Rekurencyjne sieci neuronowe, LSTM i GRU

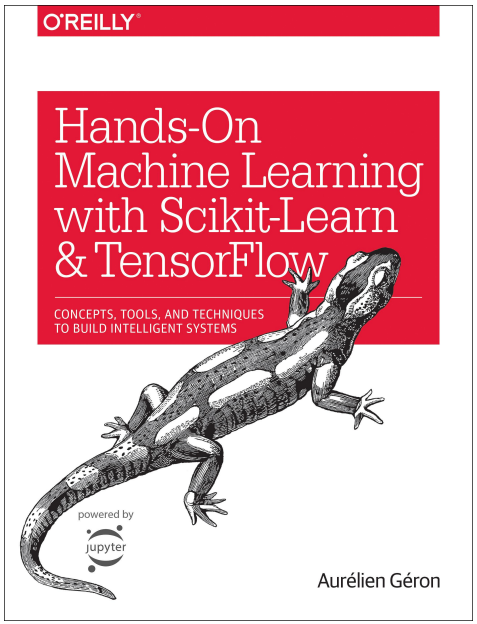
Agnieszka Ławrynowicz

Wydział Informatyki Politechniki Poznańskiej

26 kwietnia 2020

Materiał źródłowy

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



Rekurencyjne sieci neuronowe - motywacja

- ideą rekurencyjnych sieci neuronowych (ang. *Recurrent 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

Rekurencyjne sieci neuronowe - motywacja c.d.

- 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

Rekurencyjne sieci neuronowe - zastosowania

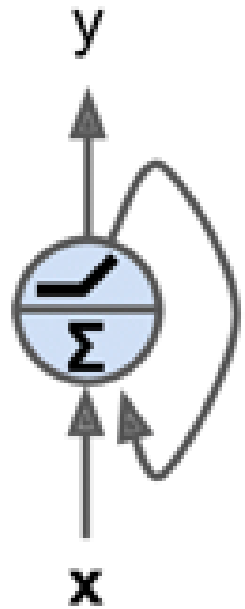
- analiza szeregów czasowych, np. cen akcji w celu predykcji kupna/sprzedaży
- przewidywanie trajektorii samochodów autonomicznych w celu unikania wypadków
- przetwarzanie języka naturalnego, np.:
  - automatyczne tłumaczenie,
  - konwersja tekstu na mowę (*speech-to-text*)
  - analiza sentymentu

Dlaczego nie "zwykłe" sieci neuronowe?

- 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

Rekurencyjny neuron

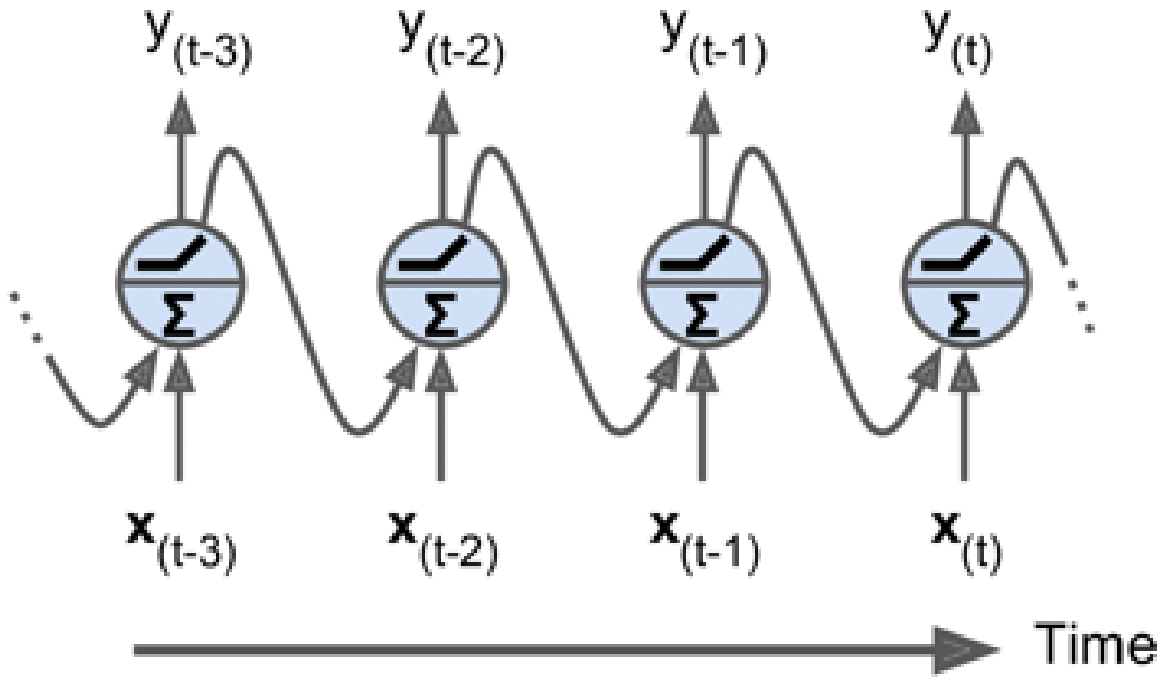
Najprostszy neuron RNN:
W każdym kroku czasowym t (zwanym również ramką) ten neuron rekurencyjny odbiera wejścia $\mathbf{x}^{(t)}$ oraz własne wyjście z poprzedniego kroku czasowego $\mathbf{y}^{(t-1)}$.



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

Rekurencyjny neuron rozwinięty w czasie

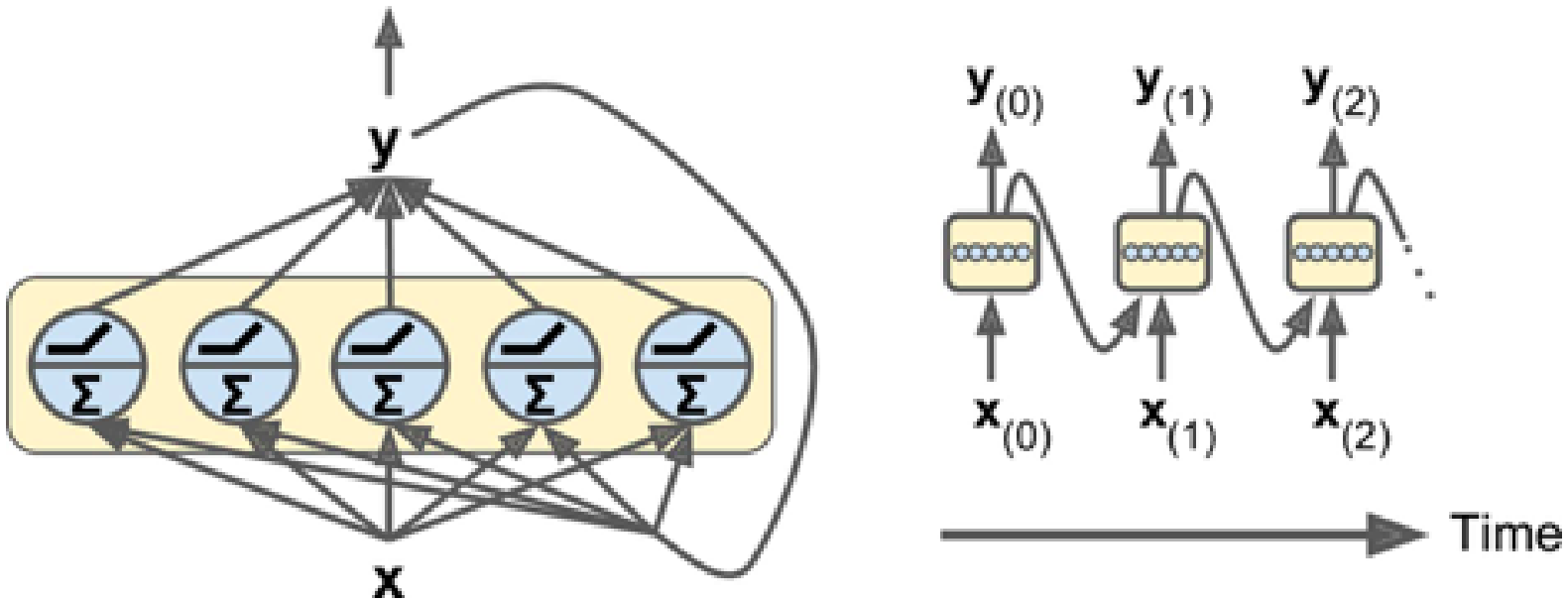
Możemy reprezentować tę małą sieć neuronową na osi czasu



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

Warstwa neuronów rekurencyjnych

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



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

Wyjście pojedynczego neuronu rekurencyjnego dla pojedynczej instancji

- każdy neuron rekurencyjny ma dwa zestawy wag:
  - $\mathbf{w_x}$: dla wejść $\mathbf{x}^{(t)}$
  - $\mathbf{w_y}$: dla wyjść z poprzedniego kroku czasu $\mathbf{y}^{(t-1)}$
- dane wyjściowe pojedynczego neuronu można obliczyć wg równania:

$$\mathbf{y}^{(t)} = \phi\left(\mathbf{x}^{(t)T} \cdot \mathbf{w_x} + \mathbf{y}^{(t-1)T} \cdot \mathbf{w_y} + b\right)$$

gdzie $\phi(\cdot)$ jest funkcją aktywacji, np. ReLU, a b to *bias*

Wyjście pojedynczego neuronu rekurencyjnego dla pojedynczej instancji: wersja zwektoryzowana

$$\mathbf{Y}^{(t)} = \phi\left(\mathbf{X}^{(t)} \cdot \mathbf{W_x} + \mathbf{Y}^{(t-1)} \cdot \mathbf{W_y} + b\right) =$$

$$= \phi\left(\left[\mathbf{X}^{(t)} \cdot \mathbf{Y}^{(t-1)}\right] \cdot \mathbf{W} + b\right)$$

gdzie

$$\mathbf{W} = \begin{bmatrix} \mathbf{W_x} \\ \mathbf{W_y} \end{bmatrix}$$

Komórki pamięci

Ponieważ wyjście neuronu rekurencyjnego w kroku t jest funkcją wszystkich danych wejściowych z poprzednich kroków czasowych, można powiedzieć, że ma postać pamięci (ang. *memory*).

Definition (Komórka pamięci)

Komórka pamięci (ang. *memory cell*) jest częścią sieci neuronowej, która zachowuje pewien stan na różnych etapach czasowych.

Pojedynczy neuron rekurencyjny jest bardzo podstawową komórką.

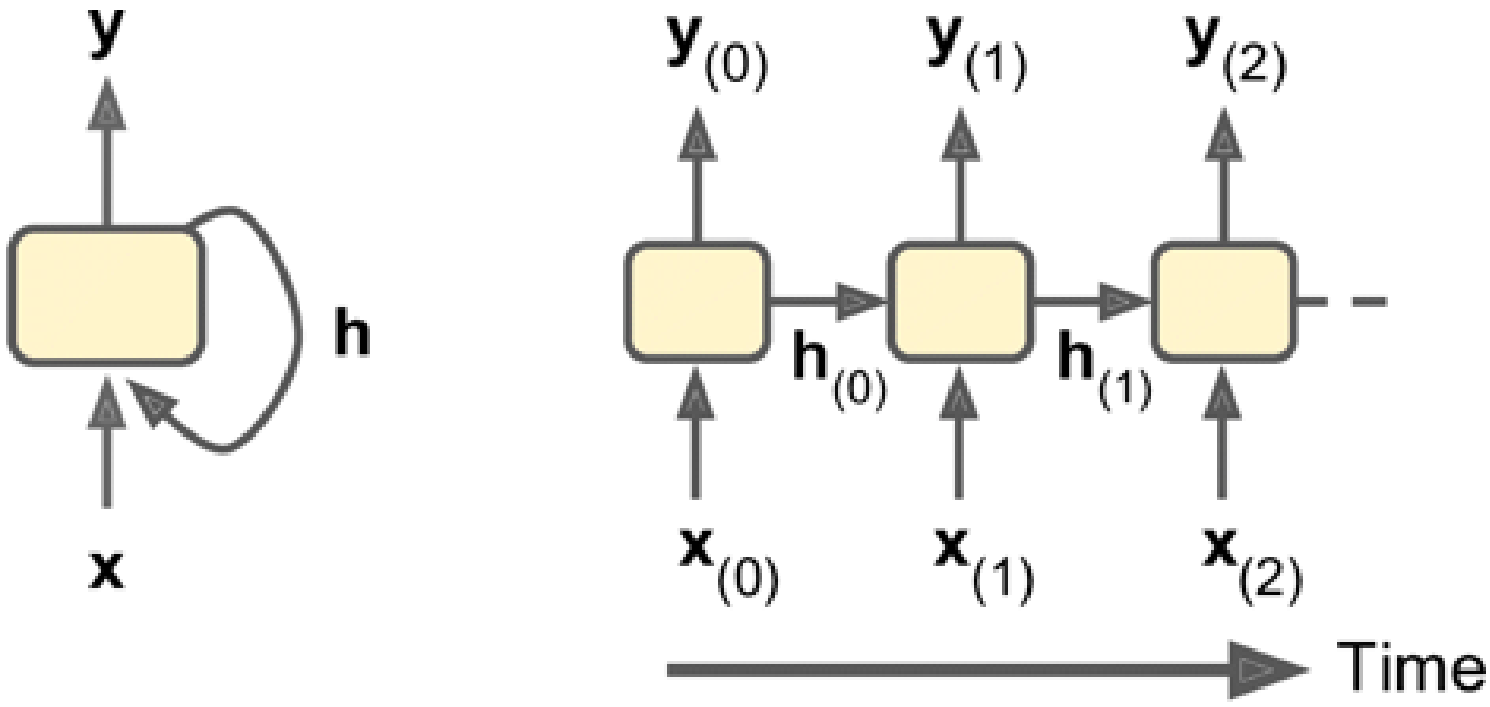
Komórki pamięci c.d.

- $\mathbf{h}^{(\mathbf{t})}$: stan komórki w kroku czasowym t (h oznacza 'ukryty' (*hidden*))
- $\mathbf{h}^{(\mathbf{t})}$ jest funkcją pewnych wejść w tym kroku czasowym i jego stanu w poprzednim kroku czasowym:

$$\mathbf{h}^{(\mathbf{t})} = f(\mathbf{h}^{(\mathbf{t-1})}, \mathbf{x}^{(\mathbf{t})})$$

- jego wyjście w kroku t, oznaczane jako $\mathbf{y}^{(\mathbf{t})}$, jest również funkcją poprzedniego stanu i bieżących wejść
- w przypadku bardziej skomplikowanych komórek, wyjście nie zawsze jest równe stanowi

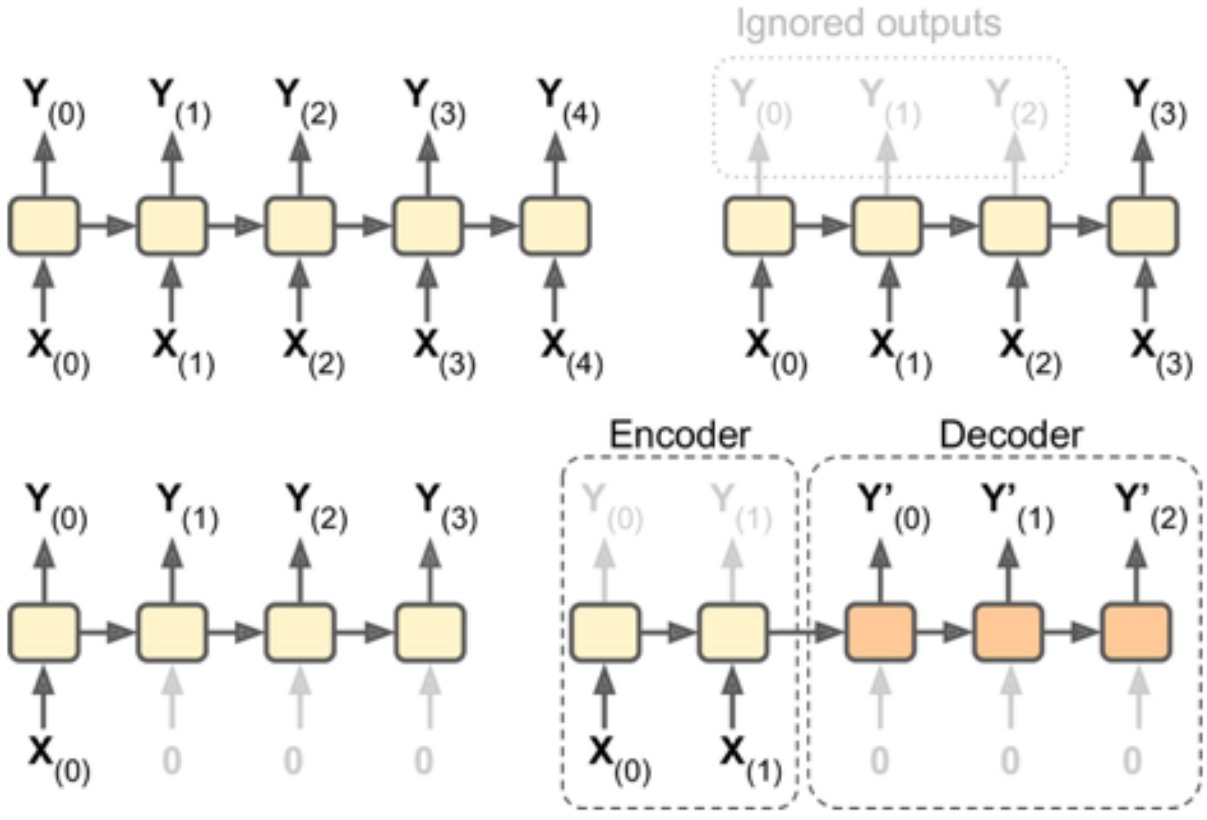
Stan ukryty komórki i wyjście mogą się różnić



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

Sekwencje wejściowe i wyjściowe

RNN może jednocześnie przyjmować sekwencję wejść i generować sekwencję wyjść



Seq to seq (lewa górna sieć), *seq to vector* (prawa górna sieć), *vector to seq* (lewa dolna sieć), opóźniona *seq to seq* (prawa dolna sieć).

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

Sekwencje wejściowe i wyjściowe - przykład

- Przykładowo, można zasilić sieć sekwencją wejść i zignorować wszystkie wyjścia z wyjątkiem ostatniego (patrz: prawa górna sieć na poprzednim slajdzie).
- Jest to sieć *seq to vector*.
- Np., można zasilić sieć sekwencją wyrazów odpowiadającą recenzji filmu, a sieć wyświetli wynik sentymentu (np. od -1 [nienawiść] do $+1$ [uwielbienie]).

Seq to seq vs opóźniona *seq to seq*

Jak myślisz, która z tych sieci (sekwencyjna czy opóźniona sekwencyjna) sprawdzi się lepiej w zadaniu tłumaczenia zdania na inny język?

Seq to seq vs opóźniona *seq to seq*

Jak myślisz, która z tych sieci (sekwencyjna czy opóźniona sekwencyjna) sprawdzi się lepiej w zadaniu tłumaczenia zdania na inny język?

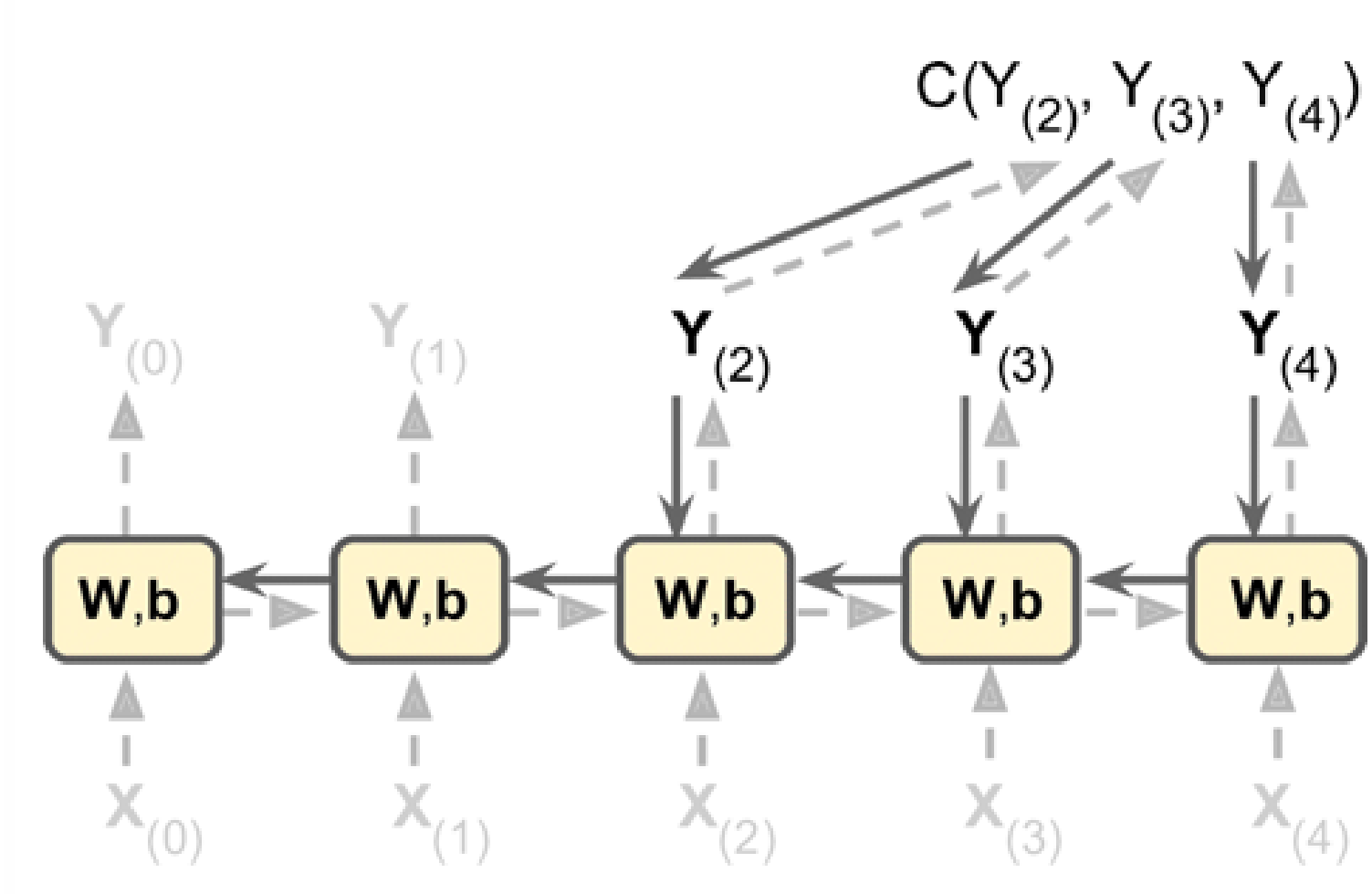
Dwuetapowy model (enkoder-dekoder) sprawuje się lepiej niż zwykła sekwencyjna sieć, która próbuje tłumaczyć zdania na bieżąco, gdyż ostatnie słowa w zdaniu mogą wpływać na znaczenie całego zdania i należy na nie poczekać zanim podejmie się tłumaczenie zdania.

Trenowanie RNN

Backpropagation through time (BPTT)

1. rozwinięcie RNN w czasie
2. użyć regularnej propagacji wstecznej

Backpropagation through time (BPTT)



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

Backpropagation through time (BPTT) c.d.

- podobnie jak w zwykłej wstecznej propagacji, występuje pierwsze przejście w przód przez rozwiniętą w czasie sieć (przerywane strzałki)
- następnie sekwencja wyjściowa jest obliczana za pomocą funkcji kosztu:

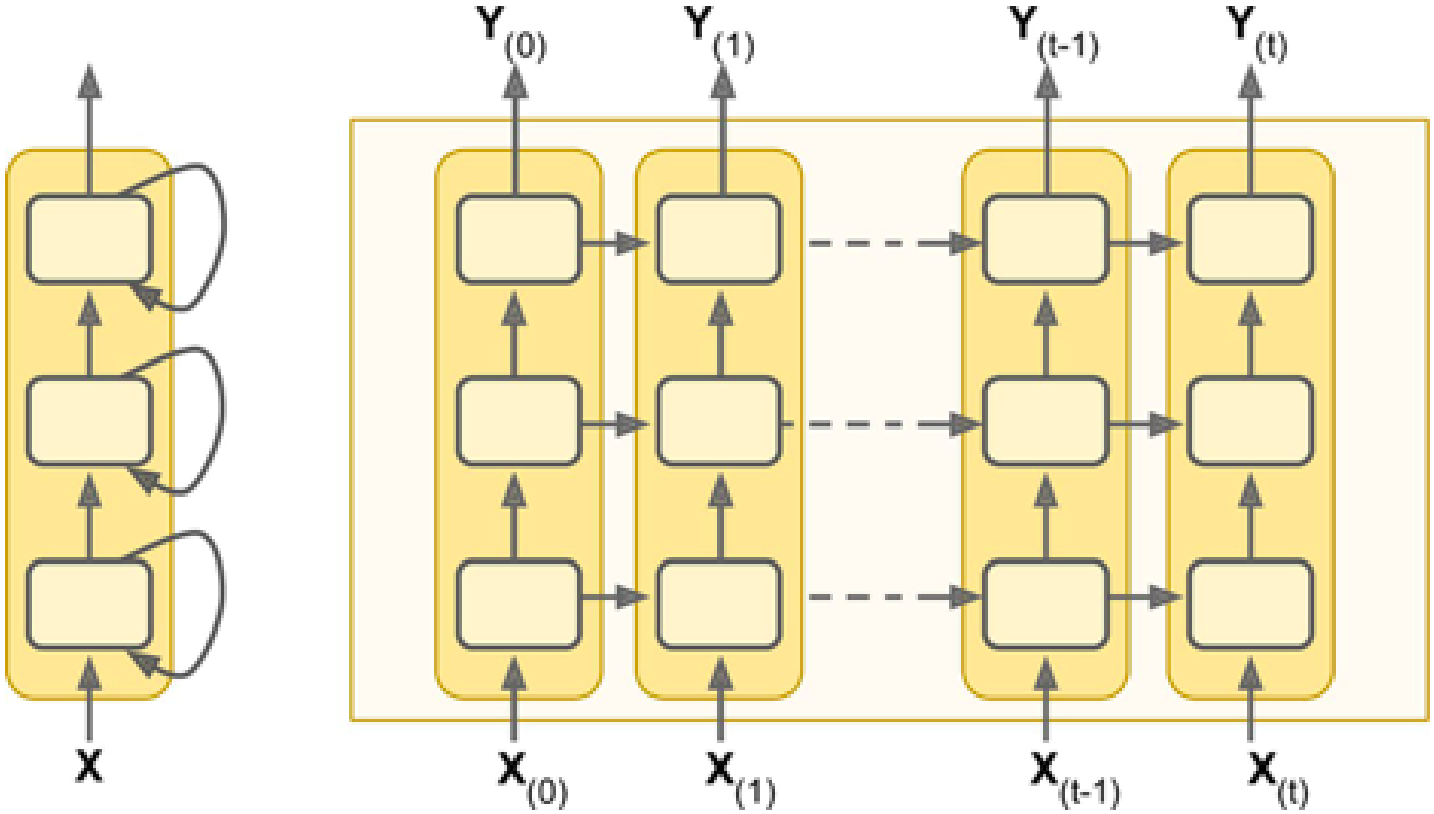
$$C(\mathbf{Y}^{t_{min}}, \mathbf{Y}^{t_{min}+1}), ..., \mathbf{Y}^{t_{max}}$$

(gdzie t_{min} i t_{max} są pierwszym i ostatnim krokiem czasu wyjściowego, nie licząc ignorowanych wyjść), a gradienty tej funkcji kosztu są propagowane do tyłu przez rozwiniętą w czasie sieć (pełne strzałki)

- wreszcie parametry modelu są aktualizowane przy użyciu gradientów obliczonych podczas BPTT

Głębokie RNN

Powszechne jest nakładanie wielu warstw komórek pamięci



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

Trudności w trenowaniu RNN przez wiele etapów czasowych

- długi czas trenowania
- pamięć pierwszych wejść stopniowo zanika, a po pewnym czasie stan RNN nie zawiera praktycznie żadnych śladów pierwszych wejść

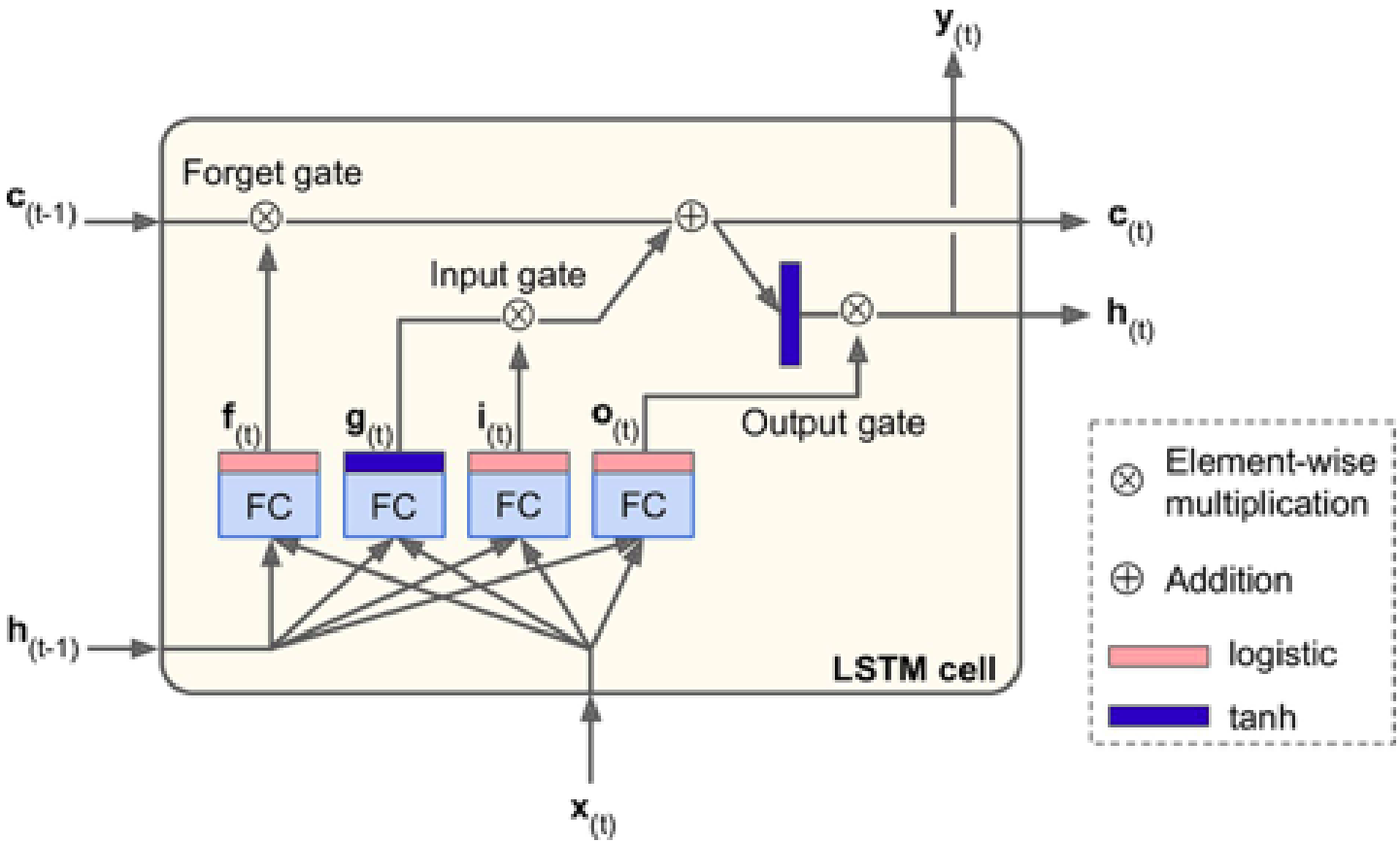
Problem zaniku pierwszych wejść - przykład

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.

Problem zaniku pierwszych wejść - rozwiązanie

- różnego rodzaju komórki z długo-terminową pamięcią
- na tyle skuteczne, że podstawowe komórki nie są już dużo używane
- najpopularniejsza: LSTM (ang. *Long Short-Term Memory*) (Sepp Hochreiter i Juergen Schmidhuber, 1997), następnie zmodyfikowana przez Gravesa, Saka, Zarembę i innych.

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.

Działanie LSTM - na zewnątrz

- traktując komórkę LSTM jako ”czarną skrzynkę”, to wygląda ona dokładnie jak zwykła komórka, z tym że jej stan jest podzielony na dwa wektory: $\mathbf{h}^{(\mathbf{t})}$ i $\mathbf{c}^{(\mathbf{t})}$ (”c” oznacza "komórkę", *cell*):
  -
  - $\mathbf{h}^{(\mathbf{t})}$ odpowiada stanowi krótkoterminowemu
  - $\mathbf{c}^{(\mathbf{t})}$ odpowiada stanowi długoterminowemu

Działanie LSTM - wewnątrz

Kluczowa idea: sieć może nauczyć się, co przechowywać w stanie długoterminowym, co wyrzucić i co z niego czytać:

- stan długoterminowy $\mathbf{c}^{(\mathbf{t-1})}$ przemierza sieć od lewej do prawej, najpierw przechodząc przez bramkę "zapomnij", porzucając niektóre wspomnienia, a następnie dodając nowe wspomnienia poprzez operację dodawania (która dodaje wspomnienia wybrane przez bramkę wejściową).
- wynik $\mathbf{c}^{(\mathbf{t})}$ jest wysyłany bez dalszej transformacji
- tak więc za każdym razem niektóre wspomnienia są porzucane a niektóre dodawne
- po operacji dodawania, stan długoterminowy jest kopiowany i przechodzi przez funkcję *tanh* (tangens hiperboliczny), a następnie wynik jest filtrowany przez bramkę wyjściową
- powoduje to powstanie stanu krótkoterminowego $\mathbf{h}^{(\mathbf{t})}$ (który jest równy wyjściu komórki dla tego kroku czasowego $\mathbf{y}^{(\mathbf{t})}$).

LSTM - warstwy

Bieżący wektor wejściowy $\mathbf{x}^{(\mathbf{t})}$ i poprzedni stan krótkoterminowy $\mathbf{h}^{(\mathbf{t-1})}$ są podawane do czterech różnych, w pełni połączonych warstw:

- jedna warstwa główna
- trzy warstwy kontrolerów bramkowych

Warstwa główna

- ma na wyjściu $\mathbf{c}^{(\mathbf{t})}$
- pełni rolę w analizie bieżących wejść $\mathbf{c}^{(\mathbf{t})}$ i poprzedniego (krótkoterminowego) stanu $\mathbf{h}^{(\mathbf{t-1})}$
- w podstawowej komórce nie ma nic więcej oprócz tej warstwy, a jej wynik idzie prosto do $\mathbf{y}^{(\mathbf{t})}$ i $\mathbf{h}^{(\mathbf{t})}$
- w komórce LSTM wynik tej warstwy jest częściowo przechowywany w stanie długoterminowym

Kontrolery bramkowe

- używają logistycznej funkcji aktywacji, więc ich wyjścia są w zakresie od 0 do 1
- ich wyjścia są podawane do operacji mnożenia: jeśli wynikiem są 0, zamykają bramkę, a jeśli 1, to otwierają bramkę.

Bramka ”zapomnij”

Bramka "zapomnij" (sterowana przez $\mathbf{f}^{(\mathbf{t})}$) kontroluje, które części stanu długoterminowego powinny zostać usunięte.

Bramka wejściowa

Bramka wejściowa (sterowana przez $\mathbf{i}^{(\mathbf{t})}$) kontroluje, które części $\mathbf{g}^{(\mathbf{t})}$ powinny być dodane do stanu długoterminowego.

Bramka wyjściowa

Bramka wyjściowa (sterowana przez $\mathbf{o}^{(\mathbf{t})}$) kontroluje, które części stanu długoterminowego powinny być odczytywane i wyprowadzane w danym momencie (zarówno do $\mathbf{h}^{(\mathbf{t})}$), jak i $\mathbf{y}^{(\mathbf{t})}$).

LSTM - obliczenia

$$\mathbf{i}^{(\mathbf{t})} = \sigma(\mathbf{W}^{(\mathbf{xi})\mathbf{T}} \cdot \mathbf{x}^{(\mathbf{t})} + \mathbf{W}^{(\mathbf{hi})\mathbf{T}} \cdot \mathbf{h}^{(\mathbf{t-1})} + \mathbf{b}^{(\mathbf{i})})$$
$$\mathbf{f}^{(\mathbf{t})} = \sigma(\mathbf{W}^{(\mathbf{xf})\mathbf{T}} \cdot \mathbf{x}^{(\mathbf{t})} + \mathbf{W}^{(\mathbf{hf})\mathbf{T}} \cdot \mathbf{h}^{(\mathbf{t-1})} + \mathbf{b}^{(\mathbf{f})})$$
$$\mathbf{o}^{(\mathbf{t})} = \sigma(\mathbf{W}^{(\mathbf{xo})\mathbf{T}} \cdot \mathbf{x}^{(\mathbf{t})} + \mathbf{W}^{(\mathbf{ho})\mathbf{T}} \cdot \mathbf{h}^{(\mathbf{t-1})} + \mathbf{b}^{(\mathbf{o})})$$
$$\mathbf{g}^{(\mathbf{t})} = tanh(\mathbf{W}^{(\mathbf{xg})\mathbf{T}} \cdot \mathbf{x}^{(\mathbf{t})} + \mathbf{W}^{(\mathbf{hg})\mathbf{T}} \cdot \mathbf{h}^{(\mathbf{t-1})} + \mathbf{b}^{(\mathbf{g})})$$
$$\mathbf{c}^{(\mathbf{t})} = \mathbf{f}^{(\mathbf{t})} \otimes \mathbf{c}^{(\mathbf{t-1})} + \mathbf{i}^{(\mathbf{t})} \otimes \mathbf{g}^{(\mathbf{t})} \quad \mathbf{y}^{(\mathbf{t})} = \mathbf{h}^{(\mathbf{t})} = \mathbf{o}^{(\mathbf{t})} \otimes tanh(\mathbf{c}^{(\mathbf{t})})$$

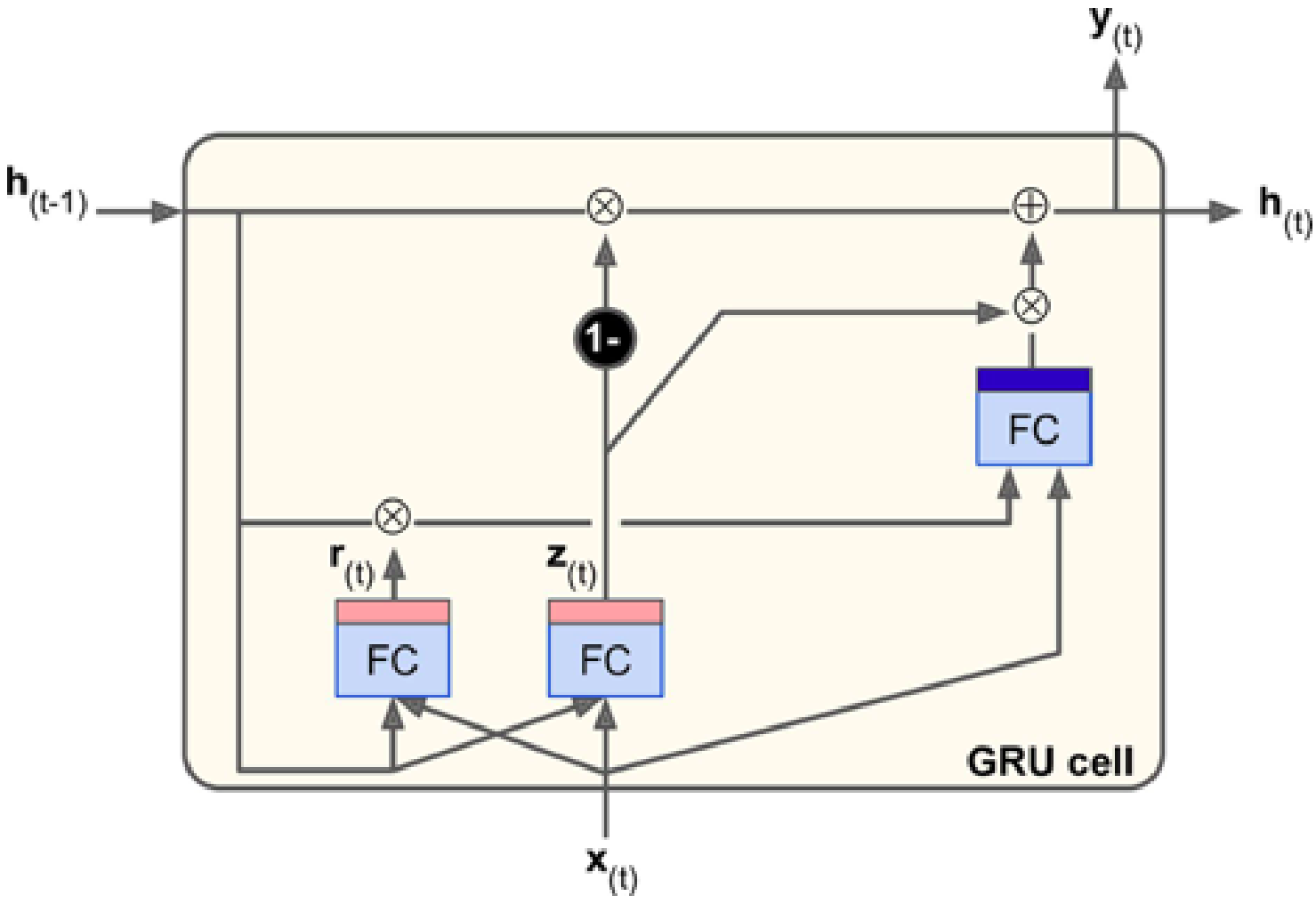
$\mathbf{W}^{(\mathbf{xi})}, \mathbf{W}^{(\mathbf{xf})}, \mathbf{W}^{(\mathbf{xo})}, \mathbf{W}^{(\mathbf{xg})}$: wagi macierzy każdej z czterech warstw dla ich połączenia z wektorem wejściowym $\mathbf{x}^{(\mathbf{t})}$
$\mathbf{W}^{(\mathbf{hi})}, \mathbf{W}^{(\mathbf{hf})}, \mathbf{W}^{(\mathbf{ho})}, \mathbf{W}^{(\mathbf{hg})}$: wagi macierzy każdej z czterech warstw dla ich połączenia z poprzednim stanem krótkoterminowym $\mathbf{h}^{(\mathbf{t-1})}$
$\mathbf{b}^{(\mathbf{i})}, \mathbf{b}^{(\mathbf{f})}, \mathbf{b}^{(\mathbf{o})}, \mathbf{b}^{(\mathbf{g})}$: *bias* dla każdej z czterech warstw

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 $\mathbf{h}^{(t)}$
- 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.

Bramka ”aktualizuj”

Bramka "aktualizuj" (sterowana przez $\mathbf{z}^{(\mathbf{t})}$) pomaga określić, ile wcześniejszych informacji (z poprzednich kroków czasowych) należy przekazać do ”przyszłości”.

Bramka ”resetuj”

Bramka "resetuj" (sterowana przez $\mathbf{r^{(t)}}$) kontroluje ile informacji z przeszłości ”zapomieć”.

GRU- obliczenia

$$\mathbf{z^{(t)}} = \sigma(\mathbf{W^{(xz)T}} \cdot \mathbf{x^{(t)}} + \mathbf{W^{(hz)T}} \cdot \mathbf{h^{(t-1)}} + \mathbf{b^{(i)}})$$
$$\mathbf{r^{(t)}} = \sigma(\mathbf{W^{(xr)T}} \cdot \mathbf{x^{(t)}} + \mathbf{W^{(hr)T}} \cdot \mathbf{h^{(t-1)}} + \mathbf{b^{(f)}})$$
$$\mathbf{g^{(t)}} = tanh(\mathbf{W^{(xg)T}} \cdot \mathbf{x^{(t)}} + \mathbf{W^{(hg)T}} \cdot (\mathbf{r^{(t)}} \otimes \mathbf{h^{(t-1)}})$$
$$\mathbf{h^{(t)}} = (1 - \mathbf{z^{(t)}}) \otimes \mathbf{h^{(t-1)}} + \mathbf{z^{(t)}} \otimes \mathbf{g^{(t)}}$$

GRU czy LSTM?

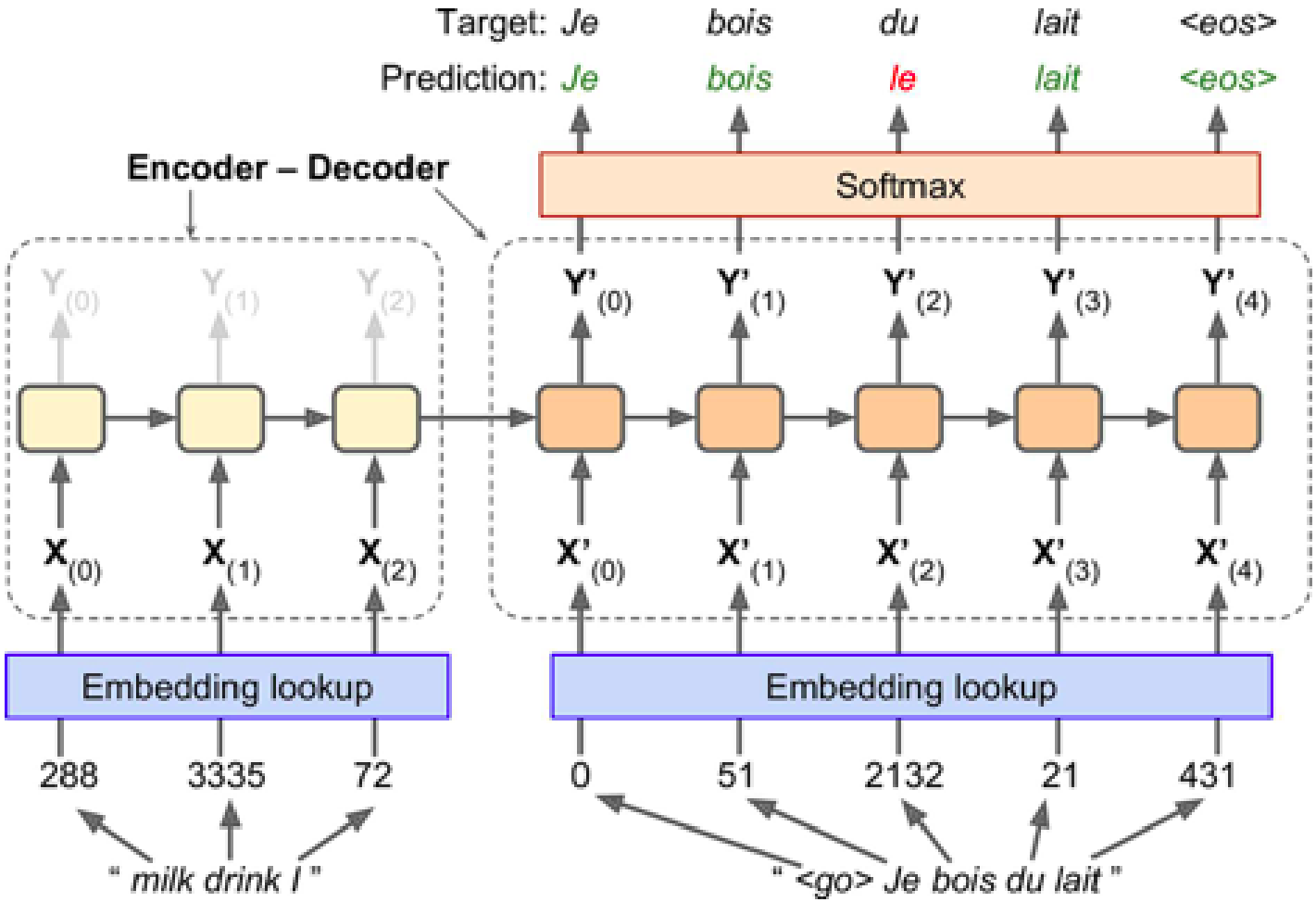
Dlaczego korzystamy z GRU, skoro mamy większą kontrolę nad siecią dzięki modelowi LSTM (ponieważ mamy w nim trzy bramki w przeciwieństwie do dwóch bramek w modelu GRU)?
W którym scenariuszu preferowana jest GRU niż LSTM?

GRU czy LSTM?

Dlaczego korzystamy z GRU, skoro mamy większą kontrolę nad siecią dzięki modelowi LSTM (ponieważ mamy w nim trzy bramki w przeciwieństwie do dwóch bramek w modelu GRU)?
W którym scenariuszu preferowana jest GRU niż LSTM?
Odp.

- GRU, przy mniejszej ilości danych trenujących w zadaniu modelowania języka trenuje się szybciej i osiąga lepsze wyniki niż LSTM
- GRU jest prostszym modelem, a zatem łatwiejszym do modyfikacji, np. dodając nowe bramki w przypadku dodatkowego wejścia do sieci
- LSTM powinny teoretycznie zapamiętywać dłuższe sekwencje niż GRU i przewyższać je w zadaniach wymagających modelowania relacji na duże odległości (np. relacji pomiędzy dwoma wyrazami, które znajdują się daleko od siebie w tekście)

Przykład sieci typu koder-dekoder do maszynowej translacji



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

Przykład sieci typu koder-dekoder do maszynowej translacji c.d.

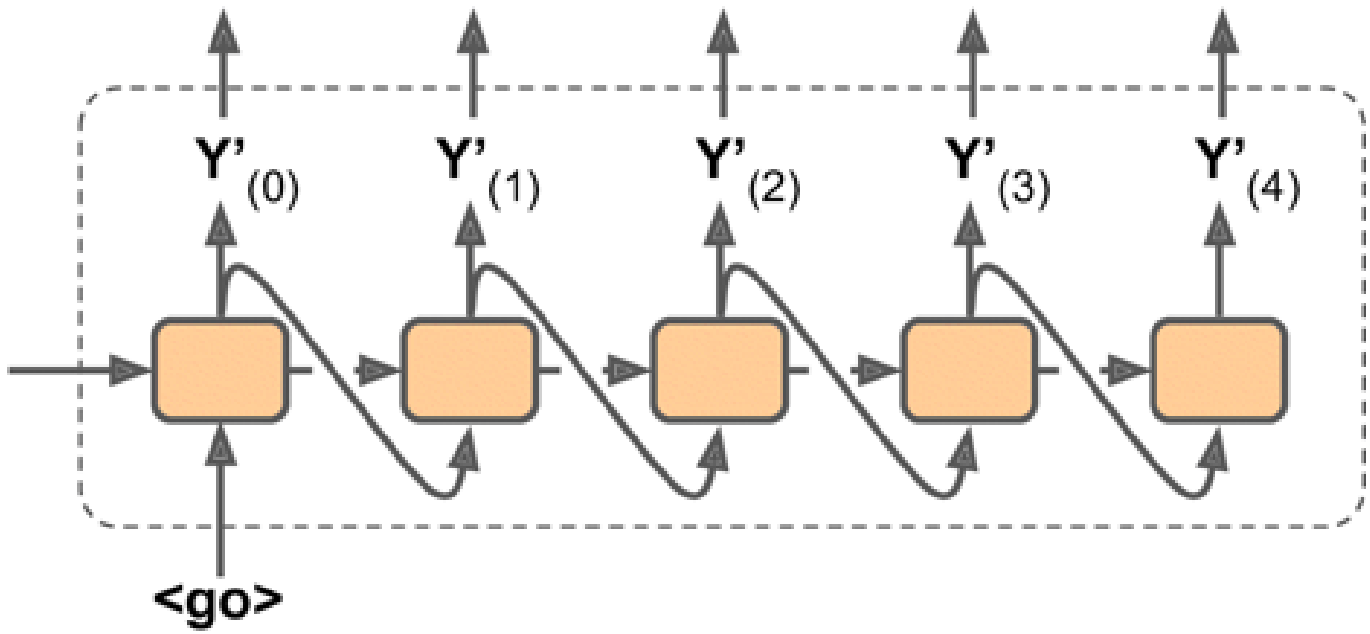
- angielskie zdania są podawane do kodera, a dekoder generuje francuskie tłumaczenia
- francuskie tłumaczenia są również używane jako wejścia do dekodera, ale cofnięte o jeden krok, tzn. dekoder otrzymuje jako dane wejściowe wyraz, który powinien mieć wyjście na poprzednim etapie
- jako pierwsze słowo otrzymuje token reprezentujący początek zdania (np. "<go>")
- oczekuje się, że dekoder zakończy zdanie za pomocą tokena końca sekwencji (EOS) (np. "<Eos>")
- angielskie zdania są odwrócone, zanim zostaną podane do kodera (np. "milk drink I." zamiast "I drink milk") (początek angielskiego zdania będzie podawany jako ostatni do kodera, co jest przydatne, bo jest to na ogół pierwsza rzecz, którą dekoder musi przetłumaczyć)

Przykład sieci typu koder-dekoder do maszynowej translacji c.d.

- każdy wyraz jest początkowo reprezentowany przez prosty identyfikator liczby całkowitej (np. 288 dla wyrazu "mleko").
- następnie wyszukiwanie oparte o *embeddingi* zwraca *embedding* wyrazu .
- *embeddingi* są tym, co faktycznie jest dostarczane do kodera i dekodera.
- na każdym etapie dekoder wyprowadza wynik dla każdego wyrazu w słowniku wyjściowym (tj. francuskim), a następnie warstwa Softmax zamienia te wyniki na prawdopodobieństwa (np. na pierwszym etapie słowo "Je" może mieć prawdopodobieństwo 20%).
- wyraz o najwyższym prawdopodobieństwie to wynik

Przykład sieci typu koder-dekoder do maszynowej translacji c.d.

W czasie wnioskowania (po trenowaniu) nie ma docelowego zdania do podania do dekodera.
Zamiast tego dostarcza się dekoderowi wyraz, które był poprzednio wyrazem wynikowym



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

Dziękuję za uwagę!