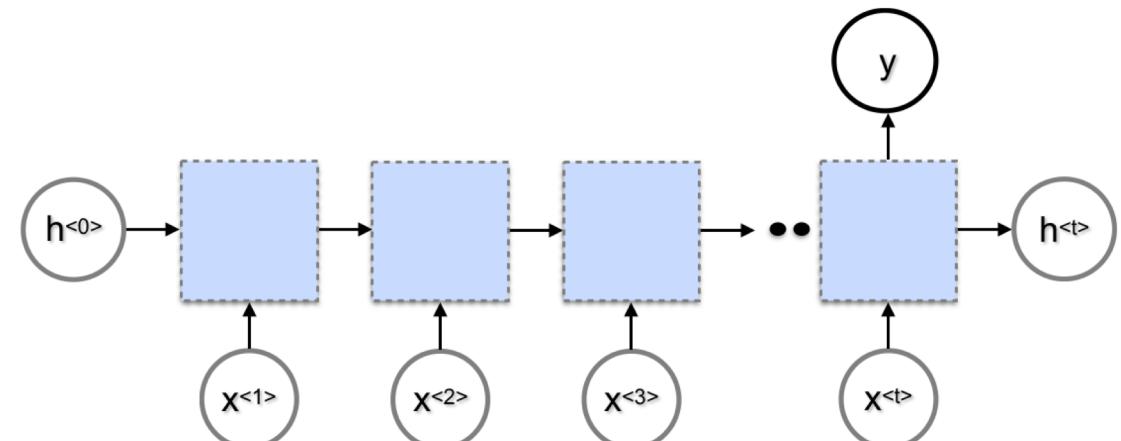
Budowa modelu sieci RNN zależy najczęściej od zastosowania. Przykładowo zilustrowany obok model reprezentuje odwzorowanie wiele-do-wielu (ang. many to many) i może być zastosowany np. w rozpoznawaniu/etykietowaniu pojęć w tekście. Inna wersja modeli wiele-do-wielu mogą np. służyć do tłumaczenia tekstu.

Modele wiele-do-jednego (przykład obok) są wykorzystywane do problemów klasyfikacji (np. klasyfikacji opinii, ang. sentymet classification).

Natomiast modele jeden-do-wielu stosowane są do generacji sekwencji, np. muzyki.



RNN typu wiele-do-wielu



RNN typu wiele-do-jednego

Sieci rekurencyjne (ang. RNN – Recurrent Neural Network)

W uczeniu modeli RNN pojawiają się często problemy typu zanikającego lub narastającego gradientu co prowadzi do braku poprawnie wyuczonej sieci.

Z tych względów (oraz ze względu na inne własności, które poznamy później) opracowano specjalne wersje modeli RNN typu:

- Gated Recurrent Unit (GRU)
- Long Short-Term Memory (LSTM)

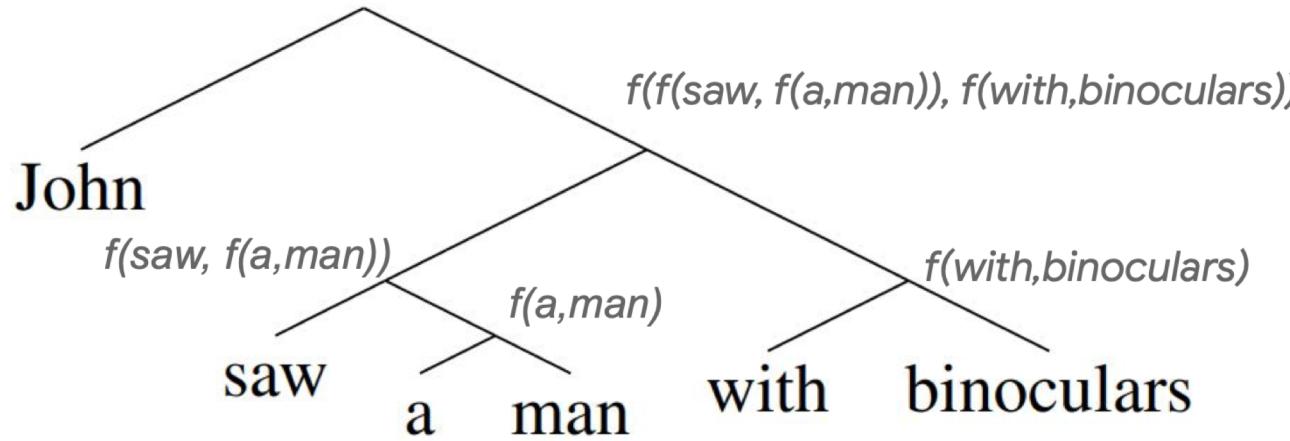
Modele te charakteryzują się możliwością pamiętania lub zapominania zdarzeń, które wystąpiły w (wielu) poprzednich momentach w czasie. Zdolność taka implementowana jest przez szereg różnego rodzaju „bramek” (ang. gates) reprezentowane przez funkcje typu:

$$\Gamma = a_x(W_x x^{<t>} + W_u h^{<t-1>} + b_u)$$

Czyli rozbudowywane są węzły sieci neuronowej tworząc model jeszcze bardziej głęboki.

Modele RvNN (ang. Recursive Neural Networks) - uogólnienie modeli RNN

$f(John, f(f(saw, f(a,man)), f(with,binoculars)))$



Źródło: [1]

Przykładowo model RvNN wykorzystuje funkcję kompozycji - f - (komórka [sieć] rekurencyjna), której celem jest utworzenie reprezentacji (kompozycji) węzłów dzieci np. (r_1) i (r_2) w węźle rodzica $f(r_1, r_2)$.

Rekurencyjność polega na takim (rekurencyjnym) przetworzeniu każdego z węzłów, aż dojdziemy do węzła-korzenia (ang. root node).

Mając nową reprezentację w węźle rodzica możemy nie potrzebować reprezentacji węzłów dzieci. Wprowadzane jest czasami pojęcie (parametr) prawdopodobieństwa egzystencji określające czy reprezentacja r_i wymaga dalszego przetwarzania ($e_i=1$) lub może być zapomniana ($e_i=0$). [2]

[1] <https://icml.cc/media/icml-2021/Slides/8993.pdf> Continuous RNN – uczenie struktury i funkcji kompozycji

[2] <https://arxiv.org/pdf/2106.06038.pdf> Modelowanie struktur hierarchicznych

Inne realizacje modeli głębokich:

- Autokodery i modele generacyjne (m.in. GAN, np. <https://arxiv.org/abs/2103.05180>),
- Uczenie ze wzmacnieniem (np. <https://arxiv.org/abs/2106.01151>),
- Sieci typu Deep Belief Network (DBN) - (np. <https://arxiv.org/abs/2107.12521>),
- Warianty kombinacyjne różnych sieci (np. Convolutional DBN, Recursive CNNs, Hierarchical Recurrent Neural Networks, Graph Neural Networks, itp.).

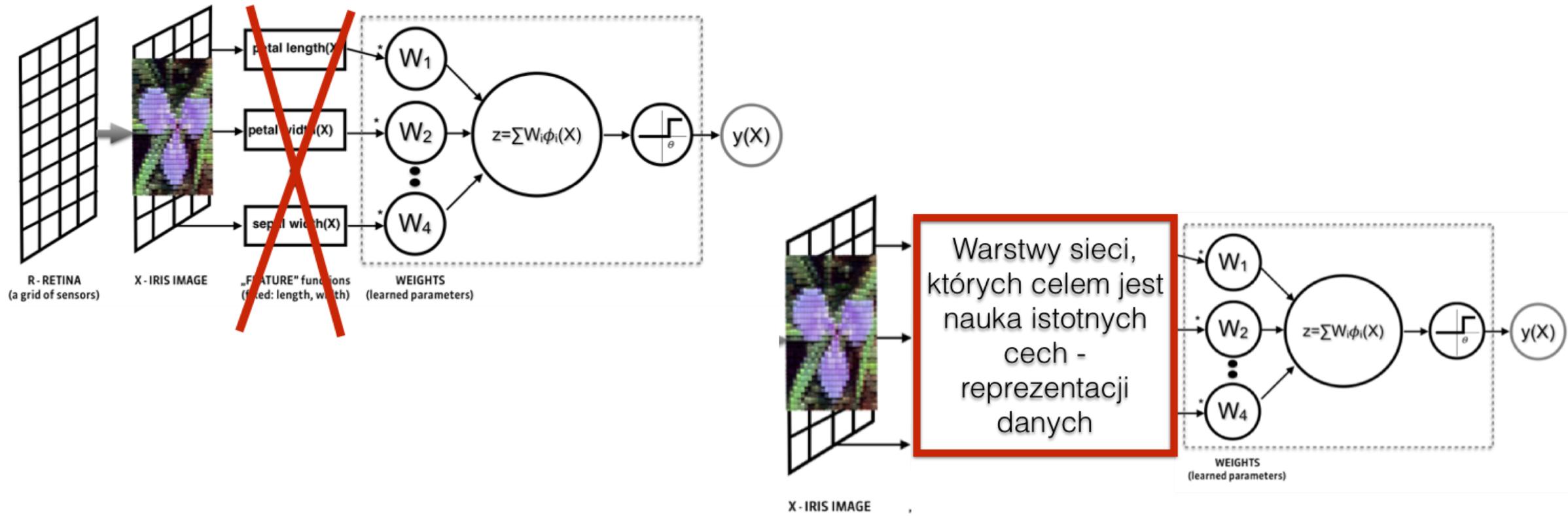
W ramach bieżącego materiału skupimy się na sieciach splotowych, rekurencyjnych, autokoderach, modelach generacyjnych oraz na uczeniu ze wzmacnieniem.

Plan wykładu:

- Uczenie głębokie – wprowadzenie do przedmiotu i tematu
- Klasy głębokich sieci neuronowych
- **Operacja splotu 1D i sieci neuronowe**
- Podsumowanie

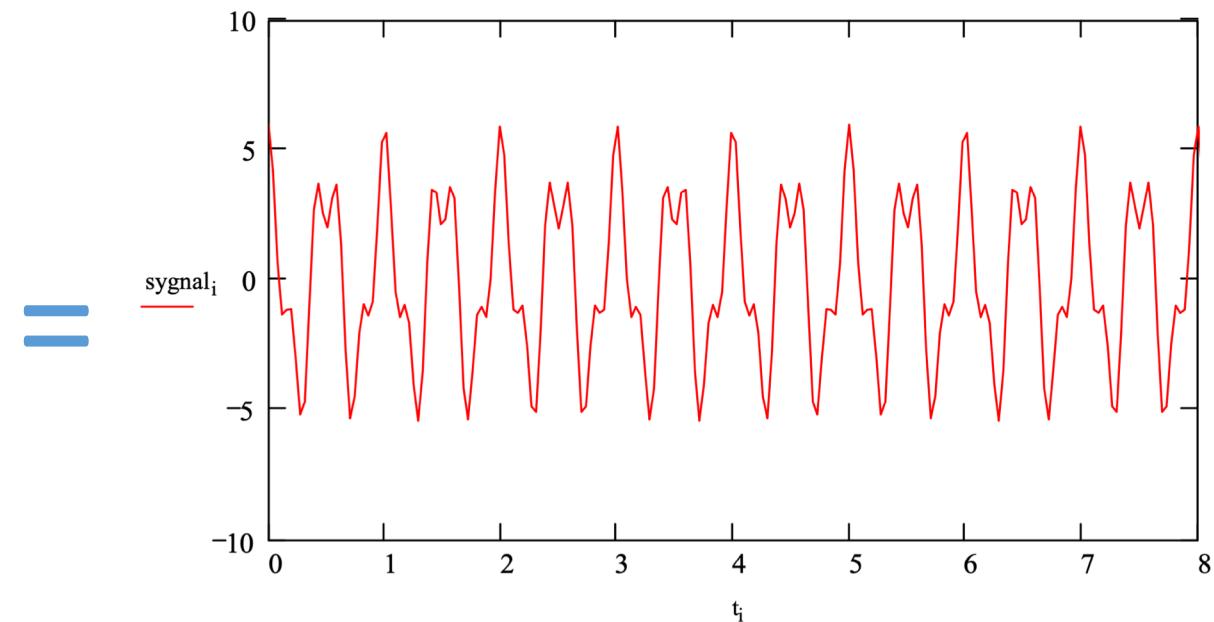
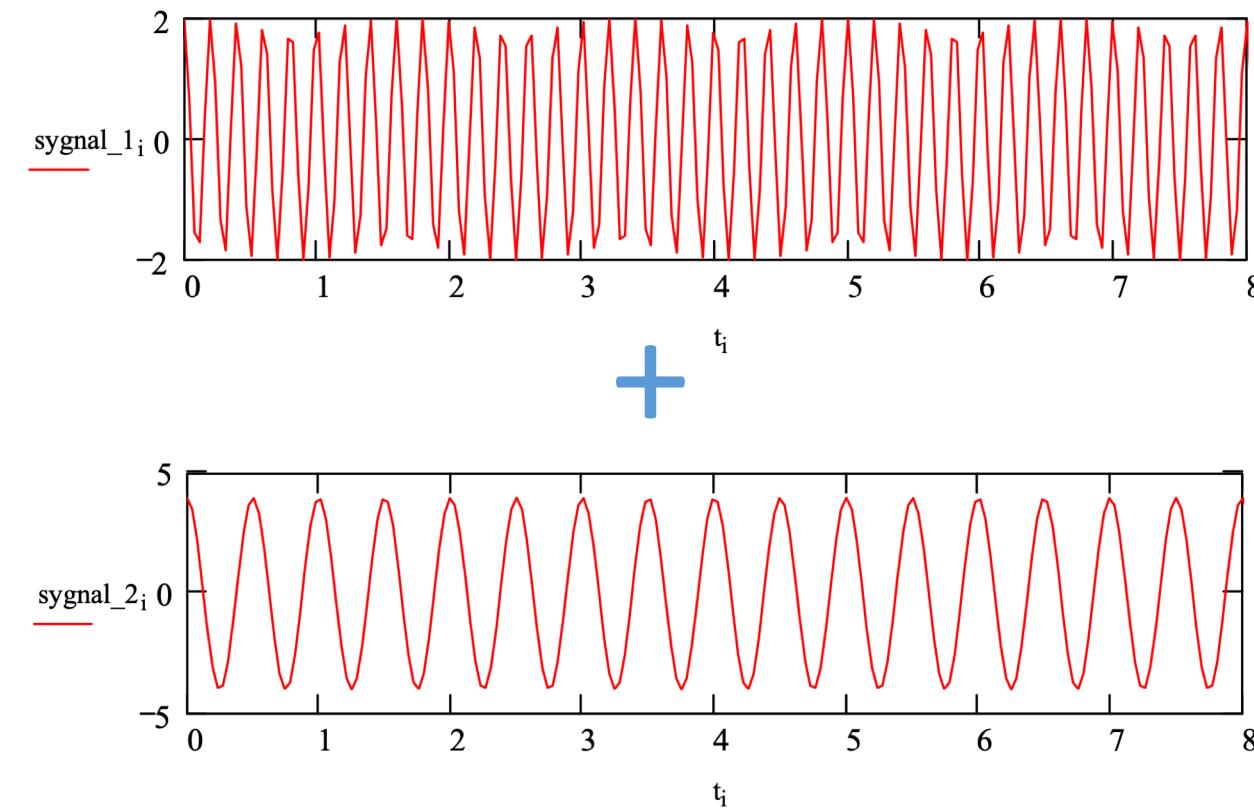
Sieci splotowe (ang. Convolutional Neural Networks)

Zastosowanie operacji splotu do zmiany reprezentacji danych - chcemy, żeby sieć sama nauczyła się jak wyodrębniać istotne cechy z danych surowych w celu osiągnięcia jak najlepszego rezultatu końcowego, np. w problemie klasyfikacji danych, regresji, itp.



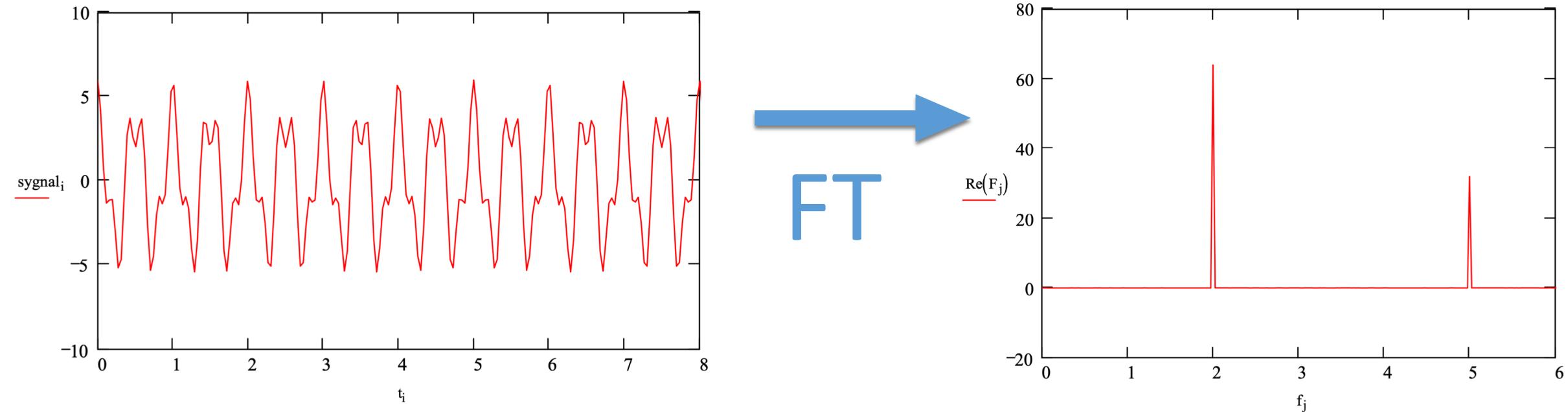
Splot (ang. convolution) - przypomnienie lub wprowadzenie

Posłużmy się prostym przykładem dwóch jednowymiarowych sygnałów cyfrowych (każdy o 1024 próbkach). Pierwszy sygnał ma częstotliwość 5Hz, drugi 2Hz. Dodajmy je do siebie ...



Splot (ang. convolution)

Obserwując wynik dodania sygnałów trudno „zobaczyć” części składowe i ich częstotliwości. Dlatego stosujemy transformację Fouriera, która przekształca dane do dziedziny częstotliwości.

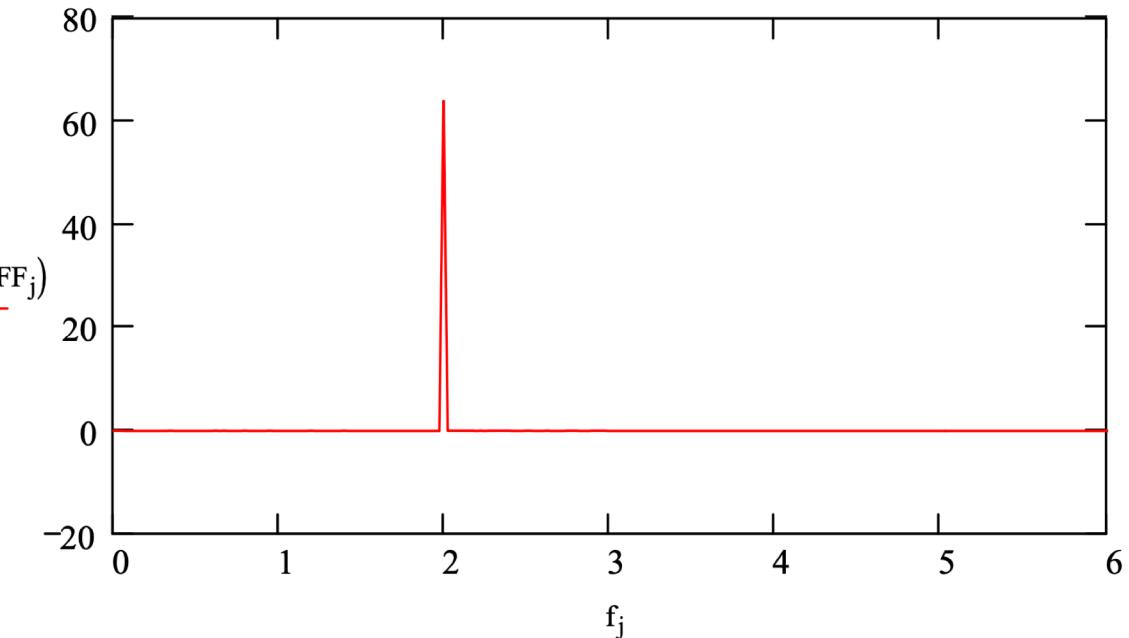
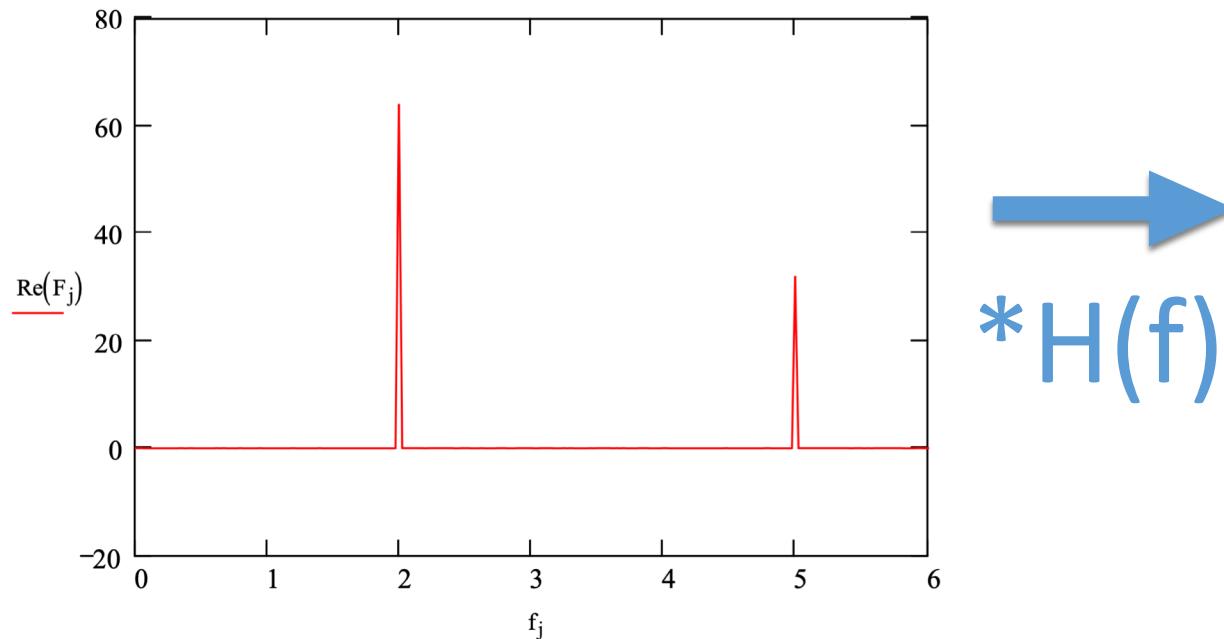


W dziedzinie częstotliwości moglibyśmy pomnożyć dane (liczby), np. przez 1 jeśli częstotliwość jest mniejsza niż 3Hz lub przez 0 w innym przypadku.

W rezultacie usuniemy w naszym przykładzie komponent o częstotliwości 5Hz.

Splot (ang. convolution)

Usunęliśmy wybrane komponenty częstotliwości na podstawie przyjętych współczynników mnożenia.

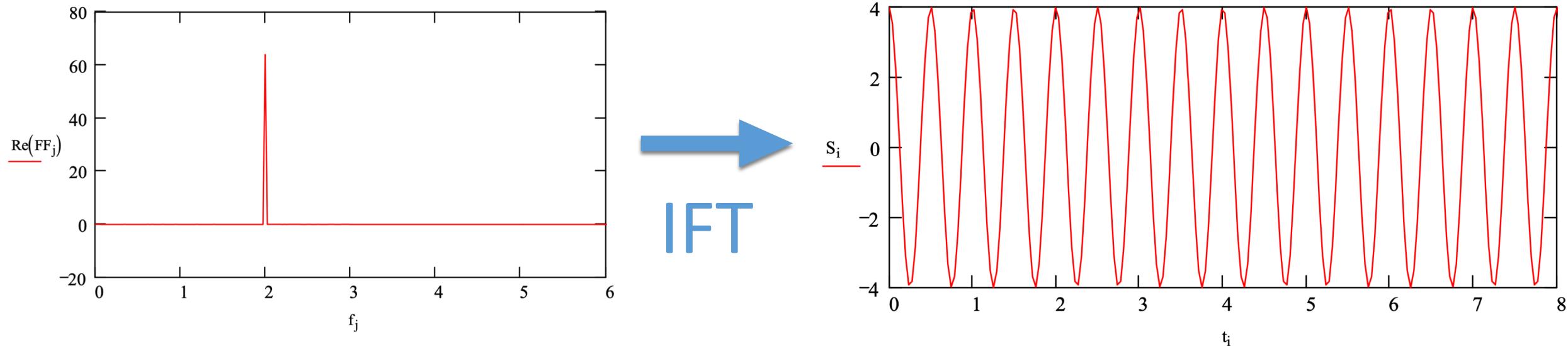


Zmodyfikowane dane w dziedzinie częstotliwości możemy przekształcie do dziedziny czasu stosując odwrotną transformację Fouriera.

W rezultacie otrzymamy przefiltrowany sygnał w dziedzinie czasu (filtr dolnoprzepustowy).

Splot (ang. convolution)

Przypomnieliśmy właśnie uproszczoną wersję metody filtracji sygnałów z wykorzystaniem mnożenia w dziedzinie częstotliwości.



WAŻNE:

Mnożenie w dziedzinie częstotliwości jest równoważne operacji splotu w dziedzinie czasu.

Możemy zatem przeprowadzić filtrację poprzez splot sygnału wejściowego z odpowiednio dobranym „sygnałem” filtru.

Splot (ang. convolution)

Splot dwóch sygnałów (w ogólności funkcji) f i g dany jest wzorem:

$$h[n] = (f * g)[n] = \sum_{m=-M}^{M} f[m]g[n-m]$$

lub alternatywnie:

$$h[n] = (f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m]$$

(jeśli sygnał g jest ograniczony do $2M+1$ próbek).

Wynik wyznaczamy obliczając wartość dla każdego n (n - liczba próbek sygnału wejściowego).

Splot (ang. convolution)

W ramach operacji splotu niejako mieszamy dwa sygnały: np. sygnał wejściowy f z innym sygnałem g (np. maską, jądrem - ang. kernel, itp.). Warto zauważyć operację odejmowania indeksów obecną w obu zapisach operacji splotu. Oznacza ona, że sygnał np. jądra musimy przekształcić geometrycznie - odbić o 180 stopni.

Jeśli wartości maski/jądra są symetryczne ułożone, to nie musimy dokonywać operacji odbicia. Wówczas wynik splotu będzie taki sam jak dla korelacji wzajemnej (ang. cross correlation):

$$h[n] = (f \otimes g)[n] = \sum_{m=-M}^M f[m]g[n+m]$$

Warto zauważyć, że we współczesnych bibliotekach oprogramowania dla uczenia głębokiego (np. TensorFlow czy PyTorch) operacje opisywane jako splot (np. Conv2D w TF, Conv2d w PyTorch) to w rzeczywistości operacje korelacji wzajemnej.

Splot (ang. convolution)

Sygnal **f** (5 próbek)

f0	f1	f2	f3	f4
----	----	----	----	----

g - kernel/maska (3 próbki)

g0	g1	g2
----	----	----

Na przykład:

1	10	1	1	1
---	----	---	---	---

1/3	1/3	1/3
-----	-----	-----

$$h[n] = (f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m]$$

Rozmiar maski jest 3, więc M = 1, m = {-1, 0, 1}, f = [1, 10, 1, 1, 1], g = [1/3, 1/3, 1/3]

Krok n = 0:

$$h[0] = f[0 - (-1)]g[-1] + f[0 - (0)]g[0] + f[0 - (+1)]g[1]$$

$$h[0] = f[1]g[-1] + f[0]g[0] + f[-1]g[1] \rightarrow \text{NIE MA } f[-1]!!! \rightarrow \text{POMIJAMY } n=0$$

Krok n = 1:

$$h[1] = f[1 - (-1)]g[-1] + f[1 - (0)]g[0] + f[1 - (+1)]g[1]$$

$$h[1] = f[2]g[-1] + f[1]g[0] + f[0]g[1]$$

$$h[1] = 1 * 1/3 + 10 * 1/3 + 1 * 1/3 = 12/3 = 4$$

...

Krok n = 4:

$$h[1] = f[4 - (-1)]g[-1] + f[4 - (0)]g[0] + f[4 - (+1)]g[1]$$

$$h[1] = f[5]g[-1] + f[4]g[0] + f[3]g[1]$$

$$\text{NIE MA } f[5]!!! \rightarrow \text{POMIJAMY } n=4$$

Splot (ang. convolution)

Sygnal **f** (5 próbek)

1	10	1	1	1
---	----	---	---	---

g - kernel/maska (3 próbki)

1/3	1/3	1/3
-----	-----	-----

Wynik splotu:

4	4	1
---	---	---

Obserwacje:

1. Sygnal wynikowy jest bardziej „wygładzony” (filtracja dolnoprzepustowa).
2. Sygnal wynikowy ma mniej probek niz wejsciowy. Na kazdym brzegu mniej o: $\text{int}(3/2)$.

W celu zachowania tego samego rozmiaru rozbudowujemy sygnal wejsciowy o nowe dane.

Uzupełniamy (dopełniamy, ang. padding) wartościami stałymi (np. 0) lub wartościami wynikającymi z przyjętych warunków brzegowych (np. okresowość sygnału, odbicie na brzegach, itp.).

Sygnal **f** (5 próbek + 2 próbki z 0)

0	1	10	1	1	1	0
---	---	----	---	---	---	---

kernel

1/3	1/3	1/3
-----	-----	-----

Wynik splotu

11/3	12/3	12/3	3/3	2/3
------	------	------	-----	-----

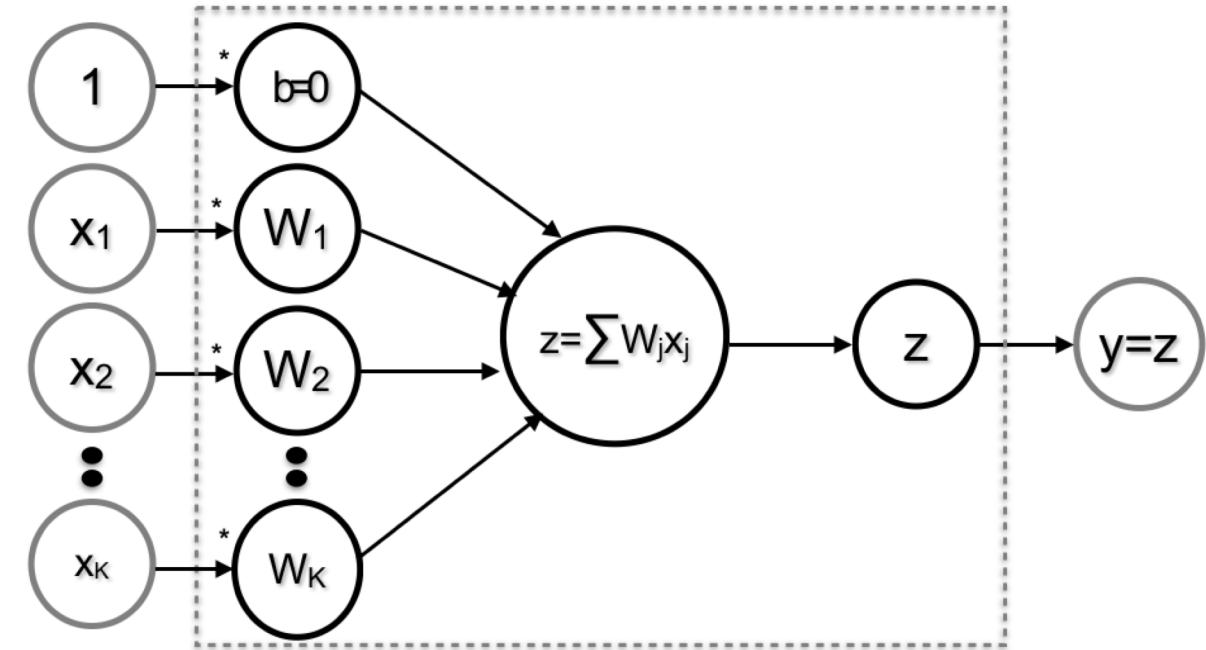
Splot i sieci neuronowe

$$h[n] = (f * g)[n] = \sum_{m=-M}^M f[m]g[n-m]$$

$$h[n] = (f \otimes g)[n] = \sum_{m=-M}^M f[m]g[n+m]$$

W uproszczenie (po spełnieniu pewnych założeń):

$$z[n + \text{int}(K/2)] = \sum_{m=0}^{K-1=2M} w[m]x[n+m]$$



Trenujemy model w celu wyznaczenia wag maski (jądra) dla operacji splotu.

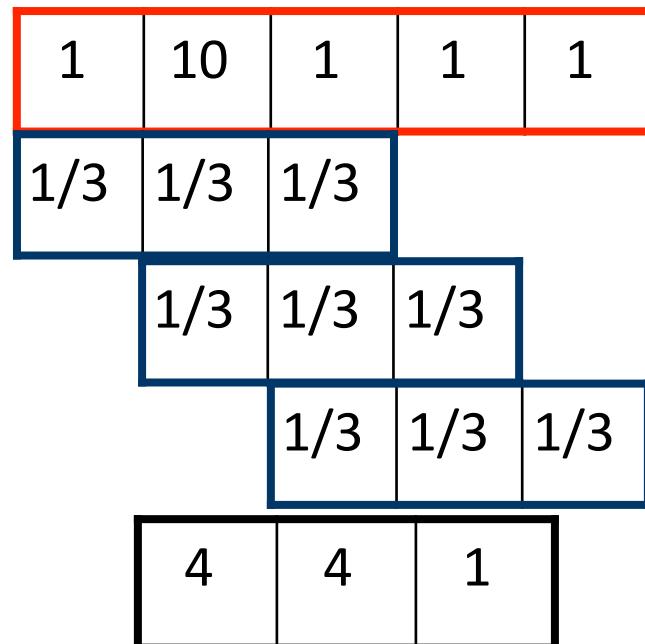
Dążymy do tego, aby w wyniku procesu uczenia model filtrował dane wejściowe w celu optymalnego wyboru cech.

Splot i sieci neuronowe (przykład dla danych 1D)

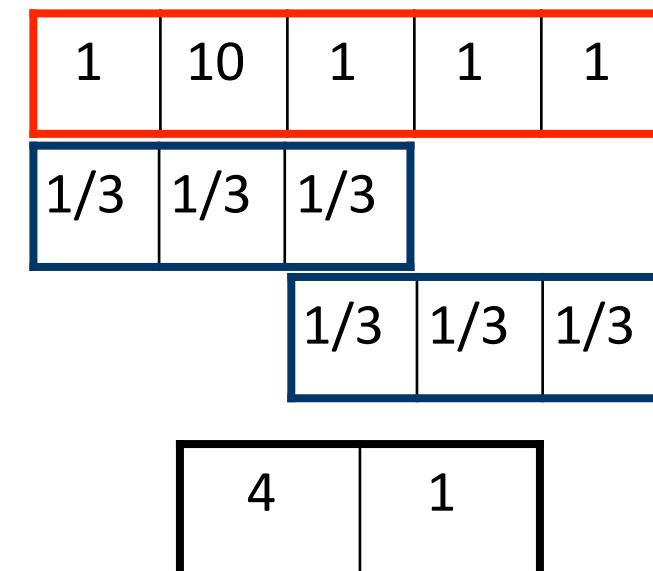
W celu zróżnicowania procesu uczenia nowej reprezentacji danych możemy wprowadzić dodatkowe zmiany w realizacji operacji splotu. Zmiany te będą opisywane poprzez parametry:

- Krok (ang. stride):

stride = 1



stride = 2

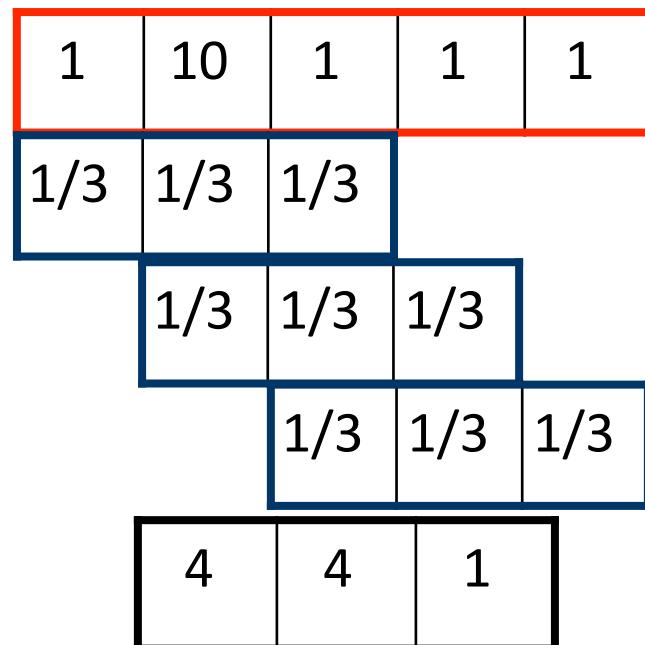


Splot i sieci neuronowe (przykład dla danych 1D)

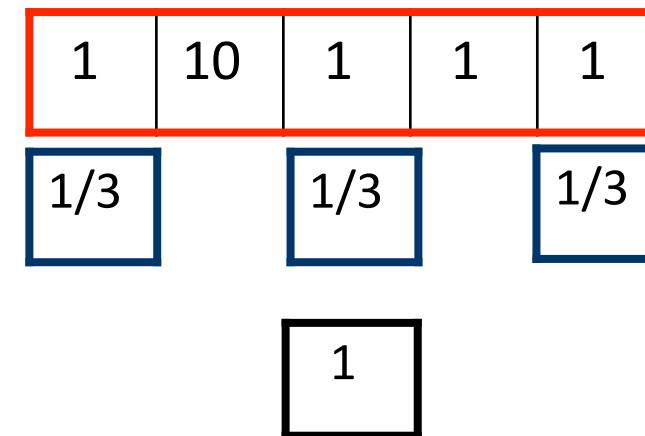
W celu zróżnicowania procesu uczenia nowej reprezentacji danych możemy wprowadzić dodatkowe zmiany w realizacji operacji splotu. Zmiany te będą opisywane poprzez parametry:

- stopień separacji (ang. dilation rate) - zmiana efektywnego położenia wartości maski splotu:

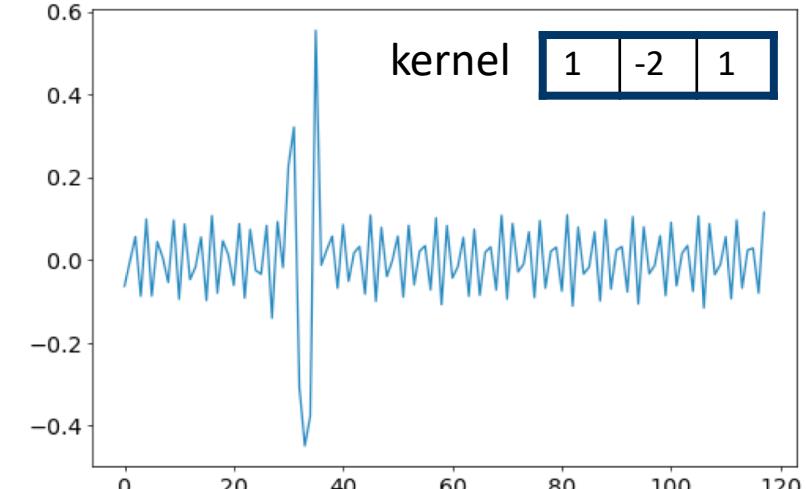
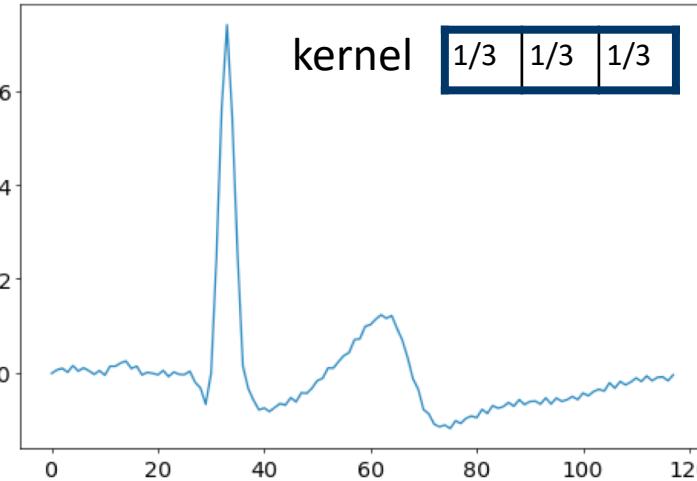
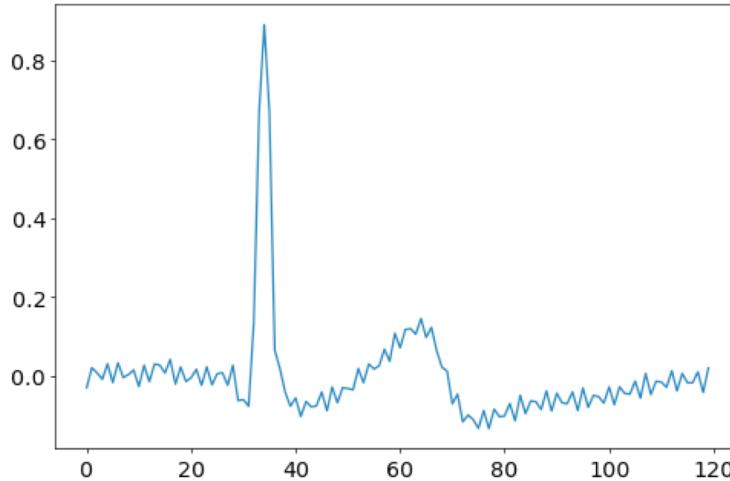
dilation = 1



dilation = 2



Splot i sieci neuronowe (przykład dla danych 1D)



`torch.nn.Conv1d`

```
in_channels=1,  
out_channels=1,  
kernel_size=3,  
stride=1,  
padding=0,  
dilation=1,  
groups=1,  
bias=False,  
padding_mode='zeros'.
```

 PyTorch

)

`tf.keras.layers.Conv1D`

```
filters=1,  
kernel_size=3,  
strides=1,  
padding='valid',  
data_format='channels_last',  
dilation_rate=1,  
groups=1,  
activation=None,  
use_bias=False,
```

 TensorFlow

)

Splot i sieci neuronowe (przykład dla danych 1D)

Założymy, że dane wejściowe mają rozmiar (N, C_{in}, L_{in}) a dane wyjściowe (N, C_{out}, L_{out}) , gdzie:

N - liczba przykładów porcji danych (batch-u),
 C - liczba kanałów/komponentów dla próbki danych,
 L - rozmiar (liczba próbek) danych wejściowych.

Wówczas rezultat operacji Conv1D (w praktycznych implementacjach) jest obliczany jako:

- dla $C_{in} = 1$:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + weight(C_{out_j}, k) \otimes input(N_i, k)$$

- dla $C_{in} > 1$:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \otimes input(N_i, k)$$

Splot i sieci neuronowe (przykład dla danych 1D)

Do zapamiętania

- 1) Stosowana jest korelacja wzajemna zamiast splotu
- 2) Sumowane są wyniku operacji korelacji wzajemnej jeśli dane wejściowe zawierają więcej niż jeden komponent dla próbek danych
- 3) Jeśli kernel size=1, a dane wejściowe zawierają więcej niż jeden komponent dla próbek danych wówczas sumowane są wartości komponentów danych wejściowych (z odpowiednią wagą, czyli w rezultacie możemy np. otrzymać wartość średnią). Ważne: na wyjściu uzyskamy „spłaszczony” zbiór danych, czyli jeden komponent na próbce danych.

Jak określić rozmiar danych wyjściowych?

Splot i sieci neuronowe (przykład dla danych 1D)

Biorąc pod uwagę poznane parametry jak: „padding”, „dilation”, „stride” czy „kernel size” rozmiar danych wyjściowych może być wyznaczony w następujący sposób:

$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times padding - dilation \times (kernel_size - 1) - 1}{stride} + 1 \right\rfloor$$

Dla często stosowanego zestawu wartości parametrów: padding=0, dilation=1, stride=1, kernel size =3 oraz np. $L_{in} = 5$ wówczas:

$$L_{out} = \left\lfloor \frac{5 + 2 \times 0 - 1 \times (3 - 1) - 1}{1} + 1 \right\rfloor = \lfloor 3 \rfloor = 3$$

Przykłady praktyczne (notatnik interaktywny DL_01_01_Convolution_Intro.ipynb)

Wprowadzenie do operacji splotu w środowisku Google Colaboratory/ Jupyter Notebook.

https://colab.research.google.com/drive/1-Z-7hUK7afbOnv_E0-jBD6iAJNHLNUUP?usp=sharing

Plan wykładu:

- Uczenie głębokie – wprowadzenie do przedmiotu i tematu
- Klasy głębokich sieci neuronowych
- Operacja splotu 1D i sieci neuronowe
- Podsumowanie

Podsumowanie

Sieci splotowe bazują na operacji splotu wykorzystywanej do uczenia się nowej reprezentacji danych.

Operacja splotu może być wykorzystana wielokrotnie dla danego kroku (wiele masek filtrów) oraz na kolejnych poziomach (warstwach ukrytych). W efekcie przechodzimy do danych surowych do uogólnionych cech tworzących często jednowymiarowy wektor danych podawany na wejście klasyfikatora lub innego algorytmu uczenia maszynowego.

W czasie kolejnego spotkania przedstawimy aspekty praktycznego zastosowania operacji splotu 2D w sieciach splotowych oraz wskażemy jak te operacje mogą być rozbudowane poprzez kolejne rodzaje warstw (np. warstwy agregujące cechy - np. pooling, itp.).

Dziękuję

Jacek Rumiński



Fundusze
Europejskie
Polska Cyfrowa

Rzeczpospolita
Polska



KANCELARIA PREZESA RADY MINISTRÓW
THE CHANCELLERY OF THE PRIME MINISTER

Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.



POLITECHNIKA
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Uczenie głębokie

Wykład 2: Sieci splotowe

Jacek Rumiński

Katedra Inżynierii Biomedycznej, Wydział ETI



Fundusze
Europejskie
Polska Cyfrowa



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu:

- **Splot 2D**
- Warstwy i sieci splotowe
- Pole recepcyjne i warstwy agregacji
- Podsumowanie

Splot 2D

Obraz (6,4)

R1	R2	R3	R4
R5	R6	R7	R8
R9	R10	R11	R12
R13	R14	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24

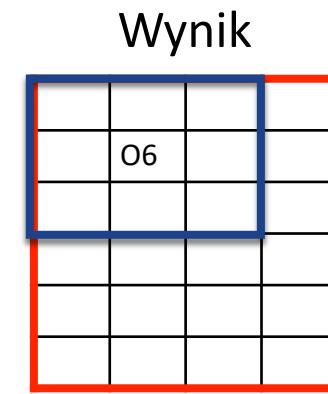
Maska (3,3)

W1	W2	W3
W4	W5	W6
W7	W8	W9

Krok operacji „splotu”

$$\begin{array}{|c|c|c|} \hline R1 & R2 & R3 \\ \hline R5 & R6 & R7 \\ \hline R9 & R10 & R11 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline W1 & W2 & W3 \\ \hline W4 & W5 & W6 \\ \hline W7 & W8 & W9 \\ \hline \end{array}$$

=



$$z = \sum_{m=0}^8 W_m R_m$$

czyli:

$$O6 = \sum (W_i * R_j) = (W1 * R1) + (W2 * R2) + (W3 * R3) + (W4 * R5) + \dots + (W9 * R11)$$

itd.

Splot 2D

O6

R1	R2	R3	R4
R5	R6	R7	R8
R9	R10	R11	R12
R13	R14	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24

O7

R1	R2	R3	R4
R5	R6	R7	R8
R9	R10	R11	R12
R13	R14	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24

O10

R1	R2	R3	R4
R5	R6	R7	R8
R9	R10	R11	R12
R13	R14	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24

O11

R1	R2	R3	R4
R5	R6	R7	R8
R9	R10	R11	R12
R13	R14	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24

obraz wyjściowy (4,2)

O6	O7		
O10	O11		
O14	O15		
O18	O19		

Splot 2D

O14

R1	R2	R3	R4
R5	R6	R7	R8
R9	R10	R11	R12
R13	R14	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24

O15

R1	R2	R3	R4
R5	R6	R7	R8
R9	R10	R11	R12
R13	R14	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24

O18

R1	R2	R3	R4
R5	R6	R7	R8
R9	R10	R11	R12
R13	R14	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24

O19

R1	R2	R3	R4
R5	R6	R7	R8
R9	R10	R11	R12
R13	R14	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24

obraz wyjściowy (4,2)

O6	O7		
O10	O11		
O14	O15		
O18	O19		

W celu uzyskania takiego samego rozmiaru na wyjściu jak na wejściu można użyć dopełnienia (padding) wartości tensora wejściowego (np. dodanie dodatkowych wierszy i kolumn na brzegach macierzy z wartościami 0 lub innymi – warunki brzegowe jak wcześniej).

Splot 2D

W celu uzyskania takiego samego rozmiaru na wyjściu jak na wejściu można użyć dopełnienia (padding) wartości tensora wejściowego (np. dodanie dodatkowych wierszy i kolumn na brzegach macierzy z wartościami 0 lub innymi – warunki brzegowe jak wcześniej).

Na przykład dla „kernel size=3” oraz dopełnienia wartościami 0 uzyskamy nowe dane wejściowe:

0	0	0	0	0	0
0	R1	R2	R3	R4	0
0	R5	R6	R7	R8	0
0	R9	R10	R11	R12	0
0	R13	R14	R15	R16	0
0	R17	R18	R19	R20	0
0	R21	R22	R23	R24	0
0	0	0	0	0	0

O1	O2	O3	O4
O5	O6	O7	O8
O9	O10	O11	O12
O13	O14	O15	O16
O17	O18	O19	O20
O21	O22	O23	O24

obraz wyjściowy (6,4)

Splot 2D

Przykładowe znacznie zastosowania operacji splotu na obrazach z przykładowymi maskami o zadanych wartościach wag.

W1	W2	W3
W4	W5	W6
W7	W8	W9

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

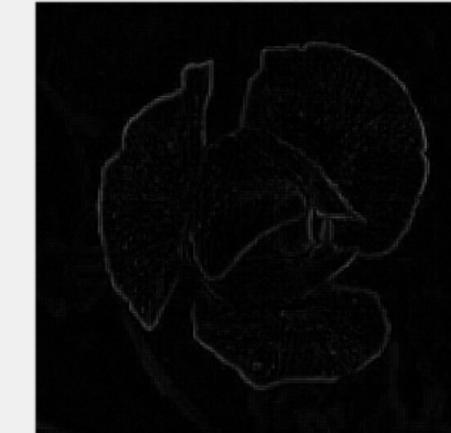
-1	-1	-1
0	0	0
1	1	1



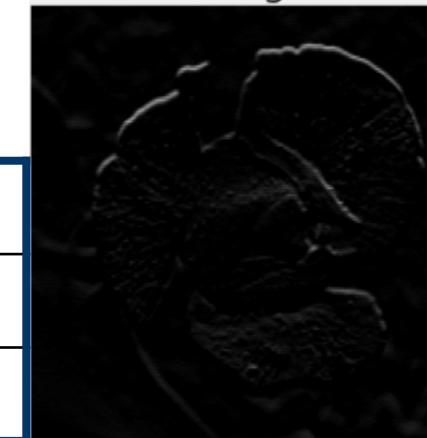
3x3 average



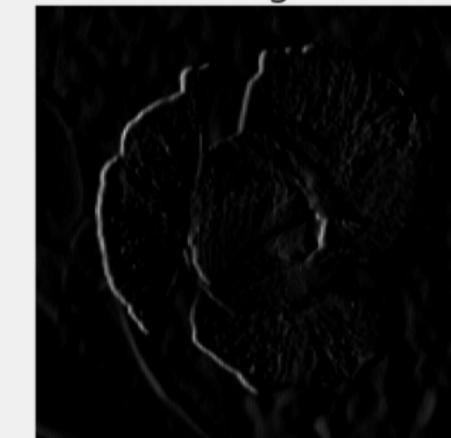
3x3 high pass



3x3 edge H



3x3 edge V



Przykłady praktyczne (notatnik interaktywny DL_02_01_Convolution_2D_Intro.ipynb)

Wprowadzenie do operacji splotu 2D w środowisku Google Colaboratory/ Jupyter Notebook.

https://colab.research.google.com/drive/1SLJ8rOAZKps3jfQ2PNDQ_lEfo6Q_604-?usp=sharing

Plan wykładu:

- Splot 2D
- **Warstwy i sieci splotowe**
- Pole recepcyjne i warstwy agregacji
- Podsumowanie

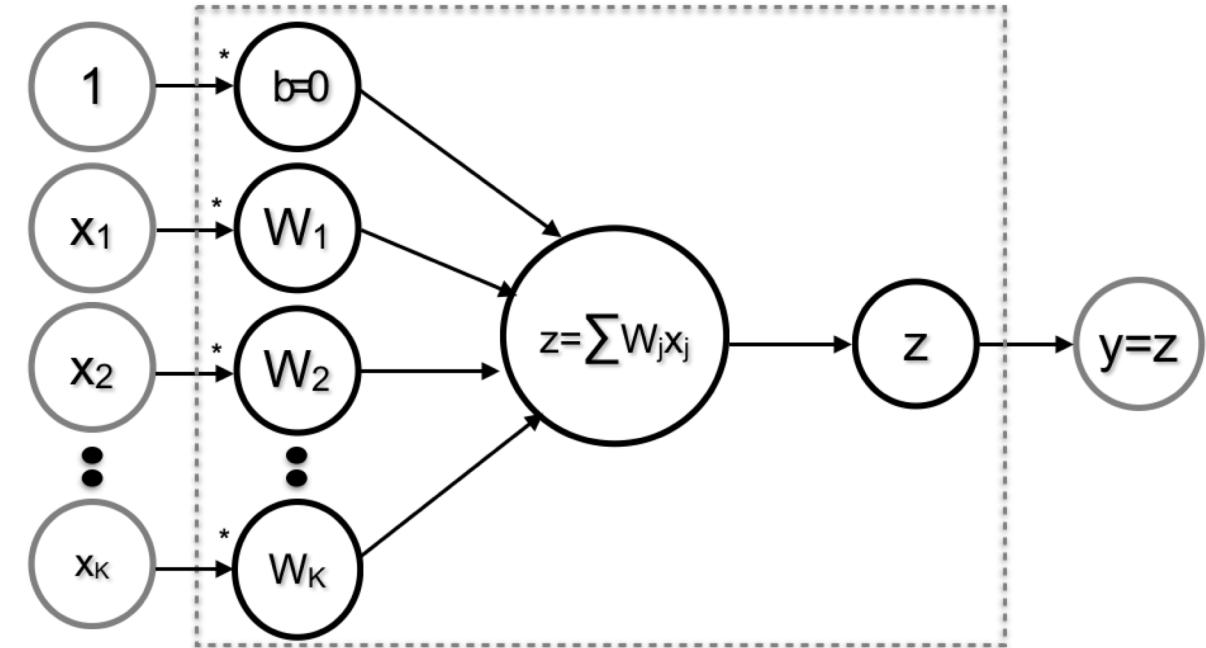
Splot i sieci neuronowe - dane 2D

Operacja splotu realizowana przez węzeł sieci neuronowej:

W1	W2	W3
W4	W5	W6
W7	W8	W9

$$z = \sum_{m=1}^9 W_m R_m$$

W procesie uczenia sieci poszukiwane są najlepsze dla danego zadania wartości parametrów W.



Poszukujemy wartość W, ale (klasycznie) ustalamy wartości (hiper) parametrów:

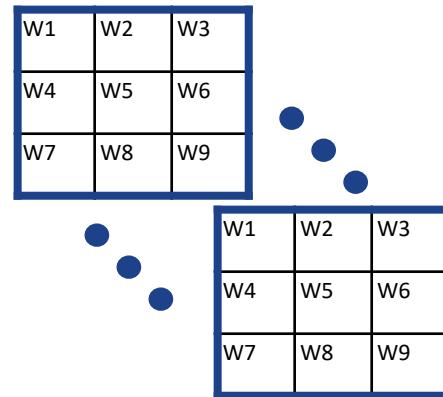
- Rozmiar maski filtru (np. 3x3 lub 5x5, itd.)
- Liczbę filtrów - masek (np. 3 maski, każda TAKIEGO SAMEGO rozmiaru)
- Szereg innych parametrów omawianych wcześniej (np. padding, dilation rate, stride, itp.).

Splot i sieci neuronowe - dane 2D

Obraz (6,4)

R1	R2	R3	R4
R5	R6	R7	R8
R9	R10	R11	R12
R13	R14	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24

Zbiór 16 masek (3,3)



Dla każdej maski

R1	R2	R3
R5	R6	R7
R9	R10	R11



W1	W2	W3
W4	W5	W6
W7	W8	W9

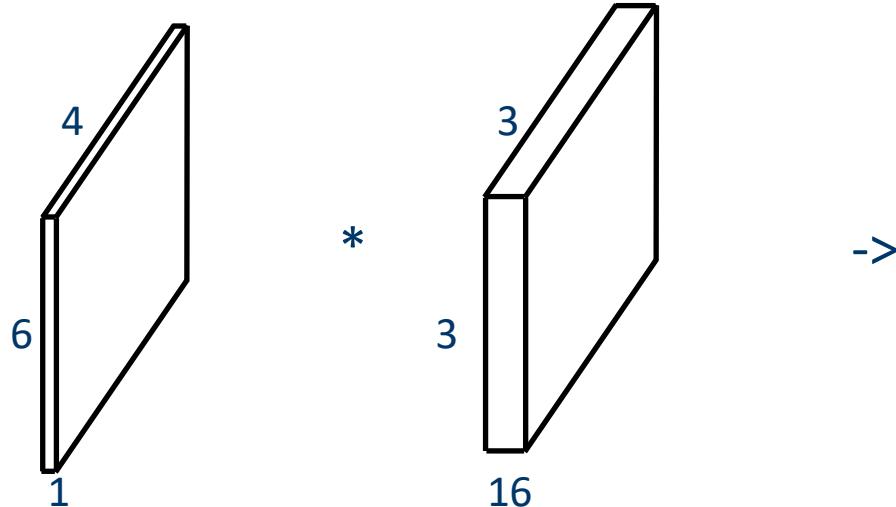


$$z = \sum_{m=1}^9 W_m R_m$$

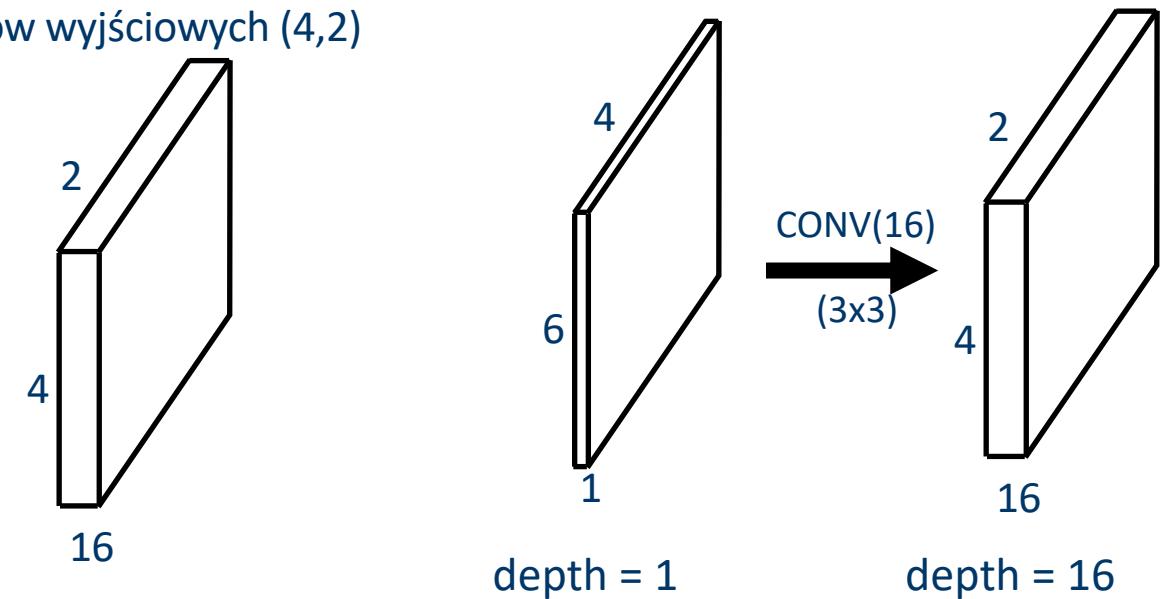
Wynik (4,2)

06	07
010	011
014	015
018	019

1 obraz wejściowy (6,4) * 16 masek (3,3) -> 16 obrazów wyjściowych (4,2)



Zmniejszamy szerokość/wysokość – zwiększamy głębokość



Splot i sieci neuronowe - dane 2D

Operacja „splotu” 2D implementowana jest w pakiecie TensorFlow poprzez klasę **tf.keras.layers.Conv2D**, a w pakiecie PyTorch przez klasę **torch.nn.Conv2d**.

Definicja w TensorFlow:

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid',  
    data_format=None, dilation_rate=(1, 1), groups=1, activation=None,  
    use_bias=True, kernel_initializer='glorot_uniform',  
    bias_initializer='zeros', kernel_regularizer=None,  
    bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,  
    bias_constraint=None, **kwargs  
)
```

W PyTorch:

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1,  
bias=True, padding_mode='zeros', device=None, dtype=None)
```

Splot i sieci neuronowe - dane 2D

Inne ważne klasy związane z ważnymi wersjami operacji na danych powiązanych ze „splotem” (na przykładzie TensorFlow):

- DepthwiseConv2D - „splot” z separowanymi maskami dla komponentów (danych głębokości).
- LocallyConnected2D - „splot” bez współdzielenia wag - różne współczynniki maski (czyli filtry) mogą być zastosowane do różnych regionów danych wejściowych.
- Conv2DTranspose - transposed convolution - przybliżenie operacji rozplotu (odwrotnej operacji do splotu). Nie jest to jednak rozplot.
- Analogiczne wersje dla 1D i 3D (Conv3D - podobnie jak dla 1D czy 2D, ale dla danych przestrzennych 3D plus dane głębokości).

Zapoznamy się z wybranymi z tych klas w ramach kolejnych przykładów jak i następnych spotkań.

Przykłady praktyczne (notatnik interaktywny DL_02_02_Convolution_2D.ipynb)

Operacja splotu dla danych 2D w środowisku Google Colaboratory/ Jupyter Notebook.

<https://colab.research.google.com/drive/14dB9Tl0lgGGaBDSZf7vGpuhIZBgdlAga?usp=sharing>

Plan wykładu:

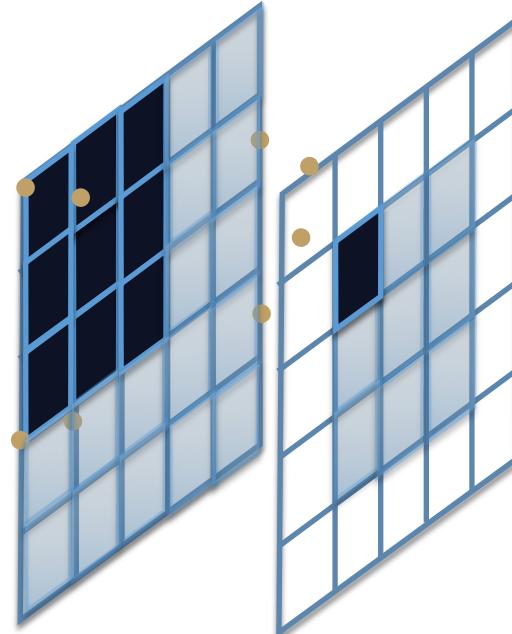
- Splot 2D
- Warstwy i sieci splotowe
- **Pole recepcyjne i warstwy agregacji**
- Podsumowanie

Splot 2D: pole recepcyjne (odbiorcze, ang. receptive field)

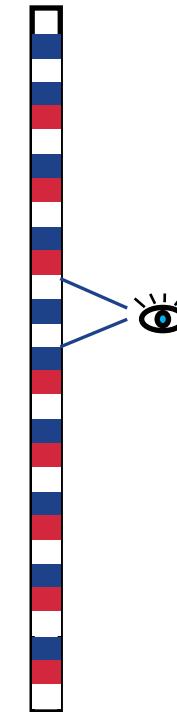
Analogia biologiczna:

„**pole odbiorcze, pole recepcyjne**, biol. obszar powierzchni zawierającej receptory (skóry, siatkówki), z którego impulsy nerwowe docierają do wspólnego elementu odbiorczego – komórki nerwowej lub włókna nerwowego;” [1].

Pole recepcyjne w CNN: zakres danych, z których wyznaczono nową reprezentację (cechy) danych do danego poziomu (warstwy) sieci neuronowej.



Pole recepcyjne = 9 (3×3)
Kernel size, $k = 3$
Stride, $s = 1$
Padding, $p=0$

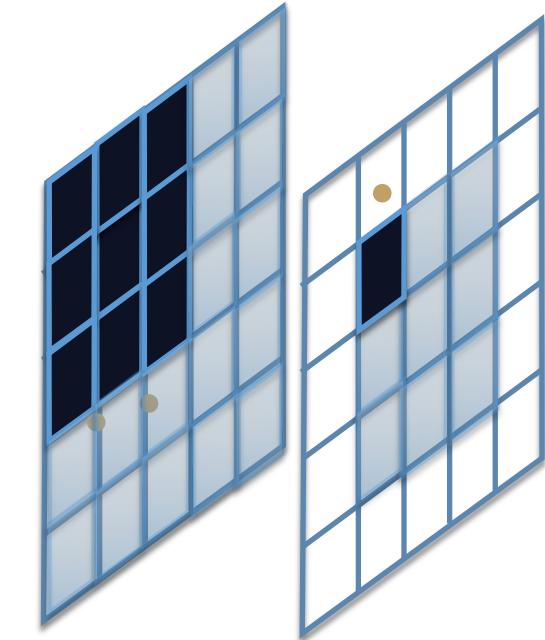


Splot 2D: pole recepcyjne (odbiorcze, ang. receptive field)

W pierwszej warstwie splotowej uzyskiwane są nowe reprezentacje danych. Każda nowa wartość jest efektem zastosowania operacji na 9 (oczywiście dla przedstawionego przykładu maski) wartościach wejściowych, skupionych lokalnie.

W przedstawionym przykładzie uzyskujemy 9 nowych wartości, każda stanowi wyniki przetwarzania lokalnie sąsiednich 9 wartości wejściowej macierzy danych. Czyli nie wszystkie dane wejściowe są podawane do neuronu realizującego operację „splotu”.

Uzyskane dane mają rozmiar przestrzenny (3,3). Jeśli sztucznie założymy, że tak wyodrębnione cechy wystarczająco reprezentują dane wejściowe wówczas musimy je przedstawić w formie odpowiedniej dla sztucznego neuronu (lub zbioru neuronów w warstwie). Neuron taki (lub każdy neuron warstwy) przetwarza WSZYSTKIE dane wejściowe (ang. Fully Connected - FC, Dense layer, itp.).



Splot 2D: pole recepcyjne (odbiorcze, ang. receptive field)

W rozpatrywanym przykładzie wynik warstwy splotowej to zbiór wartości o rozmiarze (3,3). Zmiana reprezentacji do wektora 1D może przykładowo przebiegać następująco:

```
layer_after_cnn = np.array([[1,2,3],[4,5,6],[7,8,9]])
# layer_after_cnn = np.array([[[1,1],[2,4],[3,6]],[[4,8],[5,10],[6,12]],[[7,14],[8,16],[9,18]]])
print("Input shape:", layer_after_cnn.shape)
flatten_layer_after_cnn = layer_after_cnn.flatten()
print("After flatten shape:", flatten_layer_after_cnn.shape)
print("Flatten data: ", flatten_layer_after_cnn)
```

Wynik:

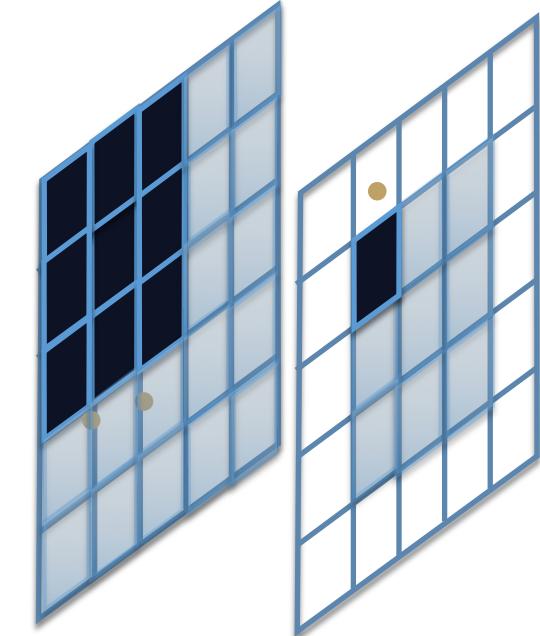
Input shape: (3, 3)
After flatten shape: (9,)
Flatten data: [1 2 3 4 5 6 7 8 9]

Input shape: (3, 3, 2)
After flatten shape: (18,)
Flatten data: [1 1 2 4 3 6 4 8 5 10 6 12 7 14 8 16 9 18]

Splot 2D: pole recepcyjne (odbiorcze, ang. receptive field)

Projektując architekturę sieci neuronowej możemy skorzystać z dostępnych pakietów, które oferują operację spłaszczenia danych jako "warstwę" przetwarzania danych (w warstwie tej nie ma uczenia parametrów czy wag). Przykładowo:

```
keras_model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(1, kernel_size=3, activation='relu',
        input_shape=x_train.shape[1:], name="conv2D_1"),
    tf.keras.layers.Flatten(), # make data flat to use dense layers
    tf.keras.layers.Dense(no_of_classes, activation='softmax')
])
```



ALE, pamiętajmy, że dane (tensor) wejściowe dla takiej operacji będzie odpowiedniego rozmiaru. Przyjrzymy się temu na kolejnym slajdzie.

Splot 2D: pole recepcyjne (odbiorcze, ang. receptive field)

Przykładowo:

```
layer_after_cnn = np.array([[1,2,3],[4,5,6],[7,8,9]])
print("Input shape:", layer_after_cnn.shape)
layer_flatten = tf.keras.layers.Flatten()
flatten_layer_after_cnn = layer_flatten(layer_after_cnn)
print("After flatten shape:", flatten_layer_after_cnn.shape)
```

Wyniki: Input shape: (3, 3) After flatten shape: (3, 3) **CZYLI NIC SIĘ NIE ZMIENIŁ!!!**

Pierwszy wymiar rozmiaru danych traktowany jest jako rozmiar batcha, a operacja (pomijając różne opcje) bazuje na wywołaniu `tf.reshape(input, reshape_size)`, gdzie `reshape_size` definiowana jest jako `[input.shape[0], -1]`. Oznacza to, że pierwsza wartość rozmiaru ma być BEZ ZMIAN, natomiast reszta ma być tak przekształcona, aby całkowity rozmiar danych się nie zmienił.

Jeśli dane wejściowe mają rozmiar (1,3,3), lub (uwzględniając podanie głębokości danych (1,3,3,1) to wynik operacji Flatten() będzie (1,9). Pierwszy wymiar bez zmian (rozmiar porcji danych) reszta spłaszczona.

Splot 2D: pole recepcyjne (odbiorcze, ang. receptive field)

Podsumujmy:

Każda wartość w „spłaszczonym” zbiorze jest obliczona jako nowa reprezentacja zestawu wartości w poprzednich warstwach z danego POLA RECEPCYJNEGO.

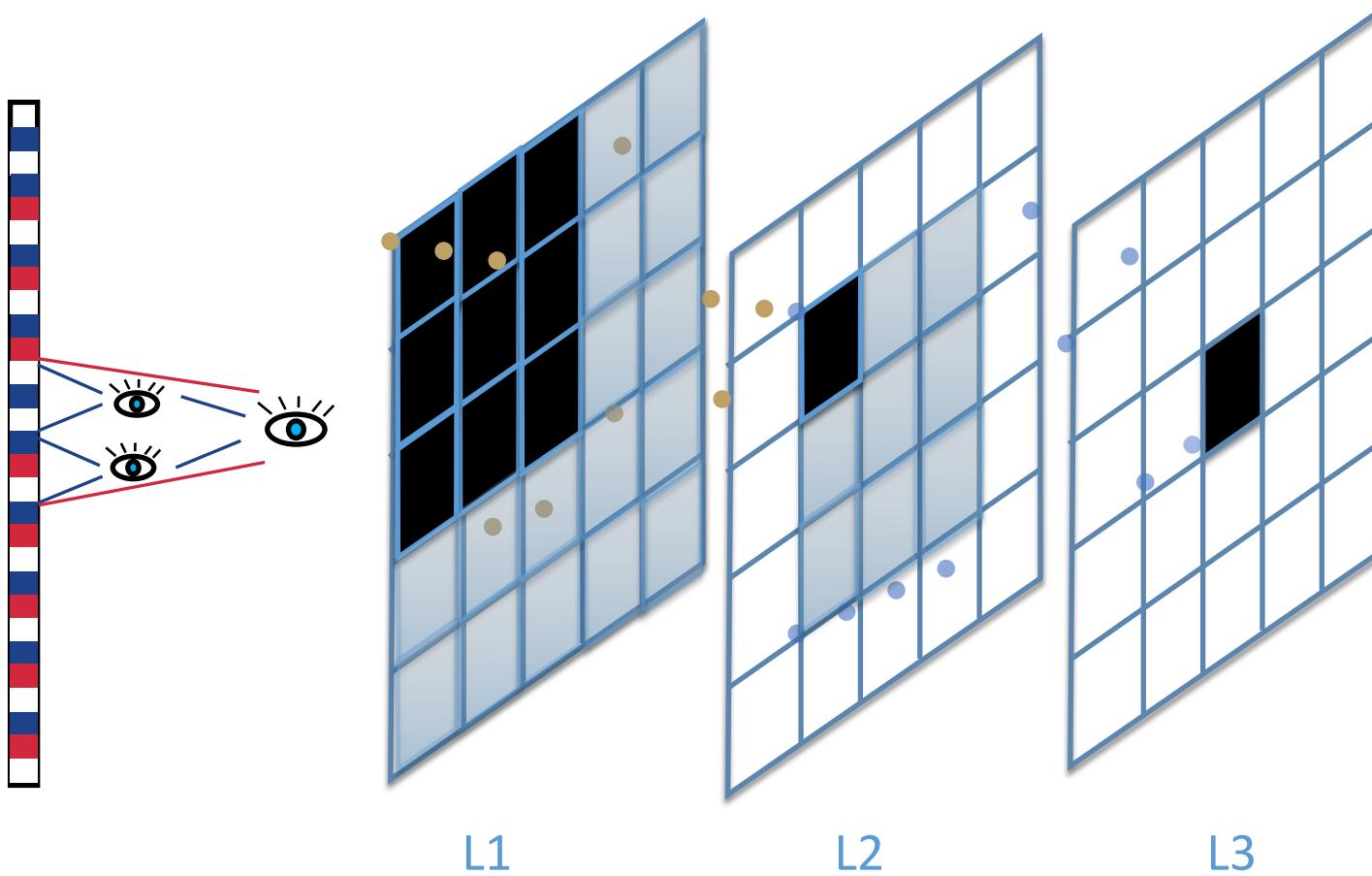
Definiując sekwencję warstw splotowych wyznaczamy w rezultacie cechy z cech (uogólniamy cechy szczegółowe z wcześniejszych warstw).

Często dążymy do tego, aby każda wartość w „spłaszczonym” zbiorze danych reprezentowała pewną cechę odniesioną do całego zbioru wejściowego (np. tak jak w tradycyjnym podejściu wyznaczamy np. deskryptory statystyczne dla danych obrazu, deskryptory opisujące teksturę, itp. - każda wartość odnosi się do całego obrazu).

Dlatego kontrolujemy pole recepcyjne poprzez odpowiedni dobór liczby i rodzaju warstw oraz szeregu ich parametrów takich jak: rozmiar maski, krok (ang. stride) operacji, stopień separacji (ang. dilation rate), rodzaj i stopień agregacji (np. ang. pooling), itp.

Splot 2D: pole recepcyjne - rozmiar i kontrola

Rozmiar pola recepcyjnego r dla L-warstwowego modelu:



$$r_0 = \sum_{l=1}^L ((k_l - 1) \prod_{j=1}^{l-1} s_j) + 1$$

Stride, $s = 1$

Padding, $p = 0$

Kernel size, $k = 3$

Pole recepcyjne (np. dla wierszy):

- L1->L2

$$r_0 = (3-1)+1 = 3 \text{ (lub } 3 \times 3 \text{ dla 2D)}$$

- L2->L3

$$r_0 = ((3-1) + ((3-1)*1)) + 1 = 2+2+1 = 5 \\ (\text{lub } 5 \times 5 \text{ dla 2D})$$

- L3->L4

$$r_0 = ((3-1) + ((3-1)*1) + ((3-1)*1)) + 1 = \\ 2+2+2+1 = 7$$

Splot 2D: pole recepcyjne - rozmiar i kontrola

Rozmiar pola recepcyjnego r dla L-warstwowego modelu ZWIĘKSZAJĄC KROK - STRIDE

$$r_0 = \sum_{l=1}^L ((k_l - 1) \prod_{j=1}^{l-1} s_j) + 1$$

Stride, $s = 2$

Padding, $p = 0$

Kernel size, $k = 3$

Pole recepcyjne (np. dla wierszy):

- L1->L2: $r_0 = (3-1)+1 = 3$ (lub 3x3 dla 2D)

- L2->L3: $r_0 = ((3-1) + ((3-1)*2)) + 1 = 2+4+1 = 7$ (lub 7x7 dla 2D)

- L3->L4: $r_0 = ((3-1) + ((3-1)*2) + ((3-1)*2*2)) + 1 = 2+4+8+1 = 15$

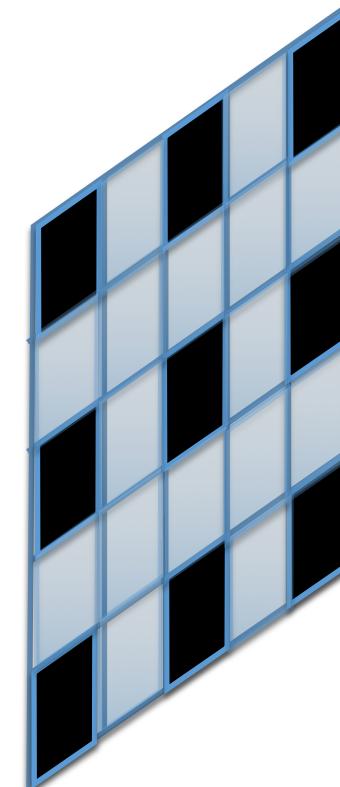
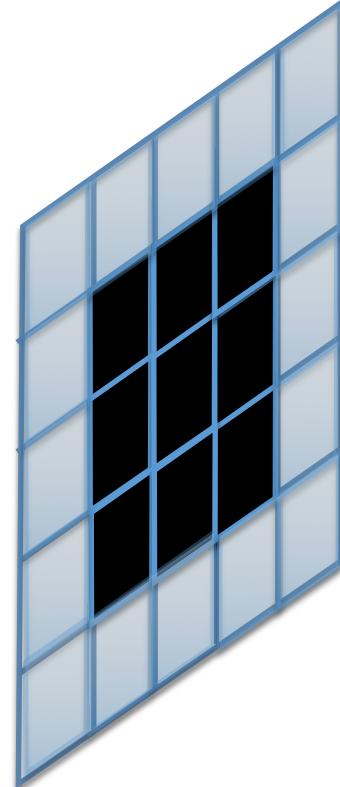
Zwiększając krok zwiększyliśmy pole recepcyjne.

Splot 2D: pole recepcyjne - rozmiar i kontrola

Rozmiar pola recepcyjnego r dla L-warstwowego modelu ZWIĘKSZAJĄC STOPIEŃ SEPARACJI - DILATION RATE

Zwiększając stopień separacji maski wprowadzamy puste miejsca w schemacie operacji splotu (tj. w wyborze danych wejściowych), wskutek czego zwiększamy pole recepcyjne.

Stopień separacji = 1
Pole recepcyjne = 3 (3 x 3)

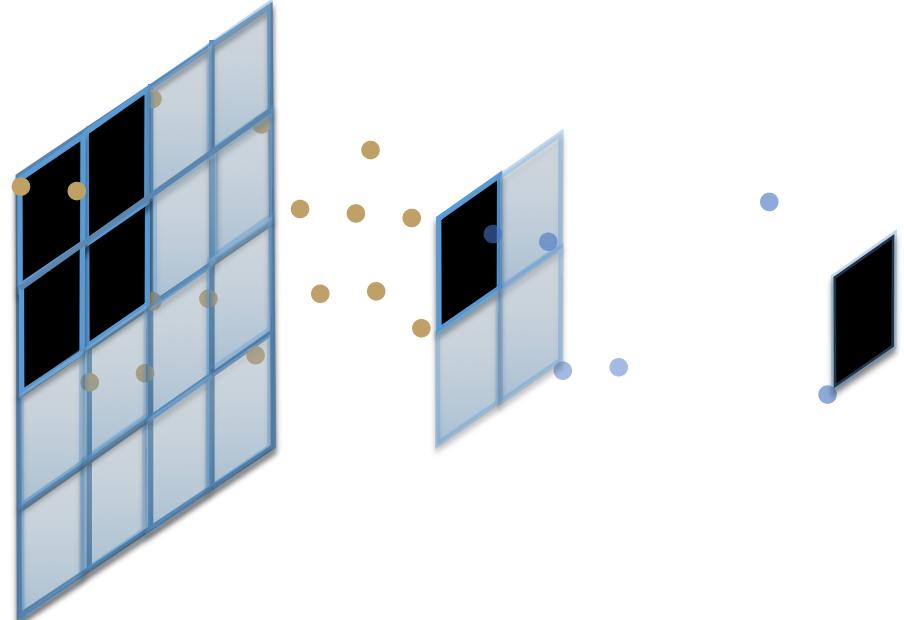


Stopień separacji = 2
Pole recepcyjne = 5 (5 x 5)

Splot 2D: pole recepcyjne - rozmiar i kontrola

Rozmiar pola recepcyjnego r dla L-warstwowego modelu WPROWADZAJĄC POOLING

Podobnie jak w przypadku warstw splotowych wprowadzamy warstwy redukcji danych przez próbkowanie lub agregację (ang. Pooling Layers). Parametrem warstwy jest rozmiar okna, w którym odbywa się redukcja, np. 2×2 (z 4 wartości uzyskamy 1 wartość). Rodzaj warstwy redukcyjnej określa typ operacji np. MaxPooling – pozostaw wartość maksymalną z okna, AveragePooling – pozostaw wartość średnią, itp. Podobnie jak dla warstw splotowych definiowane są również inne parametry, np. krok (ang. stride).



Operacja redukcji może minimalizować wpływ lokalnych perturbacji na końcowy wynik modelu.

Efekt zastosowania warstw redukcyjnych:

- 1) Zwiększamy pole recepcyjne
- 2) Redukujemy rozmiar danych przestrzennych (zwykle na rzecz danych "głębokości" reprezentujących uogólniane cechy).

Splot 2D: Pooling

O6	O7
O10	O11
O14	O15
O18	O19

P1	P2
P3	P4

O6	O7
O10	O11

=

Z6	

Pooling - redukcja w oknie:
wartość maks., średnia, itp.

$$Z_6 = \text{MAX}(O_6, O_7, O_{10}, O_{11}) \text{ lub}$$

$$Z_6 = \text{AVG}(O_6, O_7, O_{10}, O_{11})$$

Przykłady:

maska = 2×2
STRIDE = 1

O6	O7
O10	O11
O14	O15
O18	O19

O6	O7
O10	O11
O14	O15
O18	O19

O6	O7
O10	O11
O14	O15
O18	O19

Wynik (3,1)

Z6	
Z10	
Z14	

maska = 2×2
STRIDE = 2

O6	O7
O10	O11
O14	O15
O18	O19

O6	O7
O10	O11
O14	O15
O18	O19

Wynik (2,1)

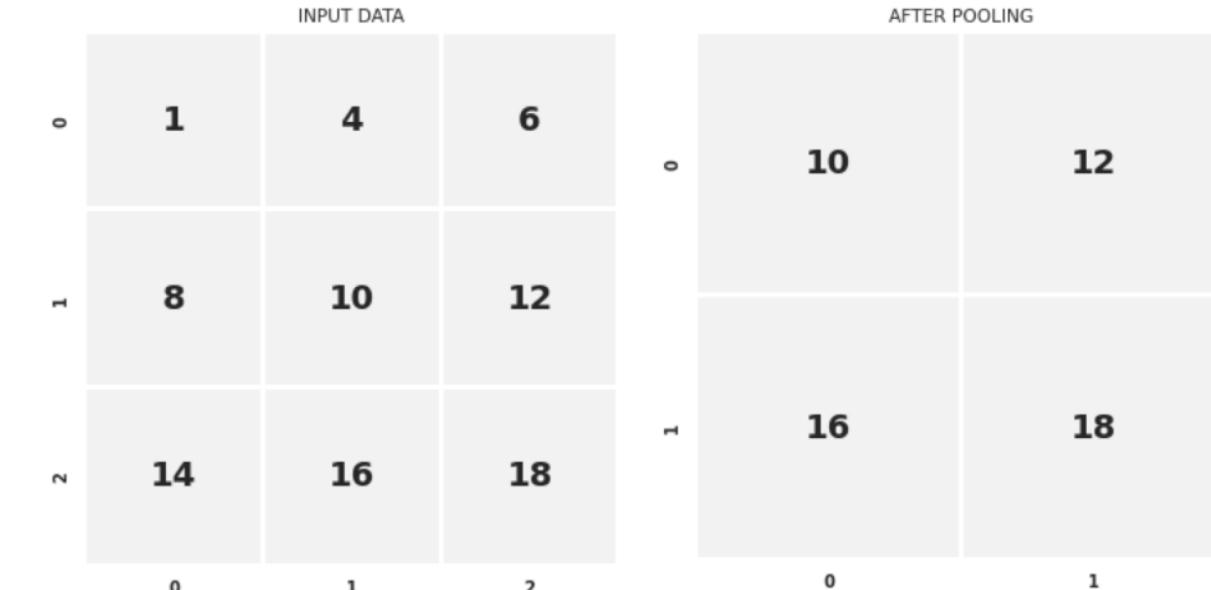
Z6	
Z14	

Splot 2D: Pooling - przykład w środowisku TF (MaxPooling2D, AveragePooling2D)

```
layer_after_cnn = np.array([[[1,1],[2,4],[3,6]],[[4,8],[5,10],[6,12]],[[7,14],[8,16],[9,18]]])  
# Shape (3,3,2) - depth = 2 -> Reshape to add batch size  
layer_after_cnn_reshape = layer_after_cnn.reshape((1,3,3,2))  
layer_pooling = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding='valid')  
pooling_layer_after_cnn = layer_pooling(layer_after_cnn_reshape)
```



Kanał /Komponent 1



Kanał/Komponent 2

Liczba kanałów bez zmian!!! Zmiana wymiaru przestrzennego.

Splot 2D: Redukcja globalna (**GlobalMaxPooling2D**, **GlobalAveragePooling2D**)

Przedstawione wyżej operacje redukcji odbywały się lokalnie - w odniesieniu do rozmiaru maski. Założymy, że ustawiлиbyśmy rozmiar maski taki sam jak rozmiar przestrzenny obrazu. Wówczas otrzymamy operację agregacji globalnej (przy zachowaniu rozmiaru tensora - shape). Ponieważ rozmiar przestrzenny danych jest znany w trakcie przetwarzania, dlatego może zdefiniować operację agregacji globalnej bez podawania parametrów typu pool_size, strides, itp.

W TF dostępne są operacje globalne: **GlobalMaxPooling2D**, **GlobalAveragePooling2D**. Stosowane są często na zakończenie bloku operacji związanych z ekstrakcją cech (zmiany reprezentacji) w celu „spłaszczenia” danych. Różnią się jednak zasadniczo od poznanej wcześniej operacji Flatten, np.:

```
layer_after_cnn = np.array([[1,1], [2,4], [3,6]], [[4,8], [5,10], [6,12]], [[7,14], [8,16], [9,18]])  
# After Flatten – all data, another representation:  
tf.Tensor([[1 1 2 4 3 6 4 8 5 10 6 12 7 14 8 16 9 18]], shape=(1, 18), dtype=int64)  
# After GlobalMaxPooling2D – aggregated data for each channel:  
tf.Tensor([[9 18]], shape=(1, 2), dtype=int64)
```

Splot 2D: Redukcja (agregacja) globalna vs. splot

Agregacja globalna stosowana jest najczęściej tuż przed blokiem warstw typu MLP. Uzyskanie jednowymiarowej reprezentacji cech podawanych na wejście MLP związane jest z prostą operacją agregacji, które nie podlega uczeniu.

Szereg cech tak starannie wyodrębnionych w poprzednich krokach jest redukowana przez definiowaną a priori operację wyboru wartości maksymalnej lub funkcję wyznaczania wartości średniej.

Zaleta: nie zwiększamy liczby parametrów podlegających uczeniu dla danego modelu.

Wada: pozbywamy się szeregu cech, które mogą mieć ważne znaczenie w dalszych krokach.

Alternatywa:

- 1.) spłaszczenie (Flatten) - może prowadzić do zbyt dużej liczby cech
- 2.) splot - dodatkowe parametry podlegające uczeniu (ryzyko np. przeuczenia modelu).

Splot 2D: Redukcja (agregacja) globalna vs. splot

Warto zauważyc, że często na końcu bloków modelu typu CNN danej mają niewielkie wymiary przestrzenne (a większą liczbę kanałów w skutek stosowania wielu masek-filtrów).

W celu spłaszczenia danych moglibyśmy wykorzystać operację splotu tak, aby nauczyć model parametrów decydujących z jaką wagą wybrać dane przestrzenne.

Tradycyjna operacja splotu realizowana w pakietach typu TF czy PyTorch sumuje wyniki korelacji wzajemnej dla każdego z kanałów. Naszym celem jest jednak utrzymanie różnorodności cech w poszczególnych kanałach.

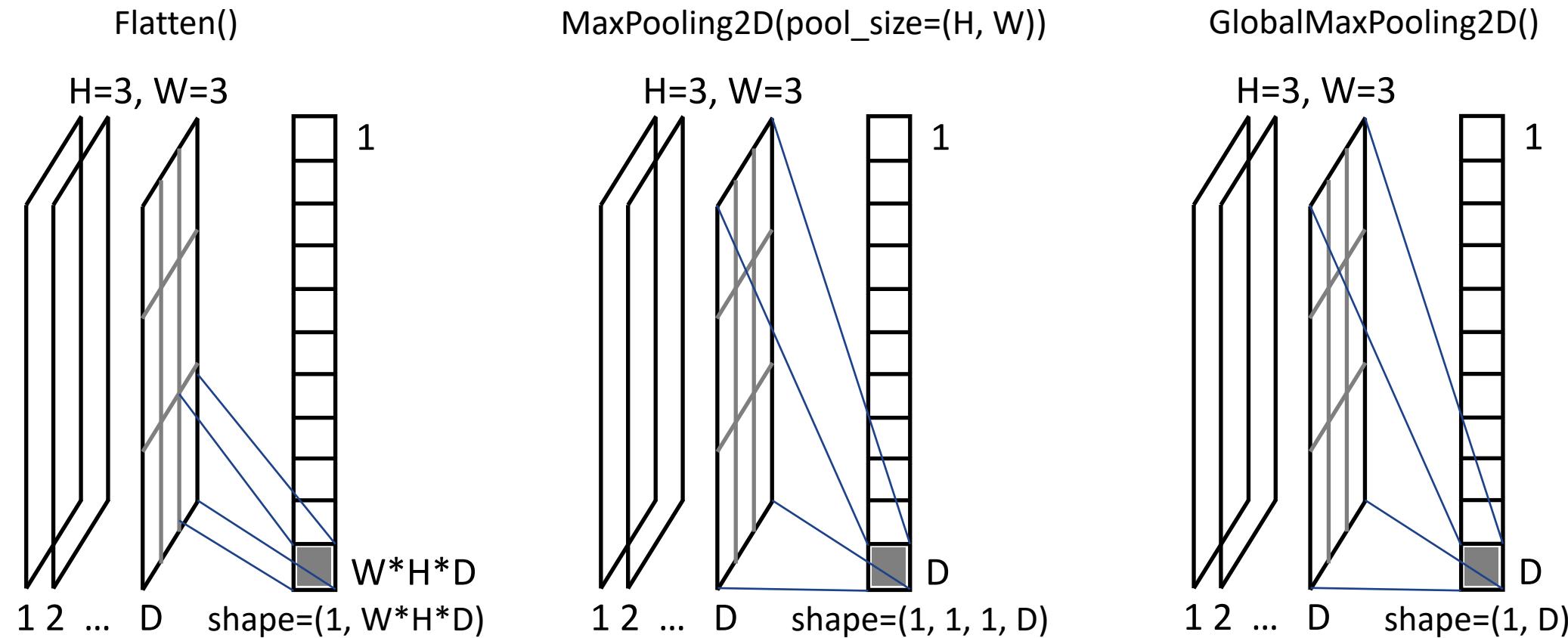
Dlatego można zastosować taką wersję operacji splotu, która wyznacza parametry dla każdego z kanałów i nie sumuje wyników: **DepthwiseConv2D**.

Splot 2D: Przygotowanie danych z wyjścia CNN dla MLP - przykłady technik

```
layer_after_cnn = np.array( [[[1,1,1],[2,4,6],[3,6,9]],  
                           [[4,8,12],[5,10,15],[6,12,18]],  
                           [[7,14,21],[8,16,24],[9,18,27]]]).astype('float')  
layer_after_cnn_reshape = layer_after_cnn.reshape((1,3,3,3))  
  
# uncomment method to test  
layer_agg = tf.keras.layers.Flatten()  
# layer_agg = tf.keras.layers.MaxPooling2D(pool_size=(3, 3))  
# layer_agg = tf.keras.layers.GlobalMaxPooling2D()  
# w_fixed = np.ones((3,3,3,1)).astype('float')  
# layer_agg = tf.keras.layers.Conv2D(1, kernel_size=(3, 3), use_bias=False, weights=[w_fixed])  
# layer_agg = tf.keras.layers.DepthwiseConv2D(kernel_size=(3, 3), use_bias=False, weights=[w_fixed])  
  
flatten_layer_after_cnn = layer_agg(layer_after_cnn_reshape)  
  
print("After flatten shape:", flatten_layer_after_cnn.shape)  
print("Flatten data: ", flatten_layer_after_cnn)
```

Splot 2D: Redukcja (agregacja) globalna vs. splot

Oczekiwany rozmiar wyjściowy to (N, M) , gdzie N – rozmiar batcha, M – liczna cech. Jeśli zastosowana technika przetwarzania danych na końcu bloków splotowych doprowadzi do uzyskania danych o innym kształcie należy zastosować zmianę rozmiaru do (N, M) .

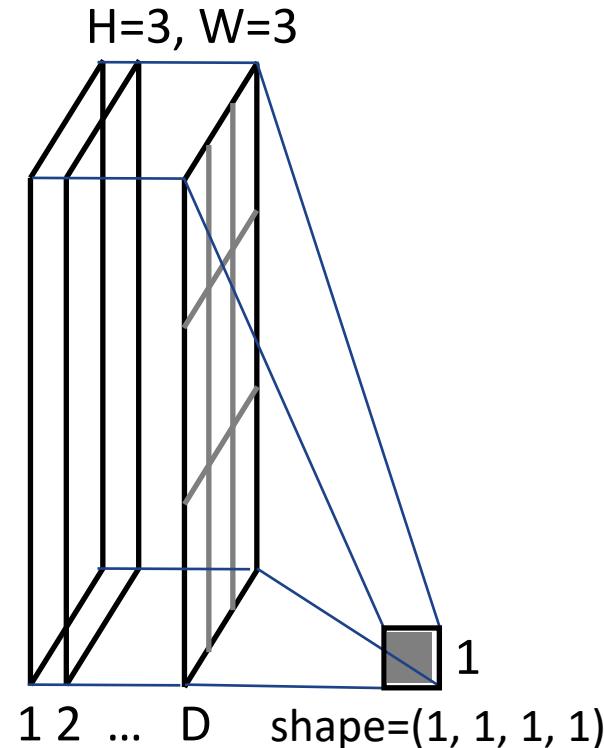


Splot 2D: Redukcja (agregacja) globalna vs. splot

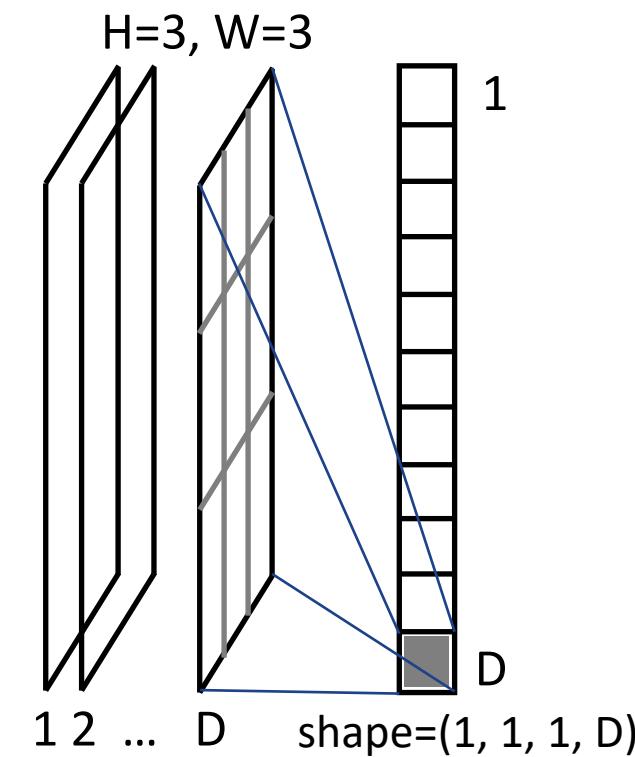
Blok Conv2D o rozmiarze pojedynczej maski (H, W) równym aktualnym wymiarom przestrzennym danych (na końcu bloku warstw splotowych) doprowadzi do uzyskania pojedynczej wartości.

Blok DepthwiseConv2D o analogicznym rozmiarze maski jak wyżej pozwoli uzyskać wartość dla każdego z kanałów oddzielnie z uwzględnieniem wyuczonych wag dla danych przestrzennych.

Conv2D(filters=1, kernel_size=(H, W))



DepthwiseConv2D(kernel_size=(H, W))

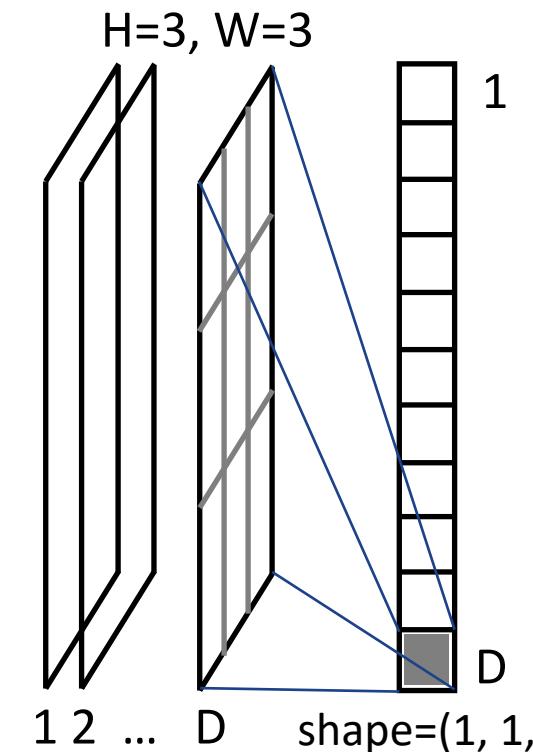


Splot 2D: Redukcja (agregacja) globalna vs. splot

Oczywiście można zaprojektować model zbudowany wyłącznie z warstw Conv2D (bez dopełniania, lub z dopełnianiem + agregacja), który w ostatniej warstwie będzie miał D filtrów:

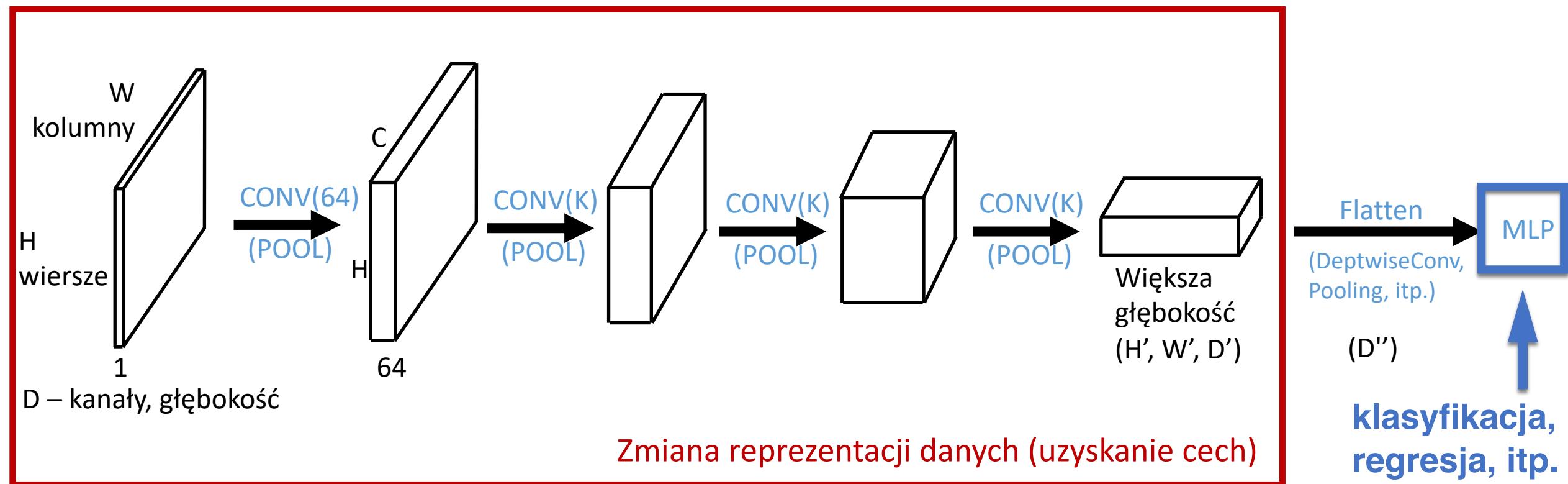
```
# For previous example:  
no_of_filters = 3  
w_fixed_3 = np.ones((3,3,3,no_of_filters)).astype('float')  
layer_agg = tf.keras.layers.Conv2D(filters=no_of_filters,  
                                   kernel_size=(3, 3),  
                                   use_bias=False,  
                                   weights=[w_fixed_3])
```

Conv2D(filters=D, kernel_size=(H, W))



Sieci splotowe

Architekturę modelu sieci splotowej tworzymy na bazie poznanych operacji – warstw/bloków operacji "splotu", agregacji, itp. Dążymy do uzyskania najlepszych cech dla MLP.



Plan wykładu:

- Splot 2D
- Warstwy i sieci splotowe
- Pole recepcyjne i warstwy agregacji
- **Podsumowanie**

Podsumowanie

Sieci splotowe umożliwiają wyodrębnienie cech z danych źródłowych poprzez uczenie parametrów filtrów (masek dla operacji splotu czy korelacji wzajemnej).

Celem jest redukcja wymiaru przestrzennego na rzecz wymiaru reprezentującego różnorodność cech.

Wprowadza się operacje agregacji celem redukcji wymiaru przestrzennego, zmniejszenie wpływu lokalnych zmian danych, zwiększenie pole recepcyjnego, itp.

Sieci splotowe mogą zwierać liczne warstwy wymagające uczenia wag masek splotu. Duża liczba parametrów może prowadzić do problemu przeuczenia sieci (ang. overfitting). Dlatego projektowanie czy dobór modelu należy odpowiednio dostosować do rozmiaru i charakteru danych wejściowych.

Można stosować dodatkowe warstwy i techniki w celu zmniejszenia przeuczenia sieci czy zwiększenia efektywności modelu. Skupimy się na tym w czasie kolejnych spotkań.

Dziękuję

Jacek Rumiński



Fundusze
Europejskie
Polska Cyfrowa

Rzeczpospolita
Polska



KANCELARIA PREZESA RADY MINISTRÓW
THE CHANCELLERY OF THE PRIME MINISTER

Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Uczenie głębokie

Wykład 3: Sieci splotowe – architektury modeli

Jacek Rumiński
Katedra Inżynierii Biomedycznej, Wydział ETI



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

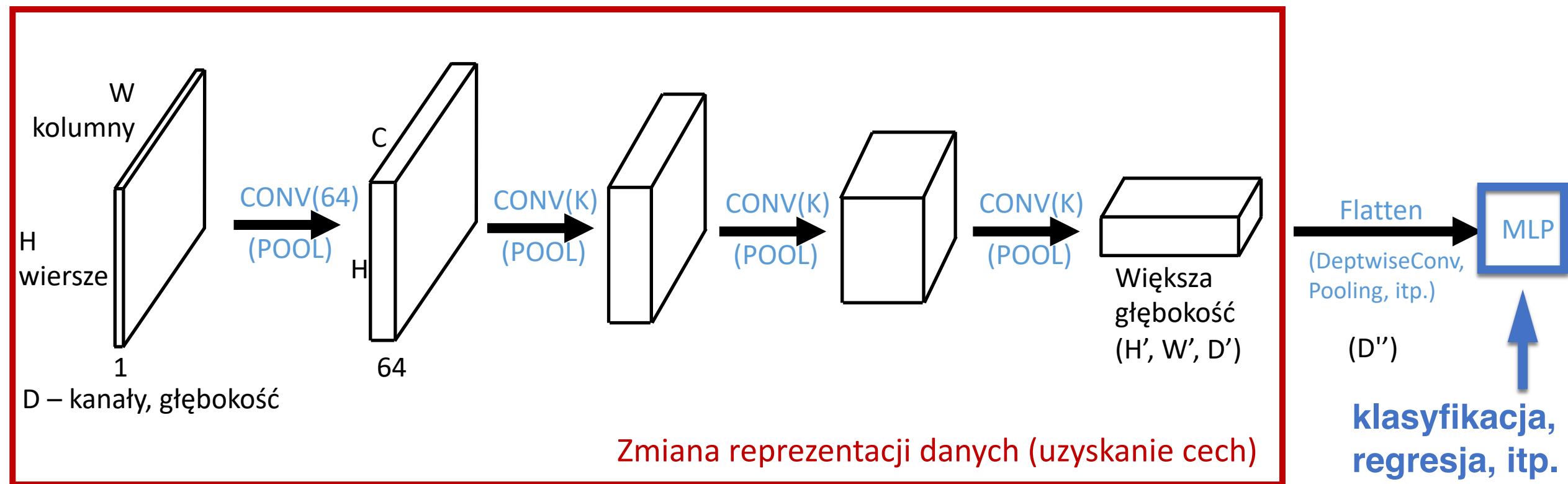
Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu:

- **Praktyczne aspekty treningu sieci splotowych**
- Model LeNet-5
- Nadmierne dopasowanie modelu do danych
- Wczesne zatrzymywanie procesu uczenia
- Augmentacja danych

Sieci splotowe

Architekturę modelu sieci splotowej tworzymy na bazie poznanych operacji – warstw/bloków operacji "splotu", agregacji, itp. Dążymy do uzyskania najlepszych cech dla MLP.



Przykłady praktyczne (notatnik interaktywny DL_03_01_CNN_Intro.ipynb)

Sieci splotowe w środowisku Google Colaboratory/ Jupyter Notebook.

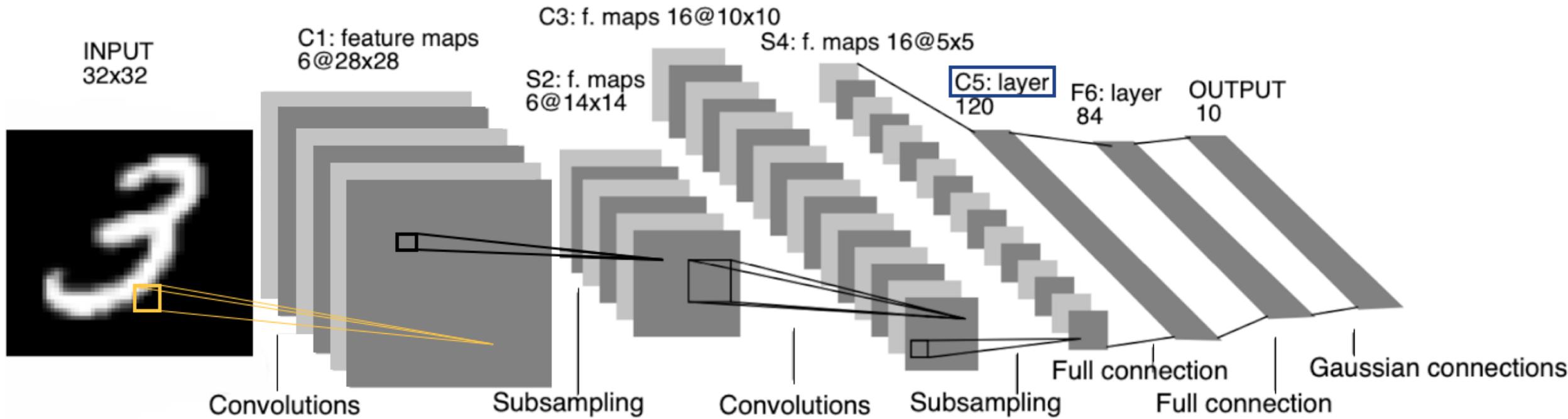
https://colab.research.google.com/drive/1V9rAL2UzFNoi_IWOFq9mNFkAx2-2YQrO?usp=sharing

Plan wykładu:

- Praktyczne aspekty treningu sieci splotowych
- **Model LeNet-5**
- Nadmierne dopasowanie modelu do danych
- Wczesne zatrzymywanie procesu uczenia
- Augmentacja danych

Architektury modeli CNN - LeNet-5 (Yann LeCun, et al., 1998 [1][2])

Model ten składał się z 7 warstw (nie licząc wejścia) i zastosowano go dla prostego zbioru danych - bazy MNIST. LeCun wykorzystał wersję bazy o rozmiarach obrazów 32x32 (obrazy w skali szarości). Architekturę zilustrowano poniżej (na podstawie [2]).



[1] <https://ieeexplore.ieee.org/document/726791>

[2] http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

Architektury modeli CNN - LeNet-5 - przykład na CIFAR10

Implementacja modelu częściowo bazującego na LeNet-5:

```
def get_LeNet5_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(6, kernel_size=5, activation='relu', input_shape=x_train.shape[1:], name="C1"),
        tf.keras.layers.AveragePooling2D(pool_size=(2,2), name="S1"),
        tf.keras.layers.Conv2D(16, kernel_size=5, activation='relu', name="C3"),
        tf.keras.layers.AveragePooling2D(pool_size=(2,2), name="S4"),
        ## use this version (2 layers)
        tf.keras.layers.Conv2D(120, kernel_size=5, activation='relu', name="C5"),
        tf.keras.layers.Flatten(), # make data flat to use dense layers
        # OR this version instead (2 layers)
        # tf.keras.layers.Flatten(), # make data flat to use dense layers
        # tf.keras.layers.Dense(120, activation = 'relu', name="C5"),
        tf.keras.layers.Dense(84, activation='relu'),
        tf.keras.layers.Dense(no_of_classes, activation='softmax')
    ])
    return model
```

Architektury modeli CNN - LeNet-5

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
C1 (Conv2D)	(None, 28, 28, 6)	456
S1 (AveragePooling2D)	(None, 14, 14, 6)	0
C3 (Conv2D)	(None, 10, 10, 16)	2416
S4 (AveragePooling2D)	(None, 5, 5, 16)	0
C5 (Conv2D)	(None, 1, 1, 120)	48120
flatten (Flatten)	(None, 120)	0
dense (Dense)	(None, 84)	10164
dense_1 (Dense)	(None, 10)	850
<hr/>		

Total params: 62,006

Trainable params: 62,006

Non-trainable params: 0

Architektury modeli CNN - LeNet-5

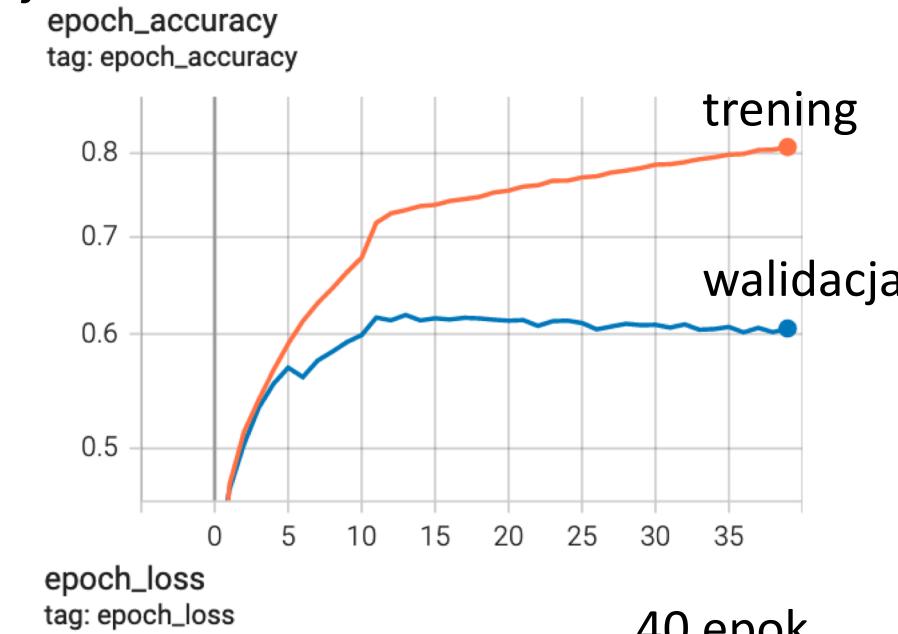
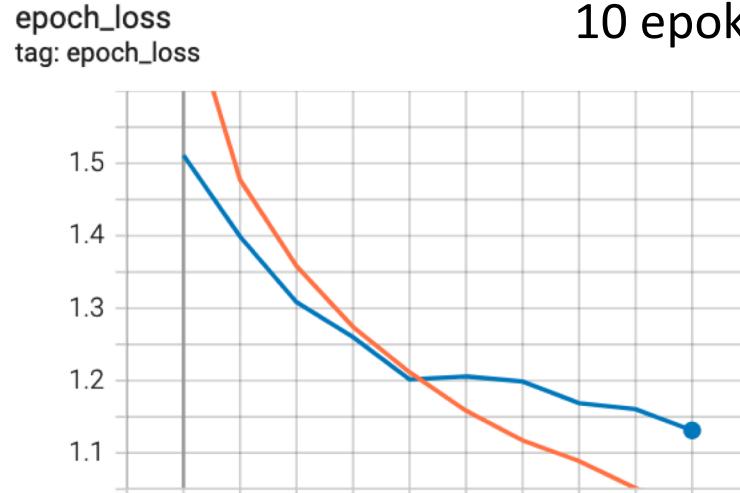
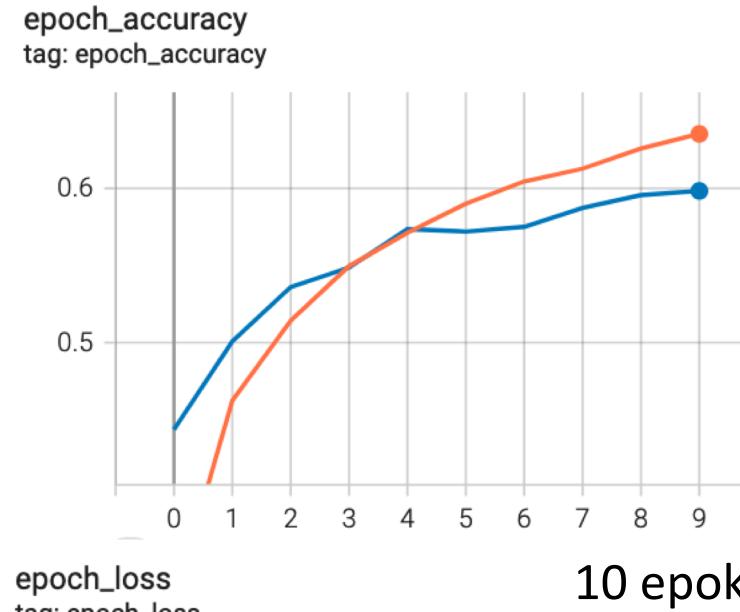
Zwróćmy uwagę na istotny szczegół: warstw C5 to warstwa splotowa (niektóre osoby odwołują się do modelu LeNet-5 i zamiast warstwy splotowej wprowadzają warstwę Flatten, a za nią Dense(120)).

Na wejście warstwy splotowej C5 trafiają dane z warstwy S4 - rozmiar przestrzenny (5x5), liczba kanałów 16. Możemy teraz zrealizować 5 warstwę w modelu LeNet-5 poprzez:

- 1.) Zastosować operację splotu Conv2D z liczbą 120 filtrów o rozmiarze 5x5 - w rezultacie każdy filtr jest W PEŁNI połączony (Fully Connected) z danymi wejściowymi ($16 \times (5 \times 5)$). Mamy zatem 120 neuronów, każdy operuje na ($16 \times (5 \times 5)$) próbkach. Ponieważ mamy 120 filtrów/neuronów dlatego uzyskamy 120 próbek na wyjściu.
- 2.) ALBO możemy zastosować operację spłaszczenia danych z S4 ($16 \times (5 \times 5)$), a następnie możemy użyć warstwę Dense (Fully Connected) dla 120 neuronów.

Architektury modeli CNN - LeNet-5 - przykład na CIFAR10

Problem nadmiernego dopasowania modelu do danych



Plan wykładu:

- Praktyczne aspekty treningu sieci splotowych
- Model LeNet-5
- **Nadmierne dopasowanie modelu do danych**
- Wczesne zatrzymywanie procesu uczenia
- Augmentacja danych

Architektury modeli CNN - LeNet-5 - OVERFITTING

Problem nadmiernego dopasowania modelu do danych

Dla trenowanych wyżej modeli LeNet-5 obserwowaliśmy sytuację, w której występowało zjawisko nadmiernego dopasowania. Przez kilka epok obserwowaliśmy spadek wartości funkcji kosztu zarówno dla danych treningowych jak i dla danych walidacyjnych. Niemniej, począwszy od pewnej epoki mogliśmy zauważać wzrost wartości funkcji kosztu dla danych walidacyjnych.

W trakcie realizacji wcześniejszych zajęć poznali Państwo wybrane metody redukcji niekorzystnego zjawiska nadmiernego dopasowania modelu do danych. W tej części materiału skupimy się na wybranych technikach modyfikacji modelu czy procesu uczenia tak, aby zmniejszać koszt (błąd) walidacji modelu nie zwiększając znacząco błędu treningu.

Techniki tego rodzaju nazywane są **technikami regularyzacji** (ang. regularization).

Architektury modeli CNN - OVERFITTING

Problem nadmiernego dopasowania modelu do danych

Do grona technik regularyzacji zaliczymy m.in.:

- wczesne zakończenie treningu (ang. Early Stopping),
- augmentacji danych (ang. Data augmentation),
- logiczną i losową redukcję neuronów (ang. Dropout),
- normalizację porcji danych (ang. Batch Normalization),
- regularyzację funkcji kosztu i wag (regularyzacja L1, L2, itp.).

Przedstawmy powyższe techniki z punktu widzenia zastosowania w projektowaniu architektur CNN i treningu modeli CNN. Zaczniemy od metody wczesnego zatrzymywania procesu uczenia.

Plan wykładu:

- Praktyczne aspekty treningu sieci splotowych
- Model LeNet-5
- Nadmierne dopasowanie modelu do danych
- **Wczesne zatrzymywanie procesu uczenia**
- Augmentacja danych

Architektury modeli CNN - wcześniejsze zatrzymanie procesu uczenia (ang. Early Stopping)

W sytuacji, gdy błąd walidacji znacznie rośnie po pewnej epoce, można zatrzymać proces uczenia (skoro i tak nie wnosi poprawy - wręcz odwrotnie).

W takim przypadku powinniśmy:

- Określić parametry (kryteria) zatrzymania procesu treningu
- Regularnie zapisywać najlepszy model.

Prostym kryterium zatrzymania może być wykrycie epoki, w której błąd walidacji jest większy niż w kroku poprzednim.

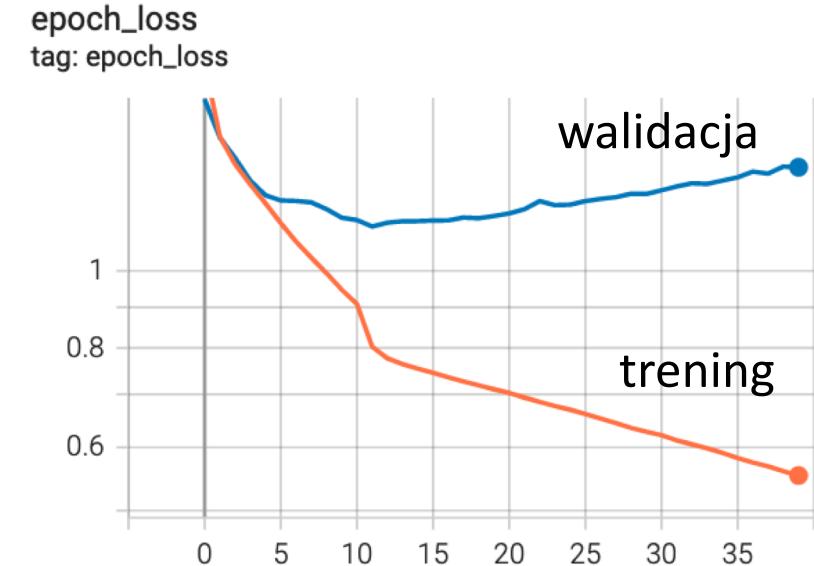
Niech:

$$E_{OPT}(t) = \min_{t' \leq t} (E_{VAL}(t'))$$

oznacza najmniejszy błąd walidacji uzyskany do epoki t.

Możemy zdefiniować również błąd generalizacji dla epoki t jako:

$$GL(t) = 100 \cdot \left(\frac{E_{VAL}(t)}{E_{OPT}(t)} - 1 \right)$$



Architektury modeli CNN - wcześniejsze zatrzymanie procesu uczenia (ang. Early Stopping)

Dla tak zdefiniowanego błędu możemy wyznaczyć kryterium zatrzymania treningu jako [1]:

Zatrzymaj po pierwszej epoce t , dla której prawdziwy jest warunek: $GL(t) > \varepsilon$

gdzie: ε - przyjmowana a priori wartość progowa dla GL.

Kolejne kryterium można zdefiniować następująco [1]:

Zatrzymaj, jeśli w s kolejnych krokach rośnie błąd walidacji:

UP_s - zatrzymaj po t jeśli wykryto warunek zatrzymania dla UP_{s-1} poi epoce $t-k$ oraz

$$E_{VAL}(t) > E_{VAL}(t - k)$$

[1] Prechelt L. (1998) Early Stopping - But When?. In: Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 1524. Springer, https://page.mi.fu-berlin.de/prechelt/Biblio/stop_tricks1997.pdf

Architektury modeli CNN - wcześniejsze zatrzymanie procesu uczenia (ang. Early Stopping)

Mechanizm zatrzymania procesu uczenia implementuje się często podobnie do wyżej określonych kryteriów. Przykładowo w TF możemy zdefiniować funkcje typu callback:

```
tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', min_delta=0, patience=0, verbose=0,  
    mode='auto', baseline=None, restore_best_weights=False  
)
```

dla której określamy parametry:

- monitor - monitorowana wielkość - miara błędu
- min_delta - minimalna (bezwzględna) zmiana błędu uznawana za poprawę błędu.
- patience - liczba epok, w których nie będzie obserwowana poprawa błędu zanim nastąpi przerwanie treningu.

Przykładowo: `callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)`

Architektury modeli CNN - wcześniejsze zatrzymanie procesu uczenia (ang. Early Stopping)

Zapis stanu parametrów lub całego modelu jest ważny w związku z możliwością wcześniejszego przerwania procesu uczenia (np. w rezultacie zastosowania określonego kryterium zrealizowane poprzez funkcję typu callback - pogorszenie błędu walidacji, bardzo długi czas uczenia, itp).

Popularne biblioteki wspierające uczenie głębokie oferując odpowiednie metody w tym zakresie. Przykładowo w TF możemy zastosować (https://www.tensorflow.org/tutorials/keras/save_and_load):

```
tf.keras.callbacks.ModelCheckpoint(  
    filepath, monitor='val_loss', verbose=0, save_best_only=False,  
    save_weights_only=False, mode='auto', save_freq='epoch',  
    options=None, **kwargs  
)
```

Funkcja ta umożliwia zapis całego modelu (save_weights_only=False) lub tylko wag w określonych sytuacjach, np. po każdej epoce (save_freq='epoch'), z którąś epokę albo zapis w sytuacji uzyskania najlepszego modelu według przyjętej miary (np. monitor='val_loss') i określonego kryterium (np. mode='min').

Architektury modeli CNN - wcześniejsze zatrzymanie procesu uczenia (ang. Early Stopping)

Zapis stanu parametrów lub całego modelu można również zrealizować poprzez wywołanie prostych funkcji:

```
model.save_weights("./folder/file")
```

Zapisane wagi można wczytać do instancji utworzonego (zgodnego) modelu, który często implementujemy za pomocą określonej funkcji, np. nazwijmy ją get_model(). Wówczas:

```
model = get_model()  
# Read weights  
model.load_weights("./folder/file")
```

Modele (wagi) zapisywane są w domyślnym formacie pakietu TensorFlow (.ckpt, SavedModel). Zapis wag i modeli możliwy jest również w innych formatach np. HDF5 (https://www.tensorflow.org/guide/keras/save_and_serialize#weights_only_saving_in_savedmodel_format).

Architektury modeli CNN - wcześniejsze zatrzymanie procesu uczenia (ang. Early Stopping)

Odczytując utrwaloną postać modelu nie musimy wcześniej tworzyć instancji modelu. Funkcja odczytu pełnego modelu (architektura, ustawienia, wag) zwraca instancję modelu:

```
model.save('serialized_model.h5')  
restored_model = tf.keras.models.load_model('serialized_model.h5')
```

Następnie możemy wykonywać typowe operacje na modelu:

```
restored_model.summary()  
restored_model.evaluate()  
restored_model.predict()
```

Wczytany model można również dodatkowo dokuzać - będzie to istotne, m.in. metodzie uczenia z przeniesieniem (ang. transfer learning), którą wkrótce omówimy.

Plan wykładu:

- Praktyczne aspekty treningu sieci splotowych
- Model LeNet-5
- Nadmierne dopasowanie modelu do danych
- Wczesne zatrzymywanie procesu uczenia
- **Augmentacja danych**

Architektury modeli CNN - Augmentacja danych

Augmentacja danych - poszerzania zbioru danych poprzez przekształcenia istniejących przykładów lub syntezę nowych przykładów z wiedzy a priori (np. z istniejących danych, z modeli opracowanych na istniejących przykładach, itp.).

W przypadku obrazów będzie rozważać możliwości modyfikacji istniejących obrazów przez transformacje geometryczne, nakładanie zniekształceń czy szumu, syntezę nowych obrazów z modeli generacyjnych, itp.

Istnieje szereg pakietów czy funkcji dedykowanych augmentacji obrazów, w tym:

- imgaug - <https://github.com/aleju/imgaug>
- Augmentor - <https://github.com/mdbloice/Augmentor>
- funkcje dostępne w OpenCV, scikit-image, Pytorch, Keras itd.

Augmentację danych można przeprowadzić na wiele sposób, wśród których najczęściej stosuje się:

- **generowanie obrazów poprzez losowo wybrane metody przekształcania obrazów w czasie treningu (generatory)**
- generowanie obrazów poprzez (losowo) wybrane metody przekształcania obrazów w do plików (pamięci) tworząc fizycznie poszerzony zestaw zbioru treningowego.

Architektury modeli CNN - Augmentacja danych

Celem augmentacji jest zwiększenie RÓŻNORODNOŚCI danych, nie zawsze zwiększenie liczby nowych obrazów. W przypadku stosowania generatorów (najczęściej) rozmiar zbioru treningowego nie zwiększa się w danej epoce. Inaczej - w każdej epoce rozmiar zbioru treningowego jest taki sam jak zbioru przed augmentacją. Zmienia się jego zawartość - w każdej epoce mamy wylosowane (inne) operacje przekształcenia danych źródłowych, których zastosowanie generuje określony zestaw danych.

UWAGA - ważne podsumowanie praktyczne dla generatorów:

- w każdej epoce mamy taki sam rozmiar danych treningowych,
- w każdej epoce (typowo) mamy różny zestaw obrazów po (losowo) wybranych operacjach przekształcania obrazów źródłowych,
- zwiększanie liczby epok powoduje, że model będzie widział więcej danych - w każdej bowiem epoce stosowane mogą być (częściowo) inne zestawy przekształceń - LIMIT: wszystkie kombinacje parametrów przekształceń obrazów jakie zostały ustawione przy definicji procesu augmentacji.

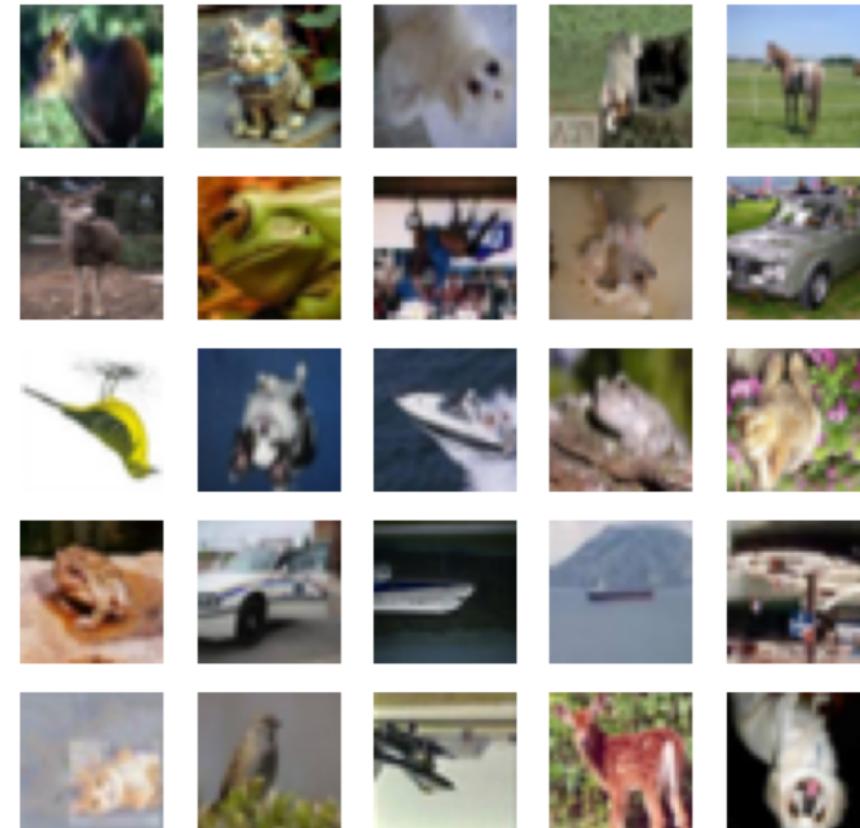
Architektury modeli CNN - Augmentacja danych

W celu zilustrowania procesu augmentacji z wykorzystaniem generatorów posłużmy się klasą `tf.keras.preprocessing.image.ImageDataGenerator` oraz bazą danych CIFAR-10.

```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(vertical_flip=True)
train_iterator = datagen.flow(x_train, y_train, batch_size=64)
batchX, batchy = train_iterator.next()
```

W procesie treningu możemy bezpośrednio dodać operację generacji nowych przykładów:

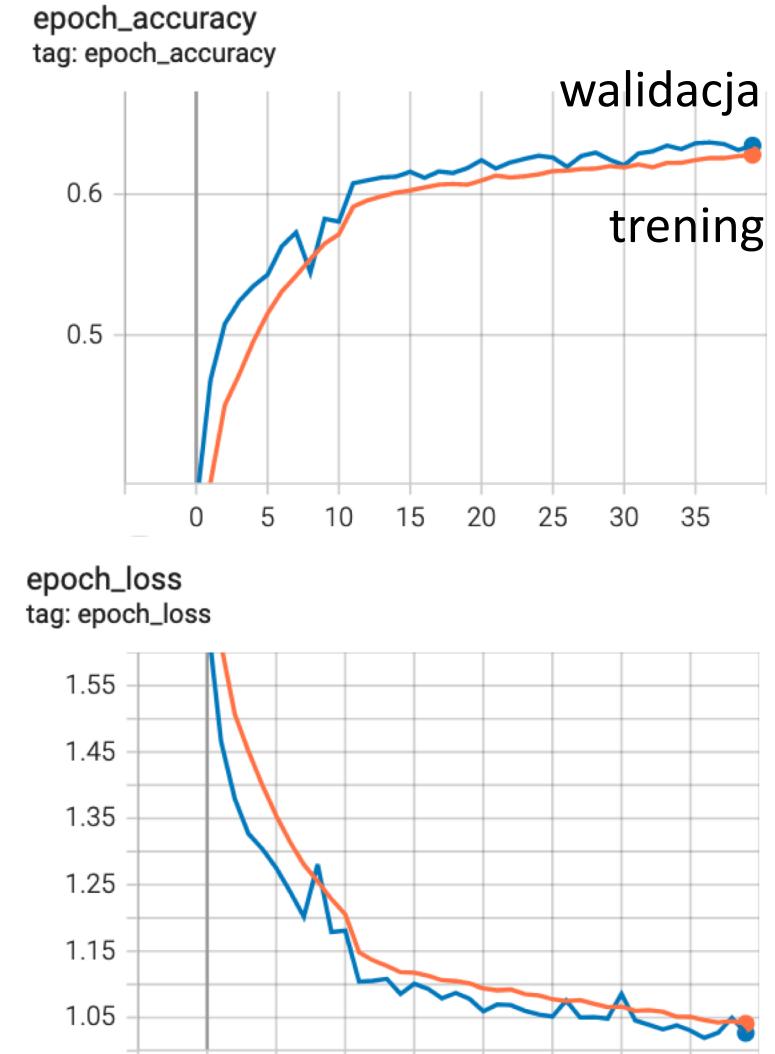
```
model.fit(x=datagen.flow(x_train, y_train,batch_size=batch_size),  
          # y=y_train,  
          epochs=epochs,  
          batch_size=batch_size,  
          # Use test data to perform validation  
          validation_data=(x_test, y_test))
```



Architektury modeli CNN - Augmentacja danych

Na rysunku obok pokazano wyniki treningu dla wcześniejszego modelu (bazującego na LeNet-5) po wprowadzeniu augmentacji danych (odbicie w poziomie i przesunięcia obrazów o 4 piksele).

Możemy zaobserwować, że błąd walidacji jest podobny do błędu treningu (mała wariancja), a jednocześnie ten ostatni nie pogorszył się znacznie względem eksperymentów bez stosowania augmentacji. Oczywiście błąd w obu przypadkach jest względnie wysoki (niska dokładność), stąd w praktyce poszukiwać będziemy innych architektur i ustawień na celu zwiększenie dokładności.



Architektury modeli CNN - Augmentacja danych

Problem wydajności augmentacji - duża różnica czasu uczenia z wykorzystaniem ImageDataGenerator (np. 10s na epokę) i bez (np. 1s na epokę dla tego samego modelu).

W TF warto stosować tf.data lub inne rozwiązania (najlepszych na danym etapie rozwoju pakietów oprogramowania w TF, PyTorch, itp.). Przykładowo w TF można wpisać operacje transformacji obrazów jako warstwy do budowanego modelu, np.:

```
augm_layers = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomFlip("horizontal"),
    layers.experimental.preprocessing.RandomTranslation(0.125, 0.125, fill_mode= constant ')
])
```

Przykłady praktyczne (notatnik interaktywny DL_03_02_CNN_Models_1.ipynb)

Model LeNet, wczesne przerywanie uczenia oraz augmentacja danych w środowisku Google Colaboratory/ Jupyter Notebook.

https://colab.research.google.com/drive/1-21oBYITOclE6Tgk_XTypIH9eTJniCVw?usp=sharing

Dziękuję

Jacek Rumiński



Fundusze
Europejskie
Polska Cyfrowa

Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Uczenie głębokie

Wykład 4: Sieci splotowe – architektury modeli c.d.

Jacek Rumiński
Katedra Inżynierii Biomedycznej, Wydział ETI



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu:

- **Logiczna i losowa redukcja neuronów (ang. dropout)**
- Standaryzacja porcji danych (ang. batch normalization)
- Algorytmy optymalizacyjne
- Regularyzacja, rozpad i normalizacja wag

Architektury modeli CNN - logiczna (i losowa) redukcja neuronów - DROPOUT

W [1] G. Hinton wraz z współpracownikami pokazali znaczenie losowej redukcji neuronów (ang. dropout) na zmniejszenie problemu nadmiernego dopasowania modelu do danych.

W kolejnych epokach losowa wybrana pula neuronów jest eliminowana z prawdopodobieństwem $q=1-p$ (q - dropout rate, p - retain ratio). Zatem można sobie wyobrazić, że w każdej epoce trenowana jest jakby trochę inną architekturą modelu. Na koniec eksperymentów mamy efekt uogólnienia różnych modeli dla wejściowego zestawu danych.

Przypomina to trochę efekt podobny jak dla poznanych w czasie wykładu z uczenia maszynowego metod typu bagging (bootstrap aggregation).

W implementacjach praktycznych nie są usuwane neurony, ale ustawiane są losowo wartości 0 na wylosowanych pozycjach tensora wejściowego danej warstwy. Pozostałe wartości tensora wejściowego są normalizowane (skalowane przez wartości $s=1/(1-q)$).

[1] G. Hinton i inni, 2012, <https://arxiv.org/abs/1207.0580>

Architektury modeli CNN - logiczna (i losowa) redukcja neuronów - DROPOUT

Metodę redukcji neuronów implementuje się jako funkcję w postaci warstwy modelu głębokiego. Przykładowo w TF: `tf.keras.layers.Dropout(rate=0.5)`.

Zapoznajmy się z prostym przykładem:

```
layer_drop = tf.keras.layers.Dropout(rate=0.5, input_shape=(2,))  
simple_input_data = np.arange(8).reshape(4, 2).astype(np.float32)+1  
print("Simple input data: \n", simple_input_data)
```

```
outputs = layer_drop(simple_input_data, training=True)  
print("\nResult of the dropout operation: \n", outputs)
```

Wynik:

Simple input data:
[[1. 2.] [3. 4.] [5. 6.] [7. 8.]]

Result of the dropout operation:

`tf.Tensor([[0. 4.] [6. 0.] [10. 12.] [14. 16.]], shape=(4, 2), dtype=float32)`

Wnioski?

Architektury modeli CNN - logiczna (i losowa) redukcja neuronów - DROPOUT

Operację dropout stosuje się dla każdego neuronu, kontrolując jego udział w sieci według zależności:

$$y_i = \frac{1}{p} \sum_{j=1}^N w_{ij} (\alpha_j \cdot x_j)$$

Gdzie: α_j - niezależna zmienna losowa dla rozkładu Bernoullego, przyjmująca wartości 1 z prawdopodobieństwem p oraz 0 z prawdopodobieństwem q=1-p (dropout rate).

Warstwę typu dropout stosuje się po warstwie splotowej (po funkcji aktywacji) lub po warstwach w pełni połączonych.

Powyższa realizacja redukcji neuronów jest czasami nazywana Drop-neuron. Istnieją jeszcze inne wersje związane z redukcją kanałów (redukujemy kanały danych dla głębokości > 1), redukcją gałęzi (ścieżek) warstw splotowych, itp. (np. [1]).

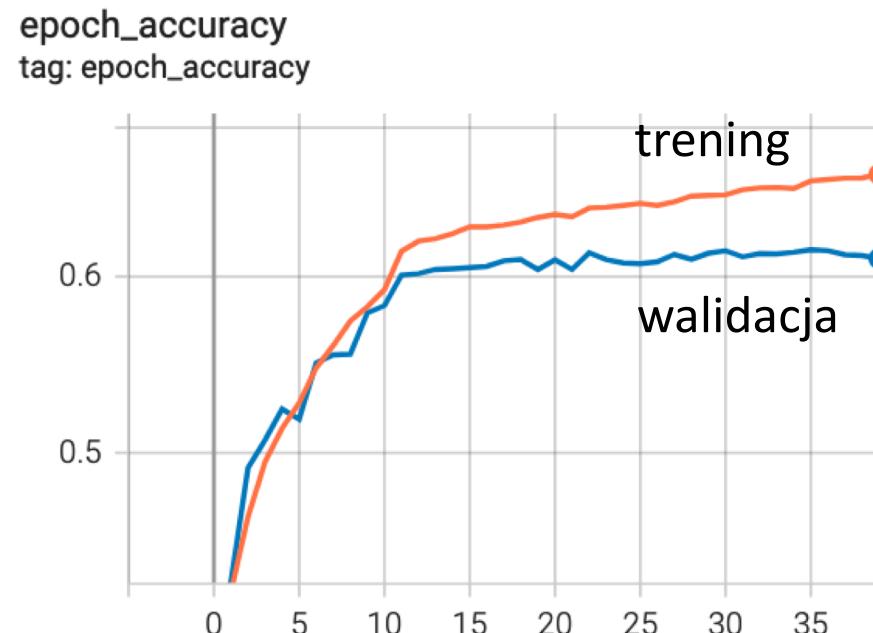
Szereg architektur modeli, które poznamy wkrótce, stosowało lub stosuje warstwę Dropout.

[1] S. Cai et al., (2019), Efficient and Effective Dropout for Deep Convolutional Neural Networks,
<https://arxiv.org/pdf/1904.03392.pdf>

Architektury modeli CNN - logiczna (i losowa) redukcja neuronów - DROPOUT

Model LeNet-5 i dropout (zamiast augmentacji danych):

```
tf.keras.layers.Dropout(0.3),  
tf.keras.layers.Dense(84, activation='relu'),  
tf.keras.layers.Dense(no_of_classes, activation='softmax')
```



Plan wykładu:

- Logiczna i losowa redukcja neuronów (ang. dropout)
- **Standaryzacja porcji danych (ang. batch normalization)**
- Algorytmy optymalizacyjne
- Regularyzacja, rozpad i normalizacja wag

Architektury modeli CNN – standaryzacja porcji danych (ang. Batch Normalization)

W ramach kształcenia z zakresu uczenia maszynowego przedstawiliśmy szereg metod wstępnego przetwarzania danych, w tym metody normalizacji i standaryzacji danych.

W ramach standaryzacji danych (dla epoki - normalizacja dla całej populacji danych) obliczamy wartość średnią każdej cechy μ_t (dla danych treningowych) oraz odchylenie standardowe σ_t , a następnie przekształcamy każdy wartość

$$\hat{x} = \frac{x - \mu_t}{\sigma_t}$$

Stosując metody optymalizacji bazujące na porcjach danych (batch-ach) można standaryzować dane w odniesieniu do porcji danych na różnych etapach modelu wielowarstwowego. Wprowadza to stabilizację rozkładu wartości danych wejściowych dla określonych warstw.

Architektury modeli CNN - standaryzacja porcji danych (ang. Batch Normalization)

W [1] autorzy zaproponowali funkcję standaryzacji porcji danych według następującego algorytmu (ang. Batch Normalization Transform):

BN (podzbiór_przykładów= $x_B = \{x_1, \dots, x_M\}$, parametry={ γ, β }):

1. Oblicz: $\mu_B = \frac{1}{M} \sum_{i=1}^M x_i$ (wartość średnia porcji danych)

2. Oblicz: $\sigma_B^2 = \frac{1}{M} \sum_{i=1}^M (x_i - \mu_B)^2$ (wariancja porcji danych)

3. Normalizuj: $\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$

3. Przekształć wynik: $y_i = \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$

4. Zwróć y_i

Parametry γ, β podlegają procedurze uczenia!!!

Architektury modeli CNN – standaryzacja porcji danych (ang. Batch Normalization)

Autorzy pracy [1] wskazują, że normalizacja porcji danych dla sieci CNN powinna być realizowana po wyjściu modelu liniowego, przez zastosowaniem funkcji aktywacji, tj. operacja $z = a(Wh + b)$

jest zamieniana na:

$$z = a(\text{BN}(Wh)).$$

W praktyce niektóre zespoły wskazują pozytywne wyniki uczenia i inferencji przy stosowaniu BN po funkcji aktywacji. Inni podkreślają, że zależy to może od rodzaju funkcji aktywacji.
Zgodnie z [2] zmienne losowe μ_B oraz σ_B są opisywane rozkładem Gaussa:

$$\mu_B \sim \mathcal{N}\left(\mu_P, \frac{\sigma_P^2}{M}\right), \quad \sigma_B \sim \mathcal{N}\left(\sigma_P, \frac{\rho+2}{4M}\right)$$

gdzie: M - rozmiar batcha, (μ_P, σ_P) parametry dla całej populacji, ρ - kurtoza rozkładu.

[1] S. Ioffe, Ch. Szegedy, (2015), Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, <https://arxiv.org/abs/1502.03167>

[2] M. Teye, et al. (2018). Bayesian uncertainty estimation for batch normalized deep networks, <https://arxiv.org/abs/1802.06455>

Architektury modeli CNN – standaryzacja porcji danych (ang. Batch Normalization)

Zatem dla modeli:

$$\mu_B \sim \mathcal{N}\left(\mu_P, \frac{\sigma_P^2}{M}\right), \quad \sigma_B \sim \mathcal{N}\left(\sigma_P, \frac{\rho+2}{4M}\right)$$

Przyjmując małe wartości rozmiaru batcha M potencjalnie zwiększymy szum. Dlatego poszukuje się najlepszej wartości M tak, aby szum nie był zbyt duży, żeby operacje zaimplementowanego algorytmu optymalizacyjnego przebiegały efektywnie (dopasowując do architektur akceleratorów - np. wielokrotność 32 dla wielu kart GPU NVIDIA), itd.

W [1] autorzy przeanalizowali regularyzacyjny charakter standaryzacji porcji danych. Przeprowadzone eksperymenty dla modeli CNN wykazały, że BN w głębokich sieciach mają cechy regularyzacji.

Architektury modeli CNN – BN i Dropout czy Dropout lub BN

W literaturze i dyskusjach praktyków pojawia się problem stosowania operacji Dropout i BN. Autorzy [1] wskazują, że w przypadku stosowania operacji Dropout przed BN pojawia się przesunięcie wariancji (ang. variance shift) i dlatego rekomendują stosowanie operacji Dropout po wszystkich warstwach typu BN.

Ponadto szereg autorów wskazuje, że stosowanie obu operacji nie zawsze ma sens ponieważ BN przynosi podobne korzyści regularyzacyjne jak Dropout i dlatego takie modele jak np. ResNet nie stosują losowej redukcji neuronów.

Zachęcam do eksperymentów z ciekawym przykładem w tym zakresie dostępnym pod adresem:
https://github.com/adelizer/kaggle-sandbox/blob/master/drafts/dropout_bn.ipynb

Warto natomiast podkreślić, że stabilizacyjne właściwości BN pozwalają stosować większe wartość współczynnika szybkości uczenia i uzyskiwać zbieżność modelu znacznie szybciej.

[1] X. Li, et al. (2018). "Understanding the Disharmony Between Dropout and Batch Normalization by Variance Shift." <http://arxiv.org/abs/1801.05134>.

Architektury modeli CNN - Praktyczne wykorzystanie BN

Operacje czy warstwy BN są implementowane w większości bibliotek dedykowanych uczeniu głębokiemu. W TF operacja BN jest definiowana:

```
tf.keras.layers.BatchNormalization(  
    axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True,  
    beta_initializer='zeros', gamma_initializer='ones',  
    moving_mean_initializer='zeros',  
    moving_variance_initializer='ones', beta_regularizer=None,  
    gamma_regularizer=None, beta_constraint=None, gamma_constraint=None, **kwargs  
)
```

Wynik operacji treningu: $\text{gamma} * (\text{batch} - \text{mean}(\text{batch})) / \sqrt{\text{var}(\text{batch}) + \text{epsilon}} + \text{beta}$

Wynik operacji inferencji: $\text{gamma} * (\text{batch} - \text{self.moving_mean}) / \sqrt{\text{self.moving_var} + \text{epsilon}} + \text{beta}$

Średnia i wariancja jest aktualizowana przy każdym użyciu danej warstwy w czasie treningu:

$$\begin{aligned}\text{moving_mean} &= \text{moving_mean} * \text{momentum} + \text{mean}(\text{batch}) * (1 - \text{momentum}) \\ \text{moving_var} &= \text{moving_var} * \text{momentum} + \text{var}(\text{batch}) * (1 - \text{momentum})\end{aligned}$$

Przykłady praktyczne (notatnik interaktywny DL_04_01_CNN_Models_2.ipynb)

Dropout i standaryzacja porcji danych w środowisku Google Colaboratory/ Jupyter Notebook.

<https://colab.research.google.com/drive/1GkJn1R9kVs6apA6ILPtYuikAkla8VA5I?usp=sharing>

Plan wykładu:

- Logiczna i losowa redukcja neuronów (ang. dropout)
- Standaryzacja porcji danych (ang. batch normalization)
- **Algorytmy optymalizacyjne**
- Regularyzacja, rozpad i normalizacja wag

Architektury modeli CNN - Metody optymalizacji

W poprzednim materiale (notatnikach Jupyter) przedstawiliśmy praktyczną operację definicji funkcji typu callback zmieniającej wartość współczynnika szybkości uczenia (ang. learning rate) wraz ze wzrostem zaawansowania procesu uczenia - czyli wraz ze zmianą epok.

Wprowadzana zmiana może być bezwzględna (np. if epoch < 10: lr = 0.01) lub względna (np. lr = lr * tf.math.exp(-0.02)).

Zmiany takie wprowadzaliśmy dla metody optymalizacji SGD, zakładając np. precyzyjne poszukiwanie rozwiązania wraz z postępem uczenia (zakładamy, że dla wyższych epok jesteśmy bliżej optymalnego rozwiązania i nie chcemy go „przegapić” robiąc zbyt duży krok).

Do tej pory stosowaliśmy głównie metodę optymalizacji SGD. Przyjrzyjmy się teraz pewnym jej odmianom, które w wielu praktycznych scenariuszach osiągać mogą lepsze rezultaty.

Architektury modeli CNN - Metody optymalizacji

W SGD wprowadzamy aktualizację parametrów podlegających uczeniu poprzez:

$$\mathbf{W}^{[s+1]} = \mathbf{W}^{[s]} - update^{[s]}, \quad update^{[s]} = \eta \cdot \nabla \mathcal{L}^{[s]} (\mathbf{W}^{[s]})$$

gdzie:

η - współczynnik szybkości uczenia

$\nabla \mathcal{L}^{[s]} (\mathbf{W}^{[s]})$ - gradient funkcji kosztu względem parametru z \mathbf{W} dla kroku s (dla danej porcji danych - batchem - związanej z krokiem s) - aktualizujemy poszczególne parametry W .

Założymy, że chcielibyśmy uzależnić aktualizację parametrów (z pewną wagą - „pamięcią”) od poprzedniej aktualizacji (tj. z kroku s-1). Wówczas:

$$update^{[s]} = \beta_1 \cdot update^{[s-1]} + \eta \cdot \nabla \mathcal{L}^{[s]} (\mathbf{W}^{[s]})$$

gdzie:

β_1 - współczynnik „pamięci” (bezwładności, ang. momentum factor), typowo $\beta_1 \in [0,1]$.

Architektury modeli CNN - Metody optymalizacji

W znanych implementacjach bibliotek uczenia maszynowego możliwe jest wykorzystanie wersji algorytmu SGD z uwzględnieniem wartości aktualizacji z wcześniejszego kroku (dla $s=0 \rightarrow update^{[s=0]} = 0$), np. w TF:

```
tf.keras.optimizers.SGD(  
    learning_rate=0.01, momentum=0.0, nesterov=False, name='SGD', **kwargs  
)
```

gdzie: momentum to β_1 - współczynnik „pamięci” (bezwładności, ang. momentum factor).

Dodatkowy parametr “nesterov” oznacza wprowadzenie kolejnej odmiany w aktualizacji wartości parametrów (ang. Nesterov's Accelerated Momentum):

$$update^{[s]} = \beta_1 \cdot update^{[s-1]} + \eta \cdot \nabla \mathcal{L}^{[s]} (\mathbf{W}^{[s]} + \beta_1 \cdot update^{[s-1]})$$

Architektury modeli CNN - Metody optymalizacji

Kolejna odmiana algorytmu optymalizacji, znana jako RMSProp [1], wprowadza następującą zmianę:

$$\begin{aligned} g^{[s]} &= \nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]}) \\ v^{[s]} &= \beta_2 \cdot v^{[s-1]} + (1 - \beta_2) \cdot (g^{[s]})^2 \\ update^{[s]} &= \eta \cdot \frac{g^{[s]}}{\sqrt{v^{[s]} + \epsilon}} = \frac{\eta}{\sqrt{v^{[s]} + \epsilon}} \cdot g^{[s]} = \eta' \cdot g^{[s]} \end{aligned}$$

gdzie: ϵ - stała wartość - zabezpieczenie przed niedozwoloną operacją i zwiększenie stabilności obliczeń (realizacja zależy od implementacji), $\beta_2 \in [0,1]$

Zauważmy, że w ostatniej wersji zapisu aktualizacji mamy podkreślenie adaptacyjnego charakteru zmiany gradientu lub zmiany współczynnika szybkości uczenia. Przykładowo w odniesieniu do podstawowej wersji algorytmu SGD w RMSProp wykorzystujemy „modyfikowaną” w czasie procesu uczenia początkową wartość współczynnika szybkości uczenia.

Architektury modeli CNN - Metody optymalizacji

Odmiana ta może również uwzględniać pewną pamięć dotyczącą poprzedniej aktualizacji i wówczas:

$$\begin{aligned} g^{[s]} &= \nabla \mathcal{L}^{[s]} (\mathbf{W}^{[s]}) \\ v^{[s]} &= \beta_2 \cdot v^{[s-1]} + (1 - \beta_2) \cdot (g^{[s]})^2 \\ b^{[s]} &= \beta_1 \cdot b^{[s-1]} + \frac{g^{[s]}}{\sqrt{v^{[s]} + \epsilon}} \\ update^{[s]} &= \eta \cdot b^{[s]} \end{aligned}$$

Implementacja w TF (rho - β_2 , momentum - β_1):

```
tf.keras.optimizers.RMSprop(  
    learning_rate=0.001, rho=0.9, momentum=0.0, epsilon=1e-07, centered=False,  
    name='RMSprop', **kwargs  
)
```

Architektury modeli CNN - Metody optymalizacji

Kolejną ważną wersją algorytmu optymalizacji jest algorytm Adam ([1]). Algorytm ten uwzględnia wcześniejsze aktualizacje łącząc wykorzystanie współczynnika bezwładności i RMSProp:

$$\begin{aligned} g^{[s]} &= \nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]}) \\ m^{[s]} &= \beta_1 \cdot m^{[s-1]} + (1 - \beta_1) \cdot g^{[s]} \\ v^{[s]} &= \beta_2 \cdot v^{[s-1]} + (1 - \beta_2) \cdot (g^{[s]})^2 \\ \hat{m}^{[s]} &= m^{[s]} / (1 - \beta_1^s) \\ \hat{v}^{[s]} &= v^{[s]} / (1 - \beta_2^s) \\ update^{[s]} &= \eta \cdot \frac{\hat{m}^{[s]}}{\sqrt{\hat{v}^{[s]} + \epsilon}} \end{aligned}$$

Również ta popularna wersja algorytmu zaliczana jest do grona metod adaptacyjnych (co łatwo zrozumieć analizując zapisaną postać aktualizacji parametrów).

[1] <https://arxiv.org/abs/1412.6980>

Architektury modeli CNN - Metody optymalizacji

Zwróćmy jeszcze uwagę na znormalizowane wartości:

$$\begin{aligned}\hat{m}^{[s]} &= m^{[s]} / (1 - \beta_1^s) \\ \hat{v}^{[s]} &= v^{[s]} / (1 - \beta_2^s)\end{aligned}$$

Zauważmy, że dla $s=0$, $v^{[s=0]} = 0$ i wówczas

$$\begin{aligned}v^{[s=1]} &= \beta_2 \cdot 0 + (1 - \beta_2) \cdot (g^{[s=1]})^2 = (1 - \beta_2) \cdot (g^{[s=1]})^2 \\ v^{[s=2]} &= \beta_2 \cdot (1 - \beta_2) \cdot (g^{[s=1]})^2 + (1 - \beta_2) \cdot (g^{[s=2]})^2 = (1 - \beta_2) \cdot (\beta_2 (g^{[s=1]})^2 + (g^{[s=2]})^2)\end{aligned}$$

itd. Ogólnie:

$$v^{[s]} = (1 - \beta_2) \sum_{i=1}^s \beta_2^{s-i} \cdot (g^{[i]})^2 \quad (1)$$

Architektury modeli CNN - Metody optymalizacji

W kolejnych krokach s uzyskujemy nowe wartości gradientu $g^{[s]}$. Zakładamy, że mamy pewien rozkład wartości gradientu, dla różnych s. Możemy przeanalizować, czy wartość oczekiwania $E(v^{[s]})$ odpowiada drugiemu momentowi $E((g^{[s]})^2)$. Zgodnie z (1 - poprzedni slajd) (oraz [1]):

$$E(v^{[s]}) = E\left(\left(g^{[s]}\right)^2\right) \cdot (1 - \beta_2^s) + \zeta$$

Jeśli założymy, że możemy kontrolować ζ ($\zeta \rightarrow 0$ np. poprzez dobór kolejnych β_1) wówczas wprowadzamy normalizację przez $(1 - \beta_2^s)$. Stąd w algorytmie Adam wykorzystano taką normalizację: $\hat{v}^{[s]} = v^{[s]} / (1 - \beta_2^s)$ i analogicznie $\hat{m}^{[s]} = m^{[s]} / (1 - \beta_1^s)$.

W TF algorytm Adam implementowany jest jako:

```
tf.keras.optimizers.Adam(  
    learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False,  
    name='Adam', **kwargs  
)
```

[1] <https://arxiv.org/abs/1412.6980>

Plan wykładu:

- Logiczna i losowa redukcja neuronów (ang. dropout)
- Standaryzacja porcji danych (ang. batch normalization)
- Algorytmy optymalizacyjne
- **Regularyzacja, rozpad i normalizacja wag**

Architektury modeli CNN - „Rozpad” wartości parametrów w procesie uczenia

Zarówno w literaturze jak i w bibliotekach programistycznych przedstawianych jest również szereg innych metod czy odmian metod optymalizacji.

W omawianych metodach optymalizacji pokazaliśmy aktualizację wartości parametrów odnosząc się do gradientu funkcji kosztu (dla porcji danych i aktualnych wartości parametrów). Dodatkowo w aktualizacji wartości parametrów możemy w różnym stopniu uwzględnić same wartości tych parametrów, np. [1]:

$$\mathbf{W}^{[s+1]} = (1 - \lambda)\mathbf{W}^{[s]} + update^{[s]}, \quad update^{[s]} = -\eta \cdot \nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]})$$

gdzie:

λ - współczynnik regularyzacji - współczynnik „rozpadu” wartości parametrów (ang. weight decay), $\lambda \in [0,1]$.

Jeśli współczynnik λ jest równy 0, wówczas mamy klasyczną wersję algorytmu.

Architektury modeli CNN - „Rozpad” wartości parametrów w procesie uczenia

Jeśli współczynnik λ jest większy od 0, wówczas nowe wartości parametrów bazują na zmniejszonych (stąd „rozpad”) wartościach parametrów z poprzedniego kroku oraz aktualizacji wynikającej ze zmian funkcji kosztu.

Pomniejszanie wartości parametrów (wag) umożliwia wprowadzenie pewnej kontroli wartości wag w celu zapobiegania problemowi eksplodującego gradientu czy nadmiernego dopasowania modelu do danych.

Problem **eksplodującego gradientu** jest obok problemu **zanikającego gradientu** jednym z dwóch głównych trudności dotyczących „stabilności” gradientu w procesie uczenia.

W przypadku eksplodującego gradientu powstaje sytuacja, w której gradienty niższych warstw (zależne również od gradientów wyższych warstw – propagacja wsteczna!) rosną gwałtownie skutkując tym, że model nie uczy się właściwie.

Eksplodujący gradient wynikać może z b. dużych wartości wag (lub wartości NaN) i dlatego chcemy kontrolować amplitudy tych wartości. Właśnie do tego możemy wykorzystać rozpad wartości wag z zastosowaniem odpowiedniego współczynnika λ .

Architektury modeli CNN - „Rozpad” wartości parametrów w procesie uczenia

W przypadku standardowego algorytmu SGD (tzn. nie adaptacyjnego) wprowadzanie operacji zubożenia wag w procesie uczenia odpowiadać będzie zastosowaniu odpowiedniego funkcji kosztu z czynnikiem regularyzacji L2:

$$\mathcal{L}^{reg[s]}(\mathbf{W}) = \mathcal{L}^{[s]}(\mathbf{W}) + \frac{\lambda}{\eta} \|\mathbf{W}\|_2^2$$

W popularnych bibliotekach uczenia maszynowego obsługiwana jest możliwość regularizacji funkcji kosztu jak i wprowadzenie mechanizmu rozpadu wag.

W PyTorchu przy definicji metody optymalizacji można wprost ustawić wartość współczynnika regularyzacji λ - `weight_decay`, np. dla SGD:

```
torch.optim.SGD(params, lr=<required parameter>, momentum=0, dampening=0, weight_decay=0,  
nesterov=False)
```

W efekcie:

$$update^{[s]} = \eta \cdot \nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]}) + \lambda \mathbf{W}^{[s]}$$

Architektury modeli CNN - „Rozpad” wartości parametrów w procesie uczenia

W przypadku środowiska TF regularyzację wag można wprowadzić do definicji odpowiednich warstw, np.:

```
tf.keras.layers.Dense( num_units,  
kernel_regularizer=tf.keras.regularizers.l2(weight_decay),  
bias_regularizer=tf.keras.regularizers.l2(weight_decay)  
)
```

Zwróćmy uwagę jeszcze na aspekt praktyczny. Jeśli dla warstwy L1, uzyskamy dwie wagi $W_{11}=2$ i $W_{12}=3$ to część regularizacyjna funkcji kosztu z punktu widzenia tej warstwy będzie miała wartość:

$$\text{dla L1: } \|W\|_1 = |2| + |3| = 5$$

$$\text{dla L2: } \|W\|_2 = \sqrt{2^2 + 3^2} = 3,606 \text{ lub w formie } \|W\|_2^2 = 2^2 + 3^2 = 13$$

Jeśli wprowadzimy drugą warstwę L2 z wagami $W_{21}=3$ i $W_{22}=4$, to licząc część regularizacyjną funkcji kosztu jako sumę po warstwach otrzymamy:

$$\text{dla L1: } \|W\|_1 = (|2| + |3|) + (|3| + |4|) = |2| + |3| + |3| + |4| = 12$$

$$\text{dla L2: } \|W\|_2^2 = (2^2 + 3^2) + (3^2 + 4^2) = 2^2 + 3^2 + 3^2 + 4^2 = 37$$

Natomiast dla $\|W\|_2 = \sqrt{2^2 + 3^2 + 3^2 + 4^2} \neq \sqrt{2^2 + 3^2 + 3^2 + 4^2}$

Architektury modeli CNN - „Rozpad” wartości parametrów w procesie uczenia

W przypadku „adaptacyjnych” algorytmów optymalizacji (np. Adam) nie ma bezpośredniej równoważności rozpadu wag i regularyzacji funkcji kosztu.

W [1] autorzy zaproponowali metodę jawnego rozdzielenia (ang. decoupling) mechanizmu rozpadu wag i regularyzacji L2.

Rozpatrzmy tradycyjną wersję SGD:

$$g^{[s]} = \nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]}) + \lambda' \mathbf{W}^{[s]} \text{ (regularyzacja związana z funkcją kosztu)}$$

$$\text{update}^{[s]} = \eta \cdot g^{[s]}$$

$$\begin{aligned} \mathbf{W}^{[s+1]} &= \mathbf{W}^{[s]} - \eta \cdot (\nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]}) + \lambda' \mathbf{W}^{[s]}) = \mathbf{W}^{[s]} - \eta \cdot \nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]}) - \eta \cdot \lambda' \mathbf{W}^{[s]} = \\ &= \mathbf{W}^{[s]} - \eta \cdot \nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]}) - \lambda \mathbf{W}^{[s]} = (1 - \lambda) \mathbf{W}^{[s]} - \eta \cdot \nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]}) \end{aligned}$$

czyli uzyskaliśmy klasyczny „rozpad” wag z współczynnikiem rozpadu: $\lambda = \eta \cdot \lambda'$ i $\lambda' = \frac{\lambda}{\eta}$.

Architektury modeli CNN - „Rozpad” wartości parametrów w procesie uczenia

Natomiast jeśli wprowadzimy do SGD mechanizm pamięci poprzednich aktualizacji (momentum, Nesterov) lub wykorzystamy metody z adaptacyjną zmianą gradientu (np. Adam) nie będzie równoważności tej operacji np. ze względu na częściowe uwikłanie współczynnika szybkości w pośrednie obliczenia, np.:

$$\begin{aligned} g^{[s]} &= \nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]}) + \lambda' \mathbf{W}^{[s]} \quad (\text{regularyzacja związana z funkcją kosztu}) \\ m^{[s]} &= \beta_1 \cdot m^{[s-1]} + \eta \cdot g^{[s]} \quad (\text{w } m^{[s-1]} \text{ jest } \eta \cdot \lambda' \mathbf{W}^{[s-1]}) \\ update^{[s]} &= m^{[s]} \\ \mathbf{W}^{[s+1]} &= \mathbf{W}^{[s]} - update^{[s]} \end{aligned}$$

Dlatego autorzy omawianej publikacji zaproponowali jawne wprowadzenie regularyzacji i rozpadu wag poprzez:

$$\begin{aligned} m^{[s]} &= \beta_1 \cdot m^{[s-1]} + r^{[s]} \cdot \eta \cdot g^{[s]} \\ update^{[s]} &= m^{[s]} + r^{[s]} \cdot \lambda' \mathbf{W}^{[s]} \quad (\text{rozpad wag}) \end{aligned}$$

gdzie: $r^{[s]}$ - dodatkowy mnożnik dla danej epoki (wartość stała lub zmienna).

Architektury modeli CNN - „Rozpad” wartości parametrów w procesie uczenia

W modyfikacji algorytmu Adam (nazwanej AdamW) wprowadzamy analogiczne zmiany :

$$g^{[s]} = \nabla \mathcal{L}^{[s]}(\mathbf{W}^{[s]}) + \lambda \mathbf{W}^{[s]} \text{ (regularyzacja funkcji kosztu)}$$

$$update^{[s]} = r^{[s]} \cdot \left(\eta \cdot \frac{\hat{m}^{[s]}}{\sqrt{\hat{v}^{[s]}} + \epsilon} + \lambda \cdot \mathbf{W}^{[s]} \right)$$

$$\mathbf{W}^{[s+1]} = \mathbf{W}^{[s]} - update^{[s]}$$

gdzie: $r^{[s]}$ - dodatkowy mnożnik dla danej epoki (wartość stała lub zmienna).

Zmodyfikowane wersje metod są zaimplementowane w pakietach TF oraz PyTorch (np. pod nazwą AdamW).

Architektury modeli CNN - „Rozpad” wartości parametrów w procesie uczenia

Na koniec tej części warto również wskazać na ważną modyfikację algorytmu Adam (AdamW) zwaną jako **amsgrad**. W algorytmie Adam założono, że w:

$$update^{[s]} = \eta \cdot \frac{\hat{m}^{[s]}}{\sqrt{\hat{v}^{[s]} + \epsilon}}$$

Wartość $\frac{\eta}{\sqrt{\hat{v}^{[s]} + \epsilon}}$ powinna maleć w ciągu treningu. Żeby to zapewnić w amsgrad wprowadza się dodatkowy zabieg:

$$\hat{v}_{max}^{[s]} = \max(\hat{v}_{max}^{[s]}, \hat{v}^{[s]}) \quad \text{ i } \quad update^{[s]} = \eta \cdot \frac{\hat{m}^{[s]}}{\sqrt{\hat{v}_{max}^{[s]} + \epsilon}}$$

Przykładowe różnice w treningu stosując Adam, AdamW i amsgrad pokazano w [2].

[1] S. J. Reddi, S. Kale, S. Kuma, On the Convergence of Adam and Beyond (2018),
<https://openreview.net/pdf?id=ryQu7f-RZ>

[2] S. Gugger, J. Howard, (2018) AdamW and Super-convergence is now the fastest way to train neural nets
<https://www.fast.ai/2018/07/02/adam-weight-decay/>

Architektury modeli CNN - „Rozpad” wartości parametrów w procesie uczenia

Uwzględnienie **amsgrad** w pakietach TF czy PyTorch odbywa się poprzez ustawienie parametru: amsgrad=True, np. w TF:

```
tf.keras.optimizers.Adam(  
    learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False,  
    name='Adam', **kwargs  
)
```

```
tfa.optimizers.AdamW(  
    weight_decay, learning_rate= 0.001, beta_1=0.9, beta_2=0.999, epsilon: = 1e-07,  
    amsgrad: False, name= 'AdamW', **kwargs  
)
```

W PyTorch:

```
torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)  
torch.optim.AdamW(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.01, amsgrad=False)
```

Architektury modeli CNN – normalizacja wag poszczególnych warstw (ang. weight constraints)

W procesie uczenia wagi zmieniają swoje wartości. Poza regularyzacją również normalizacja wag może przynieść szereg korzyści. Przykładowo normalizacja wag do wartości modułu równego 1 (`tf.keras.constraints.unit_norm`) może przyspieszyć proces zbieżności algorytmu SGD (poprawa uwarunkowania problemu optymalizacyjnego) [1].

Szereg możliwych form normalizacji dostępnych jest w stosownych pakietach oprogramowania. Przykładowo w TF: https://www.tensorflow.org/api_docs/python/tf/keras/constraints

Wybraną metodę normalizacji wag zastosować możemy przy definiowaniu warstw, np.:

```
model.add(Dense(32, kernel_constraint=max_norm(3), bias_constraint=max_norm(3)))
```

[1] T. Salimans, D. P. Kingma, (2016) Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks, <https://arxiv.org/abs/1602.07868>

Przykłady praktyczne (notatnik interaktywny DL_04_02_CNN_Models_Opt.ipynb)

Metody optymalizacji i regularyzacja wag w środowisku Google Colaboratory/ Jupyter Notebook.

https://colab.research.google.com/drive/19X11jlHq5mda_eyV8Oxr6alXzlisip9T?usp=sharing

Dziękuję

Jacek Rumiński



Fundusze
Europejskie
Polska Cyfrowa

Rzeczpospolita
Polska



KPRM
KANCELARIA PREZESA RADY MINISTRÓW
THE CHANCELLERY OF THE PRIME MINISTER

Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Uczenie głębokie

Wykład 5: Sieci splotowe – architektury modeli c.d.

Jacek Rumiński
Katedra Inżynierii Biomedycznej, Wydział ETI



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu:

- **Wybór wartości początkowych parametrów**
- Model AlexNet
- ImageNet
- Głębsze modele CNN

Architektury modeli CNN - Wartość początkowe parametrów

Omówione algorytmy optymalizacji bazujące na SGD zakładają iteracyjną aktualizację wartości wag:

$$\mathbf{W}^{[s+1]} = \mathbf{W}^{[s]} - update^{[s]}.$$

Jaka jednak powinna być wartość początkowa wag, dla kroku $s=0$? Przykładowo: Czy wartość poszczególnych wag powinna być taka sama? Czy wartość parametru obciążenia (b , \mathbf{W}_0) powinna być taka sama jak pozostałych parametrów (\mathbf{W}_1 , \mathbf{W}_2 , ...)?

Moglibyśmy rozpatrzyć dwie skrajne sytuacje: 1) wszystkie wartości początkowe mają taką samą wartość równą 0 oraz 2) wszystkie wartości początkowe mają potencjalnie różne wartości bliskie optymalnym.

Jak się można domyślać, sytuacja 1) może nie doprowadzić do zbieżności iteracyjnego algorytmu optymalizacyjnego, natomiast sytuacja 2) może powodować bardzo szybkie znalezienie optymalnego rozwiązania.

Architektury modeli CNN - Wartość początkowe parametrów

Ponieważ architektura sieci neuronowych bazuje zwykle na zwielenkrotnianiu identycznych lub podobnych jednostek dlatego stosowanie identycznych wartości początkowych dla wag mogłoby prowadzić do takich samych aktualizacji ich wartości w kolejnych krokach i braku zbieżności. Wprowadzając losowo różne wartości początkowe parametrów łamiemy symetrię wynikającą z powtarzalnością jednostek w modelu [1].

Znanych jest wiele metod losowego doboru parametrów dla sieci neuronowych, np.:

- zgodnie z rozkładem Gaussa,
- zgodnie z uciętym rozkładem Gaussa (ang. truncated normal distribution),
- zgodnie z rozkładem równomiernym (jednostajnym, ang. uniform distribution),
- zgodnie z propozycją Glorot and Bengiona (Xavier) dla rozkładu normalnego lub równomiernego,
- zgodnie z propozycją He dla rozkładu normalnego lub równomiernego,
- itd.

Architektury modeli CNN - Wartość początkowe parametrów

Jednym z prostszych rozwiązań byłoby bezpośrednie wykorzystanie rozkładu Gaussa czy rozkładu równomiernego. Niemniej podejście takie nie uwzględnia złożoności modelu (liczby warstw, neuronów, itp.) czy potencjalnych problemów jak zanikający lub eksplodujący gradient. Jak zapewne pamiętamy obliczane wartości gradientu są obliczane przechodząc zgodnie z propagacją wsteczną od wyższych warstw do niższych.

Może się zatem zdarzyć, że wartość aktualizacji (update) dla niższych warstw będzie tak niewielka (zanikająca), że nie będzie postępu w zakresie zmiany wartości wag i w procesie uczenia (**problem zanikającego gradientu**).

Analogicznie wartość aktualizacji (update) może narastać i prowadzić do bardzo dużych zmian wartości wag (**problem eksplodującego gradientu**).

Dlatego staramy się uzyskać podobne rozkłady wartości wyjściowych, aktywacji czy gradientów dla poszczególnych warstw.

Architektury modeli CNN - Wartość początkowe parametrów

Popularne metody ustawiania wartości początkowych parametrów podlegających uczeniu odnoszą się do liczby parametrów w poprzedniej warstwie (a czasami również w następnej warstwie). Rozważmy prosty przykład.

Założymy, że mamy pojedynczą warstwę sieci liniowej z N wejściami (x) i jednym wyjściem (z):

$$z = b + \sum_{j=1}^N w_j x_j$$

Założymy, że x_i i w_i pochodzą z rozkładu normalnego o wartości średnie równej zero (np. standaryzacja; $E(x_j) = 0$, $E(w_j) = 0$). Ponieważ b ma zerową wariancję dlatego:

$$\text{var}(z) = \text{var}\left(\sum_{j=1}^N w_j x_j\right)$$

Ponieważ w i x to zmienne niezależne:

$$\text{var}(w_j x_j) = E(x_j)^2 \text{var}(w_j) + E(w_j)^2 \text{var}(x_j) + \text{var}(w_j) \text{var}(x_j)$$

to zgodnie z powyższemu założeniami:

$$\text{var}(w_j x_j) = \text{var}(w_j) \text{var}(x_j)$$

Architektury modeli CNN - Wartość początkowe parametrów

Ponieważ założyliśmy już wcześniej, że w_i i x_j są niezależne (kowariancja równa zero) dlatego:

$$var(z) = var\left(\sum_{j=1}^N w_j x_j\right) = \sum_{j=1}^N var(w_j x_j)$$

Ponadto zakładaliśmy takie same rozkłady dla x_i i w_i , stąd:

$$var(z) = N \cdot var(w_j x_j) = N \cdot var(w_j) var(x_j)$$

Jeśli dązymy to tego, aby wariancje wejścia i wyjścia warstwy były takie same, dlatego musimy przyjąć, że:

$$N \cdot var(w_j) = 1$$

lub inaczej:

$$var(w_j) = 1/N$$

Architektury modeli CNN - Wartość początkowe parametrów

Jeśli wykorzystujemy dodatkowo funkcję aktywacji (potencjalnie nieliniową, $y=a(z)$), która jednak charakteryzuje się tym, że dla małych wartości wejść $var(az) \approx var(z)$ (np. $\tanh(z)$) wówczas wciąż możemy zastosować wyżej wyprowadzoną zależność.

Zgodnie z przedstawioną metodą wartości wag dla danej warstwy będziemy ustawiać dla rozkładu normalnego o zerowej wartości średniej i odchyleniu standardowym zależnym od liczby neuronów:

$$\mathcal{N} \left[0, \sqrt{\frac{1}{N}} \right]$$

Alternatywnie (w związku z propagacją wsteczną - czyli kontroli wariancji w obu kierunkach) możemy założyć:

$$var(w_j) = \frac{1}{N_{avg}} = \frac{2}{N_{in} + N_{out}},$$

$$N_{avg} = (N_{in} + N_{out})/2$$

Architektury modeli CNN - Wartość początkowe parametrów

W przypadku rozkładu równomiernego, dla zakresu $[-\text{limit}, \text{limit}]$ wariancja wynosi:

$$\text{var}(\mathcal{U}[-\text{limit}, \text{limit}]) = \frac{1}{12}(\text{limit} - (-\text{limit}))^2 = \frac{\text{limit}^2}{3}$$

Dlatego przyjmiemy warunki do generacji wartości początkowych dla parametrów:

$$\text{limit} = \sqrt{\frac{3}{N}}, \quad \text{czyli } \sqrt{\frac{3}{N}} \cdot \mathcal{U}[-1,1]$$

lub w przypadku różnej liczby parametrów warstwy wejściowej i wyjściowej

$$\sqrt{\frac{3}{(N_{in} + N_{out})/2}} \cdot \mathcal{U}[-1,1], \text{ czyli } \sqrt{\frac{6}{N_{in} + N_{out}}} \cdot \mathcal{U}[-1,1]$$

Architektury modeli CNN - Wartość początkowe parametrów

Wyżej przedstawiona metoda generacji wartości początkowych dla parametrów została zaproponowana w pracy [1] i nosi często nazwę pierwszego z autorów pracy, czyli Glorot (nazwisko) lub Xavier (imię). W TF:

```
tf.keras.initializers.GlorotNormal(),  
tf.keras.initializers.GlorotUniform(),
```

Stosuje się ją wtedy, gdy funkcje aktywacji są symetryczne (np. tanh).

Dla innych funkcji aktywacji, np. ReLU, zaproponowano w [2] metodę określania wartości odchylenia standardowego z uwzględnieniem liczby parametrów. Stosowana obecnie nazwa metody ponownie odnosi się na nazwiska pierwszego autora: He.

[1] Xavier Glorot, Yoshua Bengio (2010), Understanding the difficulty of training deep feedforward neural networks, PMLR 9:249-256, <http://proceedings.mlr.press/v9/glorot10a.html>

[2] K. He, X. Zhang et al., “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in International Conference on Computer Vision (ICCV), Feb. 2015, pp. 1026–1034. [Online]. Available: <https://arxiv.org/abs/1502.01852>

Architektury modeli CNN - Wartość początkowe parametrów

Autorzy (He i inni) zaproponowali, aby wartość odchylenia standardowego dla rozkładu normalnego wynosiła:

$$\sqrt{\frac{2}{N}}$$

Natomiast dla rozkładu równomiernego:

$$\sqrt{\frac{6}{N}}$$

(zachęcam do lektury artykułu).

Praktyczne implementacje w TF:

```
tf.keras.initializers.HeNormal(),  
tf.keras.initializers.HeUniform().
```

Architektury modeli CNN - Wartość początkowe parametrów

W literaturze zaproponowano szereg innych metod ustawiania wartości początkowych parametrów (np. [1][2][3]) co świadczy o wadze tematu w problemie uczenia sieci głębokich.

Zainteresowanych zachęcam do lektury również nowszych pozycji, które łatwo można znaleźć w literaturze.

- [1] D. Sussillo and L. F. Abbott. Random walk initialization for training very deep feedforward networks. *arXiv:1412.6558*, 2014
- [2] P.Krähenbühl, C.Doersch, J.Donahue, and T.Darrell. Data dependent initializations of convolutional neural networks. *arXiv:1511.06856*, 2015.
- [3] D. Mishkin and J. Matas. All you need is a good init. Proceedings of ICLR 2016 Conference, *arXiv:1511.06422v7 [cs.LG]*, 2016

Przykłady praktyczne (notatnik interaktywny DL_05_01_CNN_Models_Init.ipynb)

Ustawianie wartości początkowych parametrów modeli CNN w środowisku Google Colaboratory/Jupyter Notebook.

https://colab.research.google.com/drive/1gWg0crANLlny_SAaf_QQo9KYDjkChDJf?usp=sharing

Plan wykładu:

- Wybór wartości początkowych parametrów
- **Model AlexNet**
- ImageNet
- Głębsze modele CNN

Architektury modeli CNN - AlexNet

Wprowadziliśmy szereg zagadnień, które umożliwią nam omówienie kolejnych architektur modeli sieci splotowych.

W [1] przedstawiono architekturę modelu bazującego na warstwach splotowych, dedykowaną rozpoznawania obrazów z bazy ImageNet [2] dla 1000 klas. W pracy wykorzystano wersję: ImageNet Large-Scale Visual Recognition Challenge - ILSVRC, złożoną z: ok. 1.2 miliona obrazów w zbiorze treningowym, 50,000 obrazów w zbiorze walidacyjnym i 150,000 przykładów w zbiorze testowym.

Opracowany model składa się z 5 warstw splotowych i 3 warstw w pełni połączonych (w tym warstwa wyjściowa z 1000 jednostek dla 1000 klas.). Model został nazwany ponownie uwzględniając dane pierwszego autora prac jako AlexNet.

[1] Alex Krizhevsky Ilya Sutskever Geoffrey E. Hinton, (2012) ImageNet Classification with Deep Convolutional Neural Networks,

<https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>

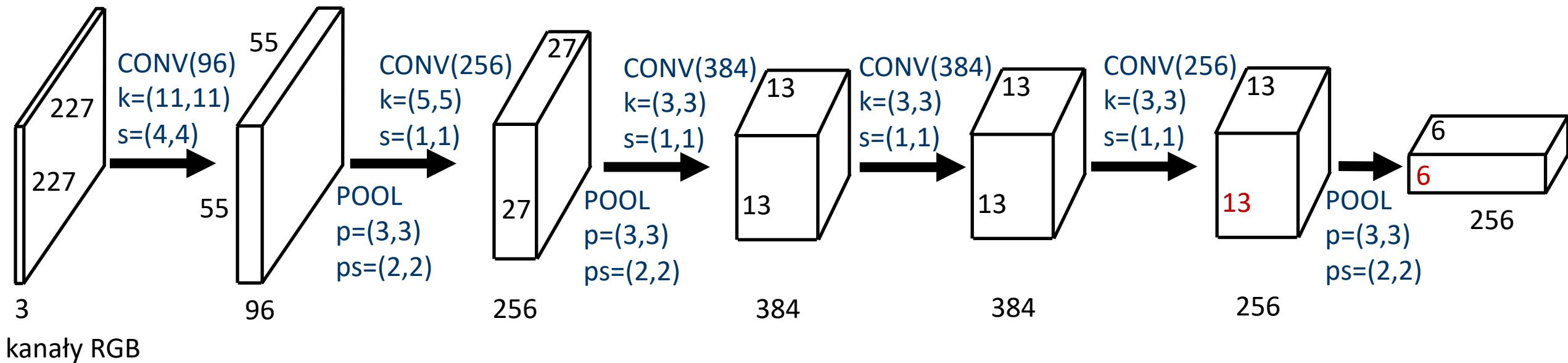
[2] ImageNet, <https://www.image-net.org>

Architektury modeli CNN - AlexNet

Warstwy splotowe modelu:

k - kernel_size - rozmiar maski splotu, s - stride - rozmiar kroku dla splotu

p - pool_size - rozmiar maski operacji redukcji danych, ps - pool_stride - rozmiar kroku dla redukcji



$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times padding - dilation \times (kernel_size - 1) - 1}{stride} + 1 \right\rfloor$$

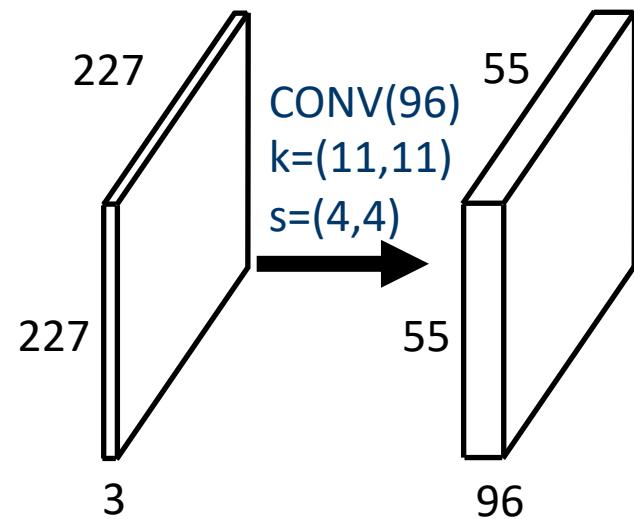
$$L_{out} = \left\lfloor \frac{13 + 2 \times 0 - 1 \times (3 - 1) - 1}{2} + 1 \right\rfloor = \left\lfloor \frac{13 - 2 - 1}{2} + 1 \right\rfloor = 6$$

Architektury modeli CNN - AlexNet

Pobrane z bazy ImageNet obrazy były przeskalowywane do rozmiaru 256x256, z których autorzy pobierali segmenty (patch) o rozmiarach 227x227^(*).

Pierwsza operacja splotu stosowała 96 masek o rozmiarach (11x11) oraz krok (stride) o 4 dla każdego wymiaru (4,4). Dlatego w wyniku tej operacji rozmiar przestrzenny wyjścia to:

$$L_{out} = \left\lfloor \frac{227 - 1 \times (11 - 1) - 1}{4} + 1 \right\rfloor = \left\lfloor \frac{216}{4} + 1 \right\rfloor = 55$$

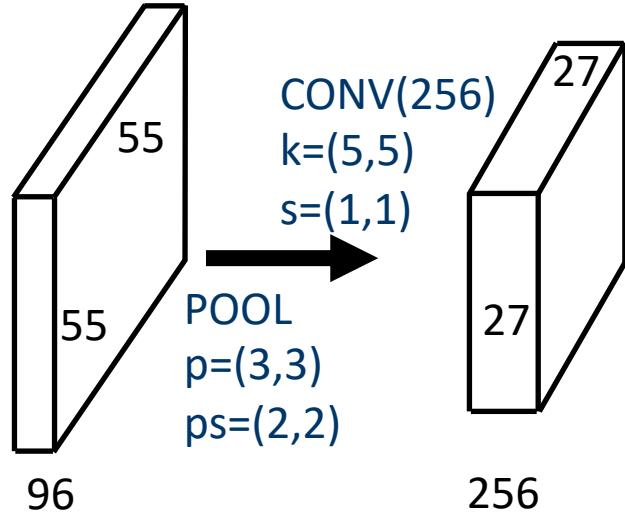


Warto dodać, że ze względu na dostępny rozmiar pamięci kart GPU autorzy podzielili model na dwie połowy (48 masek dla GPU1, 48 dla GPU2, i analogicznie po połowie dla kolejnych warstw splotowych).

(*) Uwaga! W artykule autorzy wskazują, że pobierają patche o rozmiarach 224x224 i taki jest rozmiar wejścia sieci. Jednak nie zgadza się to z dalszym opisem modelu (wymiar po splocie dla pierwszej warstwy wymaga rozmiaru 227 - [(227-11)/4 + 1]=55). Jeśli wymiar wejścia był 224x224 to autorzy musieli zastosować jakąś dodatkową operację (np. padding o rozmiarze 2).

Architektury modeli CNN - AlexNet

Następnie autorzy zastosowali operację aktywacji z poznaną wcześniej funkcją aktywacji ReLU.



Wynik ReLU, aktywację uzyskaną dla danej maski w danej lokalizacji przestrzennej oddali dodatkowej normalizacji w odniesieniu do aktywacji uzyskanych dla innych maski w tej samej lokalizacji (ang. Local Response Normalization - normalizacja jaskrawości "obrazów").

Następnie przeprowadzono operację redukcji danych z nakładaniem się okien o rozmiarze (3,3) w kolejnym krokach (2,2). (Ponieważ krok jest mniejszy niż rozmiar maski, stąd występuje częściowe nakładanie maski na wartości „widziane” w sąsiedniej lokalizacji).

Dobór parametrów normalizacji, redukcji danych oraz dobór miejsce ich stosowania w ramach architektury przeprowadzono eksperymentalnie (analizując wyniki np. dla zbioru walidacyjnego).

Architektury modeli CNN - AlexNet

W kolejnych warstwach splotowych postępowano analogicznie.

Praktycznie wszystkie operacje zakładały zastosowanie dopełnienia danych w celu utrzymania rozmiaru macierzy. Rozmiar przestrzenny zmieniał się jednak ze względu na stosowane operacje redukcji danych (tj. tam, gdzie je stosowano).

Po warstwach 3 i 4 nie stosowano redukcji danych (pooling), stąd rozmiar przestrzenny jest dla warstw 3, 4 i 5 taki sam (13×13). Po ostatniej warstwie splotowej zastosowano pooling i dlatego końcowy reprezentacja danych bo bloku splotowym to ($6 \times 6 \times 256$).

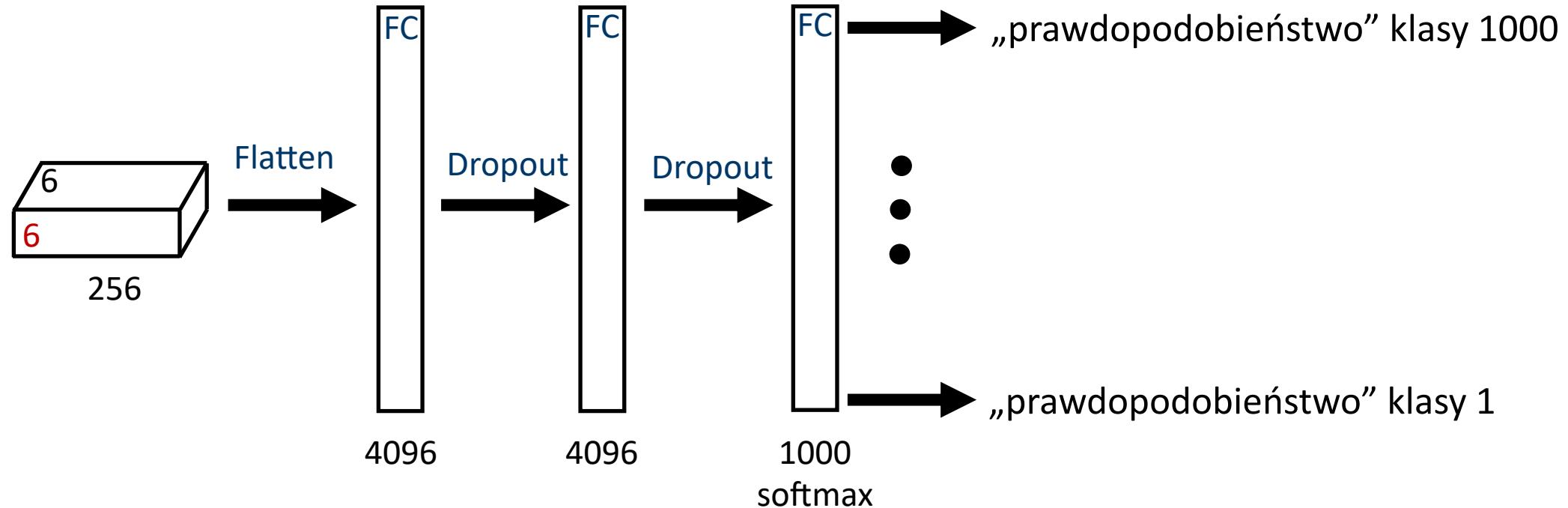
Uzyskana w ten sposób nowa reprezentacja danych poddana była na wejście warstwy w pełni połączonej, co pokazano na kolejnym slajdzie.

Architektury modeli CNN - AlexNet

Warstwy tradycyjnego modelu sztucznych sieci neuronowych modelu AlexNet:

2+1: FC - ang. fully connected - warstwy w pełni połączone

oraz warstwy typu dropout ($r=0,5$) po każdej z ukrytej warstwie FC (czyli 2 razy).



Architektury modeli CNN - AlexNet

Podsumowując architekturę modelu bazującego na AlexNet pokazano poniżej.

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 55, 55, 96)	34944
batch_normalization_5 (Batch Normalization)	(None, 55, 55, 96)	384
max_pooling2d_3 (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_6 (Conv2D)	(None, 27, 27, 256)	614656
batch_normalization_6 (Batch Normalization)	(None, 27, 27, 256)	1024
max_pooling2d_4 (MaxPooling2D)	(None, 13, 13, 256)	0
conv2d_7 (Conv2D)	(None, 13, 13, 384)	885120
batch_normalization_7 (Batch Normalization)	(None, 13, 13, 384)	1536
conv2d_8 (Conv2D)	(None, 13, 13, 384)	1327488
batch_normalization_8 (Batch Normalization)	(None, 13, 13, 384)	1536
conv2d_9 (Conv2D)	(None, 13, 13, 256)	884992
batch_normalization_9 (Batch Normalization)	(None, 13, 13, 256)	1024
		max_pooling2d_5 (MaxPooling2D)
		(None, 6, 6, 256)
		flatten_1 (Flatten)
		(None, 9216)
		dense_3 (Dense)
		(None, 4096)
		37752832
		dropout_2 (Dropout)
		(None, 4096)
		0
		dense_4 (Dense)
		(None, 4096)
		16781312
		dropout_3 (Dropout)
		(None, 4096)
		0
		dense_5 (Dense)
		(None, 1000)
		4097000
<hr/>		
Total params: 62,383,848		
Trainable params: 62,381,096		
Non-trainable params: 2,752		

Plan wykładu:

- Wybór wartości początkowych parametrów
- Model AlexNet
- **ImageNet**
- Głębsze modele CNN

Architektury modeli CNN - AlexNet i ImageNet

Implementację modelu zastosowano w rozpoznawaniu obrazów z bazy ImageNet, z wersji zawierającej 1000 klas, po ok. 1000 przykładów na klasę w zbiorze uczącym.

Wynik modelu dostarcza "prawdopodobieństwa" dla rozpoznania danej klasy. Założmy, że na wejście nauczonego modelu podamy określony obraz. Wówczas na wyjściu uzyskamy wektor zawierający 1000 wartości prawdopodobieństw. Indeks tego wektora łatwo odwzorować na tekstową etykietę klasy. Odpowiednio sortując wektor (pamiętając o indeksach) według malejących wartości prawdopodobieństw można:

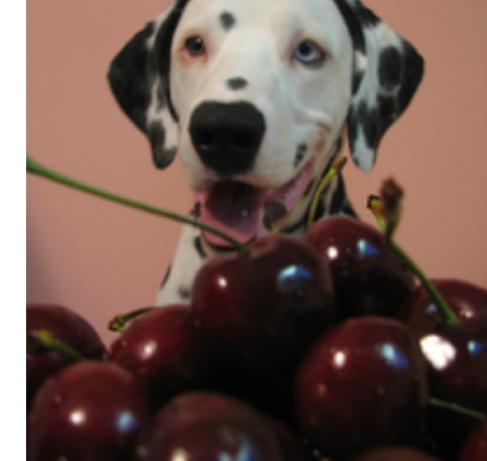
- sprawdzić jaką klasę przypisał model z największym „prawdopodobieństwem” - TOP 1
- sprawdzić jakie klasy przypisał model w grupie 5 pozycji z kolejno największymi „prawdopodobieństwami” - TOP 5.

Mając zbiór testowy, dla którego znamy prawdziwe etykiety klas możemy sprawdzić czy te prawdziwe etykiety są zgodne z:

- z etykietą podaną przez model jako TOP 1 - wówczas obliczamy top-1 error,
- z jedną z etykiet podaną przez model w grupie TOP 5 - wówczas obliczamy top-5 error.

Architektury modeli CNN - AlexNet

Przykłady z bazy ImageNet wraz z rezultatami wskazywanymi przez AlexNet:

		
Prawdziwa etykieta: leopard	Prawdziwa etykieta: mushroom	Prawdziwa etykieta: cherry
Top-5 z AlexNet: leopard (top-1) jaguar cheetah snow leopard Egyptian cat	Top-5 z AlexNet: agaric mashroom (top-2) jelly fungus gill fungus dead-man's-fingers	Top-5 z AlexNet: dalmatian grape elderberry ffordshire bullterrier curran

Architektury modeli CNN - AlexNet

Zastosowanie modelu AlexNet dla bazy ImageNet dało doskonałe wyniki znacznie przekraczające ówczesny stan wiedzy.

Błąd Top-1 Error wyniósł 37,5% (inne modele w tym czasie rzędu 45% i więcej).

Błąd Top-5 Error wyniósł 17,0% (inne modele rzędu >25%).

Było to niewątpliwe przełomowe osiągnięcie zarówno w sferze aplikacyjnej (rozpoznawanie obrazów) jak i w sferze znaczenie modeli głębokich bazujących na warstwach splotowych (CNN).

Przykłady praktyczne (notatnik interaktywny DL_05_02_CNN_Models_AlexNet.ipynb)

AlexNet w środowisku Google Colaboratory/ Jupyter Notebook.

<https://colab.research.google.com/drive/1N3p0H82BhlZQfZ5dpZQTjL57SxSfxodC?usp=sharing>

Plan wykładu:

- Wybór wartości początkowych parametrów
- Model AlexNet
- ImageNet
- **Głębsze modele CNN**

Architektury modeli CNN - coraz głębsze modele

W wyniku sukcesu modelu AlexNet powstało szereg kolejnych propozycji architektur modeli splotowych. Szczególnie ciekawe i przydatne okazały się propozycje wykorzystujące głębsze architektury z zastosowaniem również warstw z maską o rozmiarach (1,1). Wśród modeli tych należy wyróżnić:

- Network in Network [1] - 2013
- GoogLeNet [2] - 2014 (**Top-5 Error: 6.67%**)
- VGG [3] - 2014 (**Top-5 Error: 7.32%**)

[1] Lin, M., Chen, Q., and Yan, S. Network in network. In *Proc. ICLR*, 2014, <https://arxiv.org/abs/1312.4400>

[2] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014, <https://arxiv.org/abs/1409.4842>

[3] K. Simonyan* & Andrew Zisserman (2014), VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION, VGG: <https://arxiv.org/pdf/1409.1556.pdf>

Architektury modeli CNN – coraz głębsze modele

W czasie następnego spotkania poznamy m.in. wybrane aspekty związane z modelami GoogLeNet i VGG. Ten ostatni jest wciąż bardzo popularny w praktycznych zastosowaniach sieci splotowych.

Modele te z powodzeniem pokazały, jakie znaczenie ma głębsza architektura z jednoczesnym ograniczeniem liczby parametrów poprzez stosowanie operacji splotu z mniejszym rozmiarem maski (3×3 i 1×1). Jednocześnie sekwencje takich warstw pozwoliły powiększyć pole recepcyjne, zgodnie z naszym wcześniejszym materiałem.

Szczególnie ważne okazały się warstwy splotowe z rozmiarem maski $(1, 1)$. Omówimy to w czasie kolejnego spotkania.

Dziękuję

Jacek Rumiński



Fundusze
Europejskie
Polska Cyfrowa

Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Uczenie głębokie

Wykład 6: Sieci splotowe – architektury modeli c.d.

Jacek Rumiński
Katedra Inżynierii Biomedycznej, Wydział ETI



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu:

- **VGG**
- Inception
- ResNet
- Rozwój modeli CNN

Architektury modeli CNN – coraz głębsze modele

W wyniku sukcesu modelu AlexNet powstało szereg kolejnych propozycji architektur modeli splotowych. Szczególnie ciekawe i przydatne okazały się propozycje wykorzystujące głębsze architektury z zastosowaniem również warstw z maską o rozmiarach (1,1). Wśród modeli tych należy wyróżnić:

- Network in Network [1] (specjalne zastosowanie i znaczenie splotu z maską 1x1) - 2013
- GoogLeNet [2] - 2014 (**Top-5 Error: 6.67%**)
- VGG [3] - 2014 (**Top-5 Error: 7.32% - ILSVRC test error**)

- [1] Lin, M., Chen, Q., and Yan, S. Network in network. In *Proc. ICLR*, 2014, <https://arxiv.org/abs/1312.4400>#
- [2] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014, <https://arxiv.org/abs/1409.4842>
- [3] K. Simonyan & Andrew Zisserman (2014), VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION, VGG: <https://arxiv.org/pdf/1409.1556.pdf>

Architektury modeli CNN - VGG - architektura i techniki uczenia

W VGG [1] autorzy zaproponowali szereg głębszych (niż w AlexNet) modeli splotowych. Najlepsze w testach konfiguracje pokazano w tabeli obok.

Warto zwrócić uwagę, na bardzo dużą liczbę parametrów podlegających uczeniu:

- dla C: 134 milionów
- dla D: 138 milionów
- dla E: 144 milionów

Ponad 2 razy więcej param. niż dla AlexNet.

[1] K. Simonyan & Andrew Zisserman (2014),
VERY DEEP CONVOLUTIONAL NETWORKS FOR
LARGE-SCALE IMAGE RECOGNITION, VGG:
<https://arxiv.org/pdf/1409.1556.pdf>

C - 16 warstw (wag)	D - 16 warstw (wag)	E - 19 warstw (wag)
Wejście: obrazy RGB (224x224x3)		
Conv2D(k=(3x3),f=64) Conv2D(k=(3x3),f=64)	Conv2D(k=(3x3),f=64) Conv2D(k=(3x3),f=64)	Conv2D(k=(3x3),f=64) Conv2D(k=(3x3),f=64)
MAX POOLING		
Conv2D(k=(3x3),f=128) Conv2D(k=(3x3),f=128)	Conv2D(k=(3x3),f=128) Conv2D(k=(3x3),f=128)	Conv2D(k=(3x3),f=128) Conv2D(k=(3x3),f=128)
MAX POOLING		
Conv2D(k=(3x3),f=256) Conv2D(k=(3x3),f=256) Conv2D(k=(1x1),f=256)	Conv2D(k=(3x3),f=256) Conv2D(k=(3x3),f=256) Conv2D(k=(3x3),f=256)	Conv2D(k=(3x3),f=256) Conv2D(k=(3x3),f=256) Conv2D(k=(3x3),f=256) Conv2D(k=(3x3),f=256)
MAX POOLING		
Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512) Conv2D(k=(1x1),f=512)	Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512)	Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512)
MAX POOLING		
Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512) Conv2D(k=(1x1),f=512)	Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512)	Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512) Conv2D(k=(3x3),f=512)
MAX POOLING		
Dense(4096) -> Dense(4096) -> Dense(1000)+softmax		

Architektury modeli CNN - VGG - architektura i techniki uczenia

W środowisku TF łatwo możemy wczytać predefiniowany model VGG i przedstawić jego charakterystykę:

```
base_model = tf.keras.applications.vgg16.VGG16(  
    include_top=True, weights='imagenet', input_tensor=None,  
    input_shape=None, pooling=None, classes=1000,  
    classifier_activation='softmax'  
)  
  
base_model.summary()
```

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

Architektury modeli CNN - VGG - architektura i techniki uczenia

Poszczególne wersje architektur VGG to sekwencje warstw Conv2D z rosnącą liczbą filtrów. Należy zwrócić uwagę na jeden kluczowy aspekt: autorzy zamiast stosować maski o rozmiarze 11×11 czy 5×5 zastosowali zestaw warstw operacji splotu z maską 3×3 . Zestaw warstw wnosi podobne pole receptive jak w przypadku pojedynczej operacji z większą maską. Zmiana jednak jest kluczowa ze względu na:

- zmniejszenie liczby parametrów podlegających uczeniu (jeśli liczba kanałów na wejściu i wyjściu jest taka sama, C to dla dwóch operacji z maską 3×3 mamy: $2 * (3 * 3 * C * C) = 18C^2$, a dla pojedynczej operacji z maską 5×5 mamy: $1 * (5 * 5 * C * C) = 25C^2$),
- zwiększenie elastyczności funkcji decyzyjnej wynikającej z wykorzystania (dla dwóch warstw zamiast jednej) z dwóch instancji nieliniowej funkcji aktywacji.

Autorzy VGG wykorzystali następujące parametry treningu: batch=256, momentum=0,9, weight decay=0,0005, Dropout=0,5 dla dwóch warstw Dense (4096), learning_rate=0,01 (zmniejszany 10 krotnie gdy dokładność dla zbioru walidacyjnego nie poprawiała się - łącznie 3 krotnie zmieniano współczynnik szybkości uczenia), liczba epok do zatrzymania uczenia=74.

Architektury modeli CNN - VGG - architektura i techniki uczenia

Autorzy VGG zaproponowali również 11-warstwową wersję architektury (wersja A), która zawiera mniej parametrów niż wersje bardziej rozbudowane. Wyniki dla wersji A były gorsze niż dla głębszych odmian, ale i tak lepsze niż dla AlexNet.

Parametry uzyskane po trenowaniu wersji A zostały częściowo wykorzystane jako wartości początkowe wag w uczeniu głębszych odmian modeli.

Oprócz nowej architektury sieci autorzy wykazali, że odpowiednie postępowanie z danymi wejściowymi jest ważne w ewaluacji modeli. W szczególności zaproponowali:

- sprawdzenie w pełni splotowego modelu (ang. fully-convolutional net) zamieniając warstwy gęste odpowiednio dobranymi warstwami splotowymi i redukcją danych,
- sprawdzenie wielu skal dla obrazów wejściowych przed przycięciem fragmentu obrazu jako danych wejściowych do modelu (ang. multi-scale),
- sprawdzenie wielu przycięć obrazu źródłowego (fragmentów) jako źródeł danych wejściowych,
- wykorzystanie zespołu modeli (ang. ensemble); w pracy wybrali modele bazujące na D i E.

Architektury modeli CNN - VGG - architektura i techniki uczenia

Autorzy zbadali dwa podejścia w analizie **doboru skali obrazów**:

- stała skala obrazów źródłowych ($S=256$ lub $S=384$),
- zmienna skala obrazów źródłowych losowo wybrana z zakresu ($S_{min}=256$, $S_{max}=512$).

(Ciekawostka: modele z większą skalą uczyły z wartościami początkowymi wag uzyskanymi dla mniejszej skali lub jednej wybranej skali).

Autorzy zbadali dwa podejścia w analizie **doboru fragmentów (przycięć, ang. crop) obrazów** :

- pojedynczy fragment obrazu ,
- 50 różnych fragmentów obrazu (224x224) dla każdej stosowanej skali obrazu (ang. multi-crop)

Wyniki eksperymentów pokazały, że stosowanie w pełni splotowego modelu wraz ze stosowaniem zmiennej skali obrazów i wielu fragmentów pozwoliło autorom uzyskać Top-5 Error = 7,1. Stanowi to lepszy wynik niż ich pierwotny rezultat w konkursie ILSVRC.

Zastosowanie zespołu klasyfikatorów pozwoliło uzyskać błąd równy 7%.

Przykłady praktyczne (notatnik interaktywny DL_06_01_CNN_Models_VGG.ipynb)

Model VGG w środowisku Google Colaboratory/ Jupyter Notebook.

<https://colab.research.google.com/drive/1-y5m2l4vDYHulZqFEYqGMPC-i8Kojt7T?usp=sharing>

Plan wykładu:

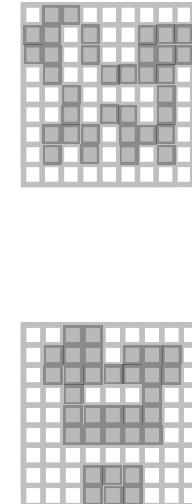
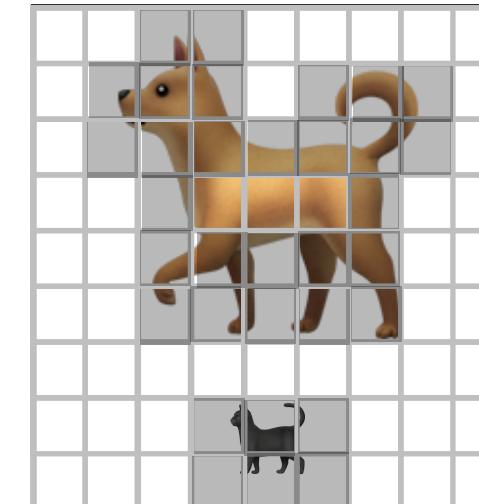
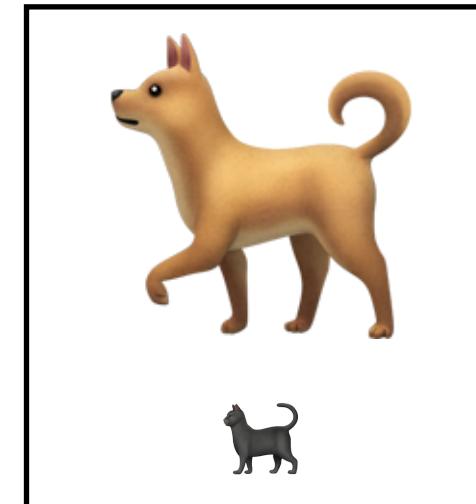
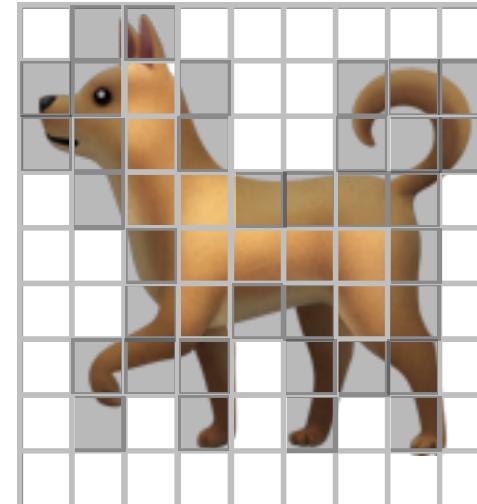
- VGG
- **Inception**
- ResNet
- Rozwój modeli CNN

Architektury modeli CNN - moduł incepacji

W sieciach splotowych stosowaliśmy do tej pory sekwencje kolejnych warstw splotowych. Przykładowo, definiując warstwę Conv2D (filters=96, ...) wprowadzaliśmy 96 masek filtrów działających na tym samym wejściu. Wszystkie jednak maski współdzieliły podstawową konfigurację, np. rozmiar maski.

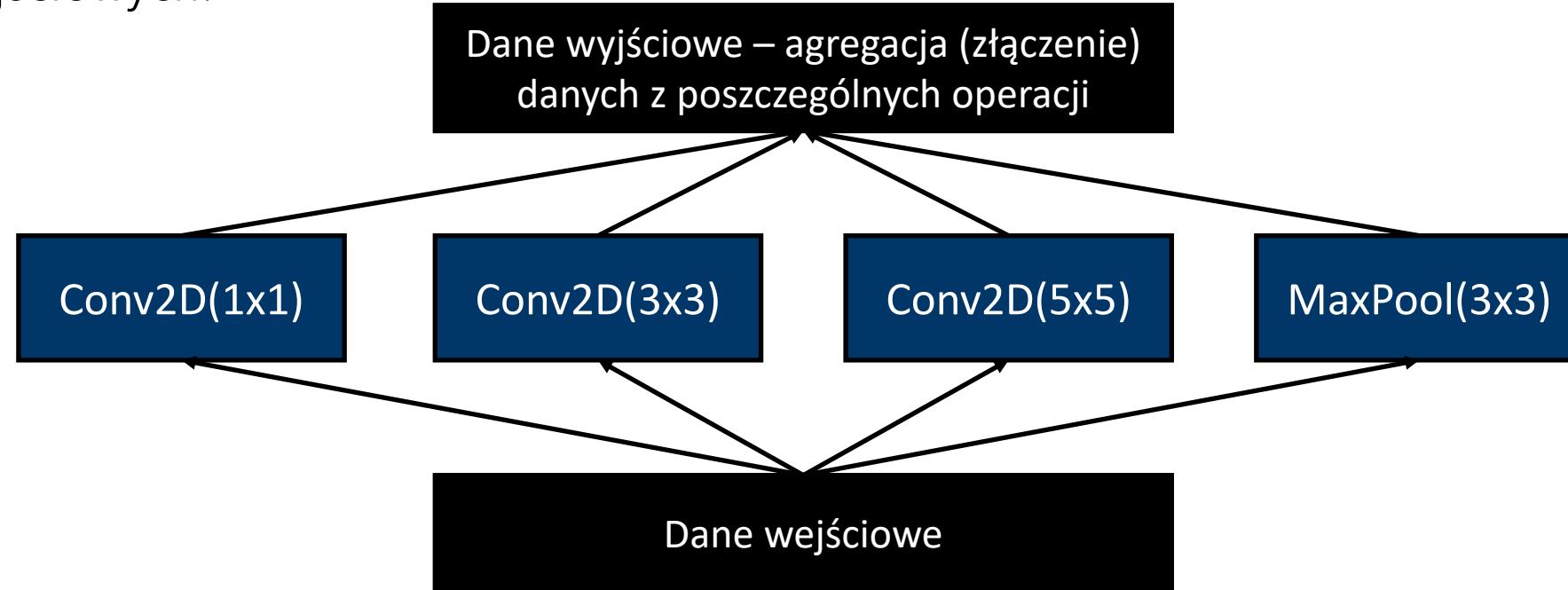
Kolejne warstwy splotowe (i np. warstwy redukcyjne) pozwalały poszerzyć pole recepcyjne sieci splotowej. Dzięki temu możliwe było uzyskanie cech istotnych np. dla obiektu widocznego na całym obszarze obrazu (np. duży pies wypełniający kadr).

Niemniej w tym samym zbiorze lub obrazie mogą występować obiekty danej klasy w różnych skalach. Zastosowanie redukcji skali obrazu (np. na wejściu lub pooling) oraz ustalenie filtrów o określonej, pojedynczej skali może ograniczać uzyskanie rozróżniającego zbioru cech.



Architektury modeli CNN - moduł incepcji

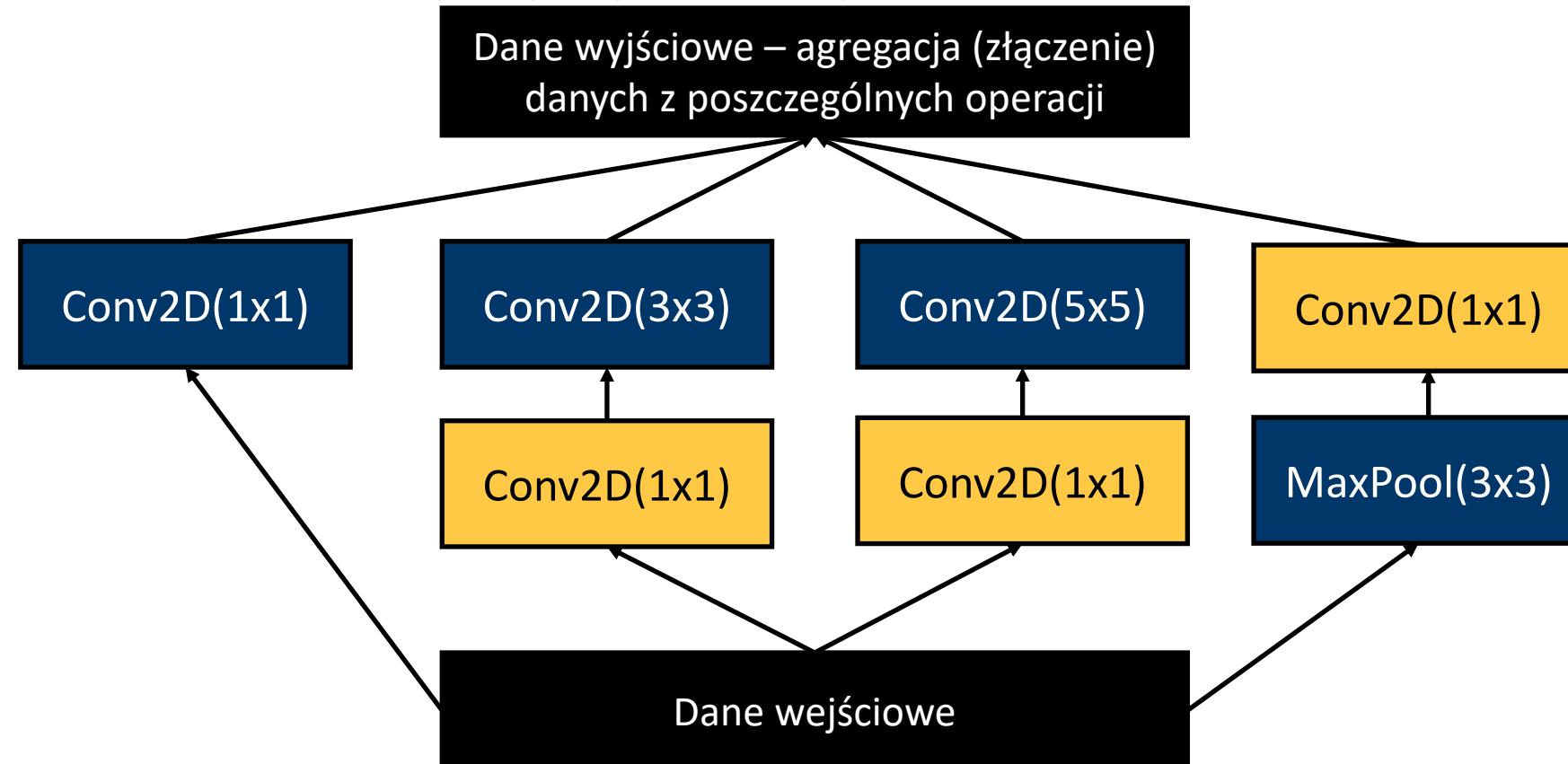
W [1] autorzy rozwinęli koncepcję wykorzystania filtrów o RÓŻNYCH skalach dla tych samych danych wejściowych:



Taki wyróżniony moduł nazwano modułem incepcji (ang. inception module). Został zaimplementowany w architekturze nazywanej GoogLeNet [1].

Architektury modeli CNN - moduł incepcji

Ze względu na dużą liczbę parametrów bazowego modułu incepcji w tej samej pracy [1] autorzy zaprezentowali wersję z redukcją wymiarowości modelu. Uzyskano to poprzez wprowadzenie operacji splotu z maską 1×1 (uzyskujemy 2 warstwy):



[1] Szegedy, C., Liu, W., et al., A. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014, <https://arxiv.org/abs/1409.4842>

Architektury modeli CNN - moduł incepcji

Znaczenie splotu z maską 1×1 i powiązane znaczenie dotyczące redukcji liczby parametrów w modelu sieci splotowej przedstawiliśmy we wcześniejszym materiale.

Moduł incepcji poszerza architekturę warstwy sieci neuronowej i ze względu na swoją budowę potencjalnie zwiększa zdolności reprezentacji danych przez sieć neuronową. Jednocześnie umożliwia redukcję liczby parametrów w modelu (np. ok. 5 mln parametrów w GoogLeNet vs. ok. 60 mln. parametrów w AlexNet).

Architektura **GoogLeNet** (zwana czasami Inception v1) składa się z 22 warstw podlegających uczeniu (bez warstw redukcji danych, itp.), złożonych z: **Conv2D(7x7)**, **Conv2D(1x1)**, **Conv2D(3x3)**, **9*Inception**, **Dense+softmax**.

Warto podkreślić, że architektura modelu w procesie uczenia jest trochę inna niż w procesie inferencji. Otóż w procesie uczenia dodano do GoogLeNet 2 dodatkowe klasyfikatory (ang. Auxiliary Classifier - dodatkowe „wyjścia” połączone do środkowych warstw modelu). Klasyfikatory te miały redukować problem zanikającego gradientu. Szczegóły w publikacji.

Architektury modeli CNN – moduł incepcji

Moduł incepcji oraz sieć GoogLeNet rozszerzono w kolejnych propozycjach przedstawiając architektury:

- Inception v.2/v.3: <https://arxiv.org/pdf/1512.00567v3.pdf> (**Top-5 Error: 6,3%-5,6%**)
- Inception v.4 / Inception ResNet: <https://arxiv.org/pdf/1602.07261.pdf> (**Top-5 Error: 5,5%-4,9% mniej niż 5% dla Inception ResNet v2, czyli lepiej niż człowiek**)

Nowe wersje wprowadzają m.in. rozkład (ang. factorization) operacji splotu z dużymi maskami na zestaw operacji z mniejszymi maskami, np. Conv2D(k, k) -> Conv2D(1, k) i Conv2D(k, 1).

Ponadto wprowadzono szereg innych modyfikacji, niektóre z nich omówimy w dalszej części materiału.

Architektury modeli CNN – moduł incepcji

Inception v2 (v3)
Wejście: obrazy RGB (299x299x3)
Conv2D($k=(3 \times 3)$, $f=32$, stride=2, no_padding) Conv2D($k=(3 \times 3)$, $f=32$, stride=1, no_padding) Conv2D($k=(3 \times 3)$, $f=64$, stride=1)
MAX POOLING (pool=(3x3), stride=(2x2))
Conv2D($k=(3 \times 3)$, $f=80$, stride=1) Conv2D($k=(3 \times 3)$, $f=192$, stride=2) Conv2D($k=(3 \times 3)$, $f=288$, stride=1)
3 x Inception v.2a (output=(17x17x768)) 5 x Inception v.2b (output=(8x8x1280)) 2 x Inception v.2c (output=(8x8x2048))
MAX POOLING (pool=(8x8))
Dense(2048)+Dense(1000)+softmax

Architektury modeli CNN - moduł incepcji

W artykule dotyczącym modelu incepcji w wersji 2 (i 3) autorzy [1] zaproponowali operację regularyzacji etykiet klas – ang. label smoothing (regularization).

W technice tej wartość funkcji kosztu bazująca na entropii wzajemnej obliczana jest nie na podstawie oryginalnych wartości etykiet reprezentowanych w kodowaniu typu one-hot, ale na ich „rozmytych” wersjach. Pozwala to również zwiększyć zdolność generalizacji dla danego modelu redukując zbyt dużą pewność (ang. overconfidence) jaką wynik ze sztywno przypisanych wartości etykiet (szczególnie przy ich reprezentacji typu one-hot i stosowaniu entropii jako miary kosztu). Czyli zamiast:

$$L = H(q, p) = - \sum_{k=1}^K q(k) \log(p(k))$$

stosuje się:

$$L = H(q', p) = - \sum_{k=1}^K q'^{(k)} \log(p(k)) = (1 - \epsilon) H(q, p) + \epsilon H(u, p)$$

gdzie: $q(k)$ - „prawdopodobieństwo” wynikające ze zbioru uczącego (ang. ground truth), $p(k)$ - „prawdopodobieństwo” określone przez model (softmax), ϵ - współczynnik, waga „rozmycia”.

Architektury modeli CNN - moduł incepcji

Przypomnijmy, że funkcja kosztu

$$L = H(q, p) = - \sum_{k=1}^K q(k) \log(p(k))$$

jest bardzo "wygodna" w metodzie gradientowej, ponieważ:

$$\frac{\partial L}{\partial z_k} = p(k) - q(k)$$

Przy kodowaniu one-hot $q(k) = \delta_{k,y}$, czyli $\delta_{k,y} = 1$ dla $k=y$ (dla danej klasy) inaczej 0.

W wersji:

$$L = H(q', p) = - \sum_{k=1}^K q'^{(k)} \log(p(k)) = (1 - \epsilon) H(q, p) + \epsilon H(u, p)$$

$$q'^{(k)} = (1 - \epsilon) \delta_{k,y} + \epsilon u(k)$$

gdzie: $u(k)$ jest rozkładem prawdopodobieństwa dla etykiet klas, niezależnym od x (przykładu).
Jeśli przyjmiemy rozkład równomierny $u(k)=1/K$, to:

$$q'^{(k)} = (1 - \epsilon) \delta_{k,y} + \epsilon / K$$

Architektury modeli CNN - moduł incepji

Rozpatrzmy prosty przykład. Niech K=3 (3 klasy).

Wówczas na wyjściu modelu z funkcją softmax uzyskamy wektor z wartościami „prawdopodobieństw”, $\mathbf{p} = [p_1, p_2, p_3]$. Etykiety referencyjne będą stosownie zakodowane (one-hot): $\mathbf{q}=[q_1, q_2, q_3]$, gdzie tylko jedna wartość będzie równa 1, pozostałe 0, np. [1, 0, 0].

Wartość „prawdopodobieństw” obliczamy na podstawie funkcji softmax:

$$\sigma(z)_y = \frac{e^{z_y}}{\sum_{k=1}^K e^{z_k}}$$

Ponieważ, obliczenia numeryczne wykonywane są z określona precyzją (np.

np.`set_printoptions(precision=5)`) oraz wprowadzone są regularizacje w implementacji funkcji (np. przeciwdziałanie dzieleniu przez 0) to biorąc to pod uwagę dla danych:

Logit: [20. 0. 0.] -> `scipy.special.softmax`:

Softmax: [1.00000e+00 2.06115e-09 2.06115e-09]

oraz parametrów: $\epsilon = 0.1$ i $K=3$ uzyskamy $((1 - 0.1)\delta_{k,y} + 0.1/3)$:

Softmax with regularization: [0.93333 0.03333 0.03333]

Logit after regularization: [2.63906 -3.3673 -3.3673]

Architektury modeli CNN - moduł incepcji

Wprowadzenie regularyzacji o obrębie etykiet klas pozwoliło autorom modelu Inception v2 zredukować błąd Top-5% z 6,3% do 6,1%. Korzyści płynące z tej techniki spowodowały, że jest ona implementowana w popularnych pakietach jak TF:

```
tf.keras.losses.BinaryCrossentropy(  
    from_logits=False, label_smoothing=0.0, axis=-1,  
    reduction=losses_utils.ReductionV2.AUTO, name='binary_crossentropy'  
)  
tf.keras.losses.CategoricalCrossentropy(  
    from_logits=False, label_smoothing=0.0, axis=-1,  
    reduction=losses_utils.ReductionV2.AUTO,  
    name='categorical_crossentropy'  
)
```

Ciekawą analizę dotyczącą znaczenia tej regularyzacji można znaleźć w [1].

Architektury modeli CNN – moduł incepcji

Rozwój nowych wersji modelu incepcji brał pod uwagę nie tylko zwiększenie wyników jakościowych (np. klasyfikacji) przy redukcji liczby obliczeń, ale szereg innych ważnych aspektów.

Wśród nich warto podkreślić starania związane z redukcją efektu gwałtownej redukcji danych przestrzennych (ang. representational bottleneck). Celem projektowym dla opracowania nowych architektur była stopniowa redukcja reprezentacji przestrzennej.

Innym celem projektowym było zbalansowania szerokości modelu (np. liczby węzłów w warstwie) względem głębokości modelu (liczby warstw).

Jedną z wersji architektury Inception jest Inception ResNet, która została opracowana po sukcesie architektur bazujących na module Residual Network, które przedstawimy na kolejnych slajdach (po prezentacji notatnikach związanego z modułem incepcji).

Przykłady praktyczne (notatnik interaktywny DL_06_02_CNN_Models_Inception.ipynb)

Moduł incepcji w środowisku Google Colaboratory/ Jupyter Notebook.

https://colab.research.google.com/drive/1V_kZiKpSiliGOS8m0bMSP8KIQx2pngfT?usp=sharing

Plan wykładu:

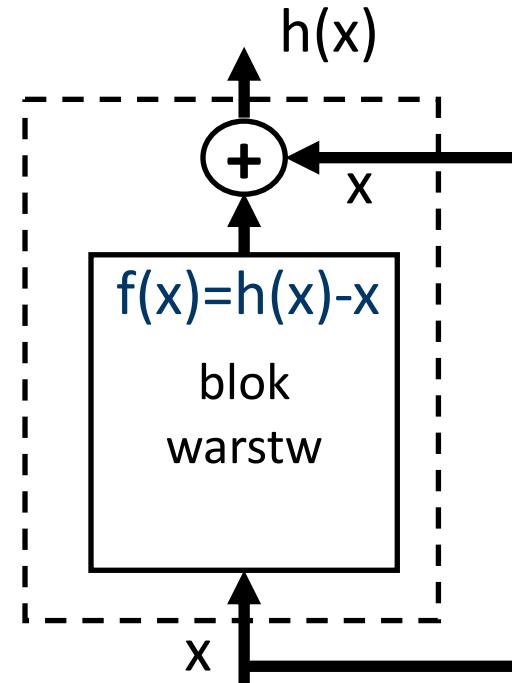
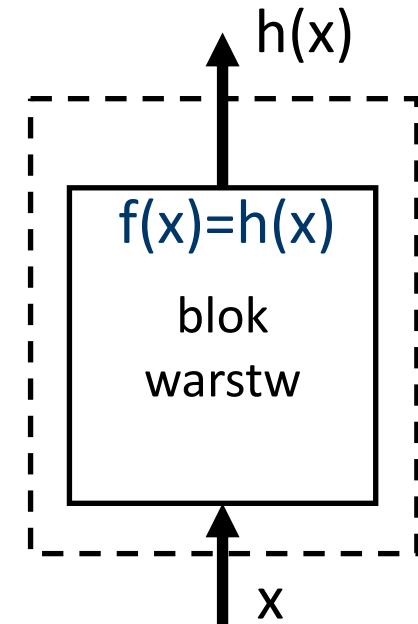
- VGG
- Inception
- **ResNet**
- Rozwój modeli CNN

Architektury modeli CNN - ResNet

W ramach architektury modelu VGG stosowaliśmy prostą sekwencję kolejnych K warstw splotowych. W module incepcji pojawiło się swoiste „rozwidlenie” w grafie reprezentującym operację na danych wejściowych. Dla tych samych danych źródłowych stosowaliśmy 4 gałęzie związane z operacją Conv2D(1x1), Conv2D(3x3), Conv2D(5x5), Pool(3x3).

W pracy [<https://arxiv.org/abs/1512.03385>] autorzy zaproponowali architekturę bazującą na sekwencji warstw jak w VGG, ale z uwzględnieniem dodatkowej gałęzi wiążącej wejście bloku warstw z wyjściem tego bloku. Blok taki nazwiemy blokiem rezydualnym, a sieć Residual Network.

Tradycyjny blok
SSN: sieć
aproksymuje
funkcję $f(x)$



Blok rezydualny:
sieć aproksymuje
funkcję $f(x)=h(x)-x$

Inaczej $h(x)=f(x)+x$

Architektury modeli CNN - ResNet

Sieć złożona z bloków rezydualnych (ang. Residual Network - ResNet) może zawierać:

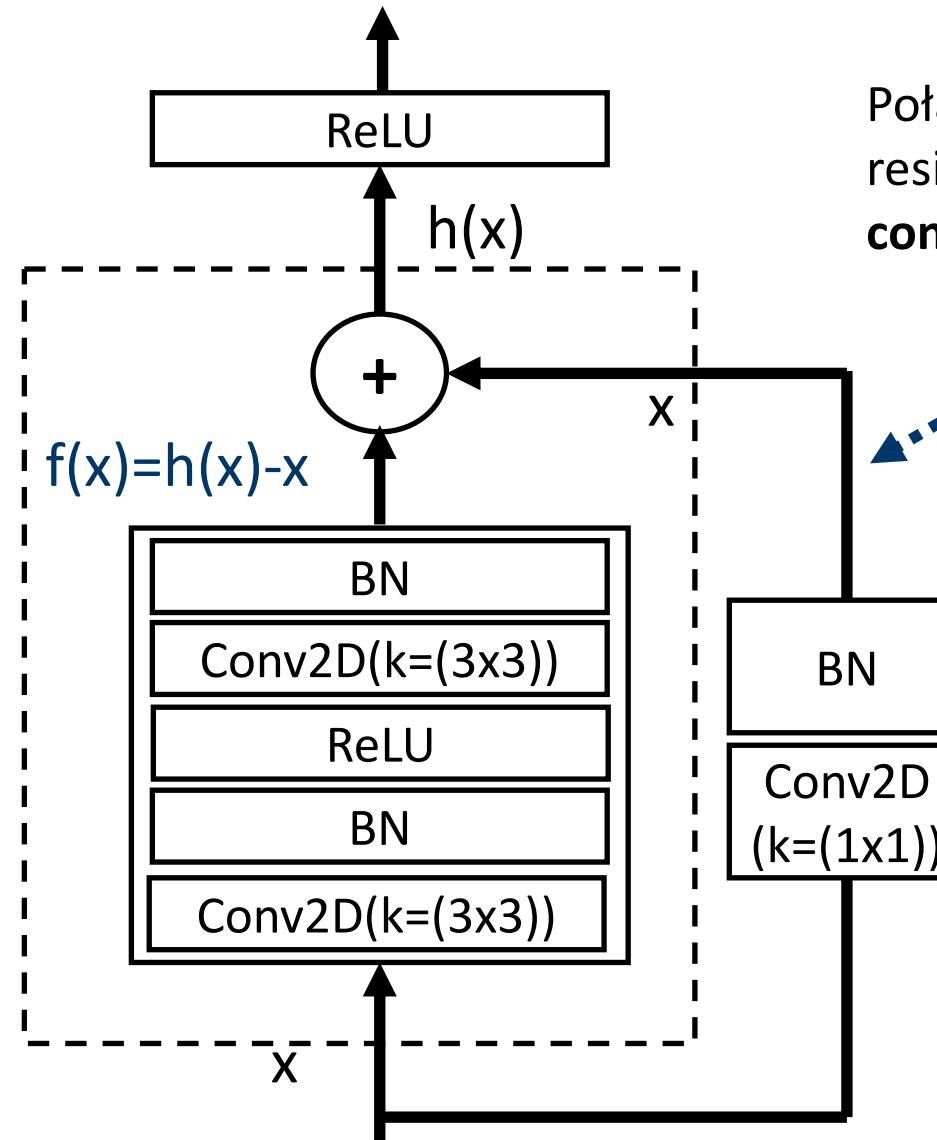
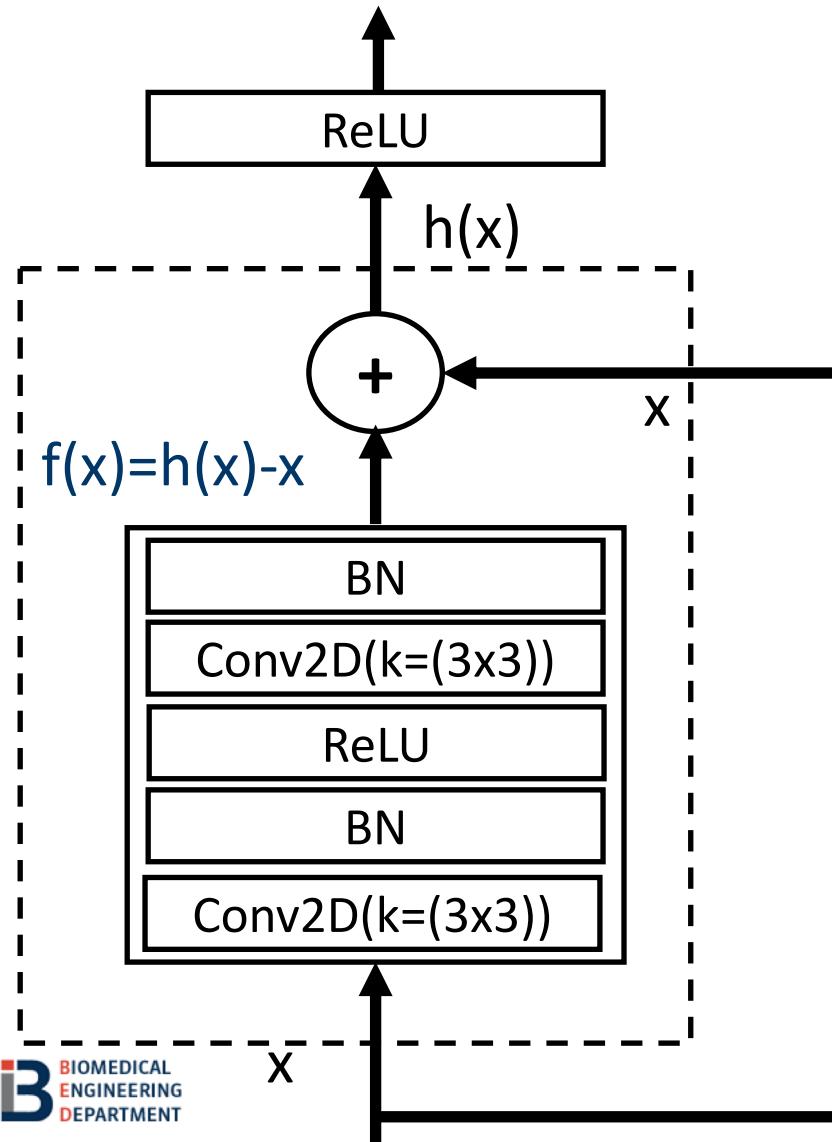
- różną liczbę bloków rezydualnych,
- różną liczbę warstw SNN w każdym z bloków rezydualnych,
- itp.

W tych względów zaproponowano różne konfiguracje architektur, które definiują zestawy K warstw zawierających bloki rezydualne, np. ResNet-18 (18 warstw parametrów, w tym 8 bloków rezydualnych, każdy po 2 warstwy splotowe - dodatkowo warstwa splotowa na wejściu i warstwa gęsta na wyjściu - łącznie 18 warstw). Analogicznie dla ResNet-34, ResNet-50, ResNet-101, ResNet-152.

Przykładowy blok rezydualny zawiera sekwencję warstw: Conv2D(3x3), BN, Relu, Conv2D(3x3), BN. Po operacji dodania wejścia wykonywana jest operacja nieliniowa Relu.

Zilustrowano to na kolejnym slajdzie.

Architektury modeli CNN - ResNet



Połączenie rezydualne (ang. residual connection lub **skip connection**)

Dodatkowa operacja splotu z taką liczbą filtrów, aby kontrolować zgodność liczby kanałów i rozmiaru danych obu torów.

Architektury modeli CNN - ResNet - przykładowe konfiguracje

ResNet-18
Conv2D($k=(7 \times 7)$, $f=64$, stride=2)
MAX POOLING (pool= (3×3) , stride= (2×2))
ResBlock(2*Conv2D($k=(3 \times 3)$, $f=64$))
ResBlock(2*Conv2D($k=(3 \times 3)$, $f=64$))
ResBlock(2*Conv2D($k=(3 \times 3)$, $f=128$))
ResBlock(2*Conv2D($k=(3 \times 3)$, $f=128$))
ResBlock(2*Conv2D($k=(3 \times 3)$, $f=256$))
ResBlock(2*Conv2D($k=(3 \times 3)$, $f=256$))
ResBlock(2*Conv2D($k=(3 \times 3)$, $f=512$))
ResBlock(2*Conv2D($k=(3 \times 3)$, $f=512$))
AVG POOLING ()
Dense(1000)+softmax

ResNet-50
Conv2D($k=(7 \times 7)$, $f=64$, stride=2)
MAX POOLING (pool= (3×3) , stride= (2×2))
3* ResBlock(Conv2D($k=(1 \times 1)$, $f=64$), Conv2D($k=(3 \times 3)$, $f=64$), Conv2D($k=(1 \times 1)$, $f=256$))
4* ResBlock(Conv2D($k=(1 \times 1)$, $f=128$), Conv2D($k=(3 \times 3)$, $f=128$), Conv2D($k=(1 \times 1)$, $f=512$))
6* ResBlock(Conv2D($k=(1 \times 1)$, $f=256$), Conv2D($k=(3 \times 3)$, $f=256$), Conv2D($k=(1 \times 1)$, $f=1024$))
3* ResBlock(Conv2D($k=(1 \times 1)$, $f=512$), Conv2D($k=(3 \times 3)$, $f=512$), Conv2D($k=(1 \times 1)$, $f=2048$))
AVG POOLING ()
Dense(1000)+softmax

Architektury modeli CNN - ResNet

Sieć typu ResNet z połączeniami rezydualnymi pozwala ograniczyć problem zanikającego gradientu pojawiającego się dla sieci głębokich. Stosując bowiem regułę łańcuchową w czasie propagacji wstecznej gradient może stawać się coraz mniejszy i zniknąć w pobliżu pierwszych warstw (warstw wejścia). Wówczas aktualizacji parametrów nie będzie właściwi realizowana ($W^{[s+1]} = W^{[s]} + 0$)

W sieciach typu ResNet w czasie projekcji wstecznej (ang. backpropagation) gradienty są propagowane również przez połączenia rezydualne, co zmniejsza szansę ich zaniku.

Model ResNet-152 (2015) w teście klasyfikacji (rozpoznawania obrazów) dla bazy ImageNet osiągnął wynik Top-5 Error poniżej 4,5%, co stanowiło znaczące osiągnięcie w stosunku do wyniku modelu AlexNet uzyskanego zaledwie (około) 3 lat wcześniej.

Obecnie znane są różne warianty i kombinacje modeli bazujących na VGG, Inception, ResNet. Szereg nowych modeli pozwala osiągnąć błąd Top-5 rzędu 1%, natomiast błąd Top-1 na poziomie <10%.

Więcej informacji na:

Przykłady praktyczne (notatnik interaktywny DL_06_03_CNN_Models_ResNet.ipynb)

ResNet w środowisku Google Colaboratory/ Jupyter Notebook.

<https://colab.research.google.com/drive/1AeFUZF8jXUjablFcd99dxNIJgc9Y3IXR?usp=sharing>

Plan wykładu:

- VGG
- Inception
- ResNet
- **Rozwój modeli CNN**

W literaturze oraz w ramach bibliotek oprogramowania związanego z uczeniem głębokim znajdziemy bardzo dużo propozycji modeli splotowych.

Przykładowo zbiór modeli (ang. model Zoo) dostępny w środowisku TF:

Xception, VGG, ResNet, Inception, MobileNet, DenseNet, EfficientNet i inne.

Więcej na stronie: <https://keras.io/api/applications/>

Bogaty zbiór modeli znajdziemy również na stronach: <https://modelzoo.co>
<https://catalog.ngc.nvidia.com/models>, itp.

Rozwijane są również otwarte formaty związane z reprezentacją modelu uczenia maszynowego, istotne przy wymianie modeli, np.: <https://onnx.ai>

Zapisane modele można wykorzystywać do inferencji lub rozbudowywać je korzystając z określonych fragmentów, np. w ramach uczenia z przeniesieniem.

Dziękuję

Jacek Rumiński



Fundusze
Europejskie
Polska Cyfrowa

Rzeczpospolita
Polska



KANCELARIA PREZESA RADY MINISTRÓW
THE CHANCELLERY OF THE PRIME MINISTER

Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Uczenie głębokie

Wykład 7b: Głębokie autokodery

Jacek Rumiński
Katedra Inżynierii Biomedycznej, Wydział ETI



Fundusze
Europejskie
Polska Cyfrowa



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

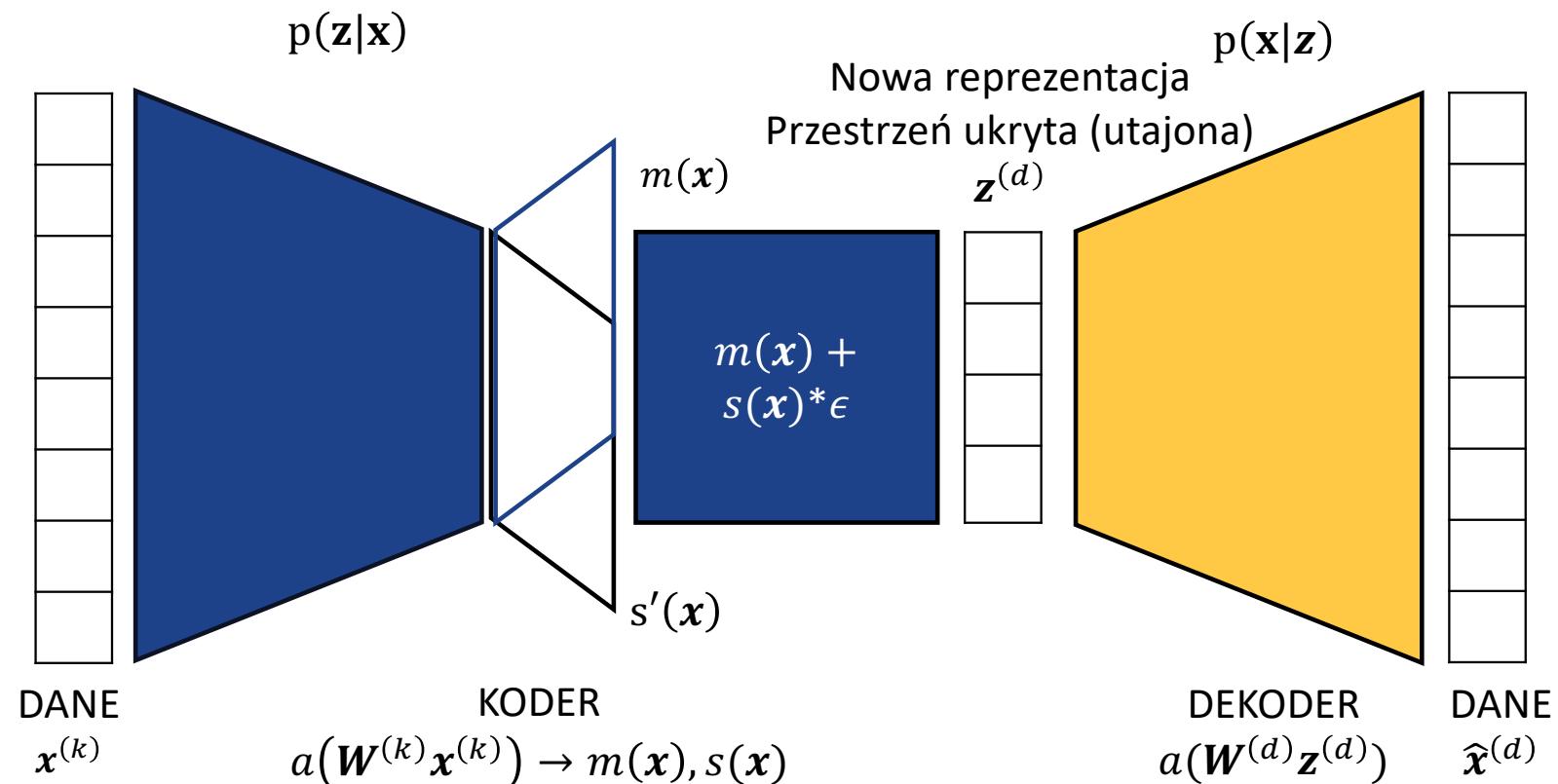
Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu:

- **Autokodery wariacyjne - przypomnienie**
- **Głębokie autokodery wariacyjne**

Autokodery

Przedstawmy prostą architekturę autokodera wariacyjnego:



Autokodery

W przypadku autokoderów wariacyjnych zakładamy, że:

$p(\mathbf{z}) = N(0, \mathbf{I})$ - standardowy rozkład normalny (wartość średnia równa 0, odchylenie standardowe opisane macierzą jednostkową \mathbf{I} - wartości jeden dla głównej przekątnej - wariancje)

$p(\mathbf{x}|\mathbf{z}) = N(f(\mathbf{z}), c\mathbf{I})$ - rozkład normalny, tak, że jeśli $E(\mathbf{x}|\mathbf{z}) = f(\mathbf{z}) = \mathbf{z}$ to $p(\mathbf{z}|\mathbf{x})$ też powinien być rozkładem Gaussa (ale w praktyce tak nie jest i stosujemy techniki aproksymacyjne).

Stosować będziemy inferencję wariacyjną, w której badamy pewną parametryzowaną gamę rozkładów prawdopodobieństwa (np. Gaussa - znane parametry to wartość średnia i macierz kowariancji).

Dla badanej gamy szukamy takich parametrów, dla których uzyskamy najlepszą aproksymację docelowego rozkładu prawdopodobieństwa.

Autokodery

Pragniemy zatem porównywać dwa rozkłady prawdopodobieństw: parametryzowany (dopasowywany) i docelowy.

Wybierzemy takie parametry rozkładu dopasowywanego, dla którego uzyskamy minimum funkcji kosztu.

Wcześniej wprowadziliśmy już powiązaną funkcję kosztu: entropia krzyżowa czy dywergencja Kullbacka-Leiblera: $D_{KL}(q||p)$.

Będziemy chcieli aproksymować $p(\mathbf{z}|\mathbf{x})$ poprzez parametryczną postać rozkładu, np. Gaussa:

$$q_{\mathbf{x}}(\mathbf{z}) = N(m(\mathbf{x}), s(\mathbf{x}))$$

Autokodery

Cel: optymalizacja funkcji $m(\mathbf{x})$ i $s(\mathbf{x})$ (dobór parametrów) tak, aby uzyskać minimalną wartość $D_{KL}(q||p)$, gdzie p to $p(\mathbf{z}|\mathbf{x})$, a q podany wyżej przykład modelu.

$$(\hat{m}, \hat{s}) = \operatorname{argmin}_{(m,s)} D_{KL}(q_{\mathbf{x}}(\mathbf{z}), p(\mathbf{z}|\mathbf{x})) = \operatorname{argmax}_{(m,s)} \left(E_{\mathbf{z} \sim q_{\mathbf{x}}} (\log(p(\mathbf{x}|\mathbf{z}))) - D_{KL}(q_{\mathbf{x}}(\mathbf{z}), p(\mathbf{z})) \right)$$

Ponieważ założyliśmy, że $p(\mathbf{x}|\mathbf{z}) = N(f(\mathbf{z}), c\mathbf{I})$ (czyli Gauss) dlatego upraszczając $\log(\exp(.))$:

$$(\hat{m}, \hat{s}) = \operatorname{argmax}_{(m,s)} \left(E_{\mathbf{z} \sim q_{\mathbf{x}}} \left(-\frac{\|\mathbf{x} - f(\mathbf{z})\|^2}{2c} \right) - D_{KL}(q_{\mathbf{x}}(\mathbf{z}), p(\mathbf{z})) \right)$$

Zauważmy, że w przedstawionym kryterium mamy dwie części:

- maksymalizację logarytmu funkcji wiarygodności (minimalizacja błędu rekonstrukcji $(\mathbf{x}-f(\mathbf{z}))$)
- wymaganie "bliskości" funkcji $q_{\mathbf{x}}(\mathbf{z})$ względem prawdopodobieństwa a priori $p(\mathbf{z})$ (dywergencja KL).

Autokodery

Funkcje $m(\mathbf{x})$ oraz $s(\mathbf{x})$ będziemy chcieli zrealizować poprzez bloki sieci neuronowych. Założymy, że pierwsza warstwa ukryta opisywana jest funkcją $h_{11}(x)$. Założymy, że wyjście tej warstwy jest wejściem dwóch kolejnych bloków $h_{21}(x)$ i $h_{22}(x)$, wówczas:

$$\begin{aligned}m(x) &= h_{21}(h_{11}(x)) \\s(x) &= h_{22}(h_{11}(x))\end{aligned}$$

Po estymacji funkcji chcielibyśmy, aby wartość z była losowana z rozkładu prawdopodobieństwa $N(m(\mathbf{x}), s(\mathbf{x}))$.

Niemniej z punktu widzenia propagacji wstecznej taki węzeł probabilistyczny może być problemem. Dlatego stosuje się reparametryzację tak, że zamiast wersji probabilistycznej:

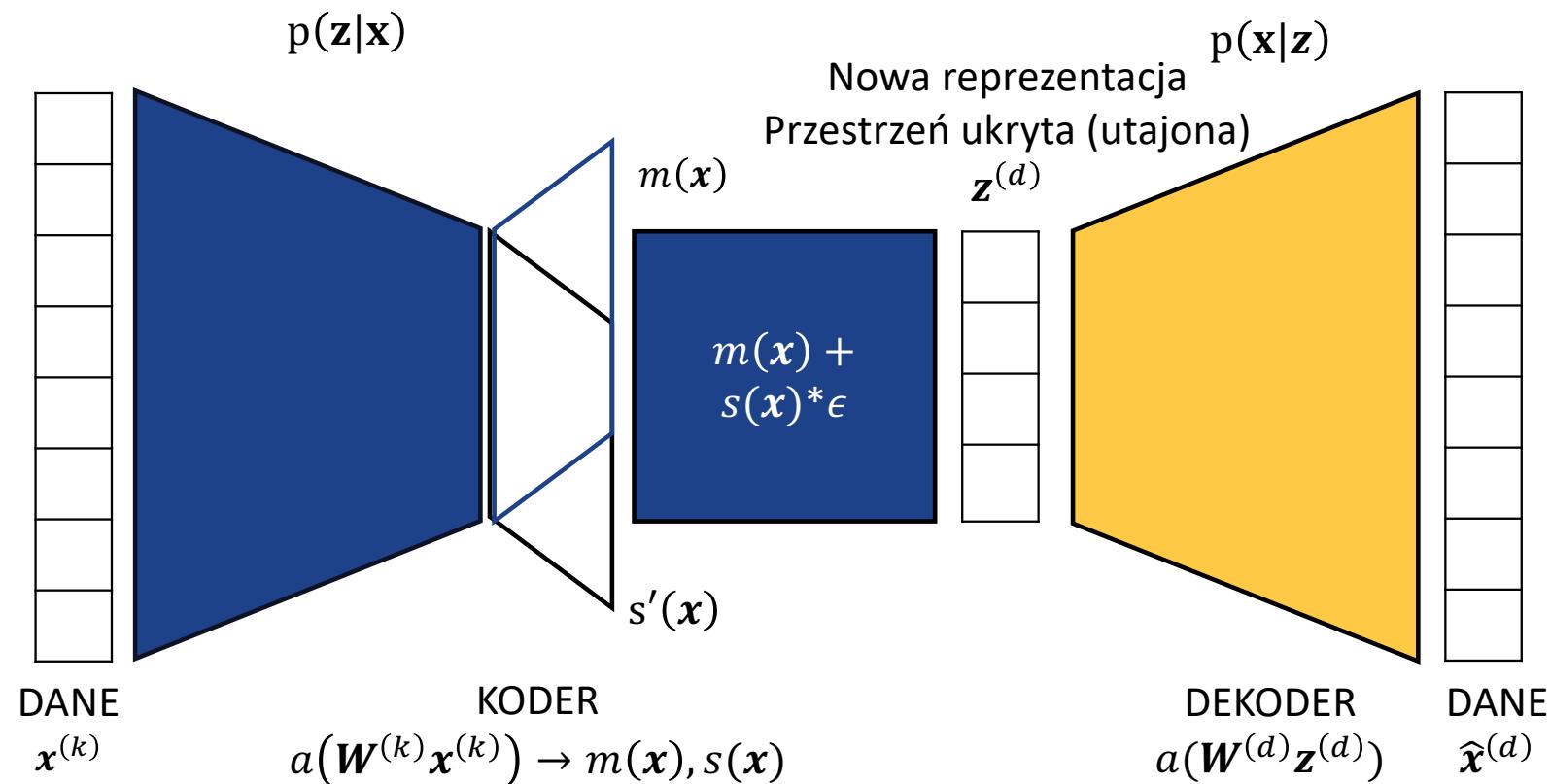
$$\mathbf{z} \sim N(m(\mathbf{x}), s(\mathbf{x}))$$

dojdziemy do postaci "deterministycznej":

$$\mathbf{z} = m(\mathbf{x}) + s(\mathbf{x}) * \epsilon$$

Autokodery

Przedstawmy prostą architekturę autokodera wariacyjnego:



Autokodery

Zaimplementujmy koder w środowisku Keras API:

```
# Encoder block (not Sequential)
x = kl.Input(shape=(data_shape,))
# h_11(x)
h = kl.Dense(hidden_units, activation='relu', name='hidden')(x)

# h_21(x)
z_mean = kl.Dense(latent_dim, name='z_mean')(h)
# h_22(x)
z_log_var = kl.Dense(latent_dim, name='z_log_var')(h)

# Perform sampling from distribution using m(x) and s'(x)
z = kl.Lambda(latent_reparam_sampling, output_shape=(latent_dim,), name='z_sampling')([z_mean,
z_log_var])

encoder = km.Model(inputs=x, outputs=z, name='vae_encoder')
```

Autokodery

Po zdefiniowaniu warstw dedykowanych uczeniu $m(x)$ i $s'(x)$ (logarytm z wariancji) zastosowaliśmy warstwę *Lambda* w celu wywołania funkcji realizującej reparametryzację:

```
# inspired by https://github.com/shashankdhar/VAE-MNIST/blob/master/VAE.py
def latent_reparam_sampling(args):
    # unpack the parameters from args, assumed to be m(x) z_mean and
    # s'(x) log of variance (instead of standard deviation) - z_log_var
    z_mean_layer, z_log_var_layer = args
    # get the shape of the batch of data
    batch_size = K.shape(z_mean_layer)[0]
    # get the data size (how many z_mean and z_log_var we have)
    data_size = K.int_shape(z_mean_layer)[1]
    # IMPORTANT - draw N(0, I) - p(z)
    epsilon = K.random_normal(shape=(batch_size, data_size))
    # Now return m(x)+s(x)*epsilon, first get std. dev. s'(x) from
    # z_log_var - > s(x) = exp(0.5*z_log_var)
    return z_mean_layer + K.exp(0.5 * z_log_var_layer) * epsilon
```

Autokodery

Zaimplementujmy dekoder w środowisku Keras API:

```
# Decoder block
decoder = km.Sequential([
    kl.Dense(hidden_units, input_dim=latent_dim, activation='relu', name='hidden'),
    kl.Dense(data_shape, activation='sigmoid', name='output')
],name='vae_decoder')
```

Zdefiniujmy na koniec funkcję kosztu uwzględniającą koszt rekonstrukcji oraz koszt warunkujący podobieństwo $q_x(\mathbf{z})$ do $p(z)$. Ponieważ, zgodnie z założeniami, obie funkcje są gaussowskie, dlatego dywergencję KL możemy przedstawić w postaci (J – wymiar przestrzeni ukrytej \mathbf{z}):

$$\begin{aligned} -D_{KL}(q_x(\mathbf{z}), p(\mathbf{z})) &= \int q_x(\mathbf{z})(\log(p(\mathbf{z})) - \log(q_x(\mathbf{z})))d\mathbf{z} \\ &= \int (N(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log(N(\mathbf{z}; 0, \mathbf{I})))d\mathbf{z} - \int (N(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log(N(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)))d\mathbf{z} = -\frac{J}{2}\log(2\pi) \\ &\quad - \frac{1}{2}\sum_{j=1}^J (\mu_j^2 + \sigma_j^2) + \frac{J}{2}\log(2\pi) + \frac{1}{2}\sum_{j=1}^J (1 + \log\sigma_j^2) = \frac{1}{2}\sum_{j=1}^J 1 + \log\sigma_j^2 - \mu_j^2 - \sigma_j^2 \end{aligned}$$

Autokodery

W wyprowadzonym wzorze:

$$D_{KL}(q_x(\mathbf{z}), p(\mathbf{z})) = -\frac{1}{2} \sum_{j=1}^J 1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2$$

uzyskaliśmy logarytm wariancji, wartość średnią i wariancję, dla każdego z wymiarów przestrzeni ukrytej \mathbf{z} . Pamiętamy, że wartości te wyznaczamy poprzez zastosowanie funkcji $m(\mathbf{x})$ i $s(\mathbf{x})$ realizowanych za pomocą sieci neuronowych.

Zaimplementujmy powyższą dywergencję w funkcji kosztu uwzględniającą koszt rekonstrukcji oraz koszt warunkujący podobieństwo $q_x(\mathbf{z})$ do $p(\mathbf{z})$:

```
loss_reconstruction = K.sum(K.binary_crossentropy(x, x_p), axis=-1)
```

```
loss_KL = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
```

```
loss_KL = -0.5 * K.sum(loss_KL, axis=-1)
```

```
vae_loss = K.mean(loss_reconstruction + loss_KL)
```

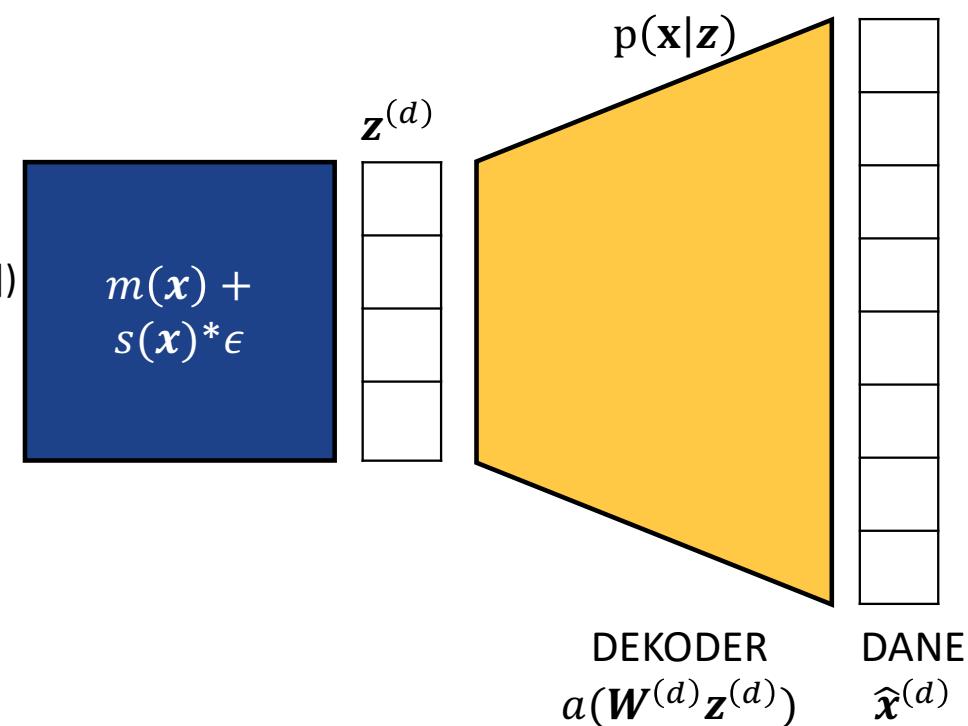
Autokodery

W wyniku uczenia uzyskamy model, który charakteryzuje się nową reprezentacją danych wyznaczoną przez parametry modeli.

Z tej nowej reprezentacji danych opisanych funkcjami gaussowskimi o wyznaczonych parametrach można wylosować dane, z których stosując jedynie dekoder można zrekonstruować nową postać danych:

```
# Wylosujmy z
latent_sample =
np.random.normal(0,1,size=[selected_gen, latent_dim])

# Stosując tylko dekoder wygeneruj nowe dane
input_decoded = decoder.predict(latent_sample)
```



Autokodery

Wyżej posłużyliśmy się trikiem reparametryzacji tak, aby można było zamiast wersji probabilistycznej:

$$\mathbf{z} \sim N(m(\mathbf{x}), s(\mathbf{x}))$$

użyć postaci "deterministycznej":

$$\mathbf{z} = m(\mathbf{x}) + s(\mathbf{x}) * \epsilon$$

Warto podkreślić, że w środowisku TensorFlow wprowadzono specjalne warstwy reprezentujące modele probabilistyczne, np. wielowymiarowy rozkład normalny: `MultivariateNormalTriL`, rozkład Bernoulliego `IndependentBernoulli`, itp.

Umożliwiają one m.in. zastosowanie modeli sekwencyjnych w definicji autokoderów. Zademonstrowaliśmy to w załączonym notatniku interaktywnym.

Literatura uzupełniająca: D.P. Kingma, M. Welling, Auto-Encoding Variational Bayes, 2014,
<https://arxiv.org/abs/1312.6114>

Autokodery głębokie - splotowe

Opracowanie warstw związanych z sieciami splotowymi sprawiło, że również zaczęto stosować tego typu warstwy w autokoderach. Przykładowy koder:

```
self.encoder = tf.keras.Sequential([
    layers.Conv2D(32, 3, activation=enc_act, padding=padding),
    layers.MaxPool2D(2),
    layers.Conv2D(64, 3, activation=enc_act, padding=padding),
    layers.MaxPool2D(2),
    layers.Conv2D(128, 3, activation=enc_act, padding=padding),
    layers.LayerNormalization(),
    # Spatial shape is (8,8) – we used MaxPool two times
    layers.Flatten(),
    layers.Dense(self.latent_space_dim, activation=enc_act),
])
```

Autokodery głębokie - splotowe

Przykładowy dekoder:

```
# We used MaxPool two times, so we need to upscale two times in the decoder
self.decoder = tf.keras.Sequential([
    layers.Dense(self.latent_space_dim, activation=dec_act),
    layers.Reshape((self.rows_columns, self.rows_columns, data_shape[3])),
    # BE CAREFUL - tune upscaling to latent space dim
    # Restore the resolution: upscale with interpolation and normal convolution
    layers.UpSampling2D((2, 2), interpolation="nearest"),
    layers.Conv2D(64, 3, activation=dec_act, padding=padding),
    layers.UpSampling2D((2,2), interpolation="nearest"),
    layers.Conv2D(32, 3, activation=dec_act, padding=padding),
    # OR - pptionally, Use Conv2DTranspose with stride=2 to upscale and pseudo-deconvolution
    # To do so, please replase 2 pairs of UpSampling2D and Conv2D layers with:
    # layers.Conv2DTranspose(64, 3, strides=(2, 2), activation=dec_act, padding=padding),
    # layers.Conv2DTranspose(32, 3, strides=(2, 2), activation=dec_act, padding=padding),

    layers.Conv2D(3, 1), # Generate 3 channels
    layers.Activation("sigmoid"), # Scale output to get values in range (0,1)
])
```

Przykłady praktyczne (notatnik interaktywny DL_07_02_Deep_Autoencoders.ipynb)

Autokodery splotowe w środowisku Google Colaboratory/ Jupyter Notebook.

Różne postaci normalizacji

W jednym z przykładów dla autokodera głębokiego wykorzystaliśmy warstwę normalizacji typu LayerNormalization.

Przypomnijmy:

BatchNormalization – oblicza statystykę (mean, var) w ramach porcji danych:

Przykład: Dane 1D: C=5 cech, N=2 przykłady w porcji -> shape (2,5), np.:

```
[[0., 0., 1., 1., 2.],  
 [2., 3., 3., 4., 4.]]
```

Obliczenia średnie po N:

$$\text{mean}[0] = 1/2 * (0+2), \dots, \text{mean}[4] = 1/2 * (2+4) \text{ etc.}$$

Dla danych 2D statystyka liczona dla po (N, H, W).

Różne postaci normalizacji

W jednym z przykładów dla autokodera głębokiego wykorzystaliśmy warstwę normalizacji typu LayerNormalization.

LayerNormalization – oblicza statystykę (mean, var) w ramach zbioru cech:

Przykład: Dane 1D: C=5 cech, N=2 przykłady w porcji -> shape (2,5), np.:

```
[[0., 0., 1., 1., 2.],  
 [2., 3., 3., 4., 4.]]
```

Obliczenia średnie po C:

$$\text{mean}[0] = 1/5 * (0+0+1+1+2), \text{ mean}[1] = 1/5 * (2+3+3+4+4) \text{ etc.}$$

Dla danych 2D statystyka liczona dla po (H, W, C).

Różne postaci normalizacji

Uogólnienie LayerNormalization to GroupNormalization:

$G = 1$ -> instance normalization

$G = C$ -> Layer Normalization.

Przykłady praktyczne (notatnik interaktywny normalizations.ipynb)

Normalizacja w środowisku Jupyter Notebook.

Głębokie autokodery wariacyjne

Analogicznie jak w przypadku płytowych autokoderów możemy wprowadzić dodatkowe wymagania w zakresie funkcji rozkładu prawdopodobieństwa dla przestrzeni ukrytej uzyskiwanej w autokoderach splotowych (ogólnie głębokich).

Głębokie autokodery wariacyjne będą miały jednak tendencję do znacznie większych problemów w zakresie zbieżności procesu uczenia (np. zajwisko „posterior collapse”.).

Dlatego istotne jest szczególne przygotowanie złożonej funkcji kosztu.

Głębokie autokodery wariacyjne

Cel: optymalizacja funkcji $m(\mathbf{x})$ i $s(\mathbf{x})$ (dobór parametrów) tak, aby uzyskać minimalną wartość $D_{KL}(q||p)$, gdzie p to $p(\mathbf{z}|\mathbf{x})$, a q podany wyżej przykład modelu.

$$\begin{aligned} (\hat{m}, \hat{s}) = & \operatorname{argmin}_{(m,s)} D_{KL}(q_{\mathbf{x}}(\mathbf{z}), p(\mathbf{z}|\mathbf{x})) = \\ & \operatorname{argmax}_{(m,s)} \left(E_{\mathbf{z} \sim q_{\mathbf{x}}} (\log(p(\mathbf{x}|\mathbf{z}))) - D_{KL}(q_{\mathbf{x}}(\mathbf{z}), p(\mathbf{z})) \right) \end{aligned}$$

Zauważmy, że dywergencja $D_{KL}(q_{\mathbf{x}}(\mathbf{z}), p(\mathbf{z}))$ związana jest z koderem, a funkcja wiarygodności z dekoderem. Może się zdarzyć, że określone wartości parametrów modelu mogą prowadzić do sytuacji, że koder dokładnie oddaje p(z), czyli dywergencja będzie 0.

Dominuje wówczas dekoder, a koszt związany z koderem jest szybko równy zero i w czasie treningu właściwie nie ma znaczenia. Zjawisko te nazywane jest „posterior collapse”. Przykładowe rozwiążane tego problemu to np. dodanie parametru do $D_{KL}(q_{\mathbf{x}}(\mathbf{z}), p(\mathbf{z}))$, który staruje od wartości zera a wraz z kolejnymi epokami jego wartość rośnie w kierunku 1. Więcej w [1].

Głębokie autokodery wariacyjne

Przypomnijmy trochę teorii. Funkcją wiarygodności zmiennej θ nazywaliśmy:

$$L(\theta|x) = P_\theta(X=x)$$

Estymatorem największej wiarygodności parametru θ nazywaliśmy:

$$\operatorname{argmax}_\theta(L(\theta|x))$$

Stosując wygodną operację logarytmowania uzyskamy funkcję log-wiarygodności:

$$l(\theta|x) = \ln(P_\theta(X=x)).$$

Dla naszego autokodera mamy dane \mathbf{x} (realizacja zmiennej losowej X). Dodatkowo zakładamy istnieje zmiennej losowej Z (nieobserwowanej -> ukrytej -> latent) przyjmujących wartości \mathbf{z} . Zakładamy również łączny rozkład wiążący te zmienne losowe $p(X, Z; \theta)$, gdzie θ to parametry rozkładu.

Głębokie autokodery wariacyjne

Parametry θ możemy estymować stosując estymator największej log-wiarygodności

$$\hat{\theta} = \operatorname{argmax}_{\theta}(l(\theta|\mathbf{x}))$$

Wiemy, że funkcja log-wiarygodności to:

$$l(\theta|\mathbf{x}) = \ln(p_{\theta}(\mathbf{x})).$$

gdzie \mathbf{x} to dane wygenerowane z dekodera (autokodera) $p_{\theta}(\mathbf{x}|\mathbf{z})$, które w procesie uczenia powinny być równoważne wejściu (autokoder!).

Jednocześnie możemy zapisać:

$$\ln(p_{\theta}(\mathbf{x})) = \ln \int p(\mathbf{x}, \mathbf{z}; \theta) d\mathbf{z} \text{ lub } = \ln \int p_{\theta}(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z}$$

Głębokie autokodery wariacyjne

Patrząc na przestrzeń ukrytą w kierunku wejścia autokodera „prawdziwy” rozkład aposteriori opisujący uzyskanie wartości z w przestrzeni ukrytej pod warunkiem x to

$$p_{\theta}(\mathbf{z}|\mathbf{x}) \propto p_{\theta}(\mathbf{x}|\mathbf{z}) p(\mathbf{z})$$

Pamiętamy jednocześnie, że „prawdziwy” rozkład aposteriori był aproksymowany w autokoderze poprzez rozkład wariacyjny $q_{\phi}(\mathbf{z}|\mathbf{x})$ (koder, inference model).

Szukamy takiej postaci funkcji $q_{\phi}(\mathbf{z}|\mathbf{x})$, aby maksymalizować jej podobieństwo do $p_{\theta}(\mathbf{z}|\mathbf{x})$ poprzez minimalizację funkcji kosztu (metoda wariacyjna).

Czyli:

$$\ln(p_{\theta}(\mathbf{x})) = \ln \int p_{\theta}(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z} = \ln E_{\mathbf{z} \sim q} \left[\frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} \right]$$

Głębokie autokodery wariacyjne

Przenosząc logarytm dla wartości oczekiwanej, uzyskujemy na podstawie nierówności Jensena:

$$\ln(p_\theta(\mathbf{x})) = \ln E_{\mathbf{z} \sim q} \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] \geq E_{\mathbf{z} \sim q} \left[\ln \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right]$$

Czynnik $\ln(p_\theta(\mathbf{x}))$ jest nazywany czasami jako „evidence”, ponieważ dla dobrze dobranego model rozkładu prawdopodobieństwa p, oraz dla dobrze dobranego zbioru parametrów θ uzyskujemy wysokie prawdopodobieństwo obserwacji \mathbf{x} .

Czynnik $E_{\mathbf{z} \sim q} \left[\ln \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right]$ jest granicą dolną (\geq) dla dowodu i nazywany jest w języku angielskim

ELBO: Evidence Lower BOund.

Głębokie autokodery wariacyjne

Zauważmy, że:

$$E_{z \sim q} \left[\ln \frac{p_\theta(x, z)}{q_\phi(z|x)} \right] = E_{z \sim q} [\ln(p_\theta(x|z))] - E_{z \sim q} \left[\ln \left(\frac{p(z)}{q_\phi(z|x)} \right) \right]$$

Czyli:

$$E_{z \sim q} \left[\ln \frac{p_\theta(x, z)}{q_\phi(z|x)} \right] = E_{z \sim q} [\ln(p_\theta(x|z))] - D_{KL} (q_\phi(z|x), p(z))$$

I dalej:

$$\ln(p_\theta(x)) \geq \mathcal{L}_{ELBO} = E_{z \sim q} [\ln(p_\theta(x|z))] - D_{KL} (q_\phi(z|x), p(z))$$

Funkcja kosztu:

$$\mathcal{L}_{ELBO} = \mathcal{L}_{REC} + \mathcal{L}_{REG} = E_{z \sim q_x} (\log(p(x|z))) - D_{KL} (q_\phi(z|x), p(z))$$

Głębokie autokodery wariacyjne

W [1] zaproponowano model wyżarzania (cyclical annealing schedule) w procesie doboru wartości beta:

$$\mathcal{L}_{ELBO} = \mathcal{L}_{REC} + \beta \mathcal{L}_{REG}$$

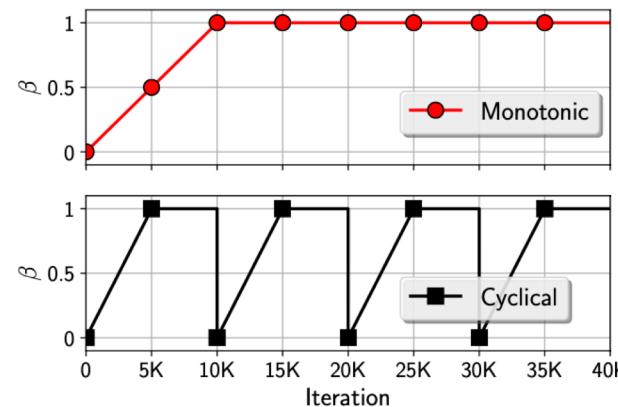


Figure 2: Comparison between (a) traditional monotonic and (b) proposed cyclical annealing schedules. In this figure, $M = 4$ cycles are illustrated, $R = 0.5$ is used for increasing within each cycle.

Głębokie autokodery wariacyjne

Problem: niech zmienna x ma rozkład Bernoulliego ($P(x=1)=p$, $P(x=0)=1-p$), funkcja masy prawdopodobieństwa:

$$\begin{aligned}f(x, p) &= p^x \cdot (1 - p)^{1-x} \\ \log(f(x, p)) &= \log(p^x) + \log((1 - p)^{1-x}) \\ \log(f(x, p)) &= x\log(p) + (1 - x)\log(1 - p)\end{aligned}$$

Uzyskaliśmy binarną entropię krzyżową ($q=1-p$).

Zauważmy jednak, że w powyższym opisie zmienna x jest dyskretna $\{0,1\}$. W częstych zastosowaniach, np. dla sygnałów, obrazów, mamy dane, dla których x jest zmienną w zakresie $[0,1]$. Wówczas warto rozważyć ciągły rozkład Bernoulliego zaproponowany w [1].

Jest to szczególnie istotne w autokoderach wariacyjnych. Wówczas do funkcji kosztu (oprócz KL i członu rekonstrukcyjnego związanego z entropią krzyżową dodamy trzeci człon funkcji kosztu tak jak w [1].

Przykłady praktyczne (notatnik interaktywny DVAE_faces_1.ipynb)

Głębokie autokodery wariacyjne w środowisku Jupyter Notebook. oraz

https://github.com/haofuml/cyclical_annealing

Dziękuję

Jacek Rumiński



Fundusze
Europejskie
Polska Cyfrowa



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Uczenie głębokie

Wykład 7: Transfer wiedzy

Jacek Rumiński
Katedra Inżynierii Biomedycznej, Wydział ETI



Rzeczpospolita
Polska



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu:

- **Transfer wiedzy**
- Pojedyncze i łączone zadania

Transfer wiedzy (ang. transfer learning)

Dwa aspekty - podejścia do terminu „transfer learning”:

1. „transfer of learning” - przenosimy wiedzę pomiędzy dziedzinami (np. wiedzę wynikającą z gry na gitarze przenosimy na grę na kontrabasie)
2. „learning to transfer” - uczymy się jak przenosić (stosować) wiedzę z jednej dziedziny do drugiej dziedziny (np. osoba uczy się jak uogólniać swoje doświadczenia poznawcze)

Celem transferu wiedzy w uczeniu maszynowym jest wykorzystanie wiedzy z powiązanej, źródłowej dziedziny w celu umożliwienia lub poprawy procesu uczenia dla innej, docelowej dziedziny. Przykładowo, prowadzić to może do znacznej redukcji potrzebnych przykładów dla dziedziny docelowej.

Ważny podział:

- transfer wiedzy dla bardzo podobnych dziedzin z jednorodną przestrzenią cech (ang. homogeneous transfer learning),
- transfer wiedzy dla odmiennych dziedzin z różnymi przestrzeniami cech (ang. heterogeneous transfer learning).

Transfer wiedzy (ang. transfer learning)

Istnieje szereg kategoryzacji metod transferu wiedzy. Niektóre z nich zostały omówione w [1][2]. Zgodnie z podaną literaturą zdefiniujmy dziedzinę (ang. domain) i zadanie (ang. task). Najpierw wprowadźmy podstawowe oznaczenia:

\mathcal{X} - przestrzeń cech obejmująca zbiór przykładów zestawu uczącego $\mathbf{X} = \{\mathbf{x} \mid \mathbf{x}_i \in \mathcal{X}, i = 1 \dots, N\}$, N-liczba przykładów w zbiorze.

\mathcal{Y} - przestrzeń etykiet dla zbioru uczącego $\{y_k \in \mathcal{Y}, k = 1 \dots, K\}$, K - liczba klas = $|\mathcal{Y}|$.

p_x - rozkład brzegowy (ang. marginal probability mass function) dla \mathcal{X} , np. dla dwóch cech - zmiennych losowych A i B i ich możliwych wartości a_j, b_i ($i/j=1, \dots$, liczba możliwych wartości cechy A/B) $p_A(a_j) = \sum_i p(a_j, b_i)$ - rozkład występowania wartości cech w zbiorze uczącym.

f - funkcja decyzyjna przypisująca etykietę $f(\mathbf{x}_i) = y_k$ lub prawdopodobieństwa: $f(\mathbf{x}_i) = \{P(y_k|\mathbf{x}_i) \mid y_k \in \mathcal{Y}, k = 1 \dots, K\}$. Funkcja ta jest uzyskiwana na podstawie założeń i danych.

Definicja 1 (Dziedzina): Dziedzinę \mathcal{D} tworzy zbiór $\mathcal{D} = \{\mathcal{X}, p_x\}$.

Definicja 2 (Zadanie): Zadanie \mathcal{T} tworzy zbiór $\mathcal{T} = \{\mathcal{Y}, f\}$.

[1] Fuzhen Zhuang, et. al (2019) A Comprehensive Survey on Transfer Learning,
<https://arxiv.org/abs/1911.02685>

[2] Ch.Tan, et al., (2018) A Survey on Deep Transfer Learning, <https://arxiv.org/abs/1808.01974>

Transfer wiedzy (ang. transfer learning)

Zakładając pojedyncze zadanie źródłowe i powiązaną z nim dziedzinę $\{\mathcal{D}_S, \mathcal{T}_S\}$ oraz analogiczną parę dla zadania docelowego $\{\mathcal{D}_D, \mathcal{T}_D\}$ transfer wiedzy wykorzystuje wiedzę związana z $\{\mathcal{D}_S, \mathcal{T}_S\}$ do uzyskania (poprawy) np. akceptowalnej funkcji decyzyjnej zadania docelowego $f^D()$ stanowiącego część \mathcal{T}_D .

Zauważmy, że w przypadku sztucznych sieci neuronowych funkcja decyzyjna jest reprezentowana przez model M sieci, powiązaną funkcję kosztu L, itp.

Rozpatrzmy przykładowy podział technik transferu wiedzy na:

1. metody bazujące na przykładach z dziedzin,
2. metody bazujące na transformacji (przekształceniu) przykładów z dziedzin,
3. metody bazujące na modelach.

Oddzielnie rozpatrzmy problem zastosowania opracowanych modeli jako praktyczny aspekt transferu wiedzy pomiędzy maszynami (transfer wiedzy pomiędzy ludźmi może być o wiele mniej efektywny - długotrwałe uczenie).

Transfer wiedzy (ang. transfer learning)

1. Metody bazujące na przykładach z dziedzin zakładają, że przykłady w dziedzinie źródłowej (źródłowych) są nieznacznie różne od przykładów dziedziny docelowej, ale mogą być wykorzystane po zastosowaniu określonej techniki przetwarzania.

Założymy, że mamy dwa zestawy danych w problemie rozpoznawania nowotworu prostaty: źródłowy o dużej liczbie przykładów w większości dla populacji osób starszych oraz docelowy o mniejszej liczbie przykładów z bardziej równomiernym rozkładem wieku.

Wiek będzie cechą X i jak możemy się domyślić prawdopodobieństwa występowania poszczególnych grup wiekowych w obu zbiorach będą różne: $P_S(X) \neq P_D(X)$. Najczęściej zakładamy jednak, że w obu zbiorach $P_S(Y|X) = P_D(Y|X)$.

W takim przypadku możemy zastosować np. strategię ważenia funkcji kosztu uzyskiwanych dla dziedziny źródłowej (minimalizujemy ryzyko):

$$\text{Ryzyko} = \min_f \left(\frac{1}{N_S} \sum_{i=1}^{N_S} \frac{P_D(\mathbf{x}_i)}{P_S(\mathbf{x}_i)} L(f(\mathbf{x}_{Si}, \boldsymbol{\theta}), y_{Si}) + \text{reg} \right)$$

Transfer wiedzy (ang. transfer learning)

W omawianej metodzie wagi są uzyskiwane na podstawie relacji prawdopodobieństw:

$$\beta_i = \frac{P_D(\mathbf{x}_i)}{P_S(\mathbf{x}_i)}$$

W innych metodach wagi mogą być estymowane innymi technikami, wyznaczane metodami iteracyjnymi itp.

2. Metody bazujące na transformacji (przekształceniu) przykładów z dziedzin. W metodach tych zakładamy, że dziedziny się różnią, ale możemy tak je przekształcić w przestrzeni cech, aby np. zredukować różnice w rozkładach prawdopodobieństw dla uzyskanych cech.

Przykładowo: poszukujemy takich funkcji uzyskiwania nowych cech Φ , które zminimalizują różnice wyrażane przez np. maksymalną średnią rozbieżność (ang. MMD - maximum mean discrepancy):

$$\min_{\Phi} \left(MMD(X_S, X_D, \Phi) = \left\| \frac{1}{N_S} \sum_{i=1}^{N_S} \Phi(\mathbf{x}_{Si}) - \frac{1}{N_D} \sum_{j=1}^{N_D} \Phi(\mathbf{x}_{Dj}) \right\|_{\mathcal{H}}^2 \right)$$

Transfer wiedzy (ang. transfer learning)

Metody bazujące na transformacji (przekształceniu) przykładów dziedzin wykorzystują szereg technik związanych z odwzorowaniem cech (np. PCA), grupowaniem cech, selekcją cech, kodowaniem cech (np. autokodery), itp.

Przykład: problem różnych dziedzin (i problem generalizacji): obrazy medyczne tej samej modalności, ale:

- a) uzyskiwane z różnych aparatów tej samej firmy,
- b) uzyskiwane z różnych aparatów różnych firm,
- c) uzyskiwane z różnych aparatów w różnych formatach (np. dane DICOM vs. dane JPEG), itp.

Integracja danych (np. w uczeniu federacyjnym) często wymaga stosownego przekształcenia przykładów.

Ważne: stosując różne wytrenowane modele trzeba zwrócić uwagę jak wcześniej przetworzono dane dla tych modeli. W TF dla szeregu modeli (np. VGG16, itp.) dostępna jest specjalna funkcja `preprocess_input(x)`, która „dopasowuje” dziedzinę wartości danych wejściowych.

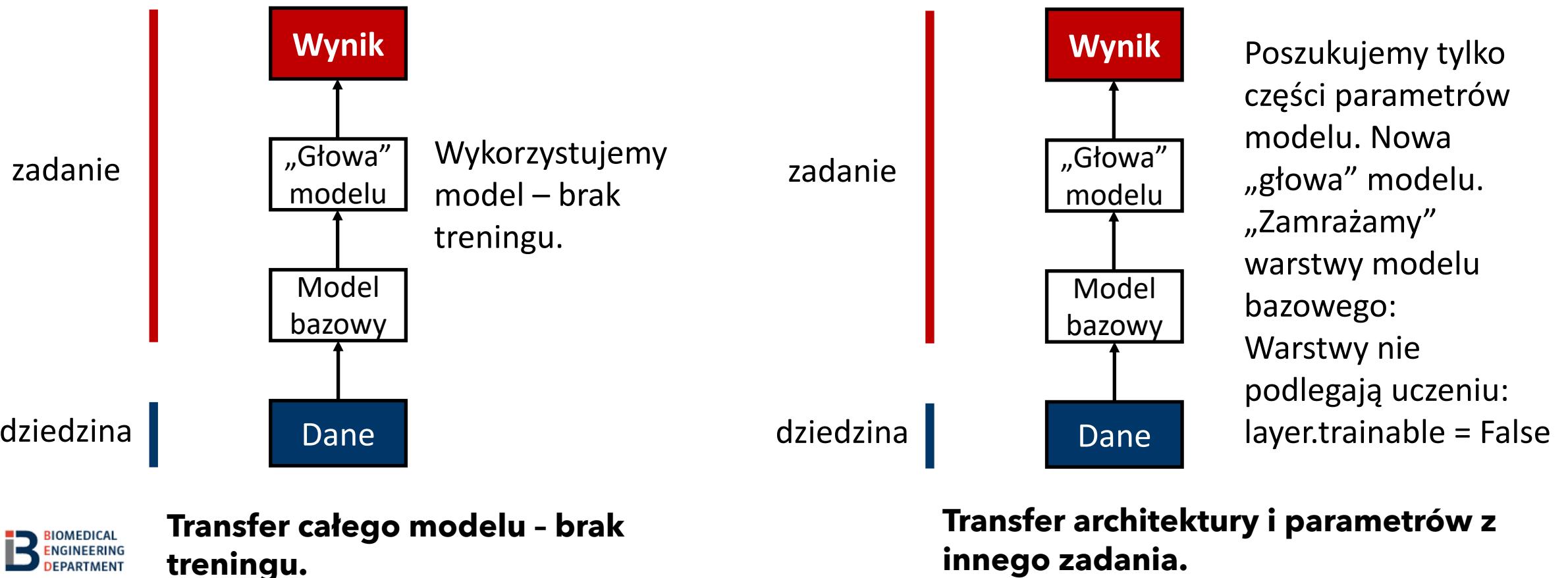
Plan wykładu:

- Transfer wiedzy
- **Pojedyncze i łączone zadania**

Transfer wiedzy (ang. transfer learning)

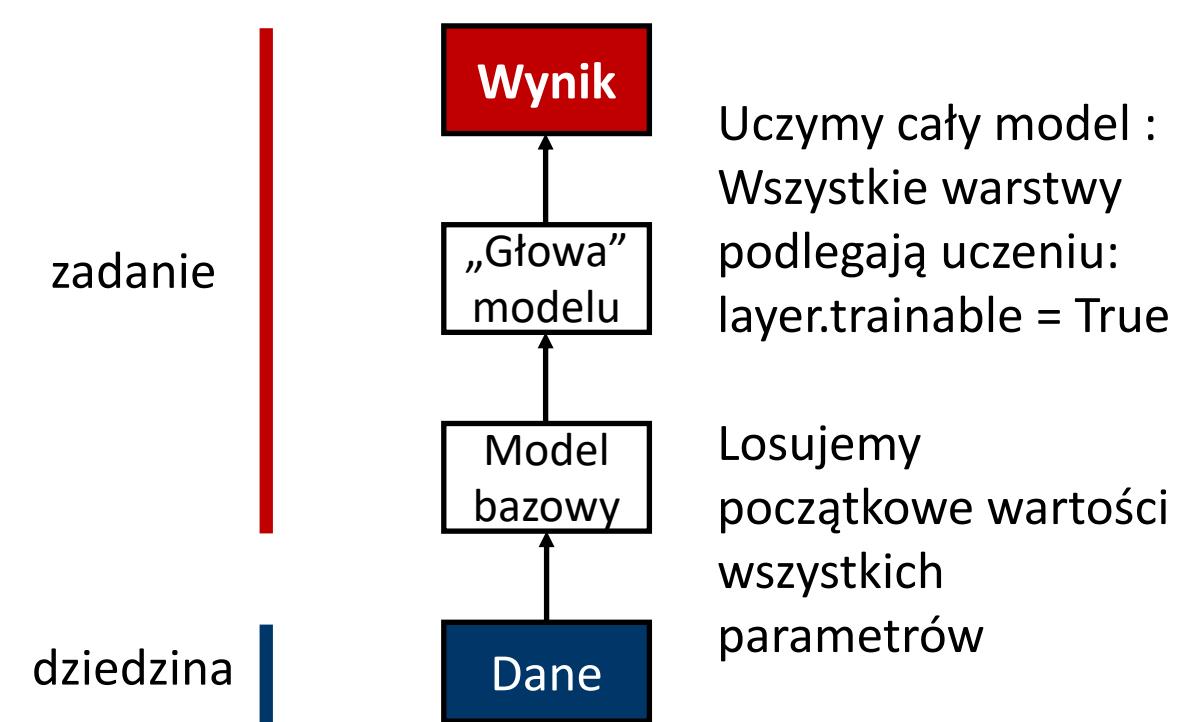
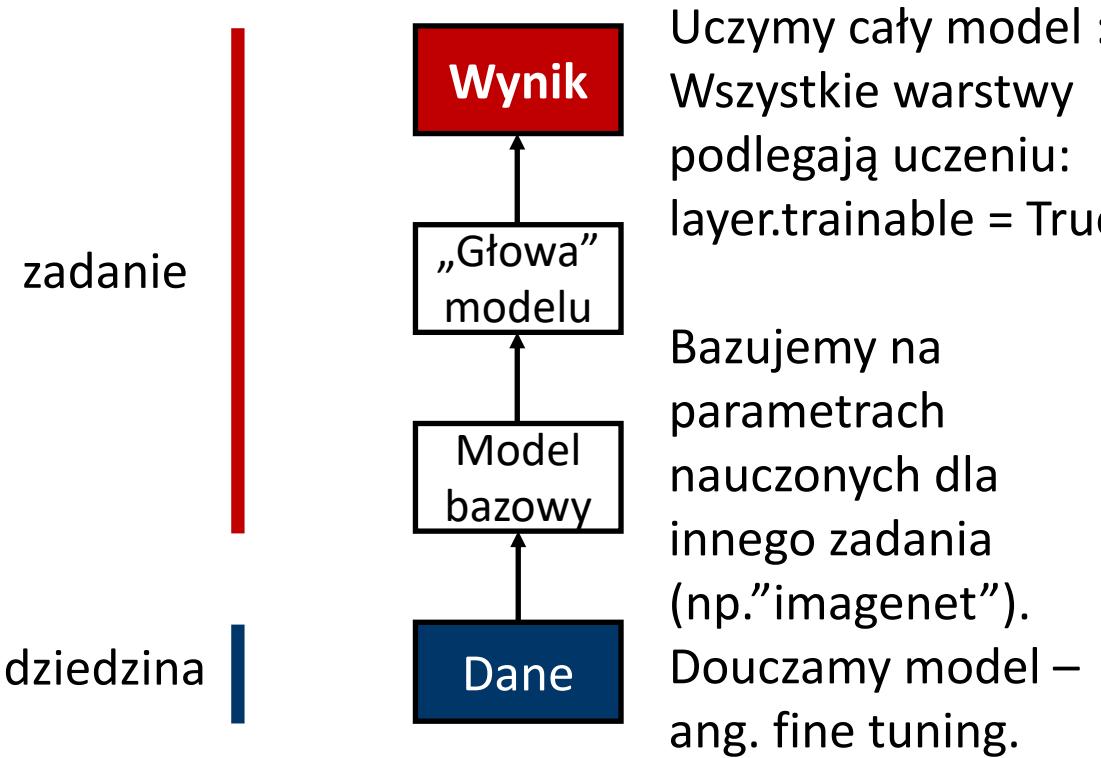
3. Metody bazujące na modelach - przykładowe rodzaje realizacji

Uczenie w realizacji pojedynczych zadań (ang. **Single-Task Learning**).



Transfer wiedzy (ang. transfer learning) : Single-Task Learning

Przykładowe realizacje c.d.

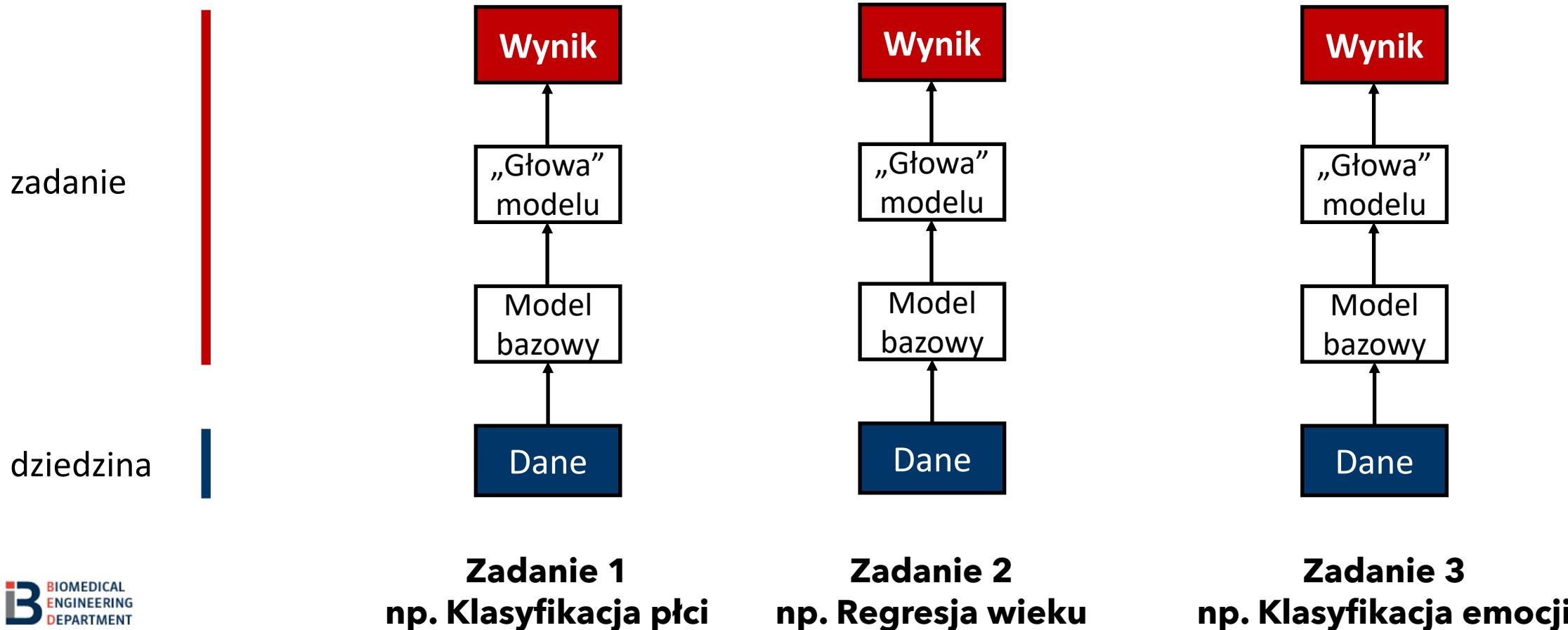


Transfer architektury i wartości początkowych z innego zadania.

Transfer architektury z innego zadania.

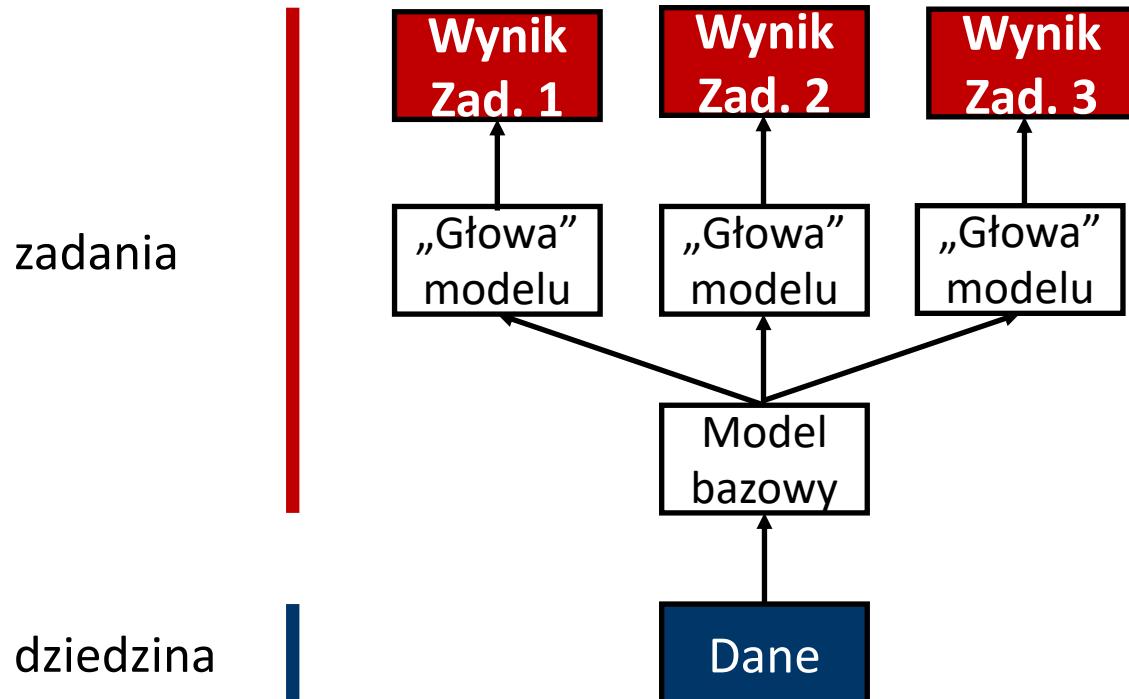
Transfer wiedzy (ang. transfer learning): Single-Task Learning

Często rozwiązuje się szereg pojedynczych zadań dla tej samej dziedziny. Przykładowo, dla obrazów twarzy moglibyśmy opracować trzy pojedyncze modele oddzielnie do: klasyfikacji płci, regresji wieku, klasyfikacji emocji (mimiki twarzy).



Transfer wiedzy (ang. transfer learning): Multi-Task Learning

Ten sam problem moglibyśmy zaadresować stosując uczenie w realizacji wielu zadań (ang. Multi-Task Learning), w ramach którego część modelu jest wspólna, na której bazują np. trzy odrębne „głowy” modelu. Proces uczenia bazuje na pojedynczym modelu poprzez odpowiednią funkcję kosztu (ang. hard parameter sharing).



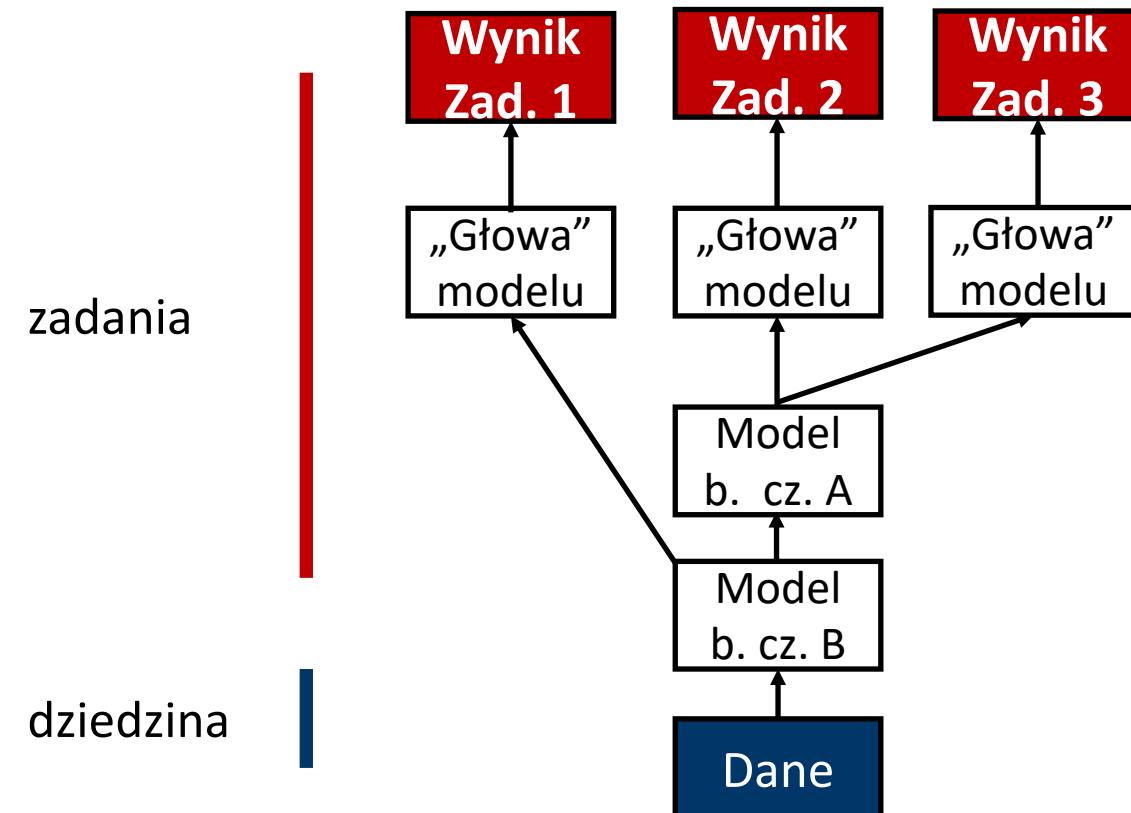
$$\operatorname{argmin}_W \left(\frac{1}{T} \sum_{t=1}^T \frac{1}{N_t} \sum_{i=1}^{N_t} \frac{1}{N_t} L(y_{ti}, f_t(\mathbf{x}_i)) \right)$$

Trzy zadania, np. klasyfikacja płci,
regresja wieku, klasyfikacja emocji

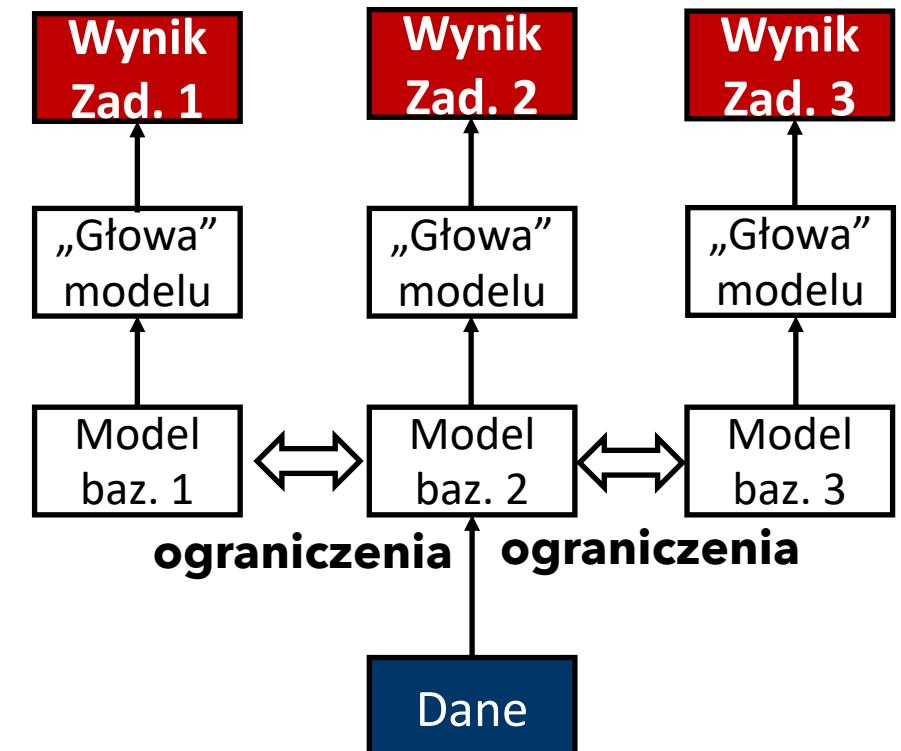
Transfer wiedzy (ang. transfer learning): Multi-Task Learning

„Głowy” modelu mogą być dołączone do różnych warstw modelu bazowego.

Modele bazowe mogą się również różnić, ale nałożone są nie dodatkowe (wzajemne) ograniczenia (ang. soft parameter sharing), realizowane np. poprzez specyficzne funkcje kosztu.



„Głowy” modelu dołączone do różnych warstw modelu bazowego.



Oddzielne modele bazowe, ale z dodatkowymi ograniczeniami.

Transfer wiedzy (ang. transfer learning): Multi-Task Learning

W uczeniu z wieloma zadaniami rozróżnia się czasami konfiguracje, w których występuje jedno z zadań jako zadanie główne (ang. main task), a pozostałe zadania są zadaniami pomocniczymi (ang. auxiliary tasks).

Warto tutaj przywołać model GoogLeNet, w którym w procesie uczenia wykorzystywano dwa pomocnicze bloki klasyfikatorów (ang. auxiliary classifiers), dołączone do różnych poziomów modelu bazowego, w celu redukcji problemu zanikającego gradientu.

W zastosowaniach praktycznych zadania pomocnicze mogą być związane z konkretnymi funkcjami kosztu dla szczegółowych zadań powiązanych z zadaniem głównym. Przykładowo, sterowanie pojazdem autonomicznym (zadanie główne) może być powiązane z segmentacją sceny (zadania pomocnicze 1), wykrywaniem obiektów (zadanie pomocnicze 2), wyznaczaniem map odległości od obiektów (zadanie pomocnicze 3).

Ciekawy przegląd metod związanych z MTL przedstawiono w [1].

Transfer wiedzy (ang. transfer learning)

Szczególnym przykładem transferu wiedzy jest zastosowanie całego modelu, dla którego wyznaczono parametry w określonym zadaniu uczenia. Model taki możemy zastosować do przykładu z tej samej dziedziny lub zbliżonej dziedziny.

```
model = ResNet50(weights='imagenet')
```

```
img_path = 'test.png'  
img = image.load_img(img_path, target_size=(224, 224))  
x = image.img_to_array(img)  
x = np.expand_dims(x, axis=0)  
x = preprocess_input(x)
```

```
preds = model.predict(x)  
print('Predicted class:', decode_predictions(preds, top=3)[0])
```



'West_Highland_white_terrier', P=0.8958,
'Sealyham_terrier', P=0.0708, 'Maltese_dog', P=0.0162



'police_van', P=0.9748,
'tow_truck', P=0.0052, 'car_wheel', P=0.0036

Przykłady praktyczne (notatnik interaktywny DL_07_01_CNN_Transfer_Learning.ipynb)

Transfer wiedzy w środowisku Google Colaboratory/ Jupyter Notebook.

https://colab.research.google.com/drive/1T-hcW9_xHvbmuiNwwVlef dgU8vo2f0j_?usp=sharing

Dziękuję

Jacek Rumiński



Fundusze
Europejskie
Polska Cyfrowa

Rzeczpospolita
Polska



KANCELARIA PREZESA RADY MINISTRÓW
THE CHANCELLERY OF THE PRIME MINISTER

Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Uczenie głębokie: Modele RNN

Szymon Zaporowski



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

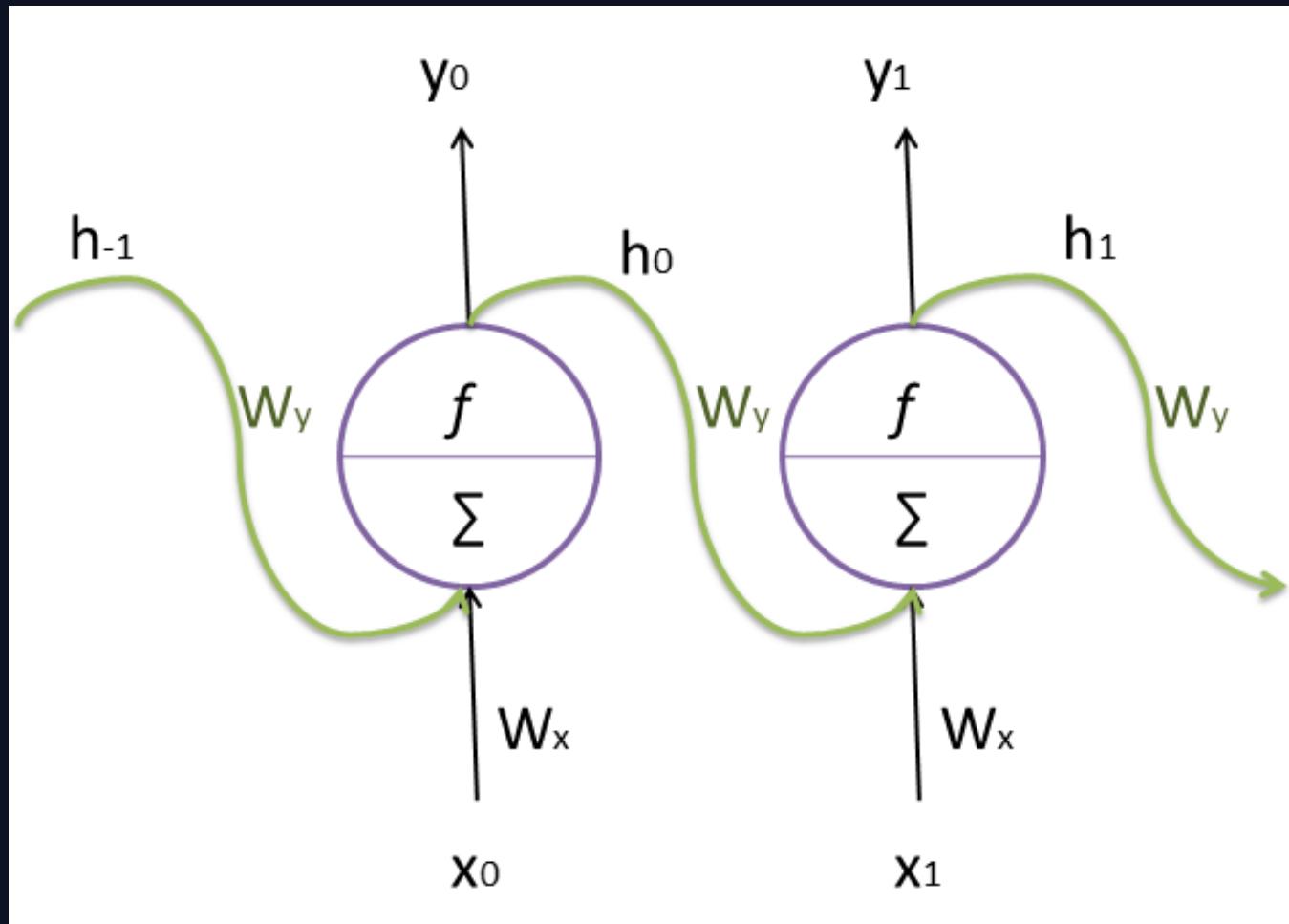
Plan wykładu

- RNN – charakteryzacja sieci
- Sposób działania sieci RNN
- Rodzaje architektur RNN i ich zastosowania
- Rozwijanie grafów sieci RNN
- Przykład działania sieci RNN – notatnik jupyter

Czym jest RNN?

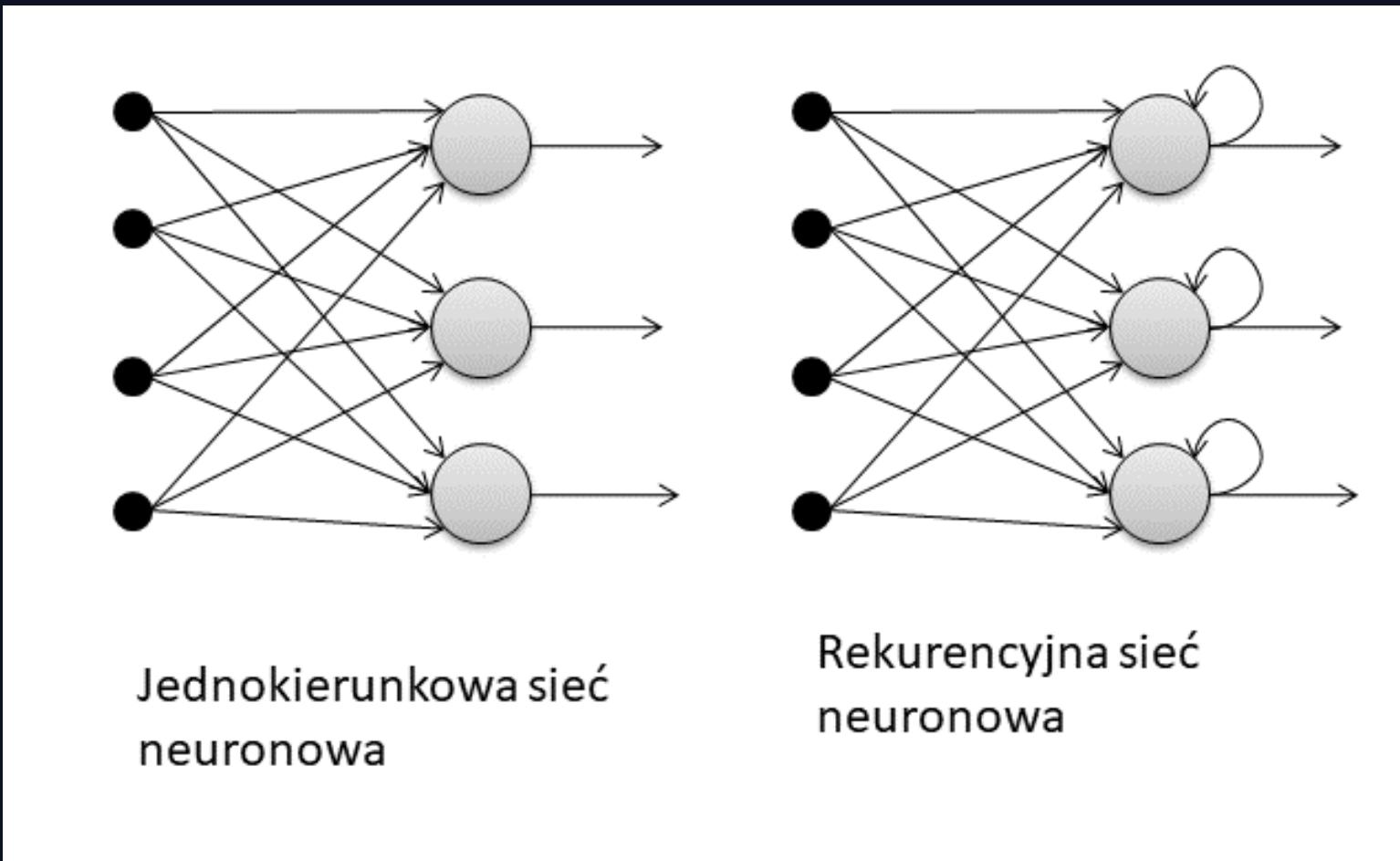
- Rekurencyjne sieci neuronowe (ang. Recurrent Neural Networks) to klasa sieci neuronowych, które umożliwiają wykorzystanie poprzednich wyjść jako danych wejściowych i posiadają ukryte stany – wykorzystane jest sprzężenie zwrotne
- W tradycyjnych sieciach neuronowych wszystkie wejścia i wyjścia są od siebie niezależne.
- W przypadku rekurencyjnych sieci neuronowych dane wejściowe z poprzednich kroków są wprowadzane na wejście stanu bieżącego.

Czym jest RNN?



- Y_n – wyjście
- X_n – wejście
- h_n – stan ukryty
- W_x – wagi połączeń dla obecnego roku czasowego
- W_y – wagi połączeń dla poprzedniego kroku czasowego

Porównanie rekurencyjnej sieci neuronowej do sieci jednokierunkowej



Sposób działania RNN

- W sieci RNN informacja przechodzi przez pętlę. Kiedy podejmuje decyzję, bierze pod uwagę bieżące dane wejściowe, a także to, czego nauczył się z danych wejściowych, które otrzymał wcześniej.
- Dzięki pamięci wewnętrznej RNN mogą zapamiętywać ważne rzeczy dotyczące otrzymanych danych wejściowych, co pozwala im bardzo precyzyjnie przewidywać, co będzie dalej.
- Rekurencyjne sieci neuronowe mogą zapewnić znacznie głębsze zrozumienie sekwencji i jej kontekstu w porównaniu z innymi algorytmami.

„RNN = najbliższa przeszłość + teraźniejszość”

Zalety sieci RNN

- Możliwość przetwarzania danych wejściowych o dowolnej długości
- Zwiększając rozmiar danych wejściowych nie zwiększa się rozmiar modelu
- Wagi są współdzielone w czasie
- Model RNN zapamiętuje każdą informację przez cały czas (pomocne w przypadku predykcji szeregów czasowych)
- Może wykorzystywać pamięć wewnętrzną do przetwarzania dowolnych serii danych wejściowych

Wady sieci RNN

- Ze względu na powtarzalny charakter obliczenia są powolne
- Trudności w dostępie do informacji z dużo wcześniejszych etapów
- Mogą pojawić się problemy eksplodowania i zanikania gradientu
- Przy zastosowaniu relu lub tanh jako funkcji aktywacji, przetwarzanie bardzo długich sekwencji staje się bardzo trudne
- Trenowanie modeli RNN może być trudne (dlaczego?)

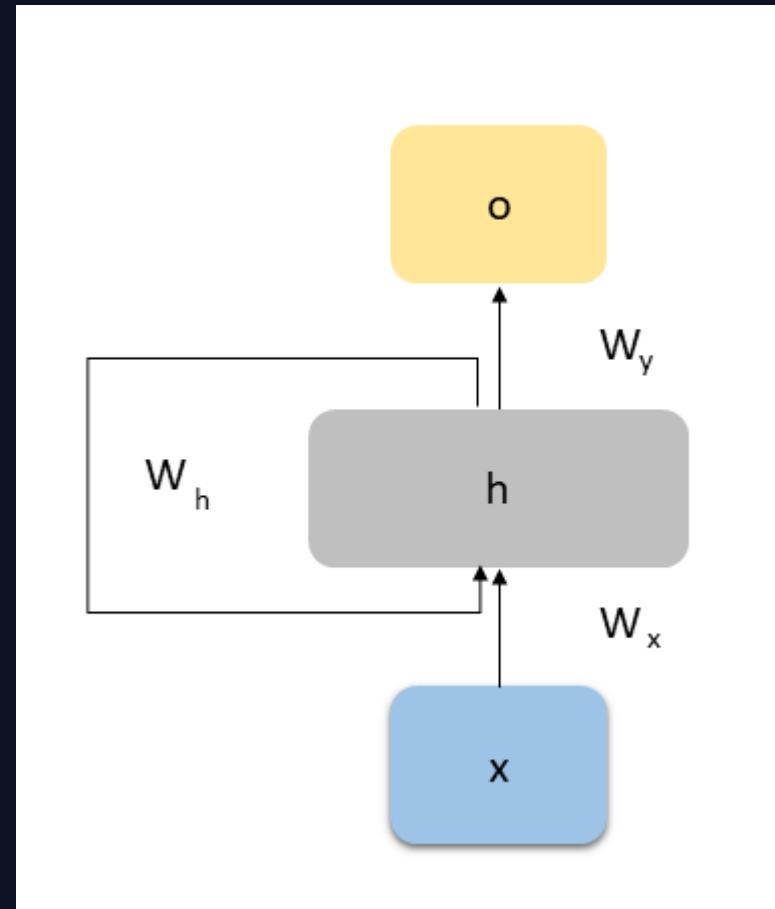
Rodzaje architektur sieci RNN

- Jeden do jednego
- Jeden do wielu
- Wiele do jednego
- Wiele do wielu

Architektura jeden do jednego

- Pojedyncze wejście jest mapowane na pojedyncze wyjście.
- Przykład: Słownik ma sześć znaków: S, Z, K, O, Ł, A, wprowadzamy pierwszy znak S jako dane wejściowe, RNN oblicza prawdopodobieństwo wystąpienia wszystkich słów w słowniku, aby przewidzieć następną literę.
- Zastosowanie: generowanie tekstu, przewidywanie słów, prognozy giełdowe

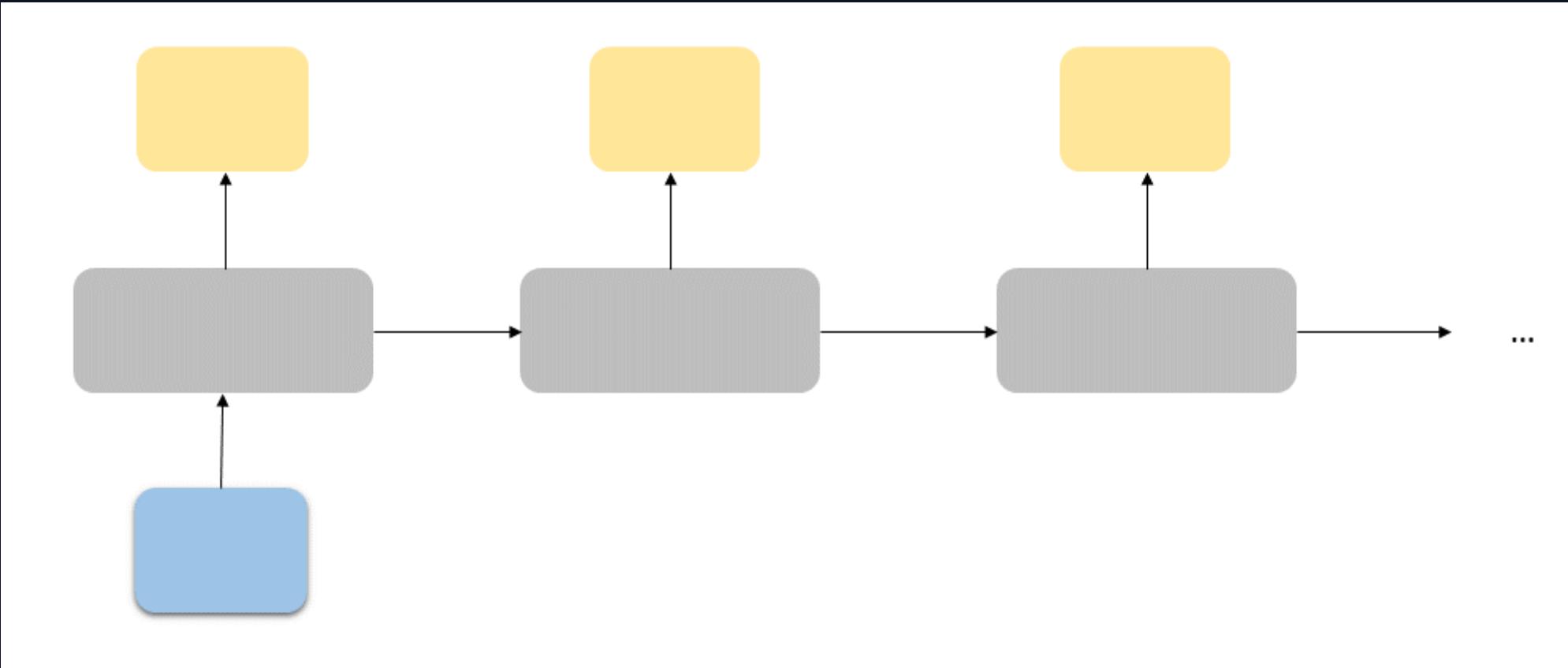
Architektura jeden do jednego



Architektura jeden do wielu

- Pojedyncze wejście jest mapowane na różne wartości wyjściowe.
- Przykład: W kroku czasowym $t=0$ słowo „dzień” jest przewidywane, a w następnym kroku czasowym $t=1$ poprzednio ukryty stan h_0 jest używany do przewidywania następnego słowa, którym jest „dobry”.
- Zastosowanie: automatyczne podpisy pod obrazami, generowanie muzyki

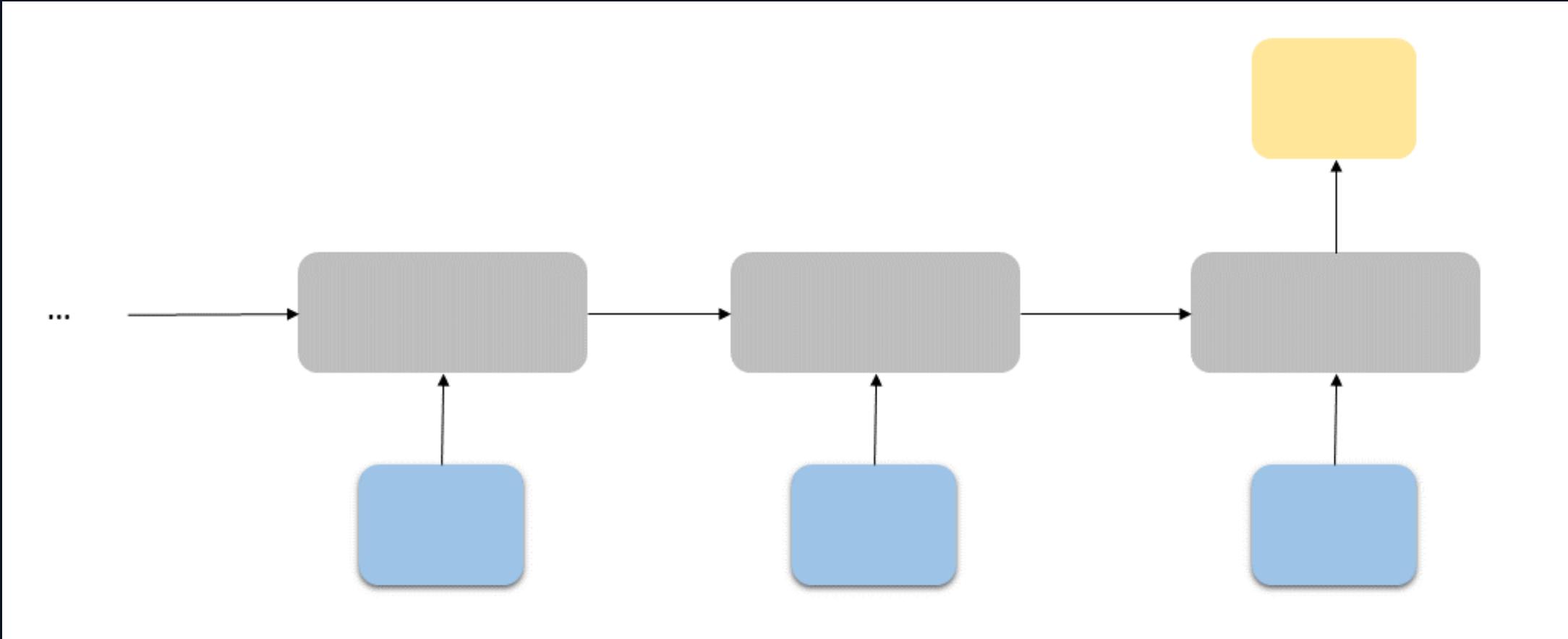
Architektura jeden do wielu



Architektura wiele do jednego

- Sekwencja wejść jest mapowana na pojedynczą wartość wyjściową.
- Przykład: Podczas trenowania modelu jako dane wejściowe model przyjmuje zdania jako sekwencje słów – każde słowo na osobnym wejściu, a jako wyjście sieć określa czy zdanie to jest nacechowane pozytywnie, czy negatywnie – nadaje zdaniu etykietę. „Ten telefon dobrze działa.” „Jestem rozczarowany.”
- Jako dane wejściowe: tekst recenzji, wyjście: ocena książki, filmu.
- Zastosowanie: ogólna klasyfikacja

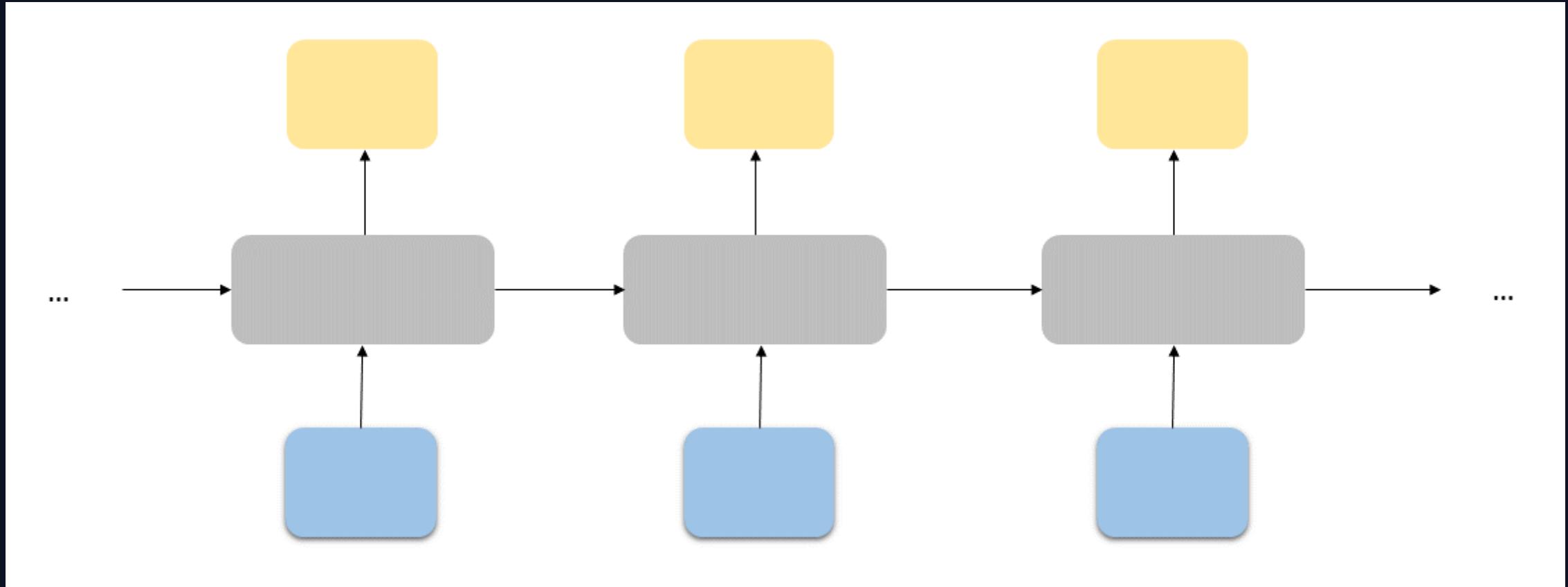
Architektura wiele do jednego



Architektura wiele do wielu: $T_x = T_y$

- Sekwencja danych wejściowych o długości T jest odwzorowywana na sekwencję danych wyjściowych o tej samej długości T .
- Przykład: Jako wejście podajemy sekwencję wideo, którą sieć odczytuje jako poszczególne klatki, a jako wyjście z sieci otrzymujemy etykiety z oznaczeniami, jakie elementy pojawiły się na danej klatce.
- Zastosowanie: Klasyfikacja wideo

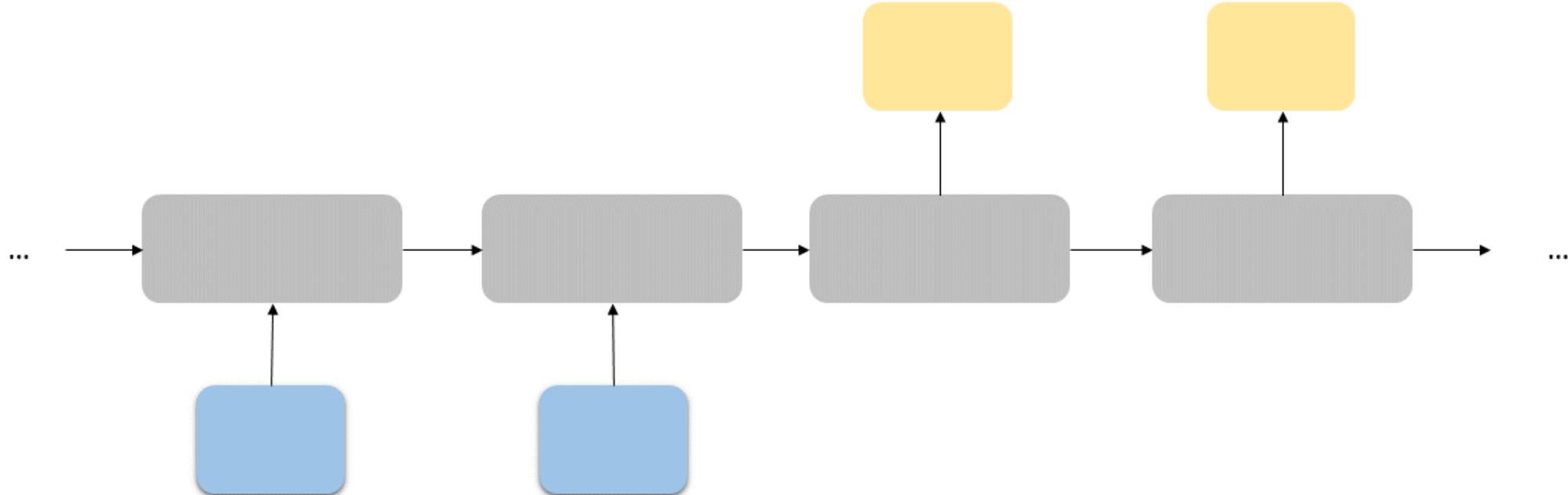
Architektura wiele do wielu: $T_x = T_y$



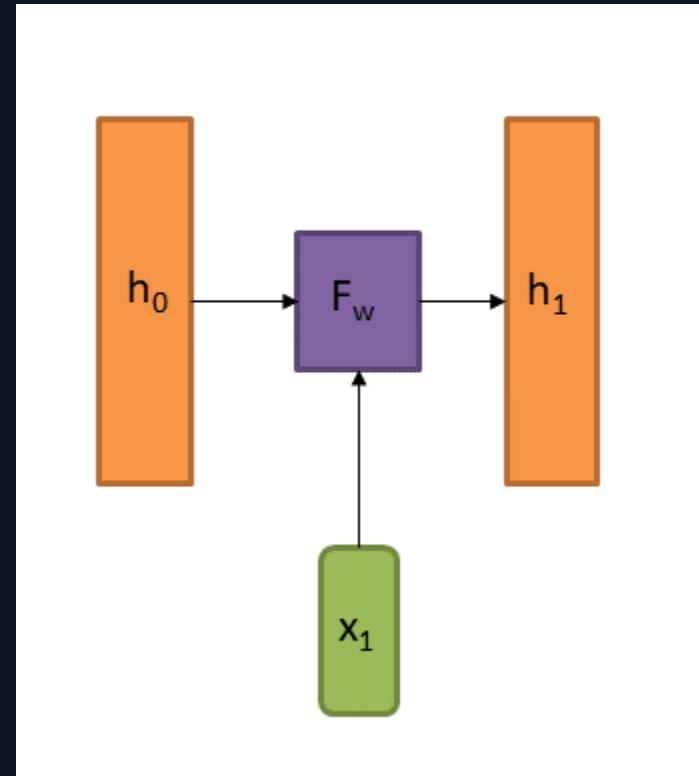
Architektura wiele do wielu: $T_x \neq T_y$

- Sekwencja danych wejściowych o dowolnej długości jest odwzorowywana na sekwencję danych wyjściowych o dowolnej długości.
- Przykład: Jako wejście do sieci podajemy zdanie w języku polskim, jako wyjście otrzymujemy zdanie przetłumaczone na język angielski. Ze względu na różne struktury gramatyczne języków liczba słów na wejściu może być inna niż na wyjściu z sieci.
- Zastosowanie: tłumaczenie maszynowe, chatboty

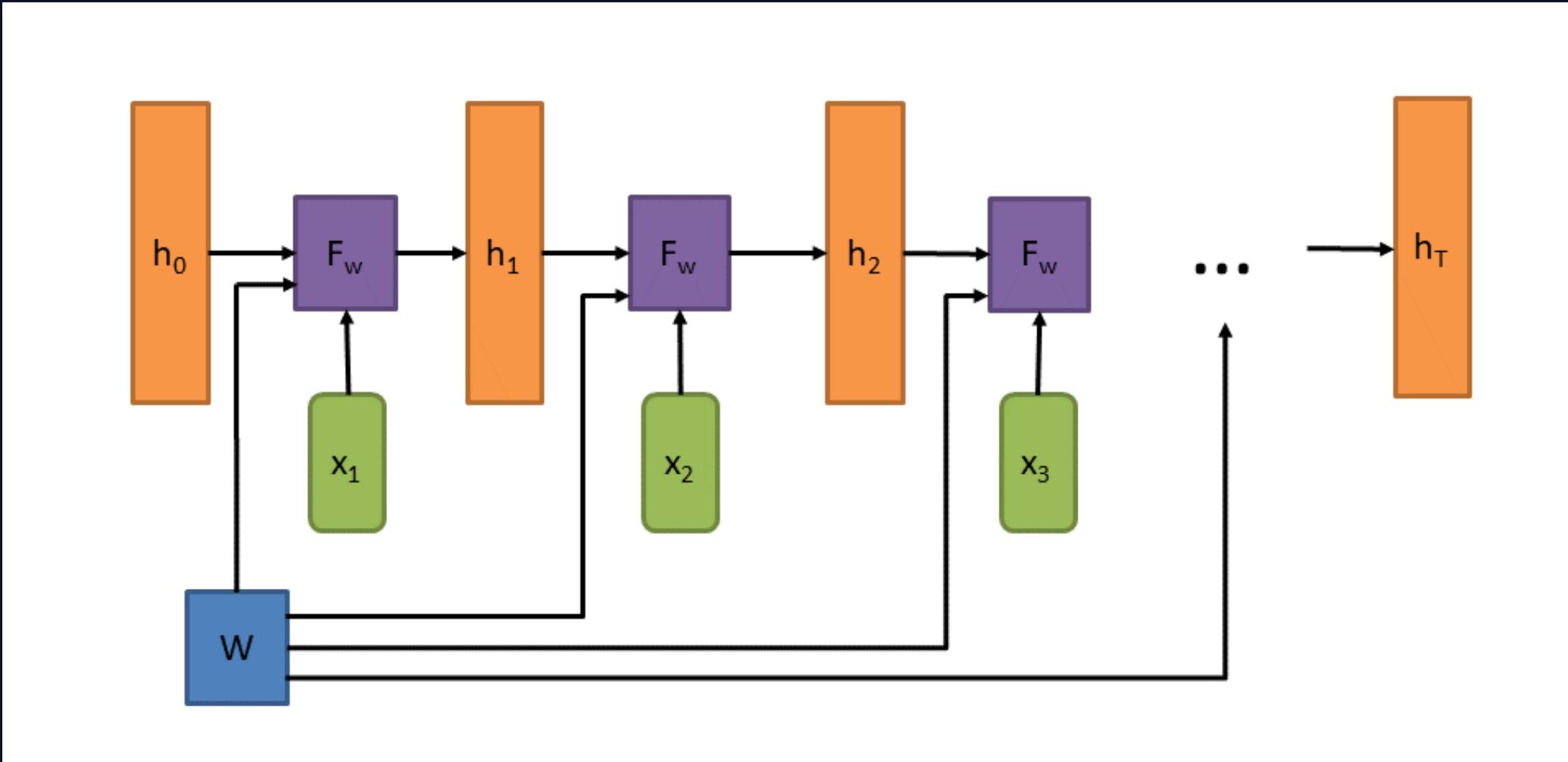
Architektura wiele do wielu : $T_x \neq T_y$



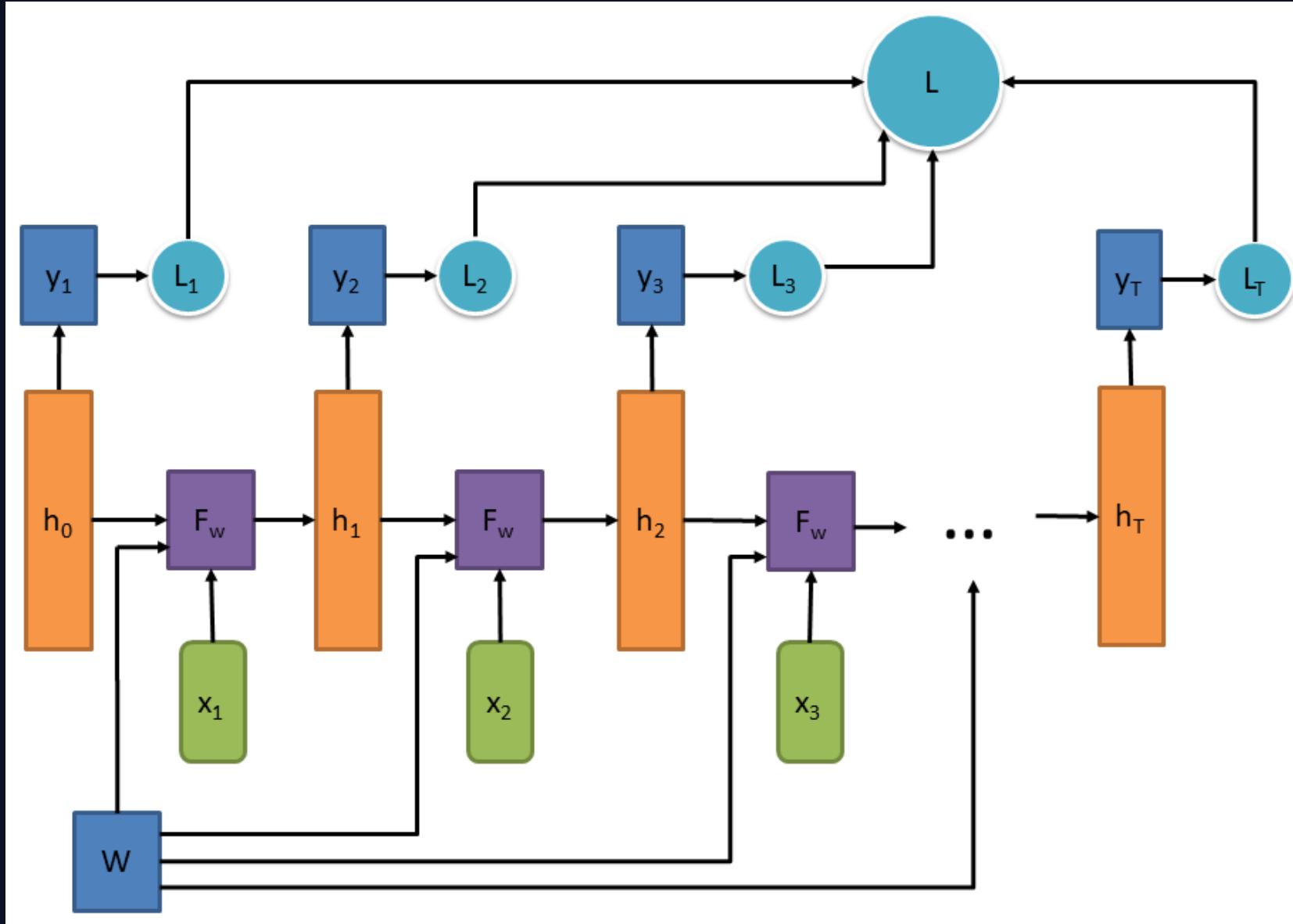
Architektury RNN – graf obliczeniowy



Architektury RNN – graf obliczeniowy



Architektury RNN – graf obliczeniowy

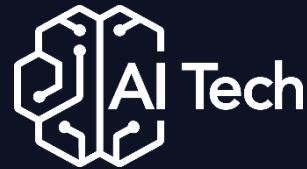


Wprowadzenie do sieci RNN w środowisku Google Colaboratory/ Jupyter Notebook

https://colab.research.google.com/drive/1BmJMY60-4_UblxpoBT4uX5p7i6hmqjPb#scrollTo=jdecyDDYyzRr



POLITECHNIKA
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Dziękuję

SZYMON ZAPOROWSKI



Fundusze
Europejskie
Polska Cyfrowa



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Uczenie głębokie: Modele RNN i ich rozwój

Szymon Zaporowski



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu

- Trening RNN Propagacja wsteczna w czasie
- Forsowanie nauczyciela
- Przycinanie gradientu – rozwiązywanie problemu eksplodującego i zanikającego gradientu
- Inne zastosowania sieci RNN
- Pierwsze sieci RNN – Echo State, sieci Hopfielda, sieci BAM
- Sieci LSTM i GRU
- Dwukierunkowe sieci RNN
- Głębokie sieci RNN

Kroki do treningu rekurencyjnej sieci neuronowej

1. W warstwach wejściowych początkowe dane są wysyłane, mają one tą samą wagę oraz tą samą funkcje aktywacji.
2. Wykorzystując aktualne dane wejściowe oraz wyjście stanu poprzedniego obliczany jest obecny stan.
3. Obecny wyliczony stan stanie się stanem poprzednim dla kolejnego kroku czasowego.
4. Kroki są powtarzane, aż do przetworzenia wszystkich danych wejściowych jednocześnie korzystając z informacji z kroków poprzednich.
5. Ostatni krok jest obliczany na podstawie aktualnego stanu końcowego i wszystkich poprzednich kroków.
6. Obliczany jest błąd poprzez obliczenie różnicy
7. Następuje wsteczna propagacja błędów oraz aktualizacja wag.

Propagacja wsteczna w czasie - wstęp

- W jakim przypadku zapamiętywanie poprzednich okresów jest przydatne?
- Łatwiej jest rozpoznać poruszający się obiekt, jeśli nasza sieć uwzględnia zmiany w scenie od czasu $t-1$ do czasu t , co wymaga pamięci czasu $t-1$.
- Propagacja wsteczna w czasie to zastosowanie algorytmu propagacji wstecznej do rekurencyjnej sieci neuronowej stosowanej do danych sekwencyjnych, takich jak szeregi czasowe.

Propagacja wsteczna w czasie

- Jest to technika służąca do rozwijania w czasie rekurencyjnych sieci neuronowych.
- Propagacja wsteczna w czasie (ang. Backpropagation Through Time) opiera się na gradiencie.
- Dokonując wstecznej propagacji w czasie powodujemy, że elementy z ostatniego stanu mogą mieć wpływ na sam początek.
- Obliczane są przejścia wstecz przez całą sekwencję dzięki czemu uzyskujemy gradienty i aktualizacje wag.
- Jedna aktualizacja z wsteczną propagacją w czasie może rozwinąć sieć, która jest później generowana przez sekwencje wejściową.

Algorytm propagacji wstecznej w czasie

1. Przedstaw w sieci sekwencję kroków czasowych par wejścia i wyjścia.
2. Rozwiń sieć, a następnie oblicz i zsumuj błędy w każdym kroku czasowym.
3. Zwiń sieć i zaktualizuj wagи.
4. Powtórz.

Koncepcja BPTT

- Rekurencyjna sieć neuronowa wyświetla jedno wejście w każdym kroku czasowym i przewiduje jedno wyjście.
- Koncepcyjnie BPTT działa poprzez rozwijanie wszystkich wejściowych kroków czasowych. Każdy przedział czasowy ma jeden przedział czasowy wejścia, jedną kopię sieci i jedno wyjście. Błędy są następnie obliczane i akumulowane dla każdego kroku czasowego. Sieć jest przywracana, a wagi są aktualizowane.
- Każdy krok czasowy rozwiniętej rekurencyjnej sieci neuronowej może być postrzegany jako dodatkowa warstwa, biorąc pod uwagę zależność kolejności problemu, a stan wewnętrzny z poprzedniego kroku czasowego jest traktowany jako dane wejściowe w kolejnym kroku czasowym.

Zalety i wady propagacji wstecznej w czasie

- + Jest szybsza dla RNN niż techniki optymalizacji ogólnego przeznaczenia, np. optymalizacja ewolucyjna.
- Ze względu na powtarzające się sprzężenia zwrotne propagacja wsteczna w czasie ma tendencje do tworzenia chaotycznych odpowiedzi na powierzchni błędu. Skutkuje to częstym występowaniem lokalnych optimów.
- Może być używana do ograniczonej liczby kroków czasowych, ponieważ przy coraz większej liczbie kroków gradient staje się zbyt mały (problem zanikającego gradientu).
- Może być kosztowny obliczeniowo wraz ze wzrostem liczby kroków czasowych.

Propagacja wsteczna w czasie

- Klasyczna propagacja wsteczna szkoli sieć neuronową w czasie „t”, natomiast propagacja wsteczna w czasie „t” oraz uwzględniając to co wydarzyło się wcześniej, czyli $t-1$, $t-2$, $t-3$ itd.
- Pochodne, które są obliczane, można wykorzystać w rozpoznawaniu wzorców, identyfikacji systemów oraz w sterowaniu stochastycznym i deterministycznym.
- BPTT można zastosować do sieci neuronowych, układów równań ze sprzężeniem do przodu, układów z opóźnieniem czasowym, układów z chwilowym sprzężeniem zwrotnym między zmiennymi (jak w równaniach różniczkowych zwyczajnych lub modelu równań równoczesnych)

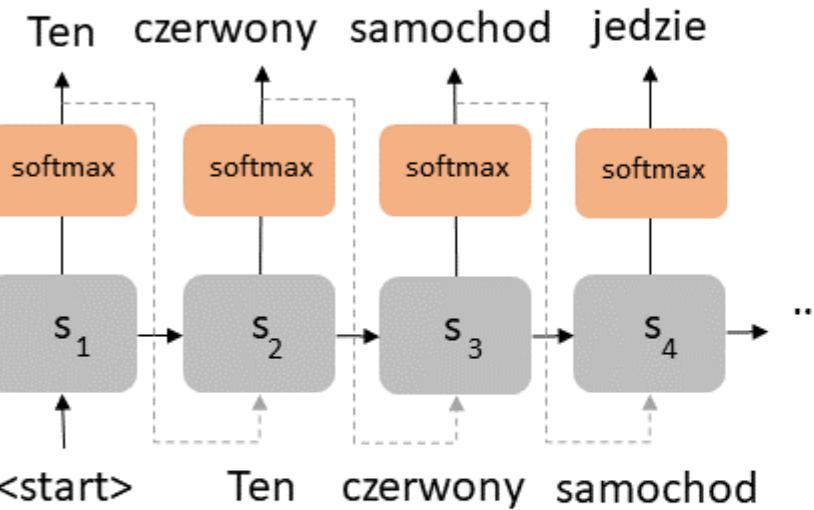
Forsowanie nauczyciela

- Z angielskiego Teacher forcing to metoda uczenia RNN, która jako dane wejściowe wykorzystuje *ground truth*, a nie dane wyjściowe z poprzedniego kroku czasowego.
- Działa poprzez wykorzystanie rzeczywistych lub oczekiwanych danych wyjściowych ze zbioru danych treningowych w bieżącym kroku czasowym $y(t)$ jako danych wejściowych w następnym kroku czasowym $X(t+1)$, a nie danych wyjściowych generowanych przez sieć.

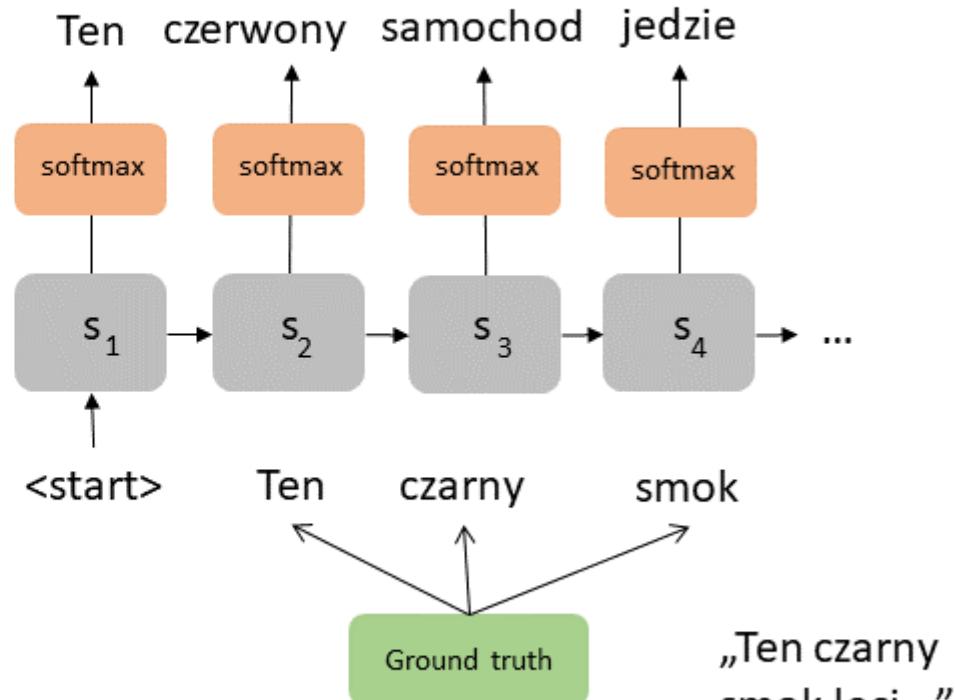
Zasada działania metody forsowania nauczyciela

- Po uzyskaniu odpowiedzi na część pierwszą, nauczyciel porówna naszą odpowiedź z prawidłową, zapisze wynik za część pierwszą i poda poprawną odpowiedź, abyśmy mogli ją wykorzystać dla kolejnej części.
- „Nauczyciel” w przypadku popełnienia błędu podaje prawidłową odpowiedź, dzięki czemu koryguje dalsze przewidywania. Błędnie przewidziane słowo nie jest podawane do kolejnego kroku (jak w przypadku zwykłej RNN).

Uczenie z forsowaniem nauczyciela i bez



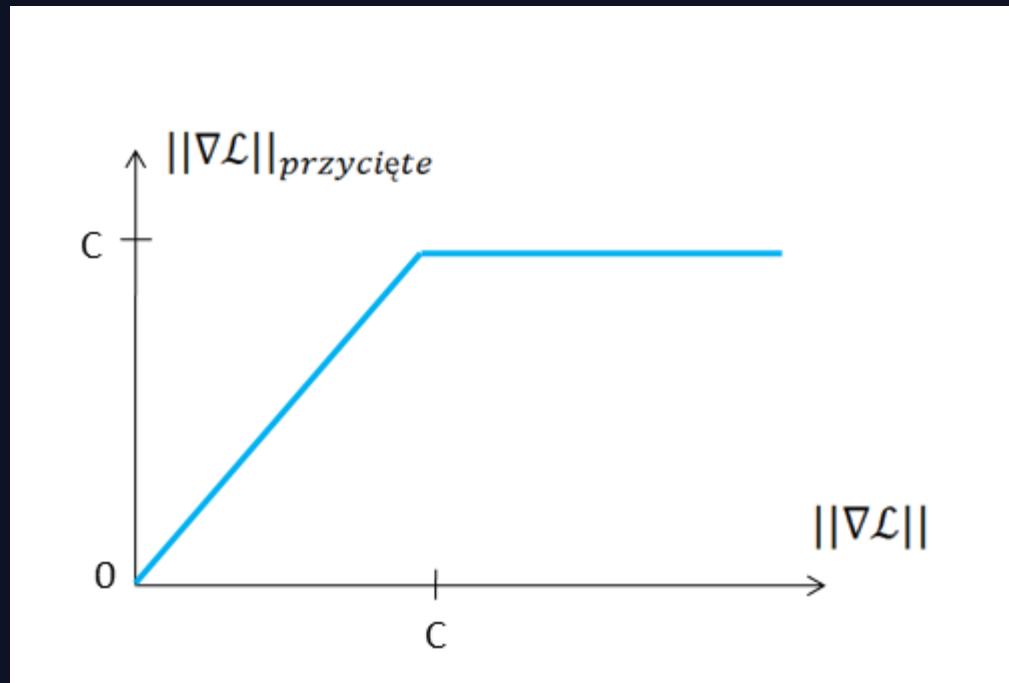
Bez forsowania nauczyciela



Z forsowaniem nauczyciela

Przycinanie gradientu

- Technika polegająca na ograniczeniu maksymalnej wartości gradientu służąca do kontrolowania zjawiska eksplodujących gradientów.



Inne zastosowania RNN

- Wykrywanie anomalii szeregów czasowych
- Nauka rytmu
- Kompozycja muzyczna
- Rozpoznawanie pisma odręcznego
- Nauka gramatyki
- Rozpoznawanie ludzkich działań
- Predykcja w obszarze zarządzania procesami biznesowymi
- Predykcja w ścieżkach opieki medycznej.
- Rozpoznawanie mowy
- Modelowanie języka i generowanie tekstu
- Podsumowanie tekstu
- Analiza Call Center
- Wykrywanie twarzy
- Aplikacje OCR jako rozpoznawanie obrazu

Sieci Echo State

- ESN zostały wprowadzone jako nowatorskie podejście do projektowania sieci RNN.
- Główne podejście ESN polega na obsłudze losowej, dużej, stałej, powtarzalnej sieci neuronowej z sygnałem wejściowym, który indukuje nieliniowy sygnał odpowiedzi w każdym neuronie w tej sieci „zbiornikowej”.
- Innym podejściem jest podłączenie pożądanego sygnału wyjściowego za pomocą możliwej do trenowania liniowej kombinacji wszystkich tych sygnałów odpowiedzi.
- Idea ESN może być rozumiana jako sposób uzyskania pożądanego docelowego wyniku z RNN, ucząc się łączenia sygnałów z losowo skonfigurowanych zestawów neuronowych.
- Proces uczenia ESN jest łatwiejszy i wymaga mniej wysiłku obliczeniowego niż zwykły RNN o tym samym rozmiarze.

Warianty Sieci Echo State - ESN

- Sieci typu Echo State mogą być konfigurowane z lub bez bezpośredniego trenowania połączeń wejścia-wyjścia, z lub bez sprzężenia zwrotnego wyjście-zbiornik, z różnymi typami neuronów, różnymi wzorcami łączności wewnętrznej rezerwuaru itp.
- Wagi wyjściowe mogą być obliczone za pomocą dowolnego dostępnego algorytmu regresji liniowej w trybie offline lub online .
- Wspólnym motywem we wszystkich tych odmianach jest wykorzystanie RNN jako *losowego nieliniowego ośrodka pobudzającego*, którego wielowymiarowa dynamiczna odpowiedź „echą” na wejściowe (i/lub wyjściowe sprzężenie zwrotne) jest wykorzystywana jako nieortogonalna podstawa sygnału do zrekonstruować żądany wynik przez kombinację liniową, minimalizując niektóre kryteria błędów.

Zastosowanie Sieci Echo State

- Metody obliczania zbiorników są alternatywą dla RNN, gdy modelowany system nie jest zbyt skomplikowany oraz gdy pożądane jest tani, szybki i adaptacyjny proces uczenia. Odnosi się to do wielu zastosowań w przetwarzaniu sygnałów, na przykład w przetwarzaniu biosygnałów, teledetekcji czy sterowaniu silnikiem robota.
- Od ok. 2010 r., sieci typu Echo State stały się istotne i dość popularne jako zasada obliczeniowa, która dobrze łączy się z niecyfrowymi podłożami obliczeniowymi, na przykład mikrochipami optycznymi, mechanicznymi nanooscylatorami, mieszaninami nanorurek węglowych/polimerów oraz sztucznymi miękkimi kończynami.

Kiedy warto użyć ESN

- ESN ze względu na swoją budową nie cierpią z powodu zjawisk typu eksplodujący/ zanikający gradient
- ESN jest szybki i nie posiada propagacji wstecznej dla rezerwuaru
- ESN dobrze radzą sobie z chaotycznymi szeregami czasowymi

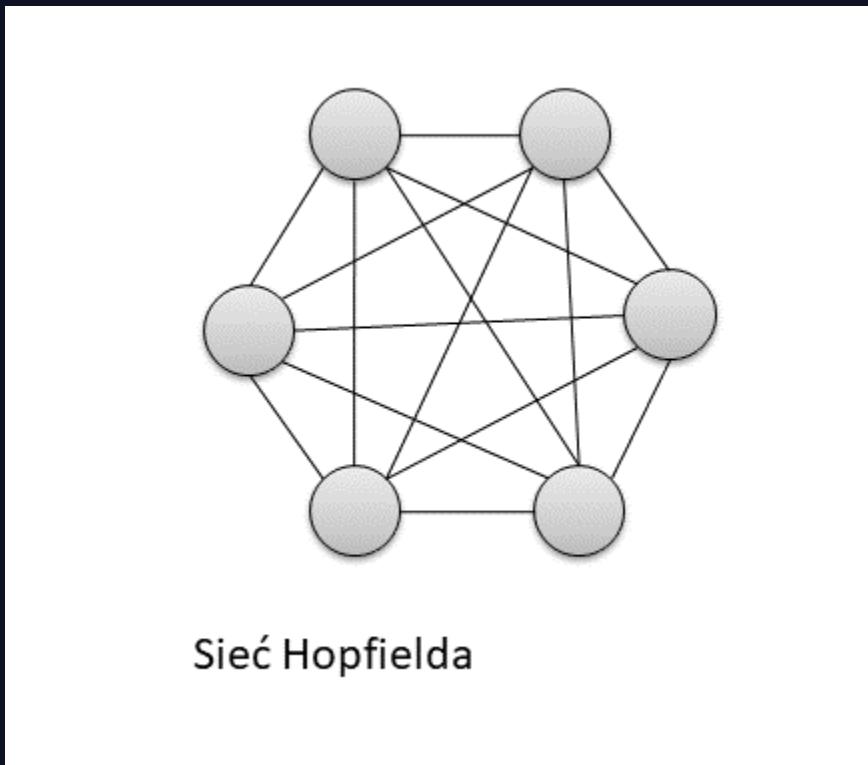
Sieci Hopfielda

- Celem sieci Hopfielda jest przechowywanie 1 lub więcej wzorów i przywoływanie pełnych wzorów na podstawie częściowych danych wejściowych.
- Sieć składa się z jednej warstwy zawierającej jeden lub więcej w pełni połączonych neuronów rekurencyjnych.
- W sieci Hopfielda wszystkie węzły są dla siebie danymi wejściowymi oraz wyjściami.
- Sieci Hopfielda służą jako systemy pamięci adresowalnej treści z binarnymi węzłami progowymi.
- Sieć neuronowa Hopfielda jest szczególnym przypadkiem sieci neuronowej Little.

Sieci Hopfielda

- Sieć Hopfielda jest konfigurowalną macierzą wag, która służy do znajdowania lokalnego minimum (rozpoznawania wzorca).
- Model Hopfielda uwzględnia pamięć asocjacyjną poprzezłączenie wektorów pamięci i jest powszechnie używany do klasyfikacji wzorców.
- Przykład: Sieć uczona grafikami będzie w stanie rozpoznać te grafiki nawet po zaszumieniu ich, lub zniekształceniu.
- Sieci Hopfielda dążą do minimalizacji funkcji energii, więc nadają się do zadań optymalizacyjnych.

Schemat Sieci Hopfielda



Dyskretna i ciągła sieć Hopfielda

- Sieć Hopfielda, która działa w sposób dyskretny to taka sieć w której wzorce wejściowe i wyjściowe są dyskretnymi wektorami, które mogą przyjmować wartości albo binarne - 0,1 lub bipolarne - +1,-1. Sieć ma symetryczne wagi bez samopołączeń.
- W porównaniu z dyskretną siecią Hopfielda, sieć ciągła wykorzystuje czas jako zmienną ciągłą. Sieć Hopfileda jest również używana w problemach związanych z autoasocjacją i optymalizacją, takich jak problem komiwojażera.

Dwukierunkowa pamięć asocjacyjna (BAM)

- BAM (ang. Bidirectional Associative Memory) to nadzorowany model uczenia się sztucznej sieci neuronowej.
- Głównym celem BAM jest działanie jako pamięć, która może nauczyć się kojarzenia ze sobą wzorców. Służy do wyszukiwania wzoru jeśli jest on zaszumiony lub niekompletny.
- BAM jest hetero-asocjacyjny, co oznacza, że dany wzorzec może zwrócić inny wzorzec, który może mieć inny rozmiar.
- Pamięć skojarzeniowa dwukierunkowa odwzorowuje bipolarne wektory binarne

Dwukierunkowa pamięć asocjacyjna

- Pamięć ludzką można nazwać asocjacyjną. Wykorzystujemy łańcuch skojarzeń myślowych, aby przypomnieć sobie wydarzenia, osoby, przedmiot: kojarzymy twarze z imionami, smaki z zapachami itp.
- W celu powiązania jednego elementu z drugim, potrzebujemy rekurencyjnej sieci neuronowej zdolnej do akceptowania wzorca wejściowego w jednym zestawie neuronów i wytwarzania pokrewnego, ale innego wzorca wyjściowego w innym zestawie neuronów.

Sieci LSTM i GRU

Aby uniknąć problemów z gradientami stworzono sieci LSTM (ang. Long – Short Term Memory) i GRU (ang. Gated Recurrent Unit)

Sieć LSTM składa się z komórki, bramki wejściowej, bramki wyjściowej i bramki zapominania.

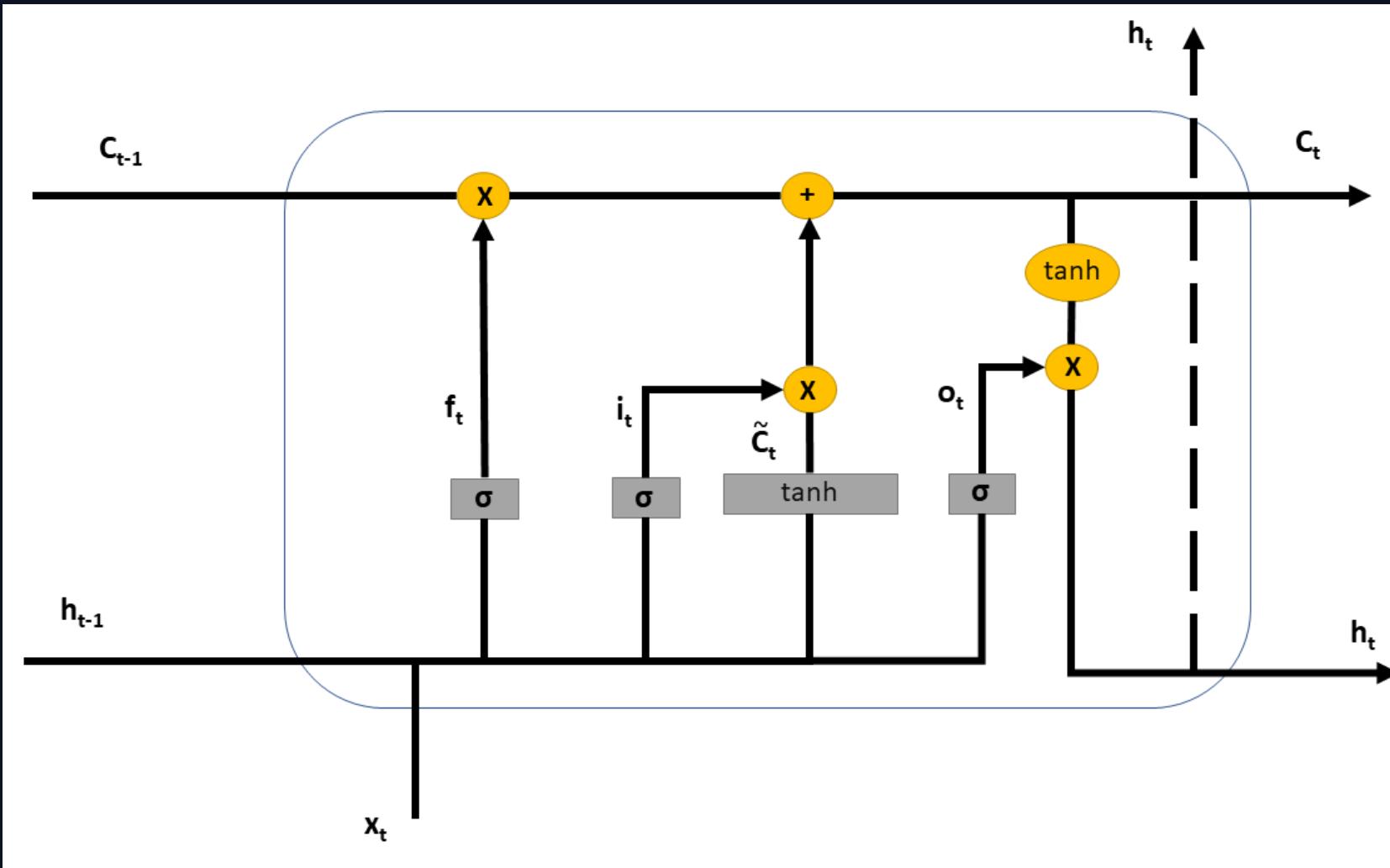
*Komórka pełni role pamięci i **przechowuje stan***

Bramka wejściowa reguluje jak dużo **nowych wartości** może znaleźć się w komórce (to co jest umieszczane i usuwane z pamięci) (co się pojawi w C^t)

Bramka zapominania odpowiada za kontrolę w jakim stopniu nowe wartości z bramki wejściowej **mogą pozostać w pamięci** (komórce) – podobnie jak bramka wejściowa (wpływ stanu C^{t-1} na C^t)

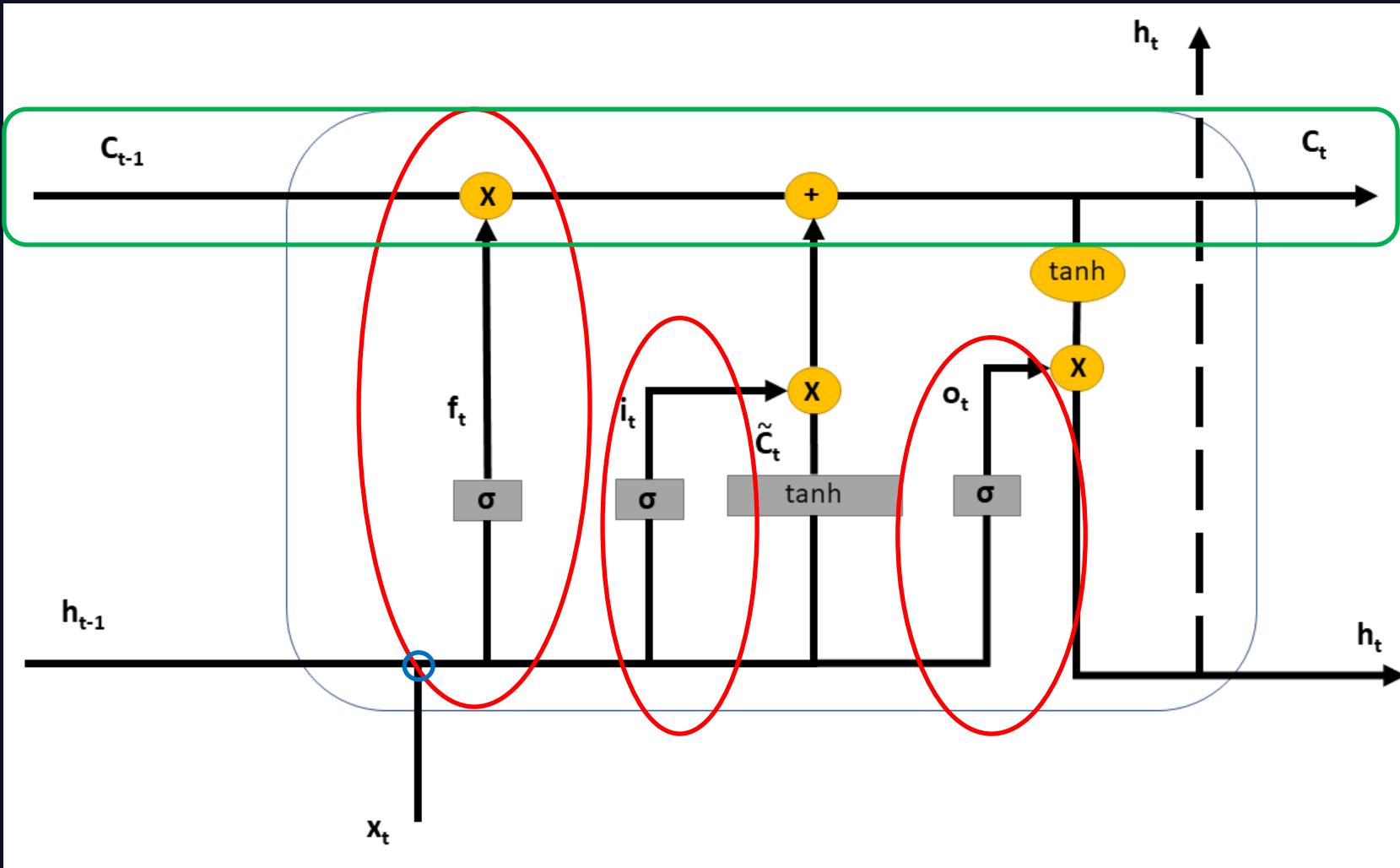
Bramka wyjściowa reguluje w jakim stopniu wartość w pamięci jest **wykorzystywana do obliczenia wartości wyjściowej** aktywacji komórki LSTM.

Sieci LSTM i GRU



Struktura LSTM

Sieci LSTM i GRU



Struktura LSTM

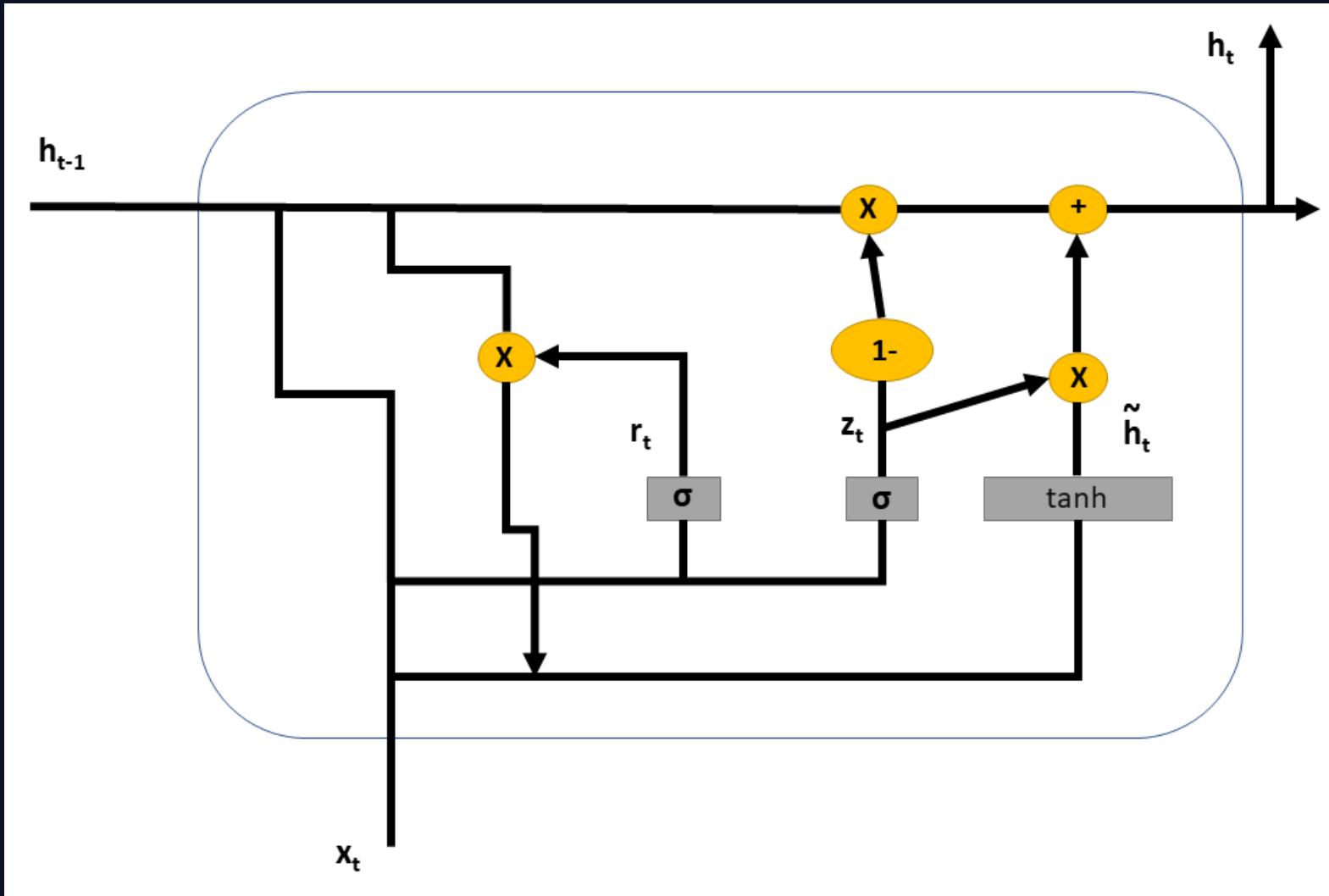
Sieci LSTM i GRU

Dzięki stosowaniu **bramek**, które kontrolują długookresowe zależności w sekwencji udaje się **uniknąć utraty** pewnych kluczowych informacji zawartych w sekwencji

GRU ma podobą konstrukcję do LSTM, różnica polega na braku bramki wyjściowej oraz pamięci C – GRU posiada *bramkę resetu* R_t i *bramkę aktualizacji* Z_t

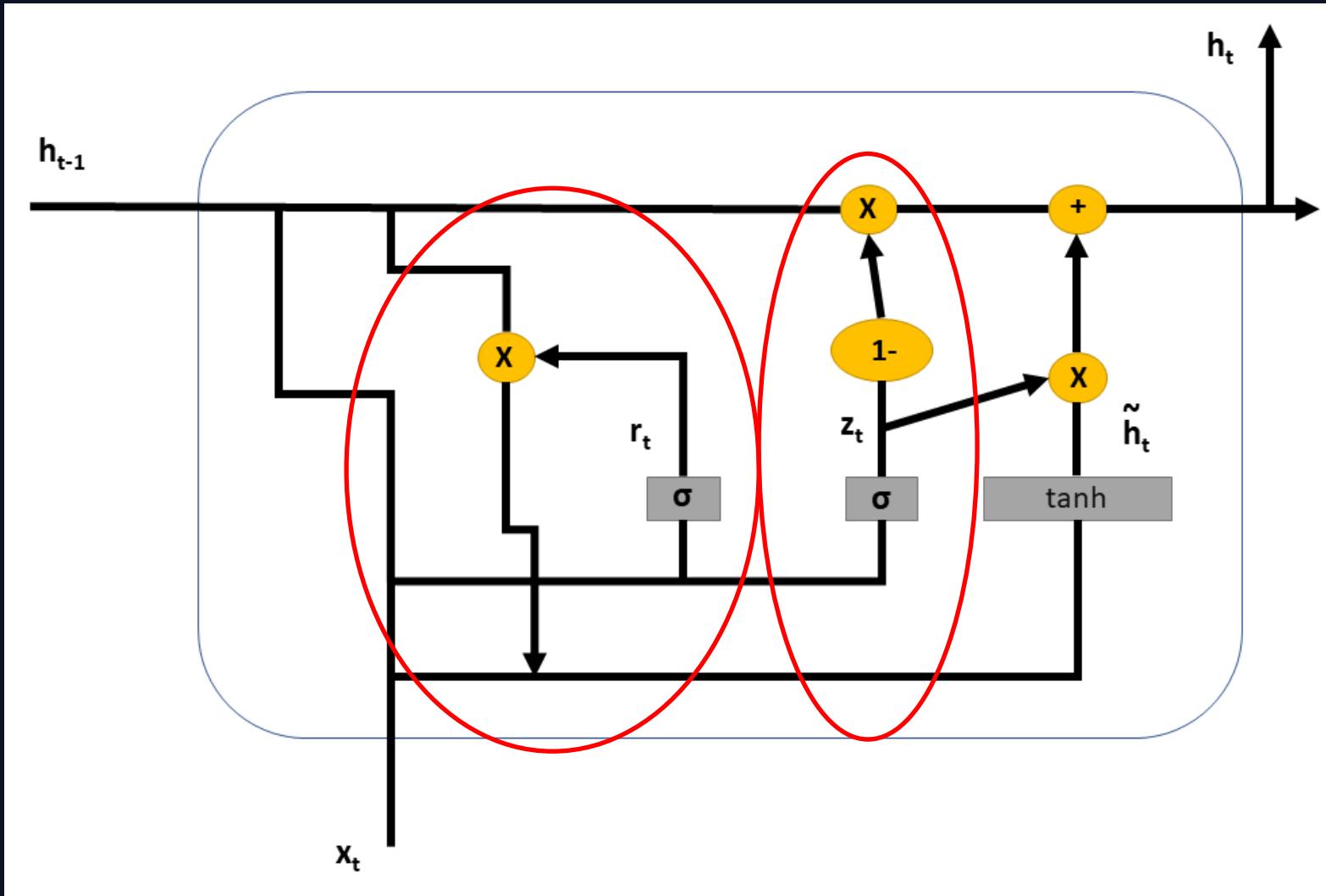
GRU posiada mniej parametrów względem LSTM przy porównywalnej jakości wyników w przypadku wielu zastosowań

Sieci LSTM i GRU



Struktura GRU

Sieci LSTM i GRU



Struktura GRU

Porównanie architektur RNN do predykcji szeregów czasowych w środowisku Google Colaboratory/ Jupyter Notebook

https://colab.research.google.com/drive/1BmJMY60-4_UblxpoBT4uX5p7i6hmqjPb#scrollTo=jdecyDDYyzRr

Dwukierunkowe RNN

- Dwukierunkowe RNN opierają się na założeniu, że dane wejściowe w czasie t mogą zależeć od poprzednich jak i przyszłych elementów sekwencji. Chcąc to zrealizować, wyjścia dwóch RNN muszą zostać zmieszane, jedno wykonuje proces w dodatnim kierunku czasu, a drugie uruchamia proces w ujemnym kierunku czasu.
- Architektura ta pozwala uzyskać warstwie wyjściowej informacje ze stanów przyszłych oraz stanów przeszłych.

Przykład dwukierunkowej RNN

- Uczenie sekwencyjne zakłada, że celem jest modelowanie następnego wyniku, biorąc pod uwagę to, co wiemy z poprzednich kroków (np. w kontekście szeregu czasowego lub modelu językowego).
- A jeżeli moglibyśmy skorzystać z przyszłych informacji?

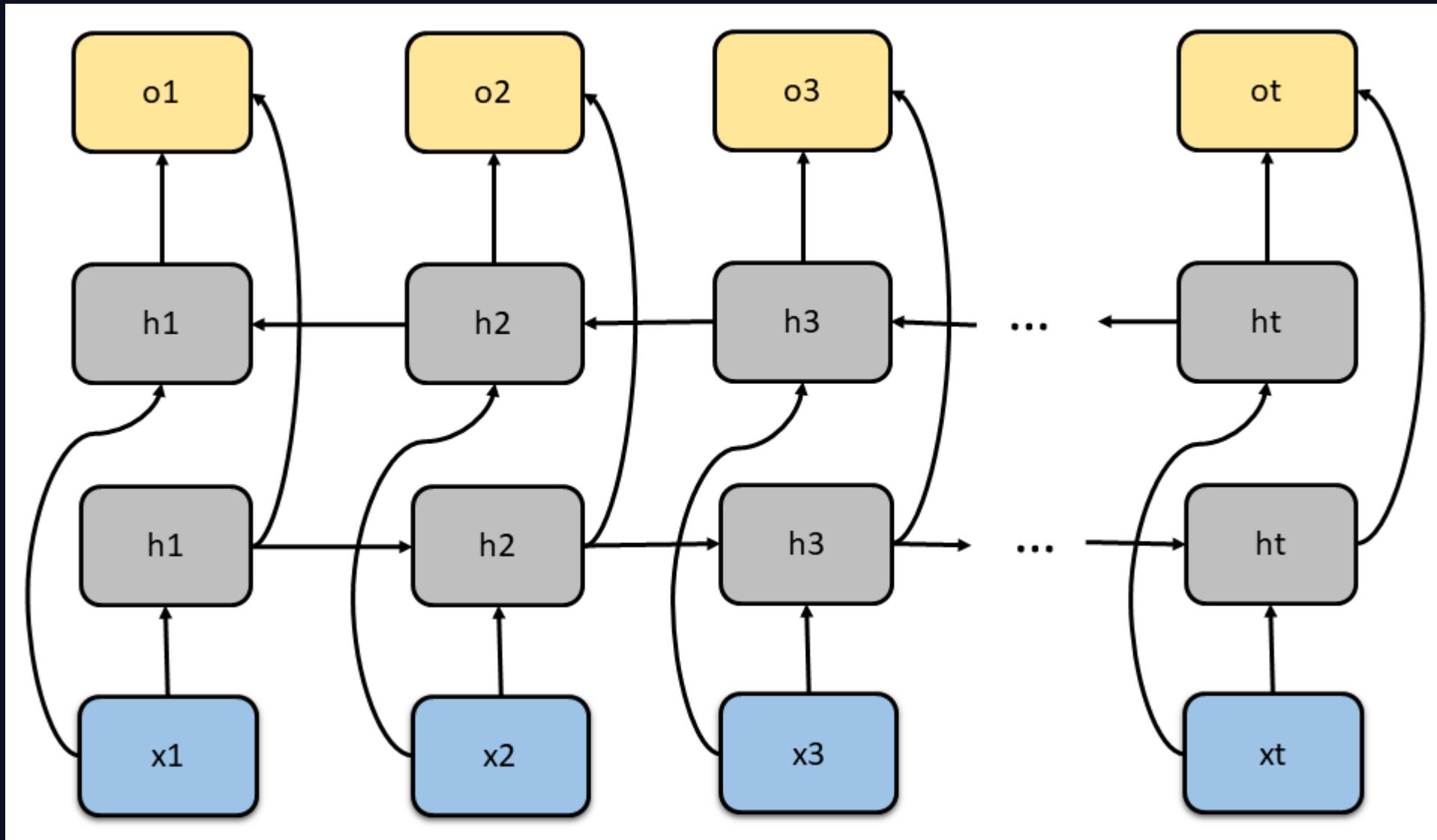
Mieszkam w _____

Mieszkam w _____ Lesie.

Mieszkam w _____ domu nad jeziorem.

- W zależności od ilości dostępnych informacji możemy wypełnić puste pola różnymi słowami, np. „Gdańsku”, „Stumilowym”, „pięknym”. Widać, że koniec frazy zawiera ważne informacje o tym, które słowo wybrać.

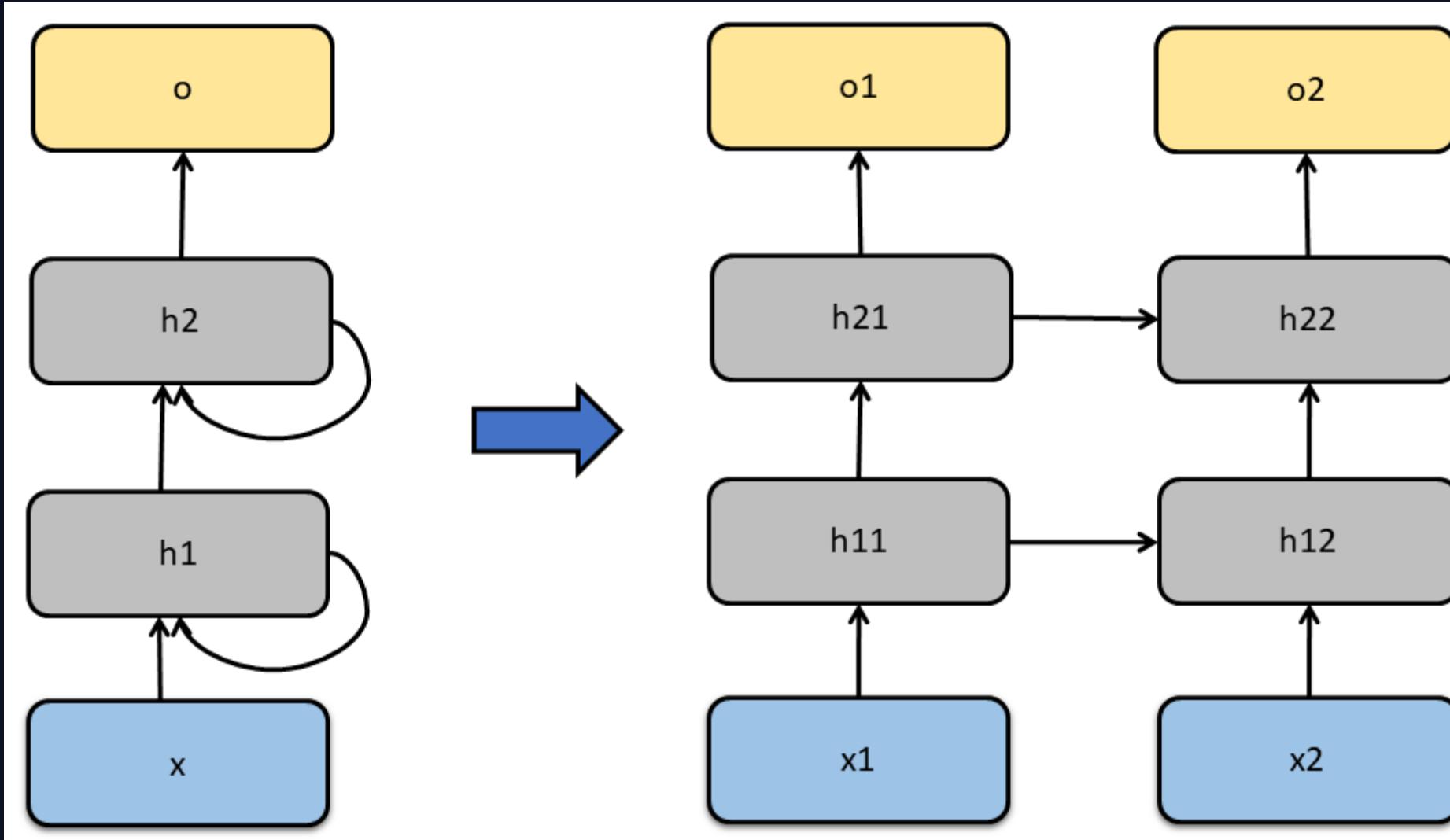
Dwukierunkowe RNN



Głębokie RNN

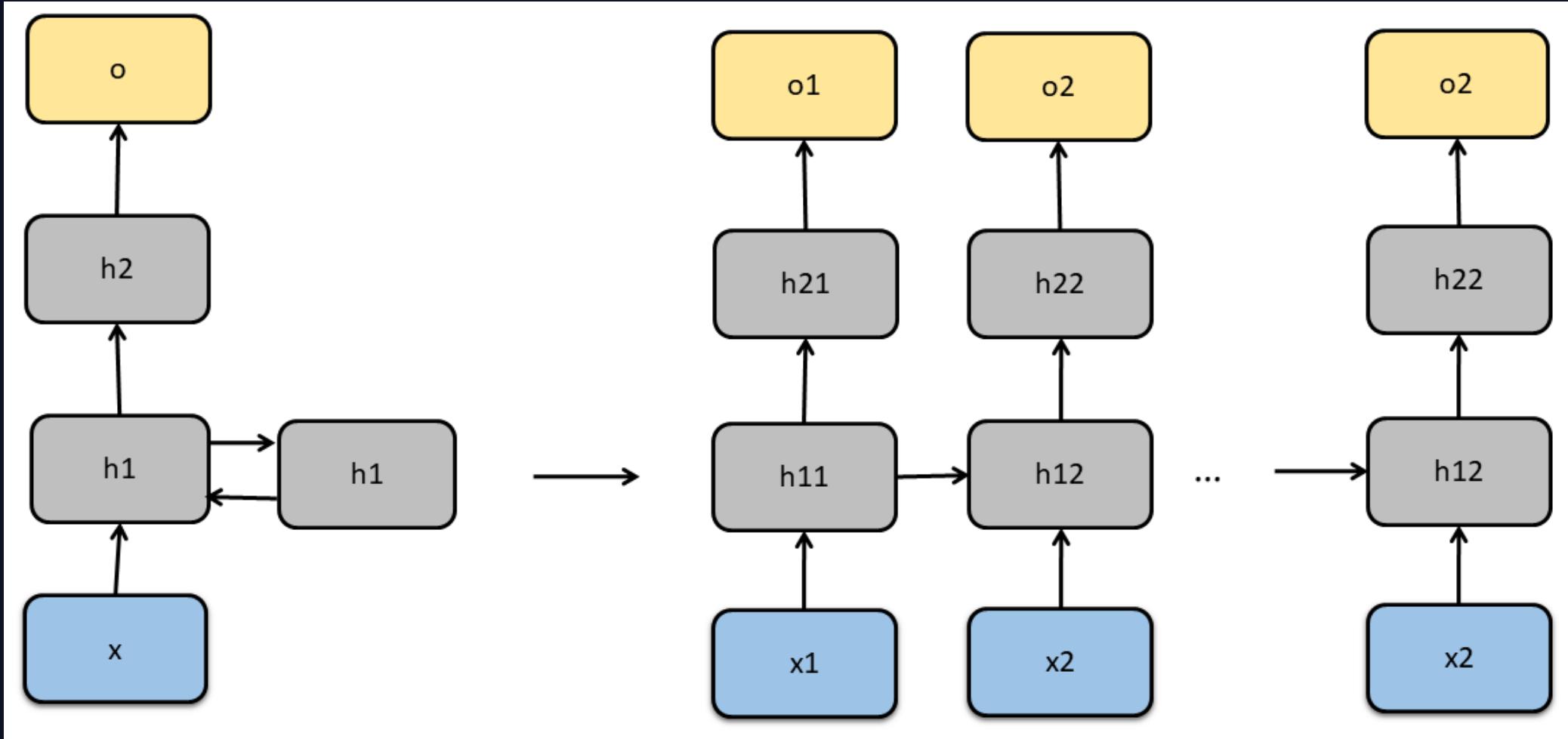
- Rekurencyjne RNN można rozszerzyć do głębokich RNN.
- Sposób na pogłębienie sieci może być wiele.
 - Ukryty stan może zostać rozdzielony na grupy zorganizowane w hierarchie.
 - Głębsze obliczenia można wprowadzić w różnych częściach.
 - Wprowadzając połączenia skokowe można osłabić efekt wydłużania ścieżki

2 warstwowa głęboka RNN



W typowej głębokiej sieci RNN operacja zapętlenia jest rozszerzona na wiele jednostek ukrytych.

Wielowarstwowa głęboka RNN



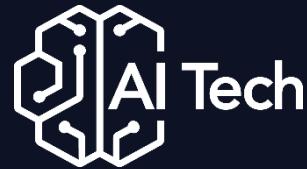
RNN można również zagłębić, wprowadzając głębię do ukrytej jednostki.

Głębokie RNN - podsumowanie

- W głębokich sieciach RNN informacje o stanie ukrytym są przekazywane do następnego kroku czasowego bieżącej warstwy i bieżącego kroku czasowego następnej warstwy.
- Istnieje wiele różnych odmian głębokich RNN, takich jak LSTM, GRU lub klasyczne RNN. Dogodnie wszystkie te modele są dostępne jako części wysokopoziomowych interfejsów API platform uczenia głębokiego.
- Inicjalizacja modeli wymaga staranności. Ogólnie rzecz biorąc, głębokie RNN wymagają znacznej ilości pracy (takiej jak szybkość uczenia się i przycinanie), aby zapewnić odpowiednią zbieżność.



POLITECHNIKA
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Dziękuję

SZYMON ZAPOROWSKI



Fundusze
Europejskie
Polska Cyfrowa



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.



POLITECHNIKA
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Uczenie głębokie: Modele RNN – zastosowanie - nie tylko NLP

Szymon Zaporowski



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan prezentacji

- Architektura koder-dekoder
- Automatyczna detekcja mowy jako przykład zastosowania nowoczesnych sieci RNN
- Wstęp do NLP – tokenizacja z wykorzystaniem najpopularniejszych algorytmów na przykładach
- Mechanizm atencyjny w sieciach RNN

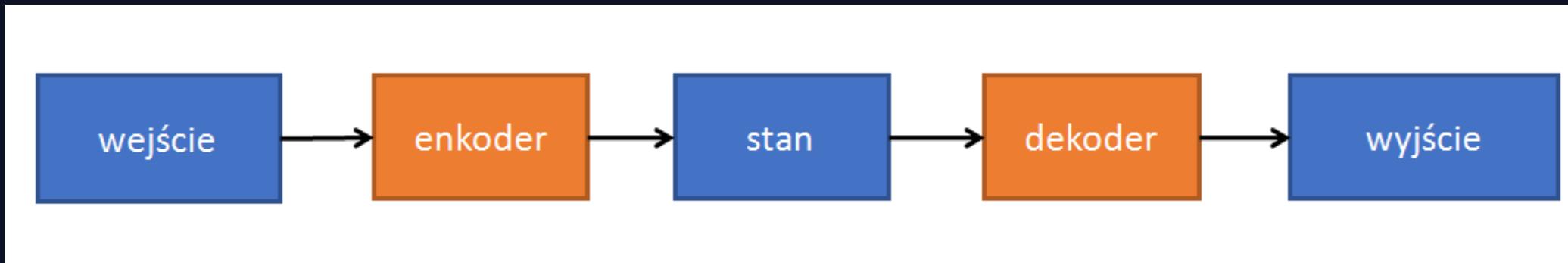
Architektura koder- dekoder

- Architektura koder-dekoder jest stosunkowo nowa i została przyjęta jako podstawowa technologia w usłudze tłumaczeń Google pod koniec 2016 roku.
- Stanowi podstawę dla zaawansowanych modeli sekwencja-sekwencja, takich jak modele Atencji, Modele GTP , Transformery i BERT.
- Jest rozwiązaniem dla przypadków, gdzie dane wejściowe i wektory nie mają stałej wymiarowości.
- Mogą być używane do mapowania sekwencji na sekwencję.
- Sequence-to-Sequence (Seq2Seq) to specjalna klasa problemów modelowania sekwencji, w której zarówno wejście, jak i wyjście są sekwencją.

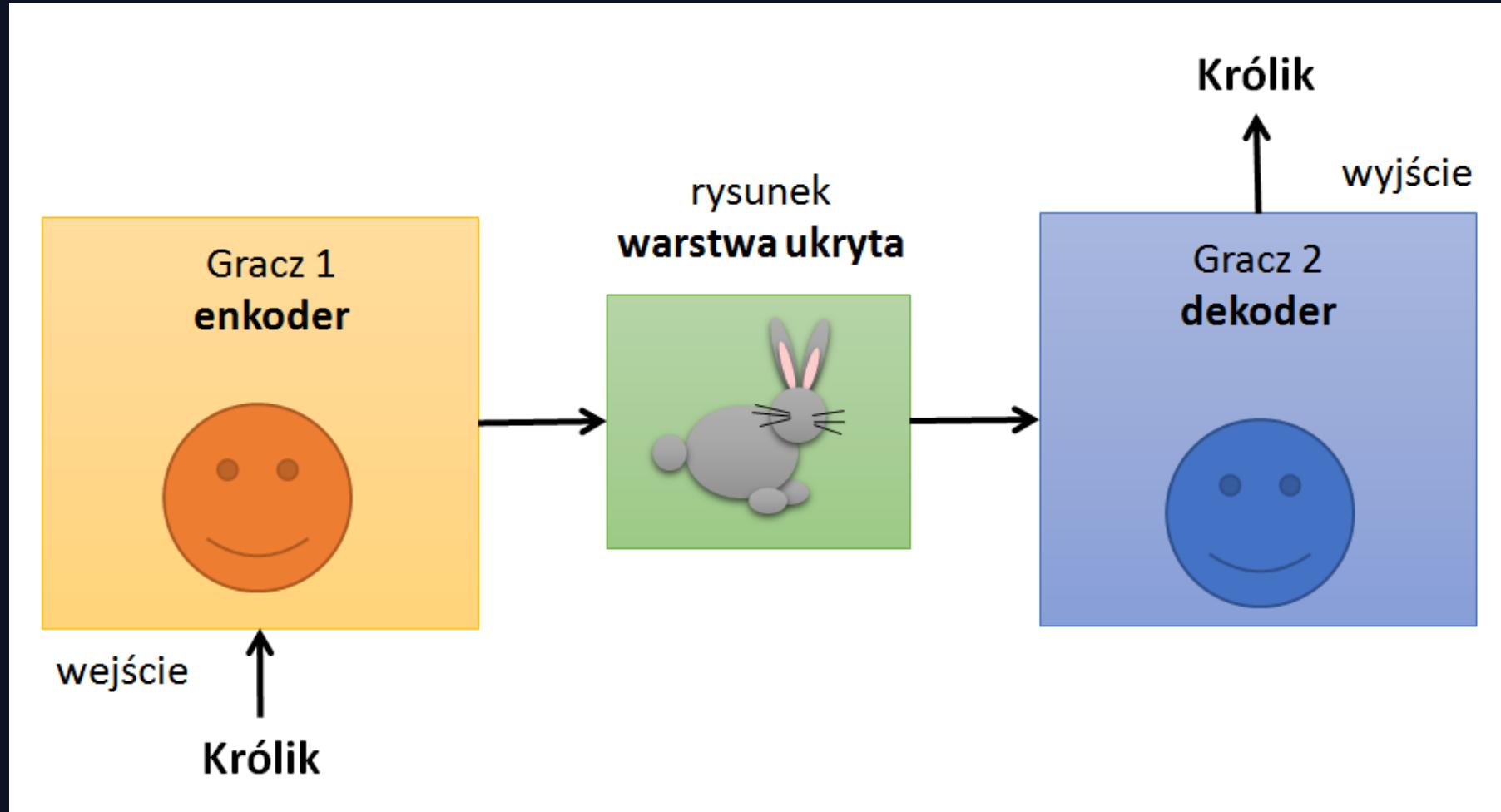
Architektura koder- dekoder

Architektura ta składa się z dwóch głównych elementów:

- **Enkoder (koder)** – jako wejście przyjmuje sekwencję o zmiennej długości i przekształca ją w stan o ustalonym kształcie
- **Dekoder** – odwzorowuje zakodowany stan o ustalonym kształcie na sekwencję o zmiennej długości.



Enkoder – dekoder – kalambury?



Enkoder i stan ukryty

- Kodowanie oznacza konwersję danych do wymaganego formatu. W przypadku gry kalambury konwertujemy słowo (tekst) na rysunek (obraz).
- W kontekście uczenia maszynowego konwertujemy sekwencję słów w na dwuwymiarowy wektor, ten dwuwymiarowy wektor jest również znany jako stan ukryty.
- Koder jest zbudowany przez układanie w stos rekurencyjnej sieci neuronowej (RNN) . Używamy tego typu warstwy, ponieważ jej struktura pozwala modelowi zrozumieć kontekst i czasowe zależności sekwencji. Dane wyjściowe kodera, stan ukryty, to stan ostatniego kroku czasowego RNN.

Stan ukryty i dekoder

- Wyjście enkodera to dwuwymiarowy wektor, który zawiera całe znaczenie sekwencji wejściowej. Długość wektora zależy od liczby komórek w RNN.
- Dekoder ma za zadanie przekonwertować zakodowaną wiadomość na zrozumiały język. W przypadku naszej gry będzie to zamiana rysunku na słowo.
- W modelu uczenia maszynowego rolą dekodera będzie przekształcenie dwuwymiarowego wektora na sekwencję wyjściową, czyli zdanie (np. w wybranym języku).

Architektura koder – dekoder: zastosowanie

- Automatyczne podpisy pod obrazami/filmami. Otrzymuje obraz jako wejście, a sekwencja słów pojawia się na wyjściu.
- Analiza sentymentu. Ta metoda wykorzystywana jest w call center do analizy ewolucji emocji klienta i jego reakcji na określone słowa kluczowe.
- Tłumaczenie na inny język. Np. tłumacz Google, model ten odczytuje zdanie wejściowe, rozumie przesłanie i pojęcia, a następnie tłumaczy je na drugi język.

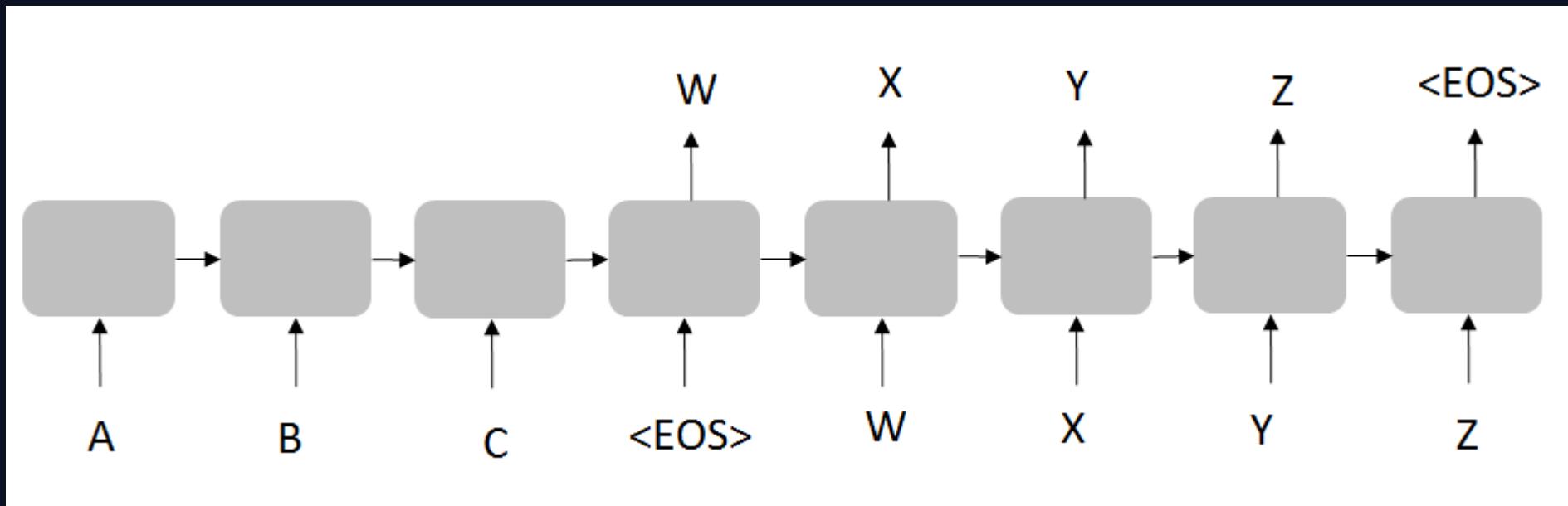
Model Sutskever NMT

- Model neuronowego tłumaczenia maszynowego opracowany przez Ilyę Sutskever i jej współpracowników, opisany w artykule z 2014 r., był jednym z pierwszych artykułów, który wprowadził model enkoder-dekoder.
- Jest to jeden z pierwszych neuronowych systemów tłumaczenia maszynowego, który przewyższał podstawowy model statystycznego uczenia maszynowego w przypadku dużego zadania tłumaczeniowego.
- Model był zastosowany do tłumaczenia zdań z języka angielskiego na francuski.
- Zadaniem było przetłumaczenie jednego zdania na raz. Token końca sekwencji (<EOS>) został dodany do końca sekwencji wyjściowych podczas uczenia, aby oznaczyć koniec przetłumaczonej sekwencji. Umożliwiło to przewidywanie sekwencji wyjściowej o zmiennej długości.

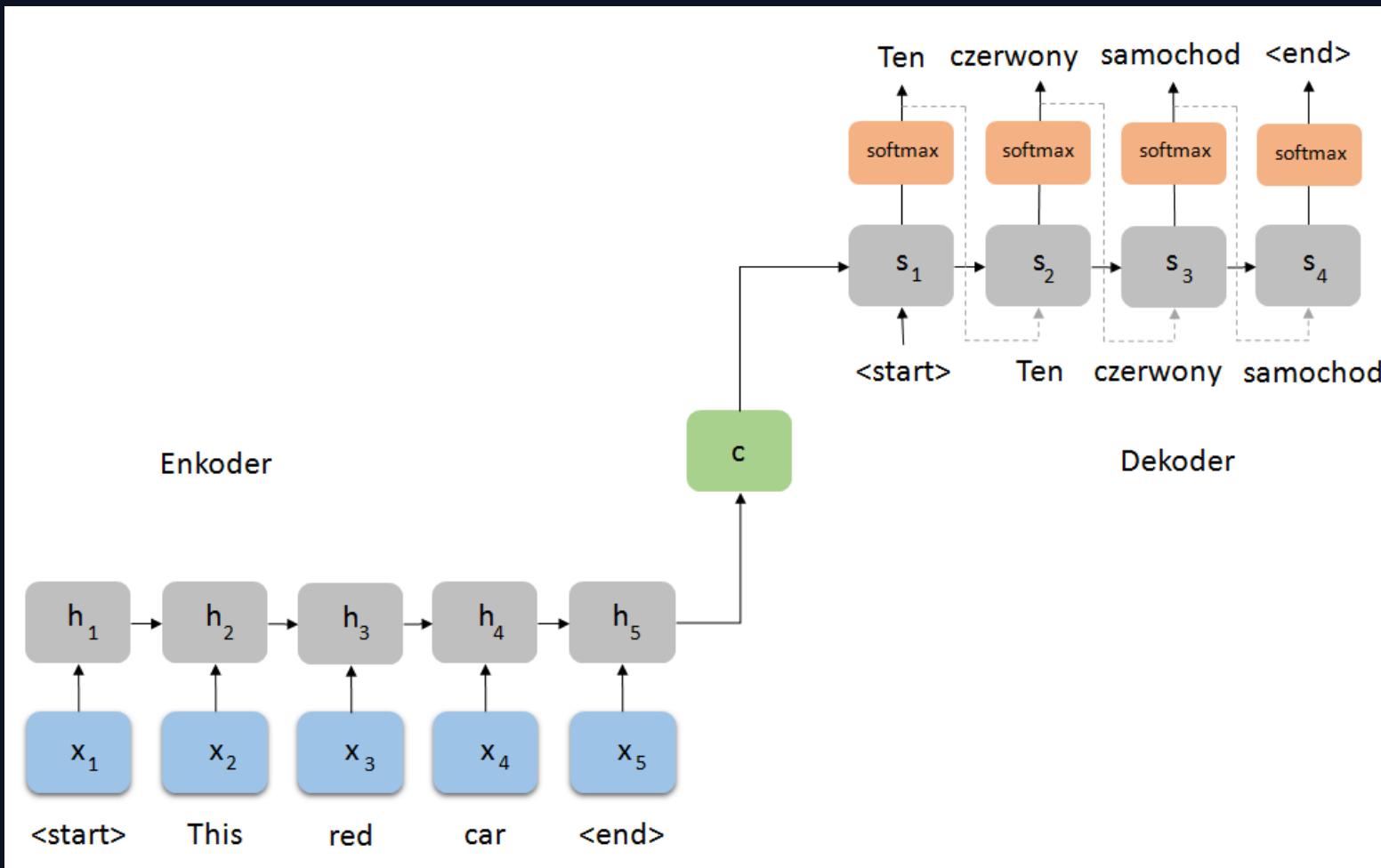
Model Sutskever NMT

- Model został wyszkolony na podzbiorze 12 milionów zdań w zbiorze danych, składającym się z 348 milionów słów francuskich i 304 milionów słów angielskich.
- Słownictwo źródłowe zostało zredukowane do 160 000 najczęstszych źródłowych angielskich słów i 80 000 najczęstszych docelowych francuskich słów. Wszystkie słowa poza słownictwem zostały zastąpione skrótem „UNK”.
- Opracowano architekturę kodera-dekodera, w której sekwencja wejściowa została w całości odczytana i zakodowana do wewnętrznej reprezentacji o stałej długości.
- Następnie sieć dekodera wykorzystywała tę wewnętrzną reprezentację do wyprowadzania słów, aż do osiągnięcia znacznika końca sekwencji. Sieci LSTM zostały wykorzystane zarówno dla kodera, jak i dekodera.

Model Sutskever NMT



Enkoder – dekoder – tłumaczenie maszynowe



Przetwarzanie sekwencji w modelu koder-dekoder

Rozumienie tekstu może być traktowane jako proces iteracyjny dla człowieka – podobnie w przypadku sieci rekurencyjnej.

Istnieje kilka architektur w przypadku modeli koder-dekoder w RNN:

- Wiele do jednego
- Jeden do wielu
- Wiele do wielu

Koder-dekoder podsumowanie

- Główną zaletą tego modelu jest to, że długość sekwencji wejściowych i wyjściowych może się różnić.
- Ograniczeniem prostego modelu koder-dekoder jest to, że wszystkie informacje muszą być podsumowane w jednowymiarowym wektorze, w przypadku długich sekwencji wejściowych, które mogą być niezwykle trudne do uzyskania.
- W konsekwencji wydajność architektury spada wraz z długimi zdaniami, ponieważ ma tendencję do zapominania części, a ukryty wektor staje się wąskim gardłem.

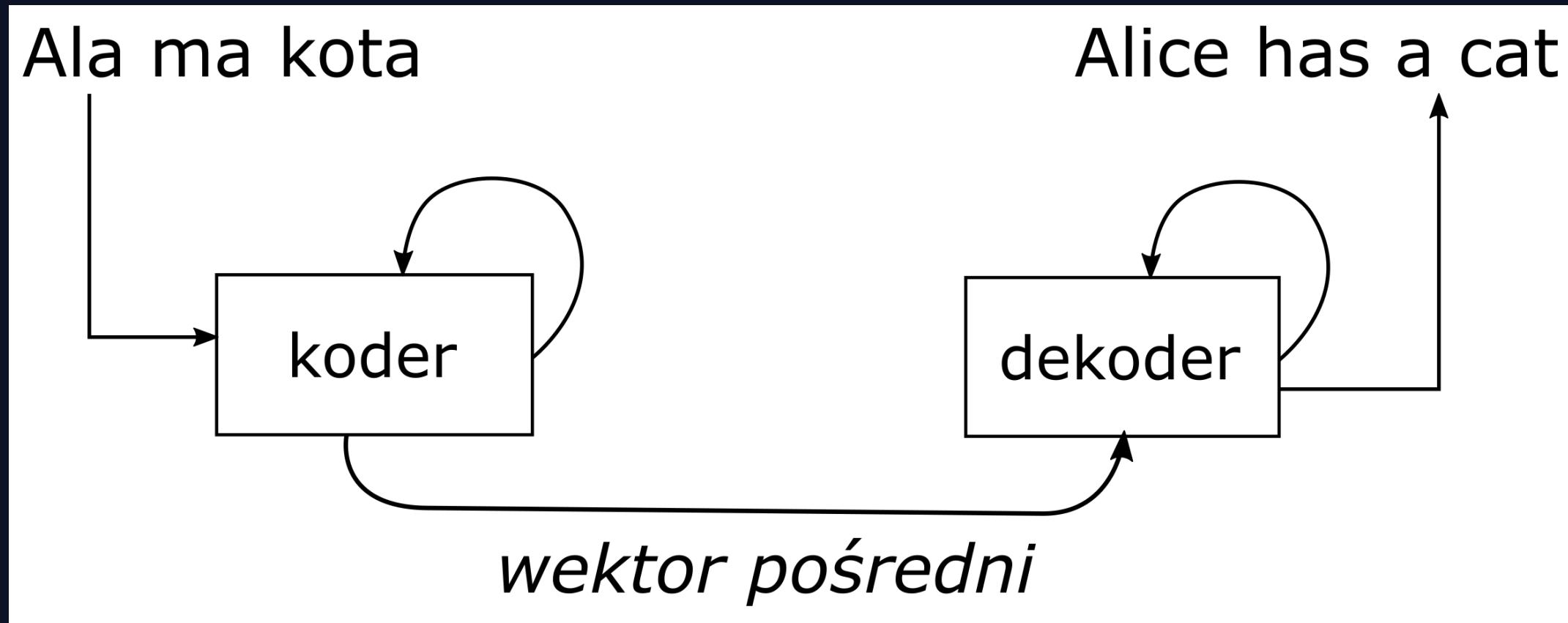
Sieci rekurencyjne z mechanizmem atencyjnym

Sieci LSTM i GRU umożliwiają poprawę wydajności RNN podczas używania dłuższych sekwencji, ale nie są idealnym rozwiązaniem np. w przypadku **translacji maszynowej**. W takim tłumaczeniu często może się zdarzyć, że konieczne jest skupienie się na **określonych słowach** z sekwencji wejściowej, które **nie odpowiadają kolejności** w sekwencji wyjściowej.

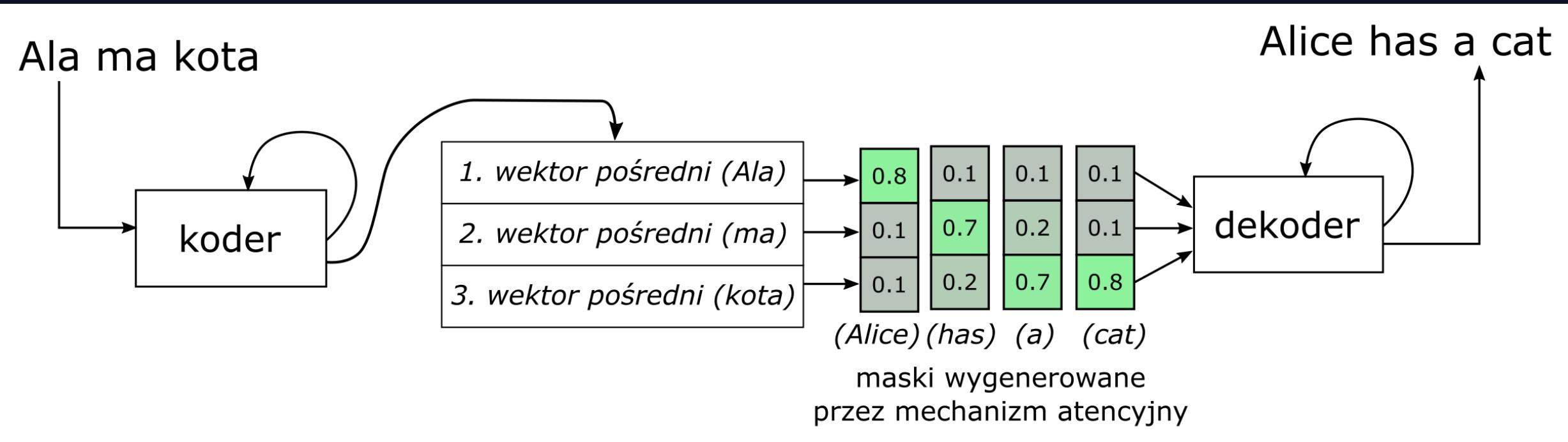
Odpowiedzią na takie problemy jest **mechanizm atencyjny**.

Wykorzystanie mechanizmu agencyjnego pozwala, aby **dekoder** skupiał się na **określonych częściach** sekwencji **wejściowej** – pozawala to prawidłowo wytworzyć **kolejną sekwencję** wyjściową

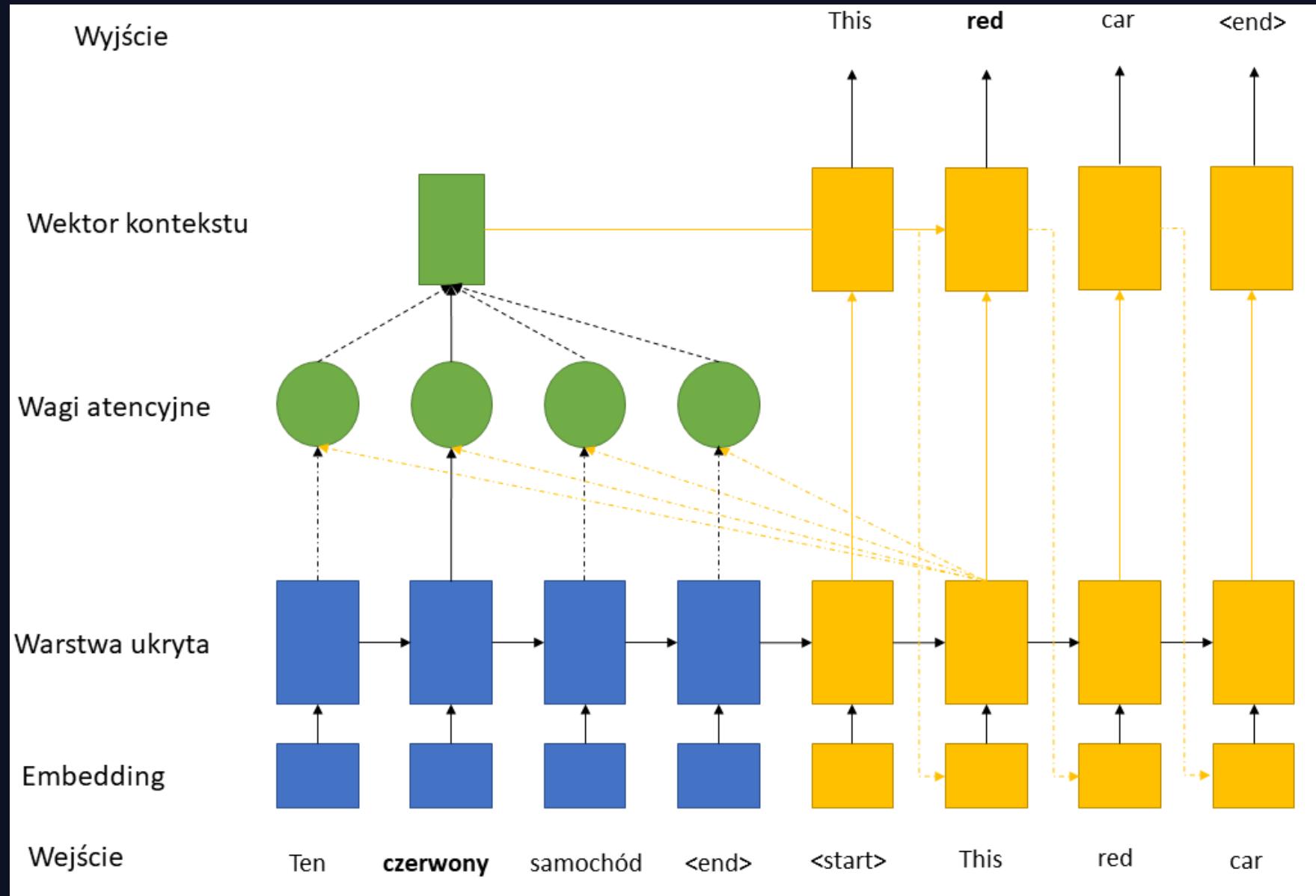
Przetwarzanie sekwencji w modelu koder-dekoder



Sieci rekurencyjne z mechanizmem atencyjnym



Sieci rekurencyjne z mechanizmem atencyjnym



Przykłady praktyczne (notatnik interaktywny
DL_09_02_Video_classification.ipynb)

Implementacja klasyfikacji obrazu
video z użyciem sieci RNN
w środowisku
Google Colaboratory/
Jupyter Notebook

https://colab.research.google.com/drive/1Z8LjdJJny3voT8Q6zcKrKil_uuvq8Y7X#scrollTo=l8TdICwXORNc

Automatyczne rozpoznawanie mowy z użyciem sieci RNN

Istnieje kilka możliwych podejść do rozpoznawania mowy z użyciem sieci rekurencyjnych

Obecnie najpopularniejsze rozwiązania to:

- architektura enkoder-dekoder (Listen, Attend and Spell)
- Hybrydowe zastosowanie sieci splotowych do ekstrakcji parametrów oraz sieci rekurencyjnych wraz z algorytmem Connectionist Temporal Classification

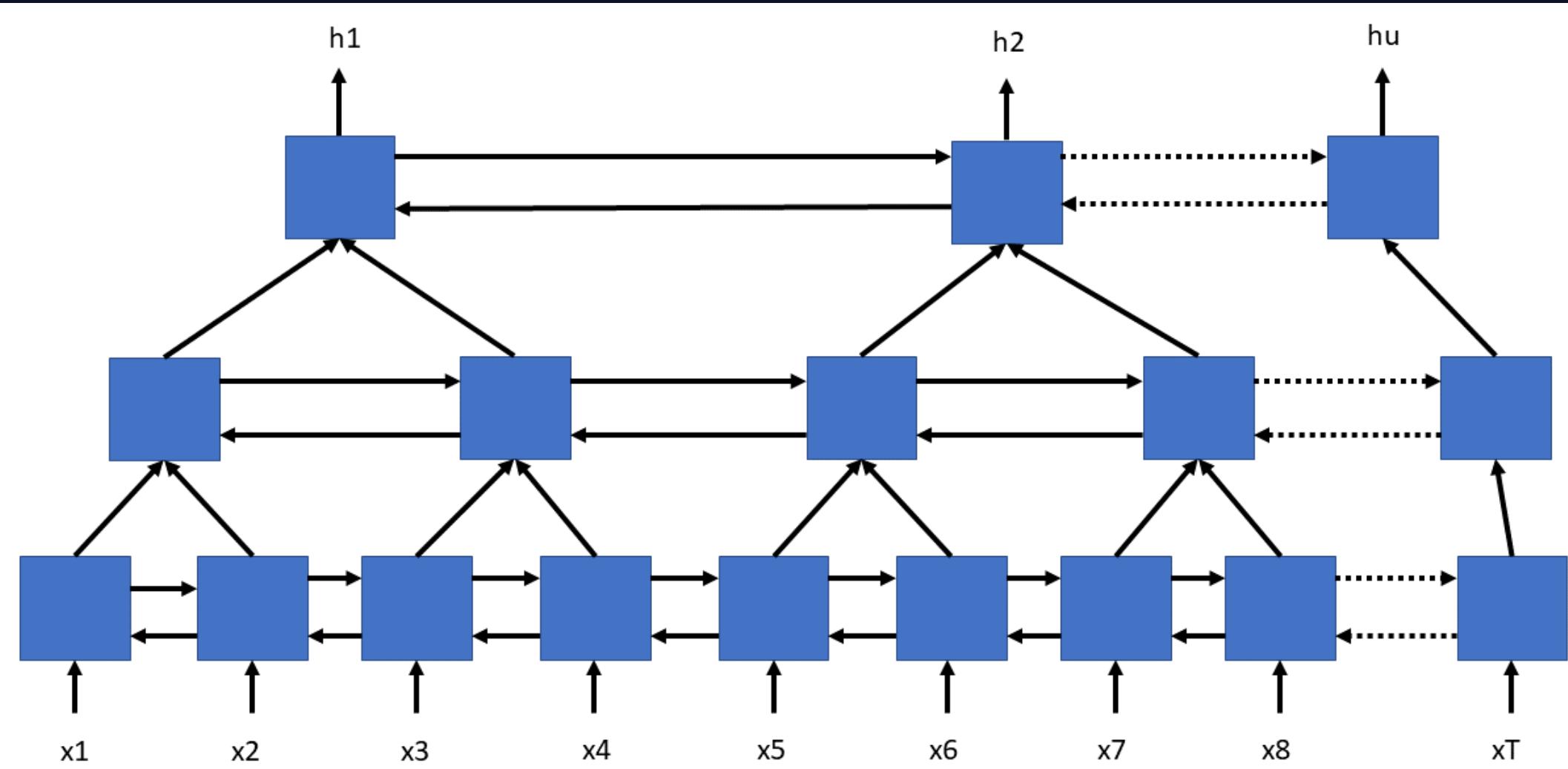
Automatyczne rozpoznawanie mowy z użyciem sieci RNN – enkoder - dekoder

Sieć typu Listen, Attend and Spell składa się z enkodera zwanego Listenerem (na dole sieci) oraz dekodera zwanego Spellerem na górze sieci.

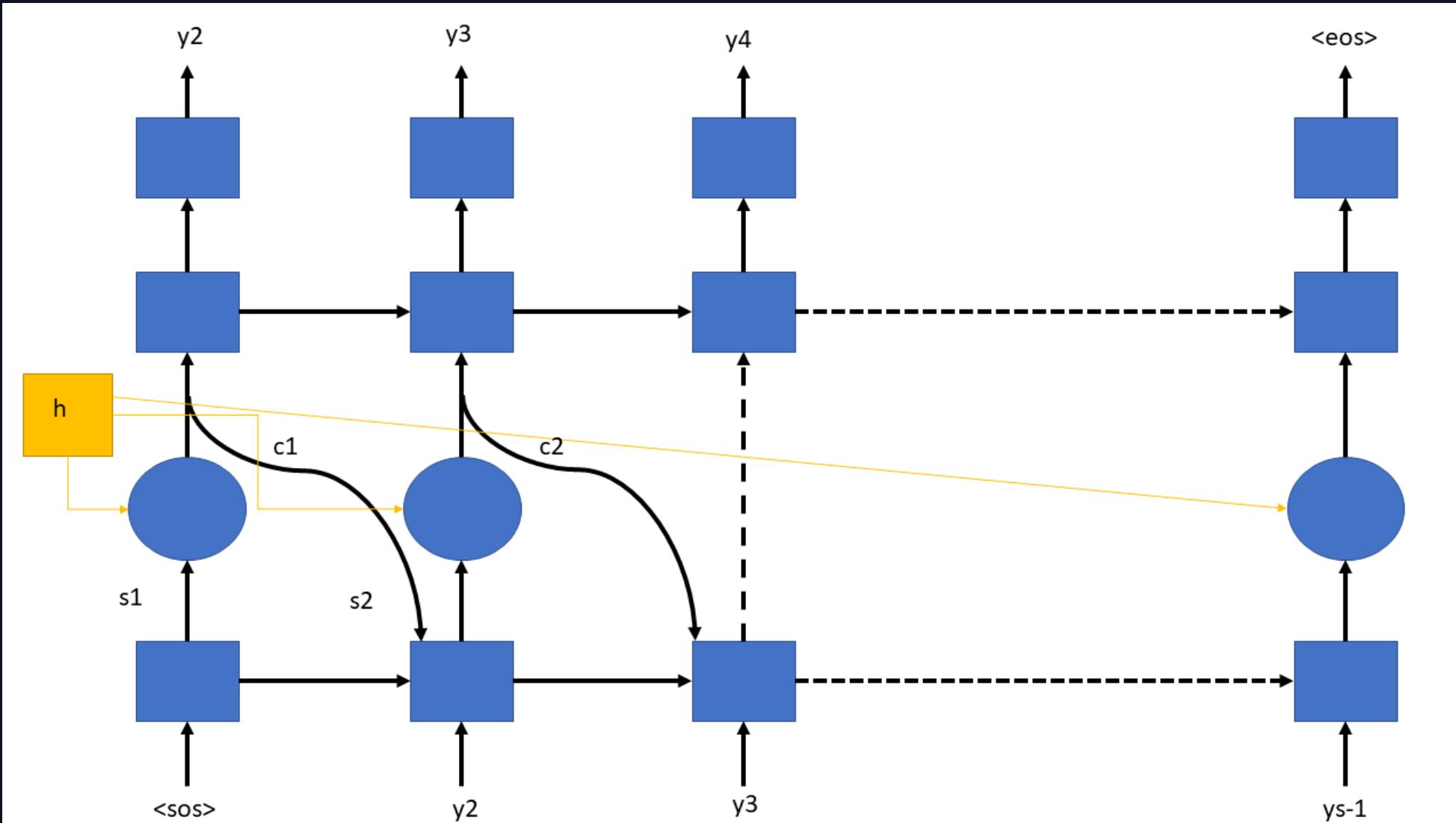
Zadaniem enkodera jest przetworzenie reprezentacji sygnału audio i enkodowanie jej do postaci „gęstszej”

Dekoder dokonuje przetworzenie tej postaci do dystrybucji prawdopodobieństwa znaków w każdym kroku dekodowania

Enkoder w ASR z użyciem RNN



Dekoder w ASR z użyciem RNN



Podejście typu LAS

- Stan dekodera zależy od wcześniejszego stanu dekodera, wcześniej wskazanego znaku i wcześniejszego kontekstu obliczonych w ostatnim kroku czasowym

$$s_i = RNN(s_{i-1}, y_{i-1}, c_{i-1})$$

Gdzie:

s_{i-1} - to wcześniejszy stan dekodera

y_{i-1} - to wcześniej wskazany znak

c_{i-1} - to wcześniejszy kontekst

Podejście typu LAS

Następnie obliczamy wyniki atencji wykorzystując stan dekodera oraz stan enkodera (h). Z użyciem MLP przetwarzamy dalej obecny stan dekodera s oraz wartość h

Następnie należy znaleźć podobieństwo między nimi poprzez wykorzystanie iloczynu skalarnego. Jest to wynik atencji dla poszczególnych elementów w wektorze h .

Obliczany jest softmax i jego wynik jest używany do obliczenia kontekstu c .

Wartość kontekstu c jest to ważone wyjście stanu enkodera h bazujący na wyniku atencji.

Podejście typu LAS

$$e_{i,u} = \langle \phi(s_i), \psi(h_u) \rangle \quad \alpha_{i,u} = \frac{\exp(e_{i,u})}{\sum_u \exp(e_{i,u})} \quad c_i = \sum_u \alpha_{i,u} h_u$$

Na końcu, dystrybucja znaków jest obliczana ze stanu dekodera i kontekstu wynikającego z atencji z użyciem MLP

$$P(y_i|x, y_{<i}) = \text{DystrybucjaZnaków}(s_i, c_i)$$

Podejście typu LAS

- Podejście typu LAS tworzy model który maksymalizuje prawdopodobieństwo ground truth.
- Podczas inferencji nie korzysta się z ground truth, w związku z tym jakość predykcji modelu może ucierpieć jeśli poprzedni stan, który został użyty do obliczenia stanu dekodera okazał się być złą predykcją

Algorytm CTC

Connectionist Temporal Classification jest to algorytm wykorzystywany do wyrównywania sekwencji wejściowych i wyjściowych, gdy wejście jest ciągłe, a wyjście dyskretne i nie istnieją jasne granice elementów, które mogą posłużyć do mapowanie wejścia na elementy sekwencji wyjściowej

Założmy, że chcemy stworzyć system ASR w oparciu o podejście hybrydowe – sieć splotową do ekstrakcji cech i sieć RNN do predykcji

Posiadamy przetworzone przez sieć typu CNN spektrogramy na mapy cech, które są podzielona na oddzielne ramki i są wejściem na sieć RNN

Dla każdej ramki sieć RNN wraz z liniowym klasyfikatorem musi przewidzieć prawdopodobieństwo wystąpienia każdego znaku ze słownika

Algorytm CTC

- Zadaniem algorytmu CTC jest przetworzenie prawdopodobieństw wystąpienia znaków i dostarczenie poprawnej sekwencji znaków.
- Dodatkowo wprowadzona zostaje koncepcja pustego znaku – oznaczonego jako „-” w słowniku. Dzięki temu prawdopodobieństwo pojawienia się znaku uwzględnia również brak pojawienia się znaku czyli granicę między dwoma znakami – jest odpowiednik „null”
- CTC może być wykorzystywane jako funkcja straty podczas treningu:
- W takim przypadku posiadamy ground truth docelowej transkrypcji i staramy się wytrenować sieć aby zmaksymalizować prawdopodobieństwo otrzymania poprawnej sekwencji

Funkcja straty CTC

- Jest to prawdopodobieństwo predykcji poprawnej sekwencji przez sieć. Algorytm tworzy listę wszystkich możliwych sekwencji i z tego zestawu danych wybiera podzestawy, które pasują do przewidywanej transkrypcji
- Lista zostaje zawężona z użyciem dwóch głównych reguł:
- Pozostają tylko prawdopodobieństwa znaków, które występują w transkrypcji np. „T”, „a”, „k”, “-”: dla słowa „Tak”
- Filtracja podzbiorów znaków dla każdej ramki i wybór takich znaków, gdzie występują one w odpowiedniej kolejności np. „T” i „a” są poprawnymi znakami, „Ta” jest poprawną sekwencją, ale „At” już nie.

Funkcja straty CTC

- Następnie wykorzystywane są prawdopodobieństwa znaków, dla każdej ramki, uwzględniając „puste” znaki np. „-T-a-k”
- Kolejnym krokiem jest złączenie wszystkich znaków, które się powtarzają, ale nie są rozdzielone znakiem „-”
- Przykładowo w angielskim słowie „Food” i wystąpieniu sekwencji ”oo” można ją złączyć do pojedynczego „o”, ale sekwencji „o-oo” nie można już złączyć. W ten sposób algorytm potrafi rozróżnić powtórzenie tych samych znaków w wyrazie
- Na koniec usuwane są „puste” znaki i otrzymuje się finalne słowo „Tak”

Zastosowania algorytmów przetwarzających sekwencje

Predykcja sekwencji – zadana sekwencja na wejście – wyjście to wartość na bazie zadanej sekwencji (np. nazwy kolejnych miesięcy)

Klasyfikacja sekwencji - zadana sekwencja na wejście – etykieta klasy na wyjściu na bazie zadanej sekwencji (np. do jakiego gatunku literackiego należy wprowadzony tekst)

Generowanie sekwencji - zadana sekwencja na wejście - wyjście to inna sekwencja bazująca na zadanej (np. generowanie tekstu, generowanie muzyki)

Predykcja sekwencja do sekwencji (seq2seq) - zadana sekwencja na wejście – wyjście to inna sekwencja (np. tłumaczenie z angielskiego na francuski, generowanie podsumowania tekstu)

Tokenizacja danych tekstowych

Najczęściej występujące algorytmy tokenizacji w NLP:

- Bag of words (BOW)
- Skip-gram
- Word-2-vec
- Byte Pair Encoding (BPE)

Tokenizacja danych tekstowych - Bag of words

- Bag of words jest techniką przetwarzania języka naturalnego wykorzystywaną do modelowania tekstu. Jest to metoda wydobywania/ekstrakcji cech z danych tekstowych.
- Metoda ta sprawdza częstotliwość występowania słów w tekście, natomiast pomija szczegóły gramatyczne i kolejność występowania wyrazów.
- BOW konwertuje tekst na równoważny wektor liczb, dzięki czemu model uczenia maszynowego działa na liczbach, a nie słowach.
- Teksty o różnych długościach mogą być konwertowane na wektory o stałych długościach.

Bag of words bez preprocessingu i z preprocessingiem

- Bez preprocessingu model bierze wszystkie słowa, znaki interpunkcyjne, cyfry itp.
- Wektor zdania może być bardzo długi, ponieważ bez preprocessingu będzie miał długość równą liczbie znanych słów w słowniku.
- Powoduje to powstanie wektora zwanego wektorem rzadkim, czyli wektora z wieloma punktami zerowymi. Skutkuje to potrzebą większej pamięci i większej liczby zasobów obliczeniowych podczas modelowania.
- Preprocessing umożliwia skrócenie wektora. Na wstępnym etapie usuwane są słowa, które zawierają w sobie mało informacji, pomijane są znaki interpunkcyjne. Następnie mogą być poprawiane błędnie napisane słowa i wykorzystywane są algorytmy rdzeniowe do redukcji słów do ich rdzenia.

Tokenizacja danych tekstowych - Bag of words - kroki

1. Zebranie danych.
2. Ujednolicenie wielkości liter.
3. Usunięcie słów o „małym znaczeniu” (np. „to”, „gdzieś”, „są”, znaki interpunkcyjne).
4. Zmiana niektórych słów na rdzenie słów (np. domowy -> dom, domku -> dom).
5. Zmiana skrótów na pełne formy (np. -> na przykład, z ang. I've -> I have .
6. Zliczanie częstotliwości występowania słów.
7. Zamiana zdań na wektory.

Tokenizacja danych tekstowych - Bag of words - wady

- Model nie zapisuje informacji o lokalizacji słów, dane te są tracone, przez co kontekst zdań może być mylnie odczytany.
- BOW nie respektuje semantyki słowa. Słowa o podobnym znaczeniu użyte w tym samym zdaniu będą oznaczone całkiem innymi wektorami, a słowa, które mają kilka znaczeń np. zamek (budowla, błyskawiczny w kurtce) użyte w pytaniu „A co to za zamek?” będą oznaczone tym samym wektorami, chociaż mają różne znaczenia w zależności jaki zamek autor miał na myśli.
- Nowe słowa na które natrafi model BOW mogą zostać zignorowane, ponieważ nie są znane jeszcze.

Bag of words - przykład

- Zdanie 1 – Ala ma kota.
- Zdanie 2 – Jaś ma 3 koty i chomika.
- Zdanie 3 – Chomik to Jaś i ma kota.

1.

-> Ala kot
-> Jaś kot chomik
-> chomik Jaś kot

2.

słowo	Liczba wystąpień
kot	3
Jaś	2
chomik	2
Ala	1

3.

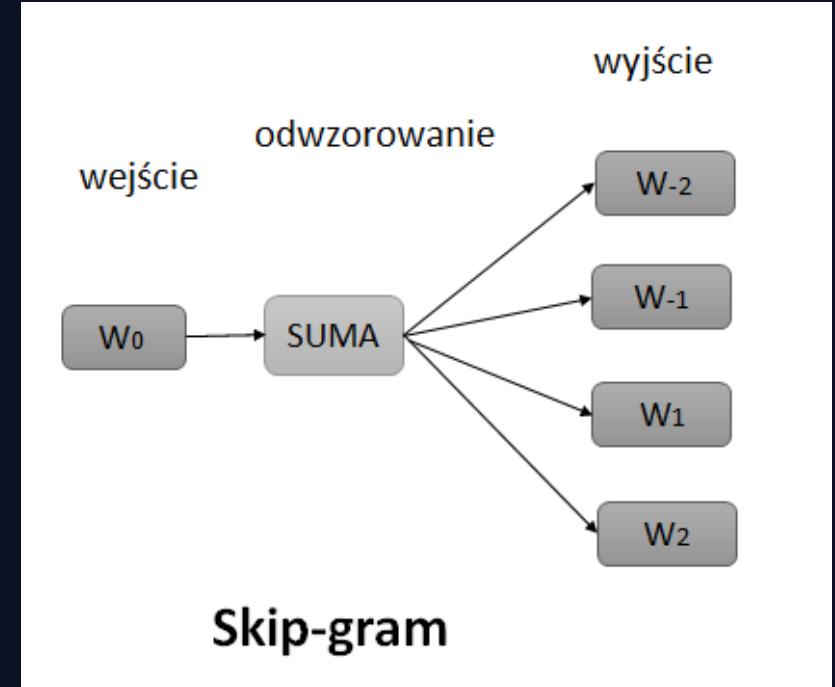
	kot	Jaś	chomik	Ala	wektor
Zdanie 1	1	0	0	1	[1001]
Zdanie 2	1	1	1	0	[1110]
Zdanie 3	1	1	1	0	[1110]

Tokenizacja danych tekstowych - Skip-gram

- Model Skip-gram (ciągłego pomijania gramatycznego) przewiduje słowa w określonym zakresie przed i po bieżącym słowie w tym samym zdaniu.
- Model zapętla się na słowach każdego zdania i próbuje użyć bieżącego słowa w celu przewidzenia jego sąsiadów (tj. jego kontekstu).
- Skip-gram jest uogólnieniem n-gramów.

Tokenizacja danych tekstowych - Skip-gram

- W_0 to docelowe słowo lub dane wejściowe. Istnieje jedna ukryta warstwa, która wykonuje iloczyn skalarny między macierzą wag a wektorem wejściowym W_0 .
- W warstwie ukrytej nie jest używana żadna funkcja aktywacji.
- Wynik iloczynu skalarnego w warstwie ukrytej jest przekazywany do warstwy wyjściowej. Warstwa wyjściowa oblicza iloczyn skalarny między wektorem wyjściowym warstwy ukrytej a macierzą wag warstwy wyjściowej.
- Następnie stosujemy funkcję aktywacji softmax, aby obliczyć prawdopodobieństwo, że słowa będą znajdować się w kontekście W_0 w danej lokalizacji kontekstu.



Tokenizacja danych tekstowych - Skip-gram

1. Przygotowanie danych dla modelu Skip-gram

„Wczorajsze kolokwium z głębokiego uczenia było bardzo trudne.”

- Usuwamy słowa o małym znaczeniu.
- Konwertujemy tekst na listę słów tokenizowanych.
- [„wczorajsze”, „kolokwium”, „głębokiego”, „uczenia”, „było”, „trudne”]

2. Wygeneruj dane treningowe dla modelu Skip-gram

- Żadne słowo się nie powtarza, więc pozostawiamy takie jakie są.
- Definiujemy „słowo kontekstowe”, które będzie następnym słowem po danym w zdaniu „słowem docelowym”.
- Skanując tekst z oknem przygotowujemy słowa kontekstowe i kluczowe.

Tokenizacja danych tekstowych - Skip-gram

- „Wczorajsze kolokwium głębokiego uczenia było trudne.”

The diagram shows a rectangular box containing the text "Wczorajsze kolokwium głębokiego uczenia było trudne." A horizontal arrow points from the word "uczenia" to the right, labeled "słowo docelowe". Another horizontal arrow points from the word "było" to the left, labeled "słowo kontekstowe".

- Pełne dane treningowe:

Przykład treningowy	Słowa kontekstowe	Słowo docelowe
1	(wczorajsze, głębokiego)	kolokwium
2	(kolokwium, uczenia)	głębokiego
3	(głębokiego, było)	uczenia
4	(uczenia, trudne)	było
5	(było)	trudne

Tokenizacja danych tekstowych - Skip-gram

- Kodowanie jednokrotne – zmiana danych tekstowe na dane liczbowe.

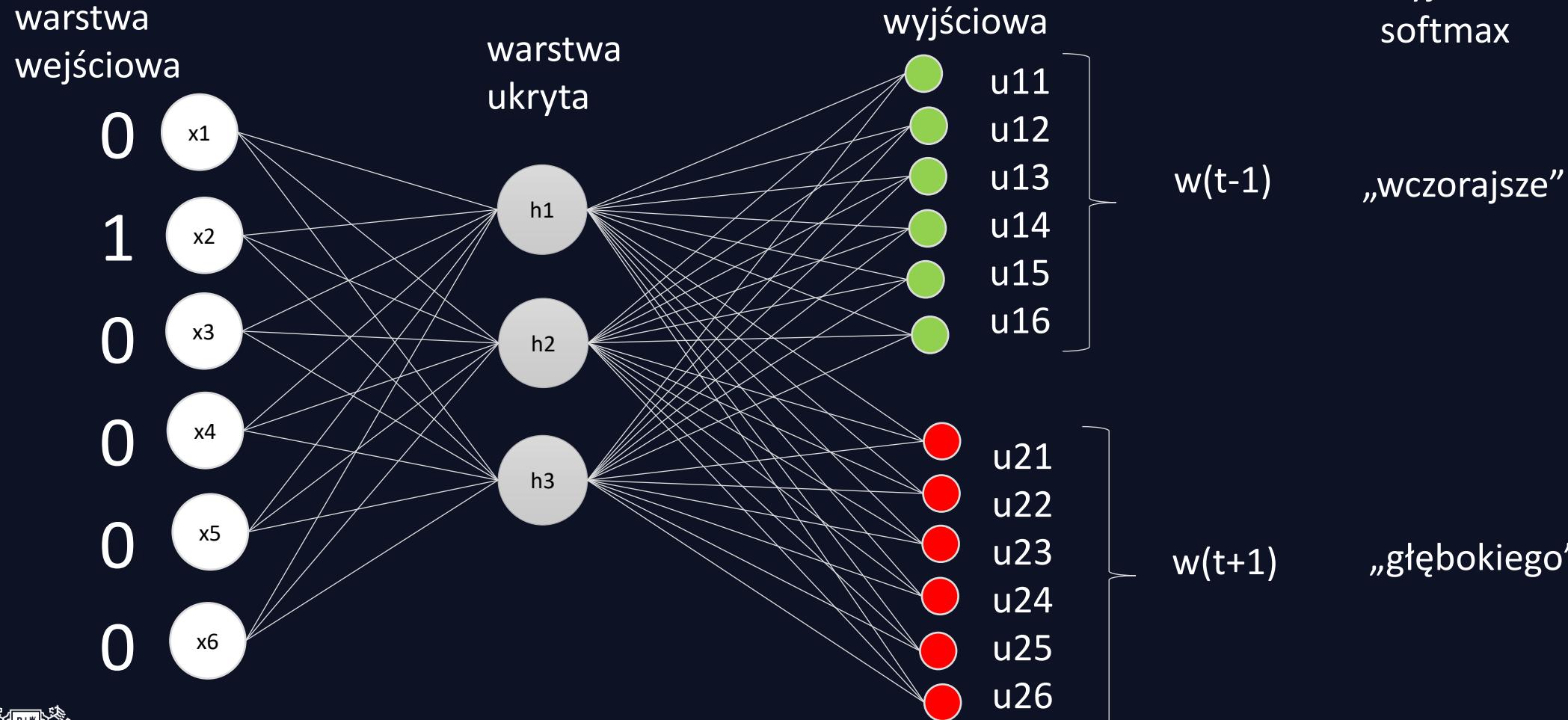
	wczorajsze	kolokwium	głębokiego	uczenia	było	trudne
wczorajsze	1	0	0	0	0	0
kolokwium	0	1	0	0	0	0
głębokiego	0	0	1	0	0	0
uczenia	0	0	0	1	0	0
było	0	0	0	0	1	0
trudne	0	0	0	0	0	1

- Kodowanie słów

Przykład treningowy	Zakodowane słowa kontekstowe	Zakodowanie słowo docelowe
1	([1000000], [001000])	[010000]
2	([010000], [000100])	[001000]
3	([001000], [000010])	[000100]
4	([000100], [0000001])	[000010]
5	([000010])	[000001]

Tokenizacja danych tekstowych - Skip-gram

3. Trening skip-gram



Tokenizacja danych tekstowych - Skip-gram

4. Obliczanie wag modelu.
5. Obliczanie błędu
 - Polega na sumowaniu wszystkich błędów – liczba błędów jest taka sama jak liczna okien kontekstowych (w tym przypadku 2)

Tokenizacja danych tekstowych - Skip-gram

- + Jest to nauka nienadzorowana, więc może pracować na dowolnym tekście bez preprocesingu.
- + Wymaga mniej pamięci w porównaniu z innymi metodami do reprezentacji wektorowych.
- Czas potrzebny do nauczenia algorytmu jest długi.

Tokenizacja danych tekstowych - Word-2-vec

- Jest to technika wykorzystująca model sieci neuronowej do uczenia skojarzeń słów z dużego korpusu tekstu.
- Każde słowo jest reprezentowane jako wektor 32 wymiarów lub wykorzystując więcej wymiarów. (BOW była to jedna liczba)
- Informacja semantyczna oraz relacja pomiędzy słowami jest uwzględniana.
- Word-2-vec to dwuwarstwowa sieć neuronowa, która przetwarza tekst poprzez „wektoryzację” słów.
- Po nauczeniu model może wykrywać słowa synonimiczne lub sugerować dodatkowe słowa do zdania częściowego.

Tokenizacja danych tekstowych - Word-2-vec

- Wektory są dobierane tak, aby podobieństwo cosinusowe między wektorami wskazywało poziom podobieństwa semantycznego między słowami reprezentowanymi przez te wektory.
- Word-2-vec opiera się na technikach CBOW (Continuous Bag of words) i Skip-gram.
- Metoda radzi sobie z nowymi słowami dodawanymi do słownika.
- Metoda opracowana w 2013 roku w Google.

CBOW vs Skip-gram

wejście

odwzorowanie

W_{-2}

W_{-1}

W_1

W_2

wyjście

SUMA

W_0

CBOW

wejście

odwzorowanie

W_0

SUMA

wyjście

W_{-2}

W_{-1}

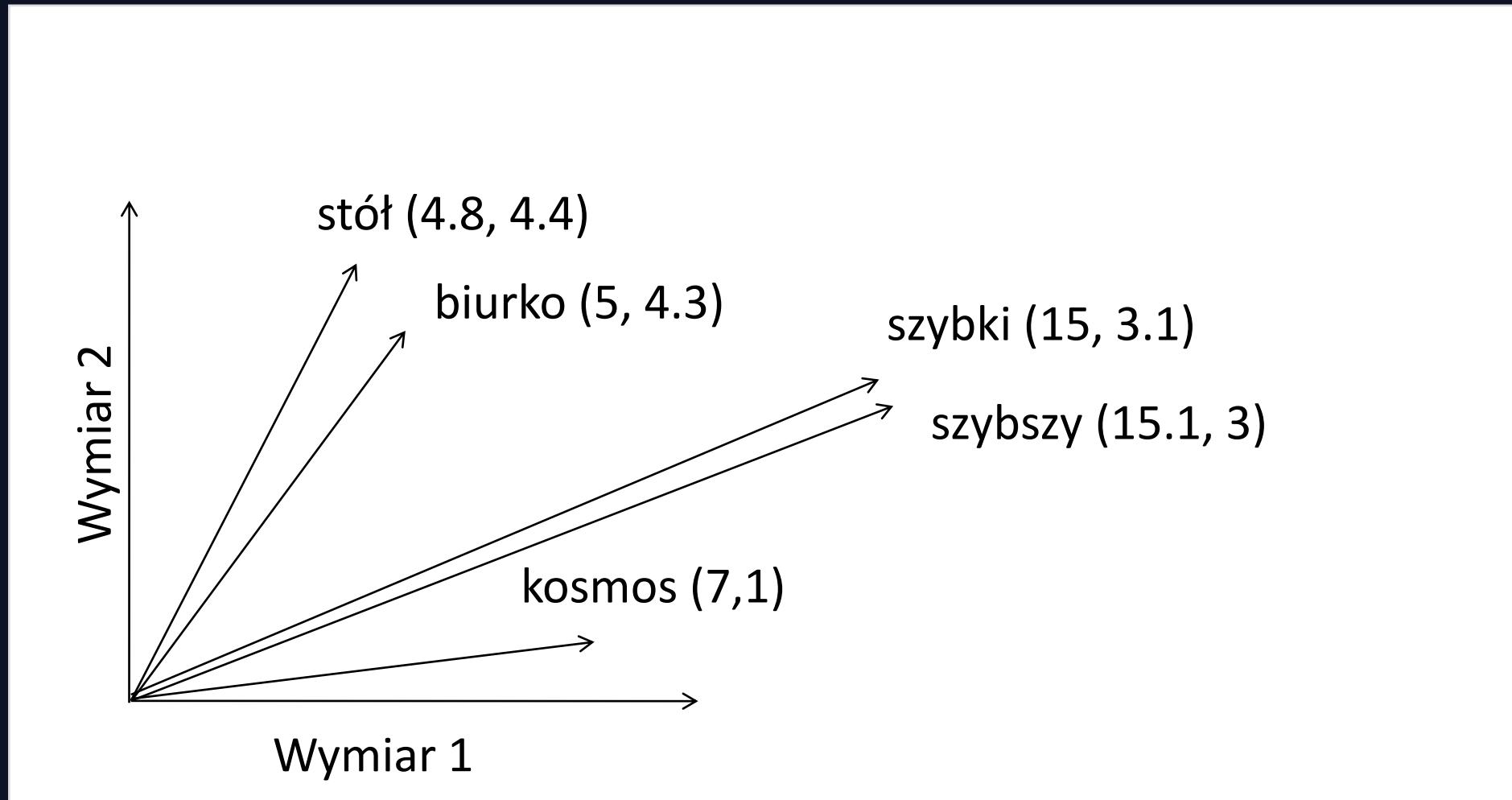
W_1

W_2

Skip-gram

A teraz wyciągamy telefony...

Tokenizacja danych tekstowych - Word-2-vec – reprezentacja wizualna



Tokenizacja danych tekstowych - Word-2-vec

student – mężczyzna + kobieta = studentka

Tokenizacja danych tekstowych - Word-2-vec

Jak poprawić dokładność Word-2-vec?

- Wybór architektury – CBOW lub Skip-gram
 - Skip-gram lepszy dla mniejszych korpusów, lepszy w reprezentacji rzadko występujących słów
 - CBOW – szybciej się trenuje i ma lepszą dokładność dla słów często występujących
- Zwiększenie zbioru testowego
- Zwiększenie wymiarowości wektora
- Zwiększenie wielkości okna
 - Zalecane wielkości okna kontekstowego to 10 dla Skip-gram i 5 dla CBOW

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

BPE jest prostą formą algorytmu kompresji danych, w której najczęstszą parę kolejnych bajtów danych jest zastępuje się bajtem, który nie występuje w tych danych. Jest to **algorytm zachłanny**.

Przykład:

- Zakładamy sekwencję **aaabdaaaabac**
- Najczęściej powtarzająca się para to **aa**, zamieniamy **aa** na **Z**
- Otrzymujemy **ZabdZabac** z podstawieniem **Z=aa**
- Parę **ab** zastępujemy **Y**, otrzymujemy **ZYdZYac**
- Powtarza się **ZY**, kodujemy **ZY = X**, otrzymujemy **XdXac** dla **X = ZY**, **Y = ab**, **Z=aa** ; Nie da się już bardziej zredukować sekwencji

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

„klucz”:8, „kluczowy”:2, „zamek”:6,
„zamkowy”:3

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

Liczba	Token	Częstotliwość
1	</w>	19
2	k	19
3	l	10
4	u	10
5	c	10
6	z	19
7	o	5
8	w	5
9	y	5
10	a	9
11	m	9
12	e	6

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

- Ponieważ w sumie mamy 19 słów, więc mamy 19 tokenów „</w>”. Najwyższym tokenem częstotliwości z liter jest „k” i „z”. W sumie mamy 12 różnych tokenów.
- Następnym krokiem w algorytmie BPE jest wyszukanie najczęstszych parowań, ich scalenie i wykonywanie tej samej iteracji w kółko, aż osiągniemy nasz limit tokenów lub limit iteracji.

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

- Scalanie pozwala na reprezentację korpusu z najmniejszą liczbą tokenów, co jest głównym celem algorytmu BPE, czyli kompresji danych. Aby połączyć, BPE szuka najczęściej reprezentowanych par bajtów. Zakładamy, że znak jest tym samym, co bajt.
- Teraz połączymy najczęstsze pary bajtów w jeden token i dodamy je do listy tokenów oraz przeliczymy częstotliwość występowania każdego tokena. Oznacza to, że nasza liczba częstotliwości zmieni się po każdym kroku scalania. Będziemy kontynuować ten krok łączenia, aż osiągniemy liczbę iteracji lub osiągniemy limit tokena.

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

Liczba	Token	Częstotliwość
1	</w>	19
2	k	19
3	l	10
4	u	10
5	c	10
6	z	19
7	o	5
8	w	5
9	y	5
10	a	9
11	m	9
12	e	6

Liczba	Token	Częstotliwość
1	</w>	19
2	k	19-10=9
3	l	10-10=0
4	u	10
5	c	10
6	z	19
7	o	5
8	w	5
9	y	5
10	a	9
11	m	9
12	e	6
13	kl	8+2+10

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

Liczba	Token	Częstotliwość
1	</w>	19
2	k	19-10=9
3	l	10-10=0
4	u	10
5	c	10
6	z	19
7	o	5
8	w	5
9	y	5
10	a	9
11	m	9
12	e	6
13	kl	8+2+10

Liczba	Token	Częstotliwość
1	</w>	19
2	k	19-10=9
3	l	10-10=0
4	u	10
5	c	10-10=0
6	z	19-10=9
7	o	5
8	w	5
9	y	5
10	a	9
11	m	9
12	e	6
13	kl	8+2=10
14	cz	8+2=10

Tokenizacja danych tekstowych - Byte Pair

Encoding – BPE

Liczba	Token	Częstotliwość
1	</w>	19
2	k	19-10=9
3	l	10-10=0
4	u	10
5	c	10-10=0
6	z	19-10=9
7	o	5
8	w	5
9	y	5
10	a	9
11	m	9
12	e	6
13	kl	8+2=10
14	cz	8+2=10

Liczba	Token	Częstotliwość
1	</w>	19
2	k	19-10=9
3	l	10-10=0
4	u	10-10=0
5	c	10-10=0
6	z	19-10=9
7	o	5
8	w	5
9	y	5
10	a	9
11	m	9
12	e	6
13	kl	8+2=10
14	cz	10-10=0
15	ucz	10

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

Liczba	Token	Częstotliwość
1	</w>	19
2	k	19-10=9
3	z	19-10=9
4	o	5
5	w	5
6	y	5
7	a	9
8	m	9
9	e	6
10	kl	8+2=10
11	ucz	10

Liczba	Token	Częstotliwość
1	</w>	19-5=14
2	k	19-10=9
3	z	19-10=9
4	o	5
5	w	5
6	y	5-5=0
7	a	9
8	m	9
9	e	6
10	kl	8+2=10
11	ucz	10
12	y</w>	5

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

Liczba	Token	Częstotliwość
1	</w>	19-5=14
2	k	19-10=9
3	z	19-10=9
4	o	5
5	w	5
6	y	5-5=0
7	a	9
8	m	9
9	e	6
10	kl	8+2=10
11	ucz	10
12	y</w>	5

Liczba	Token	Częstotliwość
1	</w>	19-5=14
2	k	19-10=9
3	z	19-10=9
4	o	5-5=0
5	w	5-5=0
6	y	5-5=0
7	a	9
8	m	9
9	e	6
10	kl	8+2=10
11	ucz	10
12	y</w>	5
13	ow	5

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

Liczba	Token	Częstotliwość
1	</w>	14
2	k	9
3	z	9
4	a	9
5	m	9
6	e	6
7	kl	10
8	ucz	10
9	y</w>	5
10	ow	5

Liczba	Token	Częstotliwość
1	</w>	14
2	k	9
3	z	9-9=0
4	a	9-9=0
5	m	9
6	e	6
7	kl	10
8	ucz	10
9	y</w>	5
10	ow	5
11	za	9

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

Liczba	Token	Częstotliwość
1	</w>	14
2	k	9
3	z	9-9=0
4	a	9-9=0
5	m	9
6	e	6
7	kl	10
8	ucz	10
9	y</w>	5
10	ow	5
11	za	9

Liczba	Token	Częstotliwość
1	</w>	14
2	k	9
3	z	9-9=0
4	a	9-9=0
5	m	9-9=0
6	e	6
7	kl	10
8	ucz	10
9	y</w>	5
10	ow	5
11	za	9-9=0
12	zam	9

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

Liczba	Token	Częstotliwość
1	</w>	14
2	k	9
3	e	6
4	kl	10
5	ucz	10
6	y</w>	5
7	ow	5
8	zam	9

Liczba	Token	Częstotliwość
1	</w>	14
2	k	9-6=3
3	e	6-6=0
4	kl	10
5	ucz	10
6	y</w>	5
7	ow	5
8	zam	9
9	ek	6

Tokenizacja danych tekstowych - Byte Pair Encoding – BPE

{„klucz</w>”:8, „kluczowy</w>”:2, „zamek</w>”:6, „zamkowy</w>”:3}

Liczba	Token	Częstotliwość
1	</w>	14
2	k	3
3	kl	10
4	ucz	10
5	y</w>	5
6	ow	5
7	zam	9
8	ek	6

Przykłady praktyczne (notatnik interaktywny
DL_09_01_Machine_translation.ipynb)

Implementacja tłumaczenia
maszynowego z użyciem sieci RNN
w środowisku
Google Colaboratory/
Jupyter Notebook

<https://colab.research.google.com/drive/17KtcMJUym1ZsXwoFptsdcdrBNsHfZGmj#scrollTo=3M0J1dZDsTZ1>



POLITECHNIKA
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Dziękuję

SZYMON ZAPOROWSKI



Fundusze
Europejskie
Polska Cyfrowa



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.



Uczenie głębokie

Wykład 10: Od Autokoderów do GAN

Jacek Rumiński
Katedra Inżynierii Biomedycznej, Wydział ETI

Plan wykładu:

- **Od AE do UNET**
- Autokodery z maskowaniem
- Wprowadzenie do GAN

Image restoration

Poprawa jakości obrazów

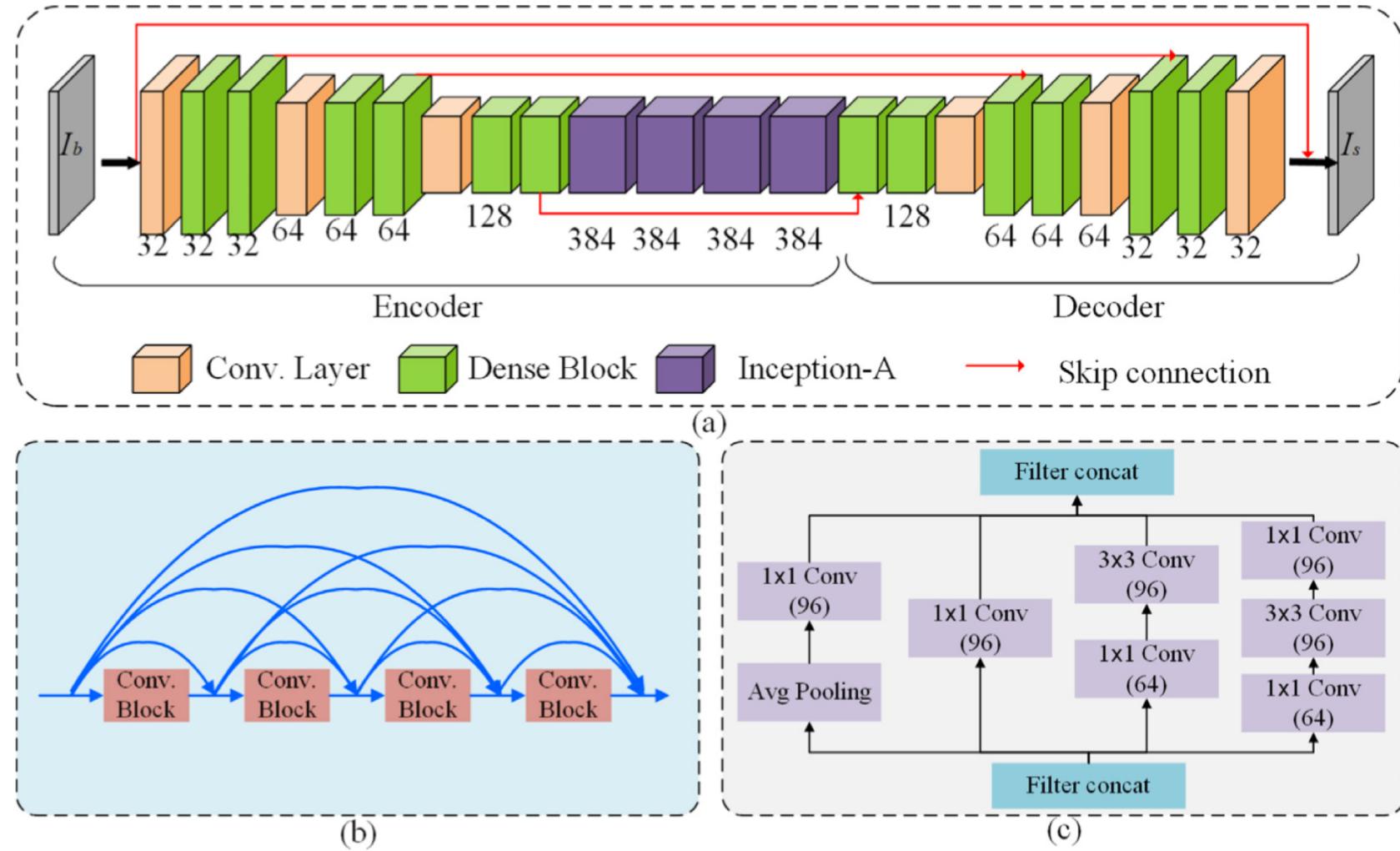


Fig. 7. Proposed MixNet network for image deblurring: (a) overall architecture, (b) DenseBlock structure, and (c) Inception-A block.

Image restoration - Poprawa jakości obrazów



Noisy input frame

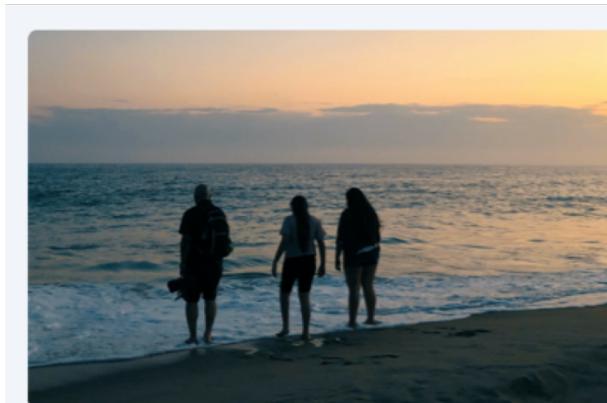


Denoised output frame

Segmentacja semantyczna

Każdy piksel obrazu klasyfikujemy do jednej z K klas.

Zatem wartości pikseli są wejściem, a wyjściem jest macierz o identycznych rozmiarach, której wartości stanowią indeksy klas. Przypisując im kolory uzyskujemy segmenty jednakowych wartości reprezentujących grupy pikseli danego rodzaju.



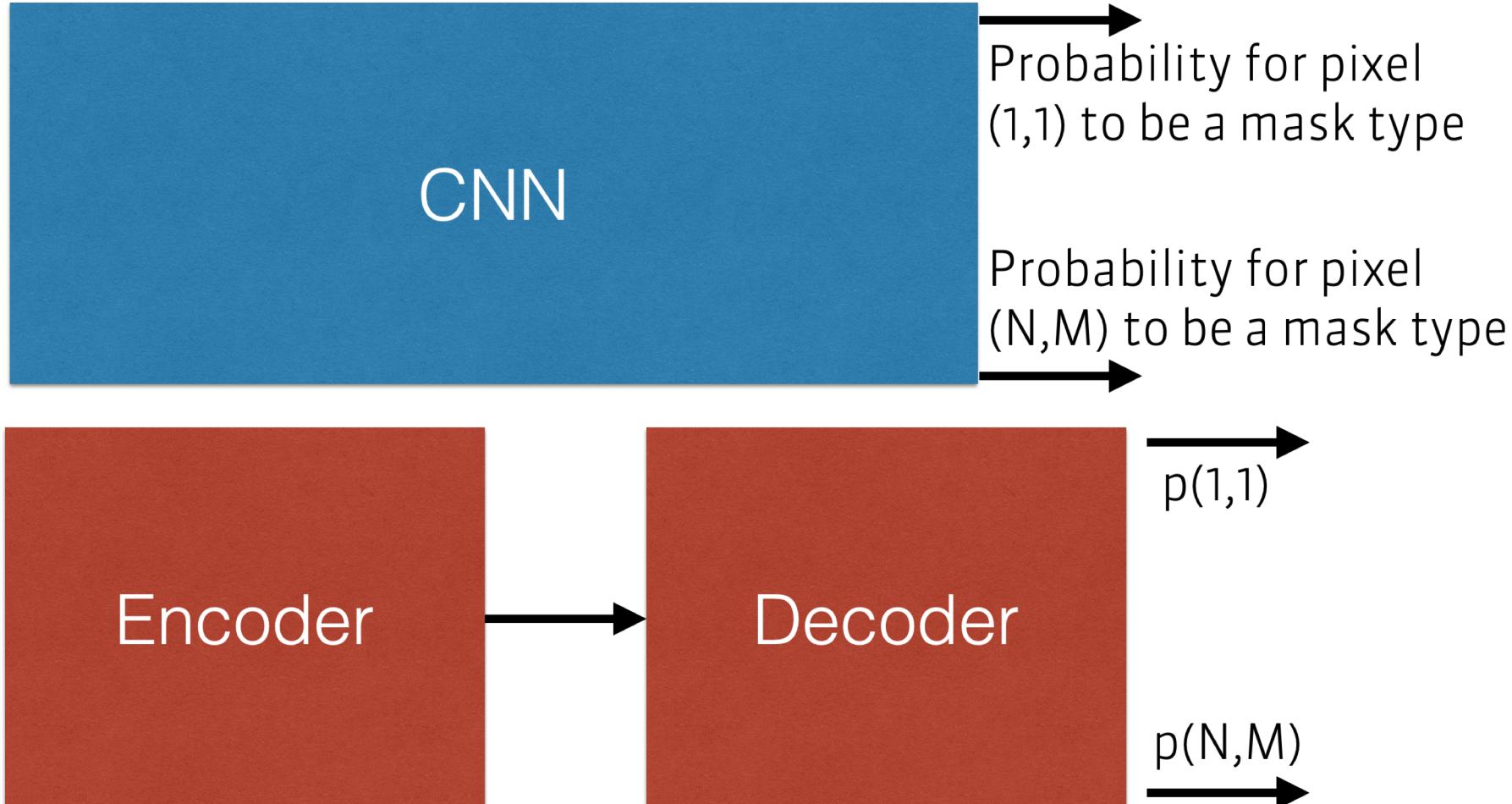
(a) Image



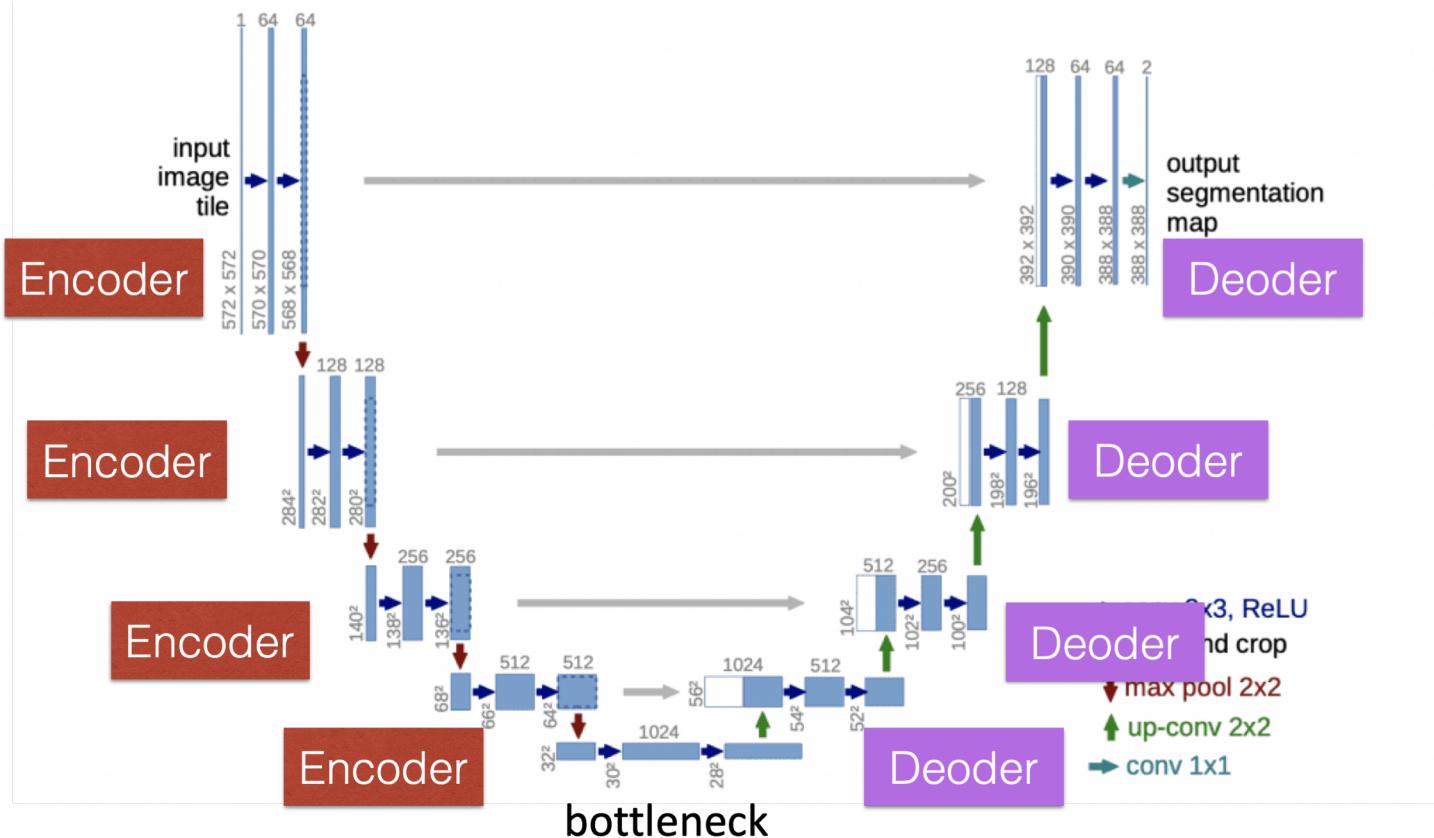
(b) Semantic Segmentation

- Class 0 (each pixel with this class label encoded with blue color)
- Class 1 (each pixel with this class label encoded with yellow color)
- Class 2 (each pixel with this class label encoded with red color)
- Class 3 (each pixel with this class label encoded with violet color)

Segmentacja semantyczna

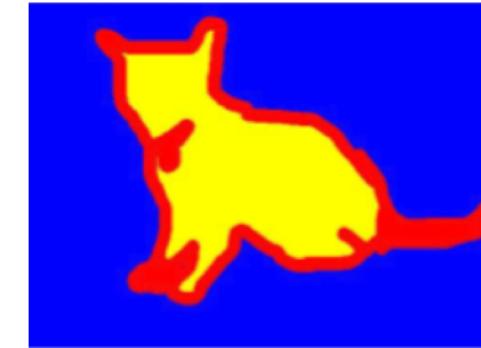


Segmentacja semantyczna - UNET



Training dataset: images with classified pixels
e.g.: The Oxford-IIIT Pet Dataset

Image matting: extracting interesting targets from an image



Trimap: foreground, background, and unknown regions in the image.

Przykłady praktyczne (notatnik interaktywny)

AE w segmentacji w środowisku Google Colaboratory/ Jupyter Notebook.

https://drive.google.com/file/d/12kzkT9E-iPCv2zsgTcag8y2Cm9FNDn0_/view?usp=sharing

Przykłady praktyczne (notatnik interaktywny)

U-NET w segmentacji w środowisku Google Colaboratory/ Jupyter Notebook.

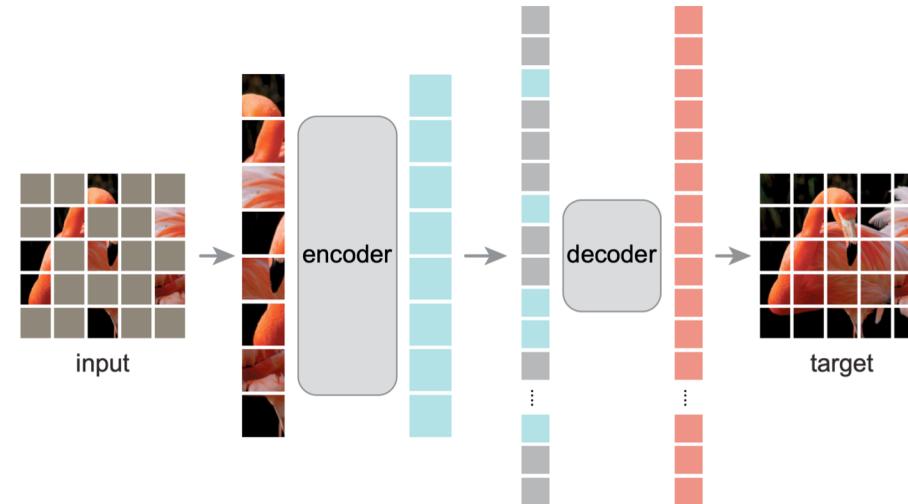
https://drive.google.com/file/d/12kzkT9E-iPCv2zsgTcag8y2Cm9FNDn0_/view?usp=sharing

Autokodery z maskowaniem (ang. Masked Autoencoders)

W procesie uczenia zamiast np. dodania szumu do wejście zasłoń część danych.

Przykładowo:

- Podziel obraz na bloki (patch) o określonych rozmiarach,
- Zdefiniuj P procent bloków podlegających losowemu maskowaniu
- Maskuj P procent bloków: przypisz wartości maski (np. 0) zamiast wartości oryginalnych.

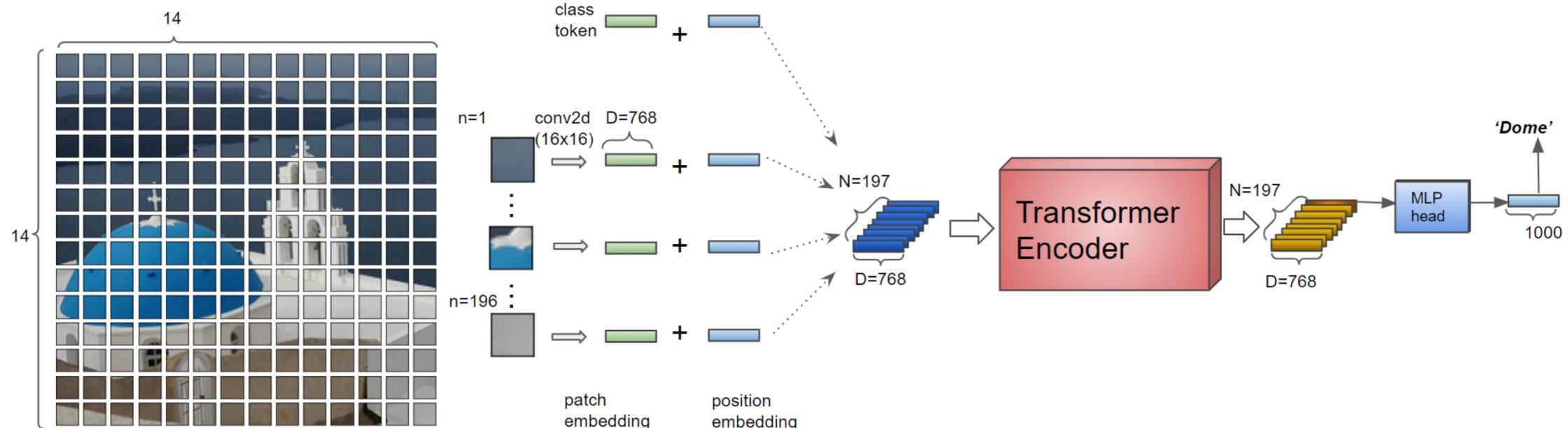


Źródło: Kaiming He et al., Masked Autoencoders Are Scalable Vision Learners, 2021, <https://arxiv.org/abs/2111.06377>
<https://github.com/EdisonLeeeee/Awesome-Masked-Autoencoders>

Autokodery z maskowaniem (ang. Masked Autoencoders)

Współczesne MAE wykorzystują Vision Transformers lub złożone sieci splotowe (np. ConvNext).

W przypadku modeli ViT od razu dostępne są bloki pikseli, które poddawane są na przetwarzaniu przez koder. (Alexey Dosovitskiy , et. al , ICLR 2021).



Źródło:

Przykłady praktyczne

Analiza kodu źródłowego MAE

W szczególności w „**models_mae.py**”:

- podział na patche
- losowy wybór patchy
- funkcja kosztu odniesiona tylko do USUWANYCH i odtwarzanych patchy.

<https://github.com/facebookresearch/mae>

Przykłady praktyczne (notatnik interaktywny)

MAE zastosowanie w środowisku Google Colaboratory/ Jupyter Notebook.

https://colab.research.google.com/github/facebookresearch/mae/blob/main/demo/mae_visualize.ipynb

Model GAN - Generatywne Sieci Przeciwstawne (Generative Adversarial Networks)

Algorytm uczący w autokoderze wariacyjnym maksymalizujący funkcję kosztu \mathcal{L}_{ELBO} uzyskuje wartość co najmniej taką jak wartość log wiarygodności ($\ln(p_\theta(\mathbf{x}))$).

$$\mathcal{L}_{ELBO} = E_{\mathbf{z} \sim q} [\ln(p_\theta(\mathbf{x}|\mathbf{z}))] - D_{KL} (q_\phi(\mathbf{z}|\mathbf{x}), p(\mathbf{z})) \leq \ln(p_\theta(\mathbf{x}))$$

Uzyskiwana w wyniku uczenia funkcja rozkładu prawdopodobieństwa dla modelu $p_{model}(\mathbf{x}; \theta)$ powinna odwzorowywać „prawdziwy” rozkład wynikający z danych $p_{dane}(\mathbf{x})$.

Jeśli rozkład aposteriori $p(\mathbf{z}|\mathbf{x})$ jest niewystarczająco aproksymowany przez $q_\phi(\mathbf{z}|\mathbf{x})$ lub rozkład apriori $p(\mathbf{z})$ jest słabo aproksymowany przez $q(\mathbf{z})$ (czy przez $N(0, 1)$) wówczas nawet dostępny nieskończony obszerny zbiór uczący czy najlepszy algorytm optymalizacyjny uzyskiwana będzie taka różnica pomiędzy \mathcal{L} i log wiarygodnością, że uzyskiwany rozkład $p_{model}(\mathbf{x}; \theta)$ nie będzie oddawał $p_{dane}(\mathbf{x})$.

Model GAN - Generatywne Sieci Przeciwnostawne

Obrazy generowane przez VAE są często rozmyte.

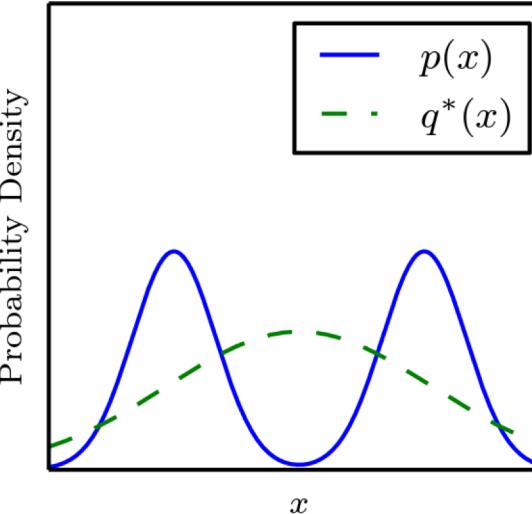
W literaturze podawano argument, że przyczyną rozmycia może być stosowanie kryterium maksymalnej wartości funkcji wiarygodności, która dąży do uśrednienia wyników.

Rozpatrzmy pierwszy rysunek:

- $p(x)$ to rozkład dla danych, w tym przypadku dwa złączone rozkłady Gaussa
- $q(x)$ to rodzina rozkładów Gaussa.

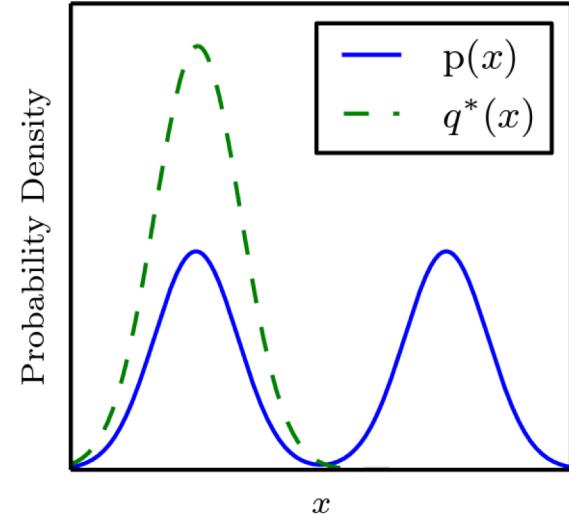
Poszukujemy takich parametrów rodziny rozkładu q , że uzyskujemy najmniejszą różnicę (największe podobieństwo) pomiędzy $p(x)$ i tym wybranym $q^*(x)$. W podanej ilustracji, ten wybrany $q^*(x)$ uśrednia wyniki z obu pików $p(x)$.

$$q^* = \operatorname{argmin}_q D_{\text{KL}}(p\|q)$$



Maximum likelihood

$$q^* = \operatorname{argmin}_q D_{\text{KL}}(q\|p)$$



Reverse KL

$D_{\text{KL}}(p_{\text{dane}}\|\|p_{\text{model}})$ preferuje przypisać wysoką wartość prawdopodobieństwa tam, gdzie występują dane x .

$D_{\text{KL}}(p_{\text{model}}\|\|p_{\text{dane}})$ preferuje przypisać niską wartość prawdopodobieństwa tam, gdzie nie występują dane x .

Model GAN - Generatywne Sieci Przeciwnstawne

Jednak, modele GAN, również mogą używać kryterium maksymalnej wartości funkcji wiarygodności, a umożliwiają uzyskanie znacznie wyraźniejszych wyników (ostre).

Badania wskazują, że przyczyną rozmycia obrazów dla VAE mogą być związane z tym, że przyjmujemy proste funkcje rozkładu prawdopodobieństwa i za bardzo aproksymujemy stosowne rozkłady. Więcej można poczytać w literaturze, m.in. w [1][2].

Model GAN zaproponowano w [3]. W metodzie tej, możemy w przybliżeniu uzyskać minimalną postać symetrycznej dywergencji Jensena-Shannona:

$$D_{JS}(p||q) = 0.5 D_{KL}\left(p\middle\|(\frac{p+q}{2})\right) + 0.5 D_{KL}\left(q\middle\|(\frac{p+q}{2})\right)$$

W modelu GAN optymalizowane są równocześnie dwa bloki: generatora (jak dekoder AE) i dyskryminatora. Ze względu na podaną postać $D_{JS}(p||q)$ i tę jednoczesną optymalizację można dobrze aproksymować „prawdziwy” rozkład $p(x)$ przy założeniu wystarczająco dobrze reprezentowanego zbioru uczącego.

[1] S. Zhao, et al., Towards a Deeper Understanding of Variational Autoencoding Models, <https://arxiv.org/pdf/1702.08658.pdf>

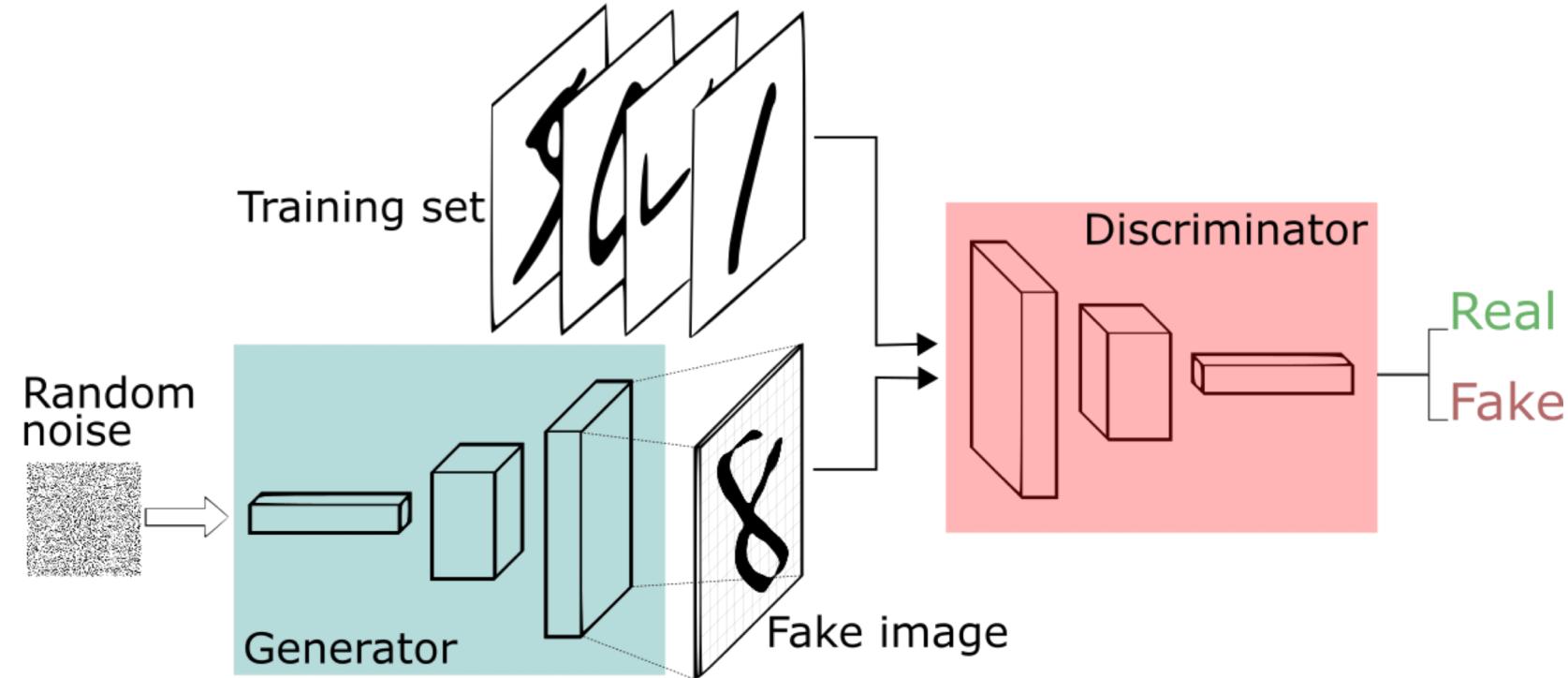
[2] A. Nowozin, et al., f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization, <https://arxiv.org/abs/1606.00709>

[3] Goodfellow, et al. Generative adversarial nets. In NIPS, pages 2672–2680, 2014.

Model GAN - Generatywne Sieci Przeciwnostawne

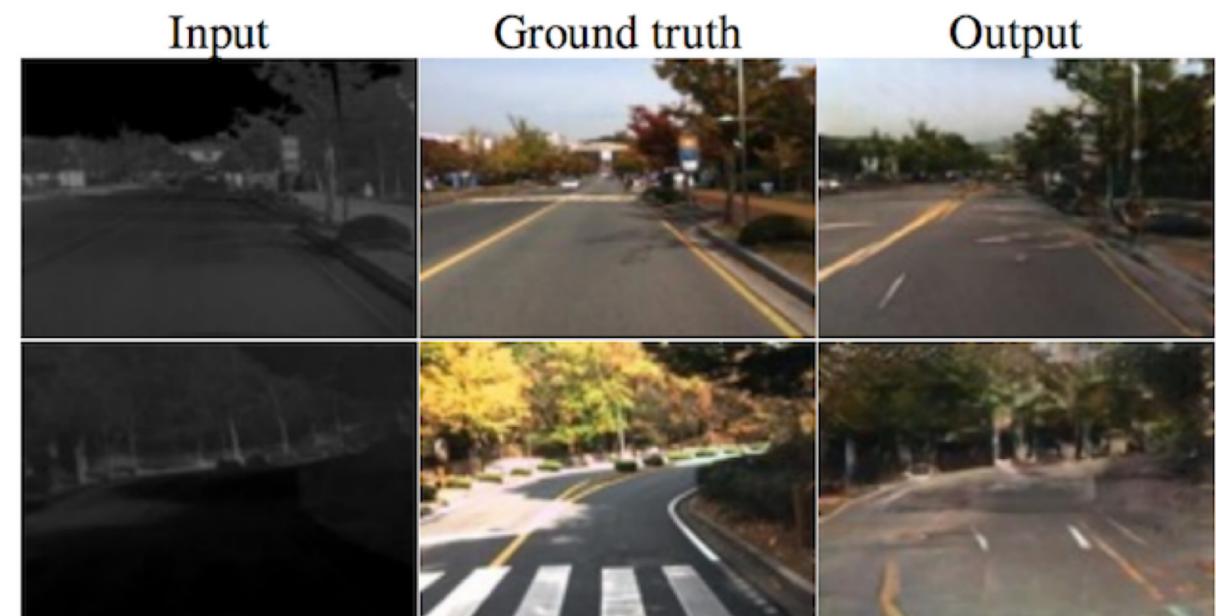
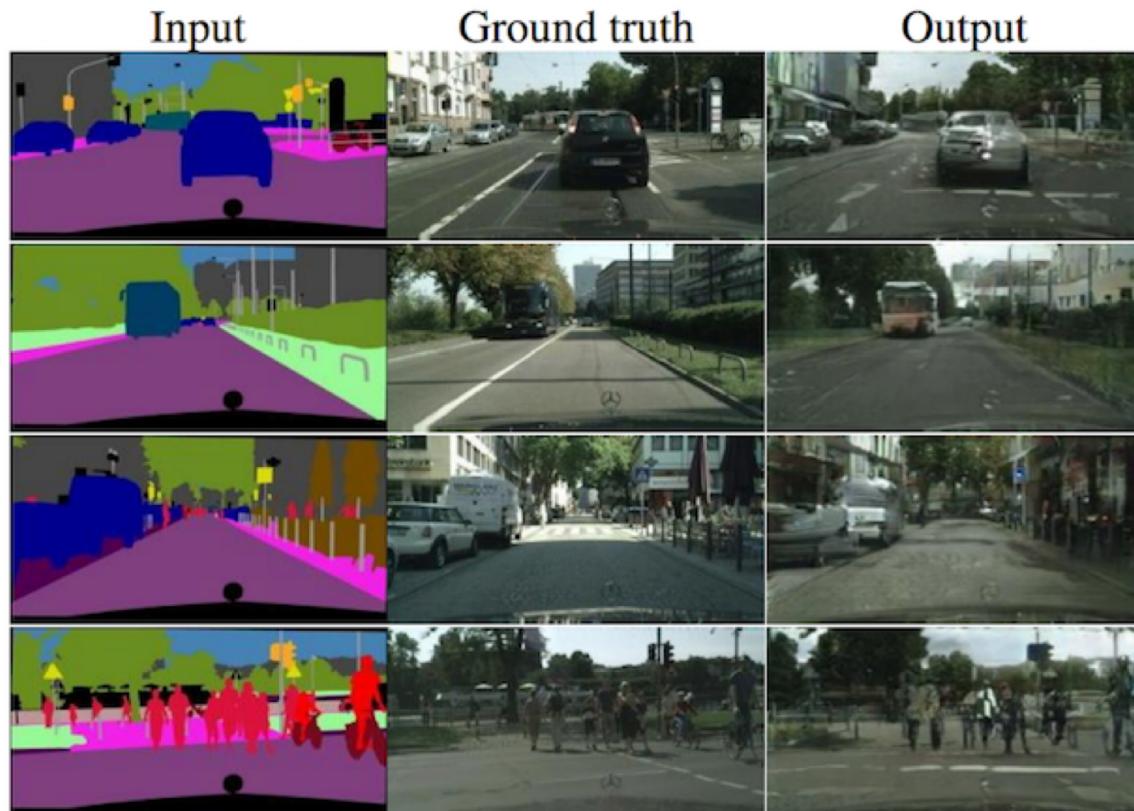
Dwa kluczowe bloki (modele):

- G - generator uczy się generować właściwe przykłady (negative training examples for discriminator),
- D - dyskryminator uczy się rozróżniać pomiędzy danymi referencyjnymi ("prawdziwymi"), a danymi z generatora.



źródło: <https://wiki.pathmind.com/generative-adversarial-network-gan>

Model GAN - Generatywne Sieci Przeciwnostawne



Model GAN - Generatywne Sieci Przeciwnstawne

W modelu GAN poprzez optymalizację dwóch bloków możemy uzyskać estymatę:

$$\frac{p_{dane}}{p_{model}}$$

dla każdego x.

Ważne:

Modele generacyjne bazujące na VAE wykorzystują aproksymację wynikającą z ELBO.
Modele GAN wykorzystując uczenie NADZOROWANE umożliwiając uzyskanie estymacji relacji dwóch rozkładów: $\frac{p_{dane}}{p_{model}}$.

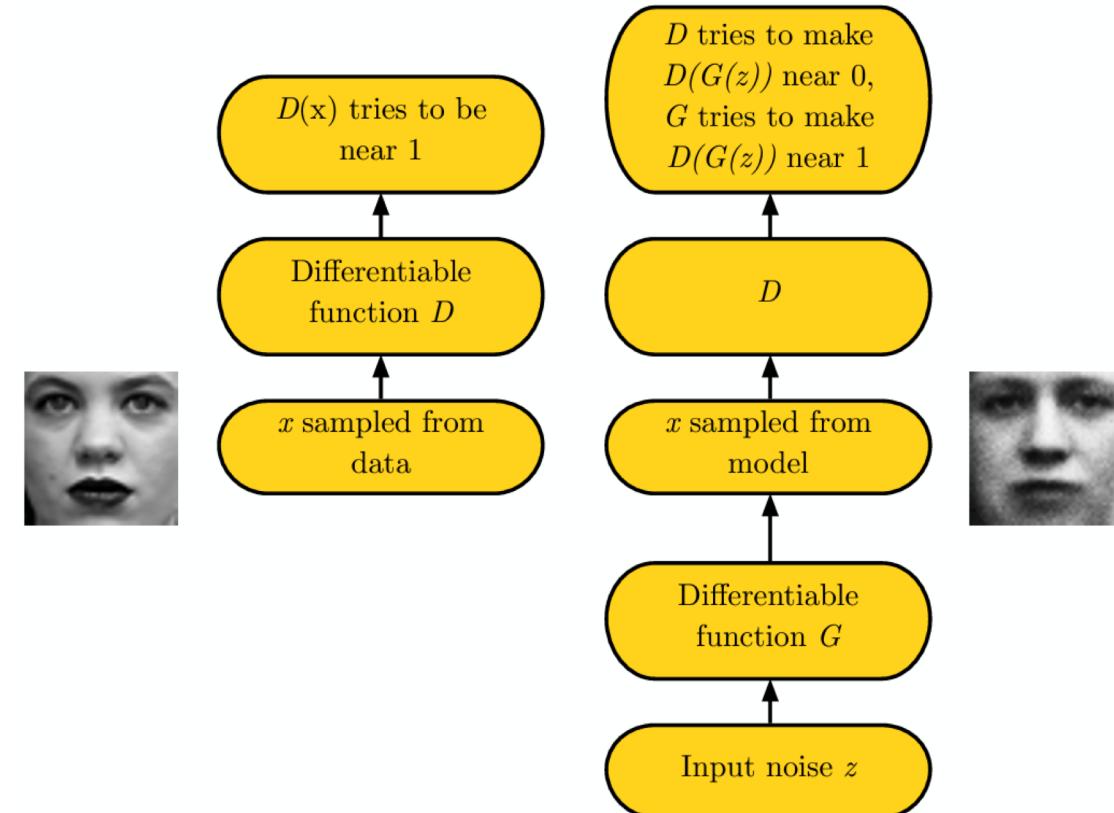
Nadzorowanie (dyskryminator) - dwie klasy: FAKE vs. REAL

Model GAN - Generatywne Sieci Przeciwstawne

W GANie generator (G) jest trenowany tak, aby wprowadzić w błąd dyskryminator (D).

Ilustracja: generator (przestępca) doskonali się w fałszerstwie obrazów znanych mistrzów, a dyskryminator (ekspert sztuki) odróżnia podróbkę od prawdziwych dzieł mistrzów. Generator będzie się w procesie uczenia doskonalił (zbliżał do rzeczywistego rozkładu w przestrzeni danych źródłowych).

Można przyjąć analogię, że generator i dyskryminator grają w grę (gra o sumie zerowej – suma kosztu graczy jest zero). Mogą być przeciwnikami (adversary) lub współpracownikami (collaborator).



Model GAN - Generatywne Sieci Przeciwnstawne

Będziemy minimalizować funkcję kosztu dyskryminatora $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ kontrolując jedynie parametry bloku dyskryminatora $\theta^{(D)}$ oraz niezależnie minimalizować funkcję kosztu generatora $J^{(G)}(\theta^{(D)}, \theta^{(G)})$ kontrolując jedynie parametry bloku dyskryminatora $\theta^{(G)}$. Jest to zatem bardziej gra niż optymalizacja.

Rozwiązaniem gry jest równowaga Nasha (strategia każdego z graczy jest optymalna) uzyskując optymalny zestaw $(\theta^{(D)}, \theta^{(G)})$ jako stosowne wartości parametrów uzyskane dla minimów, odpowiednio dla $J^{(D)}$ i $J^{(G)}$.

W procesie treningu, dla danego kroku:

- pobieramy próbki **x** oraz próbki **z** (stosowne porcje - batche),
- wyznaczamy funkcje kosztu i gradienty równocześnie dla generatora i dyskryminatora,
- obliczamy stosowne gradienty i aktualizujemy $\theta^{(D)}$, oraz $\theta^{(G)}$.

Model GAN - Generatywne Sieci Przeciwnstawne

Klasyczna funkcja kosztu dyskryminatora:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2} \mathbb{E}_{x \sim p_{dane}} \ln(D(x)) - \frac{1}{2} \mathbb{E}_z \ln(1 - D(G(z)))$$

$D(x)$ - wynik dyskryminatora dla danych ze zbioru źródłowego (etykieta 1 dla wszystkich danych - REAL)

$D(G(z))$ - wynik dyskryminatora dla danych z generatora (etykieta 0 dla wszystkich danych - FAKE)

Funkcja kosztu generatora:

1. Gra o sumie zerowej - teoretyczna: $J^{(G)} = -J^{(D)} \rightarrow \theta^{(G)*} = \operatorname{argmin}_{\theta^{(G)}} \max_{\theta^{(D)}} J^{(D)}(\theta^{(D)}, \theta^{(G)})$
2. Entropia wzajemna: $J^{(G)} = -\frac{1}{2} \mathbb{E}_z \log(D(G(z)))$
2. Metodą największej wiarygodności: $J^{(G)} = -\frac{1}{2} \mathbb{E}_z \exp(\sigma^{-1}(D(G(z))))$; σ - sigmoid

Przykłady praktyczne (notatnik interaktywny)

GAN w środowisku Google Colaboratory/ Jupyter Notebook.

<https://colab.research.google.com/drive/1o0WOsk6l25SLCTFambHXU3SP92ochS21?usp=sharing>

Przykłady praktyczne (notatnik interaktywny)

Prosty GAN zastosowanie w środowisku Google Colaboratory/ Jupyter Notebook.

https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/gan/gan.ipynb

Przykłady praktyczne (notatnik interaktywny)

Prosty CGAN zastosowanie w środowisku Google Colaboratory/ Jupyter Notebook.

https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/gan/gan-conv.ipynb

Przykłady praktyczne (notatnik interaktywny)

**Prosty CGAN z label smoothing
zastosowanie w środowisku Google
Colaboratory/
Jupyter Notebook.**

https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/gan/gan-conv-smoothing.ipynb

Model DCGAN - Deep Convolutional GAN

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Przykłady praktyczne (notatnik interaktywny)

Prosty DCGAN zastosowanie w środowisku Google Colaboratory/ Jupyter Notebook.

https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/gan/dcgan-cats-and-dogs.ipynb

Model GAN - Generatywne Sieci Przeciwnieprzeciwstawne

Wykład kolejny:

Sebastian Raschka, Introduction to Generative Adversarial Networks in Computer Vision, Summer School, Gdańsk, 2021 -
<https://2021.dl-lab.eu/schedule/> ->

<http://kibit.eti.pg.gda.pl/AI/ISSonDL2021/Sebastian%20Raschka%20Introduction%20to%20Generative%20Adversarial%20Networks%20in%20Computer%20Vision.mp4>

- Wasserstein Generative Adversarial Networks (GAN)
- Rady związane z tworzeniem modeli GAN: <https://github.com/soumith/ganhacks>
- Zastosowania GAN
- Rozwój GAN (<https://neptune.ai/blog/6-gan-architectures>): CycleGAN, StyleGAN, pixelRNN, text-2-image, DiscoGAN, lsGAN, cGAN-pix2pix (<https://www.tensorflow.org/tutorials/generative/pix2pix>)

Dziękuję

Jacek Rumiński
Katedra Inżynierii Biomedycznej, Wydział ETI



POLITECHNIKA
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Uczenie głębokie: Modele Generatywne

Szymon Zaporowski



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu

- *Model generatywny*
- *Klasyfikacja sieci generatywnych*
- *Model Autoregresyjny*
- *PixelRNN i PixelCNN jako przykład sieci autoregresyjnej*
- **Model GAN**
- **GAN Wassersteina**
- **GAN warunkowy**

Generatywne sieci przeciwnostawne

- GAN to modele *generatywne* : tworzą nowe instancje danych, które przypominają dane treningowe. Na przykład GAN mogą tworzyć obrazy, które wyglądają jak fotografie ludzkich twarzy, nawet jeśli twarze nie należą do żadnej prawdziwej osoby.
- Słowo „generatywne” opisuje klasę modeli statystycznych, która kontrastuje z modelami *dyskryminacyjnymi*.
- Modele generatywne muszą znajdować korelacje pomiędzy obiekta mi, np. nos będzie nad ustami, uszy będą ustawione symetrycznie itp. Natomiast modele dyskryminacyjne uczą się tylko różnic, np. na zdjęciu jest nos, na zdjęciu są oczy. Model ten może ignorować wiele korelacji.

Zagadka

- Mamy wyniki z kolokwium dla 1000 osób. Rozkład wyników kolokwium modelujemy za pomocą procedury:
- Rzuć trzema sześciocieinnymi kości mi.
- Pomnóż wartości z kości przez stałą „W”.
- Powtórz 100 razy i weź średnią wszystkich wyników.
- Próbujemy przypisać różne wartości dla stałej „W”, aż wynik procedury będzie równy średniej z rzeczywistych wyników kolokwium.
- Czy taki model jest modelem generatywnym czy dyskryminacyjnym?

Zasada działania GAN

GAN składa się z dwóch sieci neuronowych:

- generatora – tworzy nowe dane, które przekazuje do dyskryminatora,
- dyskryminatora – ocenia stworzone dane pod kątem autentyczności, decyduje, czy dane które przegląda są z rzeczywistego zbioru danych uczących, czy są fałszywe (wygenerowane).

Sieci generatywne opierają się na scenariuszu korzystającym z teorii gier, w którym sieć generatora musi konkurować z przeciwnikiem, czyli siecią dyskryminacyjną. Im bardziej realistyczne będą dane wygenerowane przez generator, tym większa szansa, że dyskryminator określi je jako prawdziwe i „da się oszukać”.

Model generatora

- Model generatora pobiera losowy wektor o stałej długości jako dane wejściowe, przekształca ten wektor losowy (szum) w znaczący sygnał wyjściowy i generuje nową próbę danych.
- Aby wytrenować sieć neuronową (generator), zmieniamy wagi sieci, aby zmniejszyć błąd na wyjściu. W tym celu korzystamy z propagacji wstecznej, która zaczyna się na wyjściu i wraca przez dyskryminator do generatora.
- Po treningu model generatora jest zachowywany i wykorzystywany do generowania nowych próbek.

Model dyskryminatora

- Model dyskryminatora bierze przykład z domeny jako dane wejściowe (rzeczywiste lub wygenerowane) i przewiduje binarną etykietę klasy rzeczywistą lub fałszywą (wygenerowaną).
- Prawdziwy przykład pochodzi z zestawu danych treningowych. Wygenerowane przykłady są wyprowadzane przez model generatora.
- Dyskryminator używany jest tylko do etapu uczenia, ponieważ później nie jest już potrzebny.

Dyskryminator

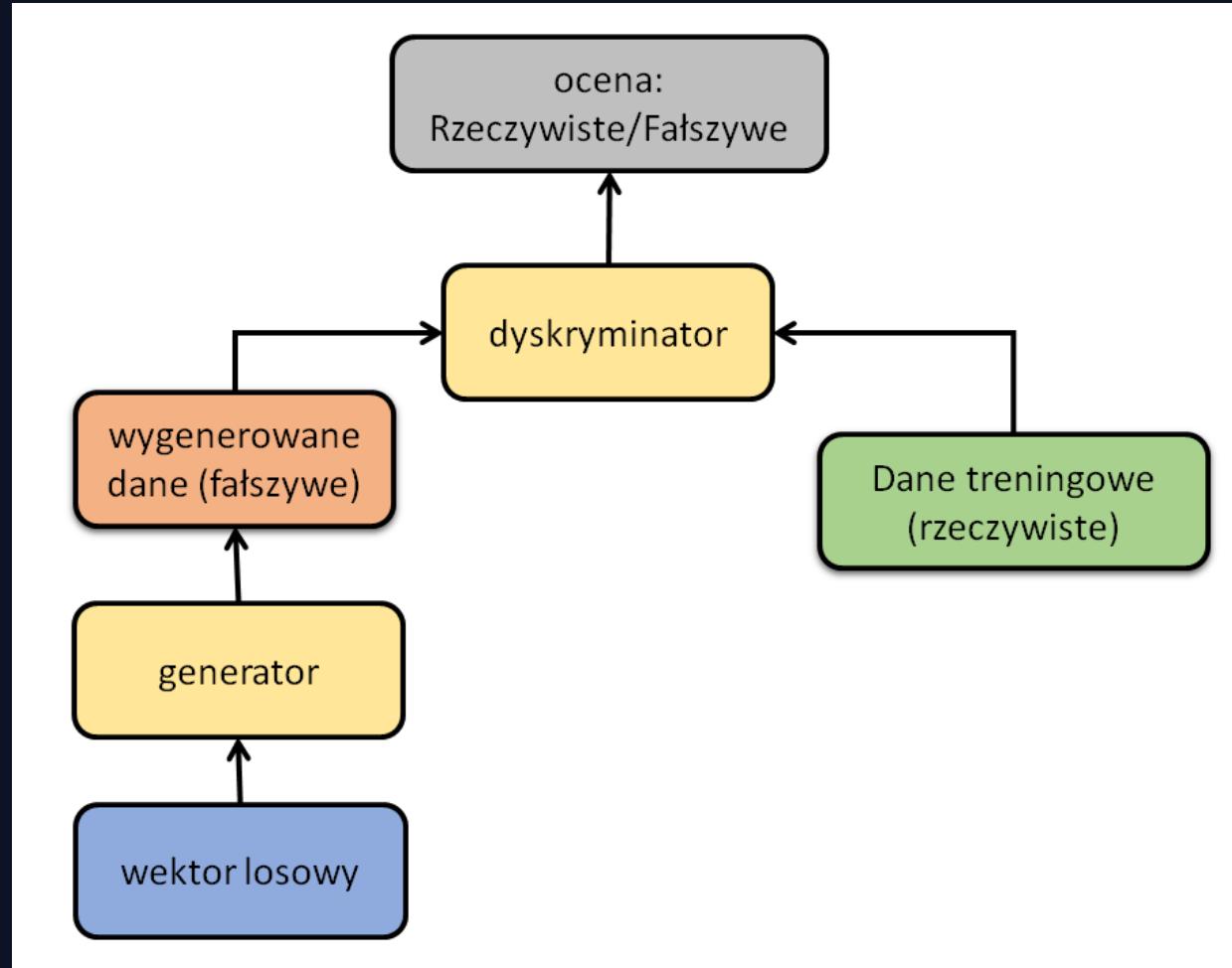


Trening dyskryminatora

Podczas treningu dyskryminacyjnego:

1. Dyskryminator klasyfikuje zarówno dane rzeczywiste i fałszywe z generatora.
2. Funkcja straty dyskryminacji karze dyskryminator za błędną klasyfikację rzeczywistej instancji jako fałszywej lub fałszywej instancji jako rzeczywistej.
3. Dyskryminator aktualizuje swoje wagi poprzez wsteczną propagację od funkcji straty dyskryminatora przez sieć dyskryminacyjną.

Architektura sieci GAN



Trening GAN

Trening GAN przebiega w naprzemiennych okresach:

1. Dyskryminator trenuje się przez jedną lub więcej epok.
2. Generator trenuje się przez jedną lub więcej epok.
3. Powtarzamy kroki 1 i 2, aby kontynuować trening sieci generatora i dyskryminatora.

Po kilku takich cyklach generator poprawia się, a wydajność dyskryminatora pogarsza się, ponieważ dyskryminatorowi nie jest już tak łatwo odróżnić rzeczywistych danych od fałszywych. Jeśli generator działa idealnie, dyskryminator ma dokładność 50%. W efekcie dyskryminator rzuca monetą, aby sklasyfikować dane, a sprzężenie zwrotne z dyskryminatora staje się z czasem mniej znaczące.

Funkcje straty i wagi w GAN

Jakie funkcje straty dla dyskryminatora i generatora?

- GAN może wykorzystywać tę samą funkcję stratę zarówno do treningu generatora, jak i dyskryminatora (lub te same straty różniące się tylko znakiem), ale nie musi tak być. W rzeczywistości częściej stosuje się różne funkcje strat dla dyskryminatora i generatora.
- W trakcie uczenia generatora gradienty rozchodzą się przez sieć dyskryminatora do sieci generatora, więc wagi w sieci dyskryminatorów wpływają na aktualizacje sieci generatorów.
- Dyskryminator nie aktualizuje swoich wag podczas uczenia generatora.

GAN – możliwe problemy i rozwiązania

- W sytuacji, gdy dyskryminator klasyfikuje za dobrze, to trening generatora może być nieskuteczny z powodu **zanikających gradientów**.
 - Modyfikacja funkcji straty minimax
 - Użycie funkcji straty Wassersteina
- **Załamanie trybu** występuje w sytuacji, gdy generator zaczyna generować dane wyjściowe bardzo podobne lub identyczne, przez co może utknąć w minimum lokalnym.
 - Funkcja straty Wassersteina
 - Rozwinięte GAN
- Sieć GAN może **nie osiągać zbieżności**.
 - Karanie wagi dyskryminatora
 - Dodawanie szumu do sygnałów wejściowych dyskryminatora

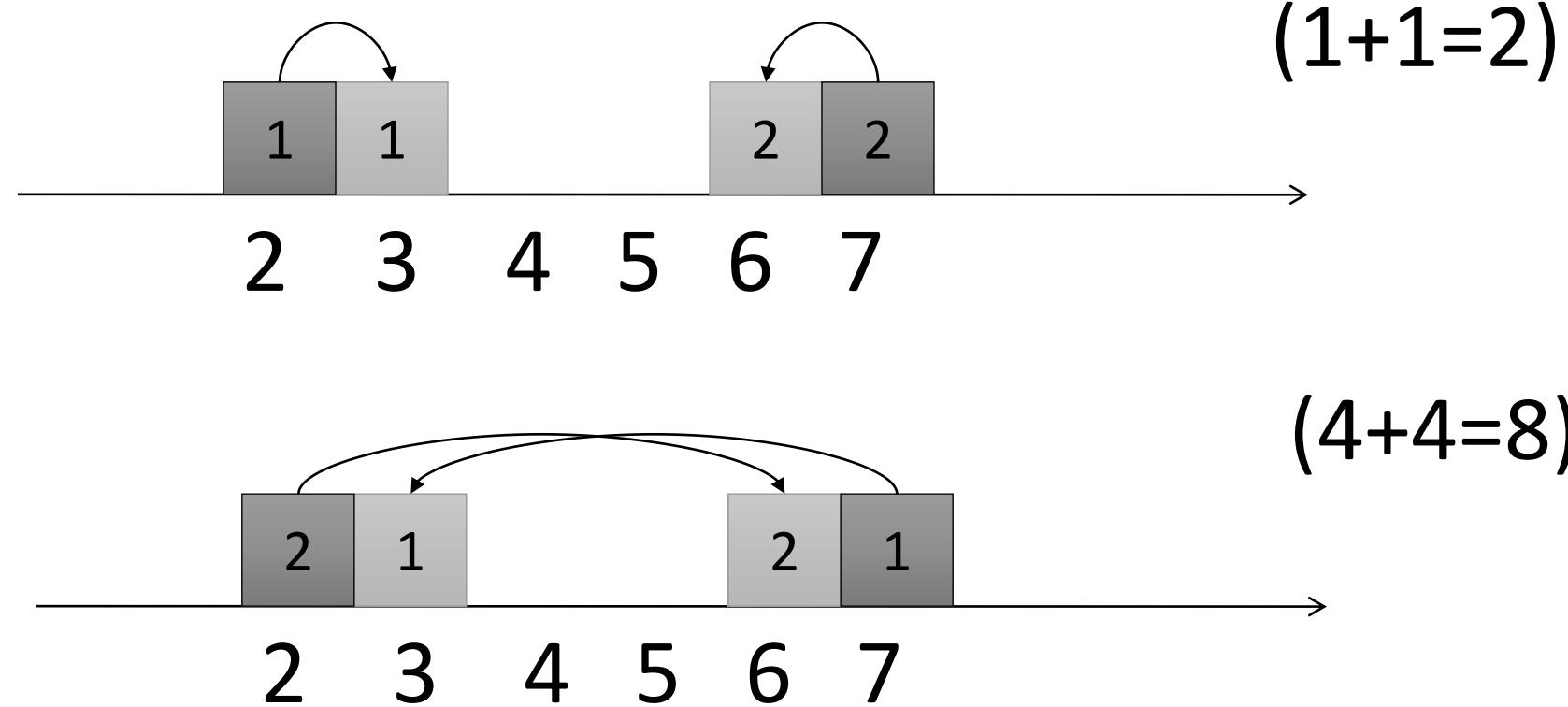
Wasserstein Generative Adversarial Network (WGAN) – czym jest

- WGAN opisany został po raz pierwszy w 2017 roku i jest rozszerzeniem generatywnej sieci kontradyktoryjnej, które zarówno poprawia stabilność podczas uczenia modelu, jak i zapewnia funkcję straty, która koreluje z jakością generowanych obrazów.
- Zamiast używać dyskryminatora do klasyfikowania lub przewidywania prawdopodobieństwa wygenerowanych obrazów jako prawdziwych lub fałszywych, WGAN zmienia lub zastępuje model dyskryminatora krytykiem, który ocenia prawdziwość lub fałszywość danego obrazu.

Metryka Wassersteina

- Odległość Wassersteina to minimalny koszt transportu masy w przeliczaniu rozkładu danych Q do dystrybucji danych P .
- Przykład: Celem jest przeniesienie pudełek z lewej strony na prawą. Koszt przeniesienia jest równy wadze pudełka pomnożonej przed odległość do docelowego miejsca. W przykładzie przyjmiemy, że waga każdego pudełka wynosi 1.

Przykład różnych kosztów transportu



Różnice w implementacji dla WGAN

1. Użyj liniowej funkcji aktywacji w warstwie wyjściowej modelu krytyka (zamiast sigmoidy).
2. Użyj etykiet -1 dla prawdziwych obrazów i 1 etykiet dla fałszywych obrazów (zamiast 1 i 0).
3. Użyj funkcji straty Wassersteina do trenowania modeli krytyka i generatora.
4. Ogranicz wagi modeli krytyka do ograniczonego zakresu po każdej aktualizacji wsadowej (np. [-0,01,0,01]).
5. Aktualizuj model krytyka więcej razy niż generator w każdej iteracji (np. 6).
6. Użyj wersji RMSProp metody gradientu prostego (gradient descent) z małym tempem uczenia się i bez rozpędu (np. 0,00005).

3. Funkcja straty Wassersteina

- W porównaniu do standardowego GAN w modelu WGAN jest użyta nowa funkcja straty, która zachęca dyskryminator do przewidywania wyniku tego, jak bardzo realnie lub fałszywie wyglądają dane wejściowe. WGAN przekształca rolę dyskryminatora z klasyfikatora w krytyka oceniania prawdziwości lub fałszerstwa obrazów, gdzie różnica między punktacjami jest jak największa.

Właściwości odległości Wassersteina

- Dyskryminator sieci neuronowej może zostać wytrenowany w celu przybliżenia odległości Wassersteina, a następnie wykorzystany do efektywnego wytrenowania modelu generatora.
- Odległość Wassersteina ma właściwości polegające na tym, że jest ciągła i różniczkowalna i zapewnia liniowy gradient, nawet jeśli dyskryminator jest dobrze wytrenowany.
- WGAN dąży do konwergencji, zmniejszając straty generatora.

WGAN - podsumowanie

- Zaletą sieci WGAN jest to, że proces uczenia jest bardziej stabilny i mniej wrażliwy na architekturę modelu i wybór konfiguracji hiperparametrów.
- Odległość Wassersteina można interpretować jako stopień realności obrazów wejściowych. W uczeniu ze wzmacnianiem nazywamy to **funkcją wartości**, która mierzy, jak dobry jest stan (dane wejściowe). Zmieniamy nazwę dyskryminatora na **krytyka**, aby odzwierciedlić jego nową rolę.

GAN warunkowy

- W sieciach CGAN stosowane jest ustawienie warunkowe, co oznacza, że zarówno generator, jak i dyskryminator są uzależnione od pewnego rodzaju informacji pomocniczych (takich jak etykiety klas lub dane) z innych modalności. W rezultacie idealny model może nauczyć się mapowania multymodalnego od danych wejściowych do wyjściowych, zasilany różnymi informacjami kontekstowymi.
- cGAN nie są nienadzorowanymi algorytmami uczenia się, ponieważ wymagają oznaczonych danych jako dane wejściowe do dodatkowej warstwy.

Trening CGAN

1. Generator jest sparametryzowany, aby uczyć się i tworzyć realistyczne próbki dla każdej etykiety w uczącym zestawie danych.
2. Dyskryminator uczy się rozróżniać fałszywe i prawdziwe próbki, biorąc pod uwagę informacje zawarte na etykiecie.
3. Ich role się nie zmieniają. Generator i dyskryminator generują i klasyfikują obrazy tak jak wcześniej, ale z warunkowymi informacjami pomocniczymi.

GAN vs CGAN

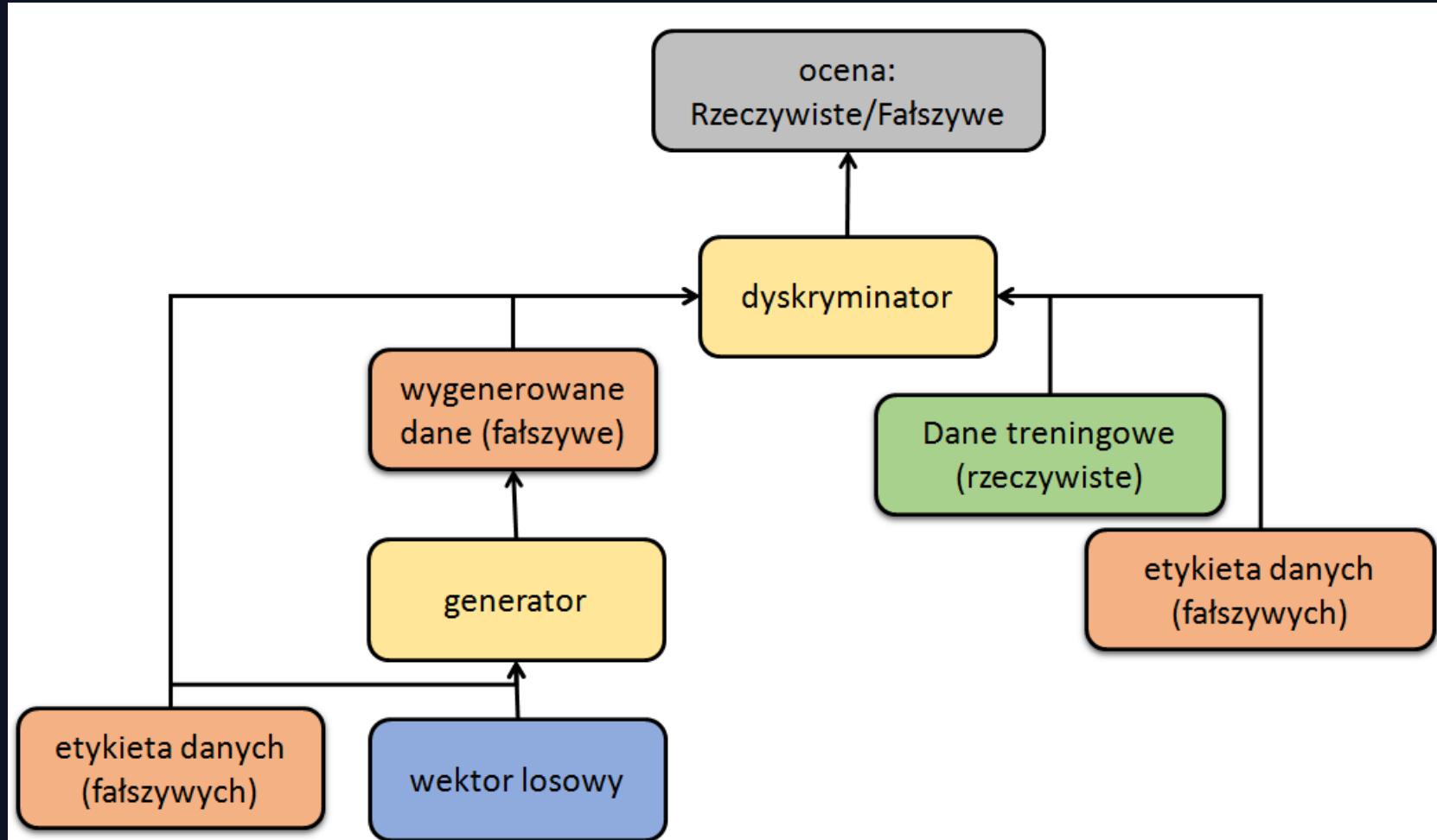
- W przypadku GAN nie można kontrolować, jakie konkretne obrazy wygeneruje generator. Oznacza to, że nie ma możliwości zażądania od generatora konkretnego obrazu cyfrowego na jego wyjściu.
- W przypadku CGAN jest inaczej, ponieważ możemy dodać dodatkową warstwę wejściową etykiet obrazu zakodowanych sposobem one-hot. Ta dodatkowa warstwa kieruje generatorem pod kątem tego, który obraz ma zostać wytworzony.

Zalety CGAN

Podając do sieci dodatkowe informacje (etykiety klas lub dane) uzyskujemy dwie korzyści:

- 1. Konwergencja będzie szybsza.**
Nawet losowa dystrybucja, którą będą kierowały się fałszywe obrazy, będzie miała pewien wzór.
- 2. Można kontrolować wyjście generatora w czasie testu**, nadając etykietę obrazowi, który chcesz wygenerować.

GAN warunkowy



Zagadka

- Mamy wyniki z kolokwium dla 1000 osób. Rozkład wyników kolokwium modelujemy za pomocą procedury:
- Rzuć trzema sześciocieinnymi kości mi.
- Pomnóż wartości z kości przez stałą „W”.
- Powtórz 100 razy i weź średnią wszystkich wyników.
- Próbujemy przypisać różne wartości dla stałej „W”, aż wynik procedury będzie równy średniej z rzeczywistych wyników kolokwium.
- Czy taki model jest modelem generatywnym czy dyskryminacyjnym?

Przykłady praktyczne (notatnik interaktywny ETI DL_10_02_Conditional_GAN.ipynb)

Implementacja warunkowej sieci typu
GAN generującej liczby ze zbioru MNIST
w środowisku
Google Colaboratory/
Jupyter Notebook

- <https://colab.research.google.com/drive/1cGdEqKnpSfw1qYlkX8Q1AbvduMVZOZA>



POLITECHNIKA
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Dziękuję

SZYMON ZAPOROWSKI



Fundusze
Europejskie
Polska Cyfrowa



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

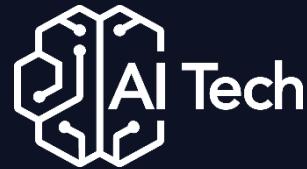
Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.



POLITECHNIKA
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Uczenie głębokie: Zastosowanie sieci GAN

Szymon Zaporowski



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Plan wykładu

- Zastosowania sieci typu GAN do generowania obiektów
- Translacja obrazów z użyciem sieci GAN
- Zastosowanie sieci GAN do wpływania na wygląd człowieka na zdjęciu
- Poprawa rozdzielczości zdjęć z zastosowaniem sieci GAN
- Łączenie zdjęć z użyciem sieci GAN
- Uzupełnianie zdjęć z zastosowaniem sieci GAN
- Predykcja wideo z wykorzystaniem GAN-ów

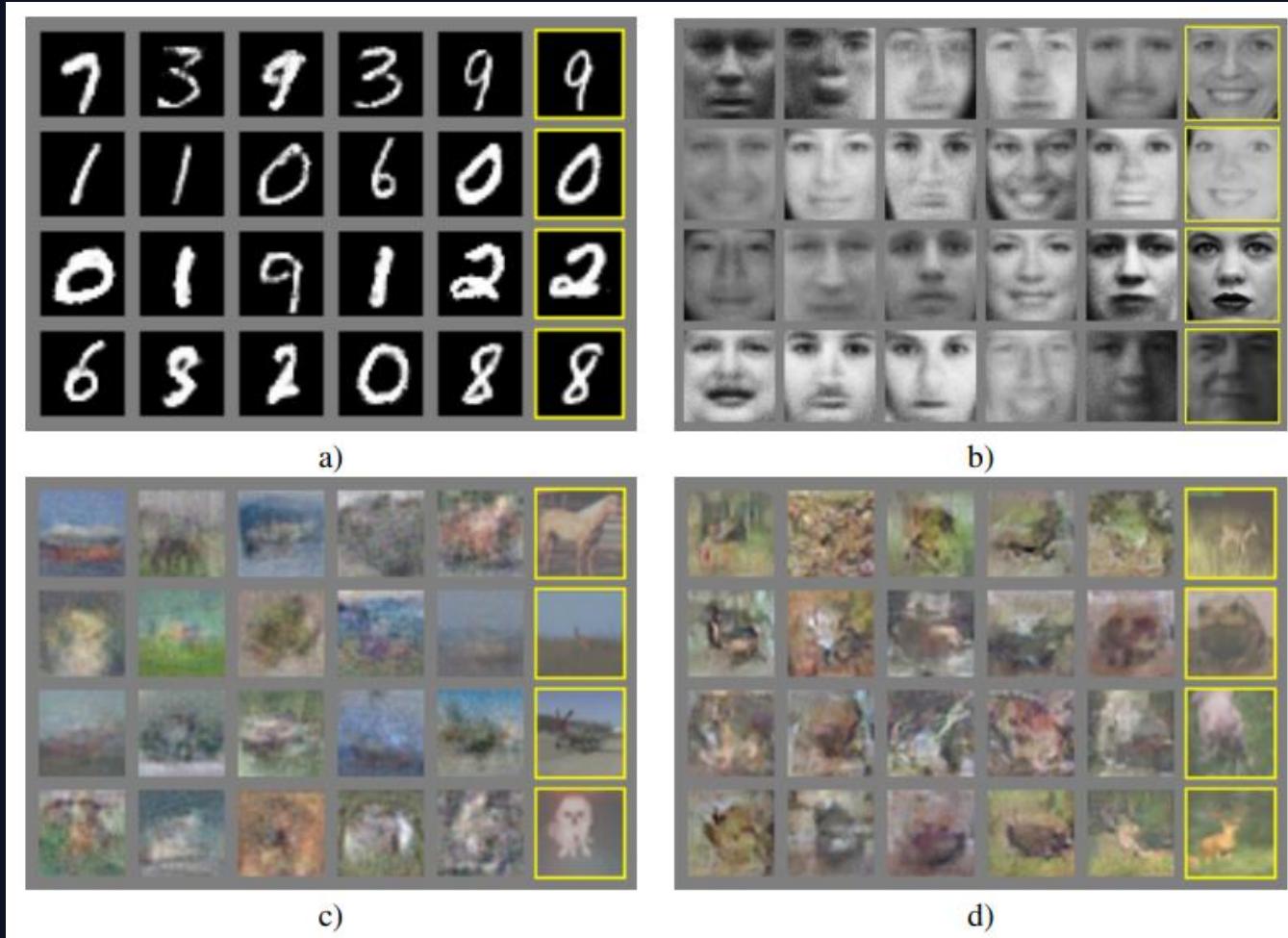
Przykładowe zastosowania sieci GAN

- Generowanie przykładów dla zbiorów danych obrazu
- Generowanie zdjęć ludzkich twarzy
- Generowanie realistycznych zdjęć
- Generowanie postaci z kreskówek
- Tłumaczenie obrazu na obraz
- CycleGAN
- Tłumaczenie tekstu na obraz
- Tłumaczenie semantyczne z obrazu na zdjęcie
- Generowanie widoku z przodu twarzy
- Generowanie nowych pozycji ludzkich
- Generowanie Emojis ze zdjęć
- Edycja zdjęć
- Postarzanie twarzy
- Łączenie kilku zdjęć w jedno
- Poprawa rozdzielczości zdjęć
- Uzupełnianie zdjęć
- Nanoszenie tekstury na odzież
- Predykcja wideo
- Generowanie obiektów 3D
- Pizza GAN

Generowanie przykładów dla zbiorów danych z obrazami

- „*Generative Adversarial Networks*”
- Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio
- Jednoczesne trenowanie dwóch modeli: modelu generatywnego G, który wychwytuje rozkład danych, oraz modelu dyskryminacyjnego D, który szacuje prawdopodobieństwo, że próbka pochodzi z danych treningowych, a nie z modelu G. Procedura ucząca dla G polega na maksymalizacji prawdopodobieństwa popełnienia błędu przez D.

Wizualizacja próbek z modelu

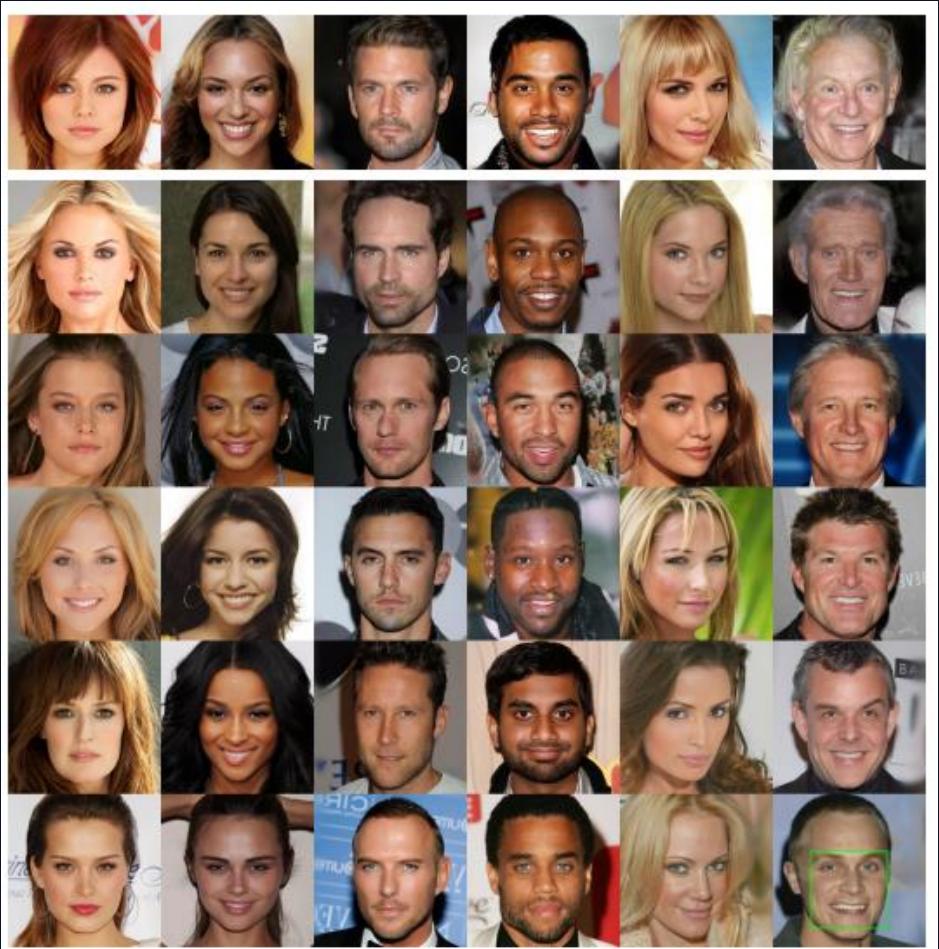


Źródło: <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>

Generowanie zdjęć ludzkich twarzy

- „*Progressive Growing of GANs for Improved Quality, Stability, and Variation*”
- Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen
- Kluczową ideą jest stopniowe rozwijanie zarówno generatora, jak i dyskryminatora: zaczynając od niskiej rozdzielczości, dodawane są nowe warstwy, które w miarę postępu treningu modelują coraz dokładniejsze szczegóły. To zarówno przyśpiesza trening, jak i znacznie go stabilizuje, pozwalając tworzyć obrazy o bardzo dobrej jakości.
- Wygenerowane twarzy zostały wytrenowane na zdjęciach celebrytów, oznacza to, że w wygenerowanych twarzach znajdują się elementy istniejących celebrytów, przez co wydają się znajome, ale nie do końca.

Przykład wygenerowanych ludzkich twarzy

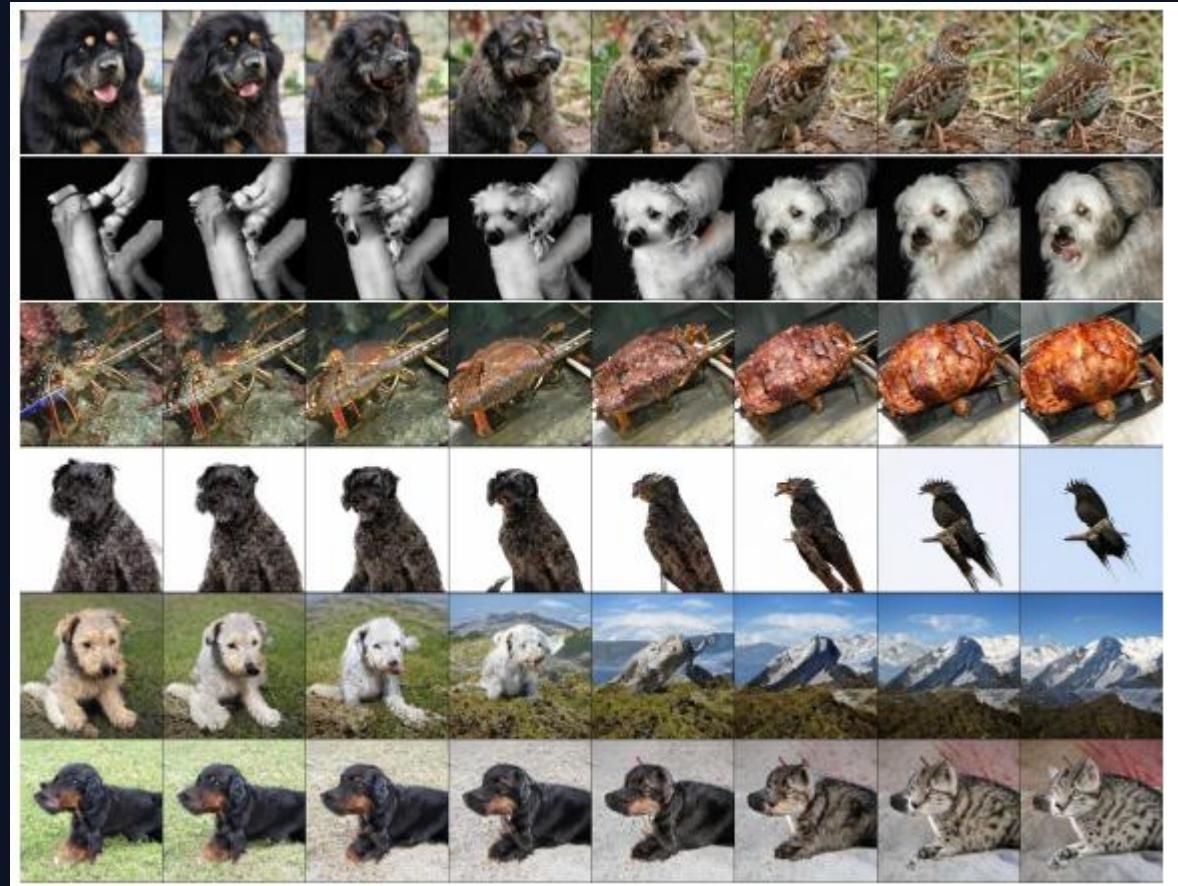


Źródło: https://research.nvidia.com/sites/default/files/pubs/2017-10_Progressive-Growing-of/karras2018iclr-paper.pdf

Generowanie realistycznych zdjęć

- „*Large Scale GAN Training for High Fidelity Natural Image Synthesis*”
- Andrew Brock, Jeff Donahue, Karen Simonyan
- W modelu BigGAN (opisany w artykule) użyto architektury ResNet GAN, ale ze zmodyfikowanym wzorcem. Zastosowano hierarchiczne ukryte przestrzenie, dzięki czemu ukryty wektor jest dzielony na kawałki o równym rozmiarze.
- Autorzy twierdzą, że mogą radykalnie poprawić stan techniki i trenować modele do 512×512 pikseli rozdzielczości bez potrzeby stosowania jawnych metod wieloskalowych.
- Klasy takie jak psy, które są w dużej mierze teksturowe i powszechnie w zbiorze danych, są znacznie łatwiejsze do modelowania niż klasy obejmujące niewyrównane ludzkie twarze lub tłumy.

Przykłady generowanych obrazów



Źródło: <https://openreview.net/pdf?id=B1xsqj09Fm>

Generowanie postaci z kreskówek

- „*Towards the Automatic Anime Characters Creation with Generative Adversarial Networks*”
- Yanghua Jin, Jiakai Zhang, Minjun Li, Yingtao Tian, Huachun Zhu, Zhihao Fang
- Profesjonalny twórca może skorzystać z zalet automatycznego generowania do inspiracji animacji i projektowania postaci do gier. Programista RPG może wykorzystywać wolne od praw autorskich obrazy twarzy, aby zmniejszyć koszty projektowania w produkcji gier.
- Trenowanie sieci GAN odbyło się z obrazów bez tagów, które można wykorzystać jako ogólne podejście do treningu nadzorowanego lub warunkowego model bez danych tagów.

Wygenerowane postacie anime



MakeGirlsMoe Home History Transition Help English Twitter Discord

Options Advanced Mode

Model: Amaryllis 128x128 Ver.170716 (3.8MB)

Hair Color: Pink | Hair Style: Random | Eye Color: Aqua

Blush: Off | Random | On | Smile: Off | Random | On | Open Mouth: Off | Random | On

Hat: Off | Random | On | Ribbon: Off | Random | On | Glasses: Off | Random | On

Noise: Random | Fixed | Current Noise: | Noise Import/Export: Import | Export

Operations: Import | Export | Reset | WebGL Acceleration: Disabled | Enabled

Generate

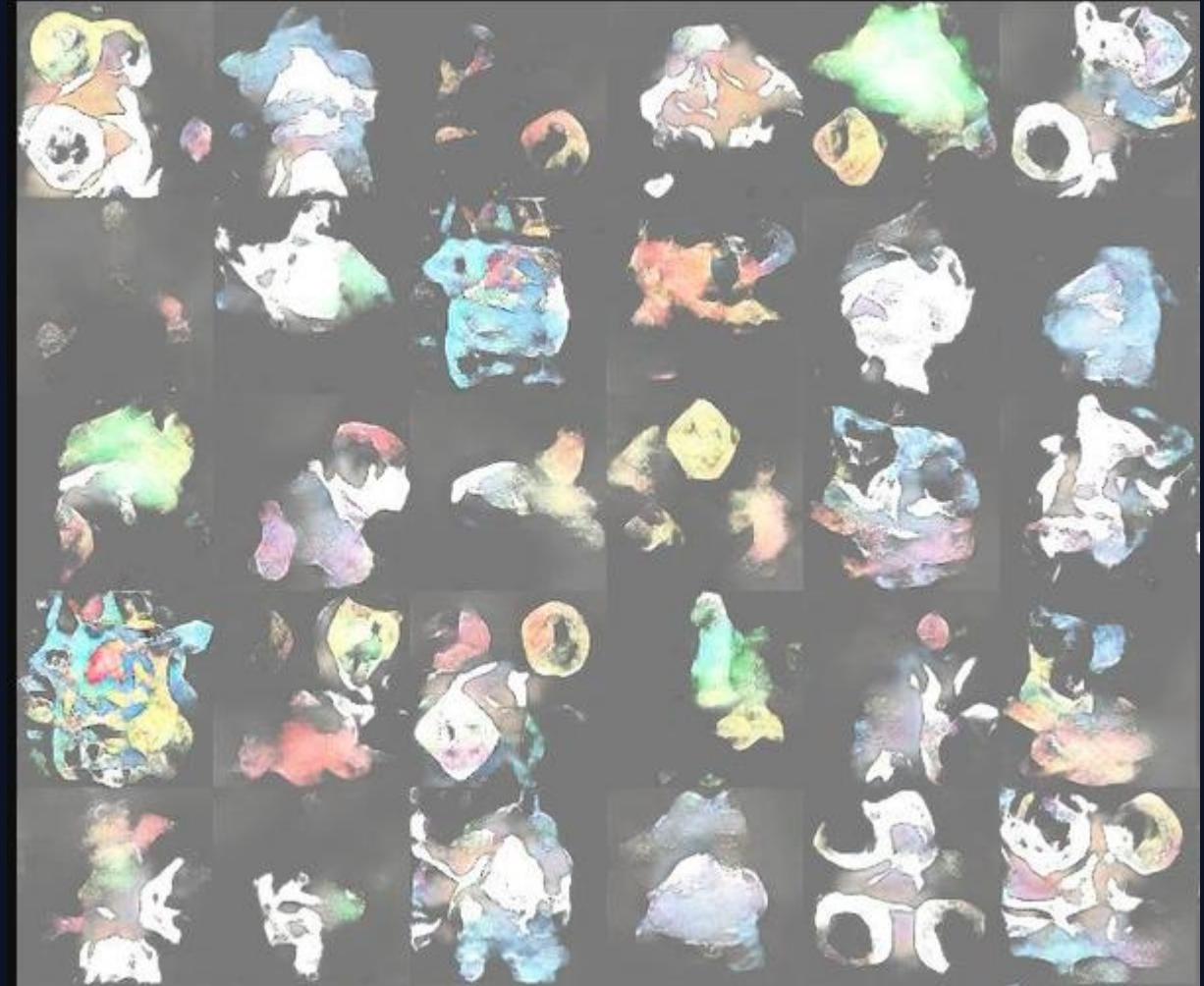
Share on Twitter

The screenshot shows the MakeGirlsMoe web interface. It includes a navigation bar at the top with links for Home, History, Transition, Help, language (English), social media (Twitter, Discord), and a logo. Below the navigation is a title "Options" with an "Advanced Mode" checkbox. A preview window shows a portrait of a character with pink hair and blue eyes. To the right are several rows of sliders and dropdown menus for generating characters, including options for hair color, style, eye color, blush, smile, open mouth, hat, ribbon, glasses, noise, and operations like import/export and WebGL acceleration. Buttons for "Generate" and "Share on Twitter" are also present.

Źródło: https://www.researchgate.net/publication/319187018_Towards_the_Automatic_Anime_Characters_Creation_with_Generative_Adversarial_Networks
<https://make.girls.moe/#/>

Generowanie postaci z kreskówek - pokeGAN

- Próby stworzenia GAN do tworzenia pokemonów nie osiągają tak dobrych rezultatów jak anime GAN.
- Jednak skrypty dostępne są na stronie **github**.



Źródło: <https://github.com/moxiegushi/pokeGAN>
<https://github.com/kvpratama/gan/tree/master/pokemon>

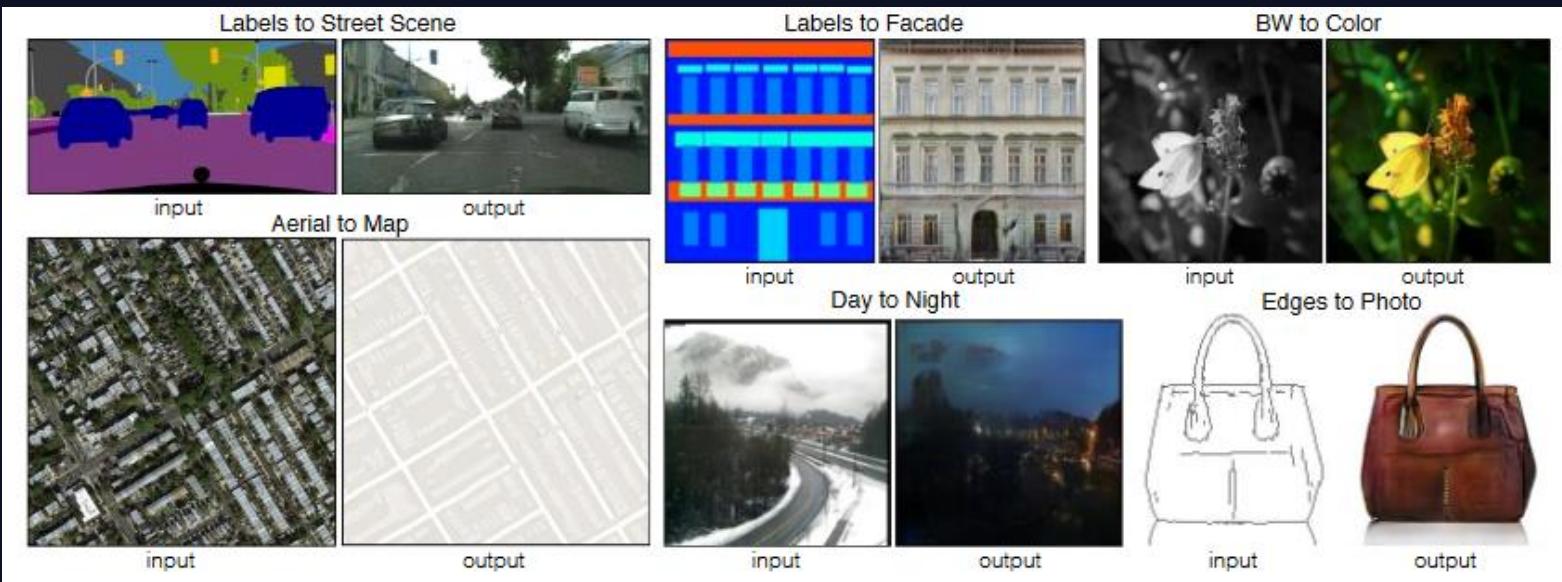
Translacja obrazu na obraz (pix2pix)

- „*Image-to-Image Translation with Conditional Adversarial Networks*”
- Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros

Przykładami takich translacji mogą być:

- Etykiety semantyczne \leftrightarrow zdjęcie
- Etykiety architektoniczne \rightarrow zdjęcie
- Mapa \leftrightarrow zdjęcie lotnicze , wykorzystano Mapy Google
- Czarno -białe zdjęcia \rightarrow zdjęcia kolorowe
- Same krawędzie \rightarrow zdjęcie
- Szkic \rightarrow zdjęcie
- Zdjęcie wykonane za dnia \rightarrow zdjęcie nocne
- Termiczne zdjęcia \rightarrow kolorowe zdjęcia
- Zdjęcie z brakującymi pikselami \rightarrow całe zdjęcie

Przykłady translacji obrazu (pix2pix)



Źródło: https://www.researchgate.net/publication/320966887_Image-to-Image_Translation_with_Conditional_Adversarial_Networks
Kod sieci jest dostępny pod adresem:
<https://github.com/phillipi/pix2pix>

Cycle GAN – translacja obrazu na obraz

- „*Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*”
- Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros

Przykłady translacji:

1. od fotografii do malarstwa artystycznego
2. malarstwa na fotografię
3. konia na zebre
4. fotografii z lata na zimę
5. zdjęcia satelitarnego na widok Google Maps
6. szkicu na fotografię
7. jabłek na pomarańcze.

Przykłady translacji wykorzystując Cycle GAN



Źródło: https://openaccess.thecvf.com/content_ICCV_2017/papers/Zhu_Unpaired_Image-To-Image_Translation_ICCV_2017_paper.pdf

Translacja tekstu na obraz (text2image)

- „*StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks*”
- Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas
- StackGAN - do generowania 256x256 fotorealistycznych obrazów uwarunkowanych opisami tekstowymi.

Generowanie obrazów podzielone jest na dwie części:

1. GAN Stage-I szkicuje prymitywny kształt i kolory obiektu w oparciu o podany opis tekstowy, dając obrazy o niskiej rozdzielczości.
2. GAN Stage-II pobiera wyniki Stage-I i opisy tekstowe jako dane wejściowe i generuje obrazy o wysokiej rozdzielczości z fotorealistycznymi szczegółami.

Translacja tekstu na obraz - przykład

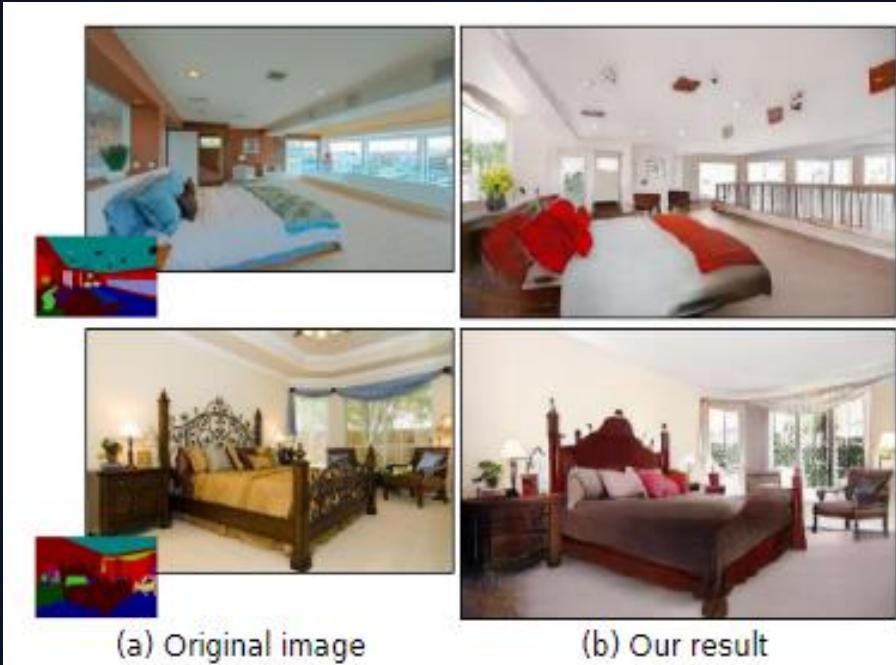
Text description	This bird is blue with white and has a very short beak	This bird has wings that are brown and has a yellow belly	A white bird with a black crown and yellow beak	This bird is white, black, and brown in color, with a brown beak	The bird has small beak, with reddish brown crown and gray belly	This is a small, black bird with a white breast and white on the wingbars.	This bird is white black and yellow in color, with a short black beak
Stage-I images							
Stage-II images							

Źródło: https://openaccess.thecvf.com/content_ICCV_2017/papers/Zhang_StackGAN_Text_to_ICCV_2017_paper.pdf

Translacja semantyczna z obrazu na zdjęcie

- „*High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs*”
- Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, Bryan Catanzaro
- Opisano nową metodę syntezy fotorealistycznych obrazów o wysokiej rozdzielczości z semantycznych map etykiet przy użyciu warunkowych generatywnych sieci kontradyktoryjnych (GAN).
- Generowane obrazy są wymiarów: 2048x1024.
- Stworzono interfejs, który przedstawia działanie sieci, umożliwia modyfikacje stylu, przesuwanie obiektów, usuwanie obiektów.

Translacja semantyczna z obrazu na zdjęcie - przykład



Źródło: https://www.researchgate.net/publication/329743810_High-Resolution_Image_Synthesis_and_Semantic_Manipulation_with_Conditional_GANs

Generowanie widoku z przodu twarzy

- „*Beyond Face Rotation: Global and Local Perception GAN for Photorealistic and Identity Preserving Frontal View Synthesis*”
- Rui Huang, Shu Zhang, Tianyu Li, Ran He
- Użyto GAN do generowania zdjęć twarzy z widokiem z przodu (na wprost twarzy) na podstawie zdjęć zrobionych pod kątem. Celem było wygenerowanie zdjęcia na wprost twarzy aby można było następnie wykorzystać je jako dane wejściowe do systemu weryfikacji lub identyfikacji twarzy.

Generowanie widoku z przodu twarzy - przykład

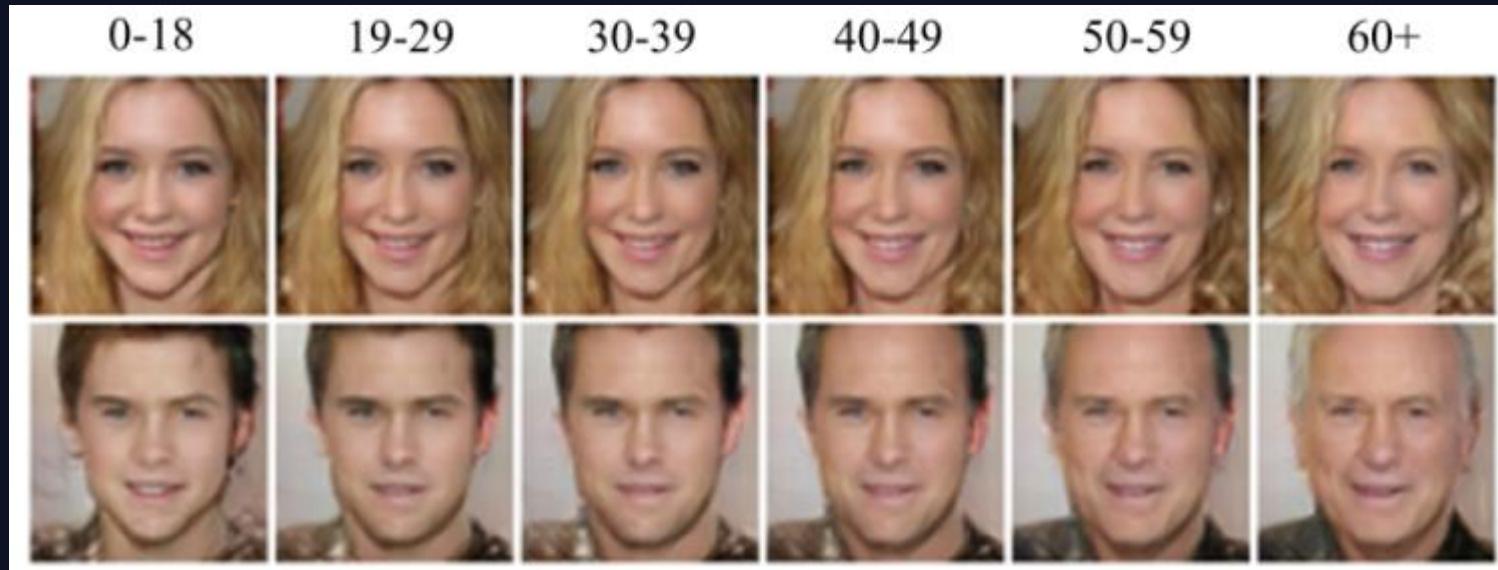


Źródło: http://gwylab.com/pdf/face_rotation.pdf

Postarzanie twarzy

- „*Face Aging With Identity-Preserved Conditional Generative Adversarial Networks*”
- Zongwei Wang, Xu Tang, Weixin Luo, Shenghua Gao
- „*Face Aging With Conditional Generative Adversarial Networks*”
- Grigory Antipov, Moez Baccouche, Jean-Luc Dugelay
- Szczególny nacisk kładziony był na zachowaniu tożsamości pierwotnej osoby w starzejcej się wersji jej twarzy. Wprowadzono nowatorskie podejście do optymalizacji „zachowania tożsamości” ukrytych wektorów GAN.
- Zaprojektowano acGAN (Age Conditional GAN).

Postarzanie twarzy - przykład



Źródło: <https://www.arxiv-vanity.com/papers/1702.01983/>

Łączenie kilku zdjęć w jedno

- „*GP-GAN: Towards Realistic High-Resolution Image Blending*”
- Huikai Wu, Shuai Zheng, Junge Zhang, Kaiqi Huang
- Artykuł opisuje sposób na łączenie obrazów wykorzystujący połączenie modeli generatywnych GAN oraz podejścia opartego na gradientach.
- Celem była integracja metod, aby uzyskać dobre aspekty z obu metod i uniknąć ich wad.

Łączenie kilku zdjęć w jedno - przykłady



(a) Source Image



(d) Composite Image A



(f) Composite Image B

Poprawa rozdzielczości zdjęć

- „*Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*”
- Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi
- Wykorzystanie SRGAN – SuperResolution GAN. Jest to pierwsza opisane struktura pozwalająca na wygenerowanie fotorealistycznych obrazów naturalnych dla współczynników skalowania 4x.

Poprawa rozdzielczości zdjęć - przykład

Źródło: [https://arxiv.org
/pdf/1609.04802.pdf](https://arxiv.org/pdf/1609.04802.pdf)



Uzupełnianie zdjęć

- „*Context Encoders: Feature Learning by Inpainting*”
- Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, Alexei A. Efros
- Autorzy opisują użycie GAN, w szczególności koderów kontekstowych, do wykonywania zamalowywania fotografii lub wypełniania otworów, czyli wypełniania obszaru zdjęcia, który z jakiegoś powodu został usunięty.
- Stworzono nienadzorowany algorytm uczenia funkcji wizualnych oparty na przewidywaniu pikseli w kontekście.
- Zaproponowano Enkodery Kontekstowe — splotową sieć neuronową wytrenowaną do generowania zawartości dowolnego regionu obrazu uwarunkowanego jego otoczeniem.

Uzupełnianie zdjęć - przykład

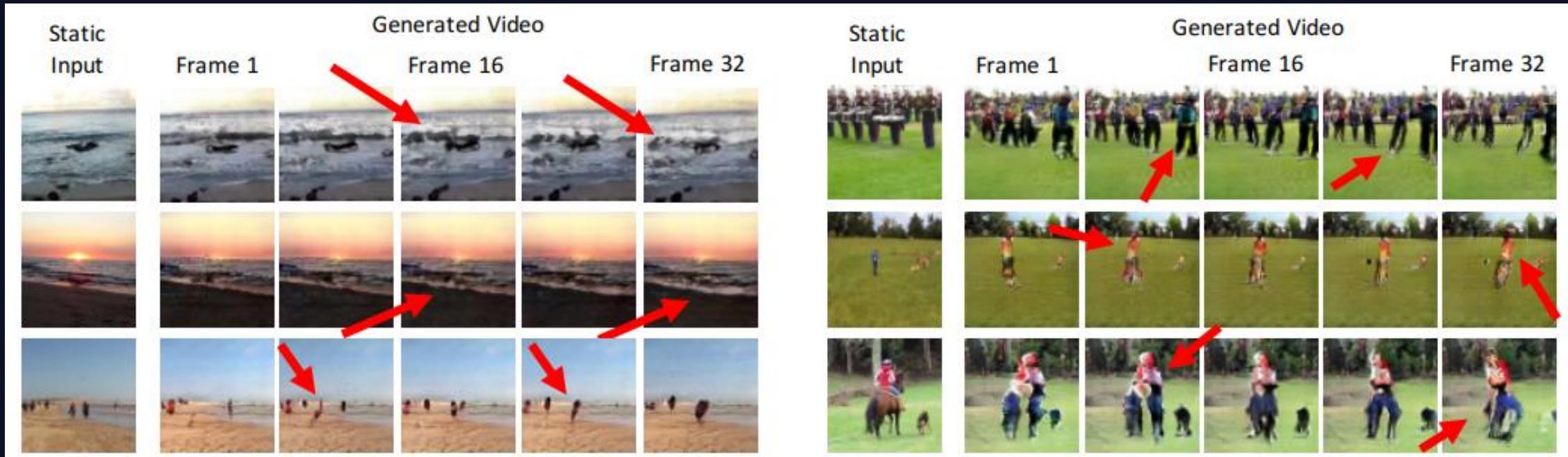


Źródło: [https://openaccess.thecvf.com
/content_cvpr_2016/papers/Pathak_Context_Encoders_Feature_CVPR_2016_paper.pdf](https://openaccess.thecvf.com/content_cvpr_2016/papers/Pathak_Context_Encoders_Feature_CVPR_2016_paper.pdf)

Predykcja wideo

- „*Generating Videos with Scene Dynamics*”
- Carl Vondrick, Hamed Pirsiavash, Antonio Torralba
- W artykule opisano jak użyto GAN do przewidywania wideo, w szczególności przewidywania z powodzeniem do jednej sekundy klatek wideo przy pełnej liczbie klatek.
- Wykorzystano generatywną sieć kontradiktoryjności dla wideo z czasoprzestrzenną, splotową architekturą, która rozplatuje pierwszy plan sceny z tła.
- Model wewnętrznie uczy się przydatnych funkcji rozpoznawania działań przy minimalnym nadzorze.

Predykcja wideo - przykład



Źródło: <https://proceedings.neurips.cc/paper/2016/file/04025959b191f8f9de3f924f094051f-Paper.pdf>

Dziękuję

SZYMON ZAPOROWSKI



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.



POLITECHNIKA
GDAŃSKA



WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI
I INFORMATYKI

Uczenie Głębokie UCZENIE ZE WZMOCNIENIEM

Adam Kurowski

Katedra Systemów Multimedialnych,
Wydział Elektroniki, Telekomunikacji i Informatyki PG



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Wprowadzenie

Wśród wielu poddziedzin uczenia maszynowego wydziela się **trzy główne paradygmaty** (najbardziej popularne spośród wielu możliwych do wyróżnienia). Są to uczenie **nadzorowane**, uczenie **nienadzorowane** oraz uczenie **wzmacniane (ang. reinforcement learning)**.

Uczenie nadzorowane i nienadzorowane wymaga przetworzenia przez algorytm **przygotowanych wcześniej zbiorów danych**. W zależności od tego, czy uczenie jest nadzorowane czy nie dane te mogą być przetwarzane razem z towarzyszącymi etykietami. Istnieje ustalony zbiór przykładów.

Taka sytuacja nie ma miejsca w przypadku **uczenia wzmacnianego**. Jest to przypadek w którym **nie dysponujemy z góry ustalonym zbiorem danych**.

Zamiast tego pozyskujemy je poprzez **podejmowanie decyzji** przez algorytm uczenia wzmacnianego i **obserwowanie skutków tych decyzji**, co pozwala na podejmowanie coraz lepszych decyzji.

Uczenie wzmacniane – podstawowe pojęcia

Centralną ideą realizowaną przez algorytmy uczenia wzmacnianego jest interakcja z tzw. **środowiskiem**. Zwyczajowo jednostkę, która wchodzi w interakcję ze środowiskiem nazywa się **agentem**.

Interakcja agenta ze środowiskiem wymaga **podejmowania decyzji**, które **następnie wywołują skutki** na podstawie których agent może uczyć się podejmowania lepszych decyzji.

Właśnie stąd pochodzi nazwa całego paradygmatu, czyli uczenie wzmacniane lub uczenie przez wzmacnianie (w domyśle – najbardziej korzystnych zachowań agenta).

Jest to proces **luźno zainspirowany zdobywaniem doświadczenia np. przez ludzi** wchodzących w interakcję z nowym, nieznanym im przedmiotem.

Uczenie wzmacniane – podstawowe pojęcia

Każda decyzja podjęta w kontekście zadanego środowiska z ramach którego operuje algorytm uczenia wzmacnianego skutkuje **informacją zwrotną** o tym, jak dobra była podjęta przed chwilą decyzja.

Informacja zwrotna przekazywana jest zwykle w postaci tak zwanego **sygnału nagrody**. Jest to **wartość liczbową**, która np. może być **dodatnia** w momencie wykonania przez agenta **korzystnych** i pożądanych akcji, a **ujemna** w momencie wykonywania akcji **niechcianych**.

Informacja płynąca z sygnału nagrody jest wykorzystywana do wzmacniania i **zachęcania agenta do podejmowania decyzji korzystnych** z jego punktu widzenia.

Trening algorytmu uczenia wzmacnianego ma na celu znalezienie dla agenta **optimalnej strategii postępowania**.

Matematyczny opis uczenia wzmacnianego

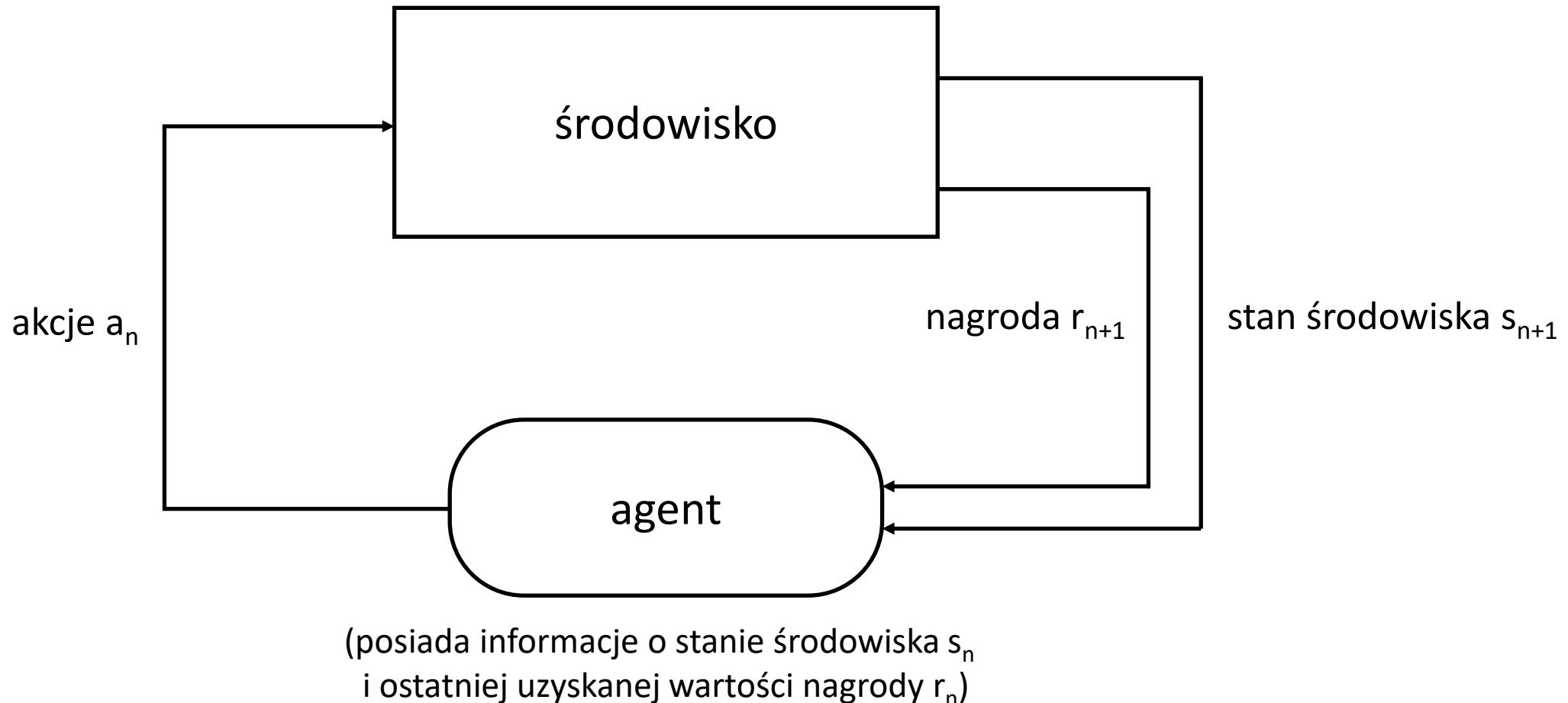
Często przyjmuje się, że proces decyzyjny, który przeprowadzany jest agenta ma **charakter markowski**. Oznacza to, że na stan **następujący** po decyzji podjętej przez agenta ma wpływ tylko i **wyłącznie stan poprzedni oraz sama decyzja**.

Proces tego typu ma charakter **etapowy**, każdy krok można oznaczyć dyskretnym indeksem. Zwyczajowo oznaczony jest on przez literę **n** .

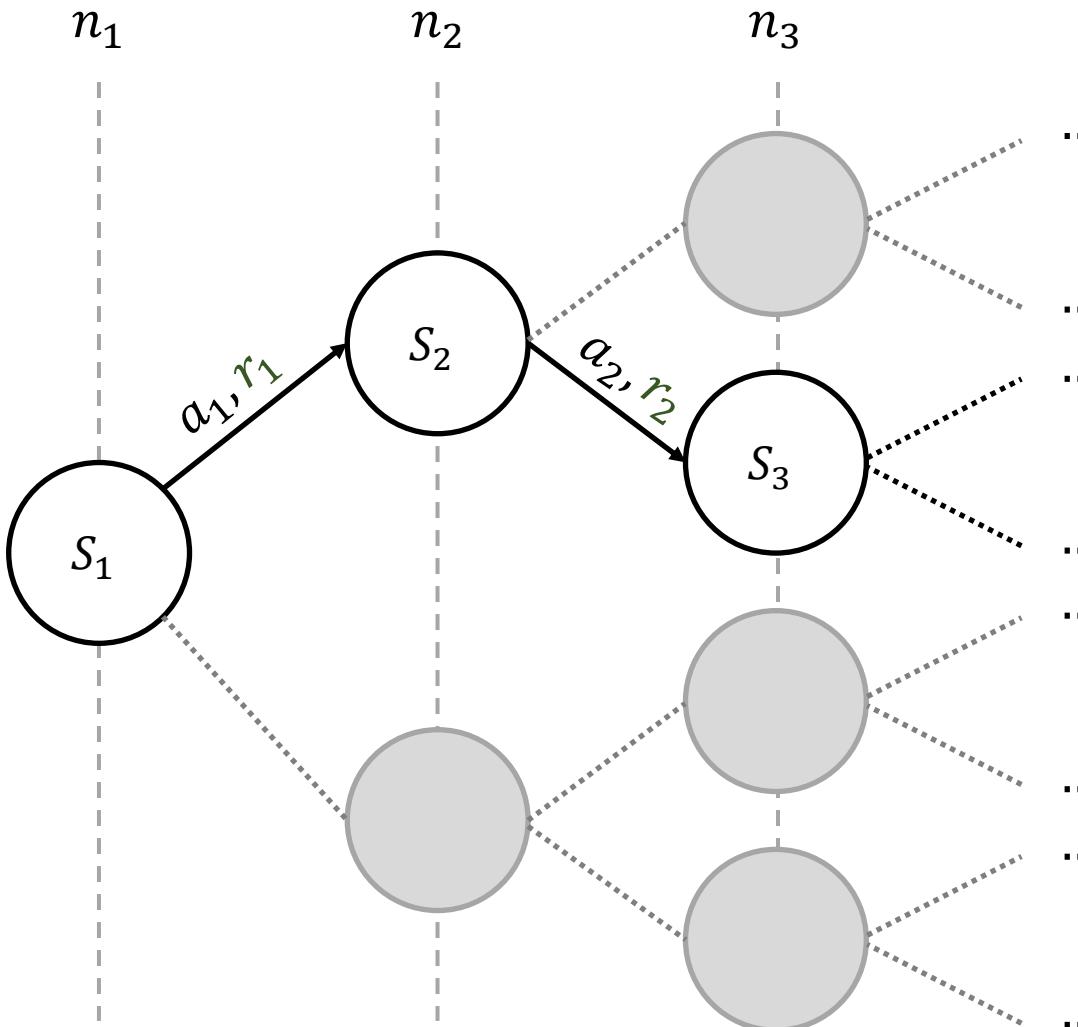
Z tego powodu agent w swoich decyzjach musi uwzględniać jedynie **obecny stan środowiska**. Przez stan rozumiemy wszystkie informacje o środowisku, jakie w danym n -tym kroku posiada agent i zwyczajowo **oznaczamy jako S_n** .

Akcję powodującą przejście ze stanu S_n do S_{n+1} **oznaczamy jako a_n** . Podjęcie akcji a_n oprócz przejścia do stanu S_{n+1} powoduje **przydzielenie agentowi nagrody r_n** .

Matematyczny opis uczenia wzmacnianego



Matematyczny opis uczenia wzmacnianego



Ilustracja realizacji procesu decyzyjnego za pomocą wykresu kratowego.

Matematyczny opis uczenia wzmacnianego

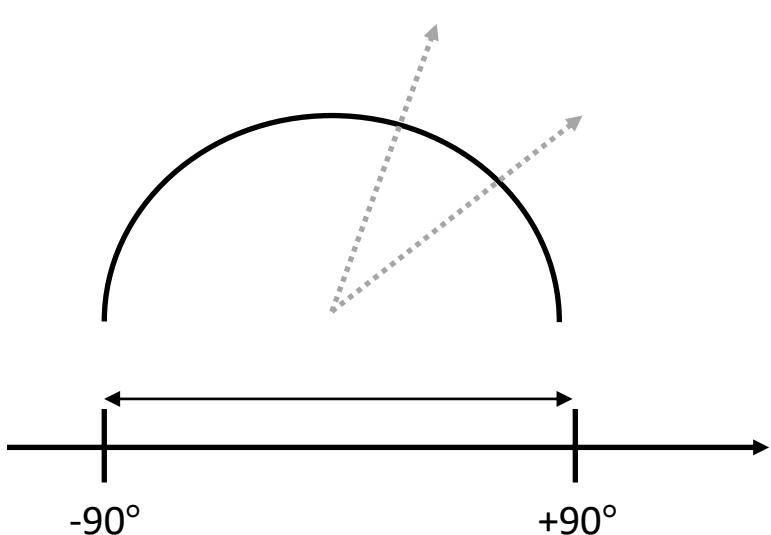
Zbiór wszystkich możliwych do podjęcia akcji oznaczamy jako A , każda akcja $a_n \in A$ i nazywamy **przestrzenią akcji** (ang. action space).

W rozważaniach dotyczących klasycznych metod uczenia ze wzmacnieniem ograniczymy się do **dyskretnych przestrzeni akcji**, jednak w ogólności istnieją także problemy w których przestrzeń ta jest **ciągła**.

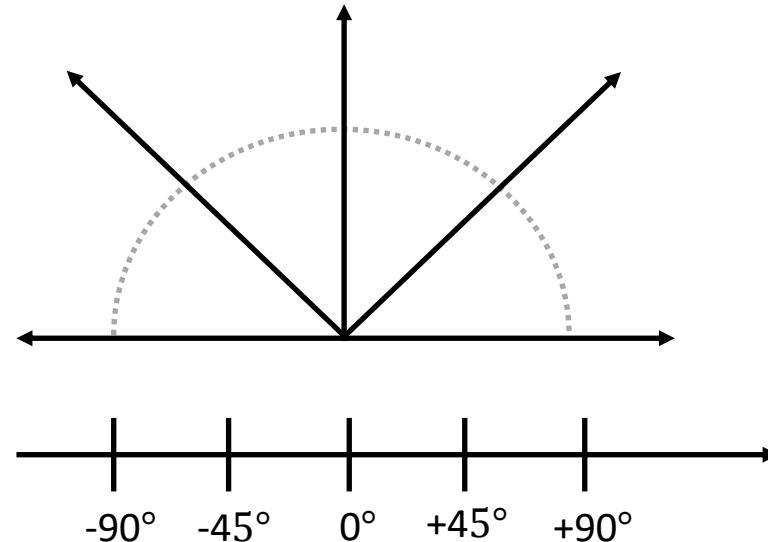
Jeżeli chcemy podejść w **klasyczny sposób** do problemu w którym **przestrzeń akcji jest ciągła**, to **popularnym rozwiązaniem jest dyskretyzacja** przestrzeni akcji poprzez podzielenie jej na podprzedziały.

Działanie takie wymaga **umiejętnego dobrania gęstości dyskretyzacji**, gdyż z jednej strony – **zbyt zgrubny** podział spowoduje że algorytm będzie **niedokładny**. Z drugiej strony **zbyt gęsty** podział **utrudni wytrenowanie** algorytmu.

Matematyczny opis uczenia wzmacnianego



ciągła przestrzeń akcji A



zdyskretyzowana przestrzeń akcji A

Przykład ciągłej i zdyskretyzowanej przestrzeni akcji dla algorytmu wyznaczającego kierunek w którym ma poruszać się pojazd sterowany przez agenta.

Matematyczny opis uczenia wzmacnianego

Agent podejmując akcje wpływa na stan środowiska. **Zwyczajowo interakcja ta kończy się** albo po spełnieniu **arbitralnego kryterium** – np. po osiągnięciu **limitu iteracji** lub zdobyciu progowej wartości **skumulowanej nagrody**.

Możliwe jest także zakończenie interakcji w momencie przejścia do specjalnego stanu, do którego **przejście jest równoznaczne z zakończeniem interakcji ze środowiskiem**.

Niezależnie od przyczyny zakończenia interakcji ze środowiskiem, stany środowiska w których interakcja ta jest zakończona nazywane są **stanami terminalnymi**.

Realizację procesu decyzyjnego (sekwencję stanów, akcji i nagród) od stanu **początkowego** aż do stanu **końcowego** nazywamy **epizodem**.

Definicja środowiska

W dalszych rozważaniach będziemy przyjmować, że stan środowiska S_n w pełni opisuje środowisko w danym czasie – jest to tzw. **środowisko w pełni obserwowalne** (*ang. fully observable environment*). Przykładem procesu decyzyjnego w takim środowisku jest gra w szachy w której zawsze znamy układ figur na szachownicy.

Warto zaznaczyć, że istnieją także środowiska tylko **częściowo obserwowalne** (*ang. partially observable environments*).

Z użytkowego punktu widzenia jako **środowisko rozumieć będziemy odwzorowanie problemu, który agent ma rozwiązać**. Może być to **symulacja** tego problemu (np. w grze komputerowej) lub **fizyczny obiekt** (np. droga na której nawiguje samochód autonomiczny).

Przykłady środowisk uczenia wzmacnianego

Problemu (i środowiska) nadające się do wykorzystania w algorytmie bazującym na uczeniu wzmacnianym są bardzo różnorodne. Przykładowo mogą to być:

- Różnego rodzaju **gry** – od tych najprostszych (np. warcaby, szachy) po te bardzo złożone (np. komputerowe strategie czasu rzeczywistego). W tym przypadku bardzo często jako **sygnal nagrody da się wykorzystać punkty zdobywane przez graczy**, agent będzie usiłował maksymalizować wyniki punktowe.
- Problem **sterowania pojazdem na drodze** – algorytm uczenia wzmacnianego może posłużyć jako podstawa do realizacji sterowników pojazdów autonomicznych (trenowanych zwykle w **symulacji**). Tu sygnałem nagrody mogą być **punkty za utrzymywanie dystansu od innych pojazdów** i dotarcie do wyznaczonego celu. Dodatkowo możliwe jest także stosowanie **ujemnej wartości** nagrody za **stłuczki** czy za **przekroczenie maksymalnego czasu** na dotarcie do celu.

Przykłady środowisk uczenia wzmacnianego

- Zadanie **zaprojektowania przedmiotu** spełniającego zadane kryteria, którego własności można przewidzieć za pomocą symulacji komputerowej. Może to być na przykład antena radiowa. Nagrodą dla agenta w takim przypadku jest **zmiana wartości parametru, który powinien być zmaksymalizowany**. Dla anteny może być to zysk kierunkowy. Agent w trakcie treningu będzie uczył się podejmować akcje prowadzące do zwiększenia zysku kierunkowego takiej anteny.
- System **rekomendujący treści multimedialne** – użytkownicy takiego systemu otrzymują wygenerowane przez agenta sugestie. Mogą to być np. polecenia filmów, muzyki, tekstów do przeczytania. **Sygnalem nagrody może w takim przypadku być ocena treści** (np. w skali od 1 do 5) rekomendowanych przez agenta.

Zwrot i nagrody odroczone w czasie

Na podstawie wartości nagrody możliwe jest **obliczenie tego jak bardzo opłaca się podjąć konkretną akcję a_n w stanie S_n** . Konieczne jest jednak odpowiednie przetworzenie cząstkowych wartości nagrody r_n .

Konieczne jest tutaj przeanalizowanie nie tylko natychmiastowej wartości nagrody r_n , jaką agent otrzymuje tuż po wykonaniu akcji a_n . Konieczne jest także wzięcie pod uwagę **nagród, jakie agent potencjalnie otrzyma w przyszłości**.

W taki sposób otrzymać można wartość nagrody skumulowanej nazywaną **zwrotem (oznaczanej przez G_n)**. Wartość zwrotu nagrody **w danym epizodzie** można obliczyć w najprostszym przypadku jako sumę nagród cząstkowych:

$$G_n = r_1 + r_2 + r_3 + \cdots + r_N = \sum_{n=1}^N r_n$$

Zwrot i nagrody odroczone w czasie

Równoważne traktowanie nagród w kroku bieżącym, jak i w krokach odległych w czasie może nie być optymalnym podejściem do treningu agenta.

Z tego względu przydatne staje się wprowadzenie tzw. współczynnika dyskontowania (oznaczanego przez γ). Wzór na G_n przyjmuje w takim przypadku następującą postać:

$$G_n = r_1 + \gamma \cdot r_2 + \gamma^2 \cdot r_2 + \gamma^3 \cdot r_3 + \dots + \gamma^N \cdot r_N = \sum_{n=1}^N \gamma^{n-1} \cdot r_n .$$

Wartość współczynnika γ zawiera się w przedziale od 0 do 1. Dzięki temu pozwala on regulować stopień czułości wartości G na potencjalne przyszłe nagrody uzyskane przez agenta.

Zwrot i nagrody odroczone w czasie

W swoim działaniu agent będzie dążył do maksymalizacji zwrotu G . Przy doborze wartości współczynnika dyskontowania γ warto pamiętać, że:

- dla **niskich wartości γ (blisko zera)** agent będzie bardziej kładł nacisk na nagrody otrzymywane natychmiastowo. Będzie podejmował akcje przynoszące szybki zysk i nie będzie brał pod uwagę potencjalnych przyszłych nagród; działania agenta będą miały charakter **krótkoterminowy**,
- dla **wysokich wartości γ** agent będzie **premiaował odroczoną gratyfikację** i jego działania będą miały charakter strategii **długoterminowej**, obliczonej na uzyskanie wysokiej nagrody w przyszłości.

Jeśli posługujemy się wartościami zwrotu G obliczonymi przy wartościach współczynnika γ większych od 0 to wartość tę można też nazwać **zdyskontowaną skumulowaną nagrodą**.

Zwrot i nagrody odroczone w czasie

Bazując na pojęciu zdyskontowanej skumulowanej wartości nagrody możliwe jest wyprowadzenie wyrażenia pozwalającego na szacowanie zwrotu G_n na podstawie zwrotu z kroku następnego G_{n+1} i współczynnika dyskontowania γ :

$$G_n = \sum_{n=1}^N \gamma^{n-1} \cdot r_n = r_1 + \sum_{n=2}^N \gamma^{n-1} \cdot r_n = r_1 + \gamma \cdot \sum_{n=1}^N \gamma^{n-1} \cdot r_n = r_1 + \gamma \cdot G_{n+1}$$

Jak później zobaczymy, tego typu struktura równania, w którym, współczynnik dyskontowania pozwala powiązać wielkość związaną ze stanem n z analogiczną wielkością ze stanu $n + 1$ pojawia się we wzorach powiązanych z uczeniem wzmacnianym dużo częściej.

Wartość stanu środowiska

Wychodząc z definicji zwrotu G_n możliwe jest zdefiniowanie tzw. **wartości stanu (ang. value)**, która oznaczana jest przez $V(S_n) = V_n$.

Wartość G_n odpowiada skumulowanej, zdyskontowanej wartości nagrody **w pojedynczym epizodzie**. Z tego względu dla różnych epizodów wartości G_n tych epizodów będą różne, bo jest ona **zmienną losową**.

Możemy zatem wartość **G_n uśrednić po wielu epizodach**. Jeśli uśrednimy wszystkie możliwe zwroty G_n uzyskane po wyjściu agenta z wybranego stanu S (oznaczmy zwrot uzyskany po wyjściu ze stanu S przez $G_n(S)$). W praktycznym przypadku, gdy dysponujemy **historią dotyczącą N_{ep} epizodów**, to uśrednienie takie można zapisać jako:

$$V_n = \frac{1}{N_{ep}} \sum_{i=1}^{N_{ep}} G_{i,n}$$

Q-learning i macierz Q-wartości

Aby zaprojektować kryterium decyzji dla agenta możemy posłużyć się logiką podobną do tej wykorzystanej przy ocenie wartości stanu środowiska i **zdefiniować wartość $Q(s_n, a_n)$ (tzw. Q-wartość)**, która będzie **oznaczać średnią skumulowaną i zdyskontowaną nagrodę uzyskaną przez agenta w stanie S_n po podjęciu akcji a_n** .

Jest to zatem wartość wartości akcji agenta w stanie S_n .

Wartości funkcji $Q(S_n, a_n)$ mogą być pozyskane poprzez **iteracyjną interakcję** ze środowiskiem (podobnie jak wartość stanu).

W takim przypadku należy założyć, że mamy do dyspozycji **dyskretny zbiór akcji** i że dysponujemy **macierzą Q-wartości** o wymiarze $N_s \times N_a$ gdzie:

- N_s to liczba **stanów środowiska**,
- N_a to liczba możliwych do wykonania **akcji**.

Q-learning i macierz Q-wartości

		akcje (rozmiar $N_a = 3$)		
		0,91	0,13	0,20
stan środowiska (rozmiar $N_s = 4$)	0,34	0,89	0,45	
	0,11	0,95	0,04	
	0,45	0,54	1,13	

Przykład macierzy Q-wartości dla środowiska posiadającego 4 stany i w którym możliwe jest wykonanie 3 różnych akcji.

Q-learning i macierz Q-wartości

Po każdym podjęciu akcji a_n możemy **uaktualnić wartość macierzy** powiązaną ze stanem s_n i akcją a_n według następującego wzoru:

$$Q(s_n, a_n) = (1 - l_r) \cdot Q(S_n, a_n) + l_r \cdot Q^*(S_n, a_n)$$

gdzie l_r jest **współczynnikiem szybkości nauki**. Jest to odpowiednik uśrednienia wykorzystanego do zdefiniowania wartości stanu.

Wartość $Q^*(S_n, a_n)$ możemy oszacować z wartości zapisanych w macierzy Q-wartości w następujący sposób (analogiczny do wzoru na zwrot G_n przy znanym zwrocie G_{n+1} i współczynniku dyskontowania γ):

$$Q^*(S_n, a_n) = r_n + \gamma \cdot \max_{a_{n+1}}\{Q(S_{n+1}, a_{n+1})\}$$

Q-learning i macierz Q-wartości

Można zatem powiedzieć, że wartość $Q^*(S_n, a_n)$ jest wartością najlepszej akcji wybranej w stanie następujący po stanie numer n powiększoną o wartość natychmiastowo otrzymanej nagrody r_n .

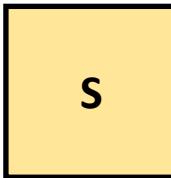
Po wykonaniu podstawienia za wartość $Q^*(S_n, a_n)$ ostateczna postać wzoru na $Q(S_n, a_n)$ przybiera następującą postać:

$$Q(S_n, a_n) = (1 - l_r) \cdot Q(S_n, a_n) + l_r \cdot [r_n + \gamma \cdot \max_{a_{n+1}}\{Q(S_{n+1}, a_{n+1})\}].$$

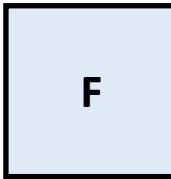
Przy zastosowaniu przedstawionego wzoru zakładamy, że **macierz Q-wartości na początku algorytmu jest zainicjalizowana losowymi wartościami**. Algorytm uczący się wartości macierzy Q według podanego wzoru nazywany jest **Q-learningiem**.

Przykład środowiska - Frozen Lake

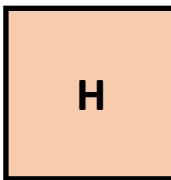
S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G



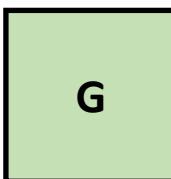
START - pole na który agent zaczyna każdy nowy epizod



FROZEN - pole „zamrożone”, po którym agent może przejść bezpiecznie (bez nagrody)



HOLE – przerębel, czyli pole po wejściu na które epizod kończy się (bez nagrody)



GOAL - pole docelowe, głównym celem każdego epizodu jest dotarcie do tego pola (po dotarciu epizod kończy się, a agent otrzymuje nagrodę równą 1)

Wizualizacja środowiska Frozen Lake (wg. konwencji z pakietu Pythona OpenAI Gym). Dodatkowym utrudnieniem jest fakt, że środowisko jest „słiskie” – ruch w wybranym przez agenta kierunku ma prawdopodobieństwo równe 1/3.

Przykład środowiska Frozen Lake

Przykład macierzy Q-wartości
wytrenowanych
do kierowania agentem działającym
w środowisku Frozen Lake.

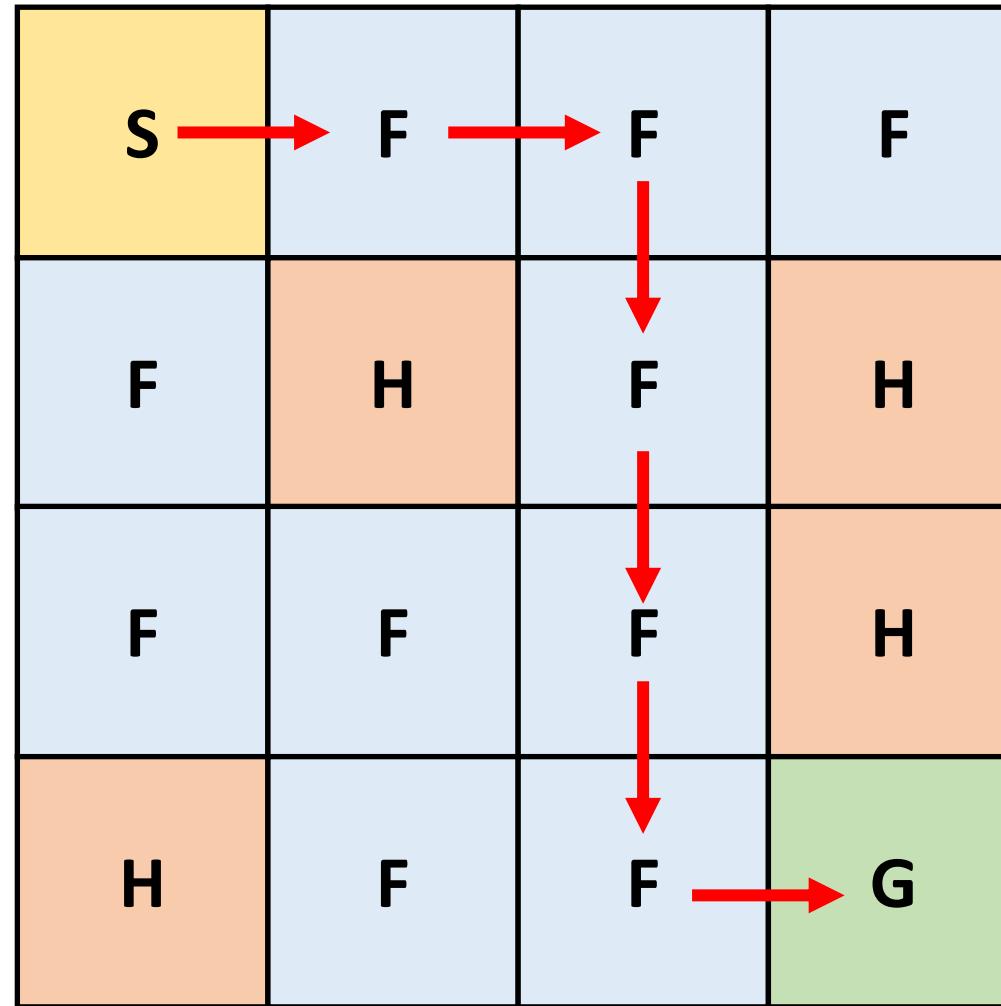
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

numery stanów
w środowisku Frozen Lake

numer stanu

	←	↓	→	↑
1	1,06	1,05	1,11	1,05
2	1,00	1,02	1,06	1,01
3	1,08	1,09	1,01	1,04
4	1,19	0,94	0,90	1,03
5	1,16	1,05	1,06	1,07
6	0,92	0,96	0,28	0,82
7	1,02	1,10	0,95	0,86
8	0,42	0,64	0,57	0,38
9	1,01	1,00	1,02	1,24
10	1,19	1,34	1,21	1,18
11	1,11	1,40	1,23	1,15
12	0,88	0,10	0,51	0,93
13	0,52	0,63	0,54	0,30
14	1,19	1,28	1,46	1,10
15	1,44	1,21	1,74	1,62
16	0,04	0,70	0,87	0,91

Macierz Q-wartości



Wizualizacja kierunków wskazywanych przez wytrenowaną macierz Q-wartości.

Realizacja celu agenta – funkcja strategii

Funkcja $Q(S_n, a_n)$ może posłużyć do **zdefiniowania zachowania agenta** w sposób pośredni.

W najprostszym przypadku można założyć, że najlepszą możliwą akcją a_n w stanie S_n jest akcja o największej wartości Q.

Możliwe jest jednak bardziej złożone podejście do tego, jakie decyzje podejmować na podstawie funkcji $Q(S_n, a_n)$ (lub na dowolnej innej podstawie).

Sposób podejmowania decyzji przez agenta w ogólności nazywany jest strategią (ang. *policy*) i jest oznaczany przez $\Pi(S_n)$. Wartością zwracaną przez funkcję strategii jest akcja a_n , która powinna być podjęta przez agenta w stanie S_n .

Realizacja celu agenta – funkcja strategii

W kontekście strategii wykonywanej przez agenta możemy teraz dokładniej zdefiniować wartość stanu jako **średni zwrot** uzyskany po rozpoczęciu interakcji ze środowiskiem w stanie S_n przy założeniu **postępowania zgodnie z wybraną strategią $\Pi(S_n)$** :

$$V_n = E_{a_n=\Pi(S_n)}\{G_n\} = E_{a_n=\Pi(S_n)} \left\{ \sum_{n=1}^N \gamma^{n-1} \cdot r_n \right\},$$

gdzie przez $E_{\Pi(S_n)}\{\}$ oznaczony został operator **uśrednienia po wszystkich epizodach** wykonanych zgodnie ze strategią $\Pi(S_n)$.

Przykłady różnych strategii

W ogólności możliwe jest zarówno **uczenie się przez agenta wprost wartości strategii $\Pi(S_n)$** , co jest często realizowane przy wykorzystaniu **sztucznych sieci neuronowych**.

W podejściu **klasycznym** częściej spotyka się zdefiniowanie funkcji strategii $\Pi(S_n)$ w **oparciu o wartość funkcji $Q(S_n, a_n)$** . Zdefiniowanie takie pozwala na modyfikowanie zachowania się agenta w zależności od potrzeb.

Przykładami strategii możliwych do zdefiniowania w ten sposób są:

- Strategia **zachłanna (chciwa, ang. greedy policy)**,
- Strategia **ϵ -chciwa**,
- Strategia typu **softmax**,
- Strategia **Boltzmanna**.

Strategia zachłanna (chciwa)

Najprostszą strategią jaką można przyjąć mając do dyspozycji funkcję $Q(S_n, a_n)$ jest wybór takiej akcji a_n , która **zapewni największą wartość Q**, co matematycznie można zapisać jako:

$$a_n = \arg \max_{a_n} \{Q(S_n, a_n)\},$$

gdzie przez $\arg \max_{a_n} \{ \}$ rozumiany jest operator znalezienia takiej akcji a_n , która **maksymalizuje wartość funkcji $Q(S_n, a_n)$** w stanie S_n .

Strategią chciwą posłużyliśmy się już wcześniej do zaprezentowania możliwego sposobu przeprowadzenia interakcji agenta ze środowiskiem Frozen Lake.

Strategia ϵ -chciwa – eksploracja i eksploatacja

Czysta strategia chciwa ma jedną bardzo poważną wadę – przypisuje ona wartość **jedynie nagrodzie uzyskanej natychmiast**. Dzieje się tak dlatego, że maksymalizacja nagrody następuje w niej jedynie przy przejściu pomiędzy dwoma **sąsiednimi stanami** ($z S_n \text{ do } S_{n+1}$).

Łatwo wskazać przykłady środowisk, w których taka strategia będzie nieskuteczna. Jednym z najprostszych jest już pokazane środowisko Frozen Lake, w którym agent musi **nauczyć się „na pamięć” drogi** z pola **START** do pola **GOAL**.

Większość czasu agent otrzymuje nagrodę równą zero, wartość nagrody równa 1 przyznawana jest jedynie po dotarciu do pola **GOAL**.

W takim przypadku strategia jest w stanie podjąć dobrą decyzję **jedynie w stanach sąsiadujących ze stanem GOAL**.

Strategia ϵ -chciwa – eksploracja i eksploatacja

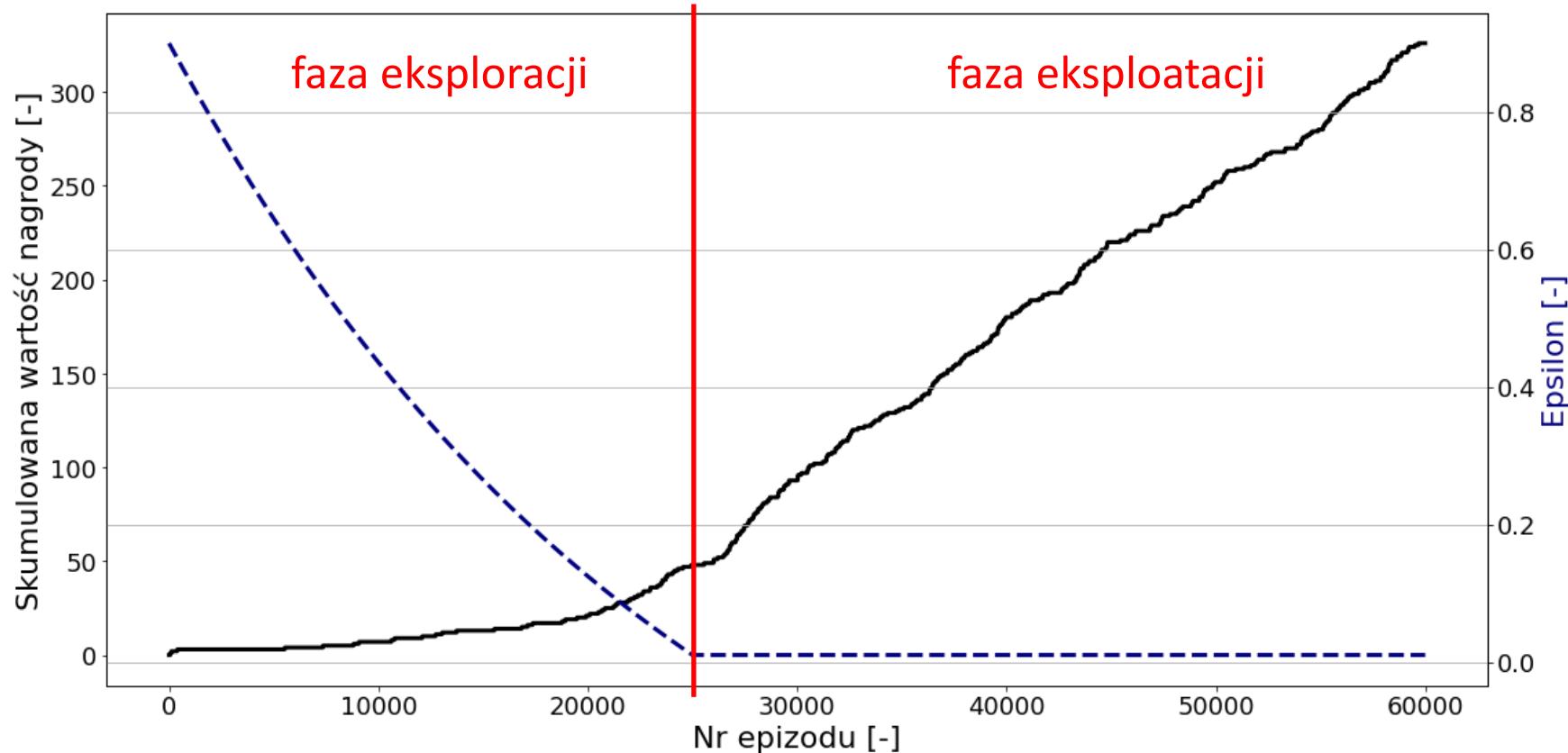
Rozwiązaniem tego problemu jest **zlagodzenie wymogu maksymalizacji** wartości Q przy każdym przejściu.

Można tego dokonać poprzez przyjęcie, że w każdym stanie S_n z prawdopodobieństwem równym pewnemu niewielkiemu **współczynnikowi ϵ** ($\epsilon \in <0,1>$) decyzja **podejmowana jest losowo** (z **równomiernym rozkładem** prawdopodobieństwa) a nie zgodnie z zasadą, że $a_n = \arg \max_{a_n} \{Q(S_n, a_n)\}$.

Dzięki takiemu podejściu pozwalamy algorytmowi na „nauczenie się”, środowiska co pozwala na szybsze (lub w ogóle umożliwia) np. na określenie najlepszej drogi wiodącej przez środowisko Frozen Lake.

Wartość ϵ często **nie jest stała**, a jest z epizodu na epizod zmniejszana tak aby na początku treningu agent skupił się na **eksploracji środowiska (ang. exploration)**, a dopiero potem zmaksymalizował uzyskiwaną nagrodę (co nazywane jest etapem **eksploatacji – ang. exploitation**).

Strategia ϵ -chciwa – eksploracja i eksplotacja



Wizualizacja wartości skumulowanej nagrody osiągniętej przez agenta realizującego strategię ϵ -chciwą w środowisku Frozen Lake.

Strategia typu softmax

Alternatywnym podejściem do strategii ϵ -chciwej, zakładającym **probabilistyczny wybór akcji**, jest **potraktowanie wartości Q jako wzorca** według którego możliwe jest obliczenie dyskretnej **gęstości prawdopodobieństwa** według której można dokonać wyboru akcji a_n w stanie S_n .

Nie jest możliwe potraktowanie wartości Q wprost jak prawdopodobieństw wyboru akcji. Konieczne jest ich znormalizowanie. Do normalizacji może posłużyć funkcja **softmax**:

$$p(a_n = a_{i,n}) = \frac{e^{Q(S_n, a_{i,n})}}{\sum_{k=1}^{N_a} e^{Q(S_n, a_{k,n})}},$$

gdzie $a_{i,n}$ oraz $a_{k,n}$ są i -tą i k -tą akcją możliwą do podjęcia w stanie S_n .

Strategia Boltzmanna

Ideę wyboru strategii według rozkładu gęstości prawdopodobieństwa zależnego od wartości $Q(S_n, a_n)$ można dalej rozbudowywać. Przykładem takiej modyfikacji jest tzw. strategia Boltzmanna, dla której gęstość prawdopodobieństwa zdefiniowana jest następująco:

$$p(a_n = a_{i,n}) = \frac{e^{\text{clip}\left(\frac{Q(S_n, a_{i,n})}{T}, c\right)}}{\sum_{k=1}^{N_a} \text{clip}\left(\frac{Q(S_n, a_{k,n})}{T}, c\right)}, \quad \text{clip}(x, c) = \begin{cases} -c & \text{dla } x < -c \\ x & \text{dla } x \in [-c, c] \\ c & \text{dla } x > c \end{cases}.$$

Rozkład ten ma dwa parametry – temperaturę T oraz współczynnik ograniczenia współczynników oznaczany przez c . Należy pamiętać o warunku, że $c > 0$.

Rozkłady decyzji podejmowanych za pomocą różnych strategii

Na przykładzie kodu z notatnika `DL_12_01_strategie_rl.ipynb` pokazane zostaną przykładowe rozkłady akcji podejmowanych przez agenta w pojedynczym kroku algorytmu uczenia wzmacnianego.

Problem wielorękiego bandyty

Jedną z trudności w projektowaniu algorytmów uczenia wzmacnianego jest **dobór parametrów strategii**. Na przykład współczynnika ϵ w strategii ϵ -chciwej albo pary współczynników T i c w strategii Boltzmanna.

Ciekawym problemem teoretycznym w ramach którego można rozważyć tego typu dobór jest tzw. **wielorękiego bandyty (*ang multi-armed bandit problem*)**.

Polega on na tym, że agent chce zmaksymalizować nagrodę za grę na **automacie wrzutowym posiadającym l dźwigni** (stąd nazwa – przez analogię do tzw. jednорękich bandytów).

Każda dźwignia charakteryzuje się **innym rozkładem prawdopodobieństwa otrzymywania nagród**. Poprzednie gry na wielorękim bandycie nie wpływają na przyszłe wyniki (czyli proces gry ma charakter markowski).

Problem wielorękiego bandyty

Agent nie zna parametrów opisujących poszczególne dźwignie. Dlatego problem nie jest dla niego trywialny i **nie jest możliwe zastosowanie chciwej strategii** polegającej na prostym wyborze najlepszej dźwigni.

Konieczna jest **eksploracja** w celu ustalenia, która dźwignia stanowi optymalny wybór.

Na początku agent jest zmuszony **wypróbować** każdą z dźwigni w celu ustalenia właściwości każdej z nich (eksploracja). Następnie dokonuje on **wyboru najlepszej** dźwigni (eksploatacja).

Poprzez symulację eksploracji w problemie wielorękiego bandyty możliwe jest **zbadanie wpływu parametrów takich jak ϵ na skuteczność procesu eksploracji**. Wraz ze zmianami wybranych parametrów zmieniać się będzie średnia wielkość nagrody uzyskiwana przez agenta.

Strategia ϵ -chciwa zastosowana do rozwiązania problemu l -rękiego bandyty

Na przykładzie kodu z notatnika `DL_12_02_l_reki_bandyta.ipynb` pokazany zostanie sposób doboru hiperparametrów strategii w kontekście rozwiązania problemu l -rękiego bandyty.

Literatura

1. Sutton, R., S., Barto, A., G., Reinforcement Learning: An Introduction, MIT Press, 1998.
2. Buduma, N., Locasio, N., Fundamentals of Deep Learning. Designing next-generation machine intelligence algorithms, O'Reilly Media, Inc., 201
3. Geron, A., Hands-On Machine Learning with Scikit-Learn & TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, Inc., 2019.
4. Deep RL Course - zbiór materiałów on-line dotyczących uczenia wzmacnianego dostępny pod adresem <https://simoninithomas.github.io/deep-rl-course/> (data dostępu: 12.01.2022)
5. Goodfellow, I., Bengio, J., Courville, A., Deep Learning, The MIT Press, 2016.

Dziękuję za uwagę



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Uczenie Głębokie

UCZENIE ZE WZMOCNIENIEM Z WYKORZYSTANIEM MODELI GŁĘBOKICH, CZ. 1

Adam Kurowski

Katedra Systemów Multimedialnych,
Wydział Elektroniki, Telekomunikacji i Informatyki PG



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Wprowadzenie

Klasyczne techniki uczenia wzmacnianego stanowią bardzo interesujące modele matematyczne, które są w stanie **zilustrować** proces interakcji intelligentnego agenta ze środowiskiem.

Jest to sposób uczenia, który ma wiele zalet – między innymi **nie istnieje potrzeba tworzenia dużego zbioru danych**, gdyż te na bieżąco są dostarczane przez środowisko z którym agent wchodzi w interakcje.

Niestety, jedną z wad tego podejścia jest fakt, że **w przypadku bardziej skomplikowanych środowisk agent bazujący na klasyczny, uczeniu ze wzmocnieniem nie jest w stanie nauczyć się skutecznej strategii**.

Stan ten zmienił się po rozwinięciu się technik związanych z **uczeniem głębokim**, które umożliwiły wykorzystanie możliwości precyzyjnych głębokich sieci neuronowych do realizowania strategii agenta.

Podejście klasyczne – rozwiążane problemy

Z wykorzystaniem klasycznego podejścia bazującego na uczeniu wzmacnianym możliwe było rozwiązywanie problemów takich jak:

- sterowanie **chodem dwunożnego robota**

Benbrahim, H., & Franklin, J.A. (1997). *Biped dynamic walking using reinforcement learning.* Robotics Auton. Syst., 22, 283-302.

- sterowanie wirtualnymi agentami tak, aby **grały w komputerową symulację piłki nożnej**

Stone P, Sutton RS, Kuhlmann G. *Reinforcement Learning for RoboCup Soccer Keepaway. Adaptive Behavior.* 2005;13(3):165-188. doi:10.1177/105971230501300301

Podejście klasyczne – rozwiążane problemy

- **sterowanie windami – ustalanie kolejności odwiedzania pięter**

Crites, R.H., Barto, A.G. *Elevator Group Control Using Multiple Reinforcement Learning Agents*. Machine Learning 33, 235–262 (1998).

- **gra w proste gry – backgammon (tryktrak)**

Gerald Tesauro. 1995. Temporal difference learning and TD-Gammon. Commun. ACM 38, 3 (March 1995), 58–68.

- **układanie planu pracy z wykorzystaniem sieci opóźnieniowej**

Wei Zhang, Thomas G. Dietterich. 1995. High-performance job-shop scheduling with a time-delay $\text{TD}(\lambda)$ network. In Proceedings of the 8th International Conference on Neural Information Processing Systems (NIPS'95). MIT Press, Cambridge, MA, USA, 1024–1030.

Podejście głębokie – rozwiążane problemy

- Agent grający w gry Atari
Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing atari with deep reinforcement learning*. arXiv preprint arXiv:1312.5602.
- Agent grający w grę strategiczną czasu rzeczywistego - **StarCraft II** (architektura AlphaStar)
Vinyals O, Babuschkin I, Czarnecki WM, et al. *Grandmaster level in StarCraft II using multi-agent reinforcement learning*. Nature. 2019 Nov;575(7782):350-354. doi: 10.1038/s41586-019-1724-z. Epub 2019 Oct 30. PMID: 31666705.
- Agent grający w grę strategiczną **DOTA** (architektura OpenAI Five):
Berner, C., Brockman, G., Chan, B., et al.(2019). *Dota 2 with large scale deep reinforcement learning*. arXiv preprint arXiv:1912.06680.

Podejście głębokie – rozwiążane problemy

- Robot rozwiązujący kostkę **Rubika** poprzez sterowanie robotem imitującym ludzką dłoń:

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., ... & Zhang, L. (2019). Solving rubik's cube with a robot hand. arXiv preprint arXiv:1910.07113.

- Agenci dokonujący **zautomatyzowanego zakupu powierzchni reklamowej**:

Jin, J., Song, C., Li, H., Gai, K., Wang, J., & Zhang, W. (2018, October). Real-time bidding with multi-agent reinforcement learning in display advertising. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management (pp. 2193-2201).

- **System rekomendujący wiadomości do przeczytania przez użytkowników**:

S. Bangari, S. Nayak, L. Patel, K. T. Rashmi, *A Review on Reinforcement Learning based News Recommendation Systems and its challenges*, 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021, pp. 260-265, doi: 10.1109/ICAIS50930.2021.9395812.

Jakie wyzwania stawiają bardziej złożone problemy?

Proste problemy zazwyczaj charakteryzują się stanami możliwymi do zapisania za pomocą **prostych struktur danych**, na przykład:

- **listy kilku do kilkunastu parametrów liczbowych**,
- **macierz** przedstawiająca stan środowiska.

Problemy **bardziej złożone** charakteryzują się stanem zawierającym **dane o bardzo skomplikowanej strukturze** (często **multimedialne**) takie jak na przykład:

- **mapy ciepła** opisujące **zasoby i pozycje jednostek** w grze strategicznej
- **klatki** zawierające **grafiki** wygenerowanej przez zręcznościową grę komputerową,
- **złożona sekwencja sygnałów zwrotnych** od osoby czytającej artykuł dostępnego na stronie internetowej.

Problem estymacji wartości Q danego stanu

Stan środowiska może zawierać dane o **charakterze multimedialnym** (na przykład zdjęcia, filmy, nagrania audio) lub też o **bardzo złożonej strukturze wewnętrznej** (dane o preferencjach użytkownika systemu rekomendacyjnego).

Stąd przydatne może się okazać wykorzystanie w algorytmie uczenia wzmacnianego **struktur przystosowanych do przetwarzania tego typu danych**.

Z pomocą mogą tu przyjść narzędzia takie jak **głębokie sieci splotowe**, głębokie sieci **rekurencyjne** i wszystkie inne algorytmy zwykle wykorzystywane w procesach uczenia nadzorowanego i nienadzorowanego.

Konieczne jest jednak **odpowiednie umieszczenie** tego typu komponentów w algorytmie realizującym uczenie ze wzmacnieniem.

Przykład sieci neuronowej estymującej wartość Q

Jednym z prostszych pomysłów, który można zrealizować jest wykorzystanie głębokiej sieci neuronowej do **ulepszenia algorytmu Q-learningu**.

Zamiast posługiwać się macierzą do odczytu wartości Q dla wybranych par stanu i akcji możliwe jest **wykorzystanie sieci neuronowej do obliczenia wektora wartości Q_n dla wybranego stanu S_n** .

Następnie można tak jak dotychczasowo wybrać akcję o największej wartości Q do realizacji.

Podejście takie umożliwia wykorzystanie np. treści graficznych w roli stanu środowiska. Tak właśnie zrealizowane są często algorytmy grające w gry komputerowe (np. Atari).

Algorytm wykorzystujący głęboką sieć neuronową do estymacji Q-wartości nazywany jest **algorytmem głębokiej Q-sieci (ang. deep Q-network, DQN)**.

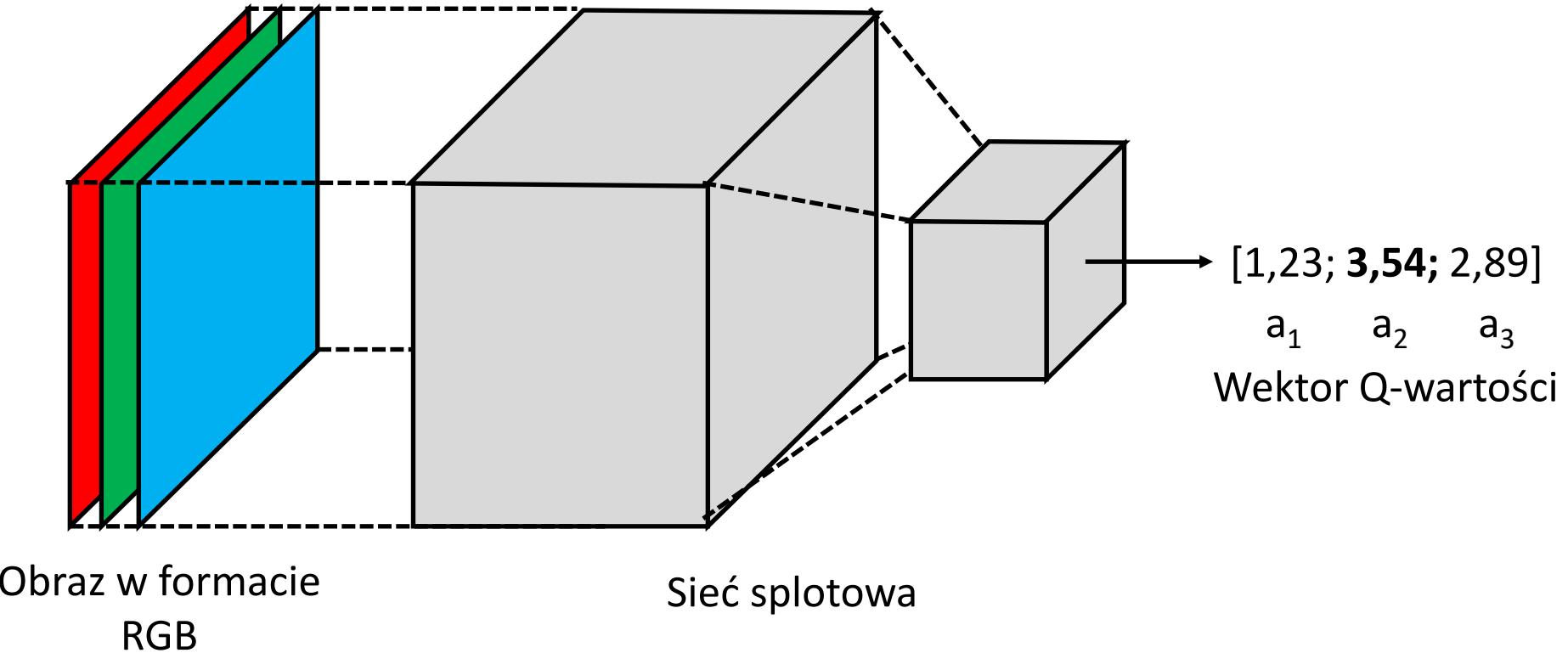
Przykład sieci neuronowej estymującej wartość Q

W przypadku algorytmu grającego w **grę** komputerową wartość Q każdej z akcji podjętej przez agenta obliczana jest na podstawie **klatki (lub częściej sekwencji klatek)** wygenerowanych przez grę.

Wejście sieci obliczającej Q wartości nie musi tylko ograniczać się do obrazów, mogą to być **ramki sygnału fonicznego** lub ich sparametryzowana forma (np. współczynniki **MFCC**).

W porównaniu do klasycznego algorytmu Q-learningu sieć zastępuje macierz Q-wartości. Dzięki temu wartość Q może być obliczana nie tylko dla przeliczalnej liczby stanów środowiska, ale możliwe jest także obliczanie jej dla środowisk, w której zmienne stanu mają charakter ciągły.

Przykład sieci neuronowej estymującej wartość Q



Schemat przedstawiający przykładową sieć obliczającą Q-wartość na podstawie obrazu wejściowego

Trening sieci DQN

Osobnym problemem jest realizacja treningu sieci DQN. Z jednej strony konieczne jest aktualizowanie wartości Q w taki sposób, aby spełniały one tzw. **równanie Bellmana** (podobnie jak w macierz Q-wartości w podejściu klasycznym):

$$Q(S_n, a_n) = E \left\{ r_n + \gamma \max_{a_{n+1}} Q(S_{n+1}, a_{n+1}) \right\},$$

gdzie $Q(S_n, a_n)$ jest wartością Q związaną z akcją a_n w stanie obecnym S_n , r_n jest nagrodą uzyskaną po podjęciu akcji a_n , a γ jest współczynnikiem dyskontowania.

Z drugiej strony uaktualnienia wartości Q muszą być zapisane w postaci funkcji straty, gdyż Q-wartości obliczane są przez głęboką sieć neuronową.

Trening sieci DQN

Stąd trening sieci DQN odbywa się poprzez wykorzystanie **następującej funkcji straty** (oznaczonej przez $L_n(\theta_n)$):

$$L_n(\theta_n) = E_{S_n, a_n, S_{n+1}} \{y_n - Q(S_n, a_n; \theta_n)\},$$

$$y_n = \begin{cases} r_n & \text{jeśli } S_n \text{ jest} \\ & \text{stanem terminalnym} \\ r_{n+1} + \gamma \max_{a_{n+1}} \{Q(S_{n+1}, a_{n+1}; \theta_n)\} & \text{jeśli } S_n \text{ nie jest} \\ & \text{stanem terminalnym} \end{cases},$$

gdzie $Q(S_n, a_n; \theta_n)$ jest Q-wartością obliczoną przez sieć z wagami θ_n .

Trening sieci DQN

Czasami zdarza się, że w estymacjach obliczanych przez sieć neuronową mogą pojawić się **wartości odstające**. Mogą one mieć **degradujący wpływ na proces treningu** sieci DQN. Z tego powodu często na bazie straty $L_n(\theta_n)$ wprowadza się stratę $L'_n(\theta_n)$, która bazuje na pojęciu **straty Hubera**:

$$L'_n(\theta_n) = E_{S_n, a_n, S_{n+1}, r_n} \{ H(L_n(\theta_n)) \},$$

$$H(L_n(\theta_n)) = \begin{cases} \frac{1}{2} (L_n(\theta_n))^2 & \text{dla } |L_n(\theta_n)| \leq 1 \\ |L_n(\theta_n)| - 1 & \text{w pozostałych przypadkach} \end{cases},$$

gdzie $E_{S_n, a_n, S_{n+1}, r_n} \{ \}$ jest operatorem uśredniającym po wszystkich przykładach w grupie danych treningowych.

Podejmowanie decyzji przez sieci DQN

Sieć DQN w algorytmie uczenia wzmacnianego podobnie jak macierz Q-wartości zwraca **wektor będący odpowiednikiem wiersza w macierzy Q-wartości**. Aby taki wiersz mógł być wygenerowany konieczne jest podanie na wejście sieci danych opisujących stan S_n .

Wiersz zwrócony przez sieć neuronową może być przetworzony **w taki sam sposób jak w klasycznym algorytmie** uczenia ze wzmacnieniem.

Można przyjąć, że w trakcie interakcji agenta ze środowiskiem sieć neuronowa oblicza wiersz macierzy Q, co jest wystarczające do **realizacji dowolnej strategii możliwej do zaimplementowania w klasyczny sposób** (np. strategii ϵ -chciwej lub Boltzmanna).

Dodatkowej **uwagi wymaga jednak sposób zbierania danych**, które służyć będą do treningu sieci DQN.

Rola i zasada działania pamięci powtórek

Do treningu konieczne jest zebranie całej grupy (ang. batch) danych. Musi być także dostępna informacja o najlepszej wartości Q_{n+1} w stanie S_{n+1} do którego agent przeszedł po podjęciu decyzji (zgodnie z równaniem uaktualniania wartości Q).

Fakt ten determinuje specyficzny sposób archiwizowania danych o interakcji agenta pochodzących z treningu sieci. Konieczne staje się zapisywanie wszystkich przejść pomiędzy stanami S_n a S_{n+1} (oznaczanych przez T_n) w postaci:

$$T_n = (S_n, a_n, r_n, S_{n+1}),$$

gdzie S_n oznacza stan z którego poprzez podjęcie akcji a_n nastąpiło przejście do stanu S_{n+1} , co poskutkowało przyznaniem agentowi nagrody r_n .

Rola i zasada działania pamięci powtórek

Pamięć powtórek pozwala **zminimalizować skorelowanie** następujących po sobie przykładów treningowych poprzez wybieranie do treningu **losowego podzbioru zapisów z pamięci**.

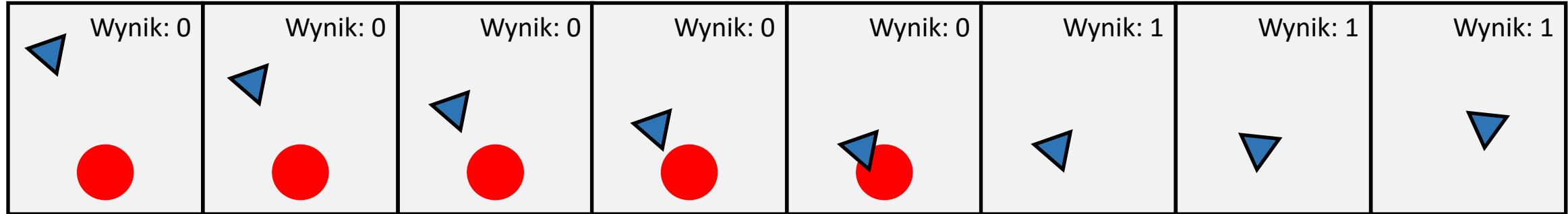
Pamięć powtórek pozwala także rozwiązać problemy związane z koniecznością ustalenia **maksymalnej wartości Q w kroku następnym (bo znany jest zawsze stan S_{n+1})**.

Pamięć powtórek posiada pewną **skończoną, z której założoną długosć**, w której po osiągnięciu jej maksymalnej pojemności **usuwane są najstarsze zapisy interakcji agenta ze środowiskiem**.

Dlatego często pamięć powtórek jest implementowana w postaci **bufora kołowego**.

Rola i zasada działania pamięci powtórek

Pamięć powtórek po kroku nr 8:



stan 1.

stan 2.

stan 3.

stan 4.

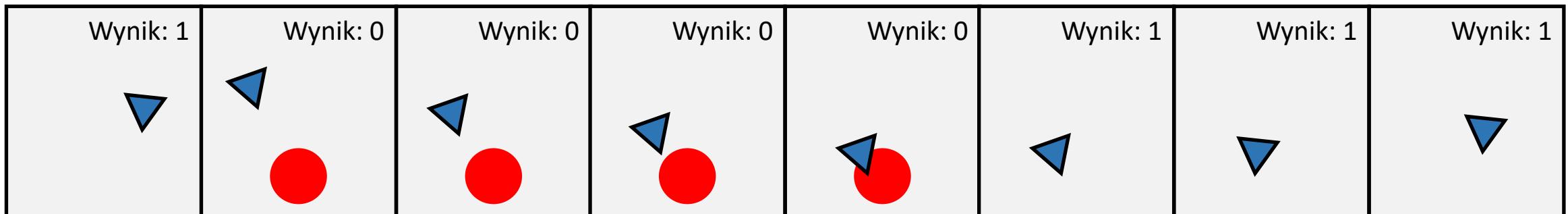
stan 5.

stan 6.

stan 7.

stan 8.

Pamięć powtórek po kroku nr 9:



stan 9.

stan 2.

stan 3.

stan 4.

stan 5.

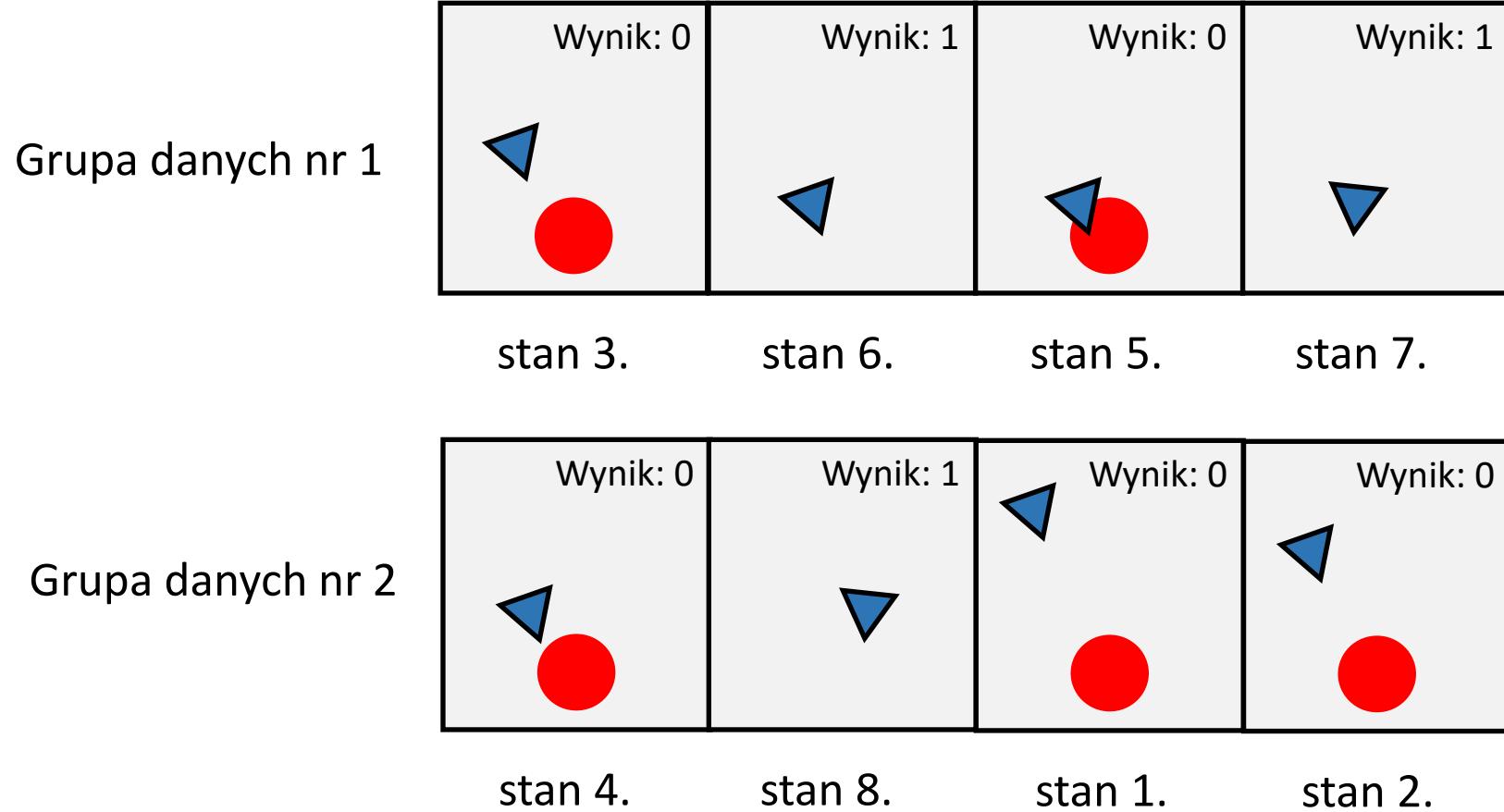
stan 6.

stan 7.

stan 8.

Pamięć powtórek o długości 8 zrealizowanej w postaci bufora kołowego – najstarsze zapisy są nadpisywane przez nowe stany interakcji agenta ze środowiskiem.

Rola i zasada działania pamięci powtórek



Zbieżność nauki wartości Q – podwójne Q-sieci

Konieczność wykorzystania kopii Q sieci i regulacji częstotliwością aktualizacji wag sieci, która w procesie treningu podejmuje decyzje, gdyż inaczej istnieje ryzyko **nieuzyskania zbieżności** przez algorytm treningu.

Dlatego też często w trakcie treningu zwykle algorytm operuje na **dwóch sieciach DQN**:

- pierwsza z nich odpowiada za **interakcję agenta ze środowiskiem** – sieć **realizująca politykę (ang. policy network)**,
- druga z nich jest siecią **obliczającą cel (ang. target network)** – oblicza ona wartość $Q(S_{n+1}, a_{n+1})$

Co pewną ustaloną **liczbę iteracji** wagi sieci realizującej **politykę** zastępowane są wagami sieci **obliczającej cel**.

Zbieżność nauki wartości Q – podwójne Q-sieci

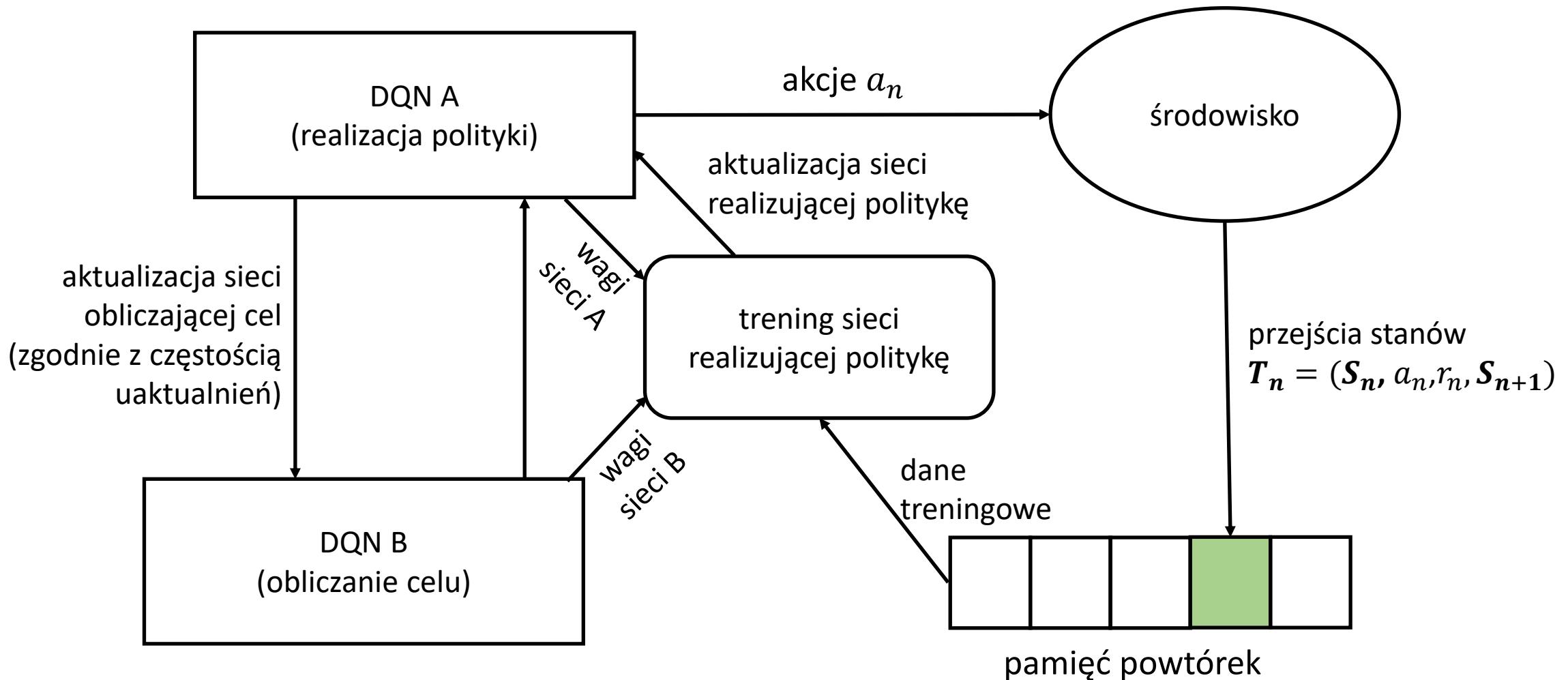
Częstotliwość aktualizacji wag sieci wchodzącej w interakcję ze środowiskiem często nazywa się **częstością uaktualnień (ang. update frequency)**.

Zabieg ten pozwala na **zachowanie ciągłości stosowania jednego sposobu obliczania docelowych wartości Q** przez wspomnianą liczbę iteracji i jej uaktualnianie dopiero po pewnym czasie.

Jeżeli nie stosowalibyśmy osobnej sieci do treningu to spotkalibyśmy się z problemem w którym **estymacje przyszłych Q-wartości zmieniają się w trenowanej sieci z iteracji na iterację**, przez co cel treningu także zmienia się z iteracji na iterację. Jest to tzw. **problem ruchomego celu (ang. moving target problem)**.

Architektura ta czasami nazywana jest **podwójną Q-siecią (ang. double DQN)**.

Zbieżność nauki wartości Q – podwójne Q-sieci



Proces treningu podwójnej sieci DQN

Architektura DDQN (dueling DQN)

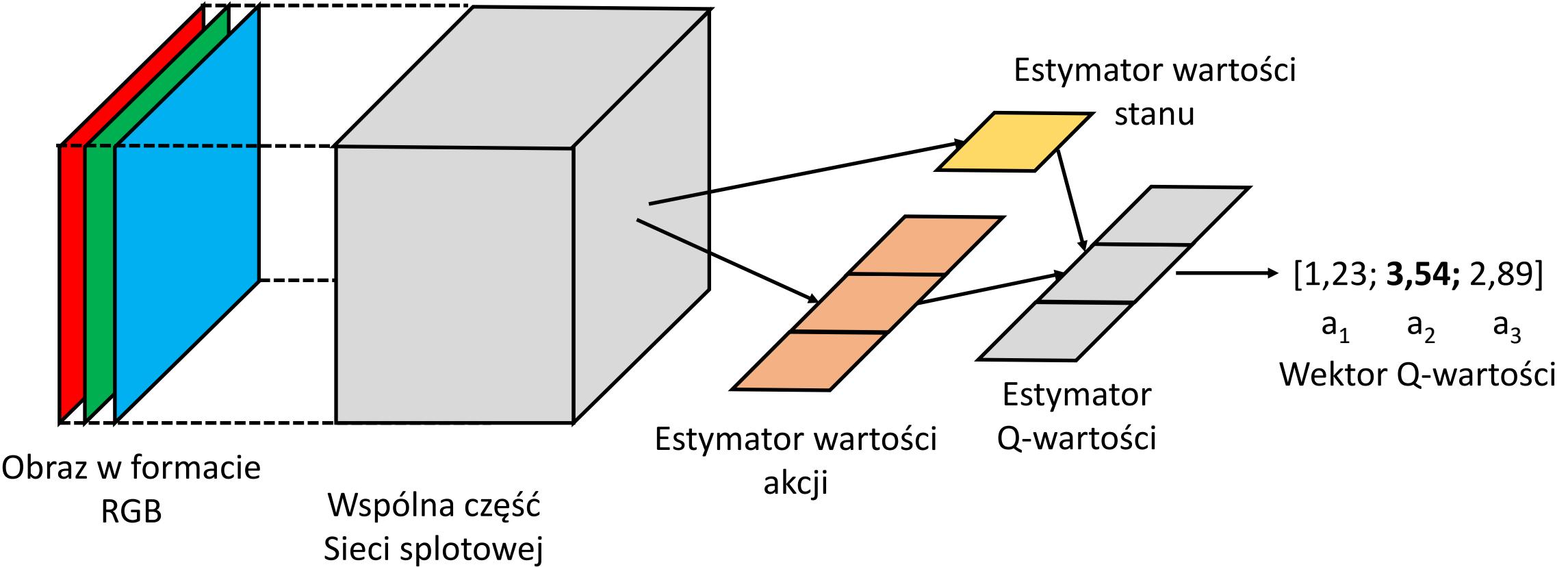
Innym mechanizmem stosowanym do polepszenia zbieżności treningu sieci neuronowej jest wymuszenie na sieci **rozdziału estymacji wartości stanu S_n od estymacji wartości poszczególnych akcji** podejmowanych w tym stanie.

Może być to przydatna cecha zwłaszcza w przypadku, kiedy w ogólności **nie wszystkie akcje są równie korzystne** lub **pewne stany środowiska są szczególnie wartościowe** niezależnie od akcji.

Rozdział oceny stanu środowiska od akcji realizuje się poprzez **zdefiniowanie w strukturze sieci DQN dwóch odrębnych odpowiedzialnych za ocenę wartości stanu i akcji**.

Sieci z opisany rozdziałem nazywane są **dwudzielnymi głębokimi Q-sieciami (ang. dueling deep Q-networks, DDQN)**.

Architektura DDQN (dueling DQN)



Zalety i wady sieci DQN i pokrewnych

Sieci DDQN (oraz pokrewne) realizują **intuicyjny algorytm** polegający na wartościowaniu stanów środowiska i akcja (lub ich par). Łatwo określić, jakie stany środowiska **rokują wysokie wartości nagrody**, a które nie.

Głębokie sieci estymujące wartość Q pozwalają na korzystanie nawet ze **stanów opisywanych przez parametry przyjmujące wartości z ciągłego przedziału wartości**.

Podejście DDQN ma jednak swoje wady, najważniejszą z nich jest fakt, że **akcje agenta nadal muszą należeć do dyskretnego, skońzonego zbioru**.

Sprawia to, że sieć DDQN nie nadaje się na przykład do zadań sterownia robotem, w którym sygnały sterujące przyjmują wartości z ciągłego przedziału wartości.

Należy też zaznaczyć, że **po wprowadzeniu dodatkowych mechanizmów zapewniających zbieżność treningu głębokich Q-sieci ich algorytm treningu staje się skomplikowany**.

Algorytm DQN – demonstracja treningu sieci z włączonymi i wyłączonymi mechanizmami pomocniczymi

Na przykładzie kodu z notatnika `DL_13_01_DQN.ipynb`.

Algorytm głębokiego gradientu strategii (DPG)

Sieci DQN uczą się interakcji ze środowiskiem poprzez pośrednią funkcję $Q(S, a)$. Strategia agenta $\Pi(S)$ realizowana jest na jej podstawie.

Mając do dyspozycji głęboką sieć neuronową możliwe jest **wykorzystanie sieci neuronowej bezpośrednio do estymowania funkcji strategii** (*ang. policy*).

W takim przypadku pomijamy pośredni krok jakim jest wybór Q-wartości i obliczamy akcję wprost jako:

$$a = \Pi(S; \theta)$$

gdzie przez θ oznaczone są wagi sieci neuronowej realizującej strategię $\Pi(S; \theta)$. Klasę sieci działających w opisany sposób nazywamy sieciami realizującymi algorytm głębokiego gradientu strategii (*ang. deep policy gradient*).

Algorytm głębokiego gradientu strategii (DPG)

Strategia zdefiniowana jako $a = \Pi(S; \theta)$ nazywana jest **strategią deterministyczną** gdyż znając stan S mamy pewność odnośnie tego, jaka akcja a zostanie podjęta.

Możliwe jest także zdefiniowanie **strategii stochastycznej**, która zwraca rozkład prawdopodobieństwa (dyskretny lub ciągły) podjęcia danej akcji, co oznaczamy jako:

$$p(a) = \Pi(a|S; \theta),$$

gdzie przez $p(a)$ oznaczamy gęstość prawdopodobieństwa związaną z podjęciem akcji a .

Strategie stochastyczne są szczególnie przydatne jeżeli agent ma się nauczyć wybierać akcję z **ciągłego przedziału wartości**.

Zasady treningu algorytmu DPG

Podobnie jak w przypadku sieci DQN konieczne jest zastanowienie się nad procesem treningu sieci neuronowej.

W przypadku realizacji funkcji strategii przez sieć neuronową **nie jest możliwe posługiwanie się wprost Q-wartościami** do zdefiniowania sposobu treningu sieci.

Zamiast tego punktem wyjścia jest **funkcja nagrody skumulowanej w stanie n -tym** (oznaczana jako $J(\theta)$):

$$J_n(\theta) = E_{\Pi(a|S; \theta)} \left\{ \sum \gamma r_n \right\} = E_{\Pi(a|S; \theta)} \{V(S)\},$$

gdzie $E_{\Pi(a|S; \theta)} \{ \}$ jest operatorem uśrednienia po wszystkich możliwych do podjęcia przez agenta akcjach a_n zgodnie ze strategią $\Pi(a|S; \theta)$, a $V(S)$ jest wartością stanu S .

Zasady treningu algorytmu DPG

Wartość funkcji $J(\theta)$ można także zapisać jako:

$$J(\theta) = \sum_{S \in D_S} d(S; \theta) \sum_{a \in A} \Pi(a|S; (\theta)) \cdot Q(S, a),$$

gdzie D_S jest przestrzenią zawierającą wszystkie możliwe stany S , a $d(S; \theta)$ jest prawdopodobieństwem znalezienia się w stanie S przy postępowaniu zgodnie ze strategią $\Pi(a|S; \theta)$, wartość $d(S; \theta)$ można też interpretować jako prawdopodobieństwo przejścia między stanami w decyzyjnym łańcuchu Markowa.

Celem algorytmu trenującego jest maksymalizacja wartości funkcji $J(\theta)$. W tym celu konieczne jest zmodyfikowanie strategii $\Pi(a|S; \theta)$, co z kolei można osiągnąć wpływając na wartości wag sieci θ_n .

Zasady treningu algorytmu DPG

Do optymalizacji funkcji $J(\theta)$ można wykorzystać gradient $\nabla_{\theta}J(\theta)$, który obliczony jest ze względu na wagi sieci θ .

Pozostaje jednak pewien problem. W wyrażeniu służącym obliczaniu wartości $J(\theta)$ występuje zarówno strategia $\Pi(a|S; \theta)$, którą chcemy zoptymalizować jak i rozkład przejść stanów $d(S; \theta)$, który też zależy od wag θ .

Jest to problem, bo uwzględnienie w gradiencie wartości $d(S; \theta)$ znacznie komplikuje rozwiązanie problemu.

Na szczęście można dowieść, że w obliczeniach możliwe jest pominięcie tej zmienności – jest to tzw. **twierdzenie o gradiencie strategii (ang. policy gradient theorem)**.

Zasady treningu algorytmu DPG

Można zatem zapisać:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \left[\sum_{S \in D_S} d(S; \boldsymbol{\theta}) \sum_{a \in A} \Pi(a|S; \boldsymbol{\theta}) \cdot Q(S, a) \right],$$

korzystając z twierdzenia o gradiencie strategii można natomiast przekształcić powyższą zależność w następujący sposób:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{S \in D_S} d(S; \boldsymbol{\theta}) \sum_{a \in A} \nabla_{\boldsymbol{\theta}} \Pi(a|S; \boldsymbol{\theta}) \cdot Q(S, a)$$

Zasady treningu algorytmu DPG

Przekształcając dalej wykorzystując mnożenie i dzielenie przez strategię $\Pi(a_n|S; \theta)$ można zapisać:

$$\nabla_{\theta} J(\theta) = \sum_{S \in D_S} d(S; \theta) \sum_{a \in A} Q(S, a) \cdot \Pi(a|S; \theta) \cdot \frac{\nabla_{\theta} \Pi(a|S; \theta)}{\Pi(a|S; \theta)}.$$

W tym punkcie możliwe jest posłużenie się następującą tożsamością:

$$\nabla \ln f(x) = \frac{\nabla f(x)}{f(x)}.$$

Zasady treningu algorytmu DPG

Po uwzględnieniu zależności wzór na $\nabla_{\theta}J(\boldsymbol{\theta})$ przyjmuje następującą postać:

$$\nabla_{\theta}J(\boldsymbol{\theta}) = \sum_{S \in D_S} d(S; \boldsymbol{\theta}) \sum_{a \in A} \Pi(a|S; \boldsymbol{\theta}) \cdot Q(S, a) \cdot \nabla_{\boldsymbol{\theta}} \ln \Pi(a|S; \boldsymbol{\theta}),$$

co skrótnie można zapisać jako:

$$\nabla_{\theta}J(\boldsymbol{\theta}) = E_{\Pi}\{Q(S, a) \cdot \nabla_{\boldsymbol{\theta}} \ln \Pi(a|S; \boldsymbol{\theta})\},$$

gdzie przez $E_{\Pi}\{\}$ rozumie się średnią policzoną po wszystkich możliwych stanach S i akcjach a przy założeniu, że prawdopodobieństwo wartości uśrednianej równe jest $\Pi(a|S; \boldsymbol{\theta})$.

Zasady treningu algorytmu DPG

Żeby ułatwić obliczenia wartości gradientu $\nabla_{\theta}J(\theta)$ możliwe jest zastąpienie wartości $Q(S, a)$ wartością zysku G i odniesienie wartości do obecnego stanu (S_n) i akcji (a_n). Należy skorzystać z następującej zależności:

$$Q(S, a) = E_{\Pi}\{G(S_n, a_n)\},$$

co ostatecznie skutkuje uzyskaniem następującej zależności:

$$\nabla_{\theta}J(\theta) = E_{\Pi}\{G(S_n, a_n) \cdot \nabla_{\theta}\ln \Pi(a_n|S_n; \theta)\}.$$

Wzór ten może posłużyć do zdefiniowania algorytmu, który poprzez iteracyjną interakcję ze środowiskiem zoptymalizuje wagi sieci θ w taki sposób, aby zmaksymalizować wartość $J(\theta)$.

Zasady treningu algorytmu DPG

Algorytm treningu wykorzystujący wyprowadzoną zależność na $\nabla_{\theta}J(\theta)$ nazywany jest algorytmem **REINFORCE** i polega na wykonywaniu następujących kroków:

1. Losowa inicjalizacja parametrów θ .
2. Przeprowadzenie interakcji ze środowiskiem zgodnie ze strategią $\Pi(a_n|S_n; \theta)$, co skutkuje uzyskaniem ciągu wartości $S_1, a_1, r_1, S_2, a_2, r_2, S_3, \dots$
3. Dla wartości $n = 1, 2, \dots, N$:
 - a) Oblicz zwrot G_n
 - b) Zaktualizuj wartości θ zgodnie z regułą $\theta \leftarrow \theta + l_r \gamma^n G_n \nabla_{\theta} \ln \Pi(a_n|S_n; \theta)$

Wartości $\Pi(a_n|S_n; \theta)$ są w praktyce wyjściami otrzymanymi z sieci neuronowej w danym stanie S_n .

Kiedy DPG ma przewagę nad DQN i DDQN?

Algorytm bazujący na estymacji funkcji strategii ma tę podstawową przewagę, że potrafi **operować na stochastycznej strategii mogącej przyjmować wartości z ciągłego zbioru**.

Jeżeli **przestrzeń akcji jest wielowymiarowa** (jest wektorem zawierającym znaczną liczbę składowych) zwykle algorytmy **wprost estymujące** wartość funkcji **strategii szybciej się uczą**.

Algorytmy opierające się na szacowaniu wartości **funkcji Q** mogą podlegać **większym oscylacjom parametrów** w trakcie **treningu**.

W swoim najprostszym wariantie – algorytmie REINFORCE metody bazujące na estymacji funkcji strategii są **znacznie prostsze w implementacji** (nie wymagają stosowania pamięci powtórek ani mechanizmów analogicznych do podwójnych sieci DQN).

Algorytm REINFORCE – demonstracja działania algorytmu w prostym środowisku

Na przykładzie kodu z notatnika `DL_13_02_REINFORCE.ipynb`.

Literatura

1. Sutton, R., S., Barto, A., G., Reinforcement Learning: An Introduction, MIT Press, 1998.
2. Buduma, N., Locasio, N., Fundamentals of Deep Learning. Designing next-generation machine intelligence algorithms, O'Reilly Media, Inc., 201
3. Geron, A., Hands-On Machine Learning with Scikit-Learn & TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, Inc., 2019.
4. Deep RL Course - zbiór materiałów on-line dotyczących uczenia wzmacnianego dostępny pod adresem <https://simoninithomas.github.io/deep-rl-course/> (data dostępu: 12.01.2022)
5. Weng, L., Policy Gradient Algorithms, materiały on-line opisujące algorytmu bazujące na gradientach strategii, url: <https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>
6. Goodfellow, I., Bengio, J., Courville, A., Deep Learning, The MIT Press, 2016.

Dziękuję za uwagę



Fundusze
Europejskie
Polska Cyfrowa



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz
Rozwoju Regionalnego



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Uczenie Głębokie

UCZENIE ZE WZMOCNIENIEM Z WYKORZYSTANIEM MODELI GŁĘBOKICH, CZ. 2

Adam Kurowski

Katedra Systemów Multimedialnych,
Wydział Elektroniki, Telekomunikacji i Informatyki PG



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.

Wprowadzenie

Klasyczne techniki uczenia wzmacnianego stanowią punkt wyjścia do realizacji **bardziej złożonego** podejścia do interakcji agenta z otoczeniem.

Spora część z tych metod bazuje jednak na podstawach teoretycznych związanych z dwoma algorytmami uczenia wzmacnianego:

- głębokimi Q-sieciami (*ang. deep Q-networks, DQN*),
- algorytmem głębokiego gradientu strategii (*ang. deep policy gradient, DPG*).

Podejścia te pomimo realizacji podobnego celu – interakcji agenta z otoczeniem różnią się jednak w sposobie realizacji.

Różne są także zalety i wady każdego z tych podejść. Stąd też potrzeba **modyfikacji** wspomnianych podejść i wprowadzania nowych algorytmów uczenia ze wzmocnieniem.

Podejścia oparte na wartości

Dla porządku warto też zaznaczyć, że algorytmy DQN zaliczają się do szerzej rozpoznawanego typu algorytmów **bazujących na wartości (ang. value-based reinforcement learning)**.

Algorytmy te charakteryzują się taką właściwością, że do **wyboru najlepszej w danym momencie akcji posługują się estymatą wartości** (np. stanu, akcji lub innej).

Przykładowo estymowane mogą być:

- wartość $Q(S, a)$ stanowiąca estymację średniego zwrotu po kontynuowaniu epizodu poprzez podjęcie akcji a w stanie S ,
- wartość $V(S)$ która oznacza wartość stanu czyli średni zwrot uzyskany przez agenta po znalezieniu się w stanie S .

Podejścia oparte na wartości – funkcja przewagi

Inną wartością możliwą do zastosowania w podejściu do uczenia przez wzmocnienie opartym na wartości jest tzw. **funkcja przewagi akcji (ang. action-adantage function)**.

Wartość ta obliczana jest według następującego wzoru (oznaczana jako $A(S, a)$):

$$A(S, a) = Q(S, a) - V(S)$$

Jest to podejście pozwalające na skupieniu się na **wartości dodanej podjęcia akcji a w stanie S** .

Ważną informacją praktyczną jest fakt, że bardzo często posłużenie się wartością $A(S, a)$ pozwala na zaprojektowanie algorytmów, które są **znacznie stabilniejsze i odporniejsze na oscylacje w trakcie treningu**.

Podejścia oparte na strategii

Podobnie jak algorytmy DQN można zaklasyfikować do kategorii algorytmów bazujących na wartości, tak algorytmy bazujące na **głębokim gradiencie strategii** (DPG) mogą być zaklasyfikowane do szerszej grupy **algorytmów opartych na strategii**.

W przypadku algorytmów opartych na strategii nadal sens ma podział na algorytmy bazujące na strategii **deterministycznej i stochastycznej**.

W ogólności algorytmy te bardzo często cierpią w swojej najprostszej postaci na problem z **dużą zmiennością gradientów wykorzystywanych do wytrenowania strategii**.

Możliwe jest posłużenie się pojęciami wywodzącymi się z definicji algorytmów opartych na wartościach do **usprawnienia algorytmu DPG**, np. algorytmu REINFORCE.

Podejścia oparte na wartości – przewaga stanu

Oryginalne równanie aktualizacji wag w algorytmie REINFORCE ma następującą postać:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + l_r \gamma^n G_n \nabla_{\boldsymbol{\theta}} \ln \Pi(a_n | S_n; \boldsymbol{\theta}).$$

Zamiast wartości zwrotu G_n możliwe jest posłużenie się wartością $A(S, a)$, co polepsza stabilność algorytmu w trakcie treningu:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + l_r \gamma^n A(S_n, a_n) \nabla_{\boldsymbol{\theta}} \ln \Pi(a_n | S_n; \boldsymbol{\theta}).$$

Zwiększenie stabilności odbywa się w tym przypadku poprzez **zmniejszenie wariancji gradientów** obliczanych przez algorytm w kolejnych krokach treningu.

Analiza zmienności wielkości gradientów w algorytmie REINFORCE

Na przykładzie kodu z notatnika `DL_14_01_reinforce_gradvar.ipynb`

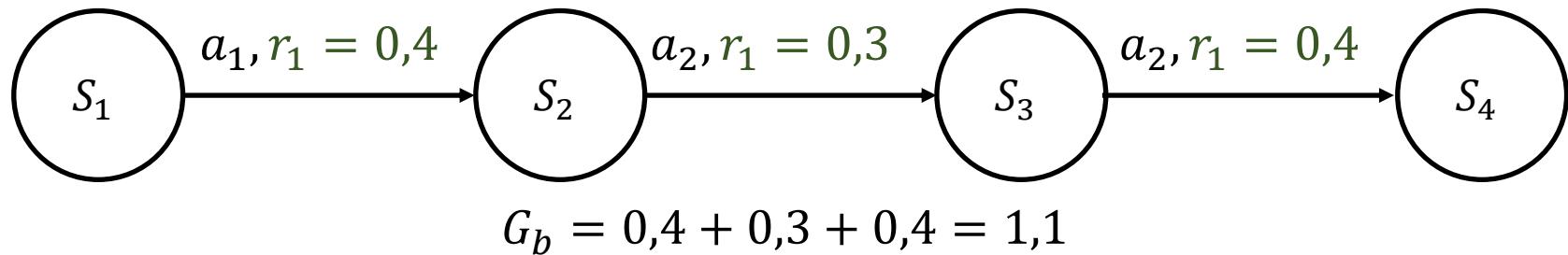
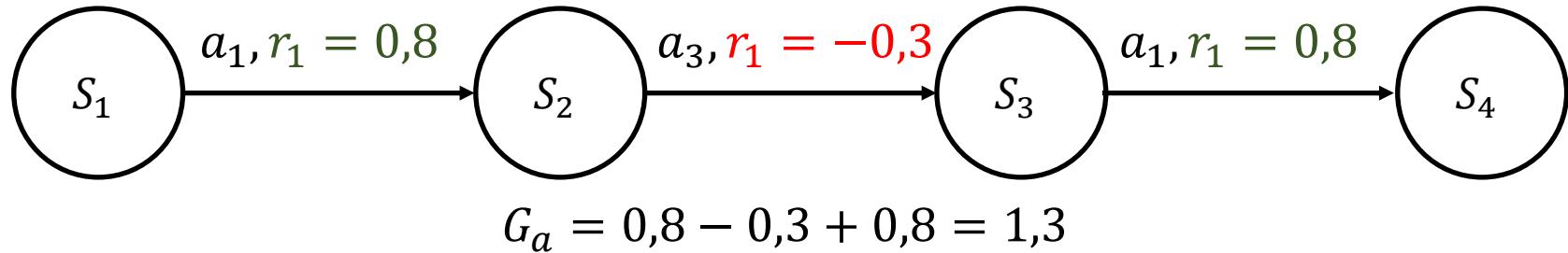
Trening – różnice czasowe i Monte Carlo

Innym ważnym rozróżnieniem jest rodzaj treningu algorytmu ze względu na **sposób aktualizacji parametrów** modelu sterującego agentem:

- trening bazujący na **różnicach czasowych (ang. *temporal difference*)**, który dokonuje uaktualnienia wag sieci neuronowej (lub innej struktury, np. macierzy Q-wartości) **co każdą podjętą akcję**. Przykładem takiego algorytmu jest metoda **Q-learningu**, czy też bazująca na głębokich **Q-sieciach**,
- trening **Monte Carlo**, który zakłada najpierw wykonanie przez agenta **pełnej interakcji ze środowiskiem**, a dopiero potem – aktualizację wag modelu. Przykładem tego typu sposobu treningu agenta jest algorytm **REINFORCE**.

Trening Monte Carlo ma istotną wadę – konieczne jest dokonanie uśrednienia obserwacji z całego epizodu. Może to powodować **włączenie do obliczanych gradientów pojedynczych złych decyzji należących do epizodów o wysokim zwrocie**.

Trening – różnice czasowe i Monte Carlo



Przykład faworyzowania przez wartość zysku G epizodu zawierającego niekorzystne akcje (na górze) względem epizodu zawierającego tylko korzystne akcje (na dole).

Algorytmy typu on-policy i off-policy

Ze względu na **sposób aktualizacji wag modelu** (lub innych jego parametrów) możliwe jest podzielenie zarówno algorytmów bazujących na wartościach jak i na strategii na dwie grupy algorytmów:

- algorytmy dokonujące aktualizacji **zgodnie ze strategią (ang. on-policy algorithms)**, które do aktualizacji parametrów modelu wykorzystują regułę obliczania akcji **identyczną** jak strategia stosowana przez agenta w interakcji ze środowiskiem,
- algorytmy dokonujące aktualizacji **niezgodnie ze strategią (ang. off-policy algorithms)**, które do aktualizacji modelu sterującego akcjami agenta wykorzystują regułę obliczania akcji **inną** niż ta zawarta w strategii stosowanej przez agenta do interakcji ze środowiskiem,

Czasami stosowanie algorytmów w wariancie off-policy pozwala na **polepszenie zbieżności treningu i zwiększenie nacisku na eksplorację środowiska**.

Q-learning typu on-policy i off-policy

W swoim klasycznym sformułowaniu **algorytm Q-learningu jest algorytmem typu off policy**.

Przykładowo – możemy do interakcji ze środowiskiem stosować strategię ϵ -chciwą, jednak równanie aktualizacji wierszy macierzy Q-wartości ma postać:

$$Q(S_n, a_n) = (1 - l_r) \cdot Q(S_n, a_n) + l_r \cdot \left[r_n + \gamma \cdot \max_{a_{n+1}} \{Q(S_{n+1}, a_{n+1})\} \right],$$

gdzie człon $\max_{a_{n+1}} \{Q(S_{n+1}, a_{n+1})\}$ oznacza tak naprawdę zastosowanie reguły wyboru akcji charakterystycznej dla **strategii chciwej**.

Jeżeli algorytm znowu miałby stać się algorytmem typu **on-policy**, to **człon $\max_{a_{n+1}} \{Q(S_{n+1}, a_{n+1})\}$ powinien zostać zastąpiony wyborem akcji według zasad strategii ϵ -chciwej**.

Przykład algorytmu typu Q-learning w wariantach on-policy i off-policy

Na przykładzie kodu z notatnika `DL_14_02_on_and_off_policy.ipynb`

DPG typu on-policy i off-policy

Podobnie dla algorytmu **DPG** (w wariancie REINFORCE) **możliwe jest określenie, że jest to algorytm typu on-policy**, gdyż strategia której agent się uczy w trakcie interakcji ze środowiskiem jest jednocześnie tą samą strategią, którą ten agent wykorzystuje do podejmowania decyzji, które interakcje podjąć.

W ogólności można jednak sobie wyobrazić sytuację, w której taki agent **uczy się strategii $\Pi(S, a)$** , jednak **do interakcji ze środowiskiem wykorzystuje inną strategię** którą możemy oznaczyć jako $\beta(S, a)$.

W takim przypadku można określić funkcję nagrody skumulowanej jako:

$$J(\theta) = \sum_{S \in D_S} d_\beta(S; \theta) \sum_{a \in A} \Pi(a|S; (\theta)) \cdot Q(S, a).$$

DPG typu on-policy i off-policy

Ważną różnicą względem oryginalnego sformułowania algorytmu DPG jest użycie funkcji przejść między stanami $d_\beta(S; \theta)$ (wykonującej strategię $\beta(S, a)$) zamiast funkcji $d(S; \theta)$ (wykonującej strategię $\Pi(S, a)$).

Jednocześnie w równaniu nadal wprost wykorzystywane są wartości strategii $\Pi(S, a)$.

Powoduje to, że efektywnie **gradient $\nabla_\theta J(\theta)$ wykorzystany do wytrenowania agenta ostatecznie przybiera postać zależną od dwóch strategii:**

$$\nabla_\theta J(\theta) = E_\beta \left\{ \frac{\Pi(a|S; \theta)}{\beta(S, a)} \cdot Q(S, a) \cdot \nabla_\theta \ln \Pi(a|S; \theta) \right\}.$$

DPG typu on-policy i off-policy

Takie sformułowanie algorytmu pozwala na przykład na **zbieranie próbek zgodnie z arbitralną strategią $\beta(S, a)$** .

Możliwe jest wstępne **gromadzenie danych** do treningu wstępnego agenta za pomocą uproszczonego algorytmu interakcji ze środowiskiem.

Dodatkowo – możliwość niezależnego zbierania danych treningowych **umożliwia także korzystanie z pamięci powtórek** do trenowania algorytmu DPG wraz ze wszystkimi zaletami takiego podejścia.

Wybór odpowiedniej (np. w skrajnym przypadku – losowej) strategii $\beta(S, a)$ pozwala na wczesnych etapach treningu na **zwiększenie nacisku na eksplorację środowiska**.

Priorytetowa pamięć powtórek

Kolejnym usprawnieniem możliwym do wprowadzenia aby polepszyć zarówno działanie algorytmów DQN (i podobnych), jak i algorytmów DPG w wariancie off-policy jest **priorytetowa pamięć powtórek (ang. prioritized experience replay, PER)**.

Problemem w przypadku zwykłej pamięci powtórek jest fakt, że **każdy przykład z tej pamięci ma taką samą szansę do bycia wybranym** do treningu sieci uczącej się strategii interakcji ze środowiskiem.

Podejście takie **nie uwzględnia faktu, że niektóre akcje są ważniejsze niż inne**. Przykładowo – w grze polegającej na odnalezieniu klucza do zablokowanych drzwi akcja prowadząca do podniesienia klucza powinna być znacznie ważniejsza i częściej uwzględniana w treningu niż akcje powodujące losowy ruch agenta.

Konieczne jest **zróżnicowanie prawdopodobieństw**, z jakimi przykłady wybierane są do treningu agenta.

Priorytetowa pamięć powtórek

Postępowanie takie wprowadza do algorytmu wyboru przykładów treningowych **preferencję (ang. bias) względem bardziej ważniejszych przykładów.**

Kryterium doboru jest w tym przypadku **wielkość błędu oszacowania wartości Q** (oznaczonego jako δ_n), który ma następującą postać:

$$\delta_n = \left[r_n + \gamma \max_{a_{n+1}} Q(S_{n+1}, a_{n+1}; \theta_n) \right] - Q(S_n, a_n; \theta_n)$$

Jest to swoista miara tego, **jak bardzo dany przykład jest nieznany agentowi.** Znając δ_n możliwe jest zdefiniowanie priorytetu p wyboru danego przykładu treningowego jako:

$$p_n = |\delta_n| + e,$$

gdzie e jest niewielkich rozmiarów stałą zapewniającą, że **żadne zdarzenie nie będzie mieć zerowego prawdopodobieństwa selekcji.**

Priorytetowa pamięć powtórek

Prawdopodobieństwo selekcji przykładu możliwe jest następnie do zdefiniowania za pomocą obliczonych wartości priorytetów jako:

$$P(i) = \frac{p_i^a}{\sum_k p_k^a},$$

gdzie $P(i)$ jest prawdopodobieństwem selekcji i -tego zdarzenia, a a jest parametrem wpływającym na zróżnicowanie prawdopodobieństw selekcji zdarzeń bardziej i mniej prawdopodobnych.

Efektem takiego postępowania może być **wielokrotne wybieranie tych samych przykładów do treningu**.

Pozwala to na **skupienie się na najważniejszych przykładach** treningowych, jednak skutkiem takiej sytuacji jest **zwiększone ryzyko przetrenowania** modelu.

Priorytetowa pamięć powtórek

Aby zminimalizować ryzyko przetrenowania stosowany jest dodatkowy sposób uzależniania wpływu gradientów obliczanych dla każdego przykładu treningowego na wagi trenowanej sieci neuronowej.

Ważenie gradientów przeprowadzane jest według następującego wzoru:

$$k_{trn}(i) = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^b,$$

gdzie $k_{trn}(i)$ jest wagą gradientu pochodzącego od i -tego gradientu, N jest liczbą przykładów w pamięci powtórek, a b jest parametrem wpływającym na zróżnicowanie wag gradientów.

Zwykle wartość b jest bliska零 na początku treningu i wraz z upływem czasu wzrasta i osiąga wartość równą 1.

Wady i zalety architektur DQN i DPG

Zarówno podejście DQN, jak i DPG posiadają z punktu użytkownika szereg charakterystycznych zalet, jak i wad.

Algorytmy z rodziny DQN pozwalają na proste **włączenie w trening pamięci powtórek i częstą aktualizację wag modelu**, co z kolei pozwala na uniknięcie sytuacji w których uczymy model na danych uśrednionych.

Jednocześnie algorytmy z rodziny **DQN nie radzą sobie z sytuacją w której potrzebna jest strategia stochastyczna**. Algorytmy z rodziny **DPG** są za to zwykle prostsze w implementacji, potrafią realizować strategie stochastyczne i dodatkowo często mają lepsze własności jeśli chodzi o zbieżność treningu.

Ze względu na różnice pomiędzy tymi dwoma wymienionymi rodzinami algorytmów bardzo kuszącą perspektywą jest możliwość zdefiniowania **rozwiązania hybrydowego, jakim jest algorytm typu aktor-krytyk**.

Algorytm aktor-krytyk

Śledząc uważnie matematyczne wyprowadzenie podstaw na których opiera się algorytm DPG warto zwrócić uwagę na **zapis gradientu nagrody skumulowanej** w postaci:

$$\nabla_{\theta} J(\theta_a) = E_{\Pi}\{Q(S, a) \cdot \nabla_{\theta} \ln(\Pi(S_n, a_n; \theta_a))\}.$$

W klasycznym podejściu uznajemy, że sieć neuronowa oblicza wartości funkcji $\Pi(S_n, a_n; \theta_a)$, czyli **strategii**. W podejściu aktor-krytyk (*ang. actor-critic*) sieć ta nazywana jest **aktorem**.

Możliwe jest jednak **wprowadzenie drugiej sieci neuronowej**, która obliczać będzie wartość $Q(S, a)$, którą możemy nazwać **krytykiem**.

Mamy zatem do czynienia z treningiem dwóch sieci neuronowych obliczających wartości funkcji $\Pi(S_n, a_n; \theta_a)$ oraz $Q(S, a; \theta_c)$, gdzie θ_a oznacza wagi sieci aktora, a θ_c oznacza wagi sieci krytyka.

Algorytm aktor-krytyk

Zgodnie z tą logiką aktualizacja wag **aktora** (odpowiedzialnego za **strategię**) ma postać:

$$\Delta\theta_a = \alpha \nabla_{\theta_a} \log(\Pi(S_n, a_n; \theta_a)) \cdot Q(S_n, a_n; \theta_c).$$

Dla wag sieci neuronowej realizującej **algorytm krytyka** aktualizacja wag ma postać:

$$\Delta\theta_c = \beta(r_n + \gamma Q(S_{n+1}, a_{n+1}; \theta_c) - Q(S_n, a_n; \theta_c)) \cdot \nabla_{\theta_c} Q(S_n, a_n; \theta_c).$$

Współczynniki szybkości nauki aktora i krytyka oznaczone są kolejno jako α i β .

Ze względu na fakt, że **nie musimy już obliczać wartości zwrotu** G_n , a możemy posługiwać się wartościami $Q(S_n, a_n; \theta_c)$. Jest to bardzo korzystny fakt, bo pozwala to na **stosowanie poprawki $\Delta\theta_a$ co krok algorytmu, a nie co epizod**.

Z tego względu **wykonanie algorytmu aktor-krytyk polega na obliczaniu i aplikowaniu naprzemiennie poprawki $\Delta\theta_a$, a następnie $\Delta\theta_c$** .

Algorytm aktor-krytyk

Pojedynczy krok algorytmu aktor-krytyk składa się z następujących etapów:

1. najpierw na podstawie stanu S_n sieć aktora oblicza wartość strategii, czego efektem jest akcja a , sieć krytyka oblicza wartość $Q(S_n, a_n; \theta_c)$,
2. agent wykonuje akcję a i otrzymuje od środowiska nową wartość stanu S_{n+1} i nagrodę r_n
3. sieć krytyka oblicza wartość $Q(S_{n+1}, a_{n+1}; \theta_c)$,
4. sieć aktora oblicza wartość aktualizacji $\Delta\theta_a$, odpowiednia aktualizacja aplikowana są przez algorytm optymalizatora,
5. sieć krytyka oblicza wartość aktualizacji $\Delta\theta_c$, odpowiednia aktualizacja aplikowana są przez algorytm optymalizatora.

Algorytm advantage actor-critic (A2C)

Możliwe jest **poprawienie zbieżności treningu** algorytmu typu aktor krytyk poprzez wykorzystanie w treningu funkcji przewagi (podobnie jak w omawianym wcześniej przypadku algorytmu REINFORCE).

Jest to tzw. algorytm aktor-krytyk bazujący na przewadze (ang. advantage actor-critic, A2C).

Z tego względu często do obliczania aktualizacji wag wykorzystywana jest wartość $A(S, a)$ obliczana według następującej zależności:

$$A(S, a) = Q(S, a) - V(S_n) = r_n + \gamma V(S_{n+1}) - V(S_n).$$

Funkcję przewagi można interpretować jako **nadwyżkę nagrody** faktycznej liczonej jako $r_n + \gamma V(S_{n+1})$ względem nagrody przewidywanej przez agenta, która ma wartość $V(S_n)$.

Algorytm advantage actor-critic (A2C)

W algorytmie A2C aktualizacja wag agenta przybiera następującą postać:

$$\Delta_{\theta_a} = \nabla_{\theta_a} \log(\Pi(S_n, a_n; \theta_a)) \cdot (r_n + \gamma V_{\theta_c}(S_{n+1}) - V_{\theta_c}(S_n)).$$

Natomiast aktualizacja wag krytyka ma postać:

$$\Delta_{\theta_c} = 2(r_n + \gamma V_{\theta_c}(S_{n+1}) - V_{\theta_c}(S_n)) \nabla_{\theta_c} (r_n + \gamma V_{\theta_c}(S_{n+1}) - V_{\theta_c}(S_n)).$$

Jak widać w przypadku algorytmu A2C wykorzystywana jest nie Q-wartość, a estymata wartości stanu $V_{\theta_c}(S_n)$.

Algorytm A2C często wykonywany jest **równolegle na wielu instancjach środowiska** i poprzez wykorzystanie wielu agentów działających równolegle.

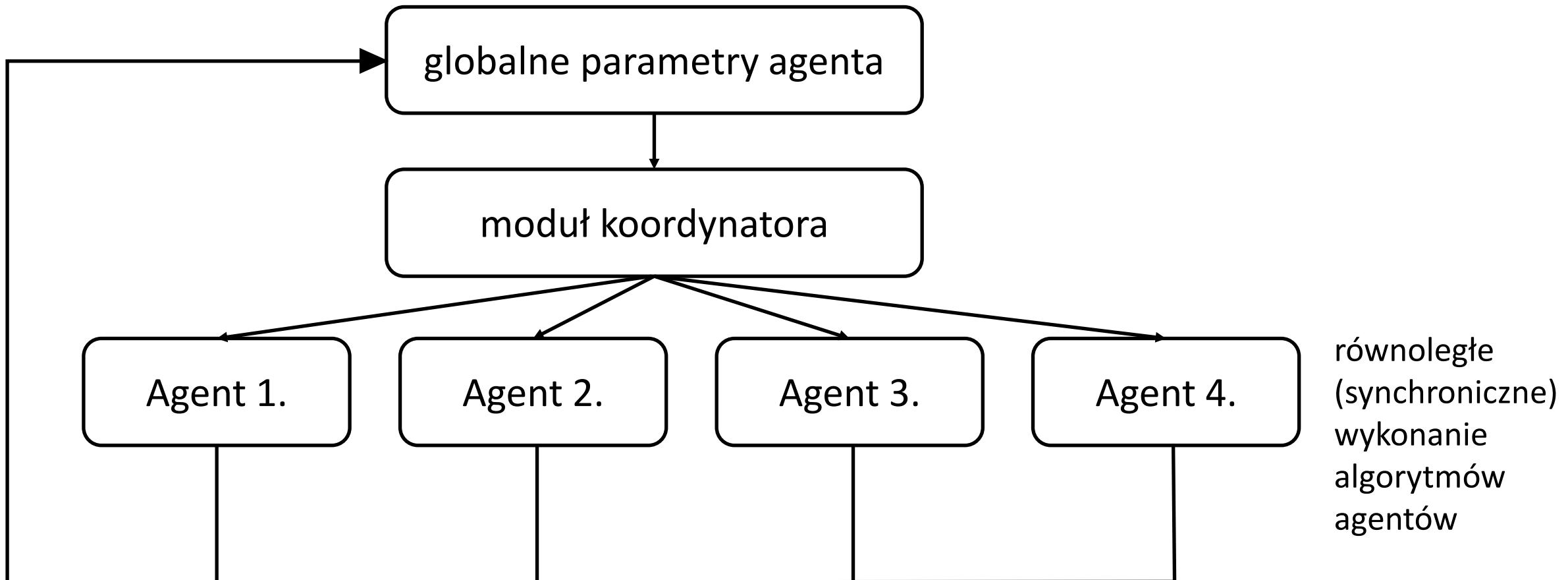
Algorytm advantage actor-critic (A2C)

Uśrednione poprawki będące efektem równoległego wykonania algorytmu każdego agenta aplikowane są do wag sieci synchronicznie w momentach gdy wszyscy agenci zakończą interakcję ze środowiskiem.

Nad synchronizacją wykonania algorytmów agentów czuwa specjalny **moduł koordynujący**.

Po aktualizacji głównych wag sieci te znowu kopiowane są do każdego agenta i proces równoległej iteracji agentów z kopiami wybranego środowiska docelowego zaczyna się od początku.

Algorytm advantage actor-critic (A2C)



Przykład równoległego wykonania algorytmu A2C przez wielu agentów.

Asynchroniczny algorytm A2C (A3C)

Poprzez modyfikację sposobu równoległego treningu wielu agentów uczenia ze wzmocnieniem możliwe jest zdefiniowanie **asynchronicznej wersji algorytmu A2C określonej skrótwcem A3C (ang. asynchronous advantage actor-critic)**.

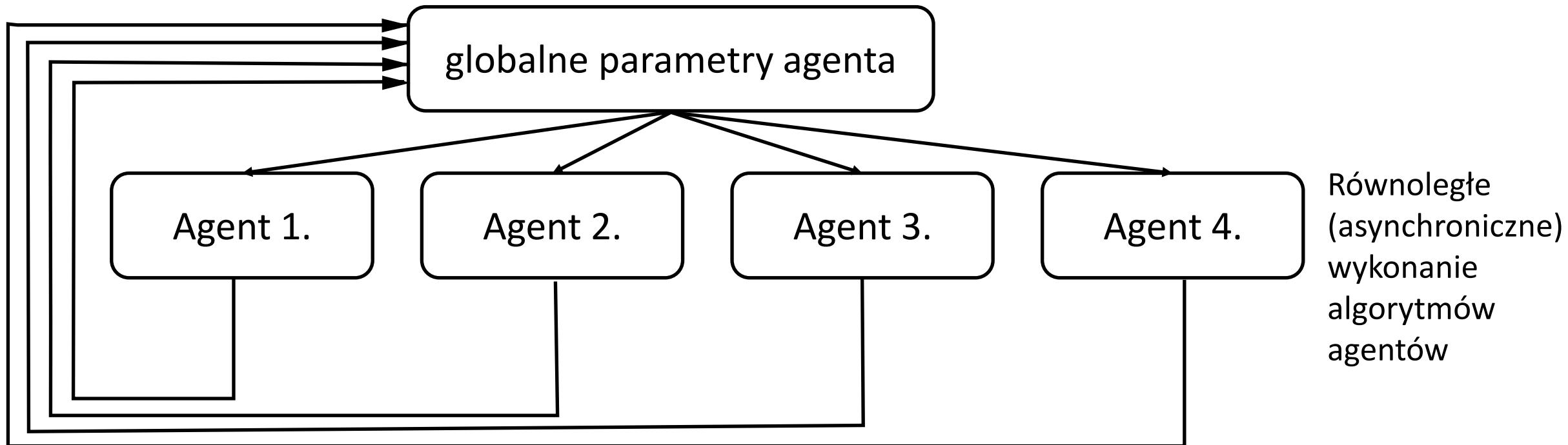
Różnicą pomiędzy algorytmem A2C i A3C jest fakt, że w algorytmie A3C **aktualizacje globalnych parametrów agenta przeprowadzane są asynchronicznie**.

Podobnie w asynchroniczny, **niezsynchonizowany** sposób odbywa się trening agentów wchodzących w interakcję ze środowiskiem.

Skutkiem takiej sytuacji jest fakt, że **w tym samym czasie mogą trwać epizody agentów mających różne wartości parametrów sieci**.

Algorytm **A3C w ogólności jest mniej wydajny niż algorytm A2C** i często lepszym wyborem jest właśnie algorytm A2C. Zwłaszcza, gdy trening odbywa się na dużych grupach (*ang. batches*) danych treningowych.

Asynchroniczny algorytm A2C (A3C)



Przykład równoległego wykonania algorytmu A3C przez wielu agentów.

Proximal Policy Optimization (PPO)

Możliwe jest dalsze rozwinięcie algorytmów klasy aktor-krytyk. Jedną z tego typu modyfikacji jest **algorytm PPO** (*ang. proximal policy optimization*), który bardzo często jest uważany za **algorytm wyznaczający standard** metod uczenia ze wzmocnieniem.

Jest to algorytm który z sukcesem został **zaaplikowany między innymi do sterowania agentami grającymi w złożone gry komputerowe (np. OpenAI Five)**.

Algorytm ten bazuje na obserwacji, że **znaczna zmienność wartości gradientów w trakcie treningu sieci agenta może wpływać degradującą na efekty treningu**. Z tego względu konieczne jest zredukowanie tej zmienności.

Konieczne jest **uniemożliwienie gwałtownych zmian wartości gradientu**, które mogłyby nastąpić z iteracji na iterację w trakcie treningu agenta.

Proximal Policy Optimization (PPO)

Aby zrealizować cel polegający na redukcji zmienności gradientu wprowadza się **współczynnik prawdopodobieństw** $r_n(\theta)$ obliczany jako:

$$r_n(\theta) = \frac{\Pi(S_n, a_n, \theta_a)}{\Pi(S_n, a_n, \theta'_a)},$$

gdzie θ_a oznacza **wagi sieci określającej strategię agenta w obecnej iteracji** algorytmu treningu, a θ'_a oznacza **wagi z poprzedniej iteracji** treningu.

Ze względu na konieczność **uwzględnienia dwóch strategii** $\Pi(S_n, a_n, \theta_a)$ i $\Pi(S_n, a_n, \theta'_a)$.

Z tego względu w trakcie treningu **konieczne jest przechowywanie dwóch zestawów wag sieci neuronowej**.

Proximal Policy Optimization (PPO)

Na bazie definicji współczynnika $r_n(\theta_a)$ możliwe jest określenie **jak bardzo dana strategia zmienia się z kroku na krok treningu**.

Do definicji celu optymalizacji możliwe jest **zdefiniowanie funkcji celu, która uniemożliwia gwałtowne zmiany strategii agenta**. Funkcja celu ma następującą postać (**jest to tzw. zastępca, ograniczona funkcja celu – ang. clipped surrogate objective function**) :

$$L^{CLIP}(\theta_a) = E\{\min(r_n(\theta_a)A_n, \text{clip}(r_n(\theta_a), 1 - \epsilon, 1 + \epsilon)) \cdot A_n\},$$

gdzie:

$$\text{clip}(r_n(\theta_a), 1 - \epsilon, 1 + \epsilon) = \begin{cases} 1 + \epsilon & \text{dla } r_n(\theta_a) > 1 + \epsilon \\ r_n(\theta_a) & \text{dla } 1 - \epsilon \leq r_n(\theta_a) \leq 1 + \epsilon \\ 1 - \epsilon & \text{dla } r_n(\theta_a) < 1 - \epsilon \end{cases}.$$

Proximal Policy Optimization (PPO)

We wzorze na wartość celu $L^{CLIP}(\theta_a)$ przez ϵ oznaczony został **współczynnik przycinania parametru** $r_n(\theta_a)$. Domyślnie często ma on wartość około 0,2.

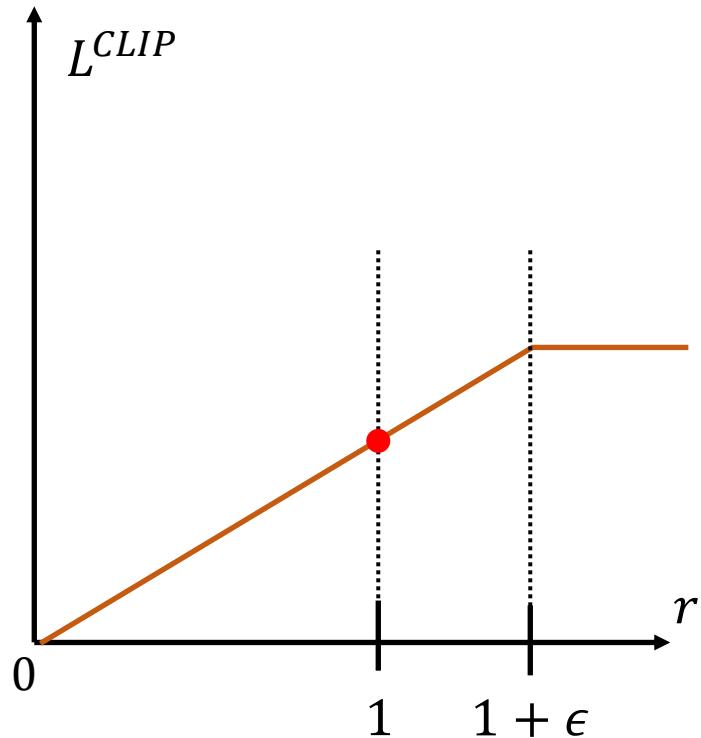
Warto zwrócić uwagę, że funkcja definiująca wartość $L^{CLIP}(\theta_a)$ zachowuje się inaczej w zależności od tego, czy wartość funkcji przewagi A_n jest dodatnia czy ujemna.

Postępowanie takie ma na celu wyłonienie tylko takich wartości $L^{CLIP}(\theta_a)$, które **faktycznie będą przyczyniać się do maksymalizacji wartości skumulowanej nagrody** pozyskiwanej przez agenta.

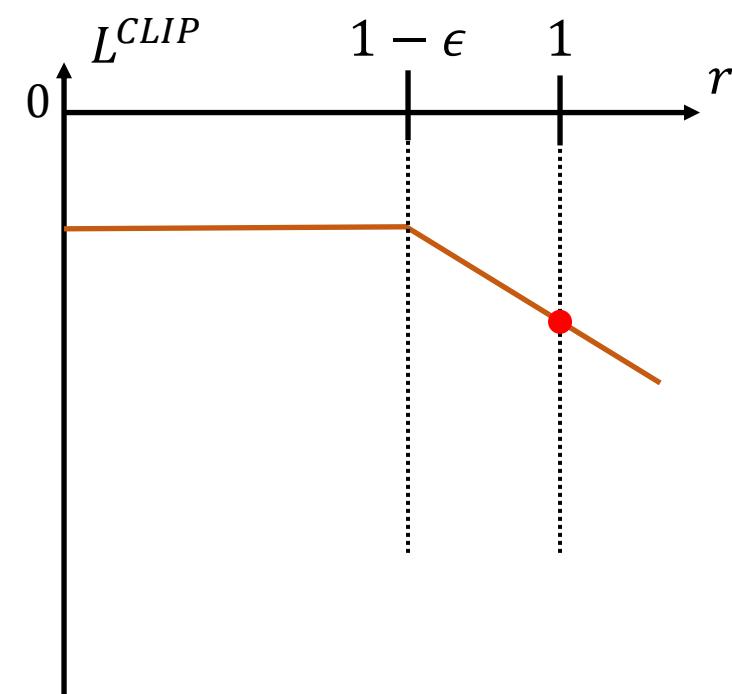
Jednocześnie istnieje **także gwarancja ograniczenia wartości gradientów** obliczanych przez algorytm treningu.

Proximal Policy Optimization (PPO)

$$A_n > 0$$



$$A_n < 0$$



Przykłady działania funkcji $\text{clip}(r_n(\theta), 1 - \epsilon, 1 + \epsilon)$ dla dodatniej i ujemnej wartości funkcji przewagi.

Proximal Policy Optimization (PPO)

Docelowo zazwyczaj oprócz celu $L^{CLIP}(\theta)$ wykorzystuje się także **dodatkowe elementy funkcji celu**. Pełna funkcja celu algorytmu PPO ma postać:

$$L^{PPO}(\theta_a) = L^{CLIP}(\theta_a) + c_1(V_{\theta_c}(S_n) - V^{\text{targ}})^2 + c_2 H(\Pi(S_n, a_n; \theta_a))$$

Oprócz wartości $L^{CLIP}(\theta_a)$ odpowiadającej za trening sieci realizującej strategię zawiera ona dwa dodatkowe człony.

Pierwszym dodatkowym członem jest wyrażenie $c_1(V_{\theta_c}(S_n) - V^{\text{targ}})^2$ które odpowiada za **trening sieci będącej krytykiem**. W przypadku algorytmu PPO krytyk dokonuje **predykcji wartości stanów**.

W wyrażeniu dotyczącym treningu krytyka wartość c_1 jest hiperparametrem mówiącym o **istotności elementu funkcji celu** dotyczącego krytyka. Wartość V^{targ} jest **docelową wartością stanu**, jaką przewidywać na krytyk.

Proximal Policy Optimization (PPO)

Drugi dodatkowy element stanowi wartość entropii rozkładu decyzji $\Pi(S_n, a_n; \theta_a)$ przemnożony przez wagę c_2 , która decyduje o **wadze członu entropii** w treningu aktora realizującego algorytm PPO.

Entropia obliczana jest za pomocą następującego wzoru:

$$H(\Pi(S_n, a_n; \theta_a)) = - \sum_{i=1}^n \Pi(S_n, a_n; \theta_a) \cdot \log(\Pi(S_n, a_n; \theta_a))$$

Entropia w trakcie treningu maleje, gdyż rozkłady prawdopodobieństwa generowane przez sieć aktora posiadają **coraz lepiej zarysowane, wyraźne maksimum**.

Wartość elementu $c_2 H(\Pi(S_n, a_n; \theta_a))$ maleje wraz z kolejnymi iteracjami treningu przez co zachowanie strategii upodabnia się do strategii ϵ -chciwej.

Proximal Policy Optimization (PPO)

Można to interpretować jako zwiększającą się „**pewność wyborów**” dokonywanych przez sieć aktora.

Dodanie członu związanego z entropią przyczynia się do **promowania zachowań agenta związanych z eksploracją środowiska**.

Wartość składowej funkcji celu bazującej na entropii jest **wysoka dla rozkładów równomiernych**, czyli takich w których sieć nie jest „pewna” swojego wyboru.

W takich momentach będzie ona dążyć do **zwiększenia tej pewności poprzez zwiększenie preferencji jednej z akcji**, jeśli tylko możliwe jest wybranie akcji **najbardziej korzystnej z punktu widzenia możliwej do otrzymania nagrody**.

Z drugiej strony **pewna doza niepewności (entropii rozkładu wybieranych akcji) w wyborze akcji jest pożądana**, gdyż preferencja taka **promuje eksplorację środowiska**.

Algorytm PPO i podejścia alternatywne

Algorytm PPO został wykorzystany do wielu skomplikowanych zastosowań i obecnie stanowi **algorytm wyznaczający standard** tego jak powinny działać i jakie skuteczności osiągać algorytmy ze wzmocnieniem.

Wszystkie omawiane na wykładzie algorytmy uczenia ze wzmocnieniem wymagają **ręcznie zaprojektowanej funkcji nagrody** i algorytm PPO także należy do tej kategorii.

Jedną z obiecujących dynamicznie rozwijających się dziedzin algorytmów są techniki **oparte na koncepcji tzw. ciekawości (ang. curiosity)**, która pozwala na obliczanie funkcji nagrody, której uczy się sam agent.

Jest to tak zwana nagroda wewnętrzna agenta (*ang. intrinsic reward*). Jest to szczególnie **przydatny mechanizm zwłaszcza w przypadku środowisk w których nagroda przyznawana jest agentowi rzadko** (np. Frozen Lake lub Montezuma's Revenge).

Literatura

1. Sutton, R., S., Barto, A., G., Reinforcement Learning: An Introduction, MIT Press, 1998.
2. Geron, A., Hands-On Machine Learning with Scikit-Learn & TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, Inc., 2019.
3. Deep RL Course - zbiór materiałów on-line dotyczących uczenia wzmacnianego dostępny pod adresem <https://simoninithomas.github.io/deep-rl-course/> (data dostępu: 12.01.2022)
4. Weng, L., Policy Gradient Algorithms, materiały on-line opisujące algorytmu bazujące na gradientach strategii, url: <https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>
5. Mnih V., Badia A., Mirza M., Graves A., Lillicrap T., Harley T., Silver D., Kavukcuoglu K., Asynchronous Methods for Deep Reinforcement Learning, Proceedings of The 33rd International Conference on Machine Learning, PMLR 48:1928-1937, 2016.
6. Schulman, J., et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
7. Pathak, D., Agrawal, P., Efros, A., Darrell, T., Curiosity-driven exploration by self-supervised prediction. International conference on machine learning. PMLR, 2017.
8. Burda, Y., Edwards, H., Storkey, A., Klimov, O., Exploration by random network distillation. arXiv preprint arXiv:1810.12894, 2018.

Dziękuję za uwagę



Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego

Program Operacyjny Polska Cyfrowa na lata 2014-2020.

Oś priorytetowa nr 3 „Cyfrowe kompetencje społeczeństwa”, działanie nr 3.2 „Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej”.

Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”.