sages

Głębokie uczenie maszynowe / deep learning

REKURENCYJNE GŁĘBOKIE SIECI NEURONOWE

Problem klasyfikacji sekwencji

- Dane to ciągi różnych długości, np. tekst, muzyka, film.

Problem klasyfikacji sekwencji

- Dane to ciągi różnych długości, np. tekst, muzyka, film.
- Elementami ciągów są “obiekty bazowe”
  - Tekst - ciąg słów (ogólniej - tokenów)
  - Film - ciąg obrazów
  - Muzyka - ciąg dźwięków

Problem klasyfikacji sekwencji

- Dane to ciągi różnych długości, np. tekst, muzyka, film.
- Elementami ciągów są “obiekty bazowe”
  - Tekst - ciąg słów (ogólniej - tokenów)
  - Film - ciąg obrazów
  - Muzyka - ciąg dźwięków
- Uwaga 1: elementy ciągów są od siebie zależne!

Problem klasyfikacji sekwencji

- Dane to ciągi różnych długości, np. tekst, muzyka, film.
- Elementami ciągów są “obiekty bazowe”
  - Tekst - ciąg słów (ogólniej - tokenów)
  - Film - ciąg obrazów
  - Muzyka - ciąg dźwięków
- Uwaga 1: elementy ciągów są od siebie zależne!
- Uwaga 2: kolejność elementów jest istotna!

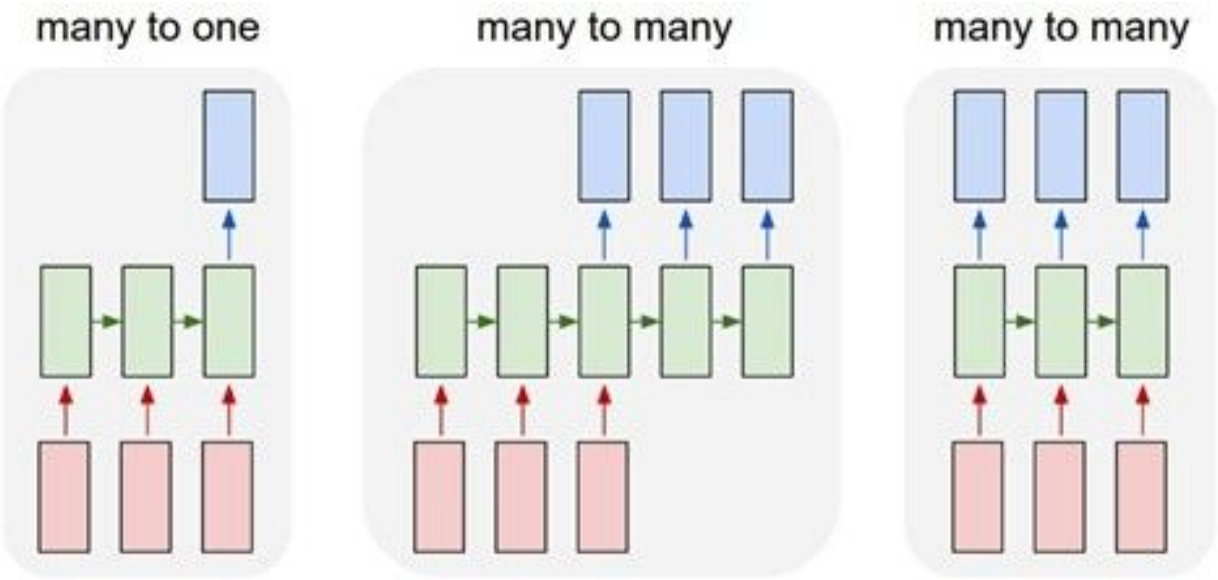
Problem klasyfikacji sekwencji

- Dane to ciągi różnych długości, np. tekst, muzyka, film.
- Elementami ciągów są “obiekty bazowe”
  - Tekst - ciąg słów (ogólniej - tokenów)
  - Film - ciąg obrazów
  - Muzyka - ciąg dźwięków
- Uwaga 1: elementy ciągów są od siebie zależne!
- Uwaga 2: kolejność elementów jest istotna!
- “Tradycyjne” metody klasyfikacji są przeznaczone dla obserwacji reprezentowanych przez wektor wspólnej długości. Zatem co możemy zrobić?

Problem klasyfikacji sekwencji

- Dane to ciągi różnych długości, np. tekst, muzyka, film.
- Elementami ciągów są “obiekty bazowe”
  - Tekst - ciąg słów (ogólniej - tokenów)
  - Film - ciąg obrazów
  - Muzyka - ciąg dźwięków
- Uwaga 1: elementy ciągów są od siebie zależne!
- Uwaga 2: kolejność elementów jest istotna!
- “Tradycyjne” metody klasyfikacji są przeznaczone dla obserwacji reprezentowanych przez wektor wspólnej długości. Zatem co możemy zrobić?
  - Sprowadzić ciągi do reprezentacji wektorowej
  - Użyć metod dedykowanych do takich danych

Schematy problemów



Analiza sentymentu

Tłumaczenie maszynowe

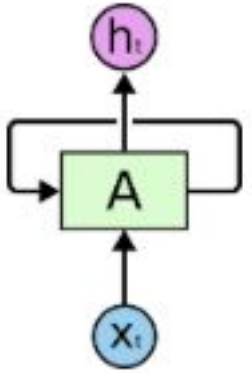
Detekcja nazw własnych

https://www.microsoft.com/en-us/cognitive-toolkit/blog/2016/11/sequence-to-sequence-deep-recurrent-neural-networks-in-cntk-part-1/

RNN (Recurrent Neural Networks)

- Są to sieci przetwarzające ciągi sekwencyjnie - sieć iteracyjnie wczytuje “element po elemencie”
- W kroku *t* sieć neuronowa A wczytuje aktualny element reprezentowany przez wektor x_t i oblicza stan ukryty h_t.
- Wartość obecnego stanu jest przekazywana do następnego kroku i wykorzystywana do obliczenia kolejnego stanu

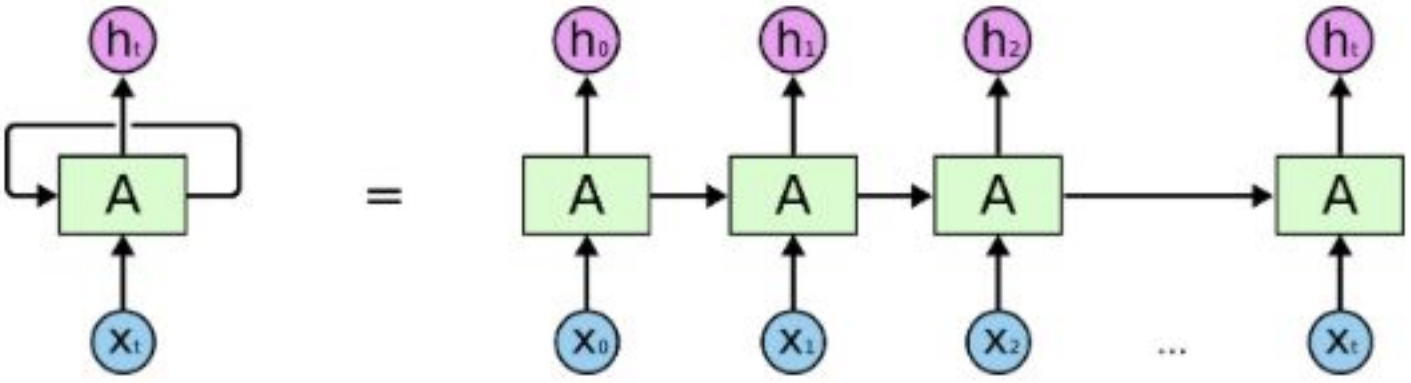
$$h_t = \tanh\left(Wx_t + Uh_{t-1} + b\right)$$



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

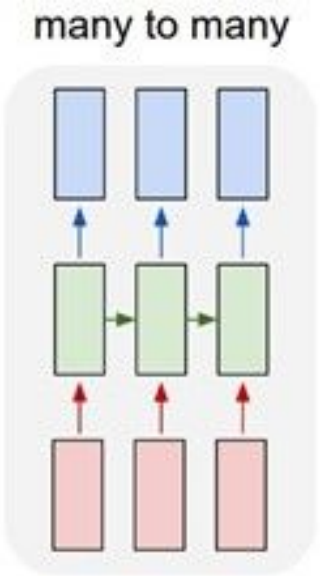
RNN (Recurrent Neural Networks)

$$h_t = \tanh\left(Wx_t + Uh_{t-1} + b\right)$$

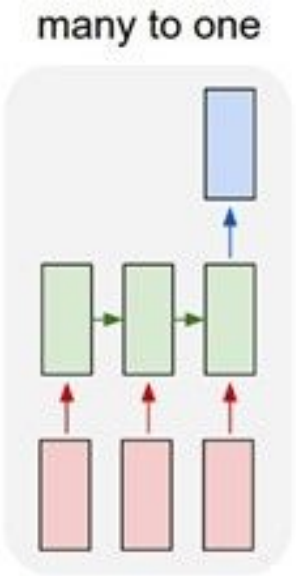


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN - predykcje



$\hat{y}_t = f(h_t)$



$\hat{y} = f(h_T)$

https://www.microsoft.com/en-us/cognitive-toolkit/blog/2016/11/sequence-to-sequence-deep-recurrent-neural-networks-in-cntk-part-1/

Przeanalizujmy co się dzieje w sieci, gdy podajemy słowa w reprezentacji *one hot*:

$$h_t = f(W^h * h_{t-1} + W^x * x_t + b)$$

Przeanalizujmy co się dzieje w sieci, gdy podajemy słowa w reprezentacji *one hot*:

$$h_t = f(W^h * h_{t-1} + W^x * x_t + b)$$

Jeśli x_t to *one-hot* z jedynką na pozycji *i* to:

$$W^x * x_t = W^x * [0, .., 0, 1_{(i)}, 0, .. 0]^T = W^x[:,i]$$

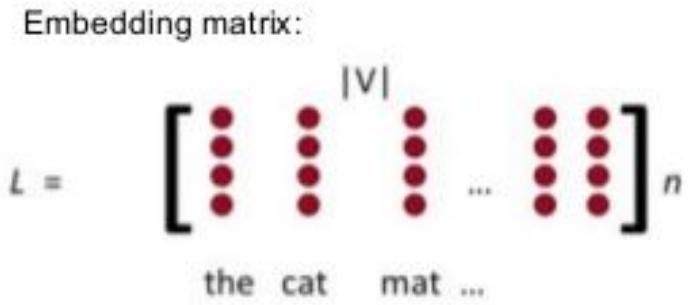
Wkład informacji ze słowa sprowadza się do wzięcia jednej kolumny macierzy wag.

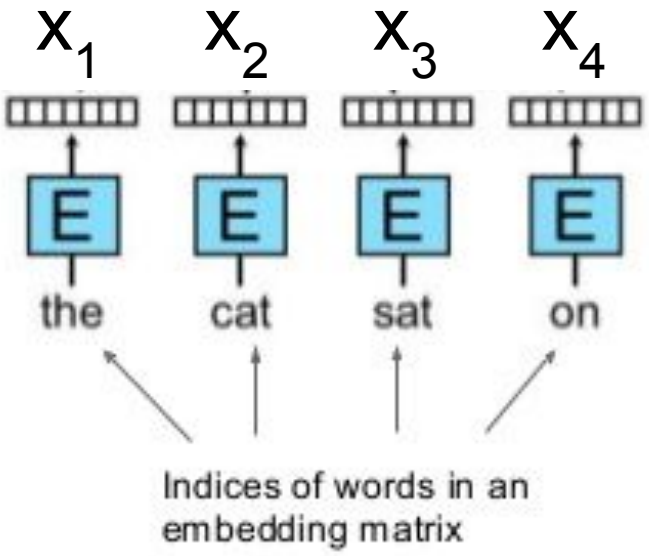
- Czyli *i*-ta kolumna macierzy wag jest w pewnym sensie reprezentacją słowa *i*-tego.
- Zatem pójdźmy krok dalej: stwórzmy dodatkową warstwę w sieci - macierz *embeddingów* EMB, zawierającą reprezentacje słów, które będą przekazywane do wyliczenia stanu ukrytego:

$$emb_t = EMB * x_t = EMB[:,i]$$

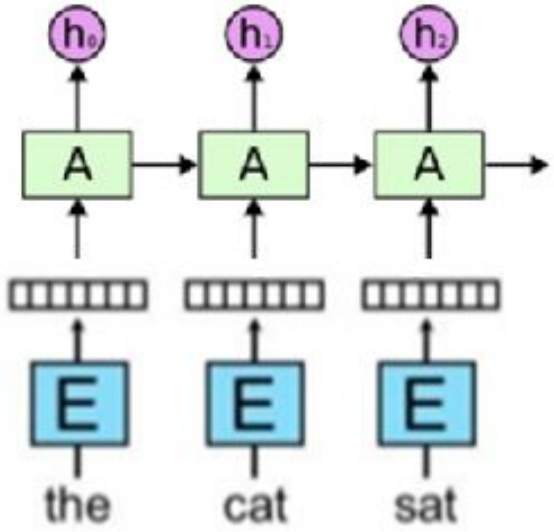
$$h_t = f(W^h * h_{t-1} + W^x * emb_t + b)$$

Embedding layer



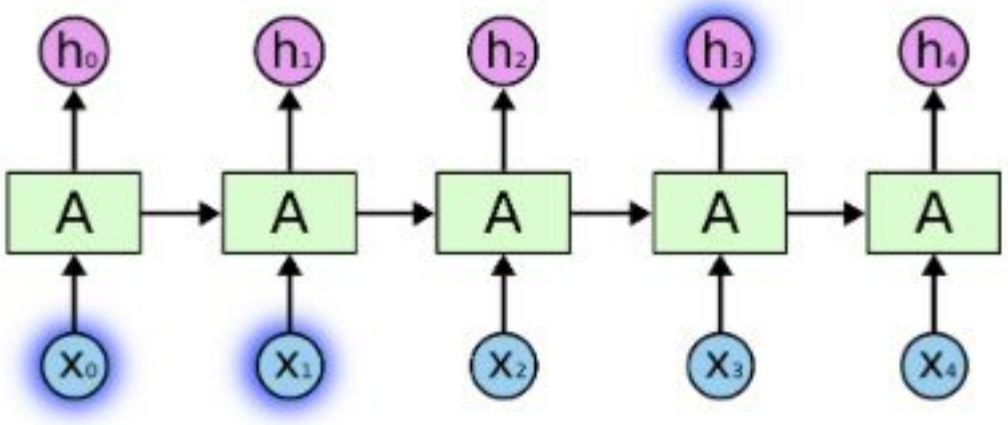


- *Embeddingi* są parametrami sieci, ale jednocześnie reprezentacją słów.
- Oznacza to, że trenując sieć, uczymy *embeddingi*, czyli uczymy się reprezentacji słów!



RNN

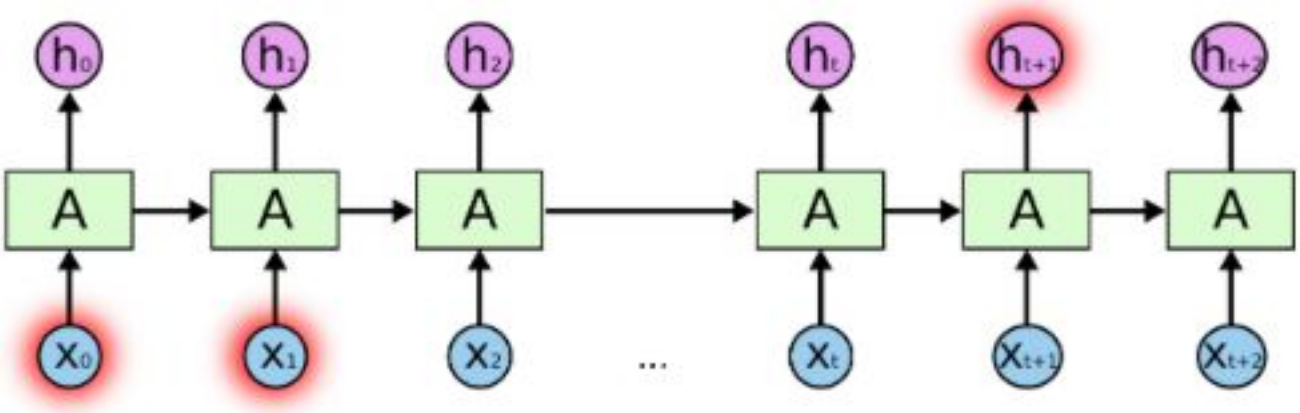
- Rozważmy problem predykcji następnego słowa po “the clouds are in the __”
- Jest to dość proste zadanie, bo odpowiedź można łatwo wywnioskować na podstawie tych kilku słów.
- W takich przypadkach zwykła sieć RNN są odpowiednią strukturą.



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN

- Próba przewidzenia “[I grew up in France… . I speak fluent ___” wymaga sięgnięcia wstecz dalej niż kilka słów.
- Ostatnie słowa sugerują tylko, że następne słowo jest nazwą języka - odgadnięcie, że chodzi o francuski, wymaga odnalezienia”France”.
- W praktyce dystans do relewantnej informacji często jest duży, a w miarę wzrostu tego dystansu, zwykła sieć RNN staje się niezdolna do wyłapania tych zależności.



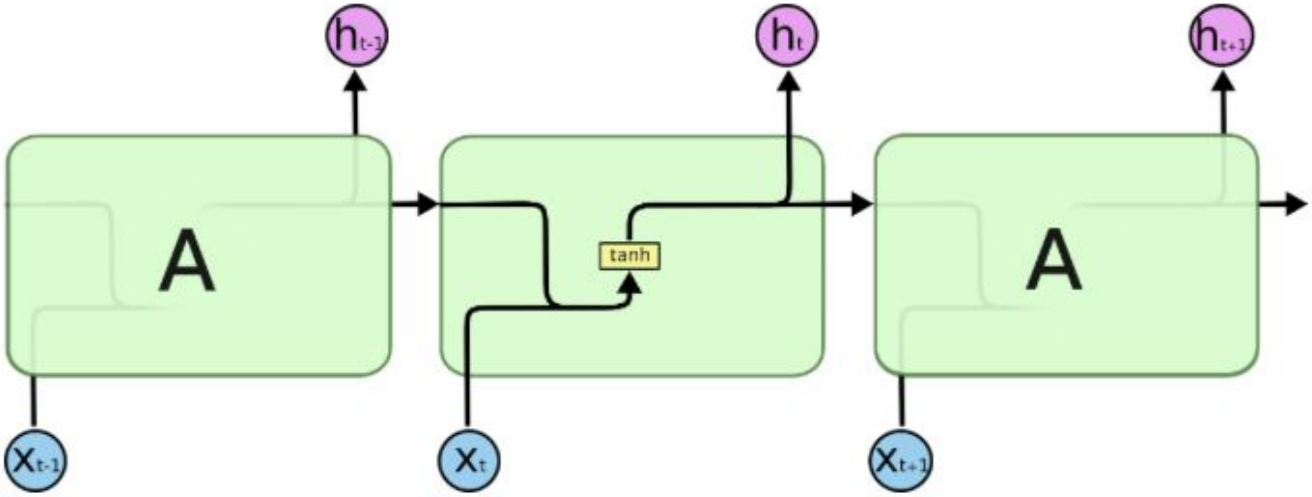
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

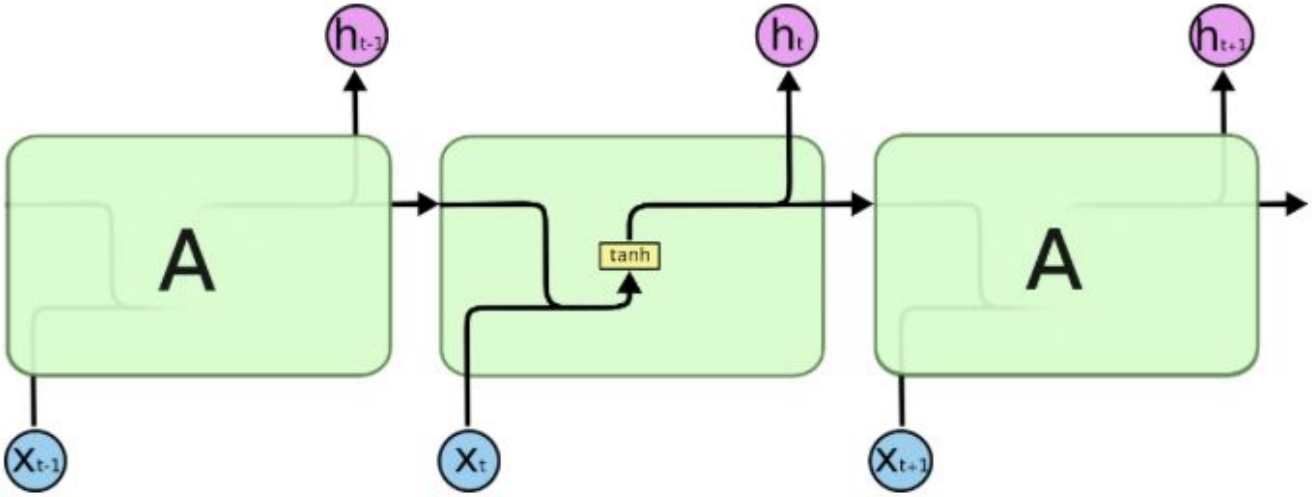
- Sieci Long Short Term Memory – zazwyczaj krótko “LSTM” – są szczególnym rodzajem sieci RNN, zdolnym do nauczenia się długodystansowych zależności. Hochreiter & Schmidhuber (1997)

- LSTMs zostały zaprojektowane w celu zaadresowania problemów z długodystansowymi zależnościami, zidentyfikowanymi w sieciach RNN.

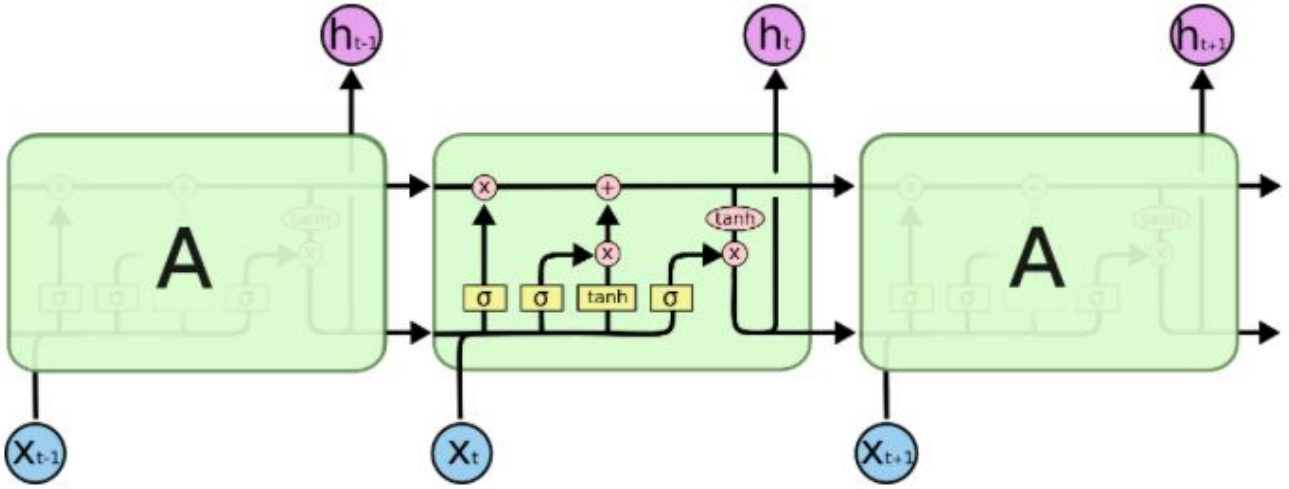
RNN

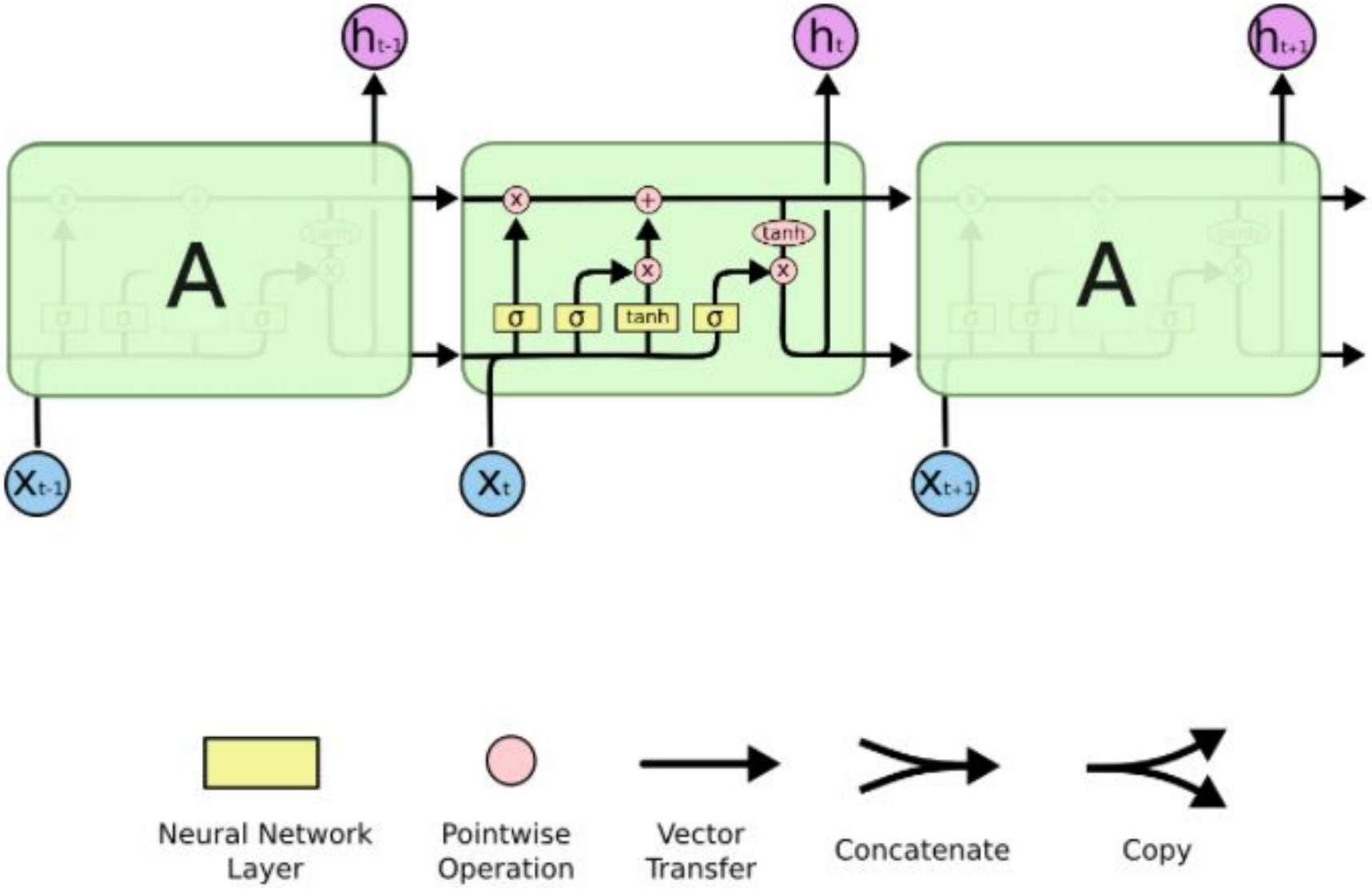


RNN

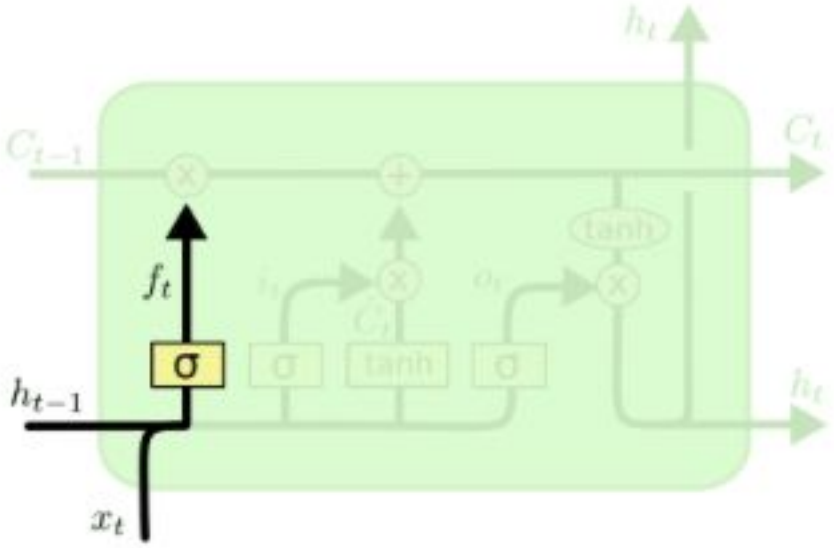

h_{t-1}
h_t
h_{t+1}
A
tanh
A
X_{t-1}
X_t
X_{t+1}

LSTM


h_{t-1}
h_t
h_{t+1}
A
σ
σ
tanh
σ
tanh
A
X_{t-1}
X_t
X_{t+1}


h_{t-1}
h_t
h_{t+1}
A
A
×
+
tanh
×
×
σ
σ
tanh
σ
X_{t-1}
X_t
X_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

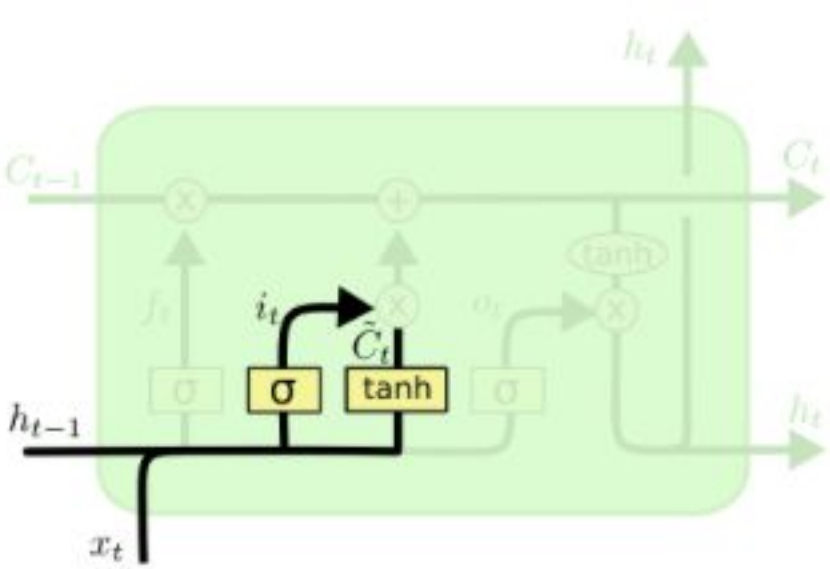
LSTM



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

- Decydujemy, których informacji mamy się pozbyć ze stanu komórki. Decyzję tą podejmuje sigmoidalna warstwa “forget gate” (bramka zapominania)
- Przyjmuje ona na wejście h_{t-1} oraz x_t , na wyjściu daje liczbę pomiędzy 0 a 1. “1” oznacza całkowicie pamiętać, “0” całkowicie zapomnieć.
- Liczby te są stosowane do każdej komórki stanu $C_{t-1.}$

LSTM

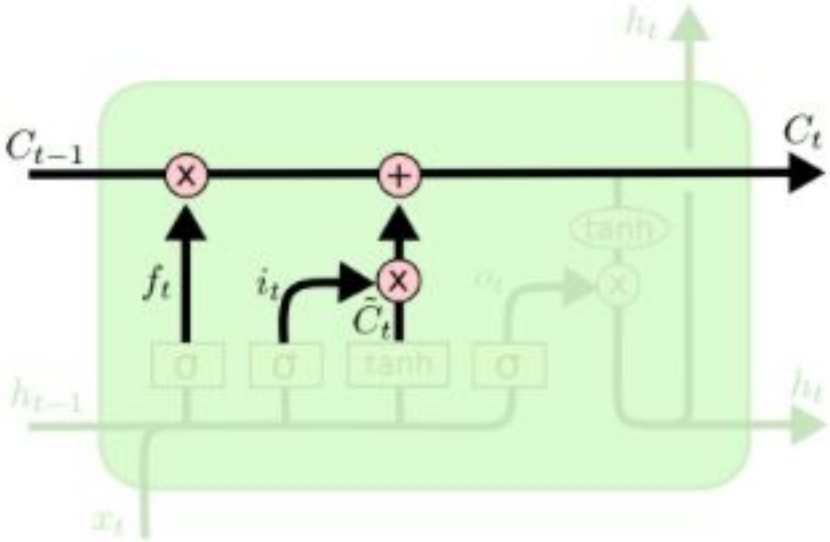


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

- W następnym kroku decydujemy, jaką nową informację umieścić w stanie komórki (cell state). Ma to dwie części.
- W pierwszym kroku, warstwa sigmoidalna “input gate” decyduje, które wartości będziemy uaktualniać.
- Następnie warstwa tanh tworzy wektor of kandydatów na nowe wartości $\check{C}_t$, które można dodać do stanu komórkowego.

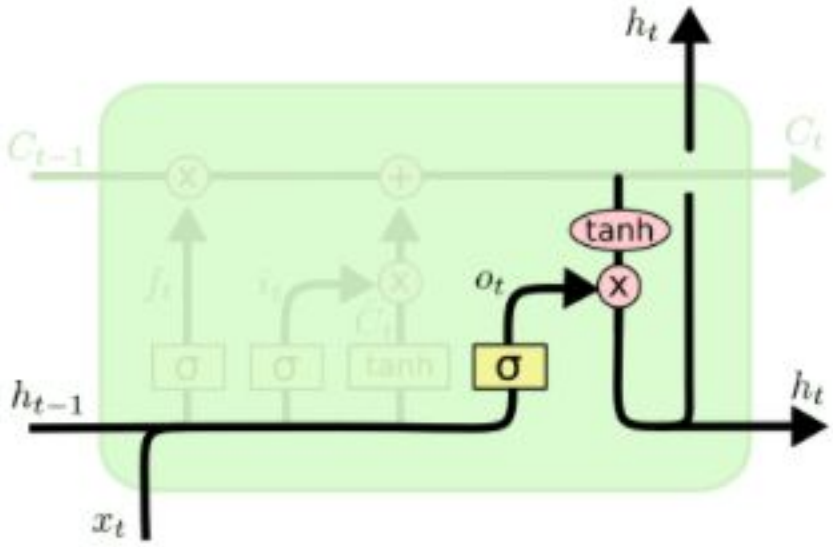
LSTM



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

- Następnie uaktualniamy stary stan komórkowy C_{t-1} nowym stanem C_t.
- Mnożymy stary stan przez f_t , zapominając to co zostało wcześniej wyznaczone.
- Następnie dodajemy $i_t * \check{C}_t$, czyli kandydatów na nowe wartości, przeskalowanych przez to, jak bardzo uaktualniamy każdą komórkę stanu.

LSTM



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

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

- Na koniec, decydujemy co zamierzamy zwrócić. Wyjście LSTM jest filtrowaną wersją stanu komórkowego (cell state).
- W pierwszym kroku uruchamiamy warstwę sigmoidalnej, która decyduje, którą część stanu komórkowego zwracamy.
- Następnie, mnożymy stan komórkowy przez tanh (otrzymując zakres -1 do 1) i to mnożymy przez wyjście warstwy sigmoidalnej, żeby uzyskać odpowiednie wyjście h_t.

LSTM równania

$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}\right), \\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}\right), \\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}\right), \\
u_t &= \tanh\left(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)}\right), \\
c_t &= i_t \odot u_t + f_t \odot c_{t-1}, \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
$$