Rekurencyjne sieci neuronowe

Grzegorz Madejski

Celem wykładu jest zrozumienie rekurencyjnych sieci neuronowych, i nauczenie ich generowania tekstu.

Na końcu wykładu pokażemy model sieci, który po długim treningu na książce „Alicja w Krainie Czarów” nauczył się mówić.

Fragment teksu:

said the king in a very grave voice, “until
all the jurymen are back in their proper places — _all_,”

Co dalej wygeneruje sieć?

said the king in a very grave voice, “until
all the jurymen are back in their proper places—_all_,”

“io you dan’t geon it ” said the daterpillar.

“iele you a doradu would ”our paje the sore,” said alice.

“whel i sas to colng,” the manch hare waid to alice to herself, “i don’t kekw it was in yhu a worlen of thene ”ou ael neae to tea

Część I: Rekurencyjne sieci neuronowe

Część II: Sieci LSTM

Część III: Generowanie tekstu poprzez LSTM

Jednokierunkowe sieci neuronowe

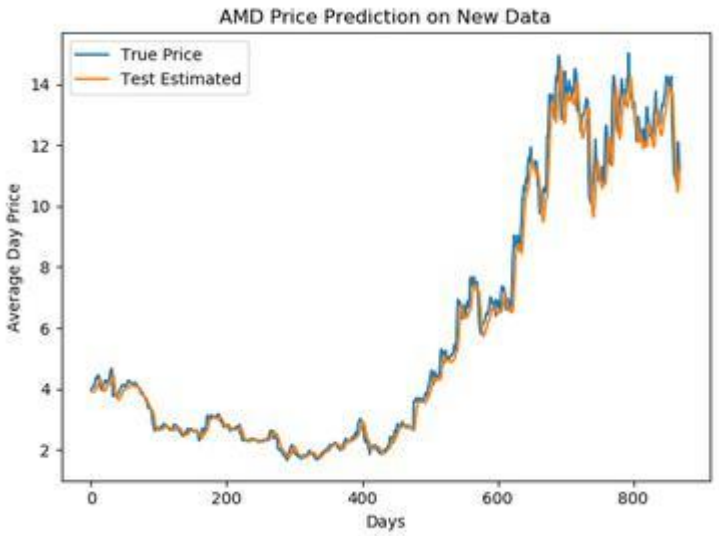
- Zwykłe (jednokierunkowe) sieci neuronowe przetwarzają próbkę danych „z danej chwili” i wyświetlają dla niej output.
- Nie są dostosowane do przetwarzania danych sekwencyjnych i szeregów czasowych.
- Nie przechowują informacji o poprzedniej próbce danych.

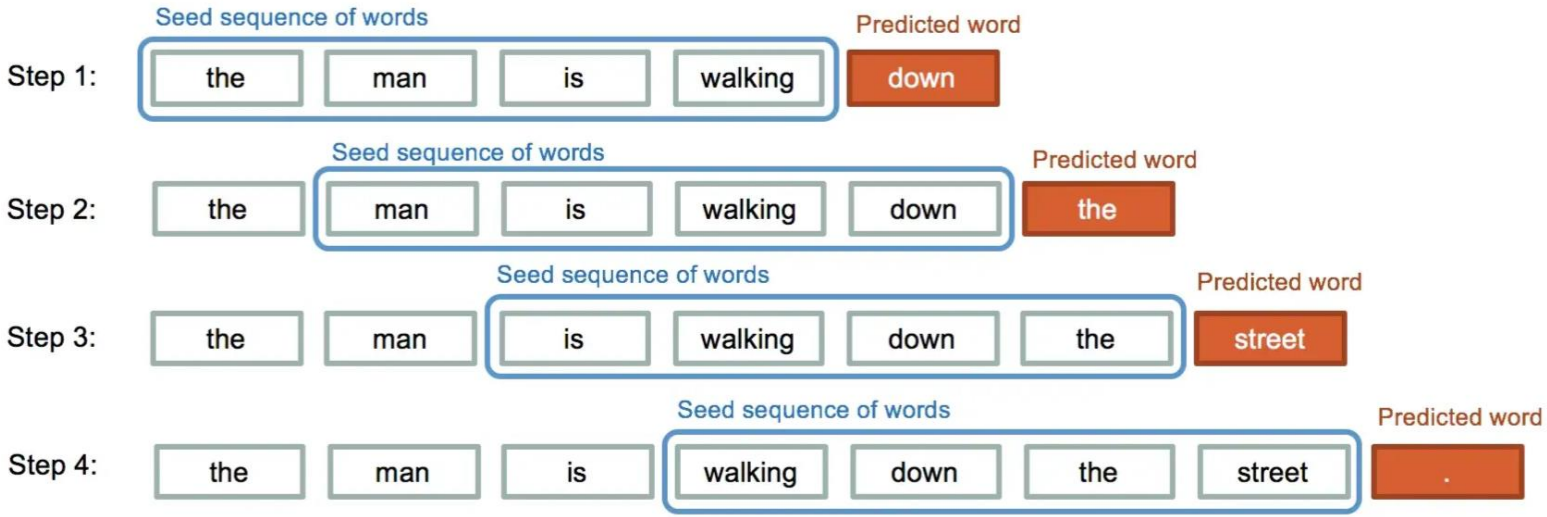
Rekurencyjne sieci neuronowe (RNN)

- Rekurencyjne sieci neuronowe pobierają wiele próbek danych z pewnego odcinka czasowego: x_{t-k}, x_{t-k+1}, ...,x_t i obliczają wynik na ich podstawie.
- Dobrze sobie radzą z danymi sekwencyjnymi i szeregami czasowymi.
- Przechowują informacje o poprzedniej próbce danych.

Zastosowania RNN

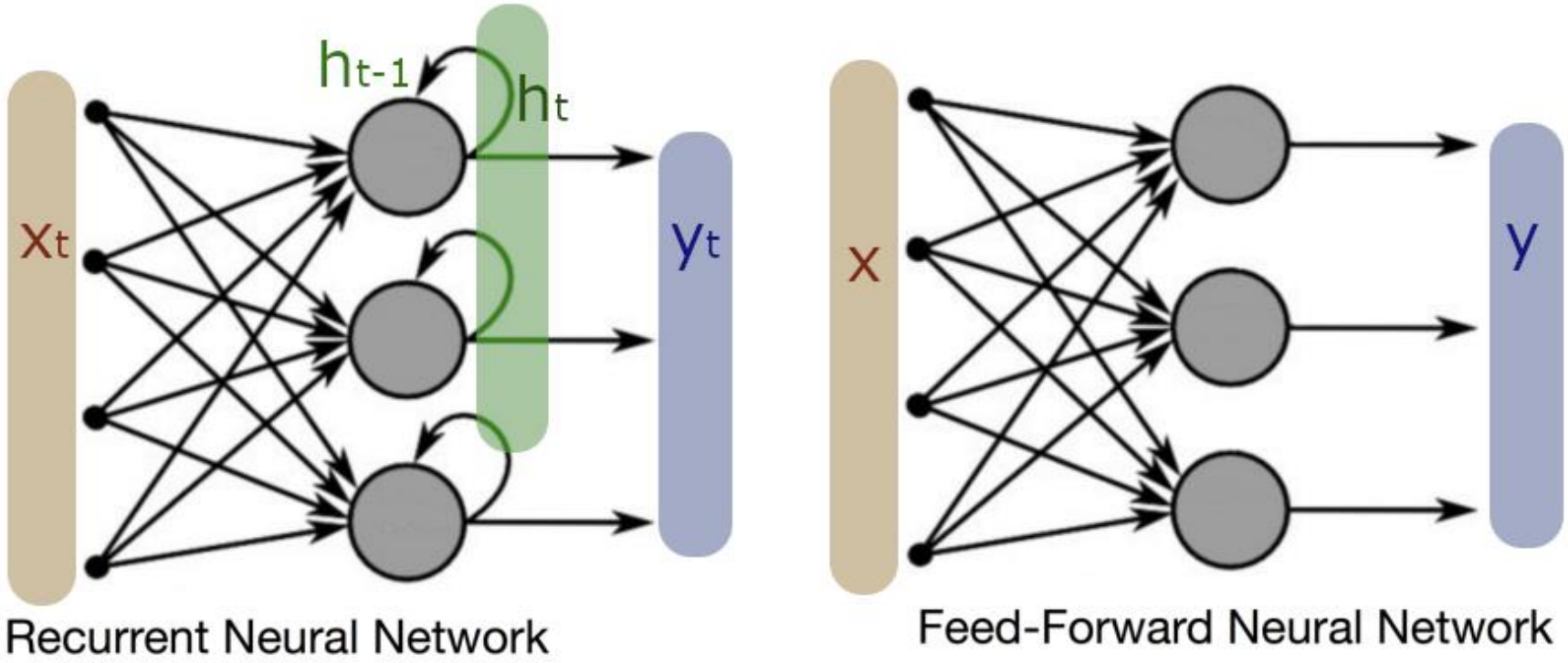
- Przetwarzanie szeregów czasowych i prognozowanie (np. przewidywanie cen towarów, akcji, itp.)
- Przetwarzanie i generowanie tekstu (np. chatboty, tłumaczenia)
- Przetwarzanie i rozpoznawanie mowy i wideo.





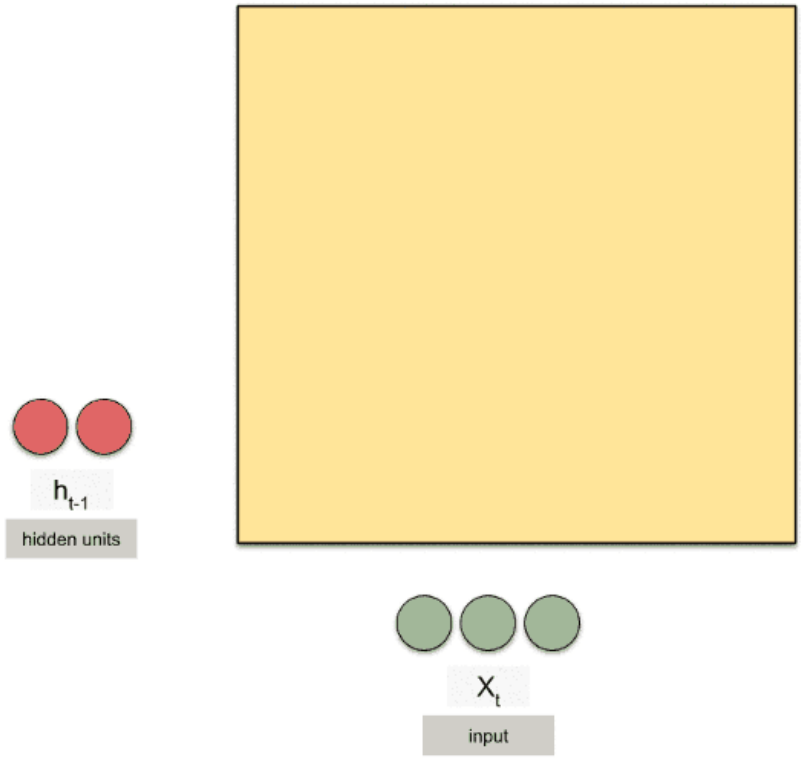
RNN: struktura

- Sieci rekurencyjna w danym kroku czasu t, dostaje aktualną próbkę danych x_t oraz informacje z poprzednich próbek danych h_{t-1}. W tym kroku czasowym sieć oblicza h_t dla następnego kroku czasowego i zwraca na wyjściu wynik y_t.



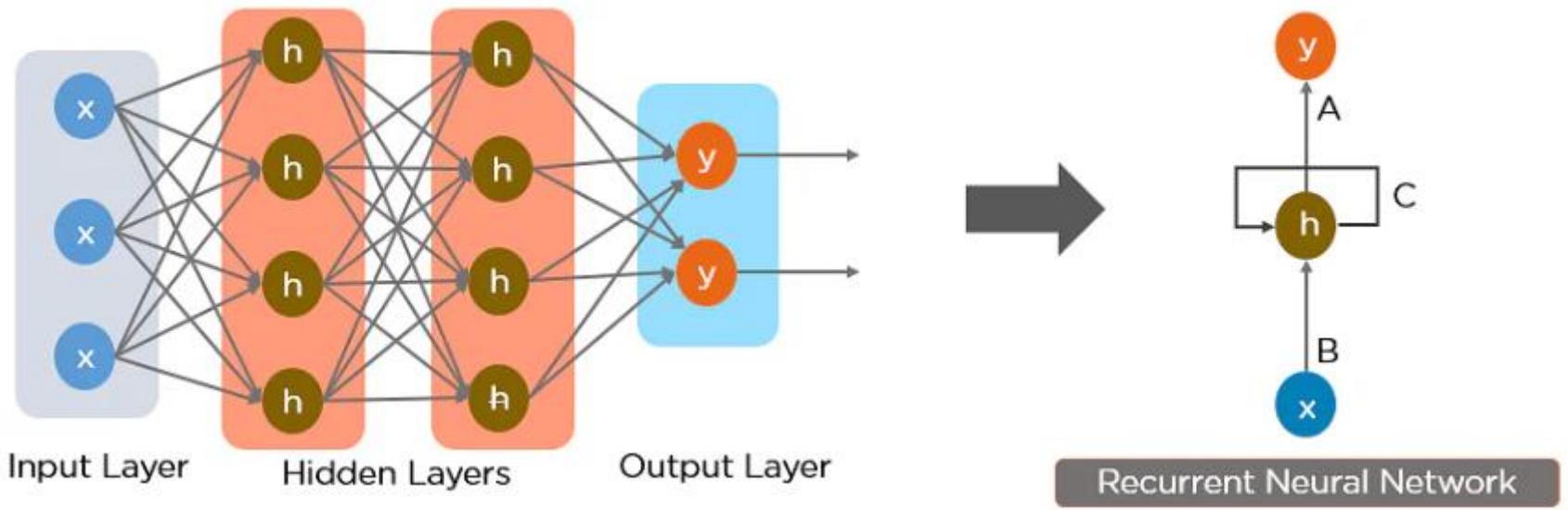
RNN: struktura

- Inne spojrzenie na wnętrze RNN. Widać 2 ukryte stany (hidden units), sklejone z wektorem wejścia o wielkości 3. Na podstawie tych 5 liczb obliczane są nowe 2 stany ukryte. Na obrazku nie uwzględniono y_t.



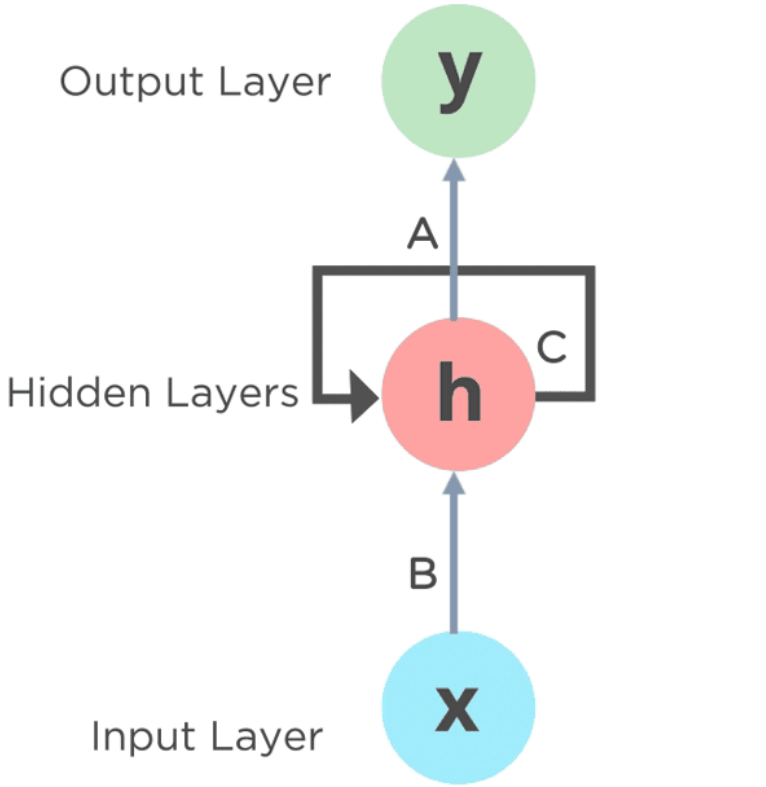
RNN: struktura

- Sieci rekurencyjne często przedstawia się w uproszczonej formie, kompresując warstwy do pojedynczych węzłów.



RNN: struktura

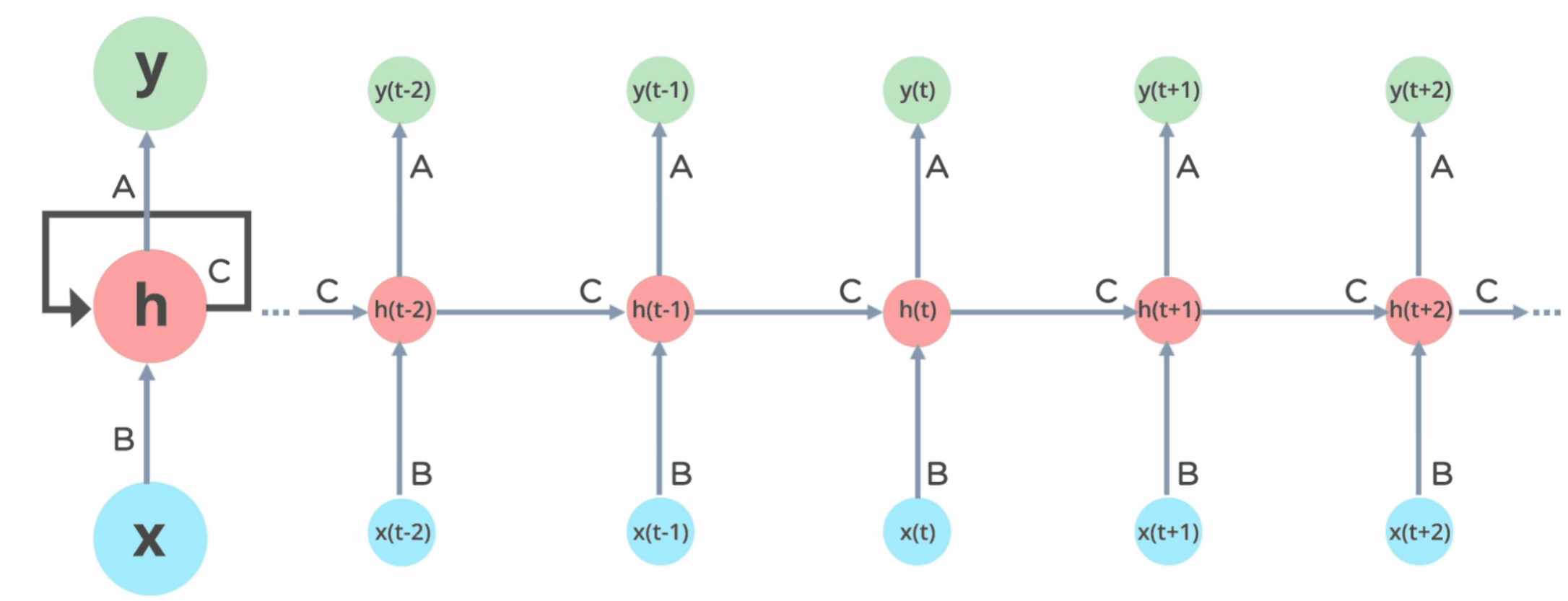
- Sieci rekurencyjne często przedstawia się w uproszczonej formie, kompresując warstwy do pojedynczych węzłów.



A, B and C are the parameters

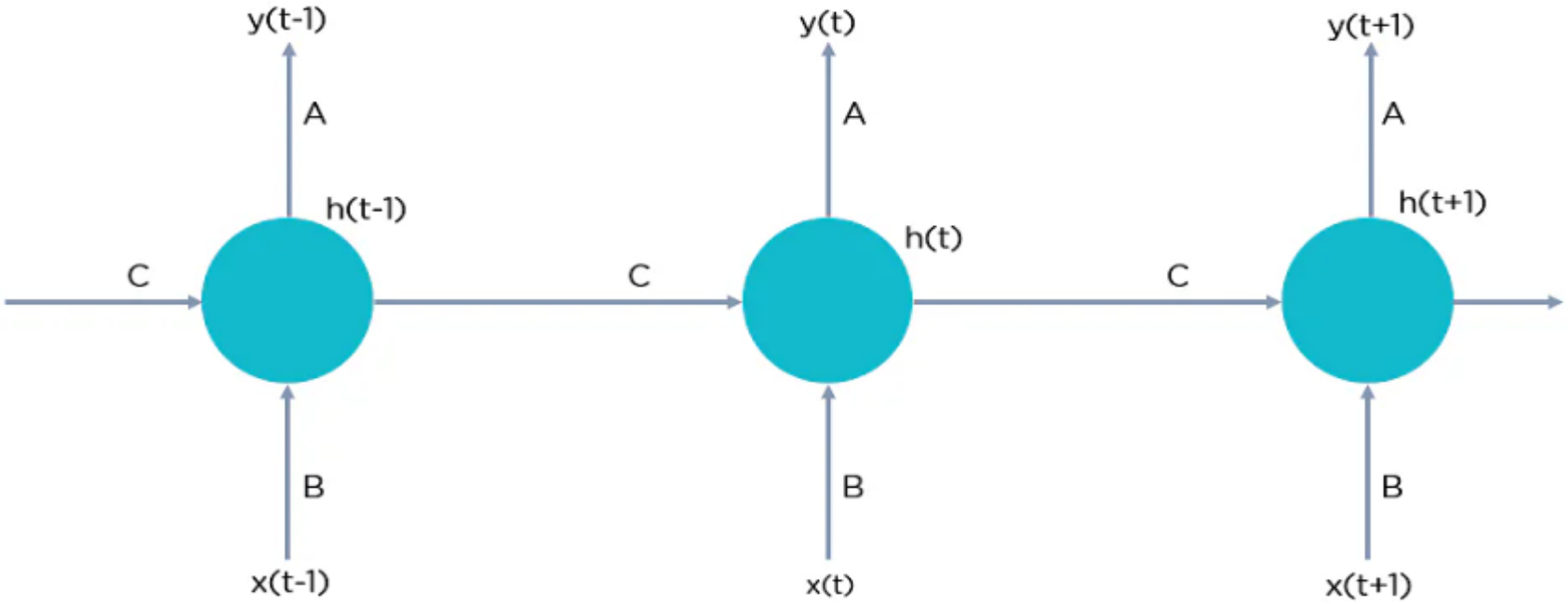
RNN: struktura

- Kolejnym krokiem, który pomaga w zrozumieniu działania sieci jest jej rozwinięcie (unrolling). Pozbywamy się zapętleń i rysujemy sieć dla kolejnych kroków czasowych.



RNN: struktura

- Kolejnym krokiem, który pomaga w zrozumieniu działania sieci jest jej rozwinięcie (unrolling). Pozbywamy się zapętleń i rysujemy sieć dla kolejnych punktów czasowych.

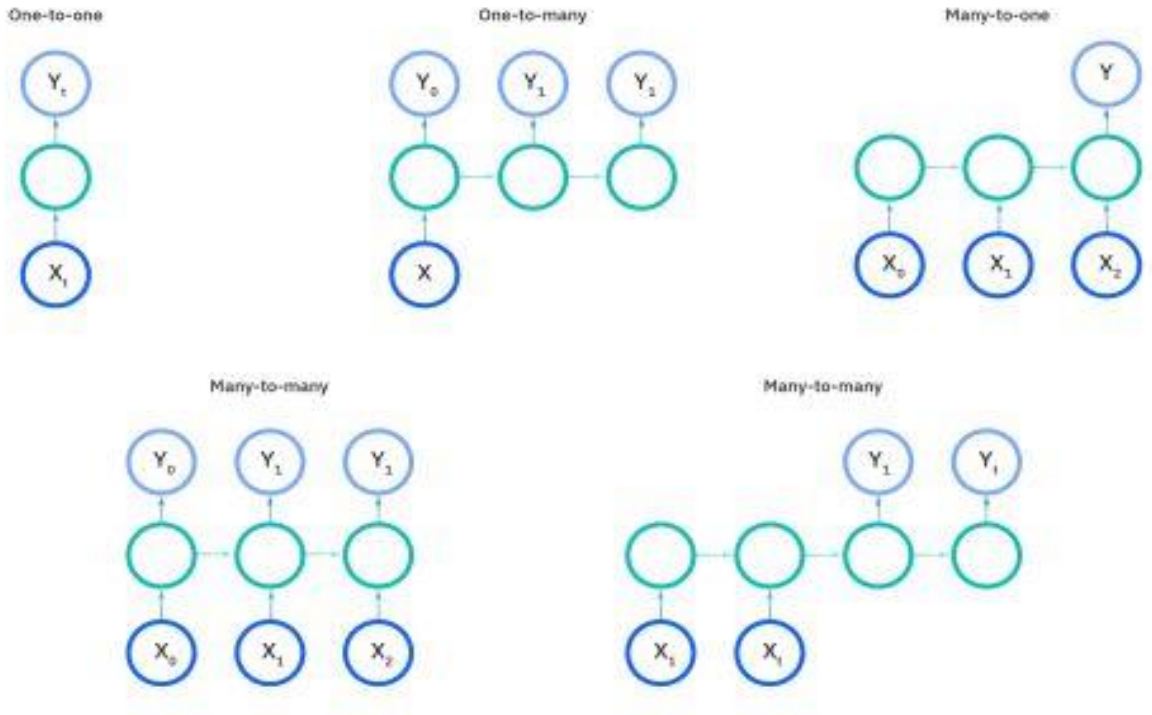


$h(t) = f_c(h(t-1), x(t))$

h(t) = new state
f_c = function with parameter c
h(t-1) = old state
x(t) = input vector at time step t

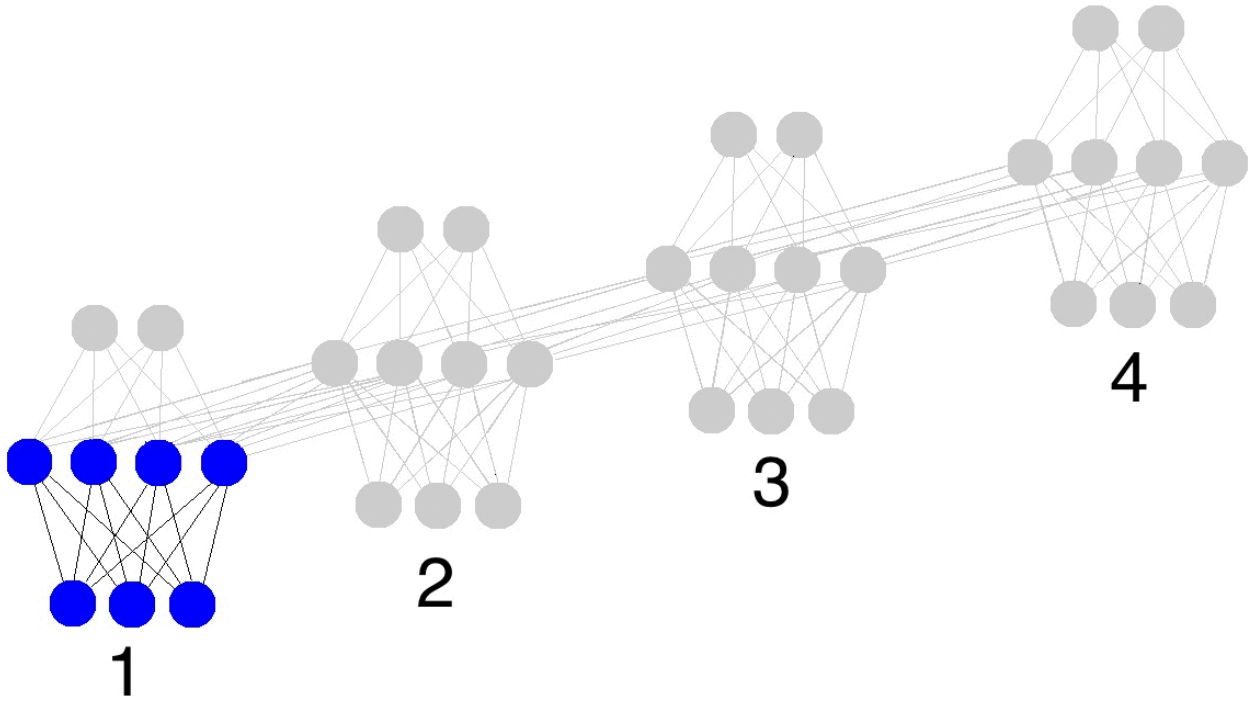
RNN: struktura

- RNN mogą korzystać z przeszłych danych z różnym stopniem zagłębienia. Mogą zwracać różne wyniki. Sieci należy dopasować do swoich potrzeb.
- Poniżej kilka typów RNN.



RNN: struktura

- RNN składa się z wielu swoich „kopii”. Liczba tych kopii mówi jak daleko zaglądamy w przeszłość (na rysunku to 4 kroki czasowe).
- Informacja z poprzednich kroków czasowych jest zapamiętywana i przekazywana do następnych kroków.



RNN: implementacja w keras

- W paczce pythonowej keras, jest możliwość wykorzystania warstwy SimpleRNN https://keras.io/api/layers/recurrent_layers/simple_rnn/
- Ważnym parametrem konfiguracyjnym jest units. Oznacza ono, ile ukrytych stanów należy przekazywać do sieci w następnym kroku czasowym (wielkość wektora h_t).
- Parametry wejściowe dla sieci mają zaś postać: wielkość_batcha x liczba_kroków_czasowych („głębokość sięgania w przeszłość”) x wielkość_inputu_x

```
inputs = np.random.random([32, 10, 8]).astype(np.float32)
simple_rnn = tf.keras.layers.SimpleRNN(4)

output = simple_rnn(inputs)  # The output has shape `[32, 4]`.

simple_rnn = tf.keras.layers.SimpleRNN(
    4, return_sequences=True, return_state=True)

# whole_sequence_output has shape `[32, 10, 4]`.
# final_state has shape `[32, 4]`.
whole_sequence_output, final_state = simple_rnn(inputs)
```

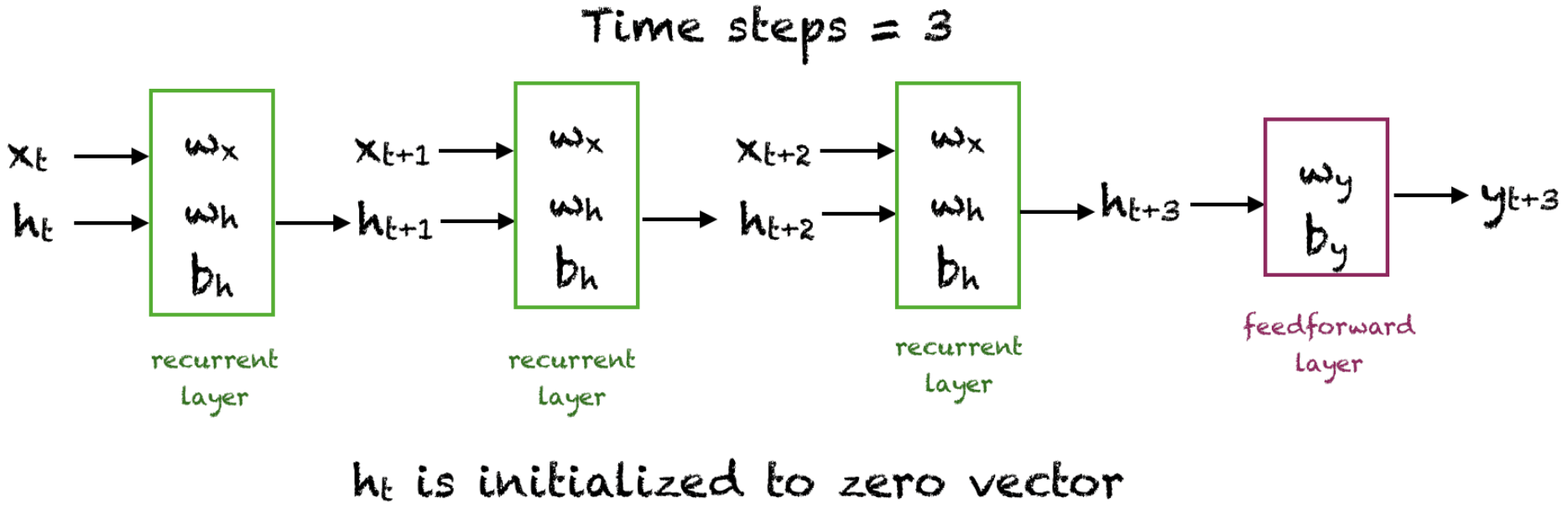
Przykład 1: Z dokumentacji SimpleRNN

- Uruchomimy i przeanalizujemy teraz załączony plik rnn01.py.

```
inputs = np.random.random([32, 10, 8]).astype(np.float32)
simple_rnn = tf.keras.layers.SimpleRNN(4)

output = simple_rnn(inputs)  # The output has shape `[32, 4]`.

simple_rnn = tf.keras.layers.SimpleRNN(
    4, return_sequences=True, return_state=True)

# whole_sequence_output has shape `[32, 10, 4]`.
# final_state has shape `[32, 4]`.
whole_sequence_output, final_state = simple_rnn(inputs)
```

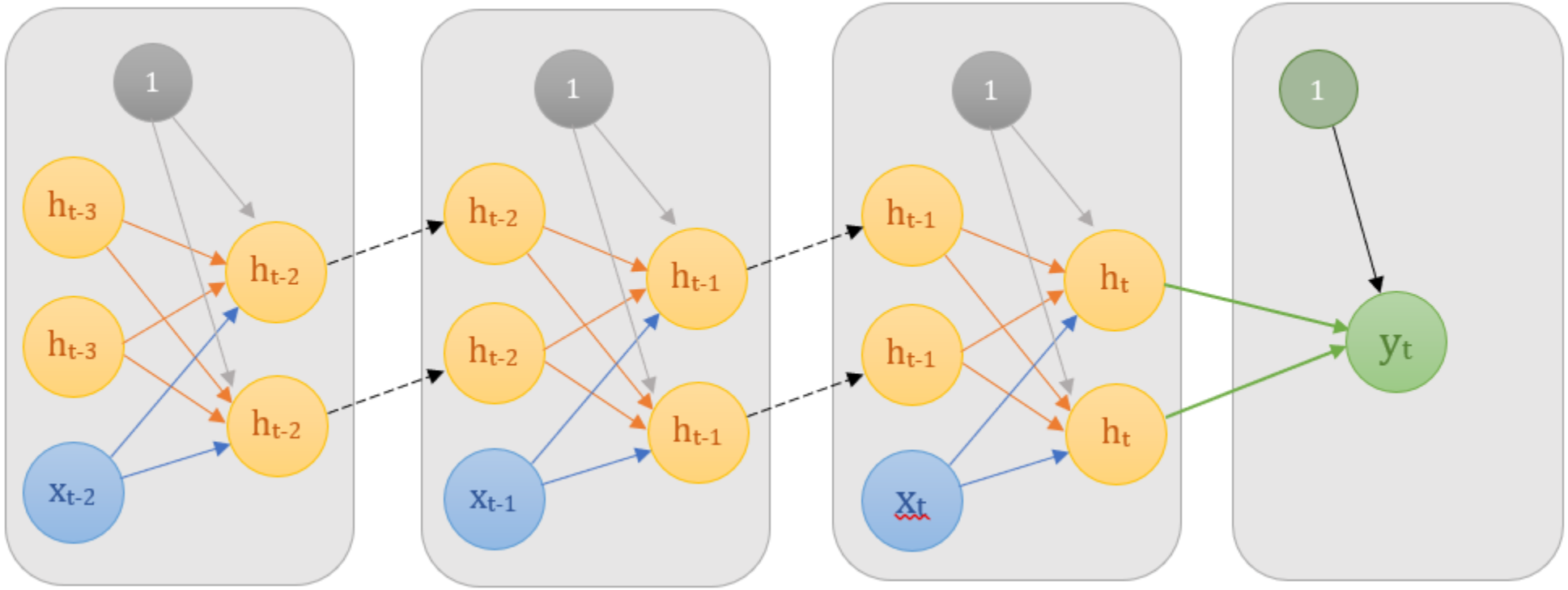
Przykład 2: Testy dla prostego modelu RNN

- Uruchomimy i przeanalizujemy teraz załączony plik rnn02.py (stworzony na podstawie „Keras SimpleRNN” z samouczka https://machinelearningmastery.com/understanding-simple-recurrent-neural-networks-in-keras/).
- Nasza sieć rekurencyjna będzie miała postać:



Przykład 2: Testy dla prostego modelu RNN

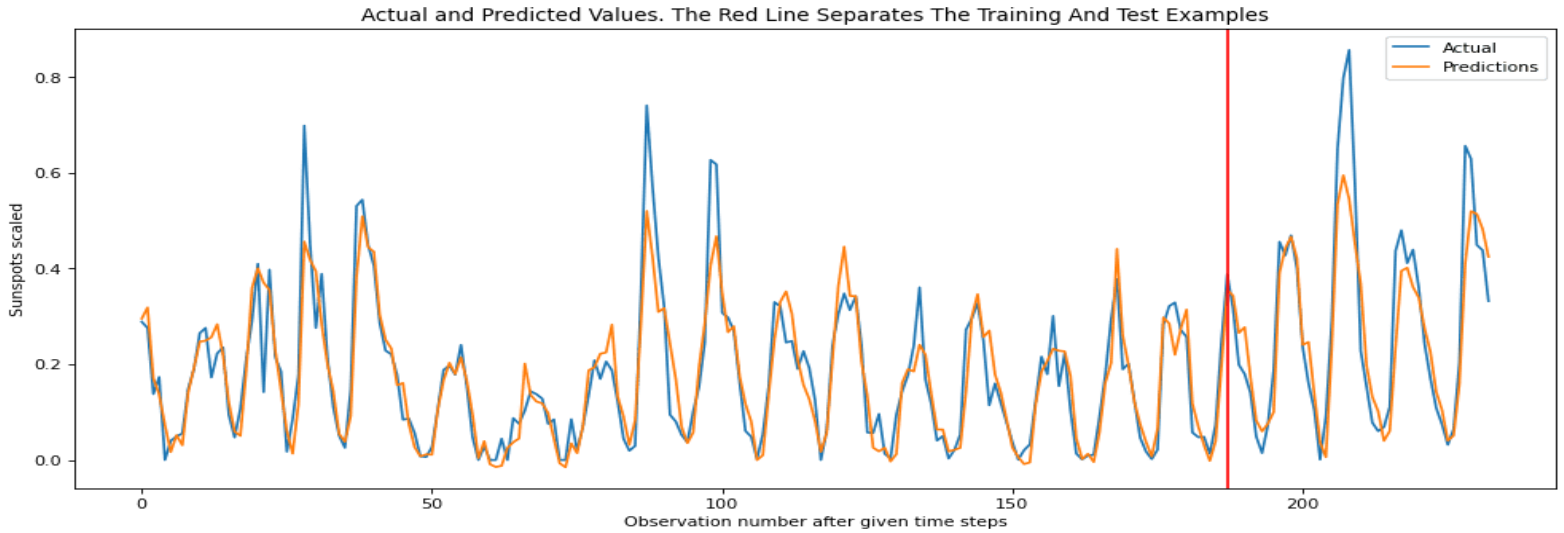
- Nieco dokładniejsza rozpiska:



- Warstwa rekurencyjna potrzebuje 4 wag pomiędzy stanami ukrytymi, 2 wag ze stanu wejściowego do ukrytych, 2 wag z bias do stanów ukrytych.
- Warstwa w pełni połączona potrzebuje 2 wag ze stanów ukrytych do wyjściowego oraz 1 wagi z bias do wyjściowego.

Przykład 3: Liczba plam słonecznych (co miesiąc od 1749 roku)

- Uruchomimy i przeanalizujemy teraz załączony plik rnn03.py (stworzony na podstawie https://machinelearningmastery.com/understanding-simple-recurrent-neural-networks-in-keras/ i https://www.kaggle.com/datasets/robervalt/sunspots).



Część I: Rekurencyjne sieci neuronowe

Część II: Sieci LSTM

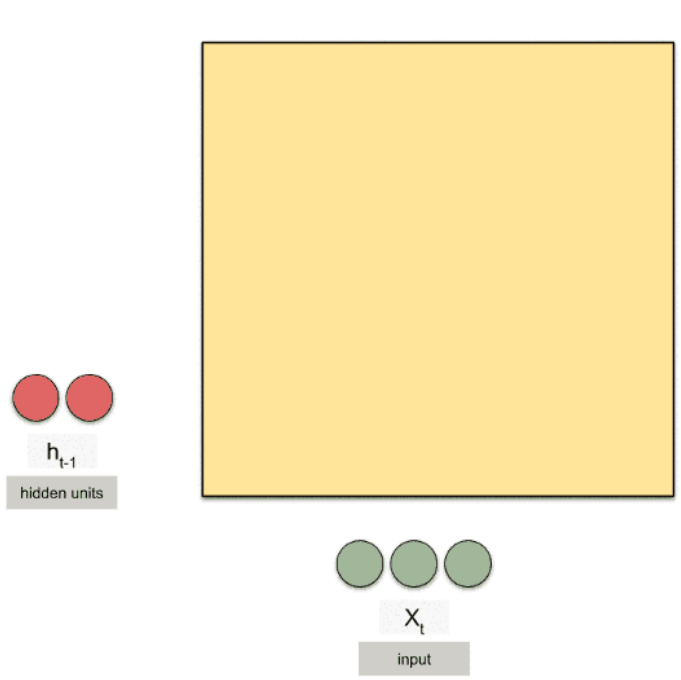
Część III: Generowanie tekstu poprzez LSTM

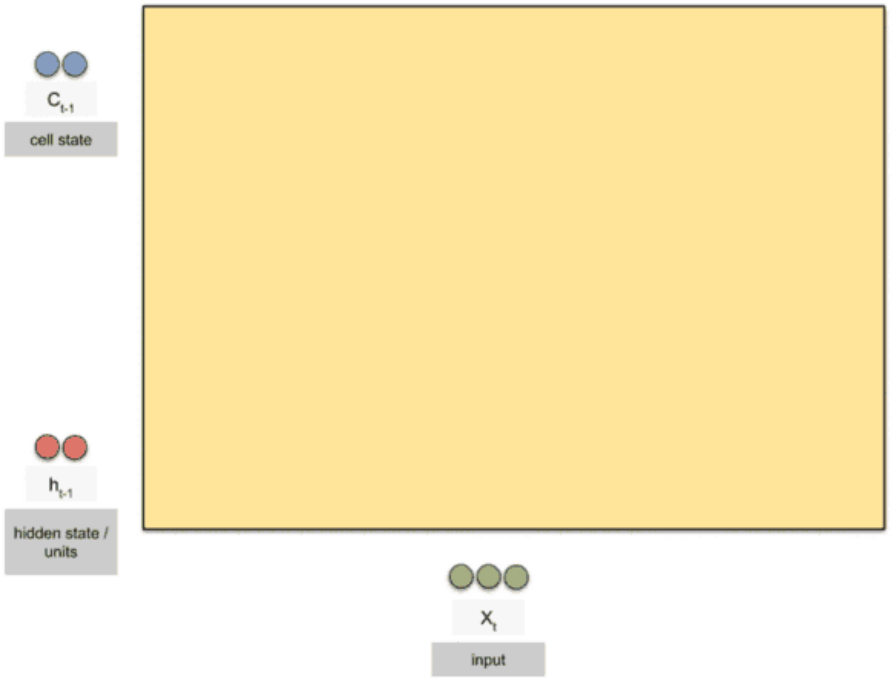
Long Short-Term Memory Network (LSTM)

- Szczególnym przypadkiem RNN są sieci LSTM, które posiadają długą pamięć krótkoterminową. Zaprojektowano je w roku 1997 (Hochreiter i Schmidhuber).
- RNN mają pamięć krótkoterminową. Nie mogą zapamiętywać zbyt wiele, bo mają problem z przechowaniem informacji (problem zanikającego gradientu, problem eksplodującego gradientu).
- Architektura LSTM pozwala na zapamiętywanie prostych informacji przez długi okres czasu. Stąd nazwa.
- Przykład zdania: „Azor jest psem rasy owczarek niemiecki. Ma brązową sierść, czarny nos i puszysty ogon. Był trenowany przez Jana Kowalskiego na psa ratownika w Tatrzańskim Parku Narodowym. Ma 8 lat”.
- Kto ma 8 lat? Park? Jan Kowalski? Nos? Ogon? Pies?

Long Short-Term Memory Network (LSTM)

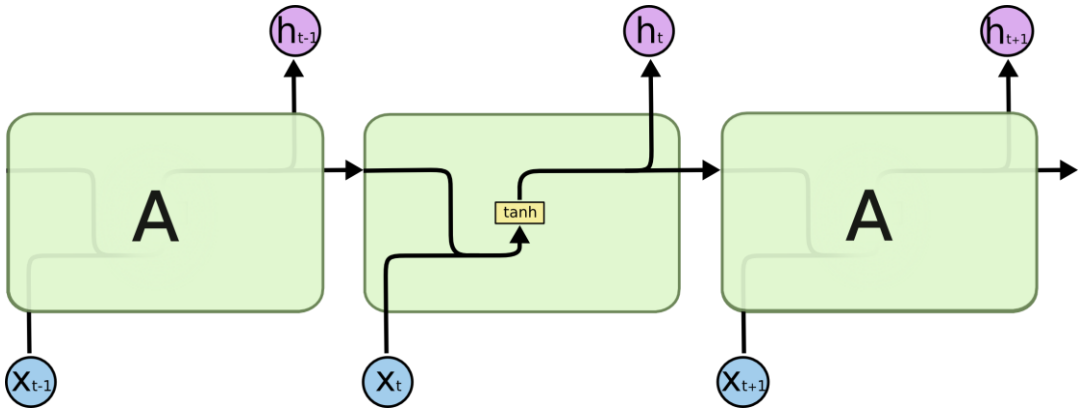
- RNN i LSTM mają podobną architekturę łańcuchową, z powtarzającym się modułem dla każdego kroku czasowego.
- LSTM mają bardziej skomplikowaną architekturę każdego modułu (RNN – po lewej, LSTM – po prawej).

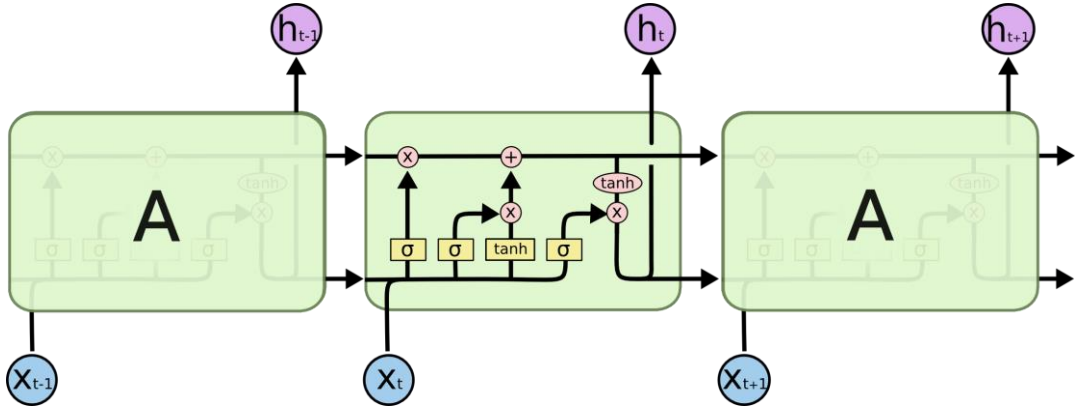




Long Short-Term Memory Network (LSTM)

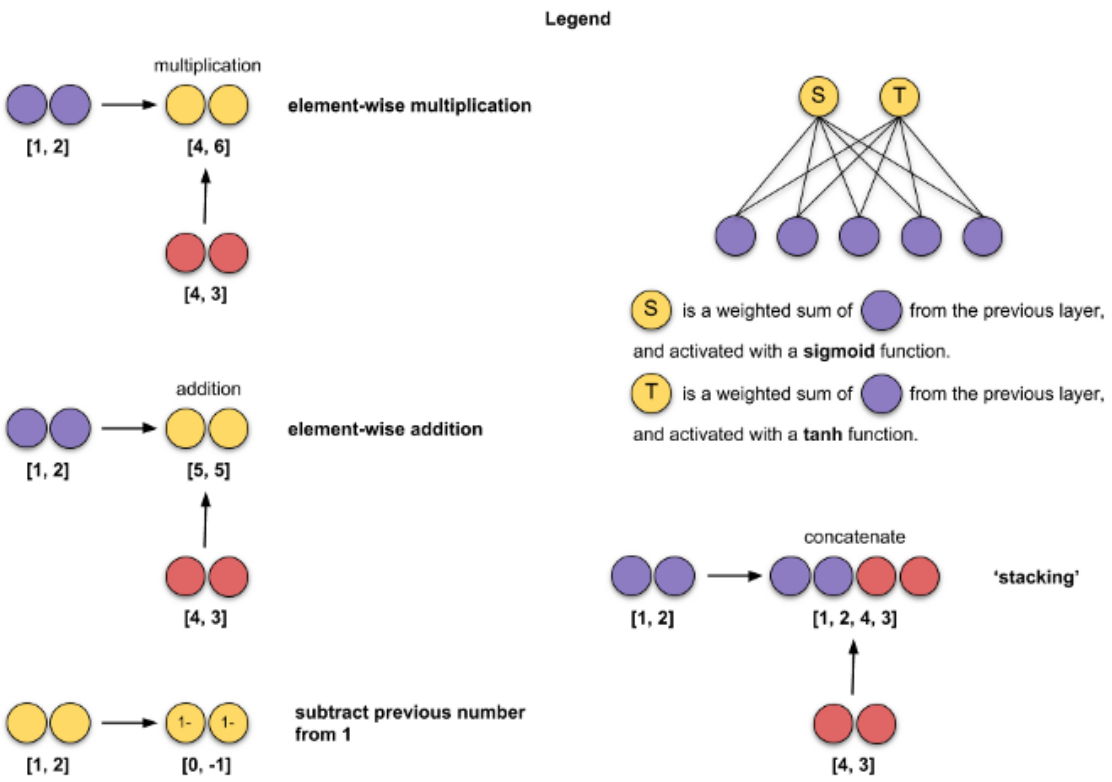
- RNN i LSTM mają podobną architekturę łańcuchową, z powtarzającym się modułem dla każdego kroku czasowego.
- LSTM mają bardziej skomplikowaną architekturę każdego modułu (RNN – po lewej, LSTM – po prawej).

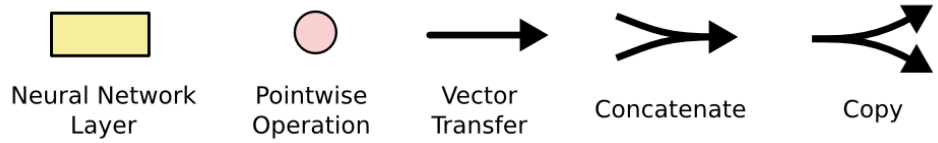


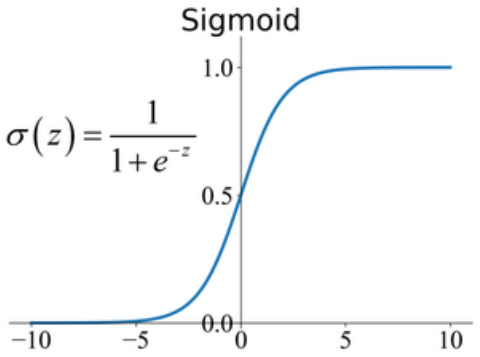


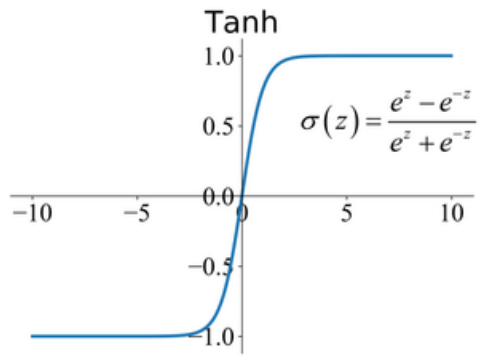
Struktura LSTM: operacje

- W architekturze LSTM pojawia się operacja mnożenia wektorów po współrzędnych, dodawania po współrzędnych, aktywacji warstwy funkcją tangens hiperboliczny (T, tanh) lub sigmoid (S, σ).



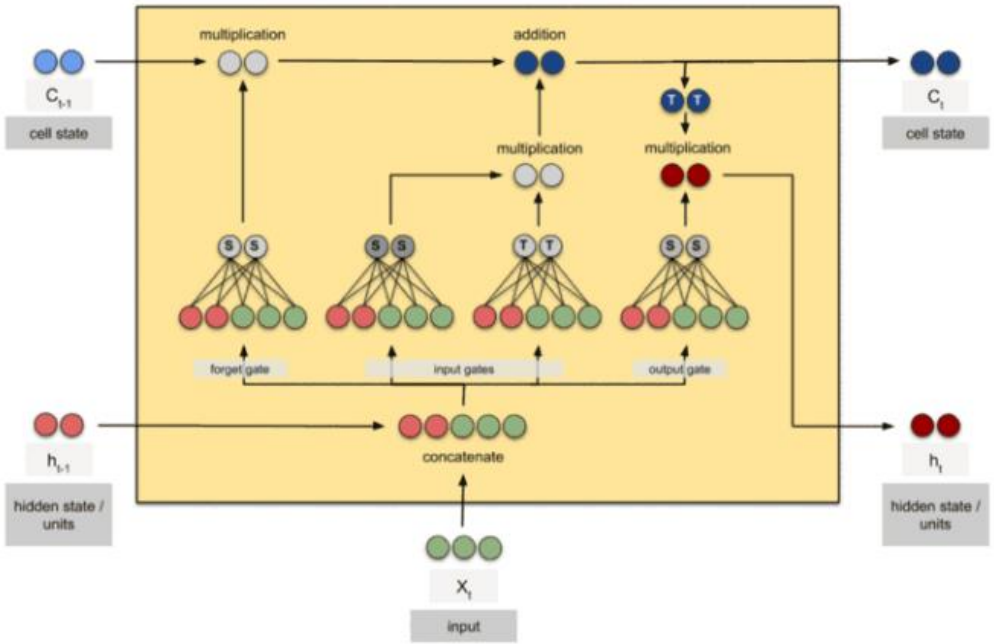


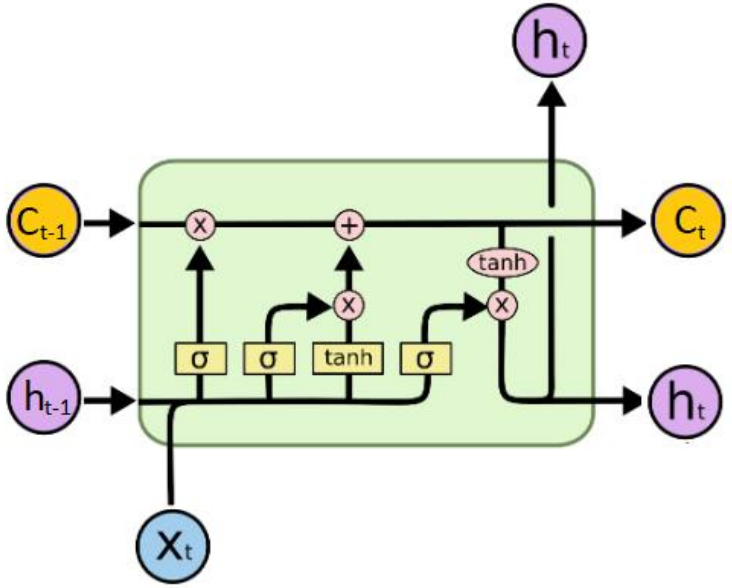




Struktura LSTM : architektura

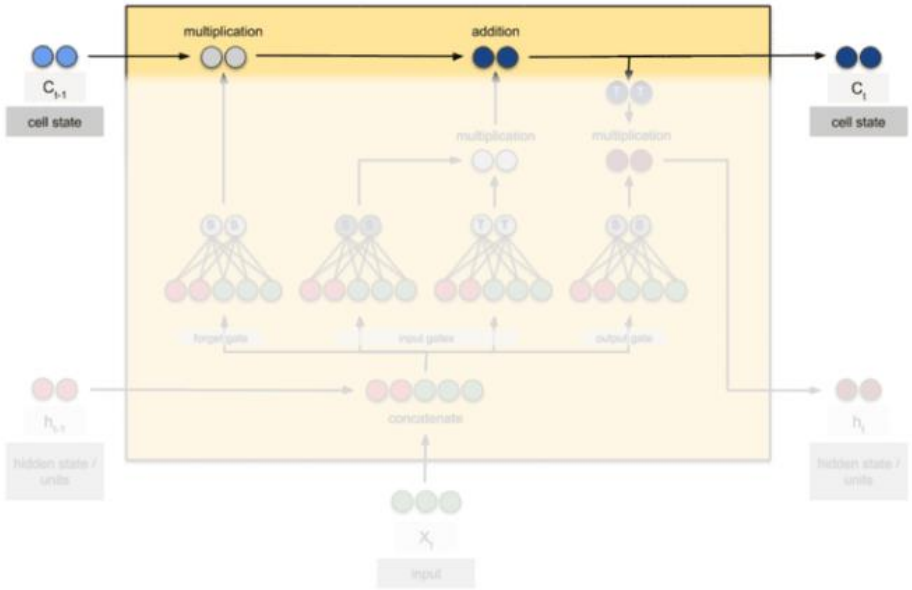
- Oba rysunki przedstawiają moduł/komórkę LSTM, ale z dwóch różnych samouczków.
- Zwróćmy uwagę na operacje mnożenia po współrzędnych, dodawania po współrzędnych, aktywacji warstwy funkcją tangens hiperboliczny (T, tanh) lub sigmoid (S, σ).

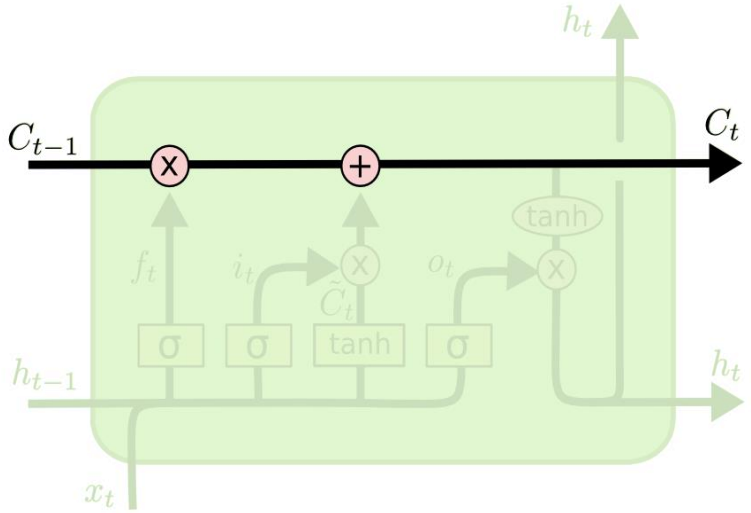




Struktura LSTM: cell state

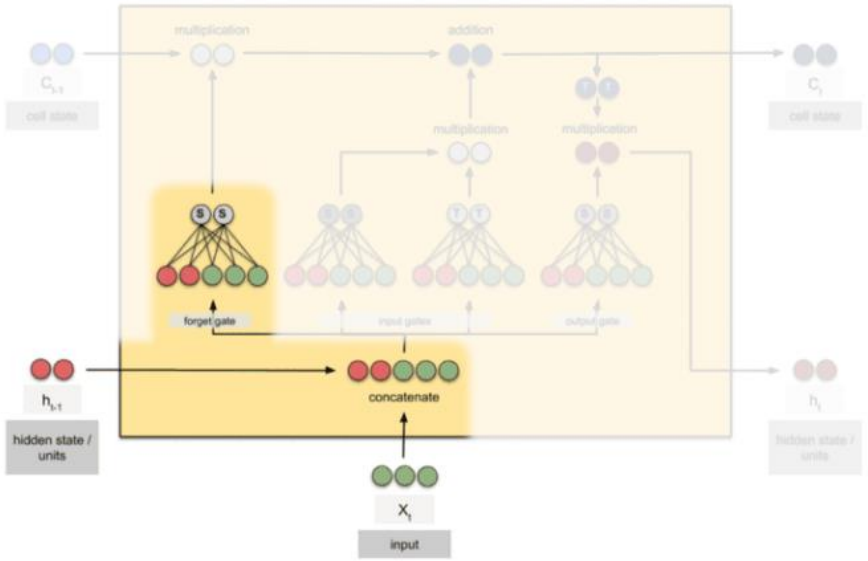
- Pierwszy komponent: stan komórki (ang. cell state) to wektor liczb C, o tej samej wielkości, co ukryte stany h. Przechowywane są w nim przefiltrowane informacje długoterminowe (przekazywane między modułami).
- Na rysunkach poniżej widać magistralę, która dostaje C_{t-1}, pobiera nieco informacji za pomocą tzw. bramek i oblicza nowy stan C_t.

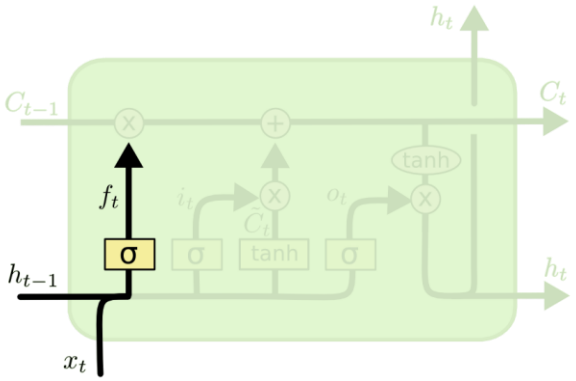




Struktura LSTM: forget gate

- Ważnym komponentem LSTM jest bramka zapominania (ang. forget gate). To mała wewnętrzna sieć neuronowa, której wejściem jest h_{t-1} i x_t. Sieć ta wyucza się, które informacje są istotne, a które nie.
- Aktywowana jest funkcją sigmoidalną (wartości (0,1)). Dane mniej ważne dostają aktywację bliską 0, a dane ważne, aktywacje bliską 1.
- Istotność informacji (od 0 do 1) jest wmnożonona do stanu komórki C_t.

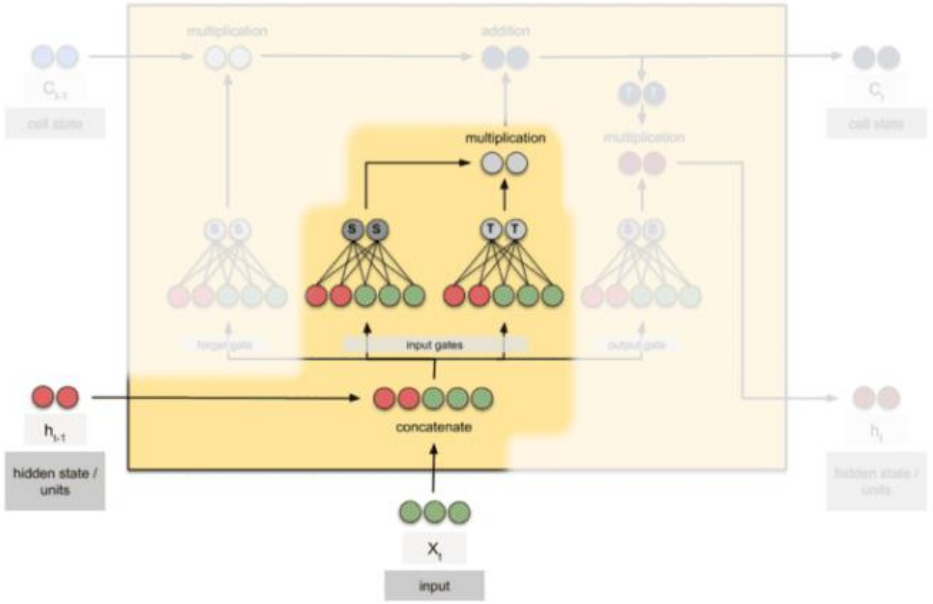


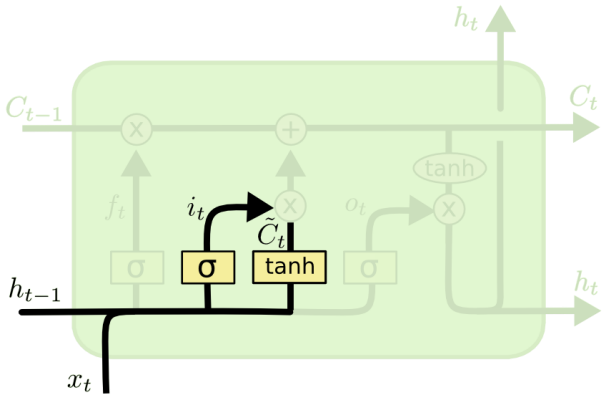


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Struktura LSTM: input gate

- Kolejny komponent to bramka wejściowa (ang. input gate). Bierze ona wektor $[h_{t-1}\, x_t]$ i wyucza się dwóch rzeczy.
  - Które informacje z wektora zapamiętać? Robi to podsieć z aktywacją sigmoidalną.
  - Jak te informacje zaktualizować? Robi to podsieć z aktywacją tanh.
- Wynik dwóch podsieci jest mnożony (istotność (0,1) razy wartość (-1,1)). Następnie dodana jest do stanu komórki C_t.



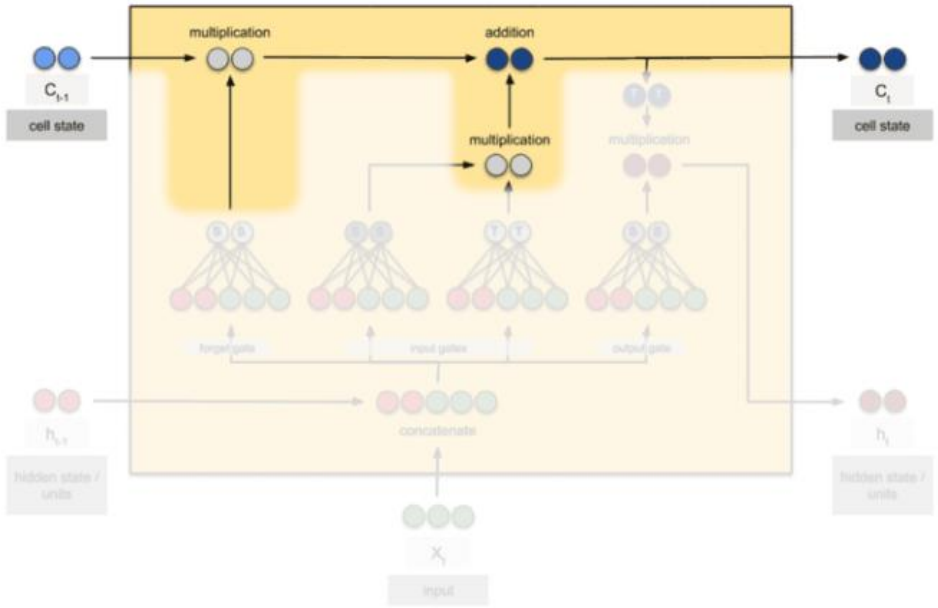


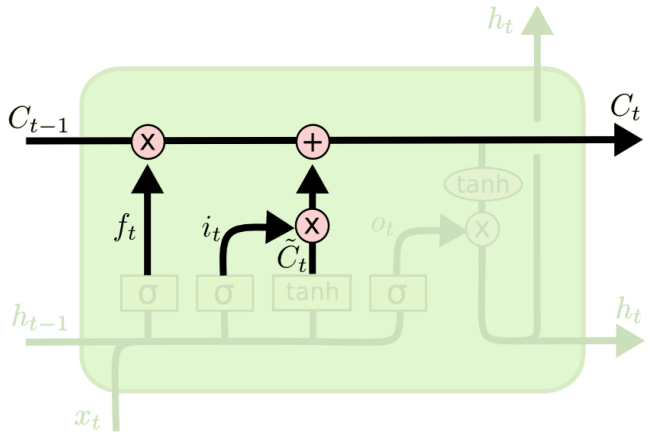
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Struktura LSTM: update cell state

- Podsumowując, bramka zapominania usunie ze stanu komórki nieistotne informacje (np. stare nieistotne zaimki, słowa nie wpływające na rozumienie tekstu).
- Następnie bramka wejściowa doda do stanu nowe informacje, uwzględniając też stopień ich istotności.

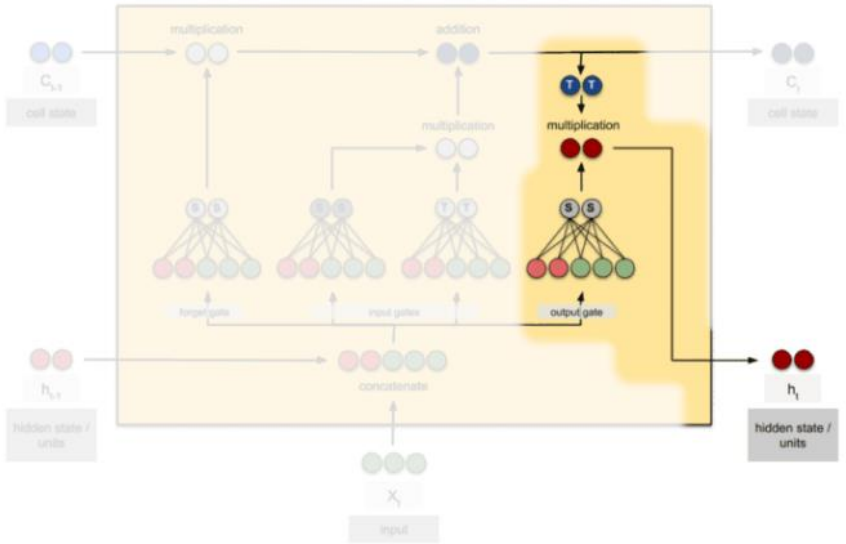


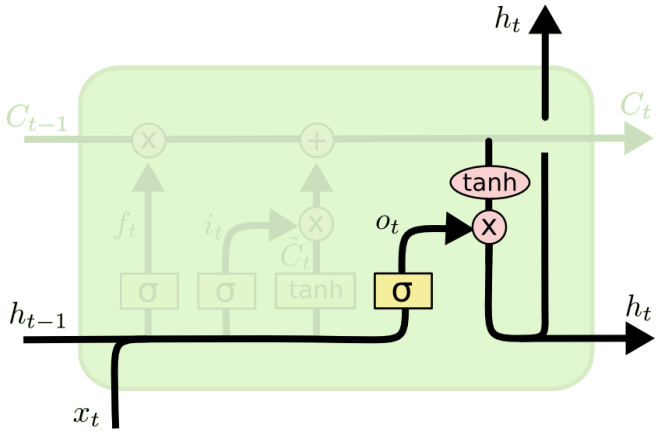


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Struktura LSTM: output gate

- Wreszcie, trzeba obliczyć output modułu, czyli wektor h_t. Bierzmy dane ze stanu komórki C_t i ściskamy je funkcją tanh do przedziału (-1,1).
- Na tym jednak nie kończymy. Ostatnią bramką jest bramka wyjścia (ang. output gate), która znowu sigmoidalnie decyduje, które dane z są istotne (wartość 1), a które nie (wartość 0).
- Po mnożeniu jej wyjścia i $\tanh(C_t)$ otrzymujemy wynik.



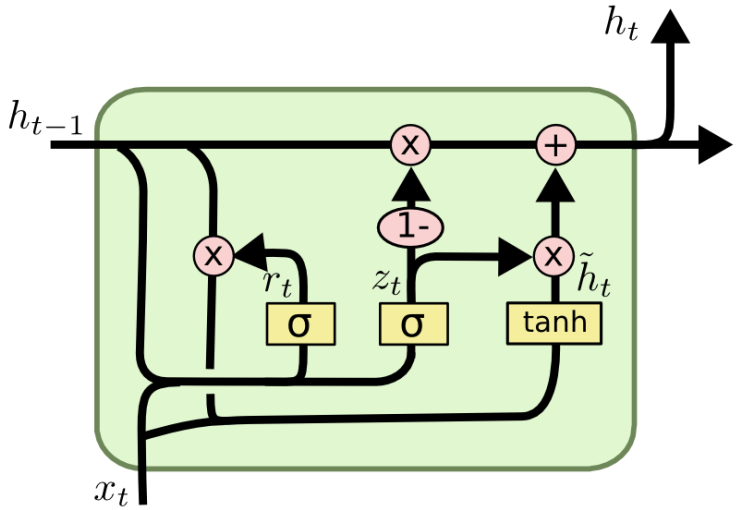


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Inne warianty LSTM

- Przez lata opracowano wiele różnych wariantów LSTM, które mają nieco inną architekturę modułów/komórek.
- Jednym z ciekawszych jest Gated Recurrent Unit (GRU) opracowany przez Cho et al., 2014. Model ten jest również bardzo skuteczny.
- W modelu tym stany h_t i C_t są sklejone w jeden. Bramki są inne niż w zwykłym LSTM.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

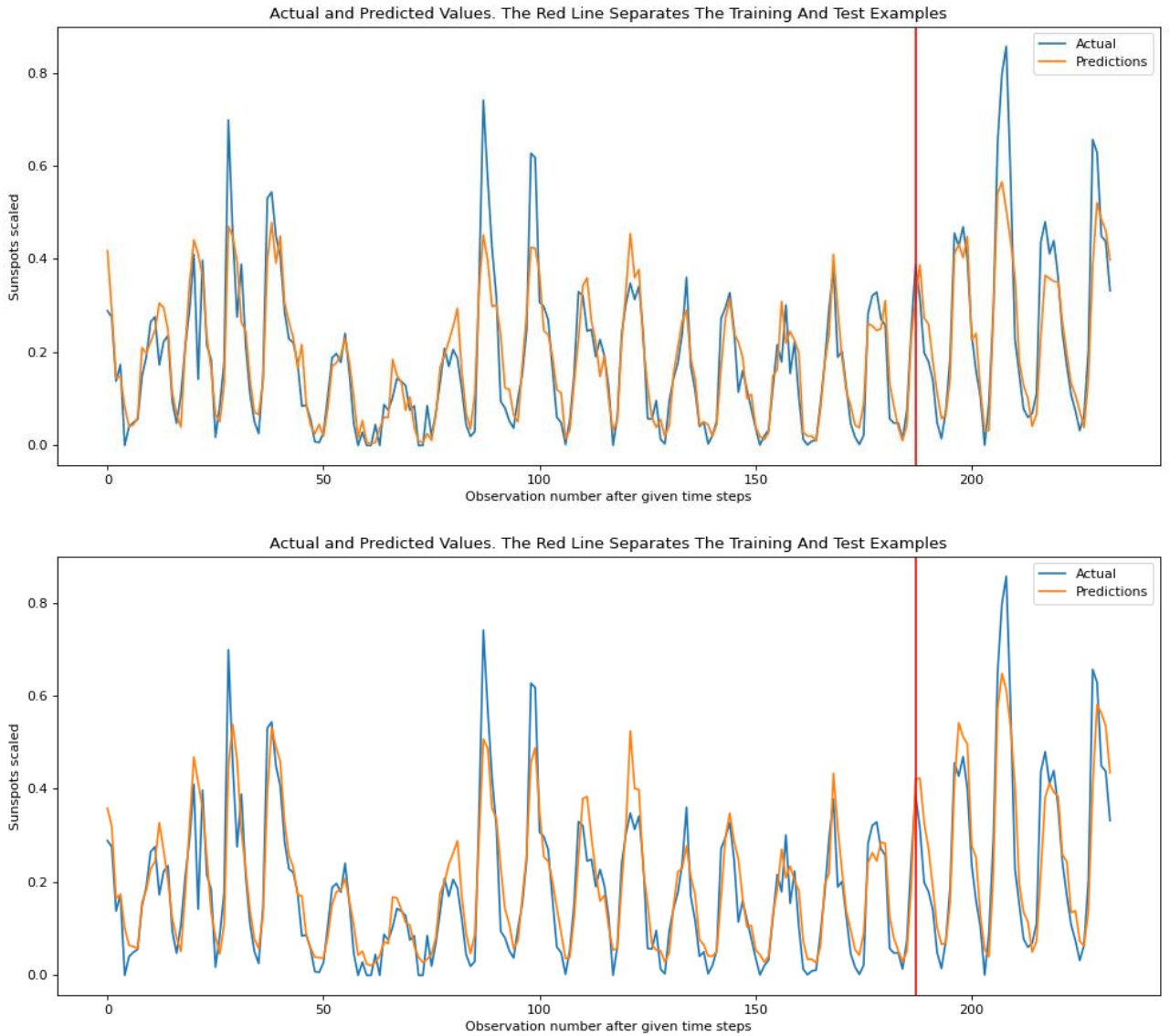
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM w praktyce

- Zmodyfikujemy zadanie rnn03.py, zastępując warstwę SimpleRNN warstwą LSTM z 5 krokami czasowymi. Przyjrzyj się zadaniu lstm01.py.
- Górny wynik: rnn03.py. Dolny wynik: lstm01.py.



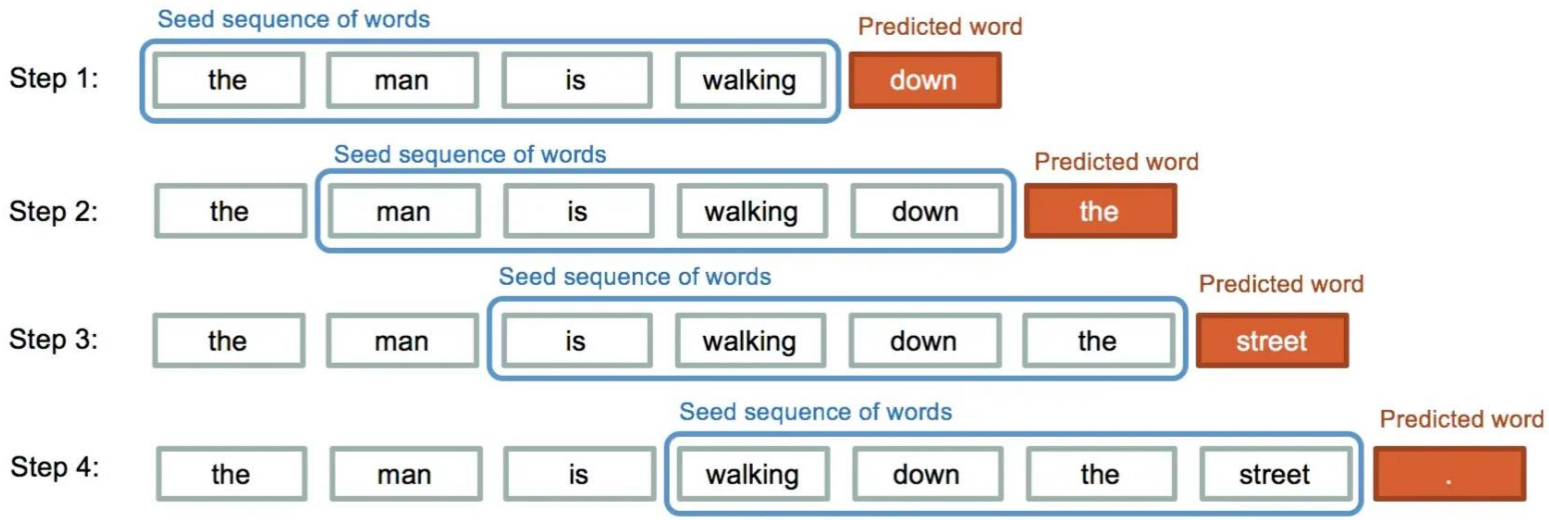
Część I: Rekurencyjne sieci neuronowe

Część II: Sieci LSTM

Część III: Generowanie tekstu poprzez LSTM

Generowanie tekstu: tekst na liczby

- Przypomnijmy obrazek z poprzednich slajdów. Sieci rekurencyjne mogą zgadywać jakie będzie następne słowo w tekście, biorąc pod uwagę słowa poprzednie:



- Sieci jednak działają na liczbach, nie tekście. Jak zakodować tekst jako liczby?

Generowanie tekstu: tekst na liczby

- Najprostszą techniką jest zamiana każdego słowa na liczbę. Każdemu słowu będzie odpowiadał jakiś klucz/indeks. Technikę tę nazwamy index-based encoding dla słów.
- Przetwarzamy tekst technikami modelu bag-of words. Zliczamy unikalne słowa. Nadajemy każdemu słowu indeks. Zamienimy tekst na zestaw indeksów.
- W jednym kroku czasowym RNN pobiera jedną liczbę (indeks aktualnego słowa). Dla wielu kroków będzie to oczywiście zestaw indeksów.

Korpus danych: [“a” , “bad” , “cat” , “good” , “has” , “he” , “is” , “mobile” , “not” , “phone” , “she” , “temper” , “this”]

Indeksy: [“a”:1, “bad” :2 , “cat” :3 , “good” :4 , “has”:5 , “he”:6 , “is”:7 , “mobile”:8 , “not”:9 , “phone”:10 , “she”:11 , “temper”:12 , “this”:13]

Przykład mapowania danych: [“this is a good phone” , “this is a bad mobile” , “she is a good cat” , “he has a bad temper” , “this mobile phone is not good”] -> [13 7 1 4 10] , [13 7 1 2 8] , [11 7 1 4 3] , [6 5 1 2 12] , [13 8 10 7 9 4]

Generowanie tekstu: tekst na liczby

- Minusem index-based encoding dla słów jest fakt, że liczb może być dużo, bo dużo jest słów w słowniku.
- Ponadto słowo nr 13564 i słowo nr 13565, mimo że są blisko wg numerów, mogą mieć zupełnie inne znaczenie. Tymczasem sieć z uwagi na podobieństwo numeryczne, może traktować je jako podobne.

Generowanie tekstu: tekst na liczby

- Prozwiązaniem poprzedniego problemu może być technika index-based encoding dla znaków.
- Każdemu znakowi przyporządkowujemy liczbę naturalną (np. numer z tabelki ASCII lub UTF-8). Tych liczb będzie zdecydowanie mniej!
- Uwaga! W tym przypadku sieć zgaduje tekst znak po znaku (na podstawie poprzednich znaków), a nie słowo po słowie (na podstawie poprzednich słów). Generowanie tekstu jest bardziej niskopoziomowe.
- „RNN” -> [18, 14, 14]

Number Substitution Cypher

A	B	C	D	E	F	G	H	I	J
1	2	3	4	5	6	7	8	9	10

K	L	M	N	O	P	Q	R	S	T
11	12	13	14	15	16	17	18	19	20

U	V	W	X	Y	Z
21	22	23	24	25	26

Generowanie tekstu: tekst na liczby

- Kolejna technika to one-hot encoding dla słów.
- Wybieramy zestaw słów z tekstu (poprzez techniki bag-of-words). Następnie każdemu słowu przyporządkowujemy wektor binarny, z jedynką wskazującą na to słowo. Moduł RNN otrzymuje taki wektor na wejście.
- To podejście ma wady. Kodowania mają jedną jedynkę i dużo niepotrzebnych zer. Zajmują mnóstwo pamięci, gdy wybieramy dziesiątki tysięcy słów.

Human-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Machine-Readable

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

Generowanie tekstu: tekst na liczby

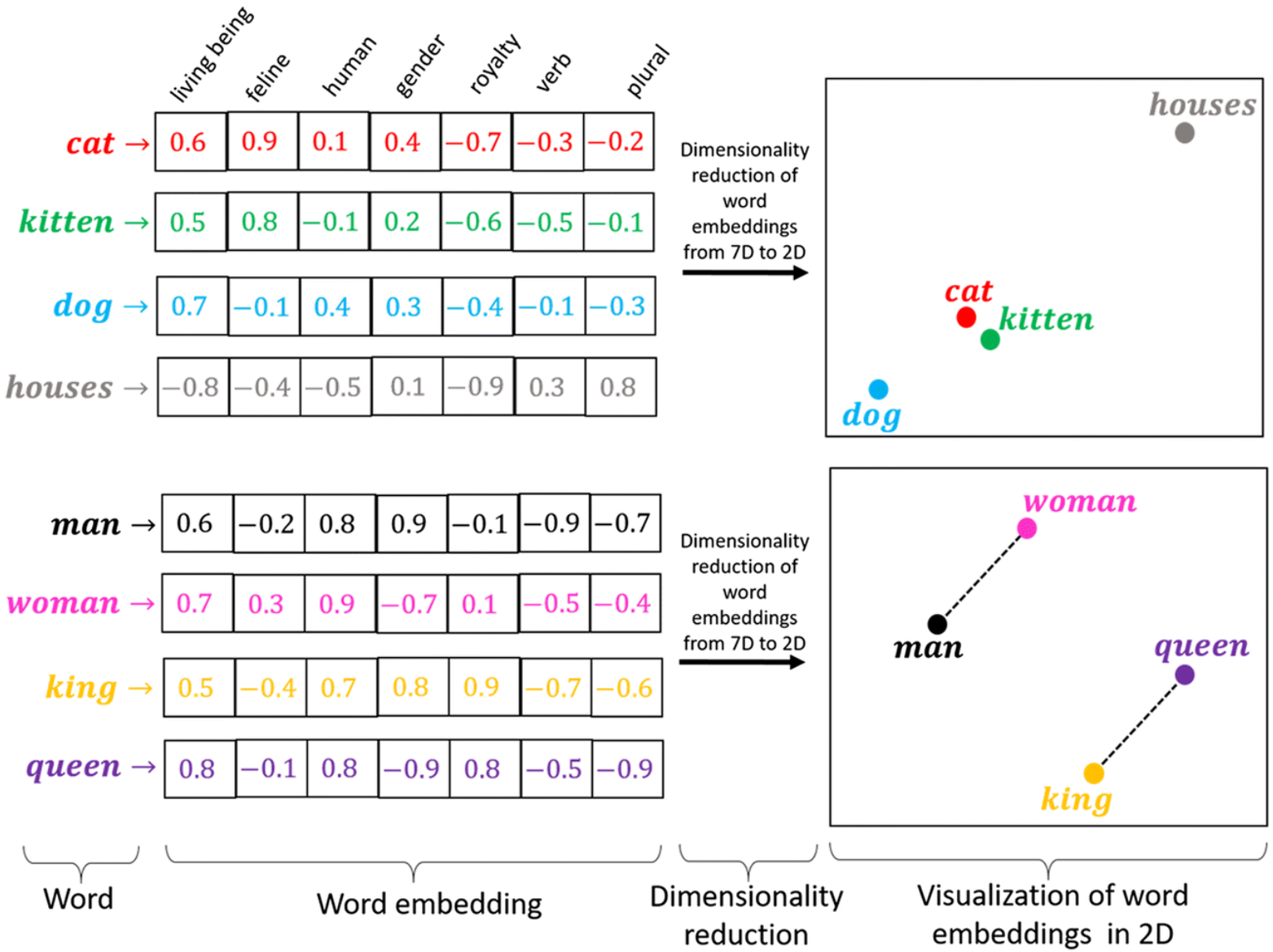
- Nastepna technika to one-hot encoding dla znaków.
- Zliczamy wszystkie występujące w tekście znaki. Każdemu znakowi przyporządkowujemy wektor binarny z jedynką wskazującą na znak.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
L →	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
U →	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
K →	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A →	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S →	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

- Kodowania są tu krótsze, bo znaków będzie pewnie tylko parędziesiąt.
- Uwaga! W tym przypadku sieć zgaduje tekst znak po znaku (na podstawie poprzednich znaków), a nie słowo po słowie (na podstawie poprzednich słów). Generowanie tekstu jest bardziej niskopoziomowe.

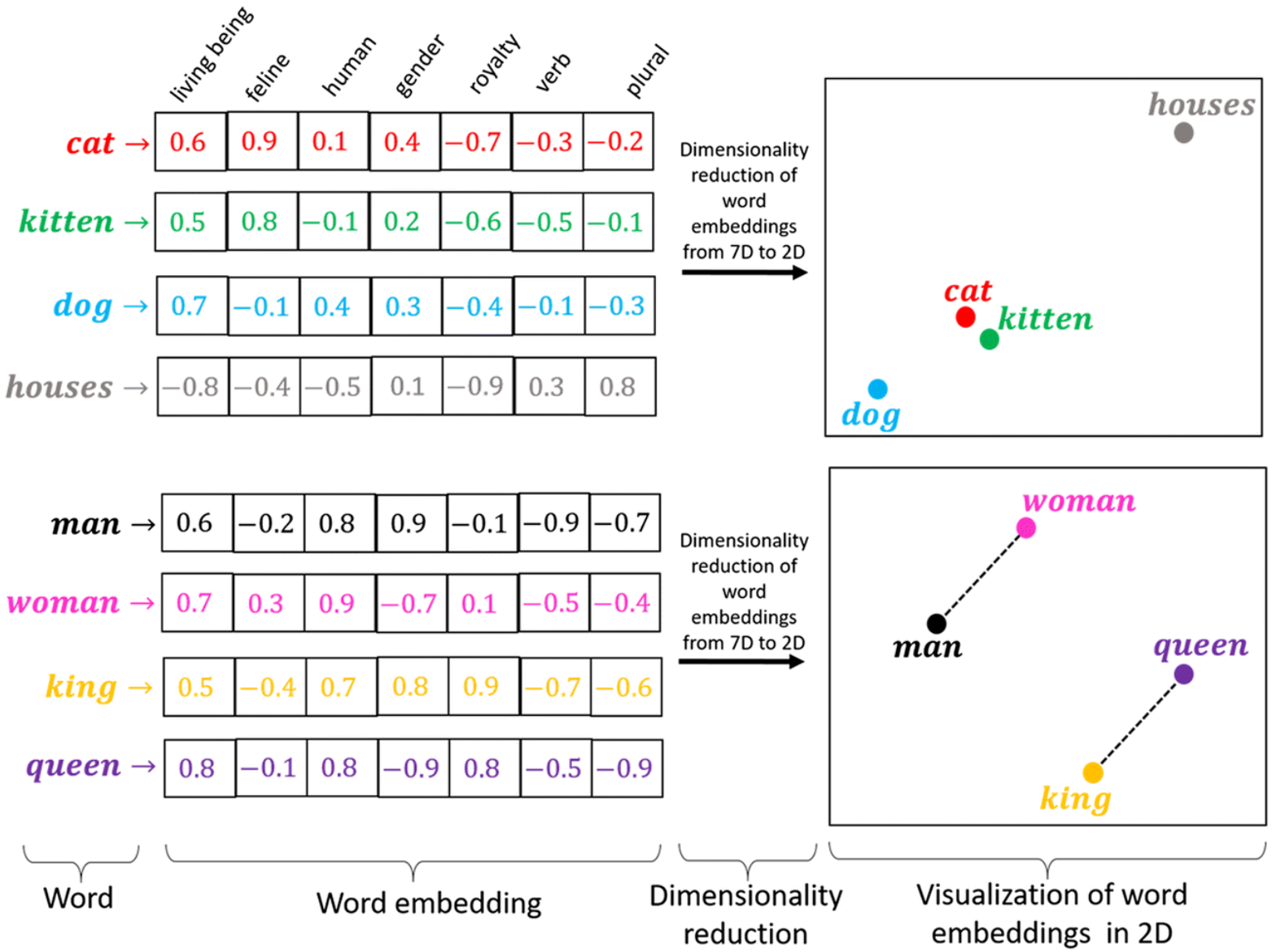
Generowanie tekstu: tekst na liczby

- Ostatnia technika to word embedding. Tutaj znowu zgadujemy słowa na podstawie poprzednich słów.
- W technice tej tworzymy specjalną macierz. Każde słowo otrzymuje wektor liczb (zmiennoprzecinkowych), które określają jak silnie dane słowo wpada do wybranego przez nas zestawu kategorii.



Generowanie tekstu: tekst na liczby

- Technika ta ma sporo zalet.
- Każde słowo kodowane jest jako dość krótki wektor liczb dla sieci RNN.
- W wektorze tym, nie ma niepotrzebnych zer.
- Wektory te uwzględniają znaczeniowe powiązania pomiędzy słowami, czego nie mogło uchwycić index-based czy one-hot encoding.



Generowanie tekstu: tekst na liczby

- Jak sensownie stworzyć macierz z word embeddings?
- Tworzenie ręczne nie ma sensu z uwagi na ogromny rozmiar słownika. Zadziała tylko w przypadku małych, sztucznie ograniczonych b danych.
- Można stworzyć zestaw tematów (ang. topics
- Następnie zbadać podobieństwo/przynależno danego słowa do każdego z tematu. Zestaw podobieństw będzie naszym word embeddingiem.
- Przydatne narzędzia word2vec, gensim (https://builtin.com/machine-learning/nlp-word2vec-python i https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/)

Generowanie tekstu za pomocą LSTM

- Stworzymy różne mdoele sieci LSTM i przetrenujemy je na książce „Alice in Wonderland” (wonderland.txt).

Alicja

(Alice)

Królik

(Rabbit)

Kapelusznik

(Hatter)

Gąsiennica

(Caterpillar)

Król Serc

(King)

- Badania robimy na bazie tutorialu: https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/

Generowanie tekstu za pomocą LSTM

- Stwórzmy sieć LSTM, która będzie generowała tekst litera po literze używając techniki index-based encoding dla znaków.
- Otwórz i uruchom plik lstm02.py
- W programie najpierw wyciągamy wszystkie możliwe znaki z książki, i mapujemy je na liczby naturalne.
- Przetwarzamy tekst i przygotowujemy próbki danych (patterns).
- Przygotowujemy model z warstwą LSTM i trenujemy go.
- Model będzie zapisywany w plikach z rozszerzeniem hdf5, przystosowanych do przechowywania danych hierarchicznych. W Visual Studio Code można zainstalować wtyczkę do odczytu tych plików.
- Trenowanie powinno trwać wiele epok (setki?). U mnie trenowanie tego modelu trwało 6h.

Generowanie tekstu za pomocą LSTM

- Przetestujemy teraz jak działa wytrenowany model.
- Otwórz i uruchom plik lstm03.py
- Do pliku wpisujemy odpowiedni filename, wskazujący gdzie zapisany jest nasz najlepszy model (ostatni plik hdf5).
- Losujemy fragment tekstu z książki (seed).
- Patrzymy jakie znaki sieć LSTM będzie generować za tym tekstem.

Generowanie tekstu za pomocą LSTM

- Przetestujmy drugi model LSTM wykorzystujący technikę index-based encoding dla słów.
- Modyfikujemy poprzednie programy, zamiast chars, mamy tokens (stworzone funkcją tokenize). Przetestuj działanie: lstm04.py (trenowanie) i lstm05.py (generowanie tekstu)
- Dla modelu seq_length = 20 (historia 20 słów) i hidden_units = 100 oraz pół godziny trenowania, wyniki nie były szczególnie ciekawe. Wynik poniżej.
- **Seed:** " the hedgehog was engaged in a fight with another hedgehog , which seemed to alice an excellent opportunity for croqueting "
- **Generated text:** , “ alice, “i’ve t’s ,” said the cat , “i’m’ stiff ,” said the king , “i proceed’t ,” said the hatter , “i’’t classics the,” said the king , “

Generowanie tekstu za pomocą LSTM

- Po zmodyfikowaniu modelu z parametrami seq_length = 100 (historia 100 słów) i hidden_units = 256 oraz więcej niż godzinie trenowania, mamy następujące wyniki:
- **Seed:** out again , so that altogether , for the first minute or two , it was as much as she could do to hold it . as soon as she had made out the proper way of nursing it , (which was to twist it up into a sort of knot , and then keep tight hold of its right ear and left foot , so as to prevent its undoing itself ,) she carried it out into the open air . “ if i don ’ t take this child away with me ,” thought alice , “
- **Generated text:** i ’ ll ’ t kill it ,” said the , “ alice , “ i ’ ll ,” said ,” said !” said the gryphon . “ i ’ archbishop said ’ t suit ,” “ the , said the dormouse . “ the , said the haven , “ considering the play the the , “ the ’ s a “ the wise , “ the finished . “ the you re the the mock history , “ the , “ ’ t verses the will rabbit want draw ,” said the hatter . “ i ’ s

Różne źródła

- https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn - tutorial z obrazkami wykorzystanymi na tym wykładzie
- https://www.ibm.com/topics/recurrent-neural-networks tutorial z obrazkami wykorzystanymi na tym wykładzie
- https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45 tutorial z obrazkami wykorzystanymi na tym wykładzie
- https://machinelearningmastery.com/understanding-simple-recurrent-neural-networks-in-keras/ pythonowy samouczek dla simple_rnn
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/ samouczek z obrazkami dla LSTM, wykorzystanymi na tym wykładzie
- https://medium.com/analytics-vidhya/nlp-text-encoding-a-beginners-guide-fa332d715854 rodzaje kodowań tekstu
- https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/ tutorial do LSTM