Распределённые векторные представления (distributed representations, embeddings)

Алексей Романов
alexey.romanov@phystech.edu

МФТИ, ABBYY

27.10.2017

План рассказа

План рассказа

Что будет:

- Введение
- Word embeddings: общая схема
- Word embeddings: подробный разбор `word2vec`
- Word embeddings: краткий разбор альтернативных подходов
- Image embeddings
- Graph embeddings

План рассказа

Что будет:

- Введение
- Word embeddings: общая схема
- Word embeddings: подробный разбор `word2vec`
- Word embeddings: краткий разбор альтернативных подходов
- Image embeddings
- Graph embeddings

Чего не будет:

- Topic modeling (LSA, LDA...)
- Архитектуры глубоких сетей
- Подробный рассказ о применениях

Терминология

Терминология

Embedding

Вложение — инъективное отображение $f : X \to Y$, сохраняющее структуру множества X.

Терминология

Embedding

Вложение — инъективное отображение $f : X \to Y$, сохраняющее структуру множества X.

Distributed representation

Распределённое представление — представление дискретных объектов вещественнозначными векторами

Терминология

Embedding

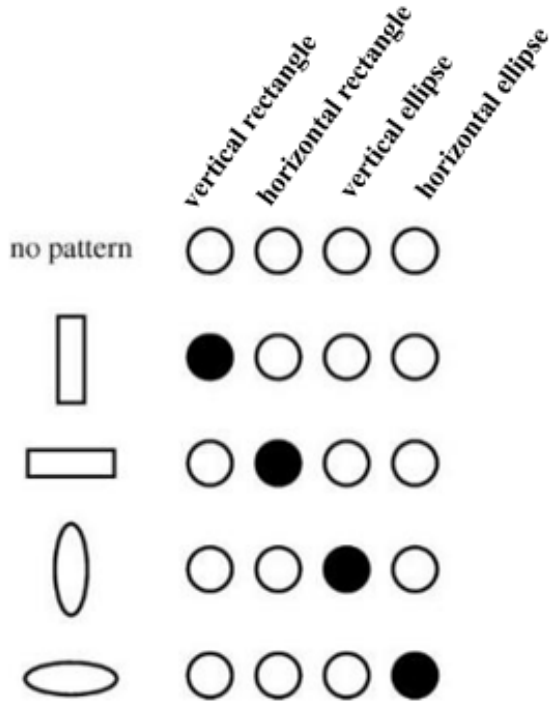
Вложение — инъективное отображение $f : X \to Y$, сохраняющее структуру множества X.

Distributed representation

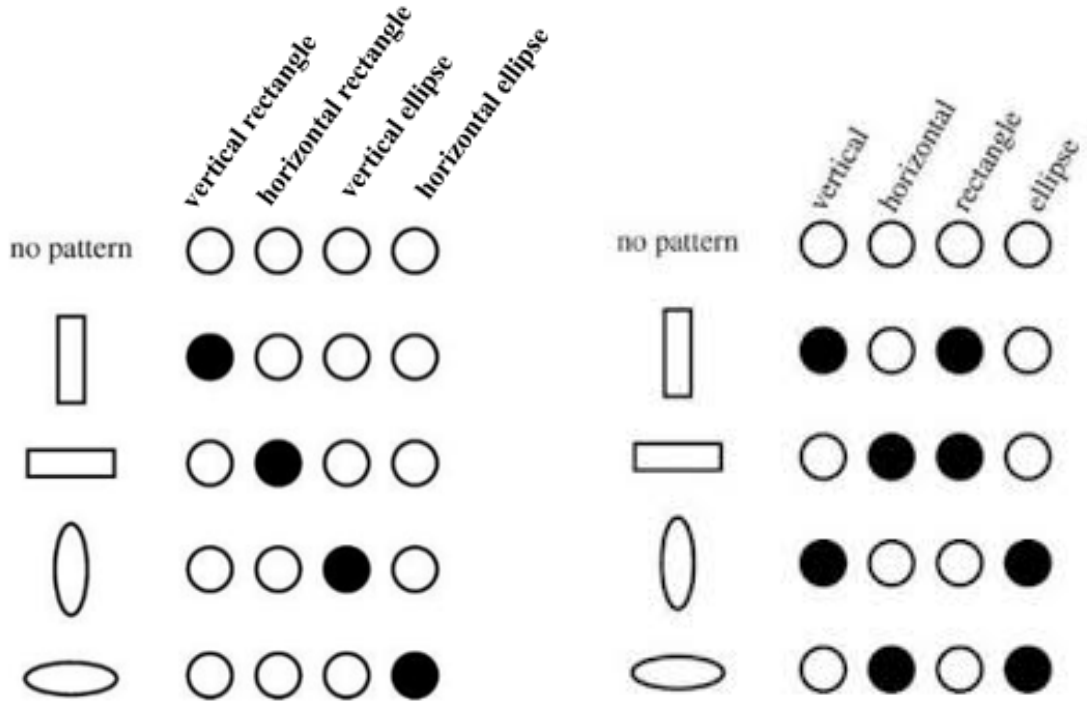
Распределённое представление — представление дискретных объектов вещественнозначными векторами

Чего мы хотим?

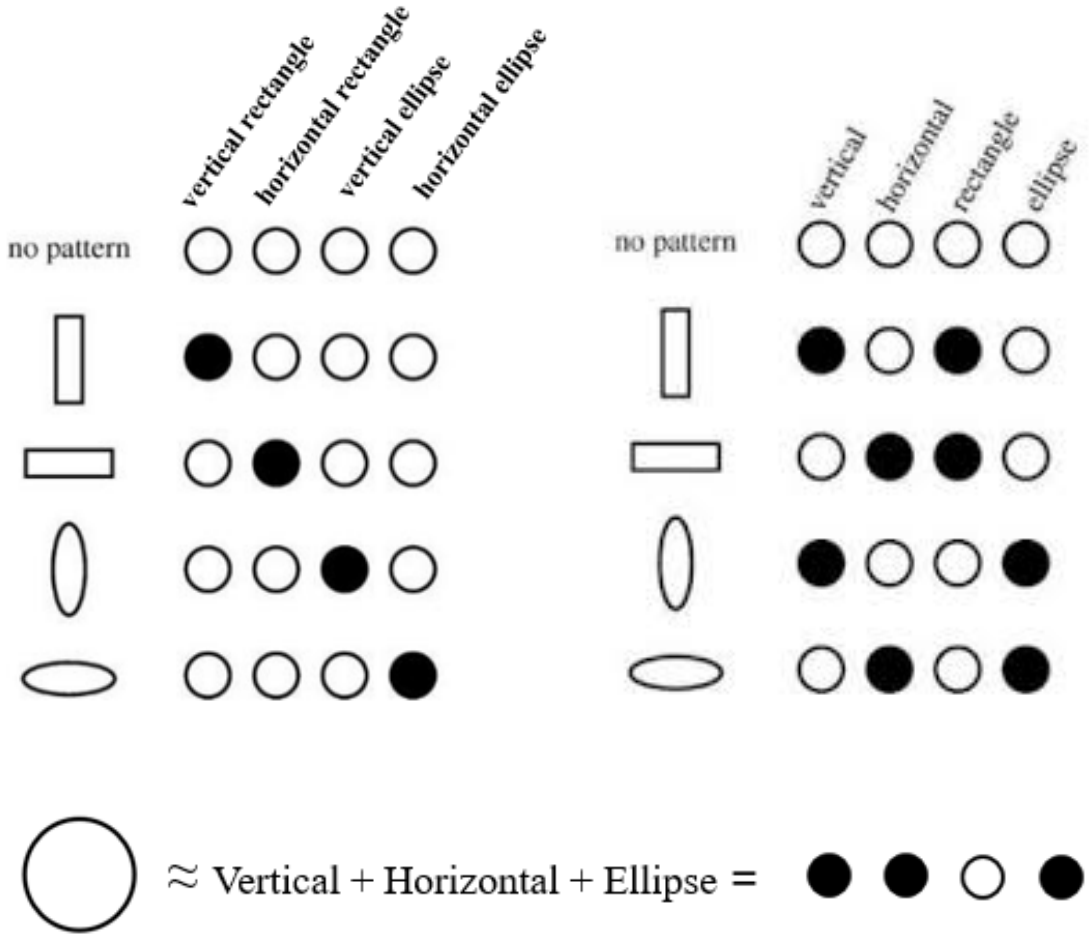
Local representation vs Distributed representation



Local representation vs Distributed representation



Local representation vs Distributed representation



Общая идея

Общая идея

Мы хотим встраивать различные объекты:

Общая идея

Мы хотим встраивать различные объекты:

- буквы, слова, тексты
- изображения, видео
- графовые структуры
- абстрактные понятия (PoS-тэги, геометрические фигуры)

Общая идея

Мы хотим встраивать различные объекты:

- буквы, слова, тексты
- изображения, видео
- графовые структуры
- абстрактные понятия (PoS-тэги, геометрические фигуры)

в пространство $\mathbb{R}^n$ таким образом, чтобы:

- разные объекты получили разные векторы
- похожие объекты получили похожие векторы

Общая идея

Мы хотим встраивать различные объекты:

- буквы, слова, тексты
- изображения, видео
- графовые структуры
- абстрактные понятия (PoS-тэги, геометрические фигуры)

в пространство $\mathbb{R}^n$ таким образом, чтобы:

- разные объекты получили разные векторы
- похожие объекты получили похожие векторы

Похожесть объектов определяется конкретной задачей.

From symbolic to distributed representations

Its problem, e.g., for web search

- If user searches for [Dell notebook battery size], we would like to match documents with "Dell laptop battery capacity"
- If user searches for [Seattle motel], we would like to match documents containing "Seattle hotel"

But

motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]^T
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

Our query and document vectors are **orthogonal**

There is no natural notion of similarity in a set of one-hot vectors

Distributional similarity based representations

You can get a lot of value by representing a word by means of its neighbors

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

One of the most successful ideas of modern statistical NLP

government debt problems turning into	banking	crises as has happened in
saying that Europe needs unified	banking	regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Word embeddings: общая схема

1. Обучающая выборка (корпус текстов)
2. Кодирование local representations (bag-of-words, . . .)
3. Выбор функции ошибки
4. Инициализация эмбеддингов
5. Оптимизация функции ошибки

Word embeddings: общая схема

1. Обучающая выборка (корпус текстов)
2. Кодирование local representations (bag-of-words, . . .)
3. Выбор функции ошибки
4. Инициализация эмбеддингов
5. Оптимизация функции ошибки

Дистрибуционная семантика предполагает, что слова, встречающиеся в одинаковых контекстах, должны получить схожие представления. Исходя из этого следует выбирать функцию ошибки.

Word2Vec: Skip-Gram

Word2Vec: Skip-Gram

Функция ошибки: минус логарифм правдоподобия обучающей выборки

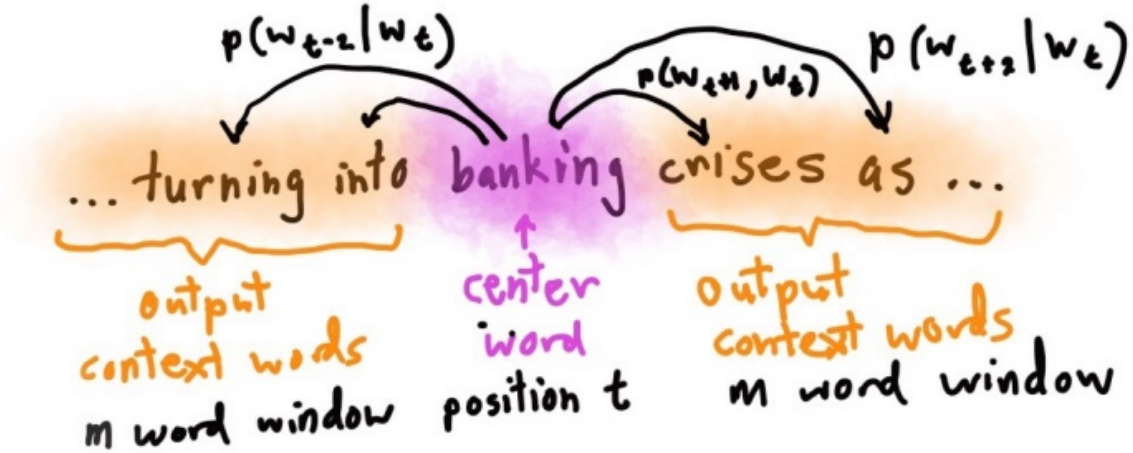
$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leqslant j\leqslant m,\\ j\neq 0}} \log p\left(w_{t+j}|w_t\right)$$

- θ — оптимизируемые параметры
- T — количество токенов в корпусе
- m — размер окна

Функция ошибки: минус логарифм правдоподобия обучающей выборки

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leqslant j\leqslant m,\\ j\neq 0}} \log p\left(w_{t+j}|w_t\right)$$

- θ — оптимизируемые параметры
- T — количество токенов в корпусе
- m — размер окна



Оценка вероятности встретить слово w_o в окрестности w_c:

$$p\left(w_o|w_c\right) = \frac{e^{\boldsymbol{u}_o^\top \boldsymbol{v}_c}}{\sum_{w=1}^{V} e^{\boldsymbol{u}_w^\top \boldsymbol{v}_c}}$$

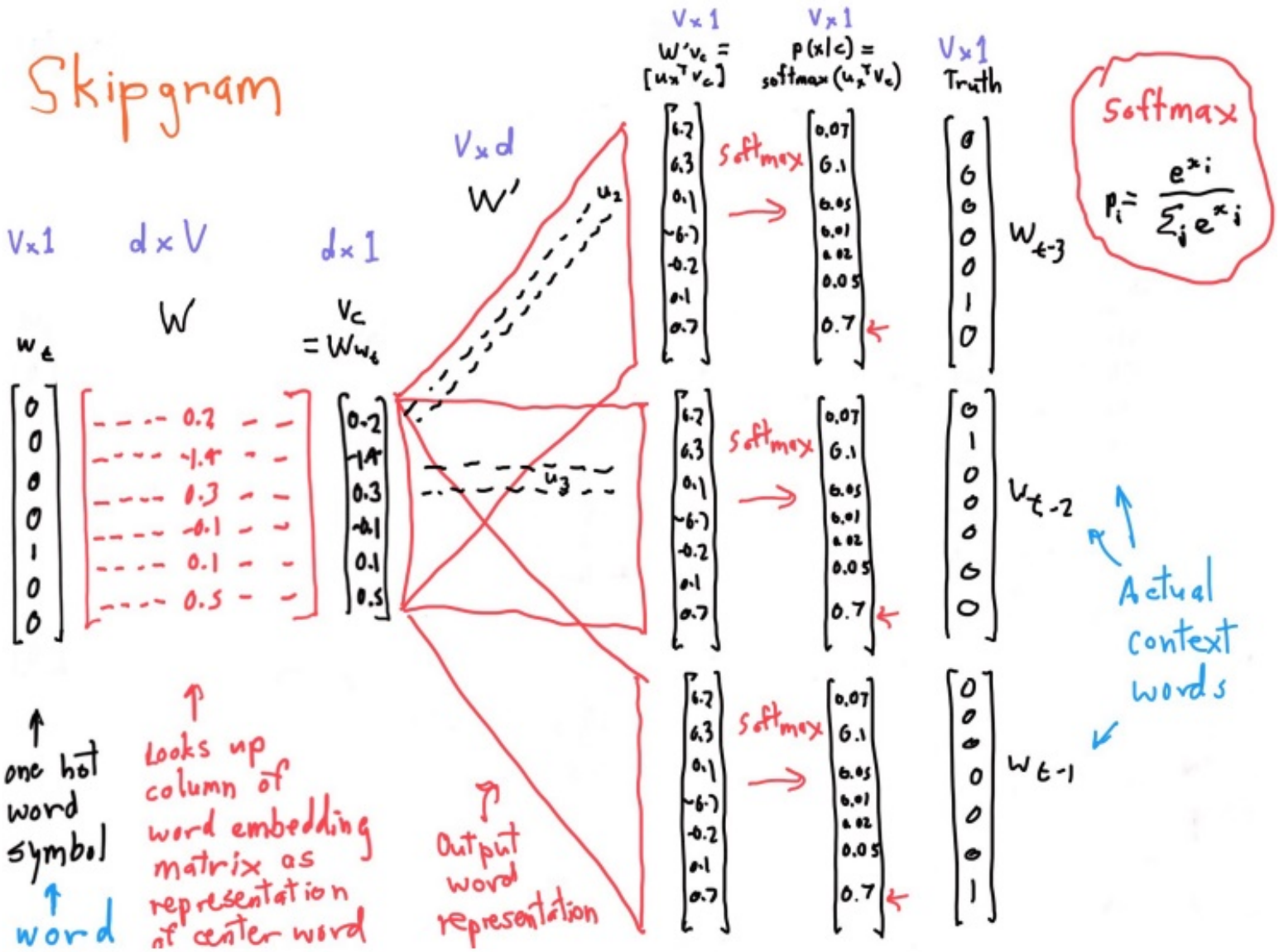
Word2Vec: Skip-Gram

Оценка вероятности встретить слово w_o в окрестности w_c:

$$p(w_o|w_c) = \frac{e^{\boldsymbol{u}_o^\top \boldsymbol{v}_c}}{\sum_{w=1}^{V} e^{\boldsymbol{u}_w^\top \boldsymbol{v}_c}}$$

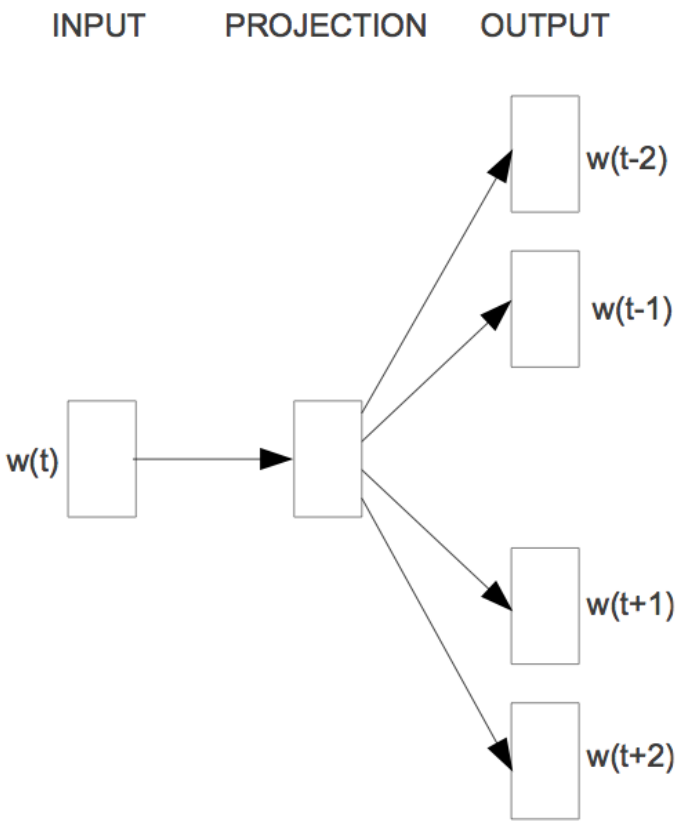
- softmax-преобразование — способ зажать значения в $(0, 1)$
- скалярное произведение $\Rightarrow$ косинусная близость
- V — размер словаря (количество уникальных слов)
- для каждого слова w нужно выучить векторы $\boldsymbol{u}_w$ и $\boldsymbol{v}_w$

Word2Vec: Skip-Gram

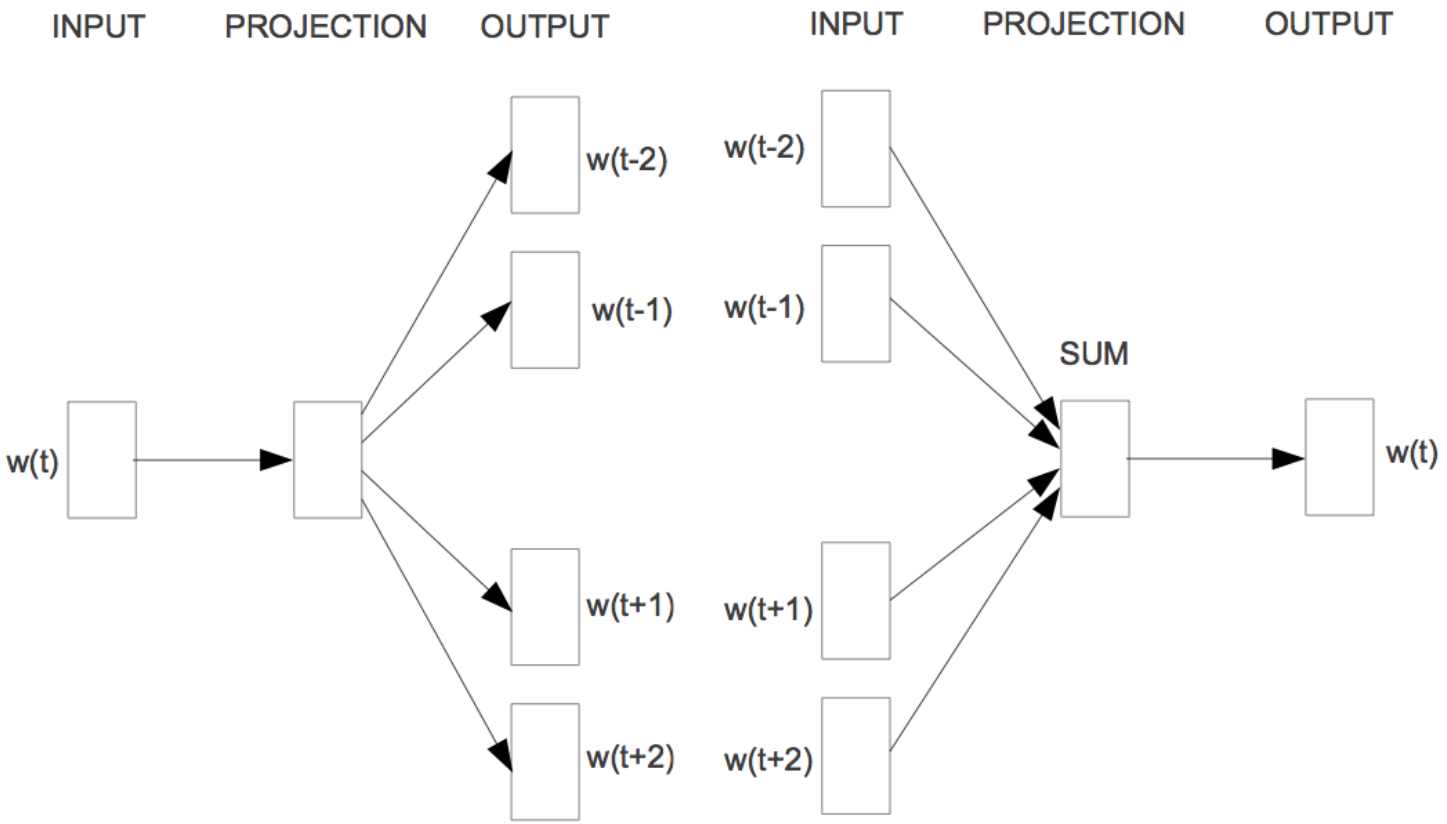


Word2Vec: Skip-Gram vs CBOW

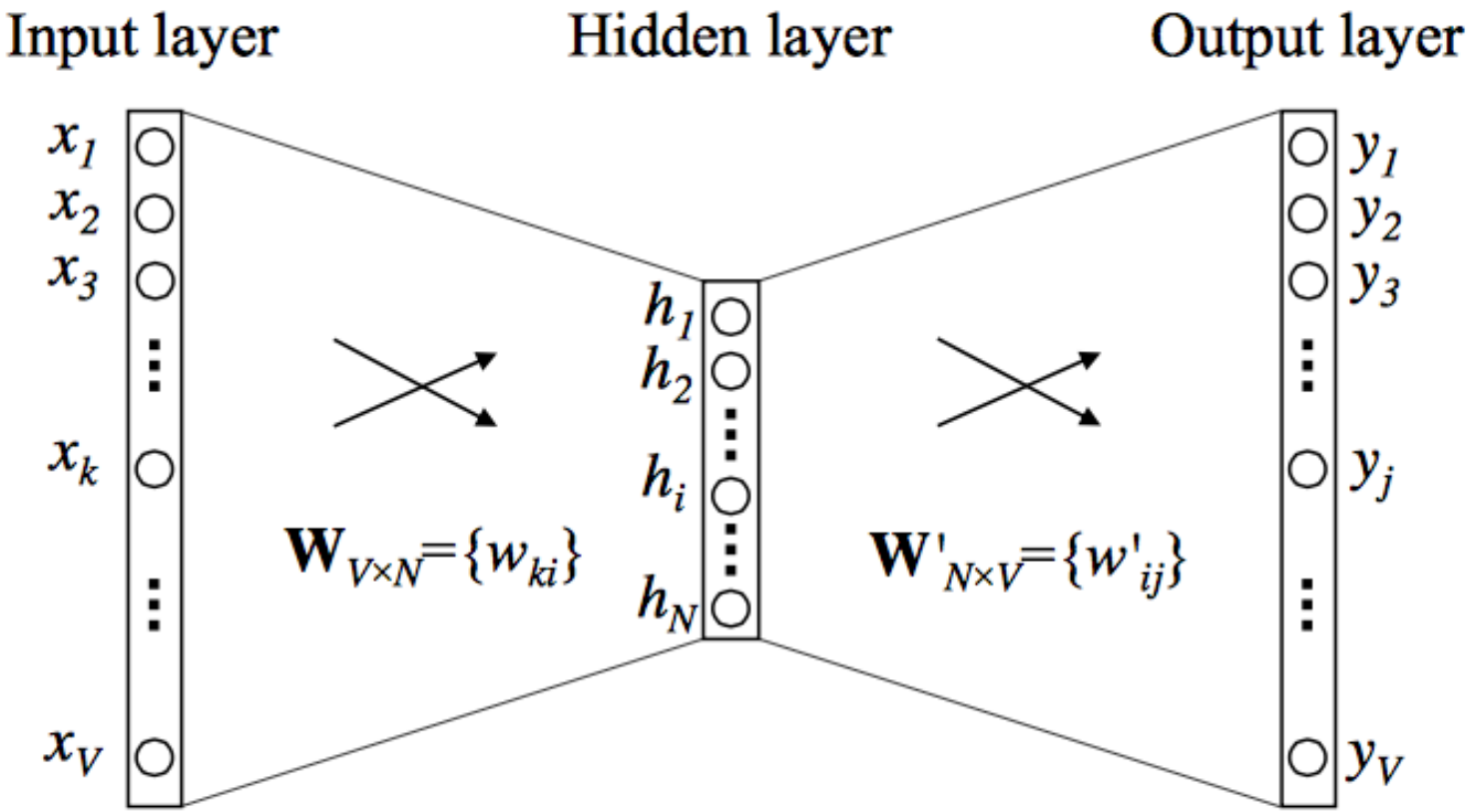
Word2Vec: Skip-Gram vs CBOW



Word2Vec: Skip-Gram vs CBOW



Word2Vec: Архитектура сети



Word2Vec: Технические проблемы

Word2Vec: Технические проблемы

Проблема: чтобы посчитать значение одного элемента на softmax-слое, нужно посчитать значения всех элементов.

- то же верно и для вычисления производных во время оптимизации

Word2Vec: Технические проблемы

Проблема: чтобы посчитать значение одного элемента на softmax-слое, нужно посчитать значения всех элементов.

- то же верно и для вычисления производных во время оптимизации

Способы борьбы с медленными вычислениями:

1. Изменение архитектуры softmax-слоя
2. Замена softmax приближённой оценкой

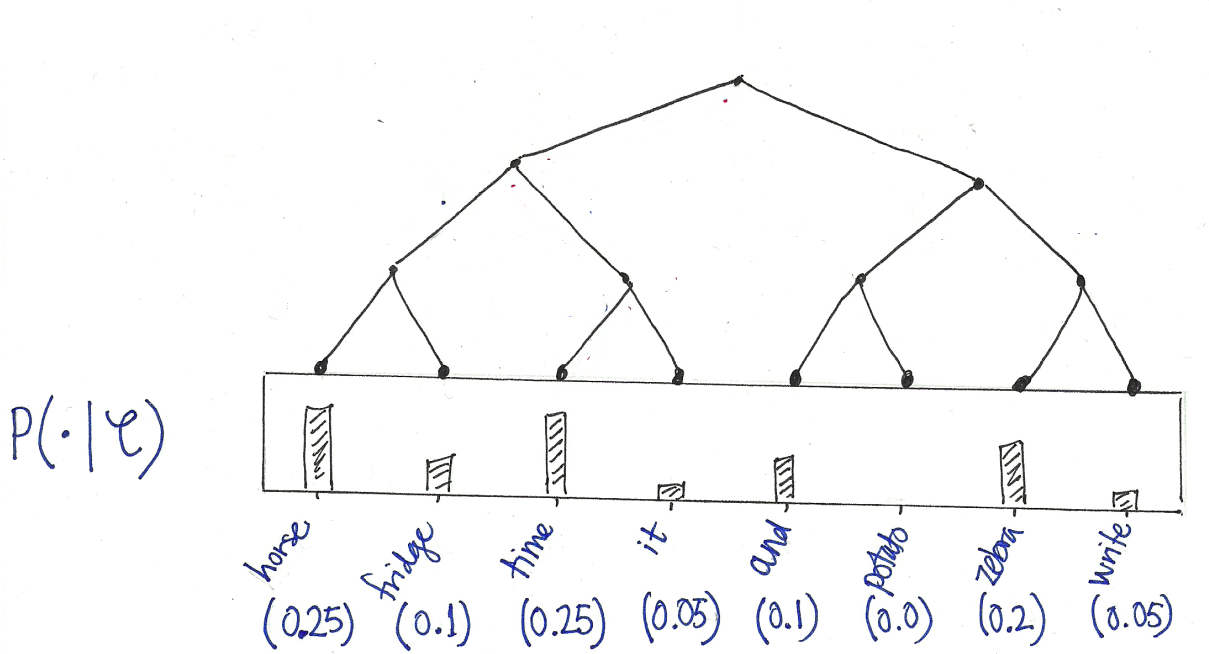
Word2Vec: Hierarchical Softmax

Word2Vec: Hierarchical Softmax

Идея: вместо «плоского» softmax, вычисляемого за $O(n)$, построить дерево и вычислять только нужные узлы за $O(\log n)$.

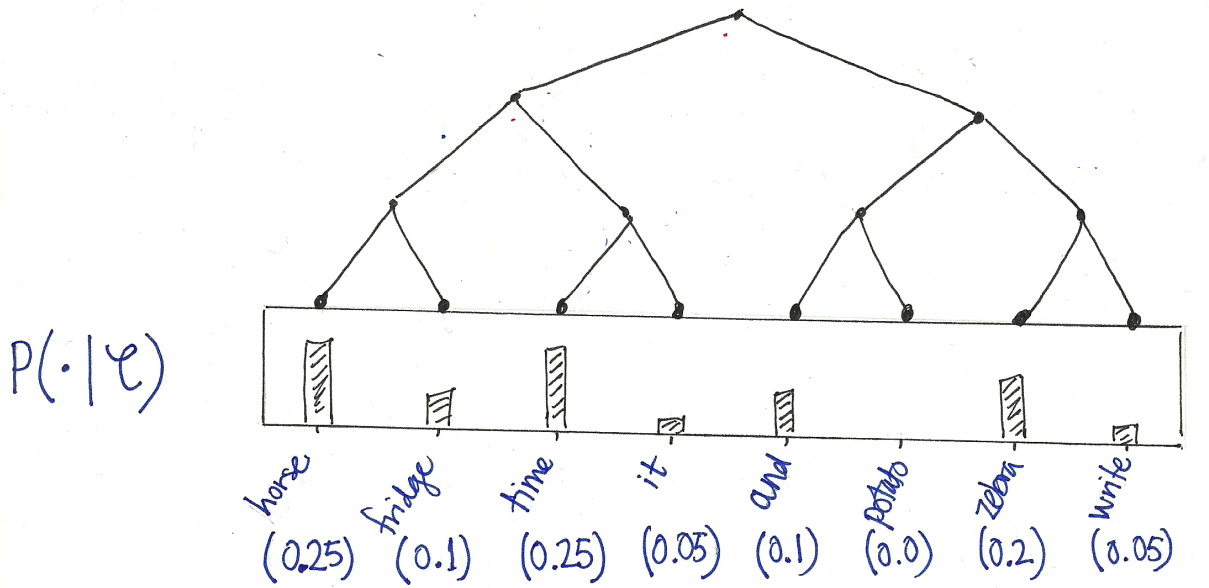
Word2Vec: Hierarchical Softmax

Идея: вместо «плоского» softmax, вычисляемого за $O(n)$, построить дерево и вычислять только нужные узлы за $O(\log n)$.



Word2Vec: Hierarchical Softmax

Идея: вместо «плоского» softmax, вычисляемого за $O(n)$, построить дерево и вычислять только нужные узлы за $O(\log n)$.



Усовершенствования:

- размещать похожие слова в соседних узлах
- Huffman tree — минимизация путей до вершин исходя из частотности слов

Word2Vec: Negative Sampling

Word2Vec: Negative Sampling

На каждом шаге обучения мы должны сделать обновление параметров так, чтобы:

- выросло значение вероятности для наблюдаемого слова
- понизилось значение вероятности для всех остальных слов

Word2Vec: Negative Sampling

На каждом шаге обучения мы должны сделать обновление параметров так, чтобы:

- выросло значение вероятности для наблюдаемого слова
- понизилось значение вероятности для всех остальных слов

Чтобы посчитать вторую часть, нужно просмотреть все слова и посчитать обновление на каждом слове.

Word2Vec: Negative Sampling

На каждом шаге обучения мы должны сделать обновление параметров так, чтобы:

- выросло значение вероятности для наблюдаемого слова
- понизилось значение вероятности для всех остальных слов

Чтобы посчитать вторую часть, нужно просмотреть все слова и посчитать обновление на каждом слове.

Идея: давайте оценивать общее изменение не на всей выборке, а на подвыборке (а-ля SGD).

- для этого набираем «негативных» примеров, т.е. случайных слов не из контекста

GloVe

GloVe

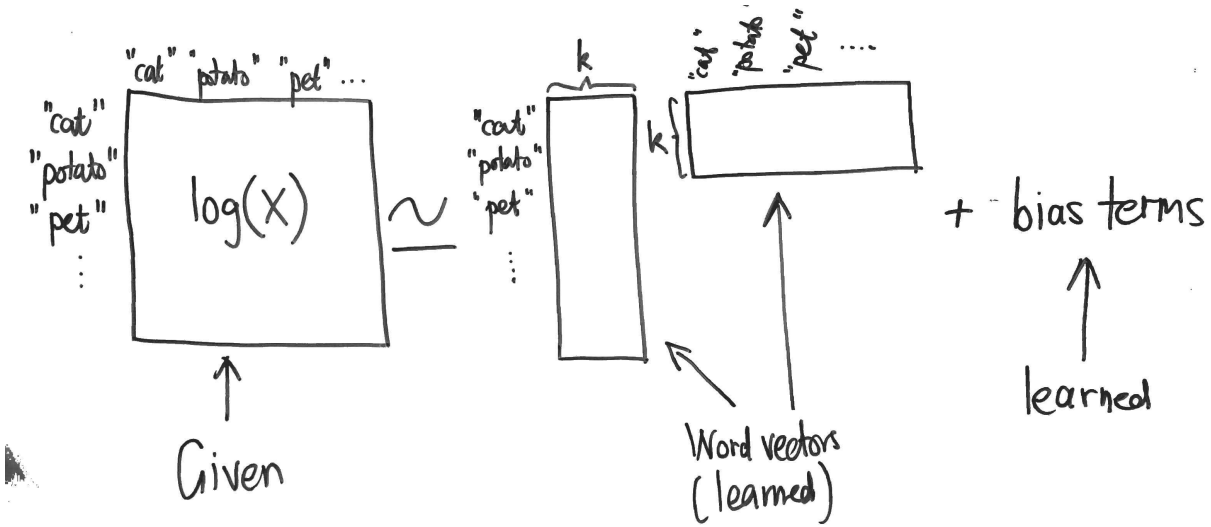
Идея: найдём наилучшую факторизацию матрицы совместной встречаемости слов, оптимизируя функции ошибки.

Идея: найдём наилучшую факторизацию матрицы совместной встречаемости слов, оптимизируя функции ошибки.

- $J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$
- X_{ij} — матрица совместных вхождений слов i и j в корпус
- $f(X_{ij})$ — весовая функция для сглаживания слишком частых и слишком редких сочетаний
- $w_i, \tilde{w}_j, b_i, \tilde{b}_j$ — обучаемые параметры

Идея: найдём наилучшую факторизацию матрицы совместной встречаемости слов, оптимизируя функции ошибки.

- $J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$
- X_{ij} — матрица совместных вхождений слов i и j в корпус
- $f(X_{ij})$ — весовая функция для сглаживания слишком частых и слишком редких сочетаний
- $w_i, \tilde{w}_j, b_i, \tilde{b}_j$ — обучаемые параметры



FastText

Идея: почему бы не вычислять вектора не только для целых слов, но и для символьных N-грамм, входящих в них?

Идея: почему бы не вычислять вектора не только для целых слов, но и для символьных N-грамм, входящих в них?

Пример: $\mathcal{G}_{words} = \{wor, ord, rds, word, ords, words\}$

Идея: почему бы не вычислять вектора не только для целых слов, но и для символьных N-грамм, входящих в них?

Пример: $\mathcal{G}_{words} = \{wor, ord, rds, word, ords, words\}$

skip-gram similarity: $s(w_t, w_c) = \boldsymbol{u}_{w_t}^\top \boldsymbol{v}_{w_c}$
FastText similarity: $s(w_t, w_c) = \sum_{g \in \mathcal{G}_{w_t}} \boldsymbol{z}_g^\top \boldsymbol{v}_c$

Идея: почему бы не вычислять вектора не только для целых слов, но и для символьных N-грамм, входящих в них?

Пример: $\mathcal{G}_{words} = \{wor, ord, rds, word, ords, words\}$

skip-gram similarity: $s(w_t, w_c) = \boldsymbol{u}_{w_t}^\top \boldsymbol{v}_{w_c}$
FastText similarity: $s(w_t, w_c) = \sum_{g \in \mathcal{G}_{w_t}} \boldsymbol{z}_g^\top \boldsymbol{v}_c$

Плюсы:

- борьба с опечатками
- грамматическое vs семантическое сходство
- векторы для слов, не встречающихся в корпусе

Word embeddings: Further reading

Word embeddings: Further reading

Как измерить качество модели?

- Внутренние критерии — perplexity и т.п.
- «Золотые» списки синонимов / аналогий
- Внешние критерии — в зависимости от решаемой задачи

Word embeddings: Further reading

Как измерить качество модели?

- Внутренние критерии — perplexity и т.п.
- «Золотые» списки синонимов / аналогий
- Внешние критерии — в зависимости от решаемой задачи

Как улучшить качество модели?

- увеличить объём обучающих данных
- настроить гиперпараметры (размерность вектора, размер окна и т.д.)
- попробовать обучать по-другому:

Word embeddings: Further reading

Как измерить качество модели?

- Внутренние критерии — perplexity и т.п.
- «Золотые» списки синонимов / аналогий
- Внешние критерии — в зависимости от решаемой задачи

Как улучшить качество модели?

- увеличить объём обучающих данных
- настроить гиперпараметры (размерность вектора, размер окна и т.д.)
- попробовать обучать по-другому:
  - учитывать расстояние между словами в контексте
  - предобработать / разметить корпус
  - придумать что-нибудь новое и опубликовать статью

Word embeddings: Applications

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Рис. 1: Построение аналогий

Word embeddings: Applications

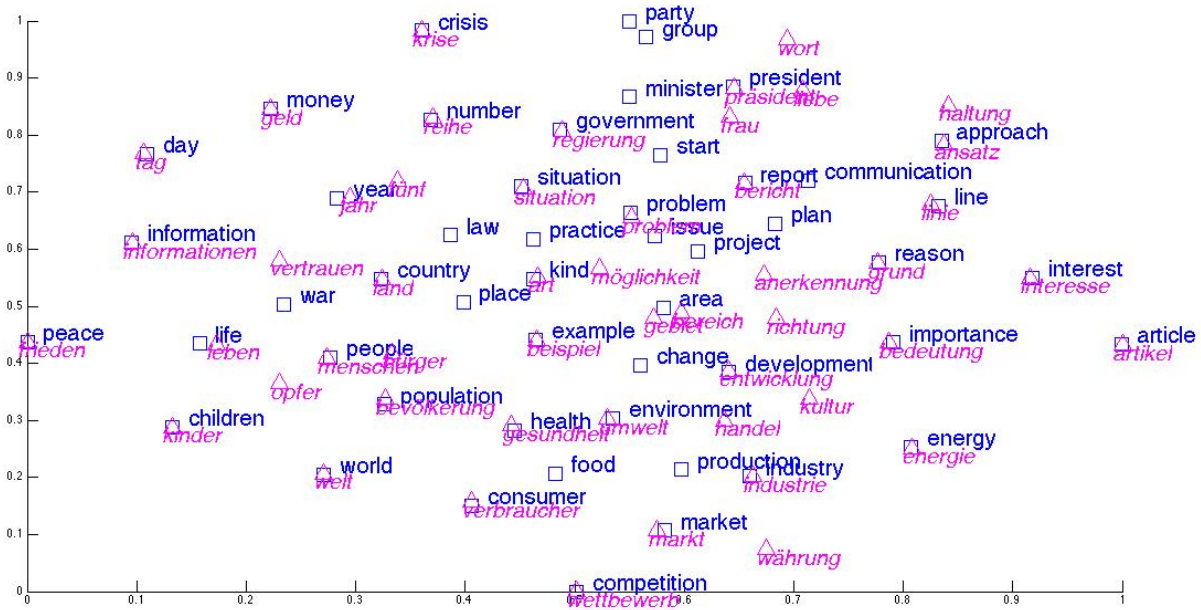


Рис. 2: Bilingual embeddings

Word embeddings: Applications

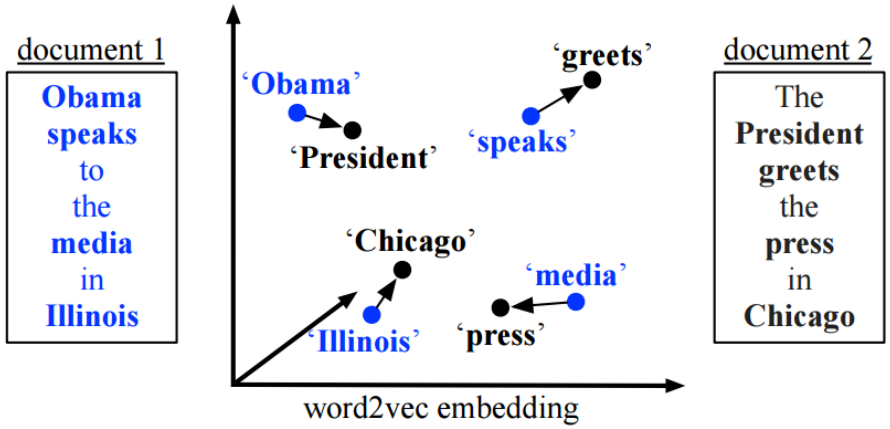


Figure 1. An illustration of the *word mover's distance*. All non-stop words (***bold***) of both documents are embedded into a *word2vec* space. The distance between the two documents is the minimum cumulative distance that all words in document 1 need to travel to exactly match document 2. (Best viewed in color.)

Рис. 3: Word Mover's Distance

Image embeddings: Чего мы хотим?

Image embeddings: Чего мы хотим?

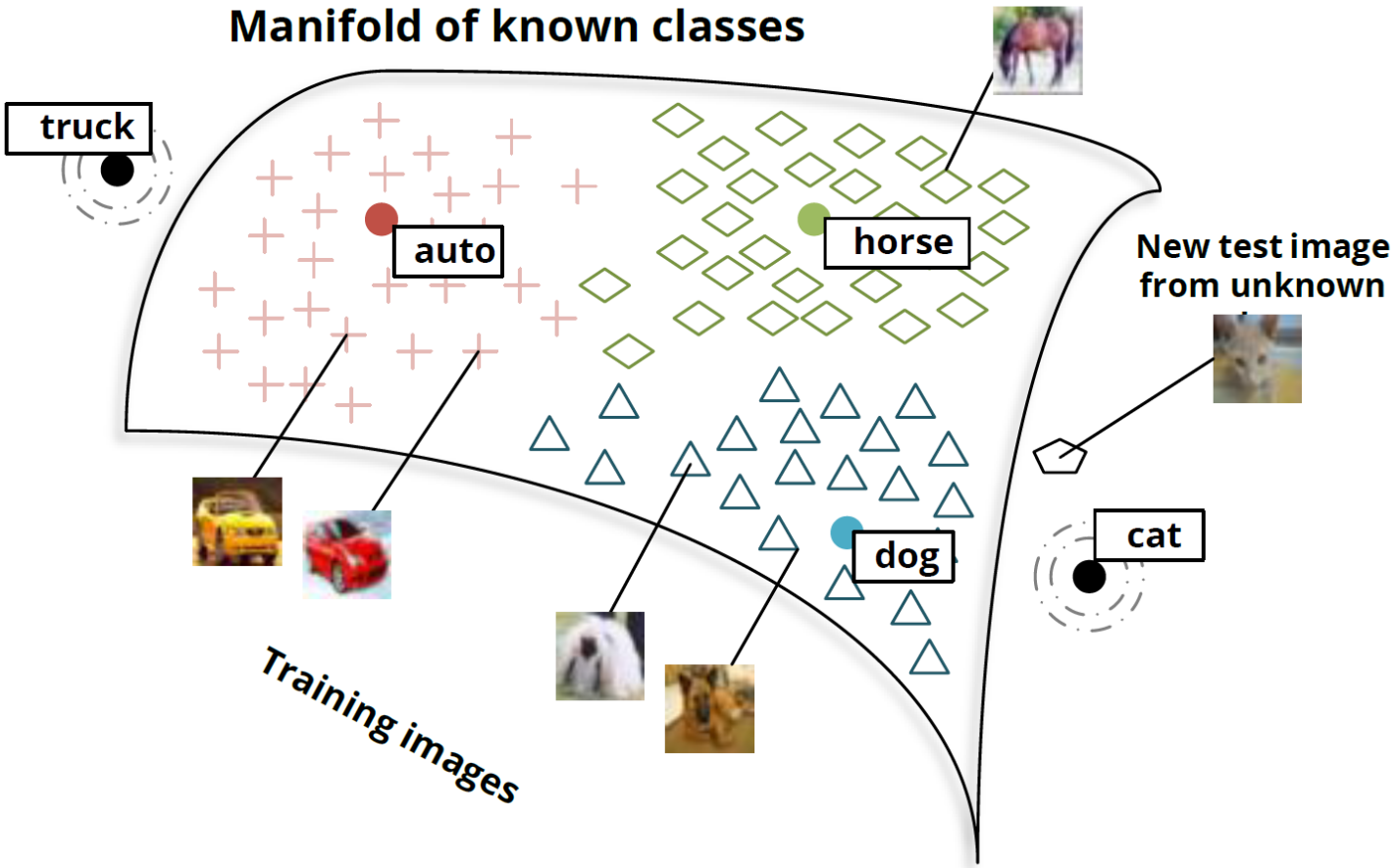
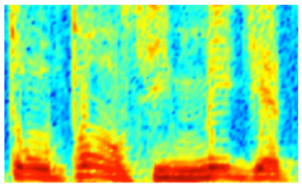


Image embeddings: Зачем?

Image embeddings: Зачем?

AUDIO



Audio Spectrogram

DENSE

IMAGES

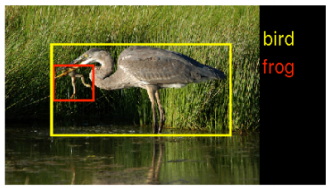
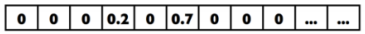


Image pixels

DENSE

TEXT

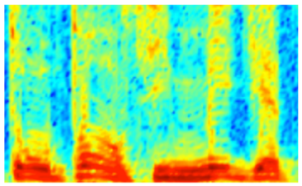


Word, context, or document vectors

SPARSE

Image embeddings: Зачем?

AUDIO



Audio Spectrogram

DENSE

IMAGES

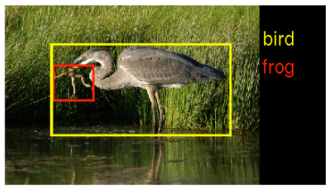


Image pixels

DENSE

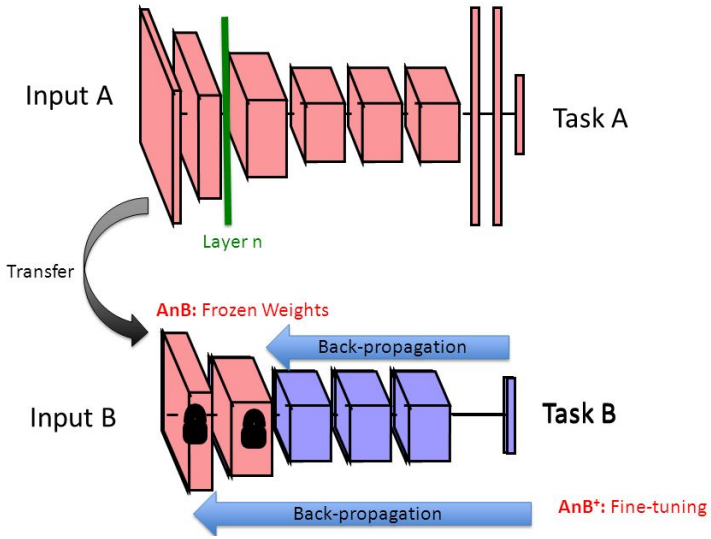
TEXT

0	0	0	0.2	0	0.7	0	0	0	...	...

Word, context, or document vectors

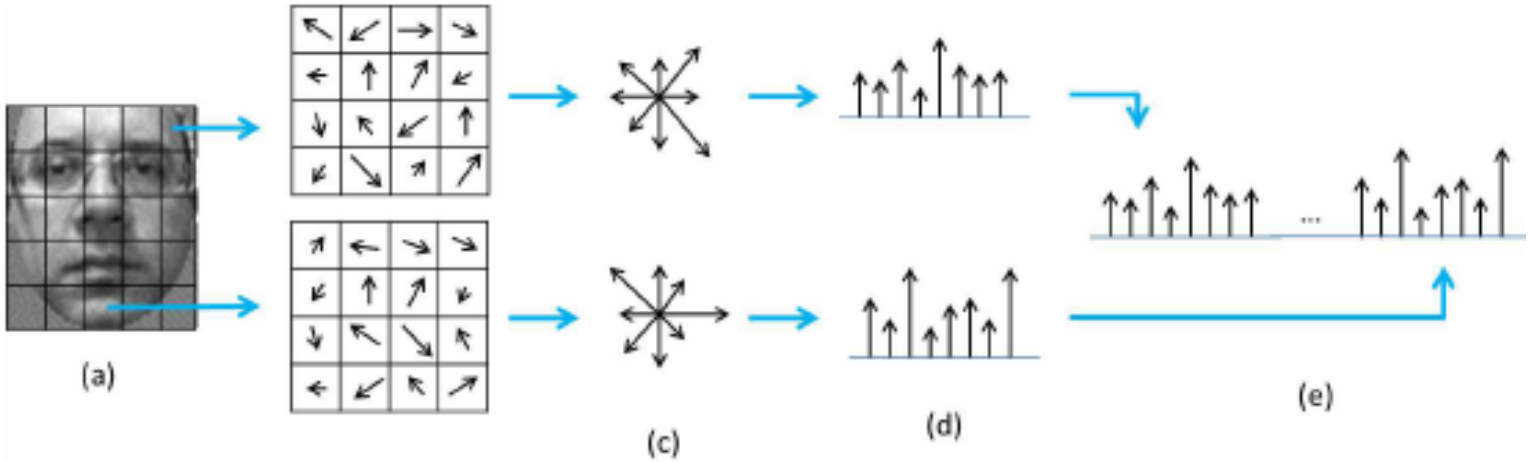
SPARSE

Transfer Learning Overview

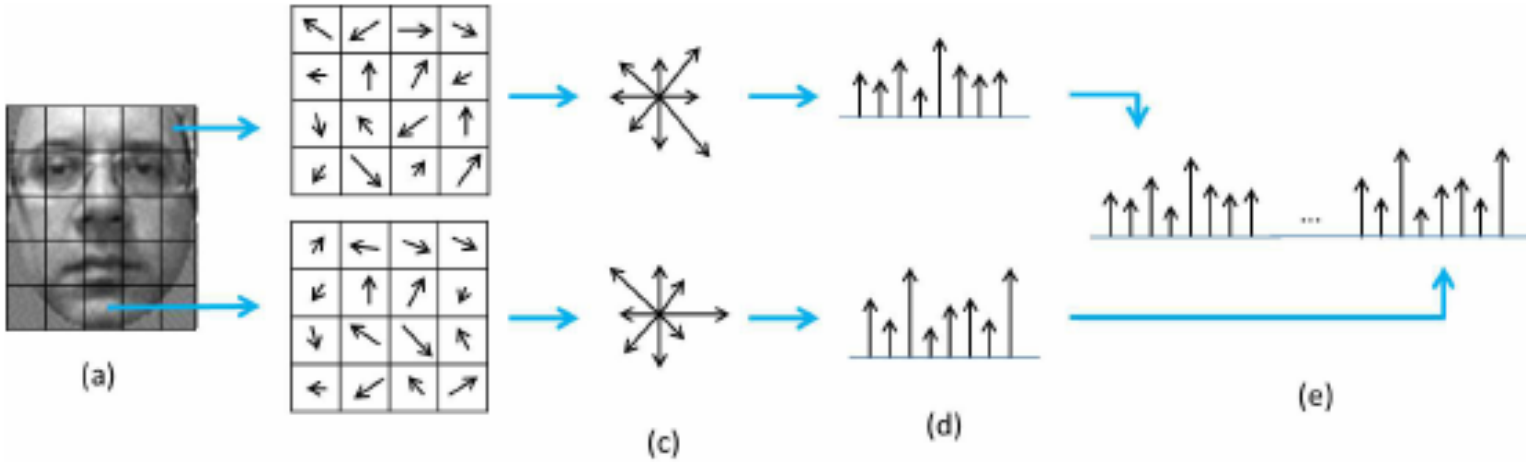


Hand-crafted image embeddings: Histogram of Oriented Gradients

Hand-crafted image embeddings: Histogram of Oriented Gradients



Hand-crafted image embeddings: Histogram of Oriented Gradients



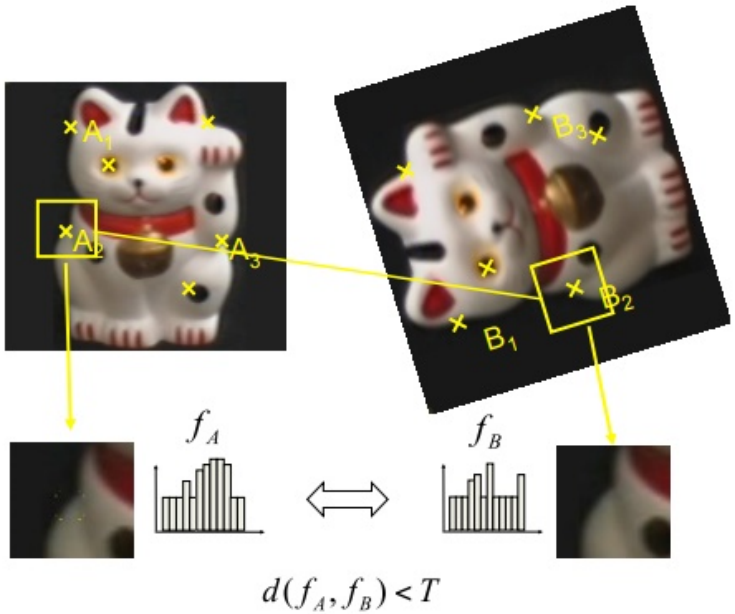


Image embeddings: Функции потерь

Image embeddings: Функции потерь

- **Classification losses** (*hinge, cross-entropy*) — просто обучаем классификатор и смотрим на предпоследний слой

Image embeddings: Функции потерь

- **Classification losses** (*hinge, cross-entropy*) — просто обучаем классификатор и смотрим на предпоследний слой
- **Pairwise losses** (*pairwise hinge, contrastive, KL divergence, histogram*) — составляем пары объектов
  - если объекты одного класса — сближаем векторы
  - если объекты разных классов — разводим векторы

Image embeddings: Функции потерь

- **Classification losses** (*hinge, cross-entropy*) — просто обучаем классификатор и смотрим на предпоследний слой
- **Pairwise losses** (*pairwise hinge, contrastive, KL divergence, histogram*) — составляем пары объектов
  - если объекты одного класса — сближаем векторы
  - если объекты разных классов — разводим векторы
- **Triplet losses** (*original triplet, triplet with global, distance ratio, large margin nearest neighbor*) – составляем тройки объектов x_0, x_+, x_- и увеличиваем $d(x_0, x_-) - d(x_0, x_+)$

Image embeddings: Функции потерь

- **Classification losses** (*hinge, cross-entropy*) — просто обучаем классификатор и смотрим на предпоследний слой
- **Pairwise losses** (*pairwise hinge, contrastive, KL divergence, histogram*) — составляем пары объектов
  - если объекты одного класса — сближаем векторы
  - если объекты разных классов — разводим векторы
- **Triplet losses** (*original triplet, triplet with global, distance ratio, large margin nearest neighbor*) – составляем тройки объектов x_0, x_+, x_- и увеличиваем $d(x_0, x_-) - d(x_0, x_+)$
- **Quadruplet losses**:
  - составляем четвёрки объектов x_i, x_j, x_k, x_l, $y_i = y_j, y_i \neq y_k, y_l, y_k \neq y_l$, увеличиваем $d(x_i, x_{kl}) - d(x_i, x_j)$ и $d(x_k, x_l) - d(x_i, x_j)$
  - если есть ранжирование классов, то разводим векторы наиболее удалённых классов и сближаем векторы «средних» классов

Image embeddings: Triplet Loss

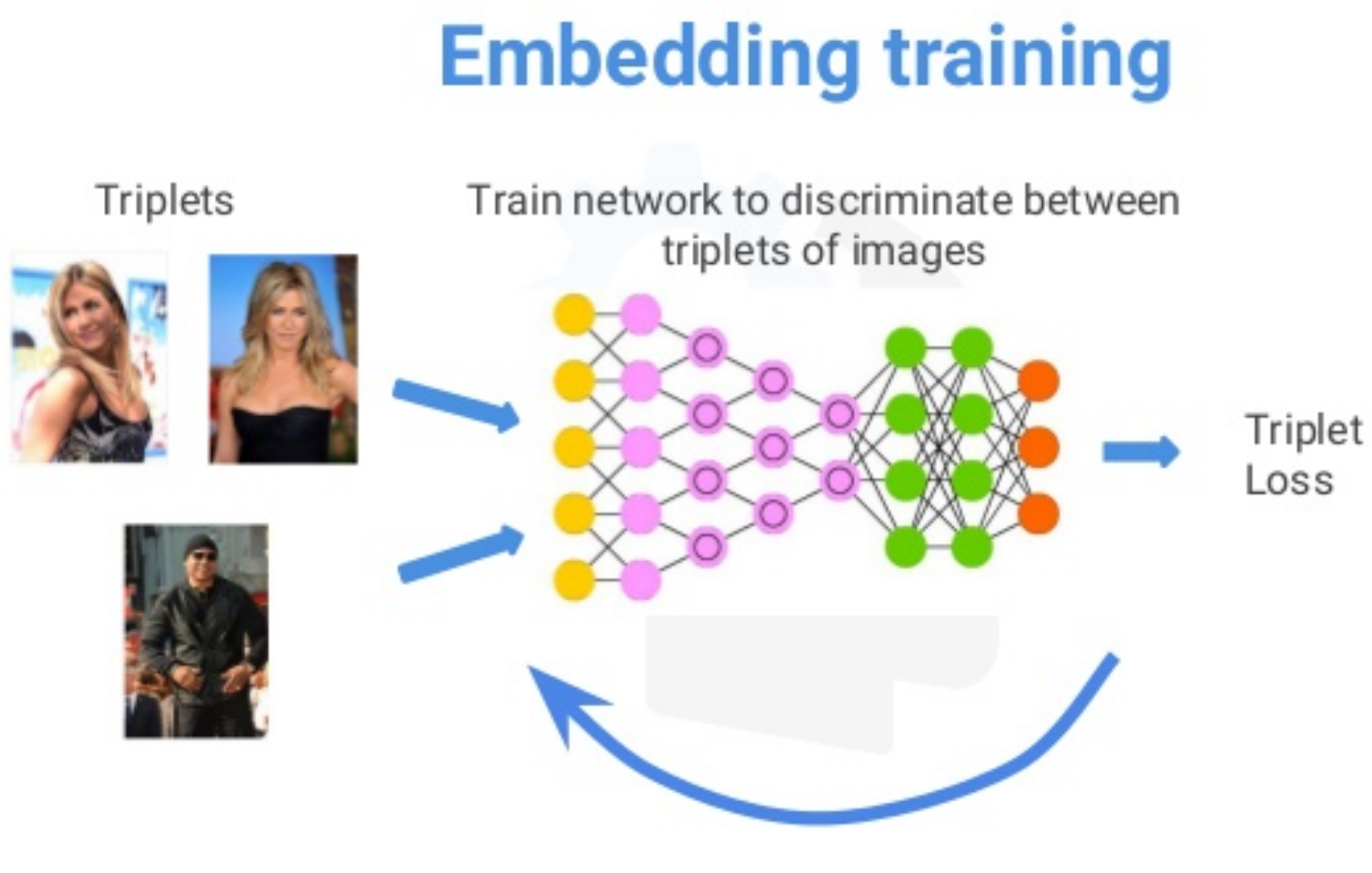


Image embeddings: Triplet Loss

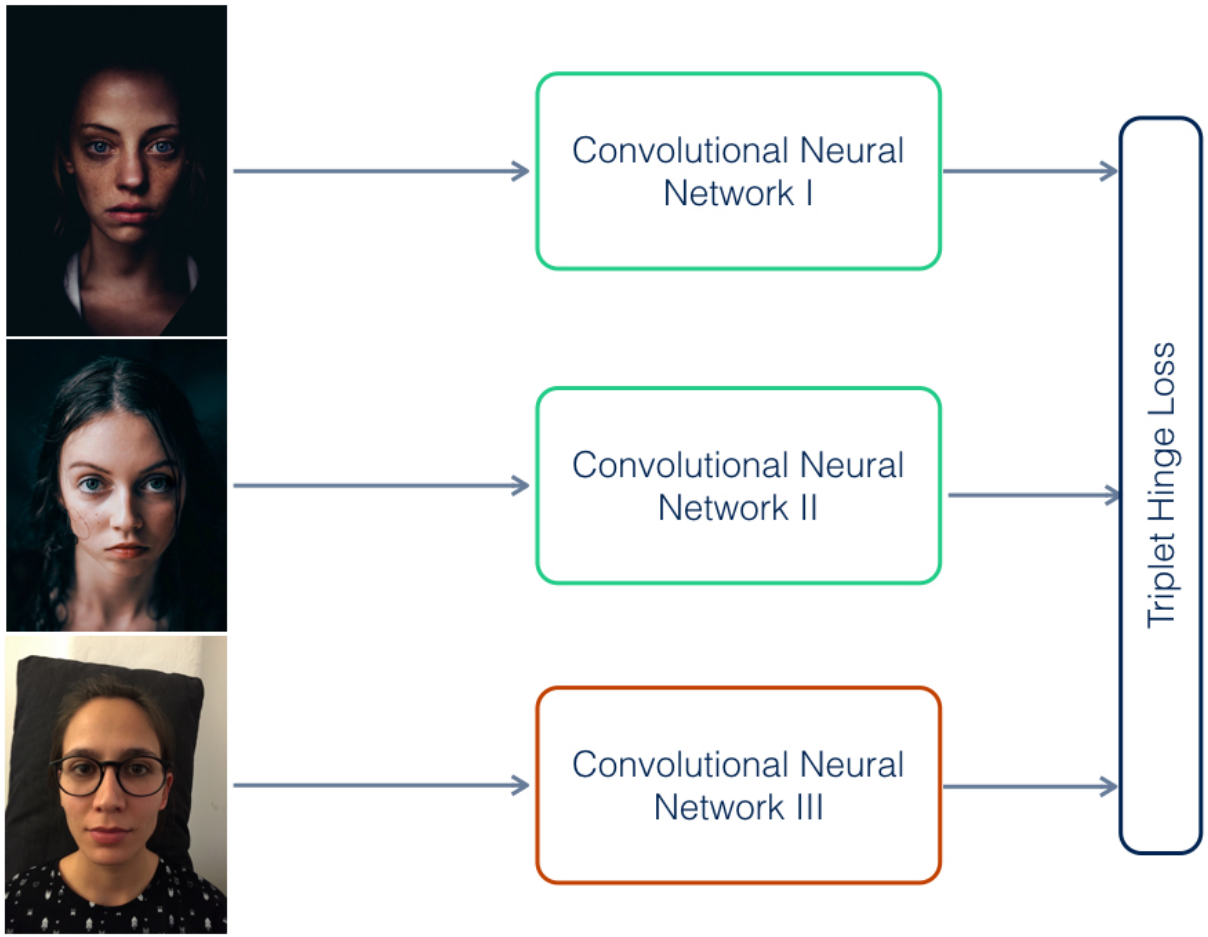


Image embeddings: Quadruplet Loss

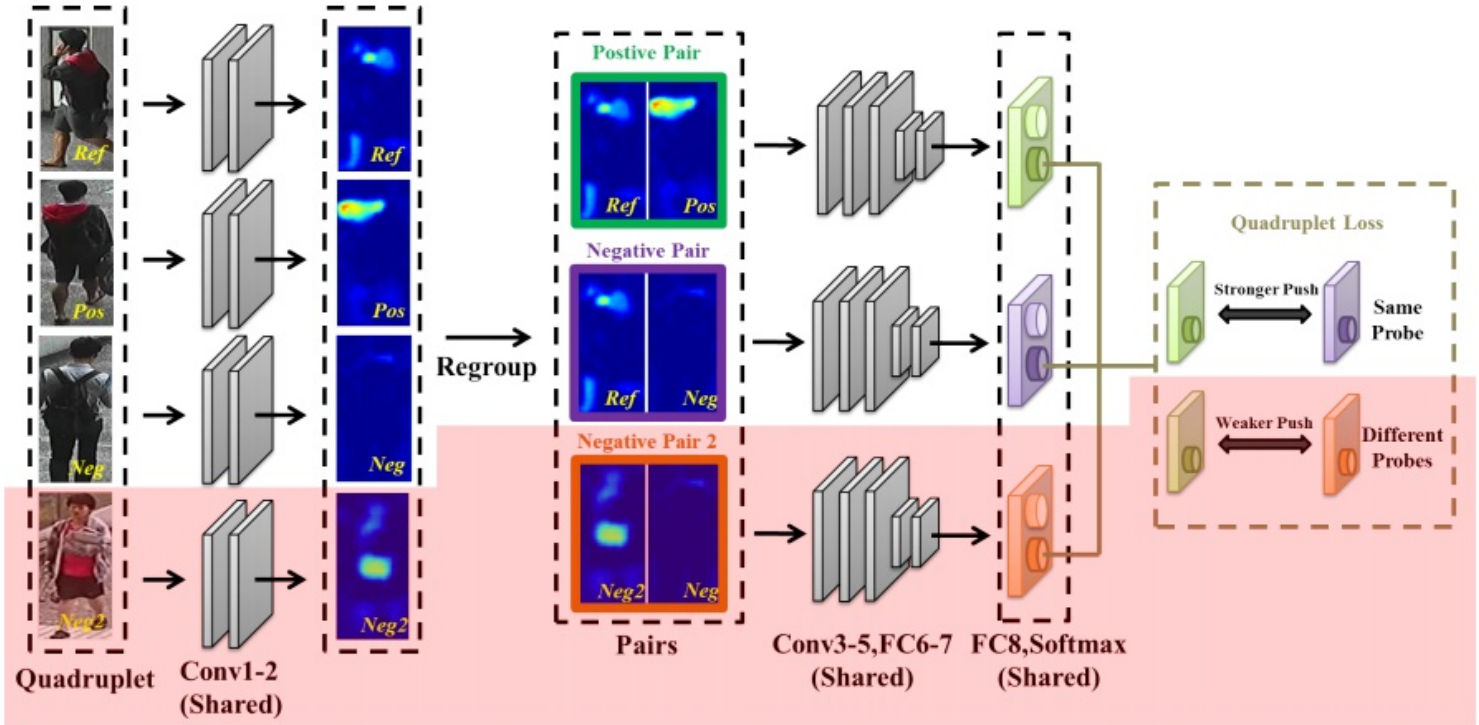


Figure 3. The framework of the proposed quadruplet deep network. The red shadow region indicates elements of the new constraint.

Image embeddings: Quadruplet Loss

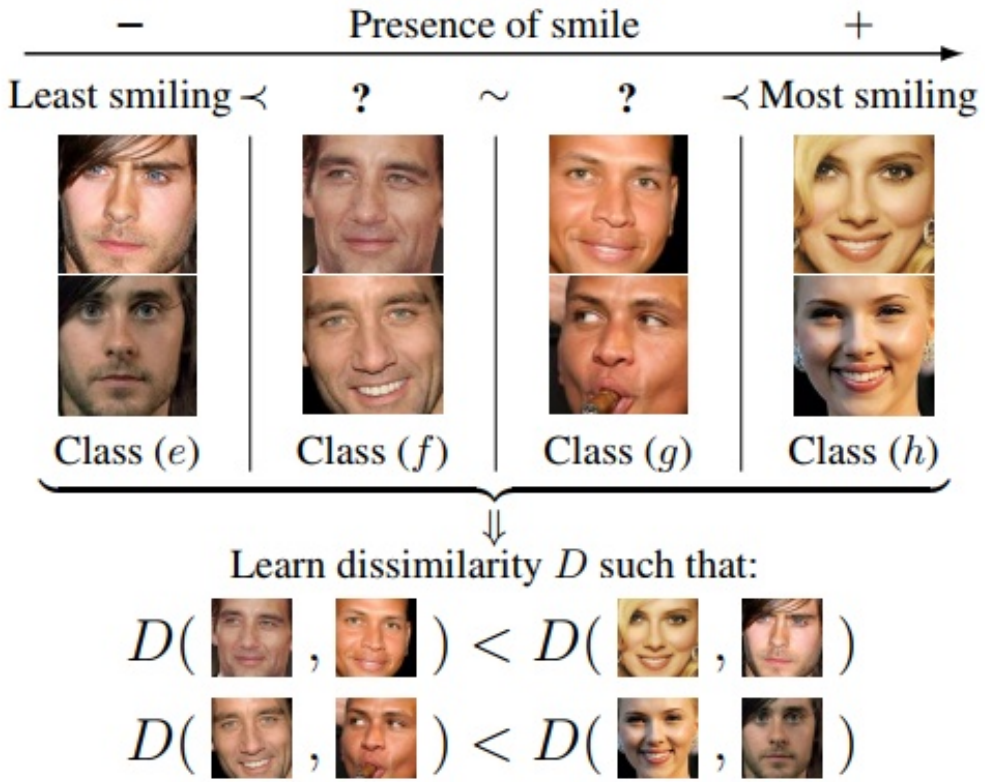


Figure 1. Quadruplet-wise (Qwise) strategy on 4 face classes ranked according to the degree of presence of smile. Instead of working on pairwise relations that present some flaws (see text), Qwise strategy defines quadruplet-wise constraints to express that dissimilarities between examples from (f) and (g) should be smaller than dissimilarities between examples from (e) and (h).

Image embeddings: Histogram Loss

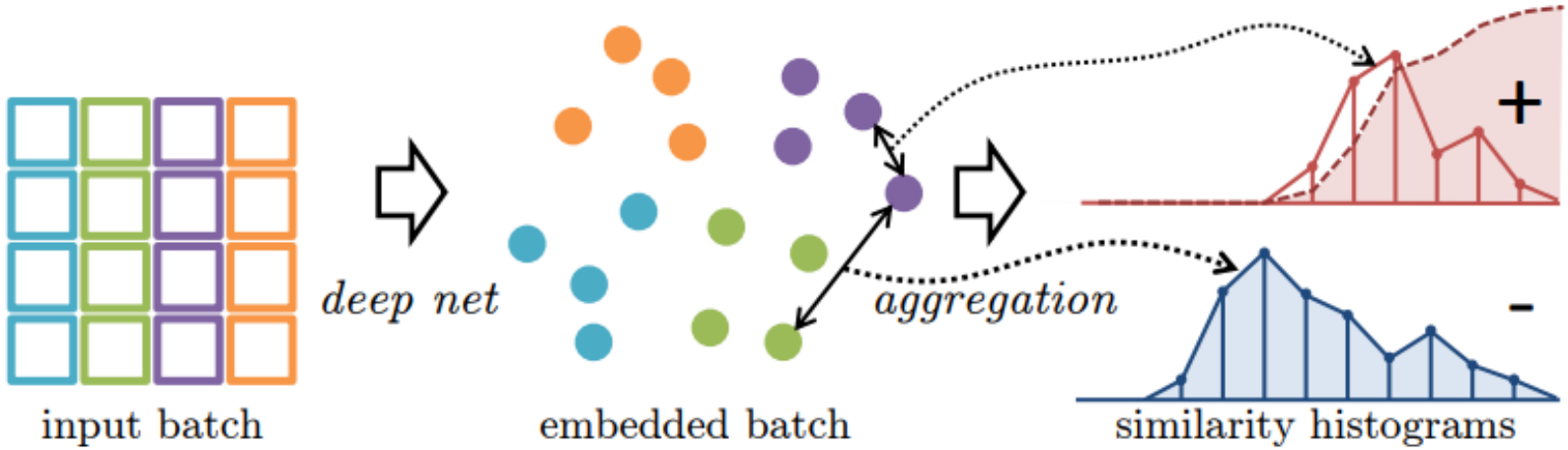
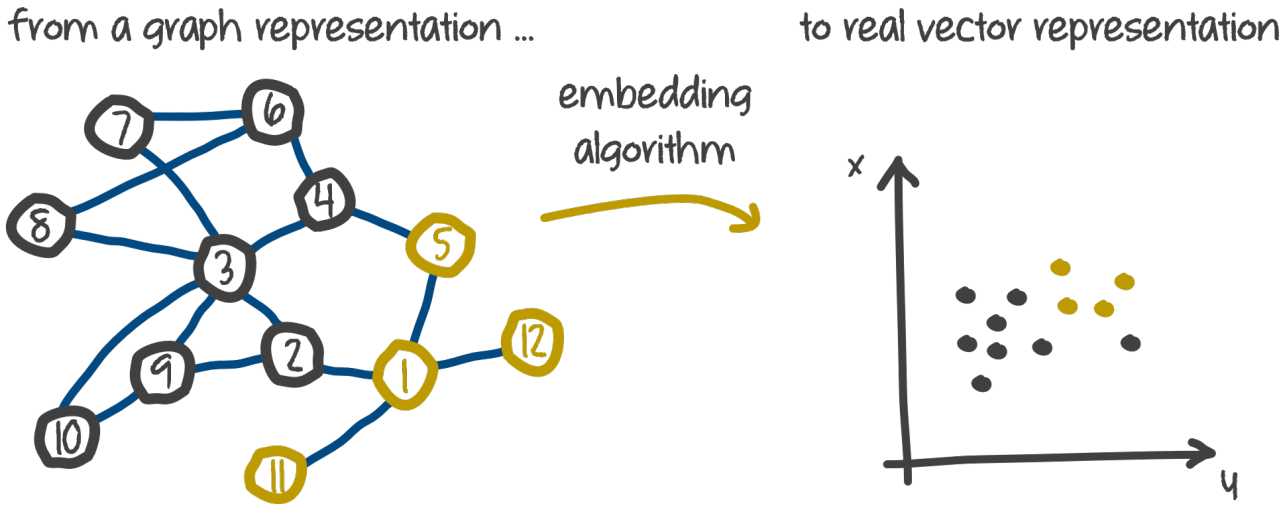


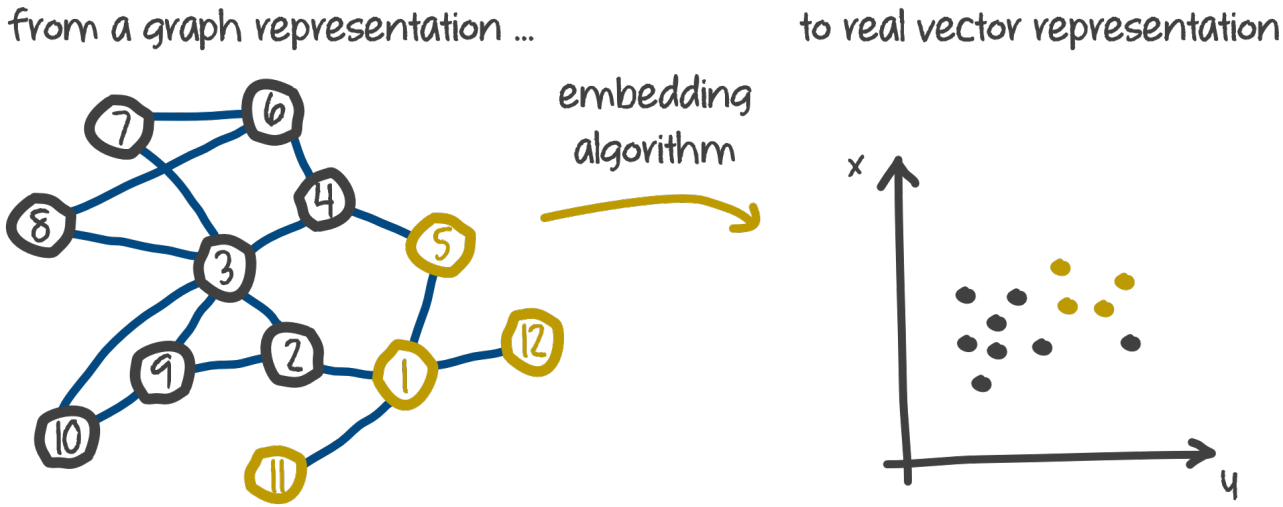
Figure 1: The histogram loss computation for a batch of examples (color-coded; same color indicates matching samples). After the batch (left) is embedded into a high-dimensional space by a deep network (middle), we compute the histograms of similarities of positive (top-right) and negative pairs (bottom-right). We then evaluate the integral of the product between the negative distribution and the cumulative density function for the positive distribution (shown with a dashed line), which corresponds to a probability that a randomly sampled positive pair has smaller similarity than a randomly sampled negative pair. Such histogram loss can be minimized by backpropagation. The only associated parameter of such loss is the number of histogram bins, to which the results have very low sensitivity.

Graph embeddings

Graph embeddings

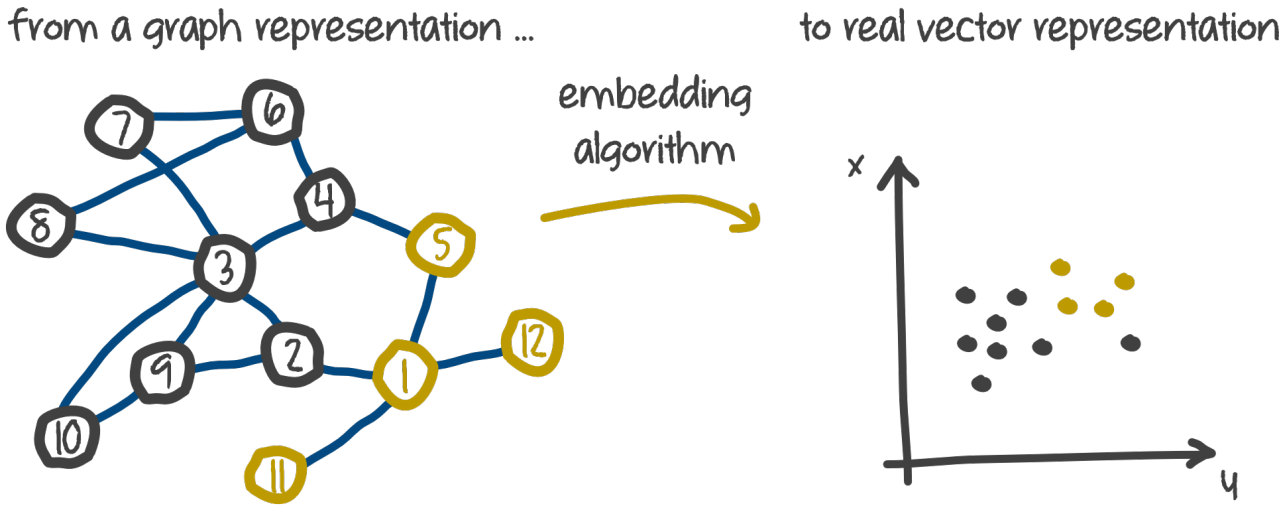


Graph embeddings



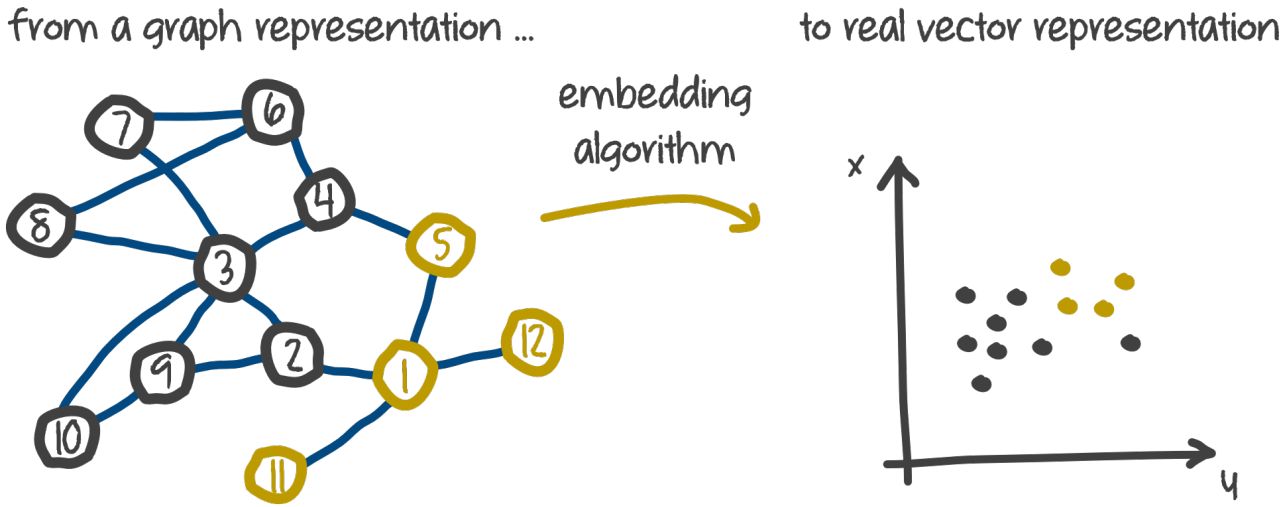
- Факторизация матрицы смежности (*Locally Linear Embedding, Graph Factorization*)

Graph embeddings



- Факторизация матрицы смежности (*Locally Linear Embedding, Graph Factorization*)
- Случайное блуждание (*DeepWalk, node2vec*)

Graph embeddings



- Факторизация матрицы смежности (*Locally Linear Embedding, Graph Factorization*)
- Случайное блуждание (*DeepWalk, node2vec*)
- Deep Learning (*SDNE*)

Multi-modal embeddings

Multi-modal embeddings

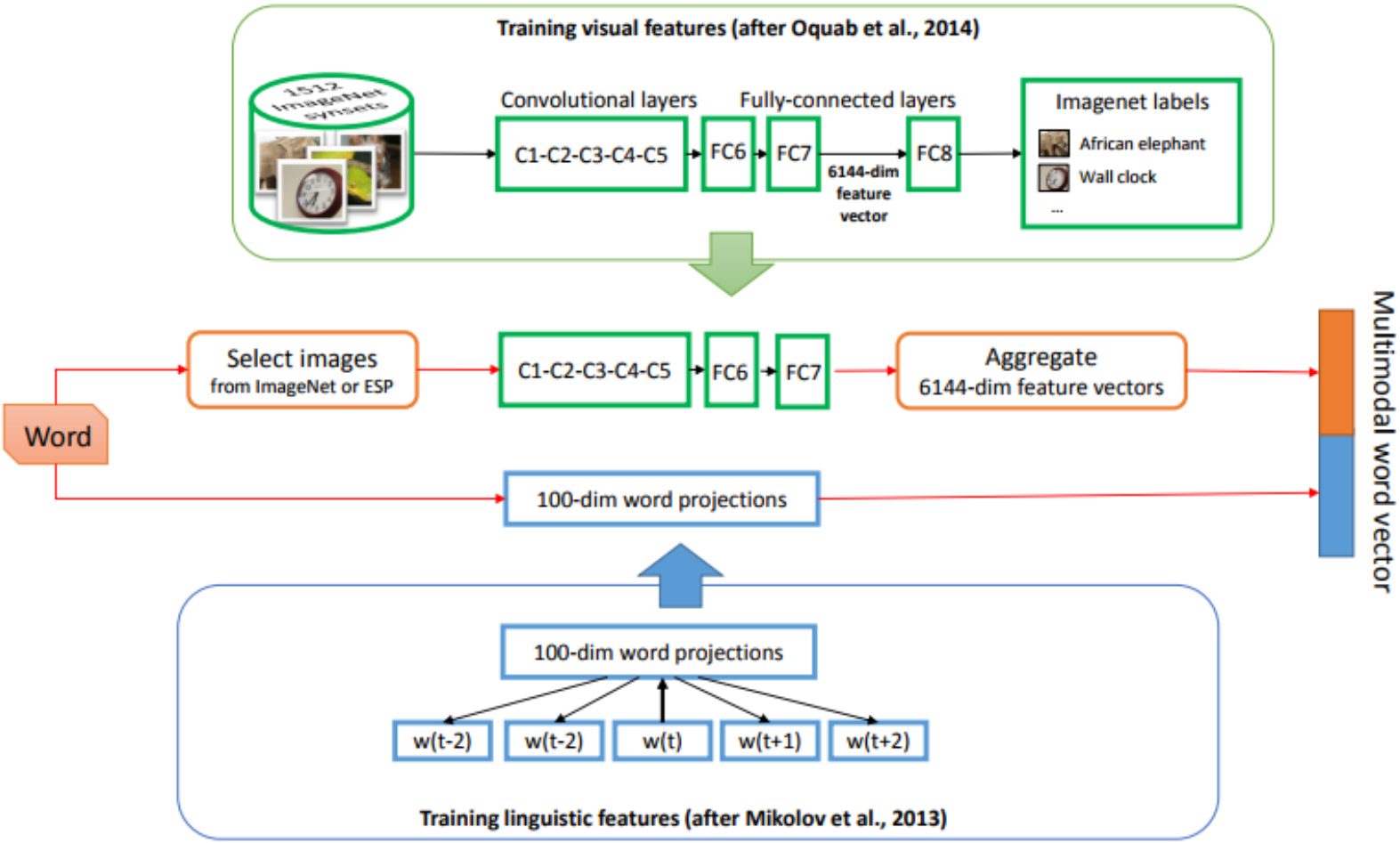


Figure 1: Computing word feature vectors.

Выводы

Выводы

1. Распределённые векторные представления удобнее для обработки ML-алгоритмами

Выводы

1. Распределённые векторные представления удобнее для обработки ML-алгоритмами
2. Если сильно захотеть, то можно векторизовать любой объект / понятие

Выводы

1. Распределённые векторные представления удобнее для обработки ML-алгоритмами
2. Если сильно захотеть, то можно векторизовать любой объект / понятие
3. Обучение эмбеддингов бывает с учителем и без учителя

Выводы

1. Распределённые векторные представления удобнее для обработки ML-алгоритмами
2. Если сильно захотеть, то можно векторизовать любой объект / понятие
3. Обучение эмбеддингов бывает с учителем и без учителя
4. Основное искусство — выбор функции потерь

Выводы

1. Распределённые векторные представления удобнее для обработки ML-алгоритмами
2. Если сильно захотеть, то можно векторизовать любой объект / понятие
3. Обучение эмбеддингов бывает с учителем и без учителя
4. Основное искусство — выбор функции потерь
5. Эмбеддинги из одной задачи можно использовать для решения других

Ссылки

- ▸ Word embeddings
- ▸ Natural Language Processing with Deep Learning, Stanford
- ▸ Deep image embeddings (various losses)

Спасибо за внимание!

alexey.romanov@phystech.edu