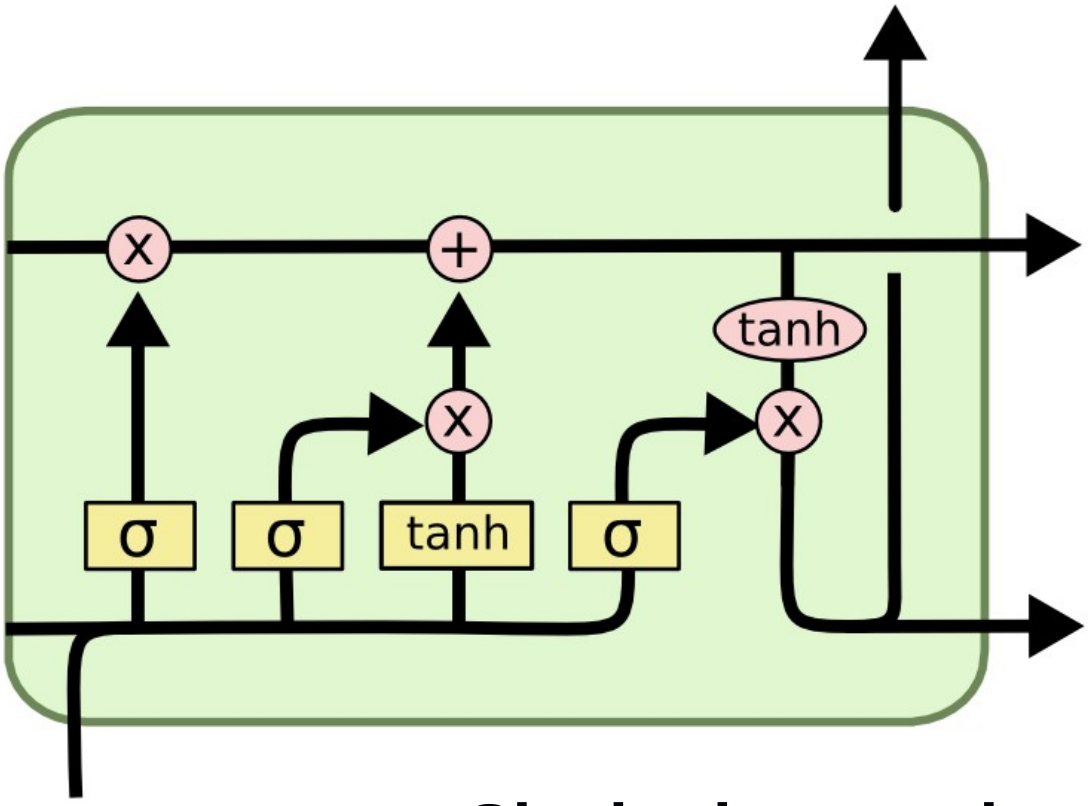
Deep Learning
Wykład 9



Sieci rekurencyjne:

- **RNN – recursive neural network**
- **LSTM – long short term memory**

Marcin Wolter, *IFJ PAN, 1 lutego 2021*

Informacyjnie

- Nasz przedostatni wykład.
- Proszę o projekty!!! Oraz o brakujące zadania

Głębokie uczenie	2018/19 ID stac.	5	Grupa la 02	la 02	26.01.2021 wtorek	08:00	10:15	WM_G_wirtualna	lab	L	Zaliczenie z oceną
Głębokie uczenie	2018/19 ID stac.	5	Grupa la 01	la 01	26.01.2021 wtorek	11:00	13:15	WM_G_wirtualna	lab	Zaj.	Zaliczenie z oceną
Głębokie uczenie	2018/19 ID stac.	5	Grupa wy 01	wy 01	27.01.2021 środa	12:15	14:30	WM_G_wirtualna	wyk	W	Egzamin
Głębokie uczenie	2018/19 ID stac.	5	Grupa wy 01	wy 01	01.02.2021 poniedziałek	12:15	14:30	WM_G_wirtualna	wyk	W	Egzamin
Głębokie uczenie	2018/19 ID stac.	5	Grupa wy 01	wy 01	03.02.2021 środa	12:15	14:30	WM_G_wirtualna	wyk	W	Egzamin

Rekurencyjne sieci neuronowe

- Ideą rekurencyjnych sieci neuronowych (ang. Recursive Neural Networks, RNNs) jest wykorzystanie informacji sekwencyjnych
  - w przypadku tradycyjnej sieci neuronowej zakładamy, że wszystkie wejścia (i wyjścia) są niezależne od siebie
  - często jest to złe założenie np. chcąc przewidzieć następny wyraz w zdaniu, lepiej zorientować się, które wyrazy pojawiły się przed nim
- RNN są nazywane rekurencyjnymi/powtarzającymi się, ponieważ wykonują to samo zadanie dla każdego elementu sekwencji, a dane wyjściowe zależą od poprzednich obliczeń
- Możemy też myśleć o RNN, że posiadają one ”pamięć”, która przechwytuje informacje o tym, co zostało obliczone do tej pory.

Zastosowania

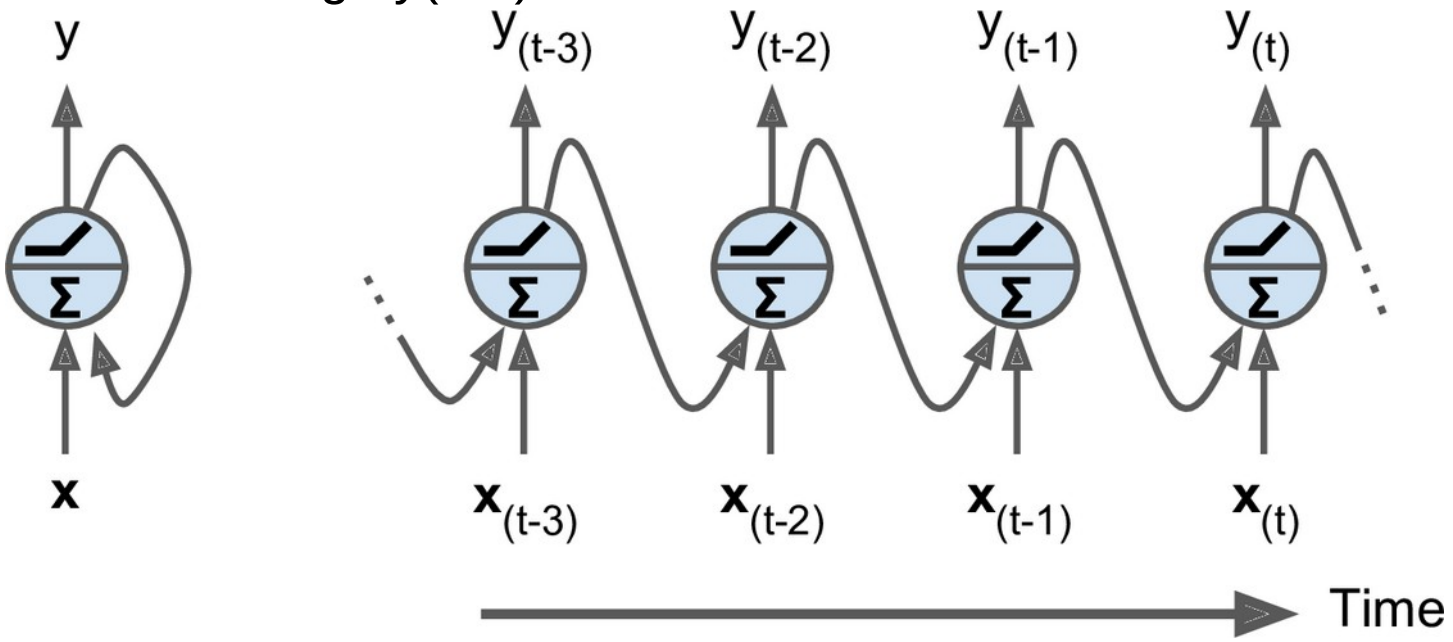
- Analiza szeregów czasowych, np. cen akcji w celu predykcji kupna/sprzedaży
- przewidywanie trajektorii samochodów autonomicznych
- przetwarzanie języka naturalnego, np.:
  - automatyczne tłumaczenie,
  - konwersja tekstu na mowę (speech-to-text)
  - analiza sentymentu

Co różni RNN od zwykłych sieci?

- W różnych przykładach, wejścia i wyjścia mogą mieć różną długość
- ”zwykłe” sieci neuronowe nie współdzielą cech nauczonych w różnych pozycjach tekstu
- sieci RNN mogą pracować na sekwencjach o dowolnej długości, a nie na wejściach o stałej wielkości, jak w przypadku ”zwykłych” sieci neuronowych

Neuron rekurencyjny

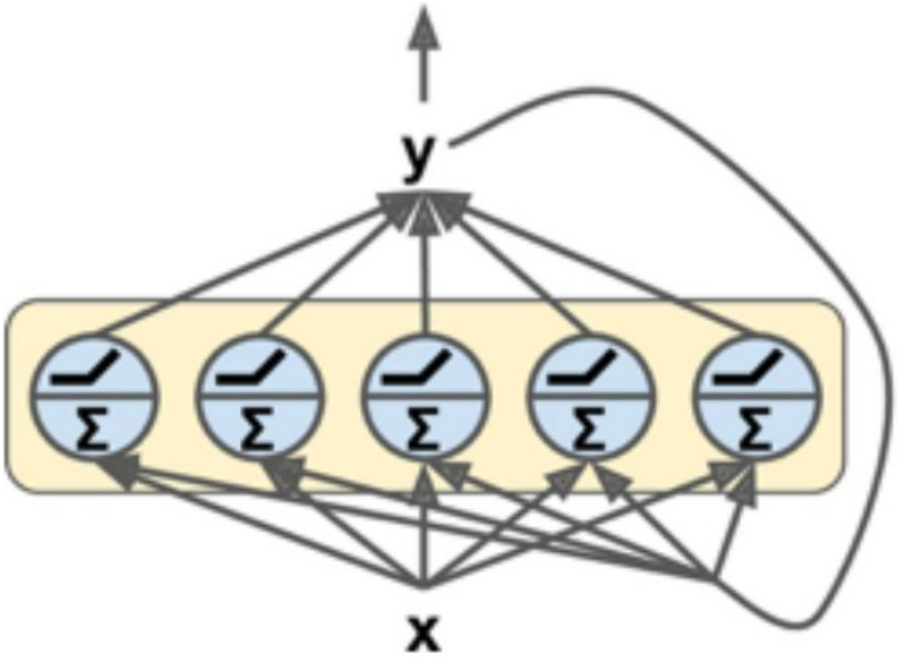
- Najprostszy neuron RNN:
  - W każdym kroku czasowym t (zwanym również ramką) ten neuron rekurencyjny odbiera wejścia x(t) oraz własne wyjście z poprzedniego kroku czasowego y(t−1)

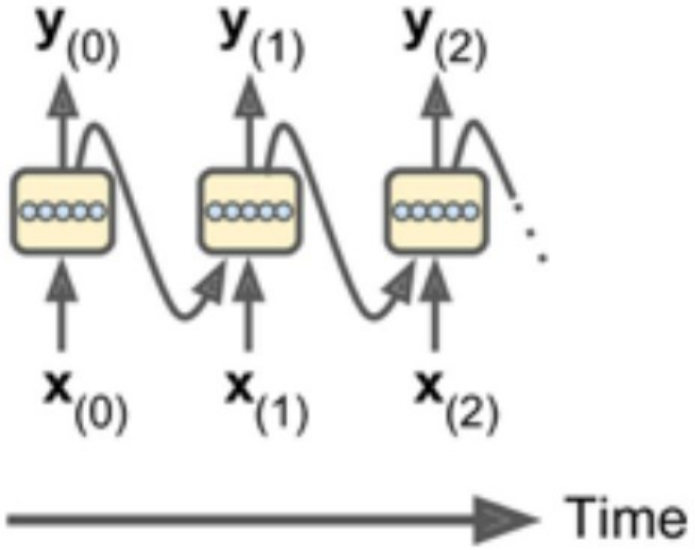


  - Możemy reprezentować tę małą sieć neuronową na osi czasu (neuron rekurencyjny rozwinięty w czasie).

Warstwa neuronów rekurencyjnych

- Można utworzyć warstwę neuronów rekurencyjnych
  - w każdym kroku t każdy neuron odbiera zarówno wektor wejściowy x(t), jak i wektor wyjściowy z poprzedniego kroku y(t−1)
  - zauważmy, że oba wejścia i wyjścia są teraz wektorami (kiedy był tylko jeden neuron, wynik był skalarem)





Zbudujmy najprostszą sieć RNN

- Nie używamy Keras, użyjemy tylko biblioteki numpy:
  - https://github.com/marcinwolter/DeepLearning_2020/blob/main/RNN_numpy.ipynb
- **Zadanie:**
  - Za pomocą serii działań arytmetycznych nauczyć sieć neuronową dodawania liczb zapisanych w systemie binarnym (liczby 1-bajtowe):

    [1 0 1 1 0 0 1 1] + [0 0 1 0 1 0 1 0] = [1 1 0 1 1 1 0 1]

    179 + 42 = 221
  - Sieć dostaje na wejście szereg losowych par liczb binarnych (zbiór treningowy X) i na wyjście wyniki dodawania (zbiór Y). Sieć przeprowadza regresję minimalizując średni błąd kwadratowy.
  - Testując sieć podajemy na wejście losowe pary liczb i porównujemy odpowiedź z oczekiwanym wynikiem.

UWAGA: porównujemy pojedyncze bity i uczymy sieć podając sekwencję bitów!

Warstwa rekurencyjna SimpleRNN w pakiecie Keras

- Prosta warstwy o nazwie *SimpleRNN* pakietu Keras:
  - *from keras.layers import SimpleRNN*
- Mała różnica: warstwa *SimpleRNN* dzieli sekwencje na wsady (operację taką wykonują wszystkie warstwy Keras), a nie przetwarza pojedynczych sekwencji tak, jak to miało miejsce w przykładzie zaimplementowanym za pomocą biblioteki Numpy. W związku z tym warstwa ta przyjmuje obiekty wejściowe o kształcie *(rozmiar_wsadu, kroki_czasu, cechy_wejściowe)*, a nie *(kroki_czasu, cechy_wejściowe)*.
- Warstwa *SimpleRNN*, podobnie jak wszystkie rekurencyjne warstwy pakietu Keras, może być uruchomiona w dwóch trybach: może zwracać pełne sekwencje kolejnych obiektów wyjściowych dla każdego kroku czasu (trójwymiarowe tensory o kształcie *(rozmiar_wsadu, kroki_czasu, cechy_wyjściowe)*) lub tylko ostatnie obiekty wyjściowe poszczególnych sekwencji wejściowych (dwuwymiarowe tensory o kształcie *(rozmiar wsadu, cechy_wyjściowe)*). Wybór trybu pracy jest dokonywany za pomocą argumentu *return_sequences*.

SimpleRNN

- UWAGA:
  - Ponieważ sieć uczy się dodawania dwóch liczb dostając sekwencyjnie pary bitów (bit n z pierwszej i drugiej liczby) ważna jest kolejność podawania bitów
  - Dodawanie można wykonać tylko jeśli zaczniemy od najmniej znaczącego bitu i w tej kolejności sieć musi dostawać bity.
  - Jeśli odwrócimy kolejność bitów sieć nie nauczy się dodawania (możemy sprawdzić)!

Problem zaniku pierwszych wejść

- Załóżmy, że chcemy przeprowadzić analizę sentymentów w długiej recenzji, która zaczyna się od słów “Kochałem ten film”, ale reszta recenzji wymienia wiele rzeczy, które mogłyby uczynić film jeszcze lepszym. Jeśli RNN stopniowo zapomni pierwszych trzech słów, całkowicie błędnie zinterpretuje recenzję.
- Rozwiązanie: różnego rodzaju komórki z długo-terminową pamięcią
  - na tyle skuteczne, że podstawowe komórki nie są już bardzo używane
  - najpopularniejsza: LSTM (ang. Long Short-Term Memory) (Sepp Hochreiter i Juergen Schmidhuber, 1997), następnie zmodyfikowana

    Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural co mputation 9.8 (1997): 1735-1780.

Warstwa Long Short-Term Memory (LSTM)

- Warstwa LSTM jest wersją SimpleRNN, do której dodano sposób przekazywania informacji między wieloma krokami czasu.
- Wyobraźmy sobie transmisyjny biegnący równolegle do przetwarzanej sekwencji. Informacje z sekwencji mogą w razie potrzeby w dowolnej chwili zostać przeniesione na ten pas i trafić do następnego kroku czasu bez żadnych modyfikacji.
- Taki mechanizm jest esencją algorytmu LSTM. **Informacje są zapisywane w celu późniejszego ich wykorzystania - starsze sygnały nie zanikają wraz z upływem czasu.**

Warstwa LSTM

- Zacznijmy od prostej komórki SimpleRNN (rysunek na następnej stronie).
- Wiele macierzy wag, litera „o” dodana do oznaczeń macierzy W i U (Wo i Uo) w celu oznaczenia obiektów wyjściowych (output)
  - W, U, V – wagi, patrz przykład https://github.com/marcinwolter/DeepLearning_2020/blob/main/RNN_numpy.ipynb
- Dodajmy mechanizm przepływu danych przenoszący informacje między krokami czasu: *Ct*. Wpływ na komórkę: dane zostaną połączone z wejściem i mechanizmem rekurencji (poprzez gęste przekształcenie: iloczyn skalarny z macierzy wag, dodawanie wartości progowej i zastosowanie funkcji aktywacji) oraz będą miały wpływ na stan przesyłany do kolejnego kroku czasu.
- Jest to sposób modulacji kolejnego obiektu wyjściowego *output* i kolejnego stanu *state* (rysunek na następnej stronie).

SimpleRNN ma problemy z pamiętaniem informacji sprzed wielu cykli

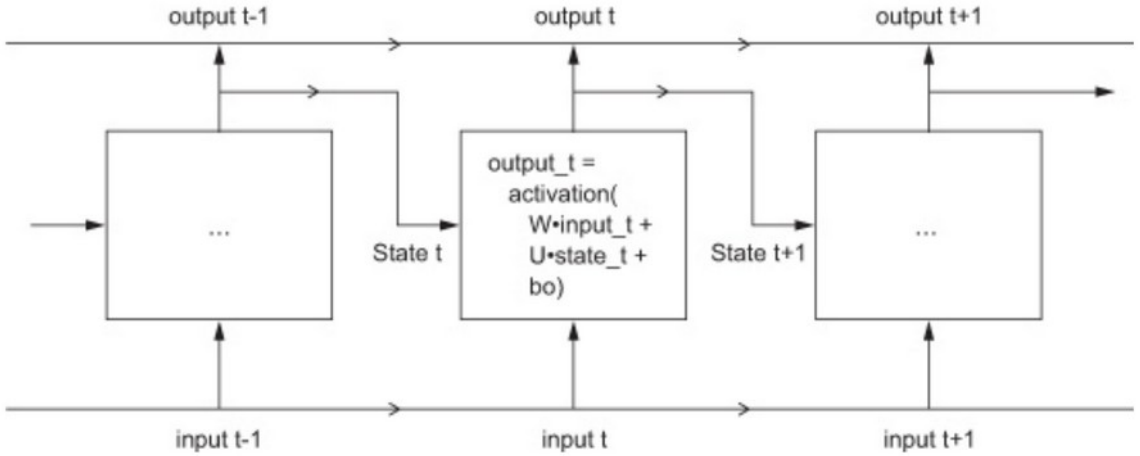


Figure 6.13 from Francois Chollet's book *Deep Learning with Python*

Dodajemy „pas transmisyjny” z zapamiętaną informacją

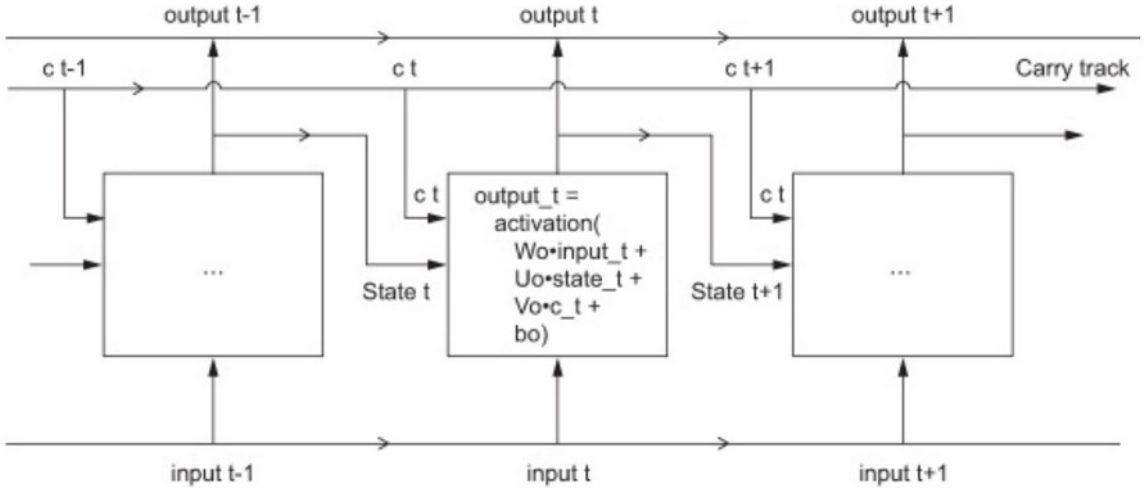


Figure 6.14 from Francois Chollet's book *Deep Learning with Python*

Warstwa LSTM

- Jeszcze nie uwzględniliśmy sposobu obliczania nowej przekazywanej wartości *c(t)*.
- Składa się on z trzech różnych przekształceń o postaci komórki SimpleRNN
- Wszystkie trzy transformacje mają własne macierze wag oznaczane indeksami w postaci liter *i*, *f* i *k*.
- Architektura algorytmu LSTM:
  - *output_t = activation(dot(state_t. Uo) + dot(input_t, Wo) + dot(Ct, Vo) + bo)*
  - *i_t = activation(dot(state_t, Ui) + dot (input_t, Wi) + bi)*
  - *f_t =activation(dot (state t. Uf) + dot (input t, Wf) + bf)*
  - *k_t = activation(dot (state t. Uk) + dot (input t. wk) + bk)*
- Następny obiekt c_t+1 uzyskujemy jako:
  - *c_t+1 = i_t*k_t + c_t*f_t*

Dodawanie nowych informacji

Wymazywanie zbędnych informacji

Patrz schemat na następnej stronie

LSTM: figure out how to update “carry” state

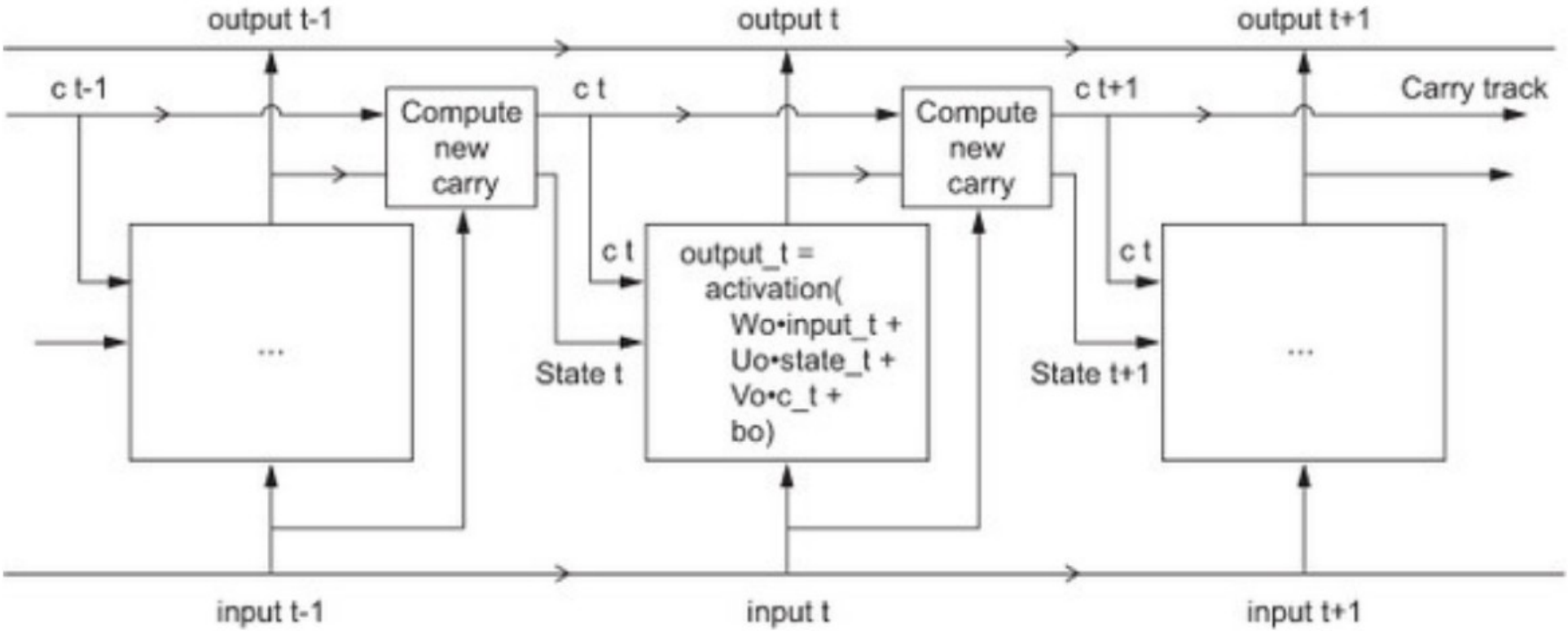
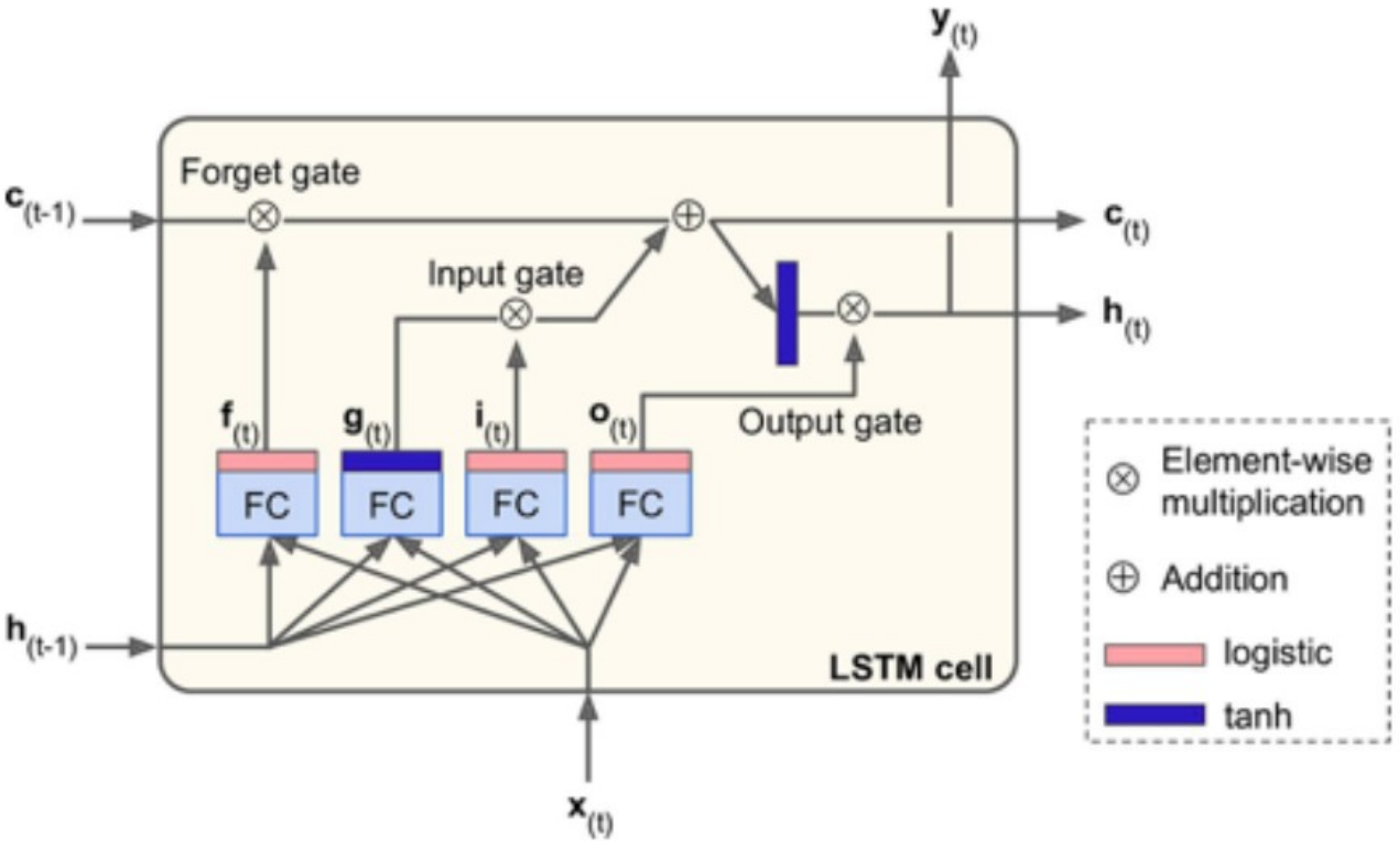


Figure 6.15 from Francois Chollet’s book *Deep Learning with Python*

Architektura LSTM

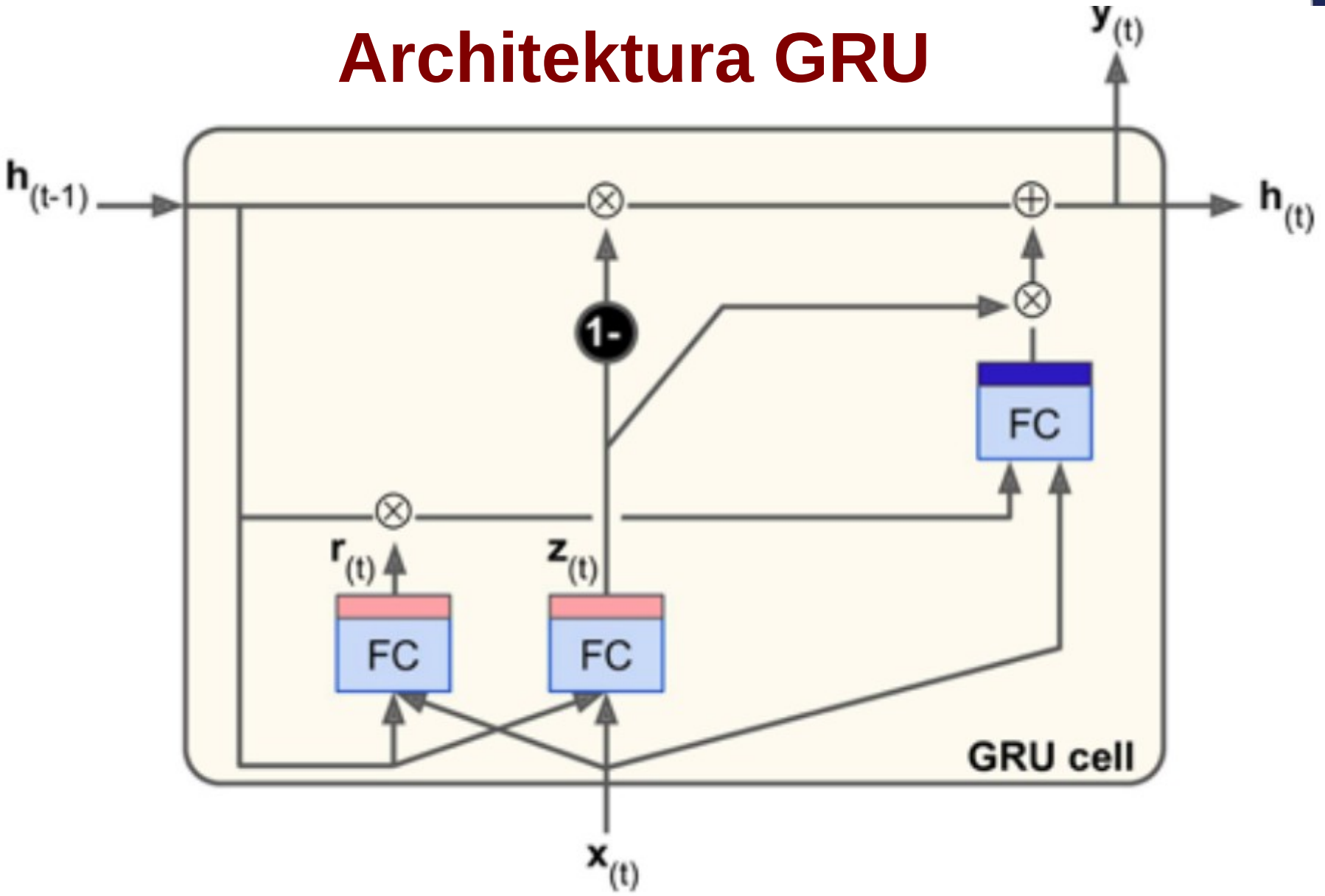


źródło: Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media, 2017.

Komórka GRU

- Komórka GRU (ang. Gated Recurrent Unit (GRU) (Kyunghyun Cho i inni, 2014) jest uproszczoną wersją LSTM:
  - oba wektory stanu są połączone w pojedynczy wektor h(t)
  - pojedynczy kontroler bramkowy kontroluje zarówno bramkę ”zapomnij”, jak i bramkę wejściową.
  - Jeśli kontroler bramki wyświetli wartość 1, bramka wejściowa jest otwarta, a bramka "zapomnij" jest zamknięta
  - Jeśli wynik wynosi 0, dzieje się odwrotnie
  - innymi słowy, kiedy pamięć musi być przechowywana, miejsce, w którym ma być zapisana, jest najpierw usuwane
  - nie ma bramki wyjściowej; pełen wektor stanu wyprowadzany jest za każdym razem. Istnieje jednak nowy kontroler bramki, który kontroluje, która część poprzedniego stanu zostanie pokazana głównej warstwie.

Architektura GRU

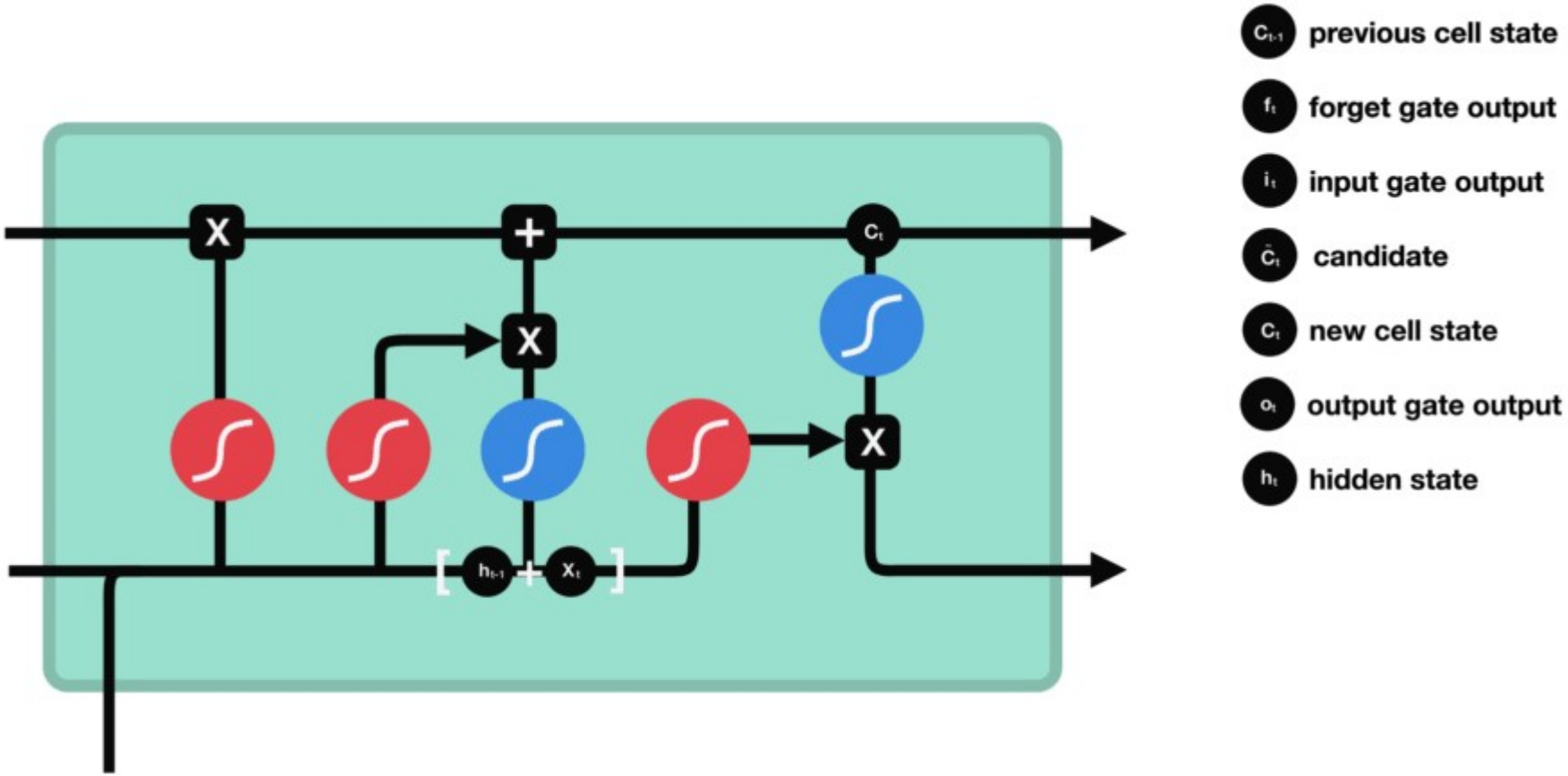


źródło: Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media, 2017.

Tutoriale

Tutorial o LSTM oraz GRU:

– https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21



Także:
https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

WIKIPEDIA

Zastosowanie LSTM

Success [edit]

There have been several successful stories of training, in a non-supervised fashion, RNNs with LSTM units.

In 2018, Bill Gates called it a “huge milestone in advancing artificial intelligence” when bots developed by OpenAI were able to beat humans in the game of Dota 2.[45] OpenAI Five consists of five independent but coordinated neural networks. Each network is trained by a policy gradient method without supervising teacher and contains a single-layer, 1024-unit Long-Short-Term-Memory that sees the current game state and emits actions through several possible action heads.[45]

In 2018, OpenAI also trained a similar LSTM by policy gradients to control a human-like robot hand that manipulates physical objects with unprecedented dexterity.[46]

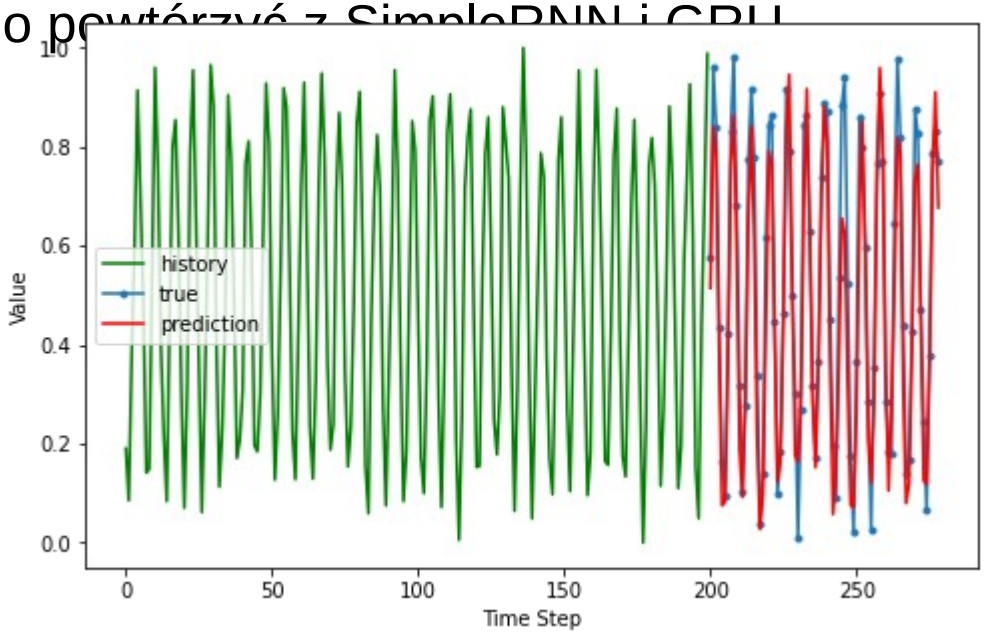
In 2019, DeepMind's program AlphaStar used a deep LSTM core to excel at the complex video game Starcraft II.[47] This was viewed as significant progress towards Artificial General Intelligence.[47]

Applications of LSTM include:

- Robot control[48]
- Time series prediction[44]
- Speech recognition[49][50][51]
- Rhythm learning[37]
- Music composition[52]
- Grammar learning[53][36][54]
- Handwriting recognition[55][56]
- Human action recognition[57]
- Sign language translation[58]
- Protein homology detection[59]
- Predicting subcellular localization of proteins[60]
- Time series anomaly detection[61]
- Several prediction tasks in the area of business process management[62]
- Prediction in medical care pathways[63]
- Semantic parsing[64]
- Object co-segmentation[65][66]
- Airport passenger management[67]
- Short-term traffic forecast[68]
- Drug design[69]

Przykłady

- Zastosowanie LSTM / GRU w naszym przykładzie:
  - https://github.com/marcinwolter/DeepLearning_2020/blob/main/RNN_numpy.ipynb
- Proste przewidywanie serii (funkcja sin(x) z szumem):
  - https://github.com/marcinwolter/DeepLearning_2020/blob/main/LSTM_time_series.ipynb
- Spróbujmy to samo powtórzyć z SimpleRNN i GRU



Inny przykład

- Bardzo rozbudowany przykład z książki F. Chollet „Deep Learning”:
  - https://github.com/marcinwolter/DeepLearning_2020/blob/main/6_3_advanced_usage_of_recurrent_neural_networks.ipynb
- Przewidywanie pogody na podstawie wielu zmiennych
- Bardzo długi czas potrzebny na obliczenia (zalecane zmniejszenie liczby epok)

Podsumowanie

- Poprzednie sieci neuronowe nie dysponowały pamięcią.
- Sieć rekurencyjna jest implementacją maszyny von Neumanna https://pl.wikipedia.org/wiki/Architektura_von_Neumanna
- Może służyć np. do analizy przebiegów czasowych (pamięta przeszłość)