Rekurencyjne obliczanie semantyki fraz na podstawie informacji składniowej z użyciem głębokiego uczenia

Norbert Ryciak
Aleksander Wawer

21 października 2016
Seminarium ZIL, IPI PAN, Warszawa

Agenda

1. Metody reprezentacji słów w sieciach neuronowych (oraz RNN)
2. Wprowadzenie do rekurencyjnych sieci neuronowych
   - Czym są RNN i LSTM?
   - Najważniejsze implementacje działające na drzewach składniowych
3. Zbiór danych: Stanford Sentiment Treebank
4. Implementacja sieci powstała w ramach Clarin2
   - Przetwarzanie drzew składniowych, rekurencyjne obliczanie fraz
   - Szczypta technikaliów Theano
   - Wyniki eksperymentów
5. Polskojęzyczny zbiór zdań: pierwsze kroki

Reprezentacja słów w sieciach neuronowych

Słowu lub tokenowi odpowiada wektor liczb rzeczywistych długości N.

1. Wektory losowane, np. z rozkładu jednostajnego
2. Wektory “przewidywane” (prediction-based)
   - word2vec
3. Wektory “zliczane” (count-based)
   - Typowo: PMI + SVD
   - Stanford Glove
4. Dowolny z 1-3, dopasowany do problemu (fitted)

Reprezentacja słów w sieciach neuronowych: word2vec

- Wektory słów są warstwą płytkiej sieci neuronowej, trenowanej w zastosowaniu podobnym do modeli językowych.
- Mając dany kontekst, sieć próbuje przewidzieć słowo na określonej pozycji – i vice versa.
- Główny powód atrakcyjności: liniowe podobieństwo wektorów

vec(“king”) – vec(“man”) + vec(“woman”) =~ vec(“queen”)

Polskie modele: http://mozart.ipipan.waw.pl/~axw/models/

Distributed Representations of Words and Phrases and their Compositionality. Mikolov et al. 2013

Reprezentacja słów w sieciach neuronowych: glove

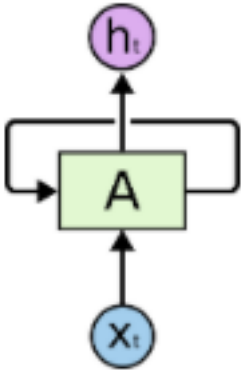
- Metoda korzysta z faktoryzacji ważonej macierzy współ-wystąpień słów
- Funkcja ważąca jest formą regresji
- Autorzy podają intuicyjne (?) uzasadnienie, dlaczego powinniśmy w ten sposób otrzymać pożądane, liniowe podobieństwa pomiędzy wektorami słów.

GloVe: Global Vectors for Word Representation. Pennington et al. 2014
http://nlp.stanford.edu/projects/glove/

RNN (Recurrent Neural Networks)

- Są to sieci neuronowe z pętlą, pozwalającą na przechowanie informacji.
- Sieć neuronowa A, wczytuje wejście x_t, produkuje wyjście h_t.
- Pętla pozwala przekazać informacje między poprzednim a następnym krokiem działania.

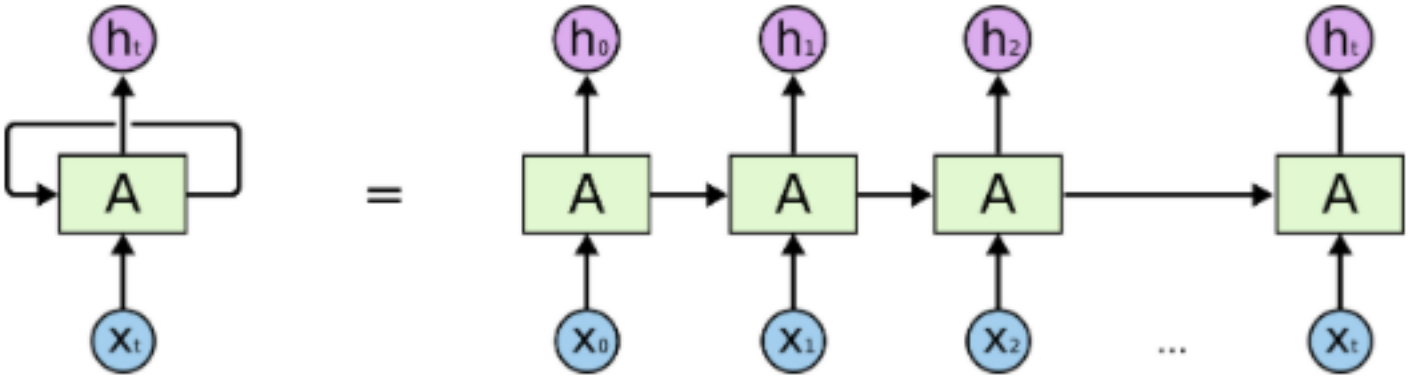
$$h_t = \tanh\left(Wx_t + Uh_{t-1} + b\right)$$



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

RNN (Recurrent Neural Networks)

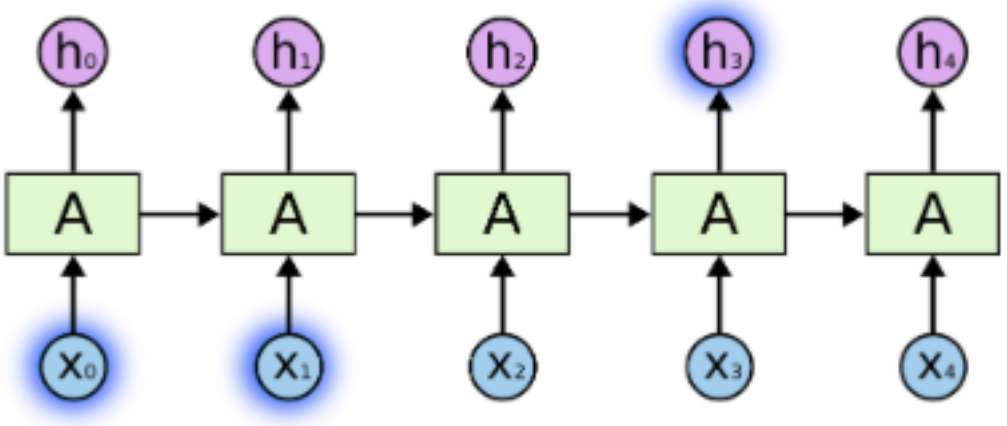
$$h_t = \tanh\left(W x_t + U h_{t-1} + b\right)$$



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

RNN

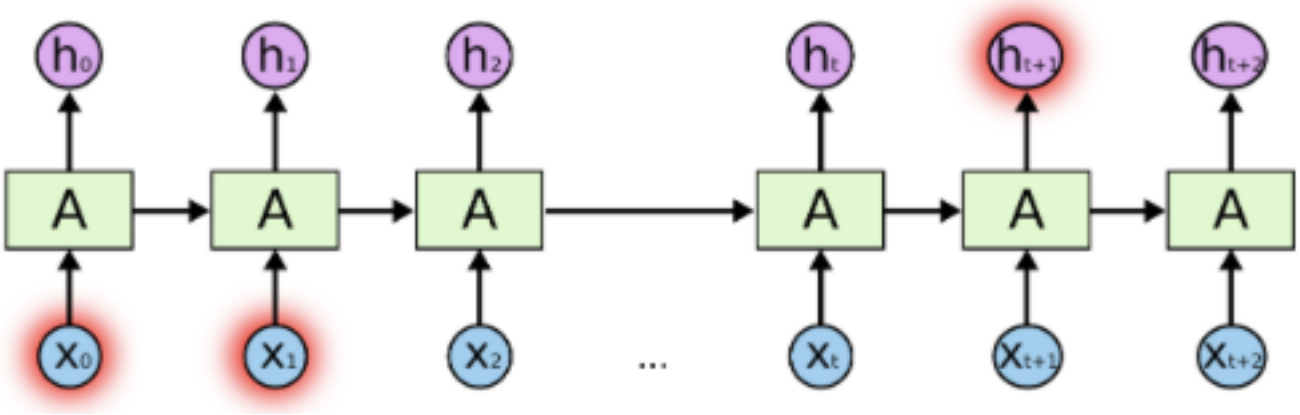
- W niektórych przypadkach, potrzebujemy tylko niedawno widzianych elementów sekwencji wejściowej.
- Przykładowo, model przewidujący następne słowo na podstawie poprzednich.
- Próbując przewidzieć następne słowo w “[the clouds are in the] *sky*” nie potrzebujemy dalszego kontekstu, następnym słowem musi być sky.
- W takich przypadkach sieci RNN są odpowiednią strukturą.



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

RNN

- Próba przewidzenia “[I grew up in France… I speak fluent] *French*” wymaga sięgnięcia wstecz dalej niż jedno zdanie.
- Ostatnie słowa sugerują tylko, że następne słowo prawdopodobnie jest nazwą języka.
- Odgadnięcie, że chodzi o francuski, wymaga odnalezienia France.
- Czasem dystans do relewantej informacji jest bardzo duży.
- W miarę wzrostu tego dystansu, sieci RNN stają się niezdolne wykorzystania relewantnych informacji - Hochreiter (1991) i Bengio, et al. (1994) opisują kilka przyczyn.

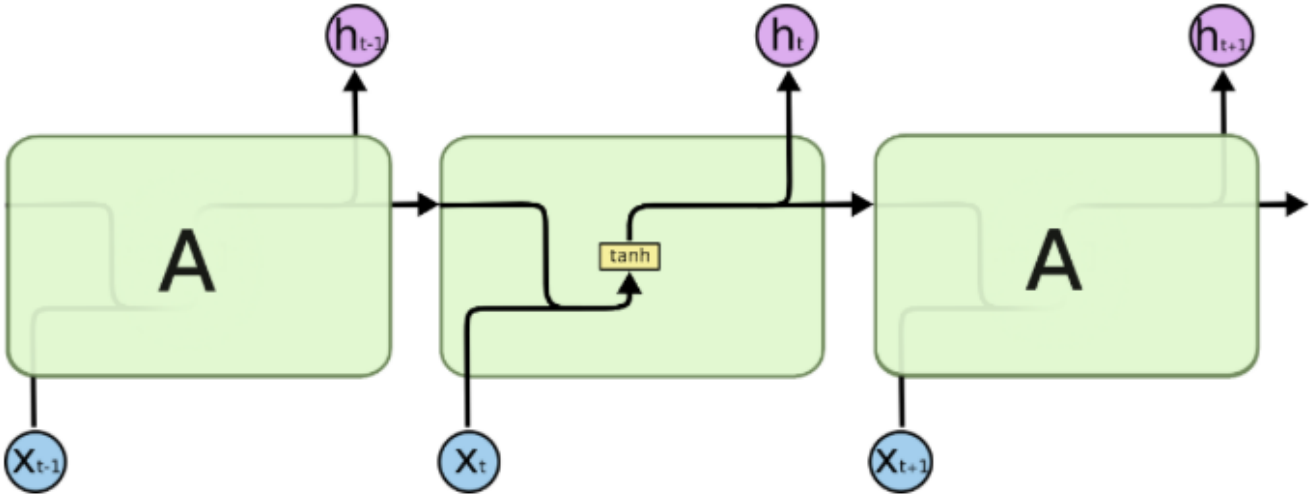


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

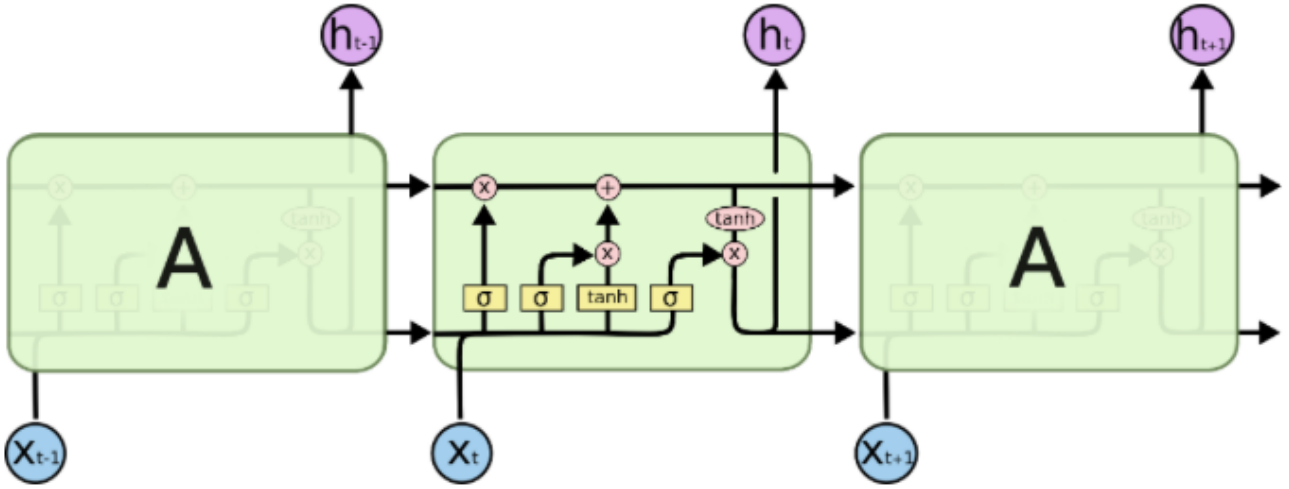
LSTM

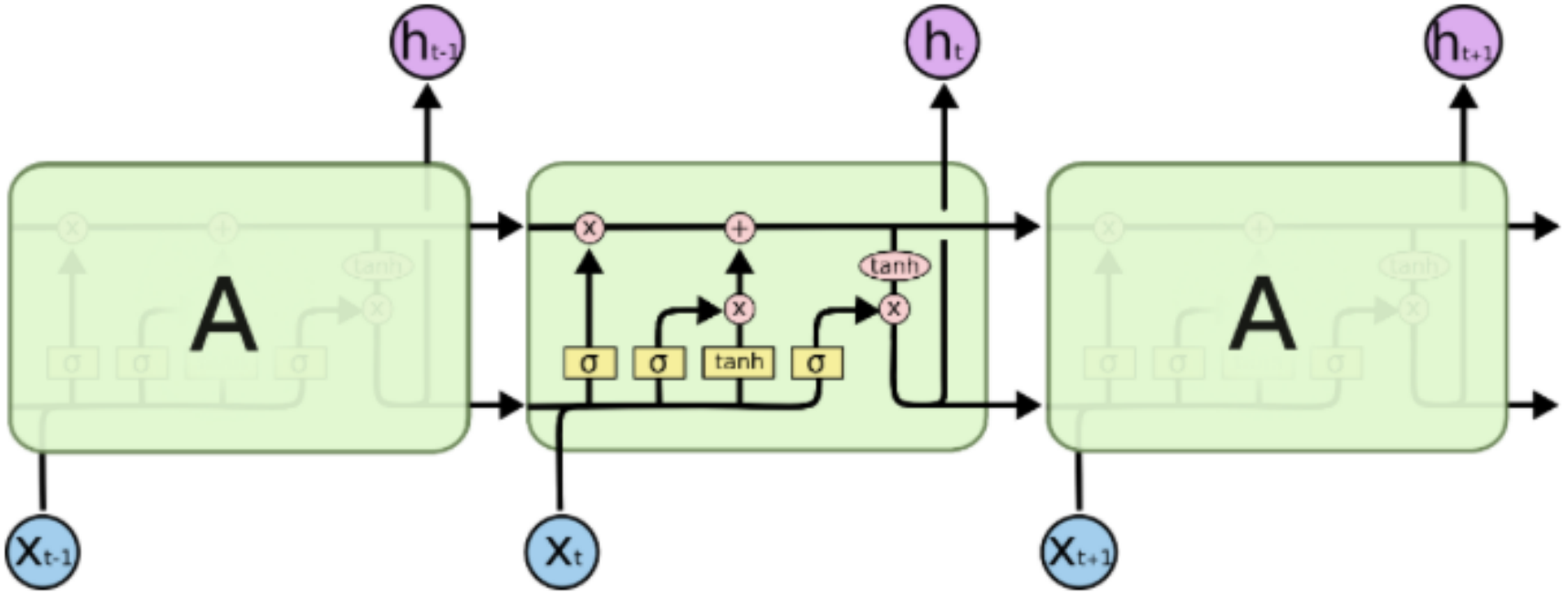
- Sieci Long Short Term Memory – zazwyczaj krótko “LSTM” – są szczególnym rodzajem sieci RNN, zdolnym do nauczenia się długodystansowych zależności. Hochreiter & Schmidhuber (1997)
- LSTMs zostały zaprojektowane w celu zaadresowania problemów z długodystansowymi zależnościami, zidentyfikowanymi w sieciach RNN.

RNN


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}



Neural Network Layer

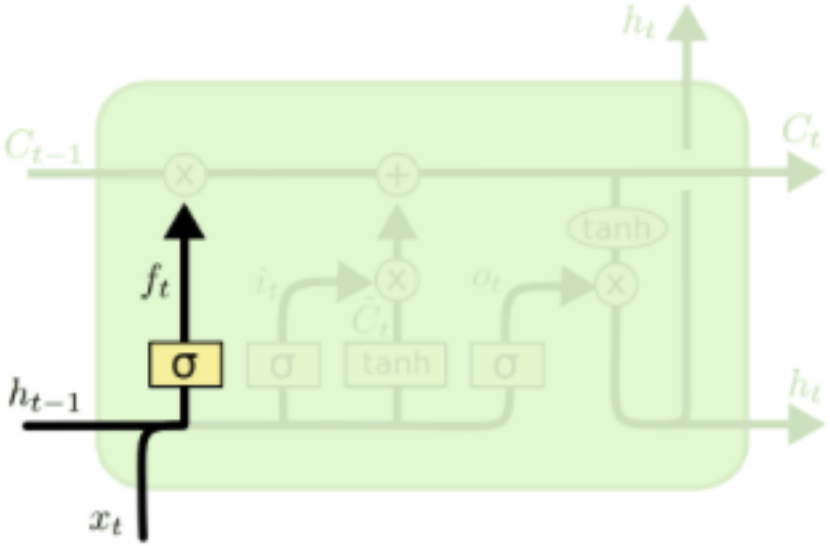
Pointwise Operation

Vector Transfer

Concatenate

Copy

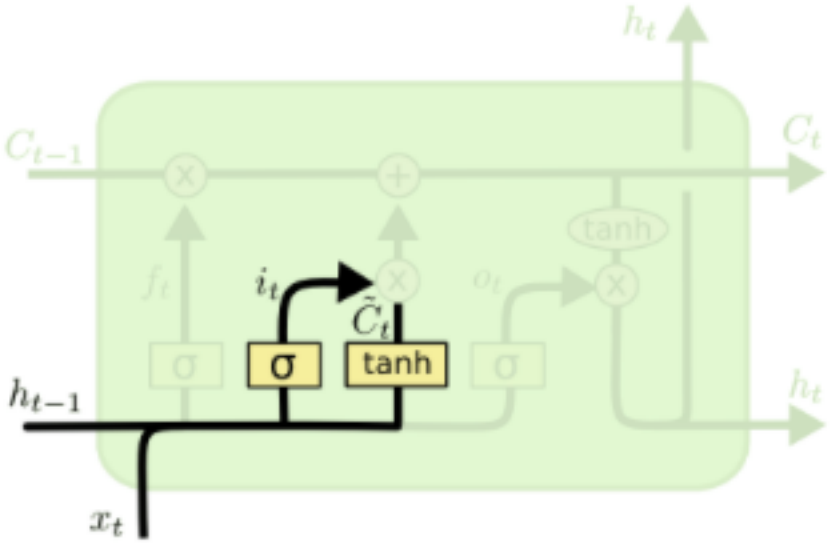
LSTM



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

- Decydujemy, których informacji mamy się pozbyć ze stanu komórki. Decyzję tą podejmuje sigmoidowa 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

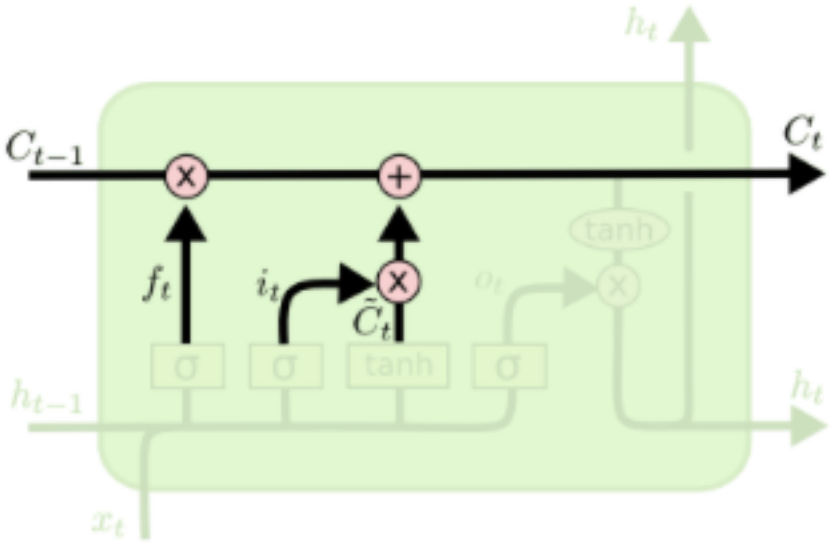


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

- W następnym kroku decydujemy, jaką nową informację umieścić w stanie komórki (cell state). Ma to dwie części.
- W pierwszym kroku, warstwa sigmoidowa “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.

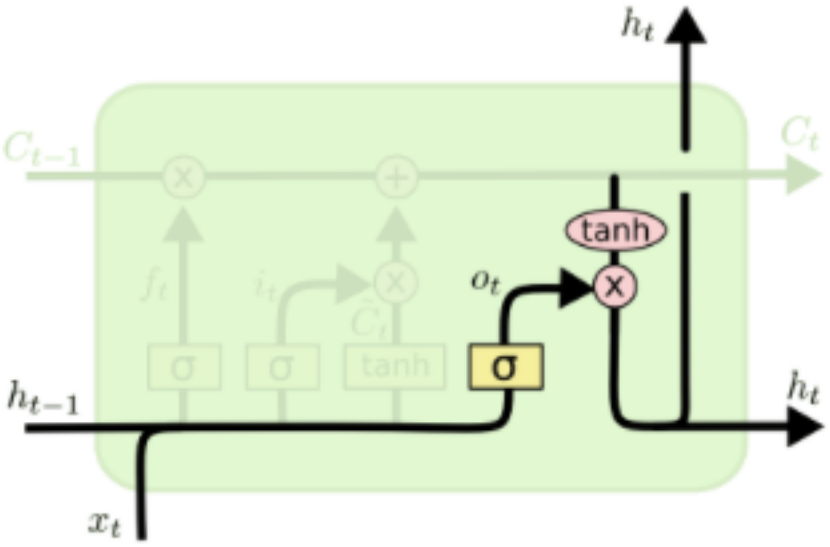
LSTM



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

- 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



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

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

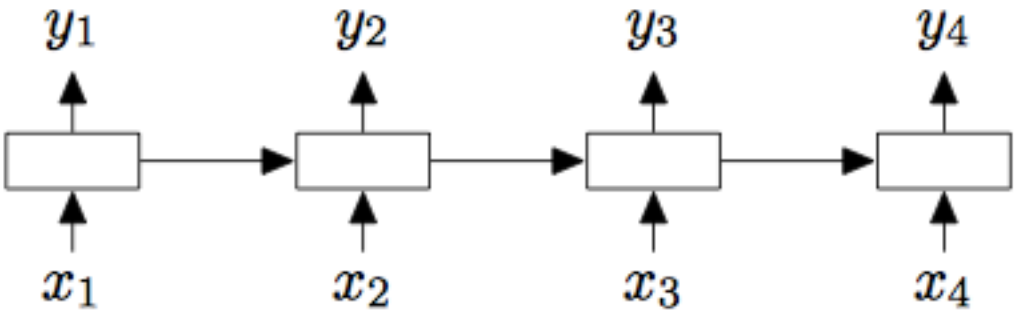
- Na koniec, decydujemy co zamierzamy zwrócić. Wyjście LSTM jest filtrowaną wersją stanu komórkowego (cell state).
- W pierwszym kroku uruchamiamy warstwę sigmoidową, która decyduje, którą część stanu komórkowego zwracamy.
- Następnie, mnożymy stan komórkowy przez tanh (otrzymując zakres -1 do 1) i to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

Tree LSTM

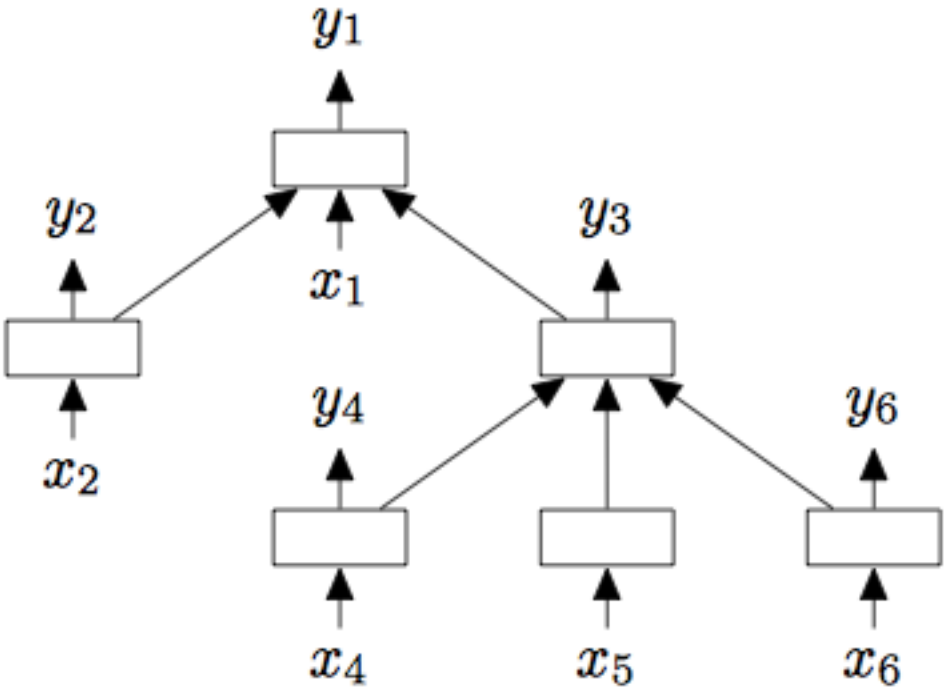
- Oryginalna wersja LSTM pozwala na przewidywanie danych sekwencyjnych, podobnie jak RNN.
- Istnieją jednak generalizacje sieci LSTM do drzewiastych struktur wejściowych.

Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. Kai Sheng Tai, Richard Socher*, Christopher D. Manning. ACL 2015

Tree LSTM



Sekwencyjne LSTM tworzą reprezentację zdań biorąc pod uwagę kolejność tokenów



Drzewiaste LSTM obliczają znaczenie każdej frazy i zdania z konstytuujących je pod-fraz, zgodnie z daną strukturą składniową

Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. Kai Sheng Tai, Richard Socher*, Christopher D. Manning. ACL 2015

Tree LSTM

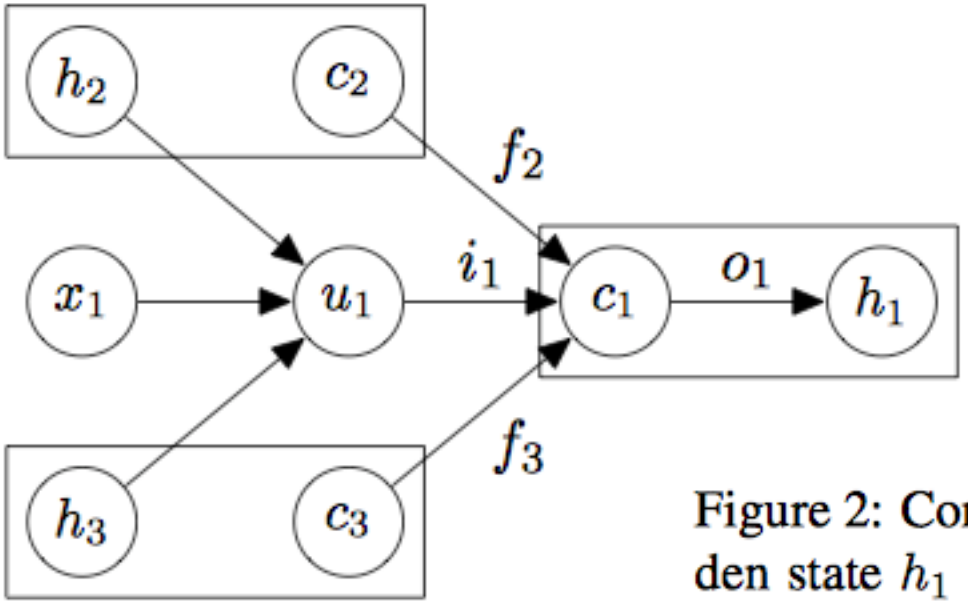


Figure 2: Composing the memory cell c_1 and hidden state h_1 of a Tree-LSTM unit with two children (subscripts 2 and 3). Labeled edges correspond to gating by the indicated gating vector, with dependencies omitted for compactness.

Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. Kai Sheng Tai, Richard Socher*, Christopher D. Manning. ACL 2015

Child-Sum Tree-LSTMs

- W drzewiastych LSTM, bramki i uaktualnienia pamięci komórek zależą od stanów potencjalnie wielu komórek-dzieci.
- Wariant Child-Sum Tree-LSTM agreguje stany ukryte dzieci
- Jest odpowiedni dla drzew z nieuporządkowanymi dziećmi oraz zmienną liczbą dzieci
- Jest odpowiedni dla drzew zależnościowych, gdzie liczba podrzędników może być zmienna.

Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. Kai Sheng Tai, Richard Socher*, Christopher D. Manning. ACL 2015

LSTM

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

Child-Sum Tree LSTMs

$$
\begin{aligned}
\tilde{h}_j &= \sum_{k \in C(j)} h_k, \\
i_j &= \sigma\left(W^{(i)}x_j + U^{(i)}\tilde{h}_j + b^{(i)}\right), \\
f_{jk} &= \sigma\left(W^{(f)}x_j + U^{(f)}h_k + b^{(f)}\right), \\
o_j &= \sigma\left(W^{(o)}x_j + U^{(o)}\tilde{h}_j + b^{(o)}\right), \\
u_j &= \tanh\left(W^{(u)}x_j + U^{(u)}\tilde{h}_j + b^{(u)}\right), \\
c_j &= i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \\
h_j &= o_j \odot \tanh(c_j),
\end{aligned}
$$

Wyniki Tree LSTM vs inne prace

Method	Fine-grained	Binary
RAE (Socher et al., 2013)	43.2	82.4
MV-RNN (Socher et al., 2013)	44.4	82.9
RNTN (Socher et al., 2013)	45.7	85.4
DCNN (Blunsom et al., 2014)	48.5	86.8
Paragraph-Vec (Le and Mikolov, 2014)	48.7	87.8
CNN-non-static (Kim, 2014)	48.0	87.2
CNN-multichannel (Kim, 2014)	47.4	**88.1**
DRNN (Irsoy and Cardie, 2014)	49.8	86.6
LSTM	46.4 (1.1)	84.9 (0.6)
Bidirectional LSTM	49.1 (1.0)	87.5 (0.5)
2-layer LSTM	46.0 (1.3)	86.3 (0.6)
2-layer Bidirectional LSTM	48.5 (1.0)	87.2 (1.0)
Dependency Tree-LSTM	48.4 (0.4)	85.7 (0.4)
Constituency Tree-LSTM		
– randomly initialized vectors	43.9 (0.6)	82.0 (0.5)
– Glove vectors, fixed	49.7 (0.4)	87.5 (0.8)
– Glove vectors, tuned	**51.0** (0.5)	88.0 (0.3)

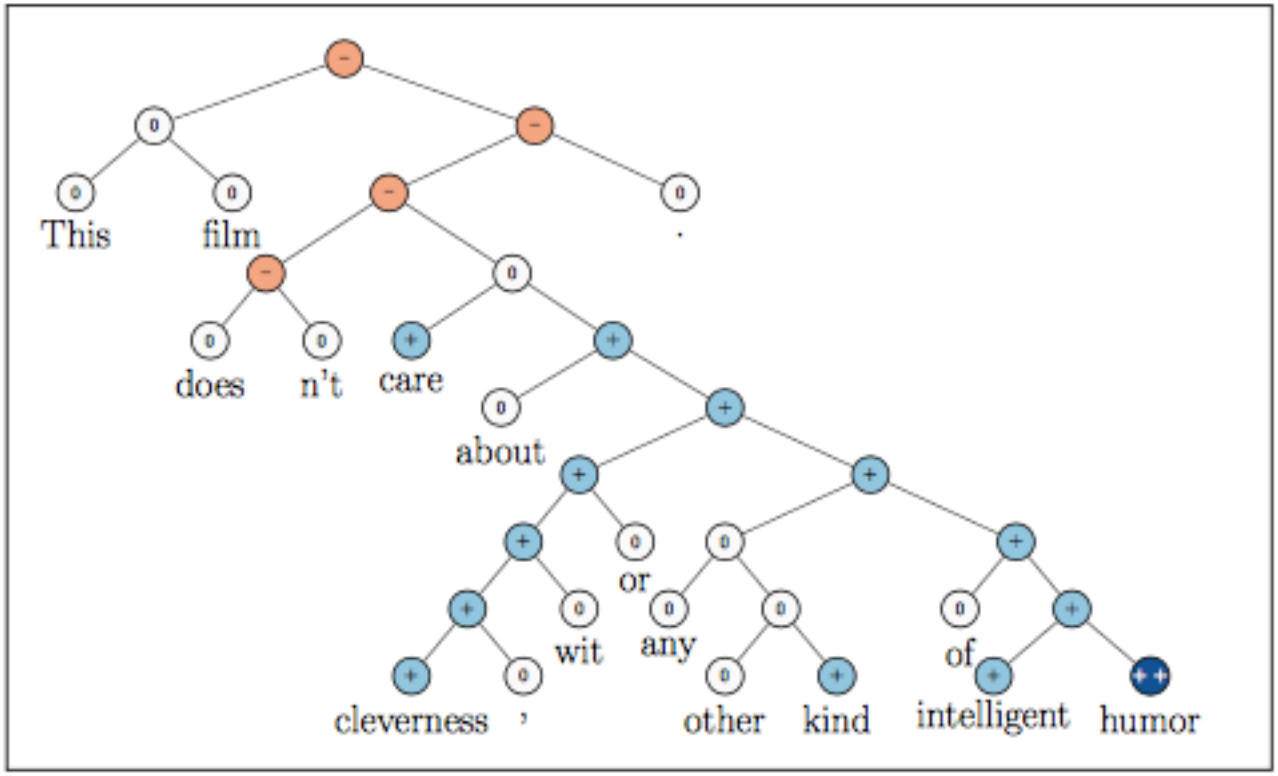
Stanford Sentiment Treebank

- Korpus zdań – wycinków recenzji filmowych z rottentomatoes.com pierwotnie opisany w Pang and Lee (2005)
- 10,662 zdań przetworzonych stanfordzkim parserem składnikowym (Klein and Manning, 2003).
- Anotacja crowdsourcingowa: platforma Amazon Mechanical Turk.
- 215,154 oznaczonych ręcznie fraz

Stanford Sentiment Treebank

Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. Socher et al. 2013

Stanford Sentiment Treebank



Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. Socher et al. 2013

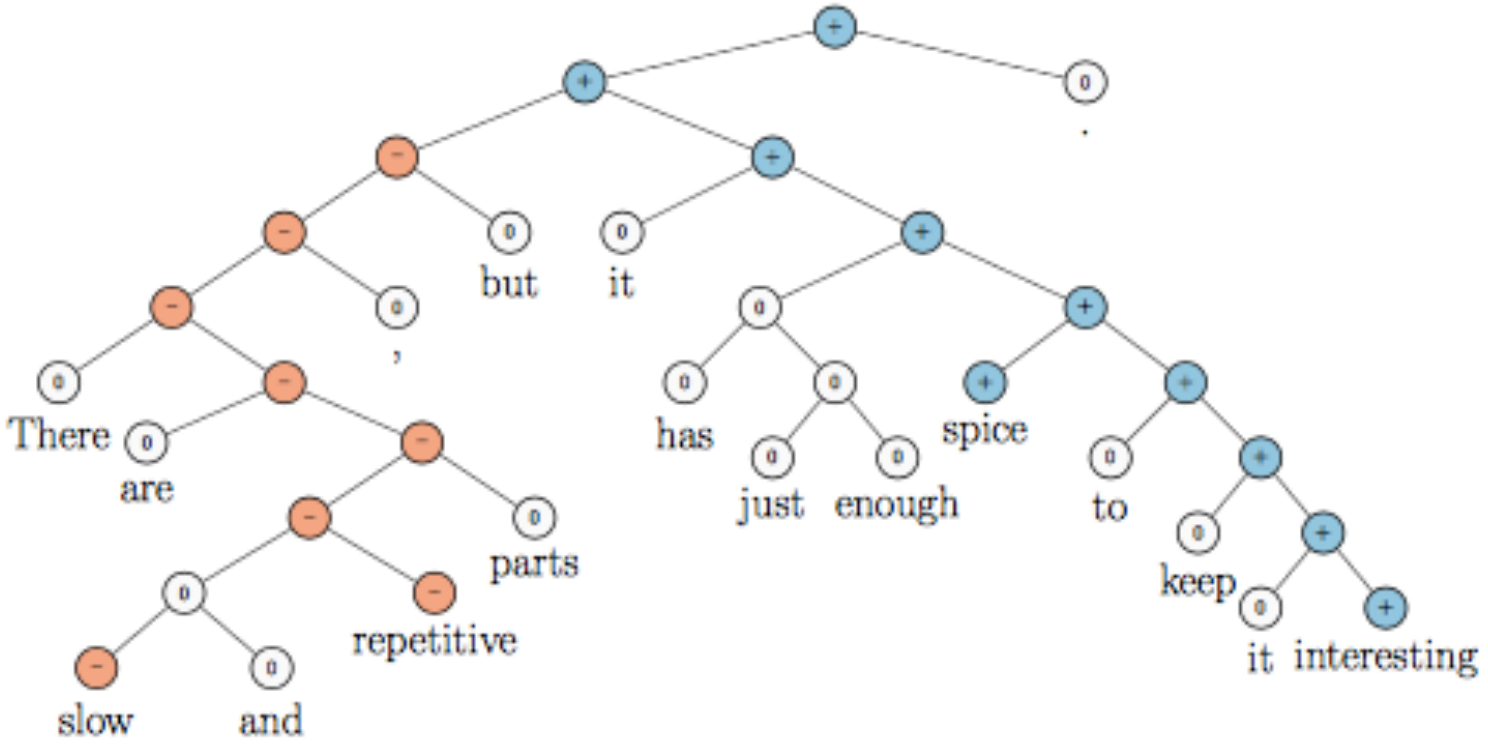
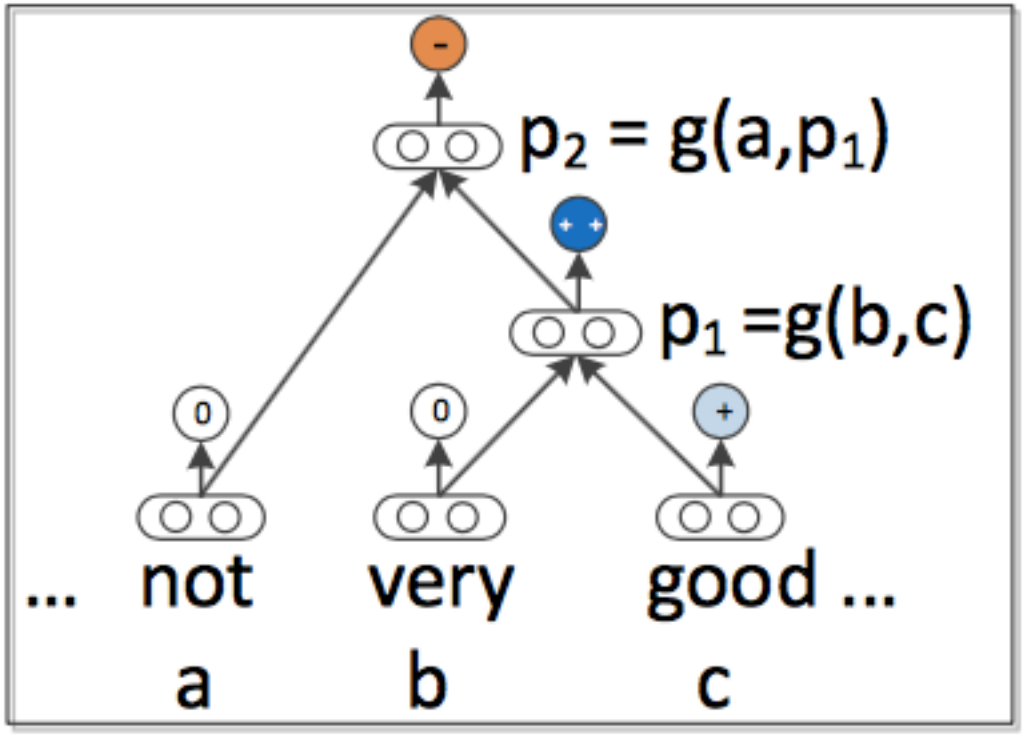


Figure 7: Example of correct prediction for contrastive conjunction *X but Y*.

Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. Socher et al. 2013

Rekursywne sieci neuronowe



Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank.
Socher et al. 2013

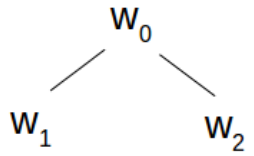
Implementacja rekurencyjnej sieci w Clarin 2

Cele tworzonego narzędzia:

- Integracja z zależnościowym, a nie składnikowym i binarnym rozbiorem zdaniowym
- Narzędzie w języku python (np. Theano)
  - Tree LSTM niestety w Torch/Lua
- Zbadanie zależności między rozmiarem danych uczących a wynikami

Struktura sieci

Klasyfikujemy frazę “$w_1\ w_0\ w_2$” o strukturze:

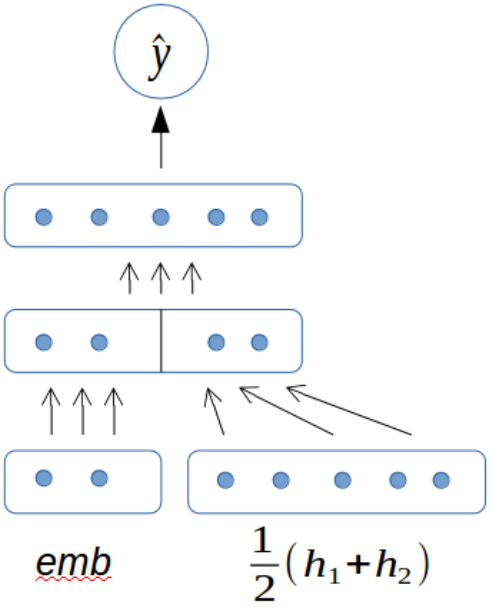


Wyjście: $\hat{y} = softmax(W_y \cdot h_d)$

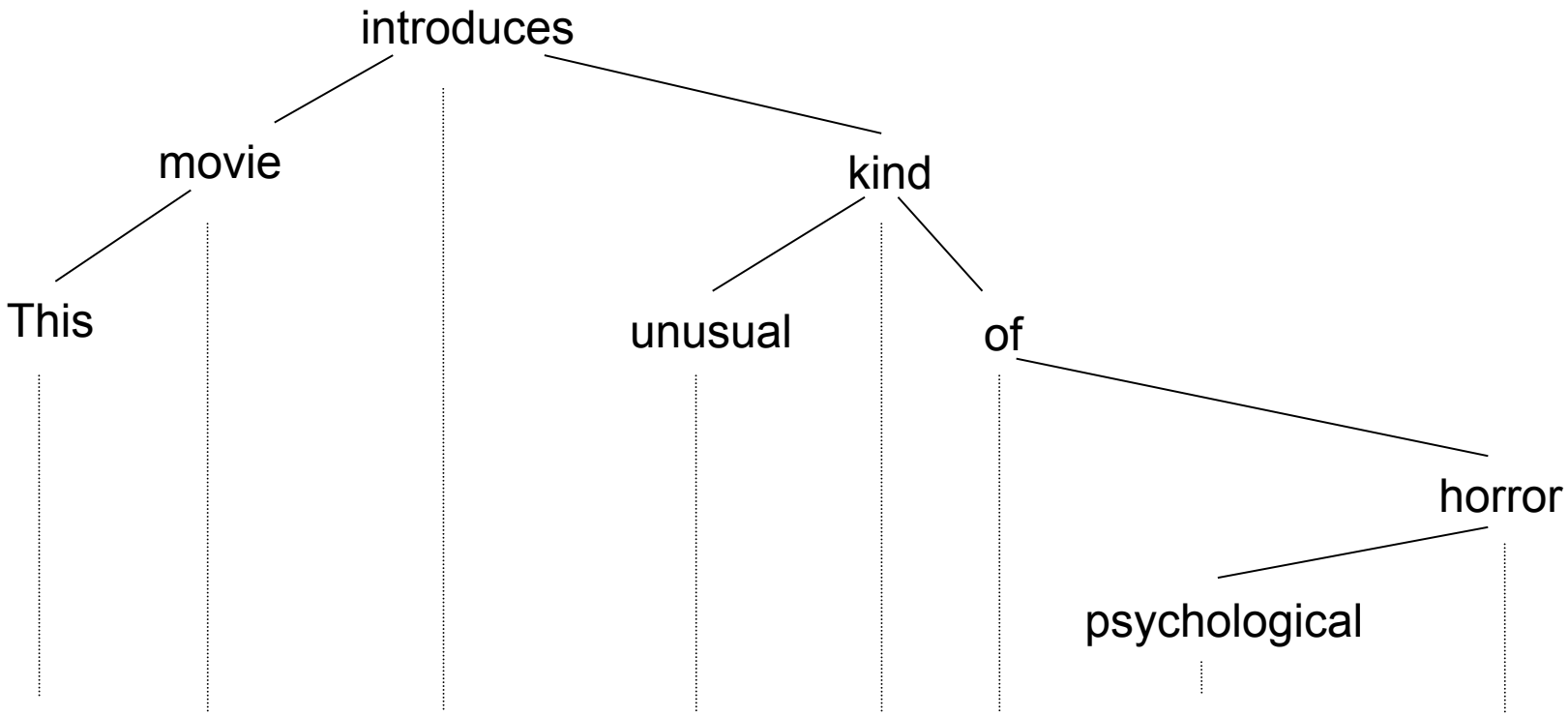
Warstwa *dense*: $h_d = W_d \cdot h_0$

Stan ukryty: $h_0 = [W_e \cdot embedding(w_0); W_h \cdot \frac{1}{2} \sum_{i=1,2} h_i]$

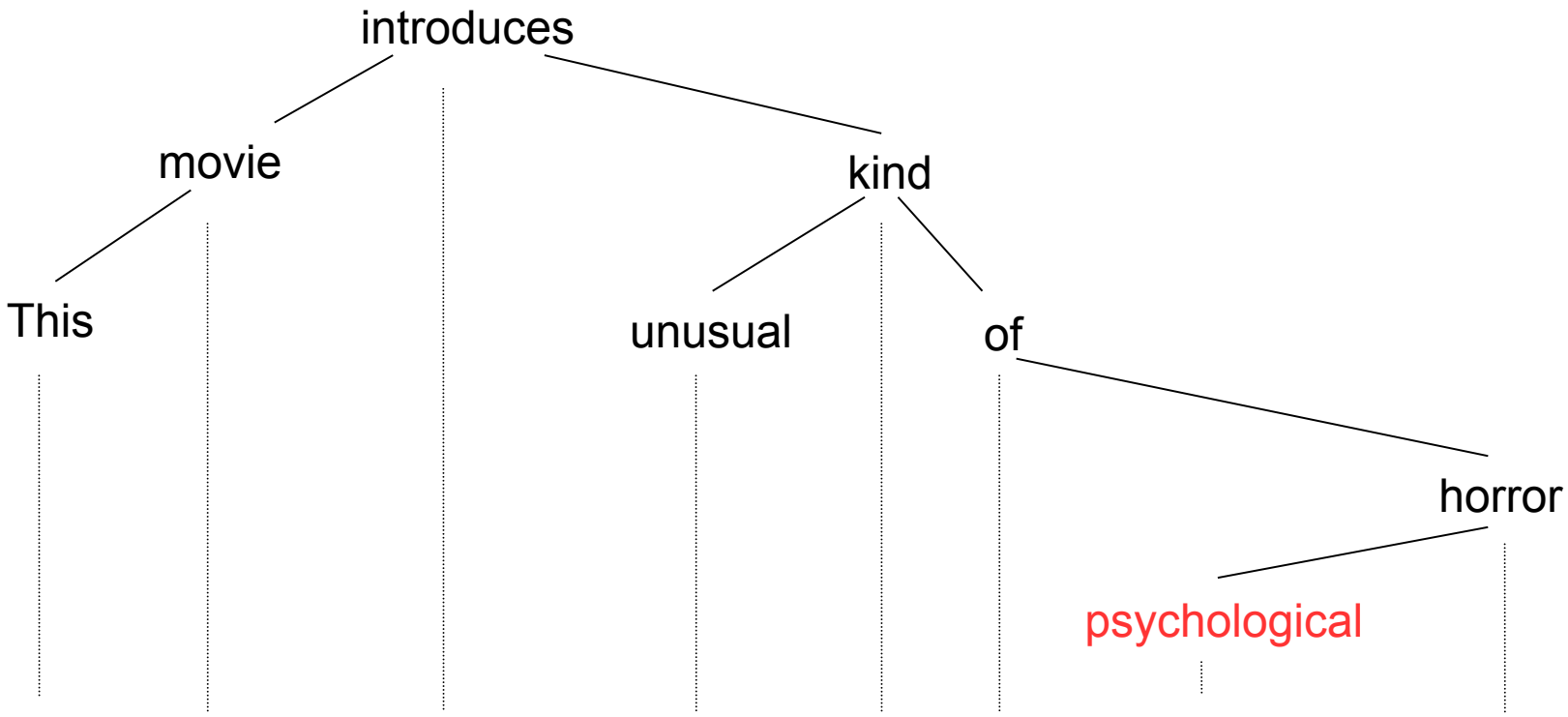
Wejście: $embedding(w_0), h_1, h_2$



Przechodzenie po drzewie

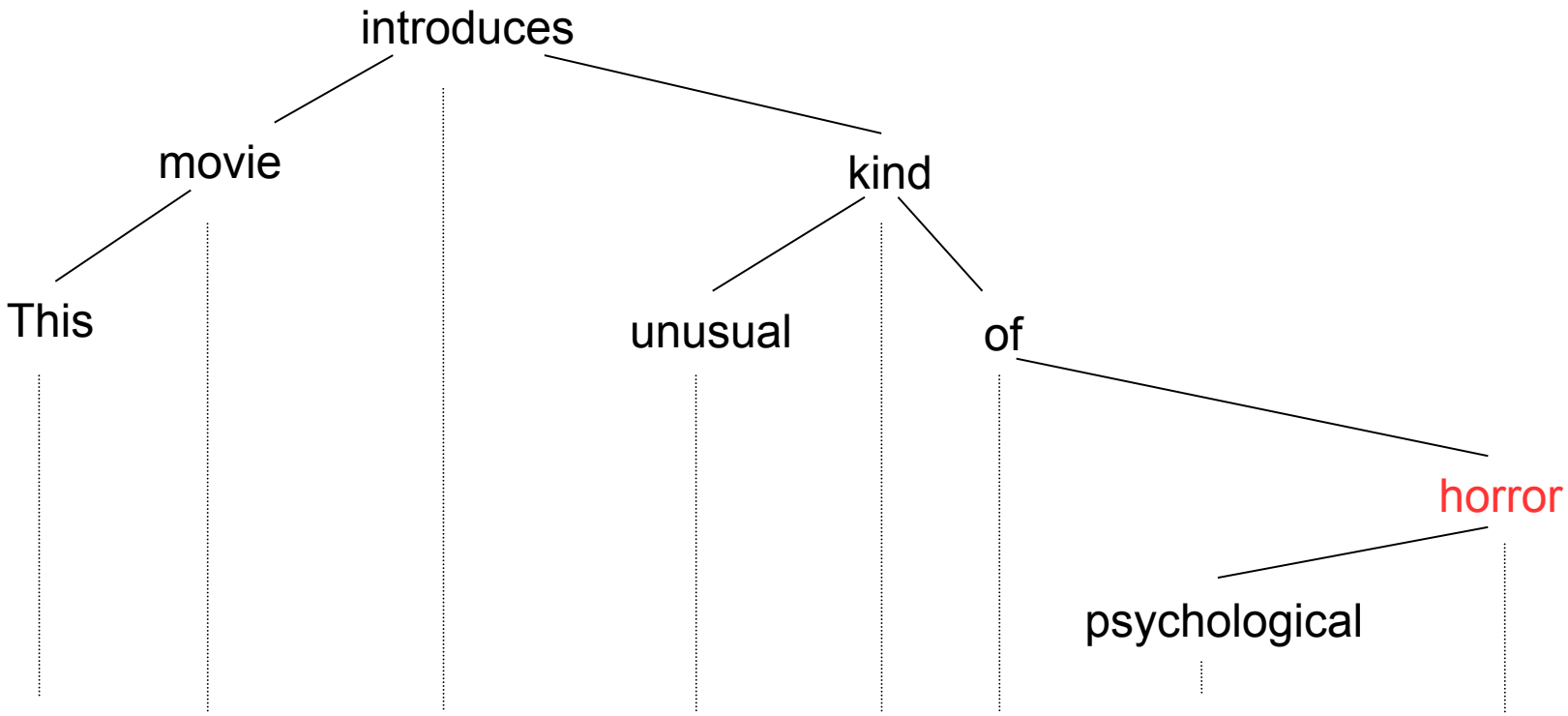


Przechodzenie po drzewie



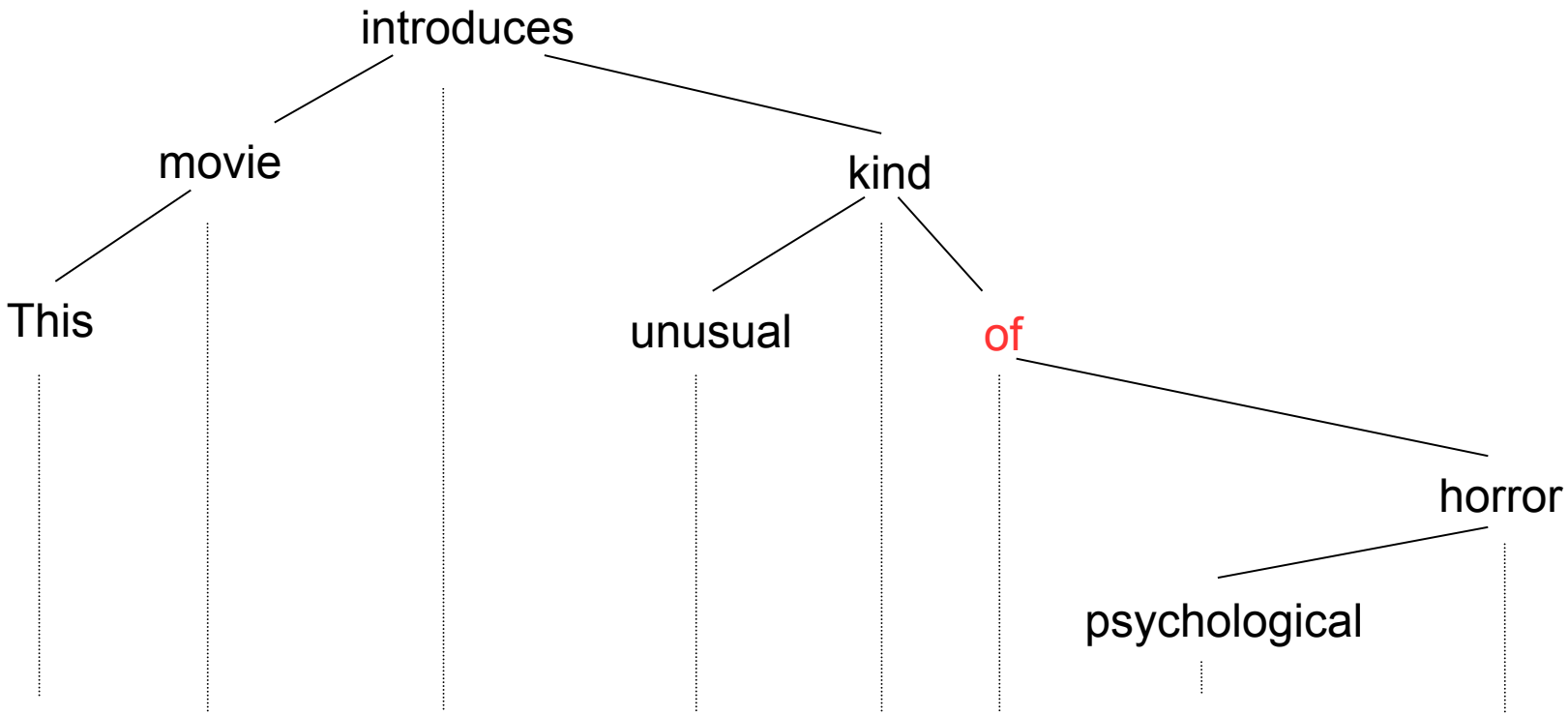
Kolejność odwiedzania węzłów:
psychological

Przechodzenie po drzewie



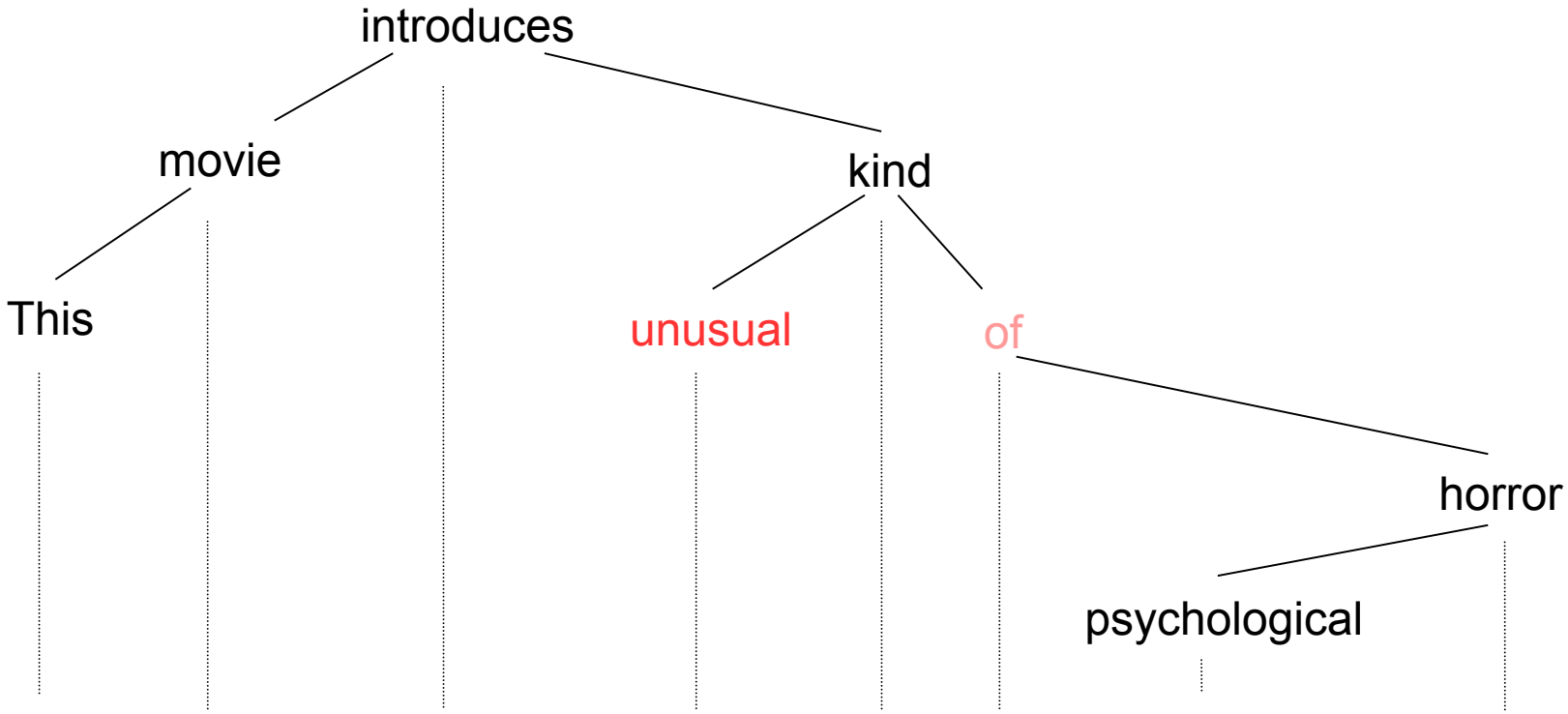
Kolejność odwiedzania węzłów:
psychological, horror

Przechodzenie po drzewie



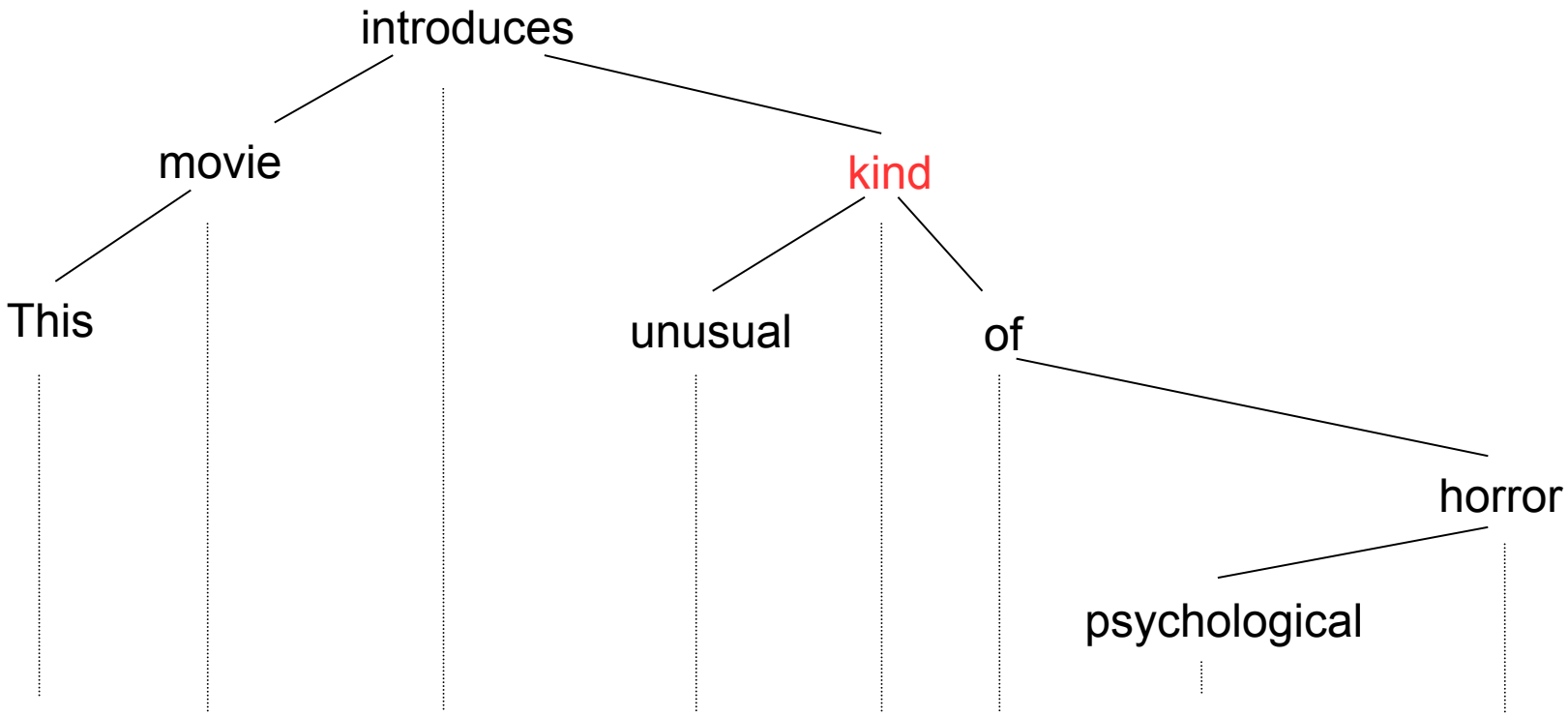
Kolejność odwiedzania węzłów:
psychological, horror, of

Przechodzenie po drzewie



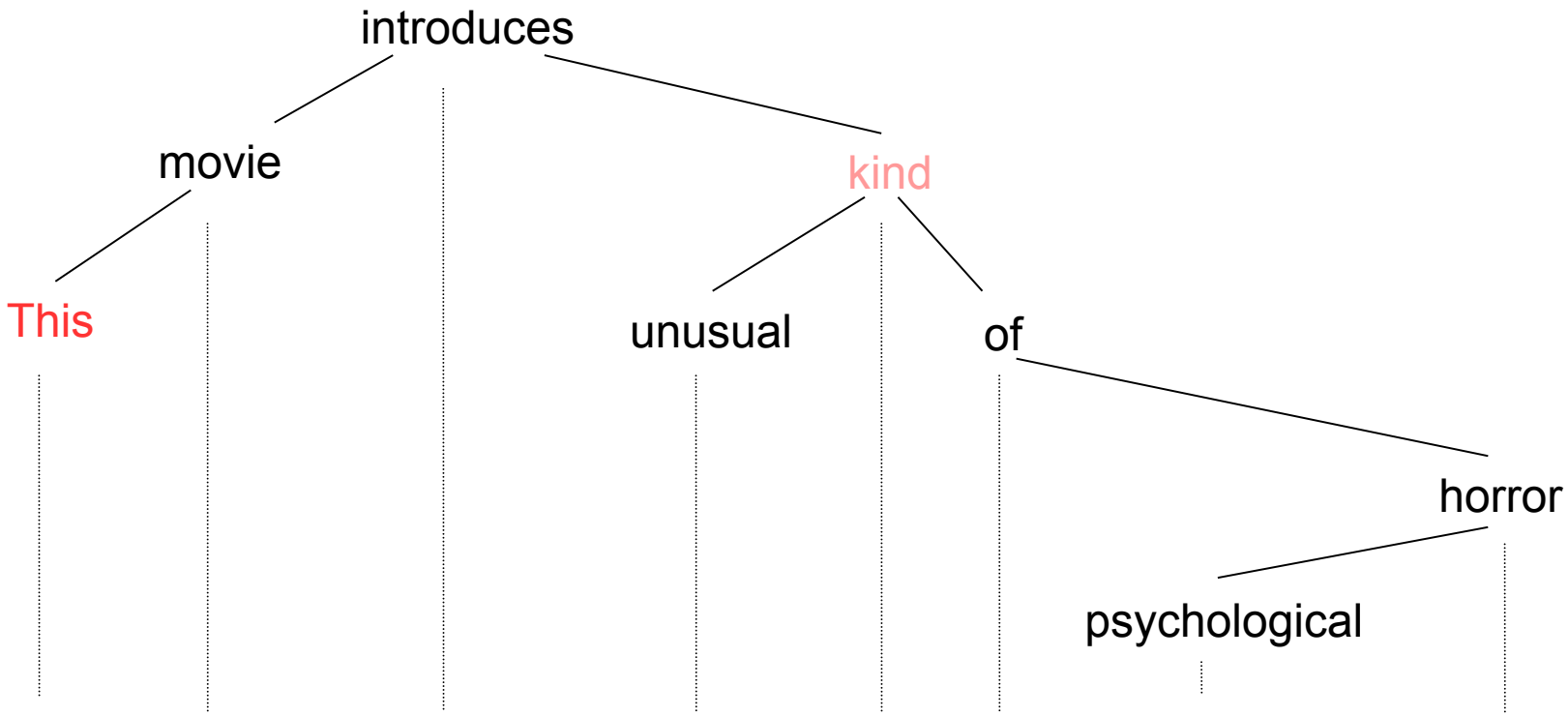
Kolejność odwiedzania węzłów:
psychological, horror, of, unusual

Przechodzenie po drzewie



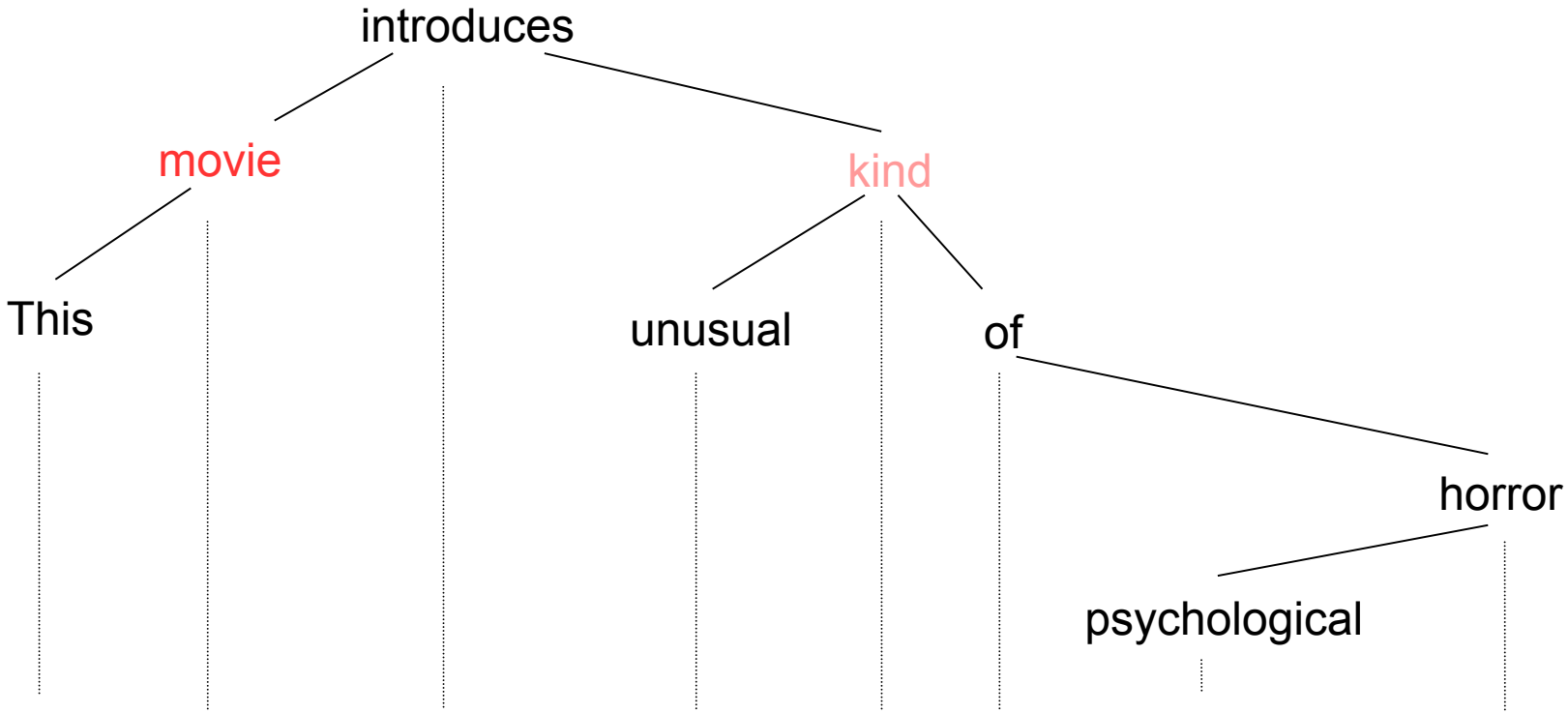
Kolejność odwiedzania węzłów:
psychological, horror, of, unusual, kind

Przechodzenie po drzewie



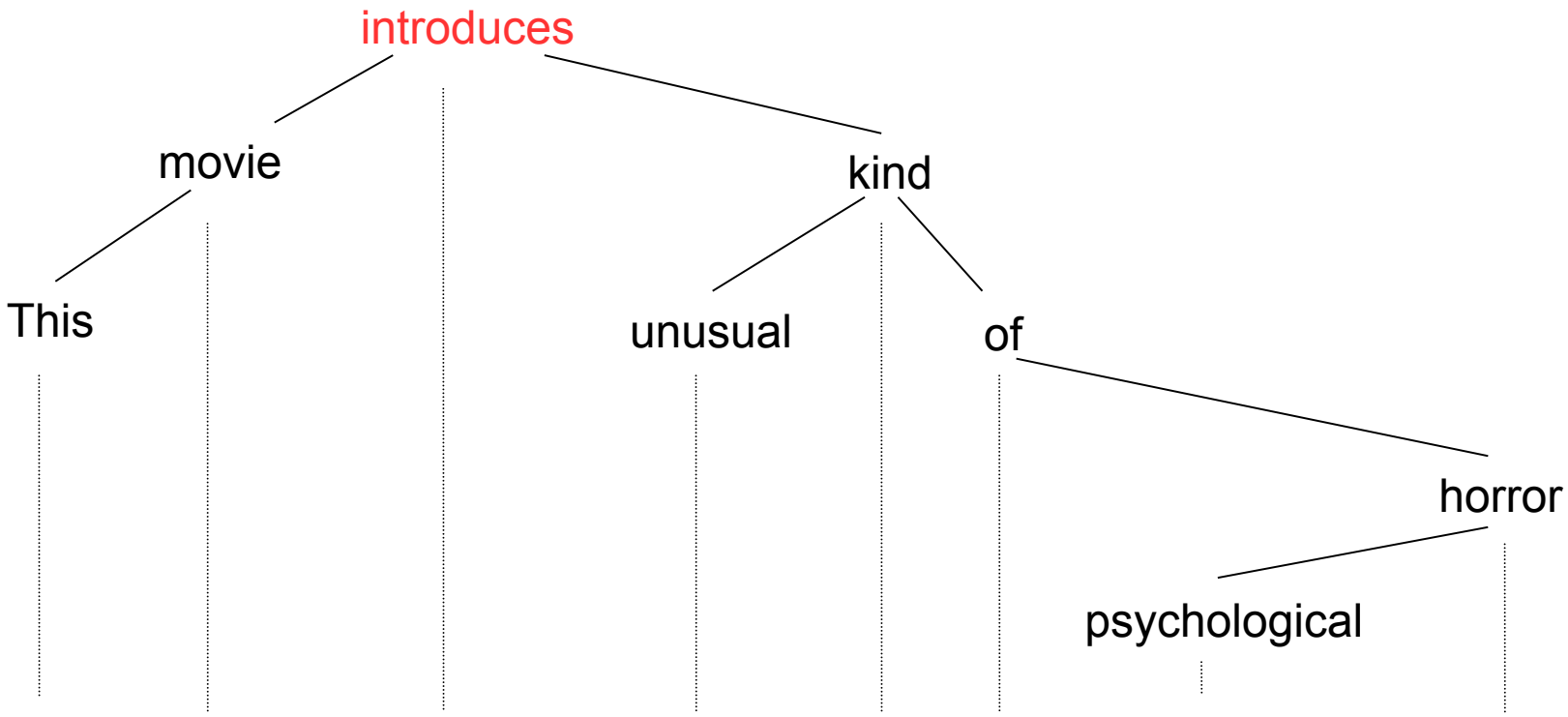
Kolejność odwiedzania węzłów:
psychological, horror, of, unusual, kind, This

Przechodzenie po drzewie



Kolejność odwiedzania węzłów:
psychological, horror, of, unusual, kind, This, movie

Przechodzenie po drzewie



Kolejność odwiedzania węzłów:
psychological, horror, of, unusual, kind, This, movie, introduces

Implementacja

```
def one_step(word_id, children_ids, children_idx, y_true, i, hidden_states, learning_rate):

    '''Funkcja wywoływana w węźle.'''

    idx_tmp = (children_idx>=0).nonzero()
    tmp = T.zeros_like(children_idx)
    tmp2 = T.set_subtensor(tmp[idx_tmp], 1)
    number_of_children = T.eq(tmp2.sum(dtype = dataType)) # liczba dzieci danego węzła
    number_of_children = ifelse(T.eq(number_of_children, shared_zero), shared_one, number_of_children)

    schh = hidden_states[children_idx].sum(axis=0) / number_of_children # średnia stanow ukrytych dzieci

    h = T.concatenate([T.dot(self.emb[word_id],self.W_e_h), T.dot(schh,self.W_sh_h)]) # stan ukryty

    zeros_subtensor = hidden_states[i]
    hidden_states_new = T.set_subtensor(zeros_subtensor, h) # zapisujemy stan

    h2 = T.dot(h, self.W_h_h2) # warstwa dense

    y_prob = T.nnet.softmax(T.dot(h2,self.W_h2_y)) # predykcja

    ce = -T.log(y_prob[0][y_true]) # kara (crossentrony)

    updates = OrderedDict([(self.W_h2_y, self.W_h2_y-learning_rate*T.grad(ce, self.W_h2_y)),
                (self.W_h_h2, self.W_h_h2-learning_rate*T.grad(ce, self.W_h_h2)),
                (self.W_e_h, self.W_e_h-learning_rate*T.grad(ce, self.W_e_h)),
                (self.W_sh_h, self.W_sh_h-learning_rate*T.grad(ce, self.W_sh_h)),
                (self.emb, self.emb-learning_rate*T.grad(ce, self.emb))
                ]) # nowe wartości wag

    return (i+1,hidden_states_new, y_prob), updates
```

Implementacja

```
y_probs, updates = theano.scan(
                          fn=one_step,
                          sequences = [words, children_ids, children_positions,y],
                          outputs_info = [theano.shared(0),
                                          theano.shared(np.zeros((self.max_phrase_length,2*nh))),
                                          None],
                          non_sequences = lr,
                          n_steps = words.shape[0]
                          )
```

```
self.train = theano.function( inputs  = [words,children_ids, children_positions, y, lr],
                              outputs = []
                              updates = updates,
                              allow_input_downcast=True,
                              mode='FAST_RUN' )
```

```
self.classify = theano.function(
                             inputs=[words,children_ids,children_positions], outputs=predictions[1],
                             allow_input_downcast=True,
                             mode='FAST_RUN' )
```

Rekursywne sieci neuronowe

Model	Fine-grained		Positive/Negative	
	All	Root	All	Root
NB	67.2	41.0	82.6	81.8
SVM	64.3	40.7	84.6	79.4
BiNB	71.0	41.9	82.7	83.1
VecAvg	73.3	32.7	85.1	80.1
RNN	79.0	43.2	86.1	82.4
MV-RNN	78.7	44.4	86.8	82.9
RNTN	**80.7**	**45.7**	**87.6**	**85.4**

Table 1: Accuracy for fine grained (5-class) and binary predictions at the sentence level (root) and for all nodes.

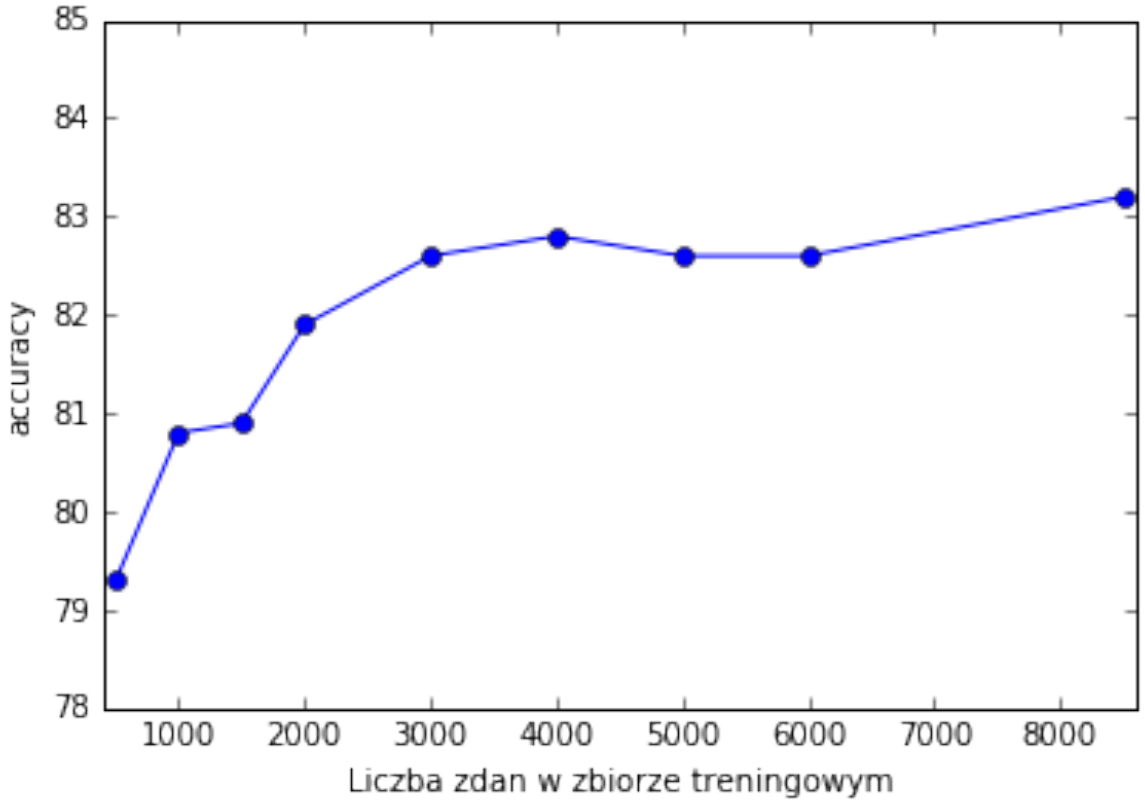
Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. Socher et al. 2013

Uzyskane rezultaty

Baseline dla *all*: 78%
Baseline dla *root*: 28% dla fine-grained i 41% dla pos/neu/neg.

Model	Fine- grained		Pos / neu / neg	
	all	root	all	root
Struktura bazowa (przedstawiana)	83.2	32.6	85.0	45.8
Struktura uproszczona (bez warstwy *dense*)	82.7	30.6	85.2	51.3

Zależność jakości predykcji od wielkości zbioru uczącego



Predykcja przy użyciu struktury bazowej dla 5. klas, liczona dla wszystkich fraz.

Tworzone polskojęzyczne zasoby (1)

- Polskojęzyczny treebank sentymentowy (w ramach Clarin 2)
- Ze Składnicy zależnościowej (wersja mid-2013) wybraliśmy 915 zdań
  - Sposób wyboru: w zdaniu wystąpiło co najmniej jedno słowo z wydźwiękiem według dwóch dostępnych słowników
- Zdania poddane ręcznej anotacji fraz przez lingwistów
  - Z reguły, usunięty został nadmiarowy sentyment
  - Skutek: po pierwszym etapie anotacji pozostało 30% zdań

Tworzone polskojęzyczne zasoby (2)

- Obecny etap: ręczna anotacja *wszystkich* fraz w wybranym w ten sposób podzbiorze zdań.
- Poszczególne frazy wygenerowane metodą „zwijania” drzewa do góry.
- Pełna anotacja tego typu prawdopodobnie zostanie ograniczona do wyodrębnionych obecnie ok 300 zdań.
- Plany na przyszłość
  - Wygenerowanie większych zbiorów treningowych metodami automatycznymi (np sentipejd)

Dziekujemy za uwagę.