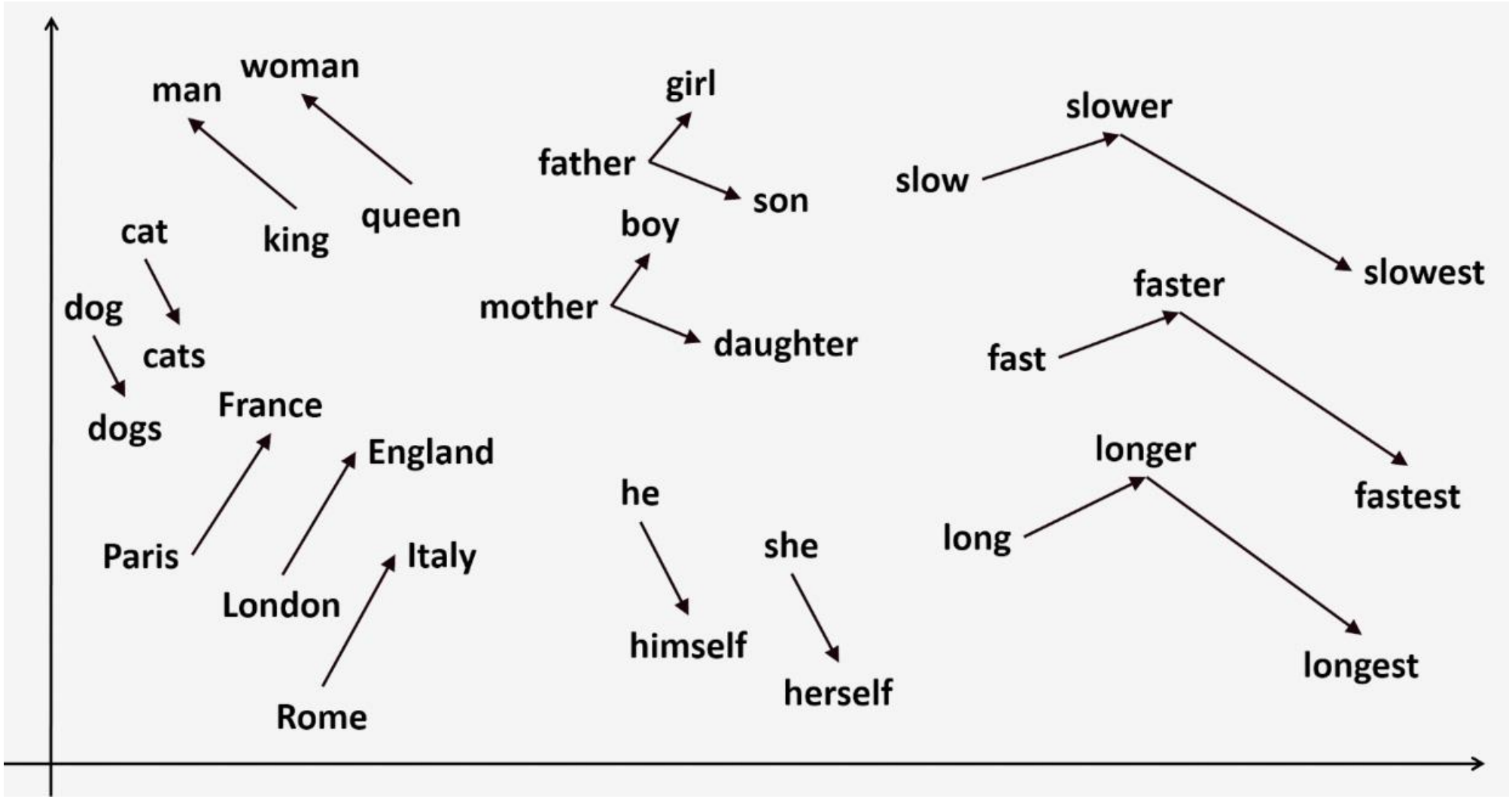
23/04/25

Векторизация текста

Векторизация текста и простые модели на основе нейронных сетей (TF-IDF, Word2Vec, FastText, Glove, Elmo) и их использование в простых нейронных сетях, таких как полносвязные сети для классификации текстов.

Подготовил:
Поляков Михаил

Векторные представления слов



TF-IDF

$\text{tf-idf}(t, d, D)$ — это мера важности слова t для документа d в наборе документов D.

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

Term Frequency $\text{tf}(t, d)$ считается для слова и документа

$$\text{tf}(t, d) = \frac{n_t}{\sum_k n_k}$$

n_t — количество раз, которое слово t встречается в документе d

$\sum_k n_k$ — количество слов в документе d

Документ #1

Term	Term count
a	1
dog	1
eats	1
meat	1

Документ #2

Term	Term count
a	1
dog	1
hunts	1
cat	1

tf("a", doc_1) = 1 / 4 = 0.25

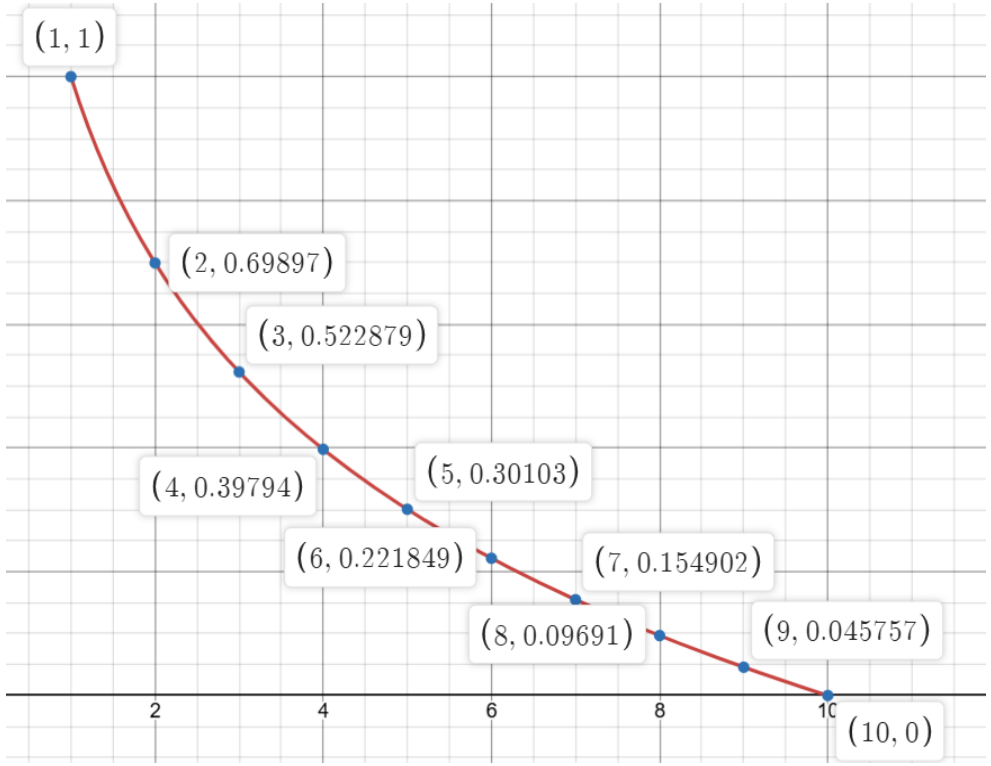
tf("a", doc_2) = 1 / 4 = 0.25

Inverse document frequency $\text{idf}(t, D)$
считается для слова и набора документов

количество документов в наборе

$$\text{idf}(t, D) = \log \frac{|D|}{|\{ d_i \in D \mid t \in d_i \}|}$$

количество документов, в которых встречается слово t



Значения idf для |D|=10

$$\text{idf}(\text{"a"}, D) = \log(2 / 2) = \log(1) = 0$$

$\text{tf-idf}(t, d, D)$ — это мера важности слова t для документа d в наборе документов D.

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

Документ #1

Term	Term count
a	1
dog	1
eats	1
meat	1

Документ #2

Term	Term count
a	1
dog	1
hunts	1
cat	1

tf ("a", doc_1) = 1 / 4 = 0.25
tf ("a", doc_2) = 1 / 4 = 0.25

idf ("a", D) = log (2 / 2) = log(1) = 0

tf-idf("a", doc_1, D) = 0.25 * 0 = **0**
tf-idf("a", doc_2, D) = 0.25 * 0 = **0**

1. [0, ..., 0, ... 0.075, ... 0, ..., 0, ..., 0, ...]
2. [0, ..., 0, ... 0, ..., 0.075, ... 0.075, ..., 0.075, ...]

a dog eats hunts cat meat

Еще примеры использования TF-IDF:

- Ранжирование поисковой выдачи;
- Выделение ключевых слов и суммаризация текста.

Плюсы:

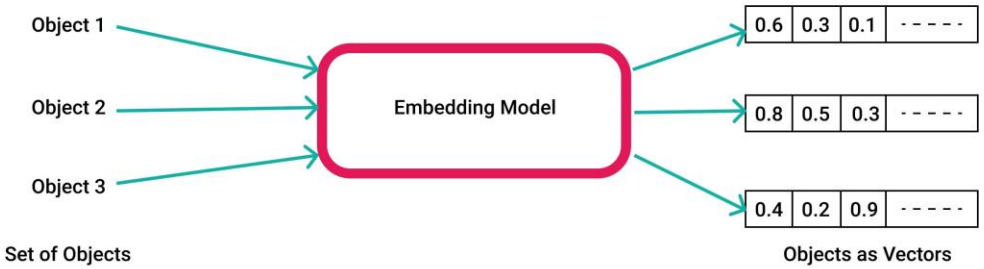
- Векторы имеют больший смысл, чем при BoW;
- Возможность решать такие задачи, как ранжирование документов и выделение ключевых слов;

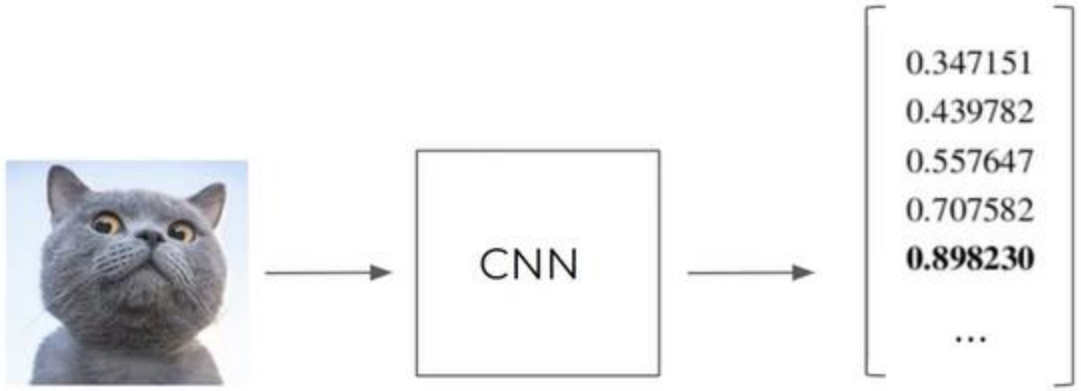
Недостатки:

- Векторы довольно разрежены;
- Фиксированный размер словаря.
- При изменении коллекции документов векторы нужно пересчитывать.

Эмбеддинги

Эмбеддинг - это способ преобразования чего-то абстрактного, например слов, в векторы чисел, которые отражают суть или семантику исходного объекта и могу быть сравнимы по некоторой метрике.

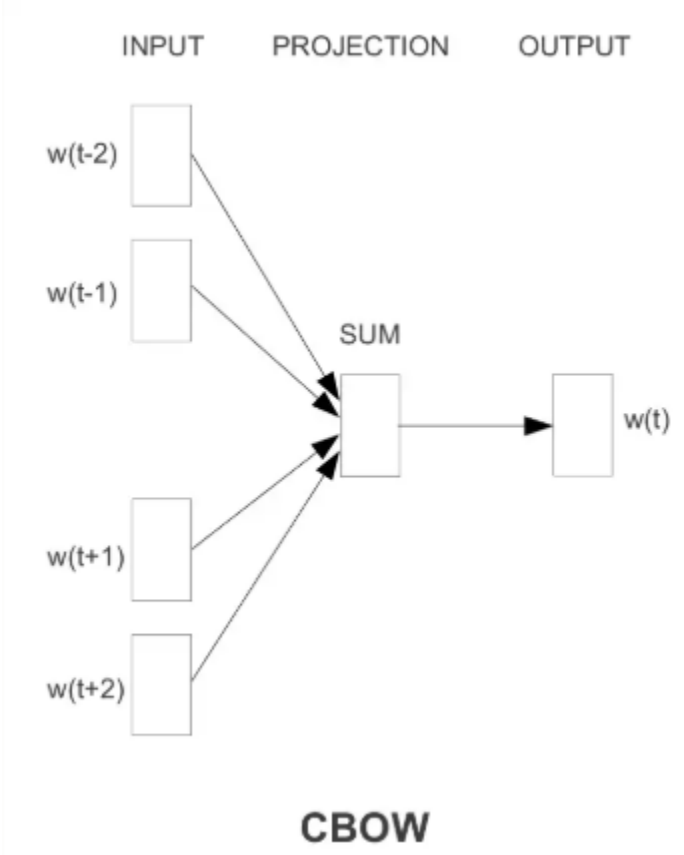




Word2Vec

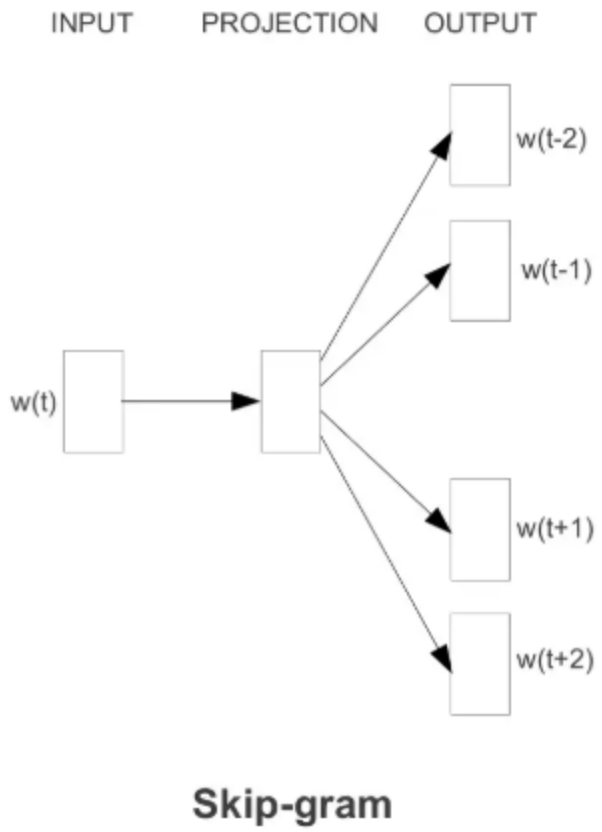
CBOW

Предсказание центрального слова по его контексту

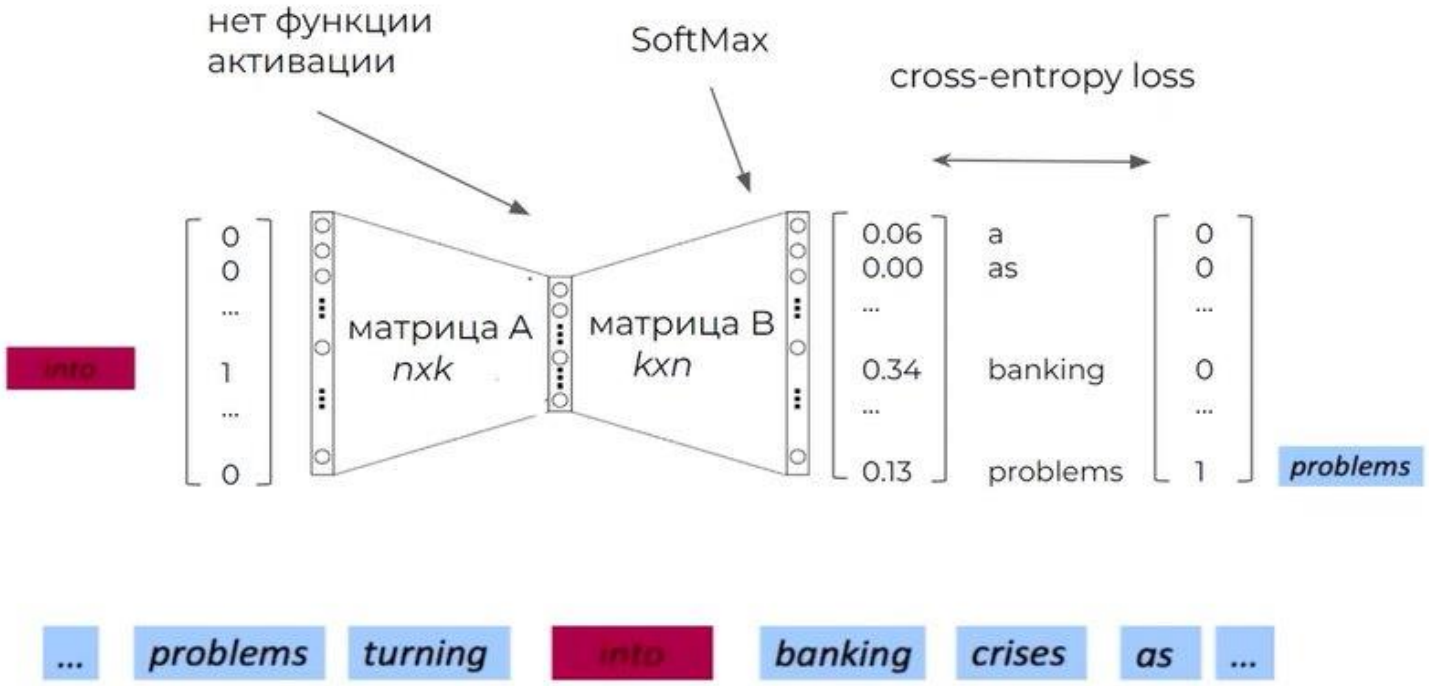


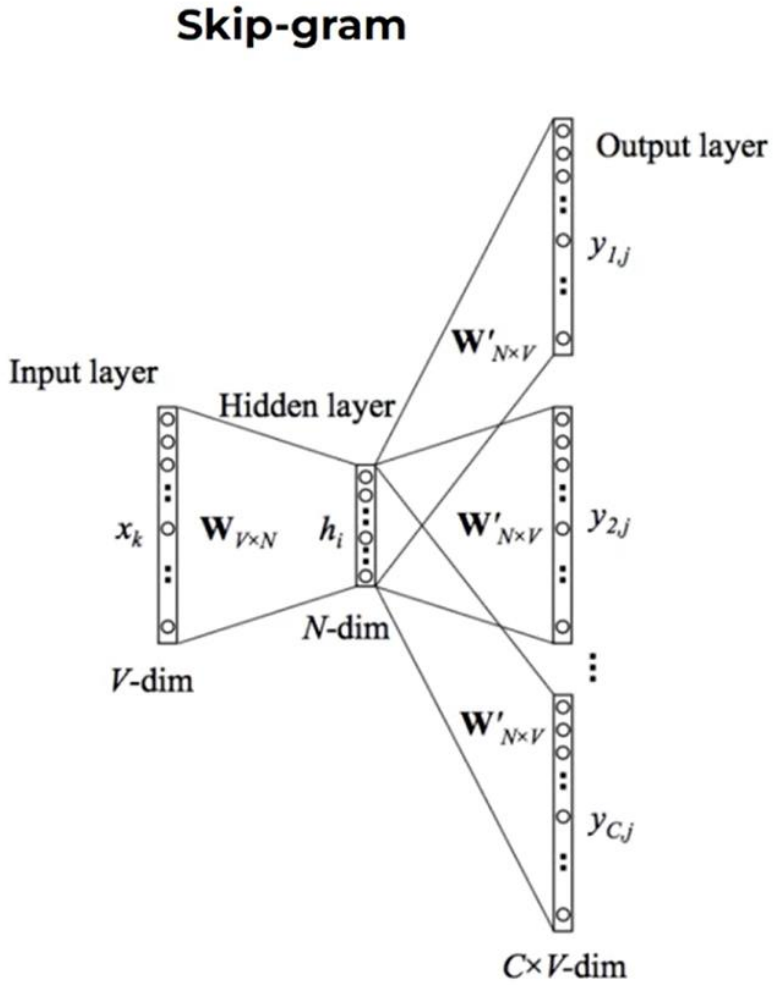
Skip-gram

Предсказание контекста по центральному слову



Skip-gram





Общая схема для контекста

Skip-gram

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

На векторах word2vec можно проводить векторную арифметику:

$$\mathbf{v}(\text{king}) - \mathbf{v}(\text{man}) + \mathbf{v}(\text{woman}) \approx \mathbf{v}(\text{queen})$$

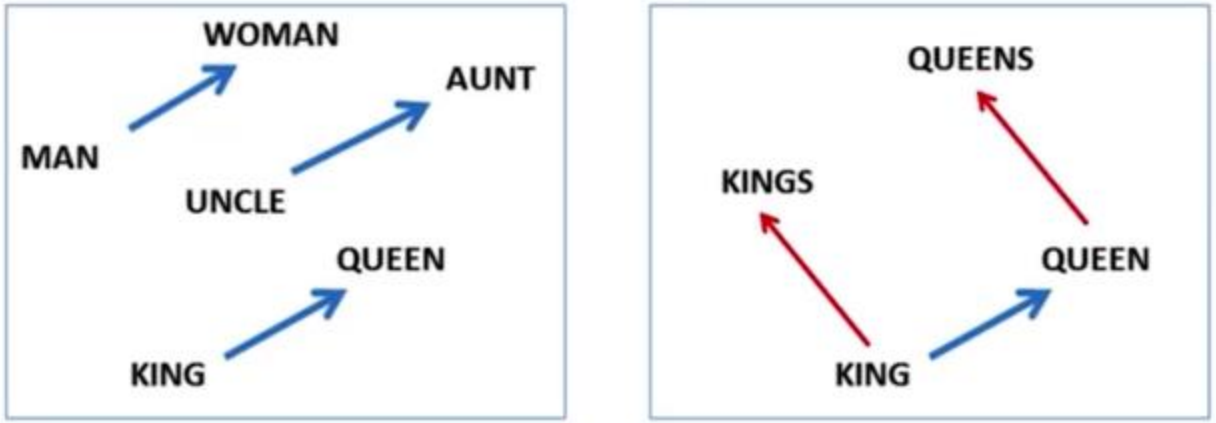


Иллюстрация эмбеддингов после понижения размерности

CBOW

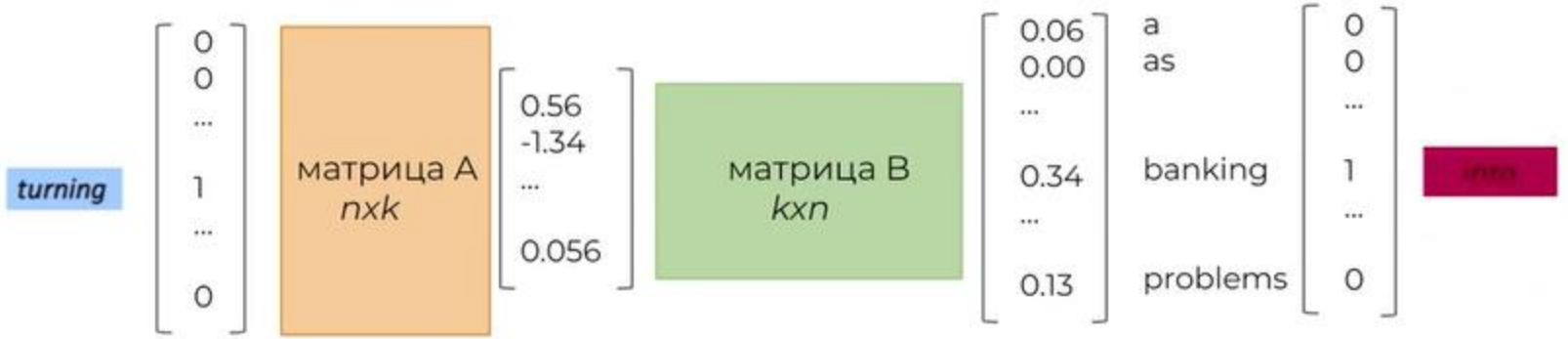
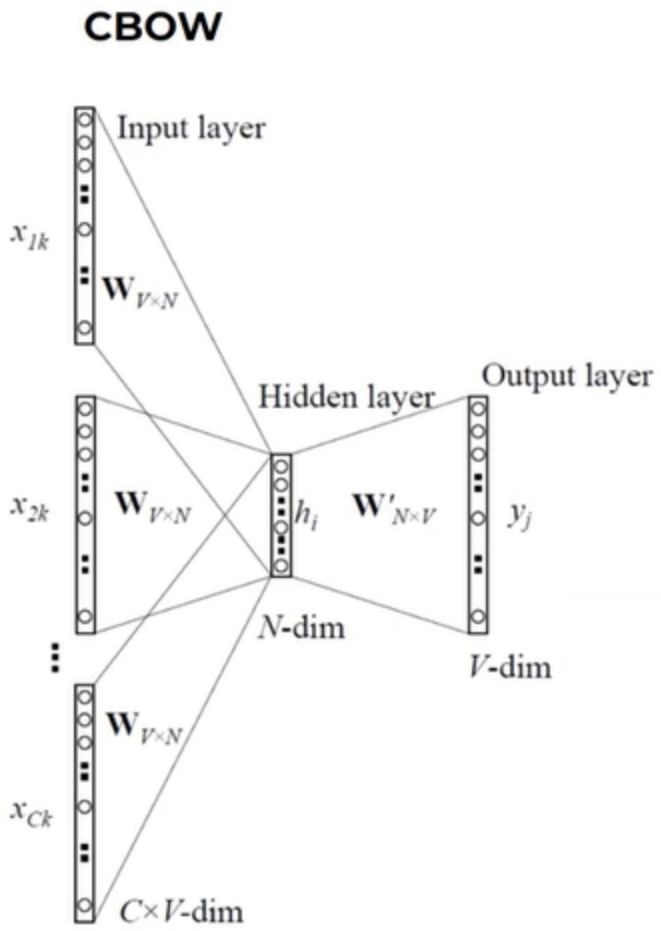


Схема модели



Общая схема для контекста

Преимущества:

- Векторы лучше отражают смысл слов.
- Размерность векторов не зависит от размера словаря.
- При добавлении документов векторы можно до обучить.

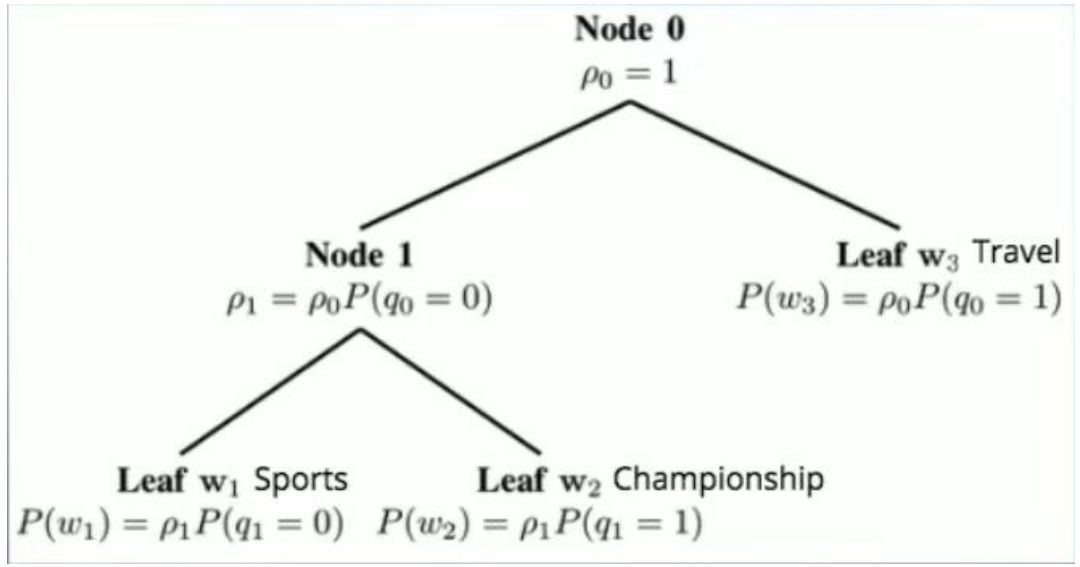
Недостатки:

- Фиксированный размер словаря.
- Для редких слов эмбеддинги получаются неоптимальными.
- Слова, имеющие один корень, обрабатываются нейросетью по-разному. $eat, eater, eating$ } **FastText**
- Не использует глобальную статистику слов. → **Glove**

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Идеи решения проблемы:

- Иерархический SoftMax;
- Negative sampling;



Skipgram

shalt	not	make	a	machine

input	output
make	shalt
make	not
make	a
make	machine

Negative Sampling

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

Glove

	close	looking	january	deal	billion	1	startup
close	0	0	1	1	0	0	0
looking	0	0	0	0	0	0	1
january	1	0	0	0	0	0	0
deal	1	0	0	0	1	1	0
billion	0	0	0	1	0	1	1
1	0	0	0	1	1	0	1
startup	0	1	0	0	1	1	0

Обозначим эту матрицу X

X_{ij} - количество раз слово j было в контексте слова i

$$X_i = \sum_{k=1}^{V} X_{ik}$$

$$P_{ij} = P(i|j) = \frac{X_{ij}}{X_i}$$ - вероятность слова j в контексте i

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}, \tag{1}$$

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}. \tag{2}$$

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}}, \tag{3}$$

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}, \tag{4}$$

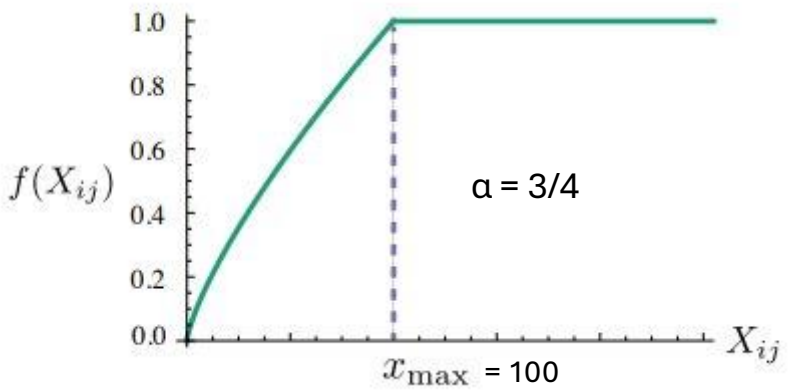
$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}. \tag{5}$$

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i). \tag{6}$$

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik}). \tag{7}$$

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2, \tag{8}$$

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}.$$



Обучение

- Оптимизация через **стохастический градиентный спуск (SGD)**.
- После обучения **векторы слов** (w_i) и **контекстные векторы** ($\tilde{w}_j$) либо усредняются, либо используются по отдельности.

Преимущества:

- Простая архитектура.
- Модель быстрая, и более эффективная для небольших задач.
- Осмысленные эмбеддинги, за счёт использования глобальной статистики

Недостатки:

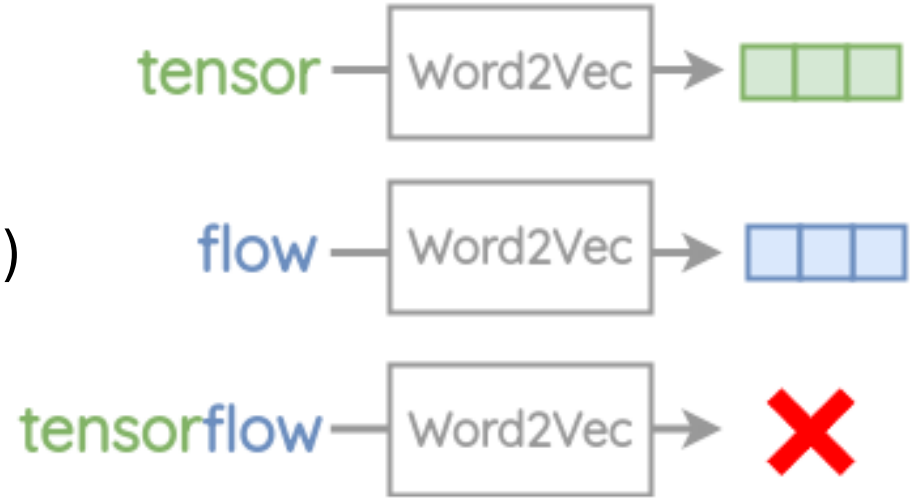
- Требует построения дополнительной матрицы слов.
- Сложно до обучить на новых данных.
- Все ещё достаточно плохо обрабатывает неизвестные и редкие слова.

fastText

1) Shared radical

eat eats eaten eater eating

2)

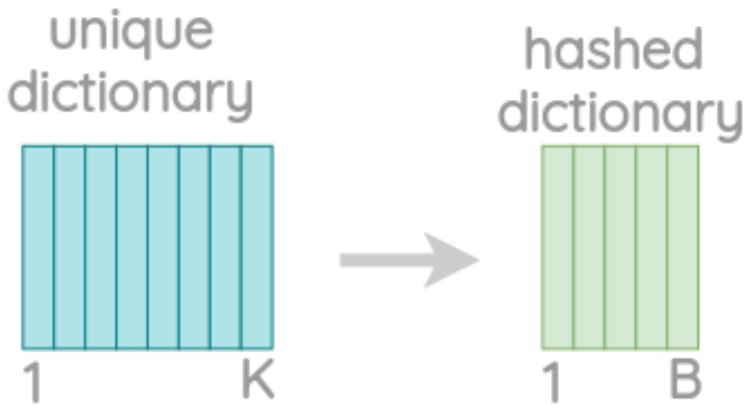


1) eating → <eating>

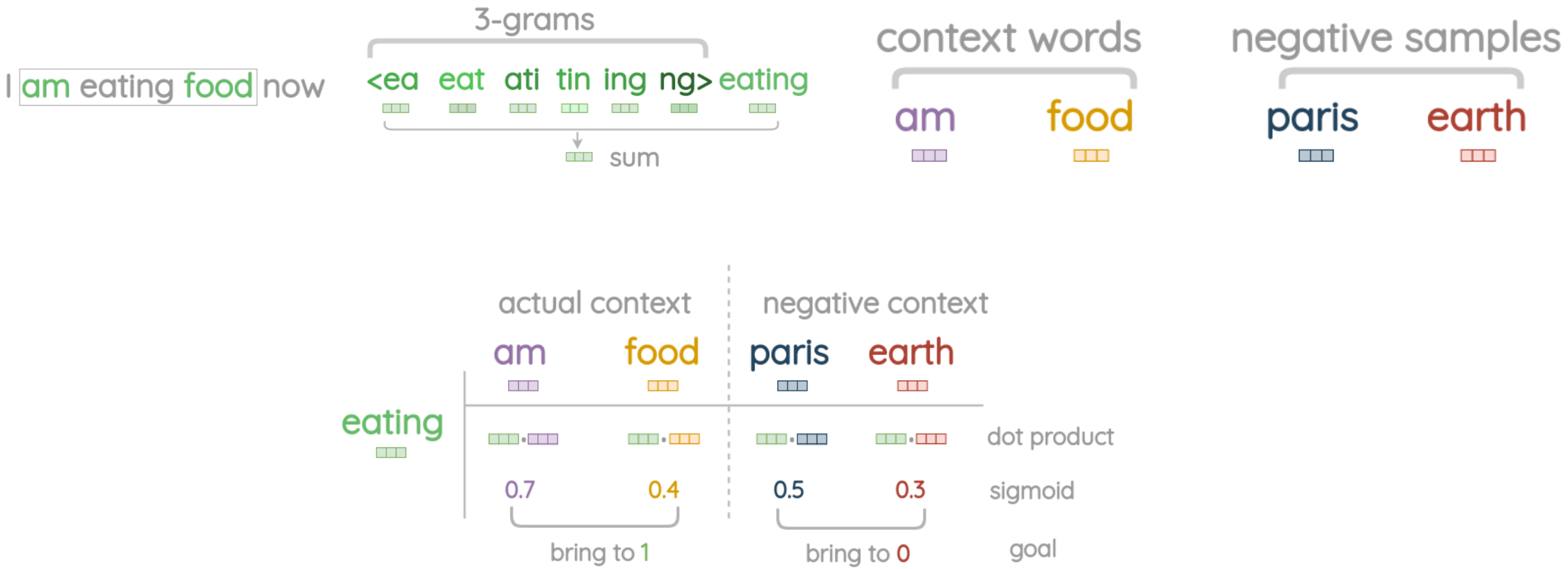
2) <eating> 3-grams <eating>

<ea eat ati tin ing ng>

Word	Length(n)	Character n-grams
eating	3	<ea, eat, ati, tin, ing, ng>
eating	4	<eat, eati, atin, ting, ing>
eating	5	<eati, eatin, ating, ting>
eating	6	<eatin, eating, ating>



Размерности словаря

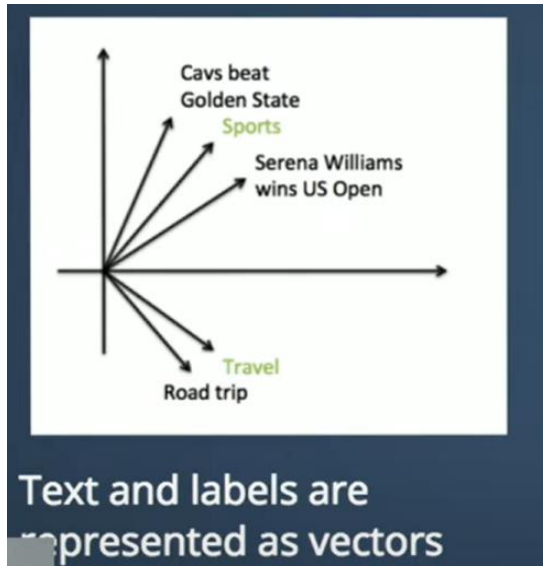


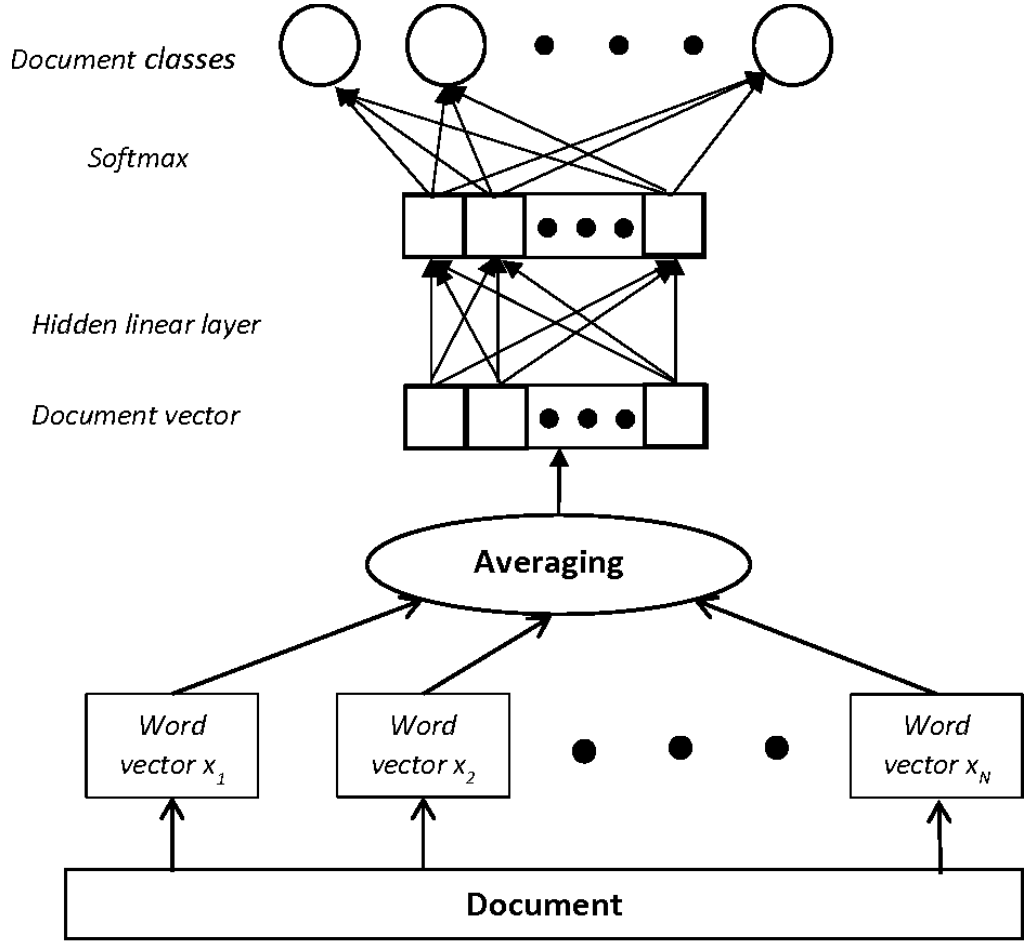
$$\mathcal{L} = -\sum_{(w,c)\in D}\left[\log\sigma(v_c \cdot v_w) + \sum_{k=1}^{K}\log\sigma(-v_{n_k}\cdot v_w)\right]$$

- v_w – вектор слова,
- v_c – вектор контекста,
- n_k – "негативные" примеры (случайные слова),
- σ – сигмоида.

$$\mathcal{L} = -\sum_{i=1}^{N} \log \left(\frac{\exp(W_{y_i} \cdot x_i)}{\sum_{j=1}^{C} \exp(W_j \cdot x_i)} \right)$$

- x_i – усредненный вектор текста,
- W_{y_i} – веса для правильного класса,
- C – количество классов.





Достоинства и недостатки

Плюсы:

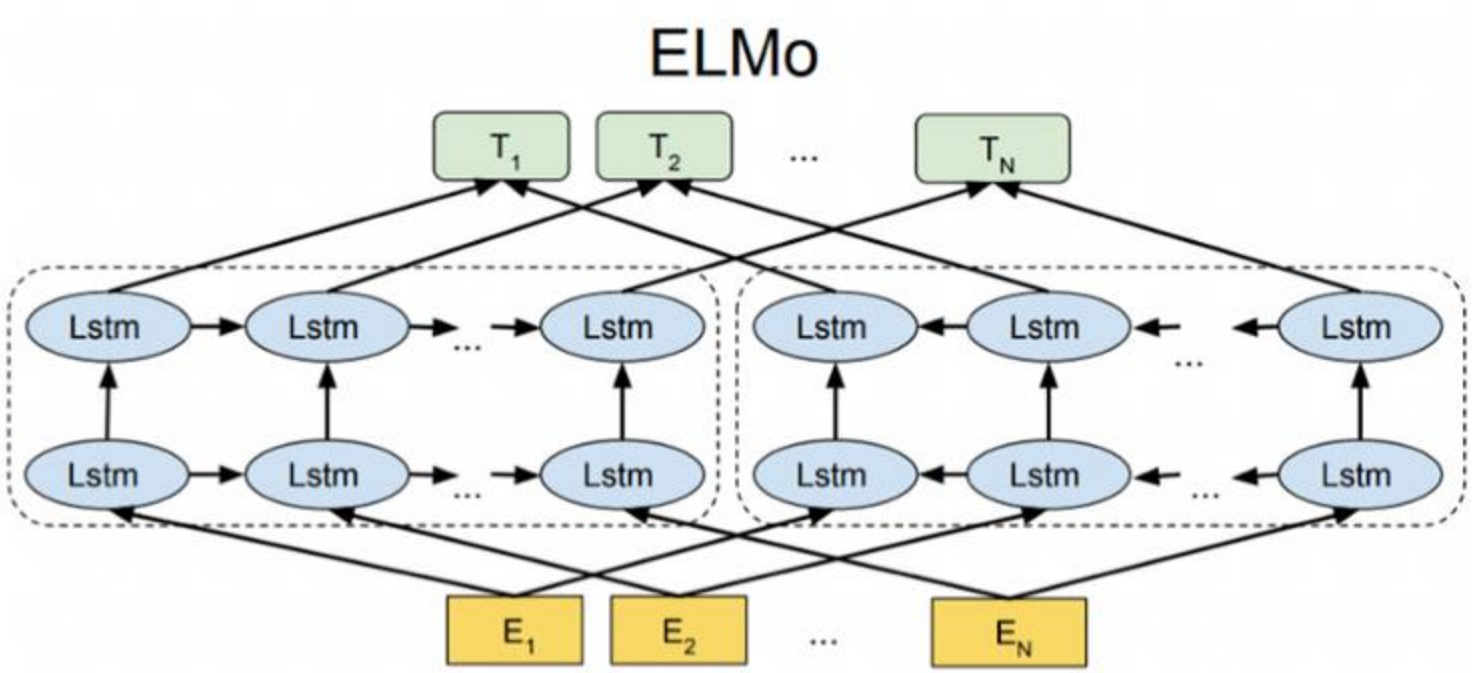
1. Эффективная работа с редкими и неизвестными словами
2. Учёт морфологии языка
3. Быстрое обучение
4. Можно использовать как для получения эмбеддингов, так и классификации

Недостатки:

1. Один вектор для слова во всех контекстах.
2. Линейная модель для классификации.

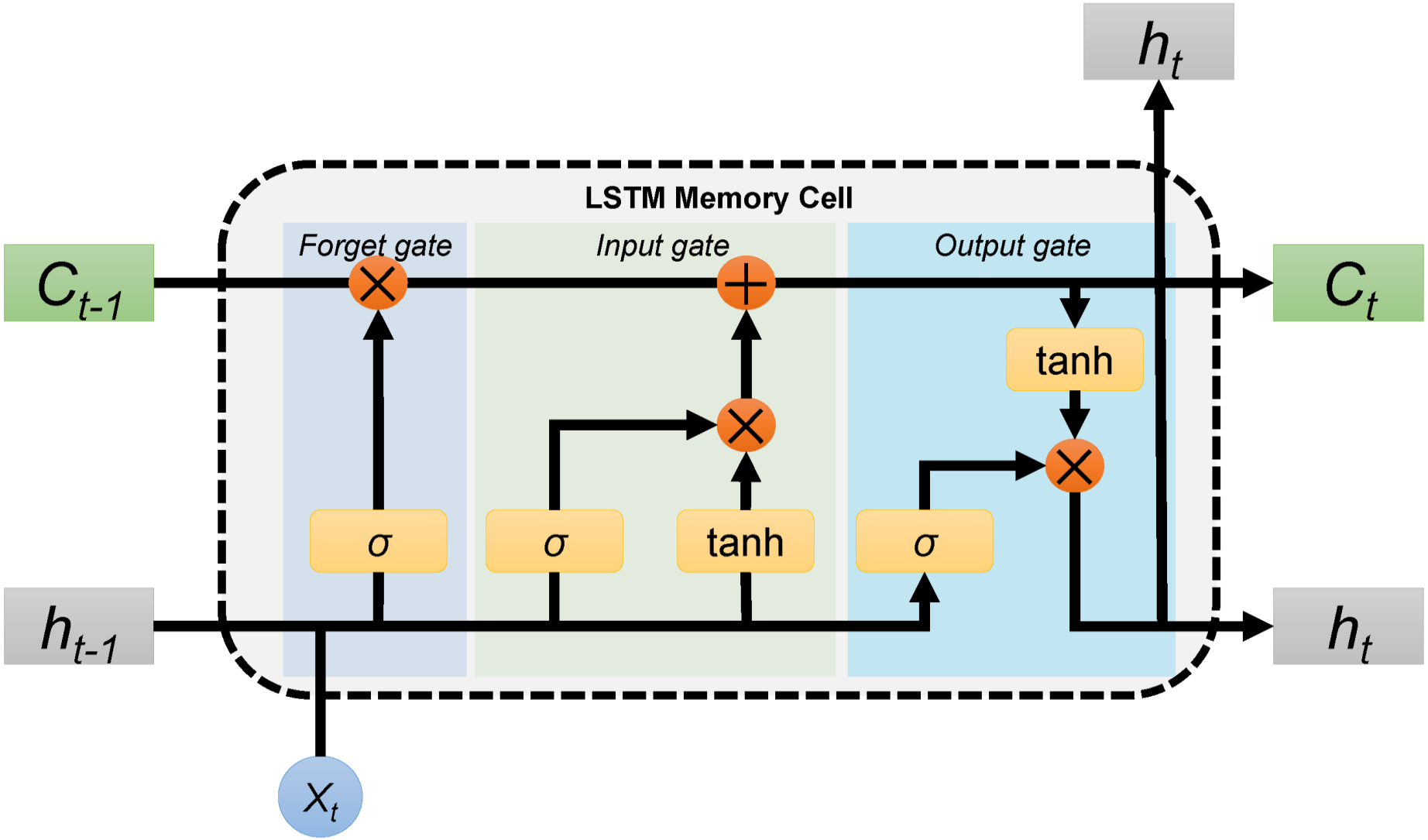
ELMo

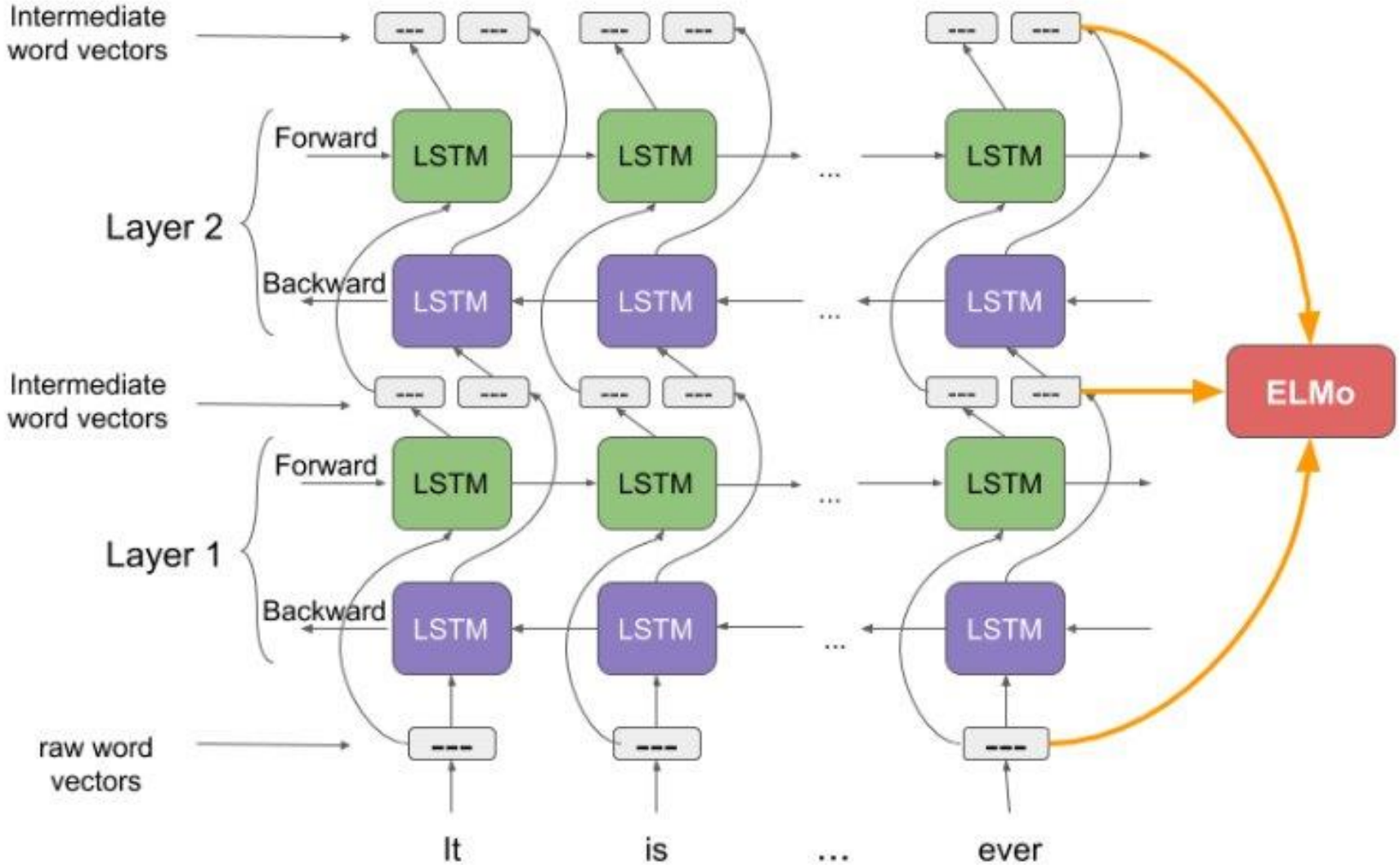
ELMO (1980)

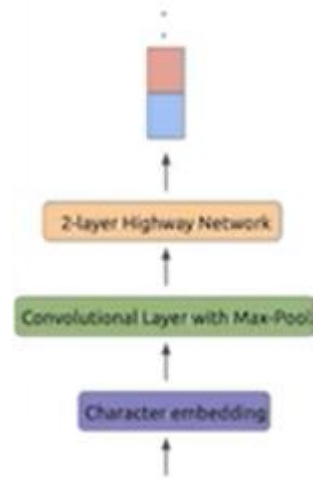


ELMo (2018)

LSTM


h_t
LSTM Memory Cell
Forget gate
Input gate
Output gate
C_{t-1}
C_t
tanh
σ
σ
tanh
σ
h_{t-1}
h_t
X_t

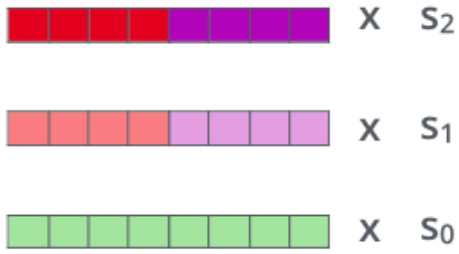

Intermediate word vectors
Forward
LSTM
Layer 2
Backward
Intermediate word vectors
Forward
Layer 1
Backward
raw word vectors
ELMo
It
is
...
ever


2-layer Highway Network
Convolutional Layer with Max-Pool
Character embedding

Embedding of “stick” in “Let’s stick to” - Step #2

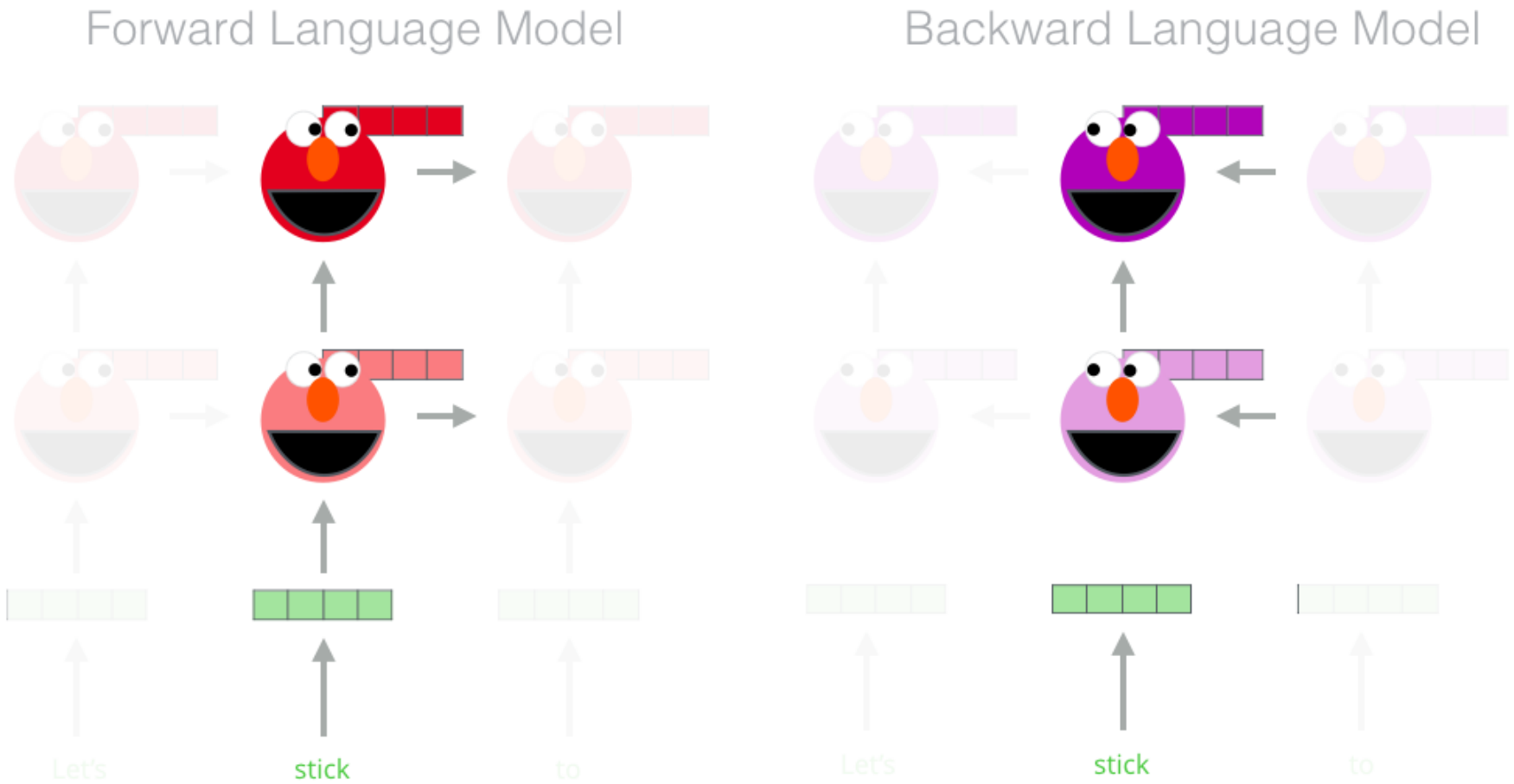
1- Concatenate hidden layers

2- Multiply each vector by a weight based on the task



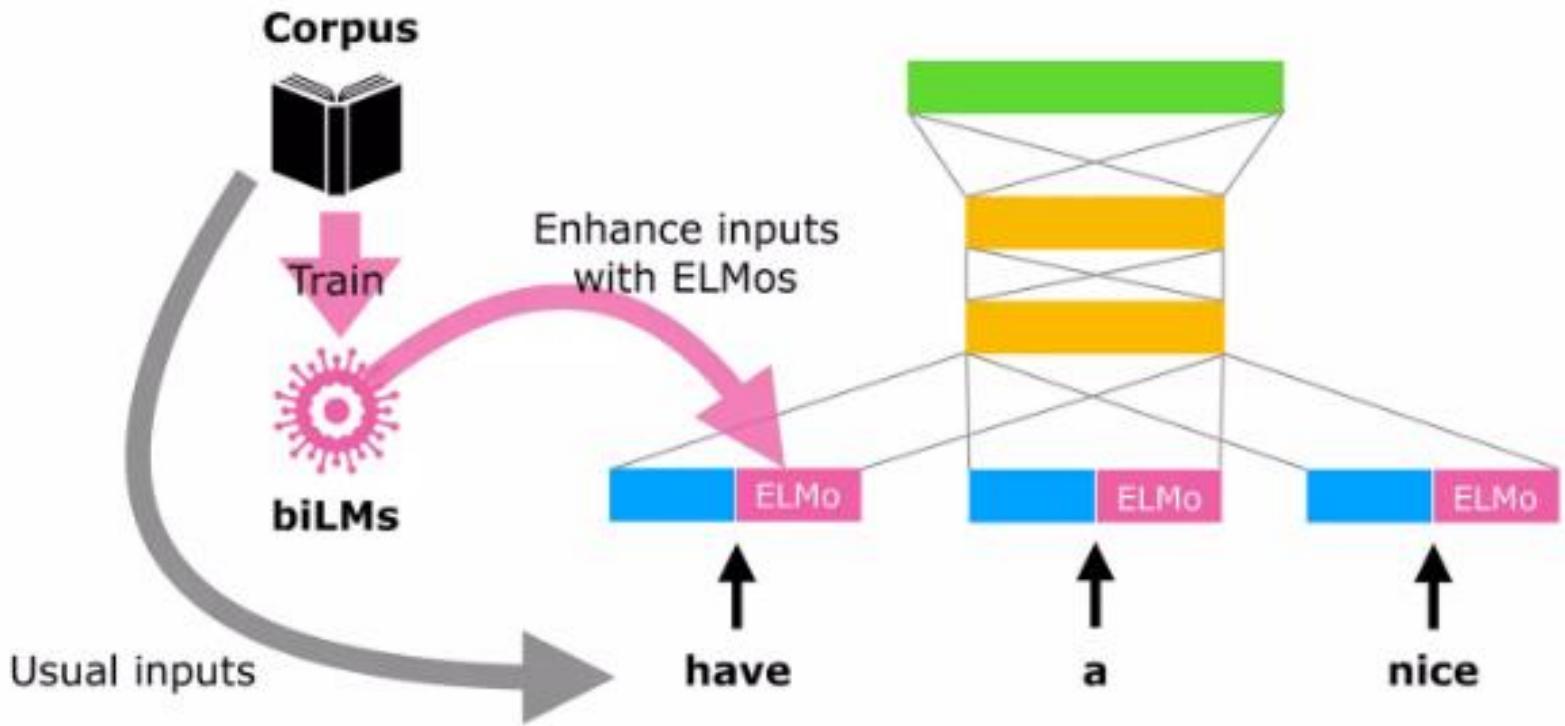
3- Sum the (now weighted) vectors

ELMo embedding of “stick” for this task in this context



$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

ELMo can be integrated to almost all neural NLP tasks with simple concatenation to the embedding layer



Плюсы:

1. Контекстные эмбеддинги.
2. Работает с OOV
3. Можно добавить в любую модель.

Минусы:

1. Медленная модель (LSTM)
2. Длина контекста ограничена ~512 токенами
3. Устарел. BERT и GPT лучше по всем параметрам.

Литература

https://www.youtube.com/watch?v=EOgwODW67PE TF-IDF

https://www.youtube.com/watch?v=RKGi26Yk-5A&t=584s

https://www.youtube.com/watch?v=WbtQzAvhnRI Word2Vec

https://habr.com/ru/articles/446530/

https://habr.com/ru/companies/ods/articles/329410/

https://nlp.stanford.edu/pubs/glove.pdf GloVe

https://arxiv.org/pdf/1607.04606v2

https://www.geeksforgeeks.org/fasttext-working-and-implementation/ FastText

https://amitness.com/posts/fasttext-embeddings

Лекция 22. ELMo & BERT

https://habr.com/ru/articles/487358/https://www.youtube.com/watch?v=Q4HVS6c92qU ELMo

https://www.slideshare.net/slideshow/a-review-of-deep-contextualized-word-representations-peters-2018/102809428#8

https://arxiv.org/pdf/1802.05365

https://translated.turbopages.org/proxy_u/en-ru.ru.3603fa21-6806a000-31457dab-74722d776562/https/www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/

Спасибо за внимание!