Lecture 3

Корбут Даниил
Deep Learning Research Engineer, Insilico Medicine

telegram: @rtriangle
vk: rtriangle
email: korbut.daniel@gmail.com

Оценка за курс

- 5 дз по +10 баллов (55% оценки)
- итоговый проект +10 баллов (30% оценки)
- работа на семинарах и задания в ноутбуках с семинаров (15% оценки)
- бонусы +inf (inf%)

Проект Kaggle

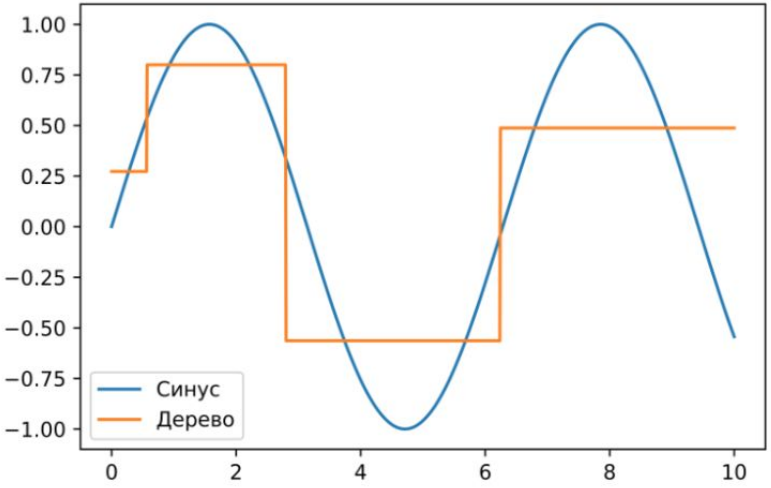
- Описание данных
- .ipynb с экспериментами
  - exploratory data analysis
  - генерация признаков
  - разбиение train-dev
  - эксперименты с моделями
  - финальный сабмит
  - идеи для улучшения
- Ссылка на соревнование

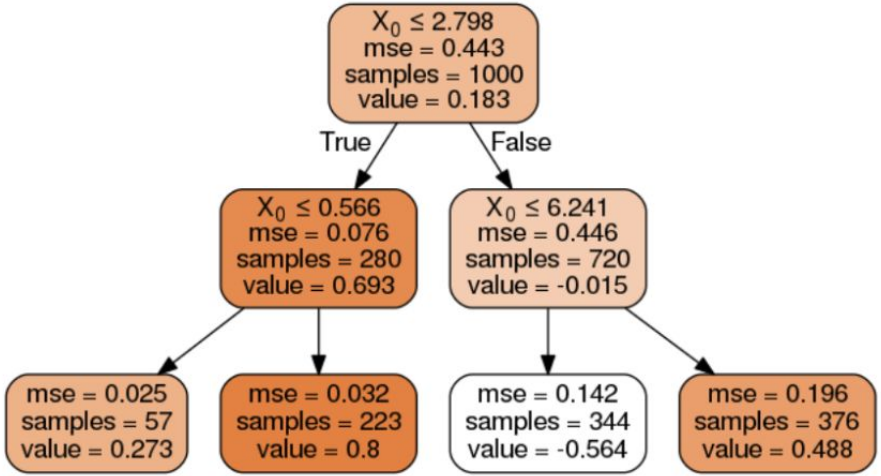
http://www.shivambansal.com/blog/kaggle-bot/

План занятия

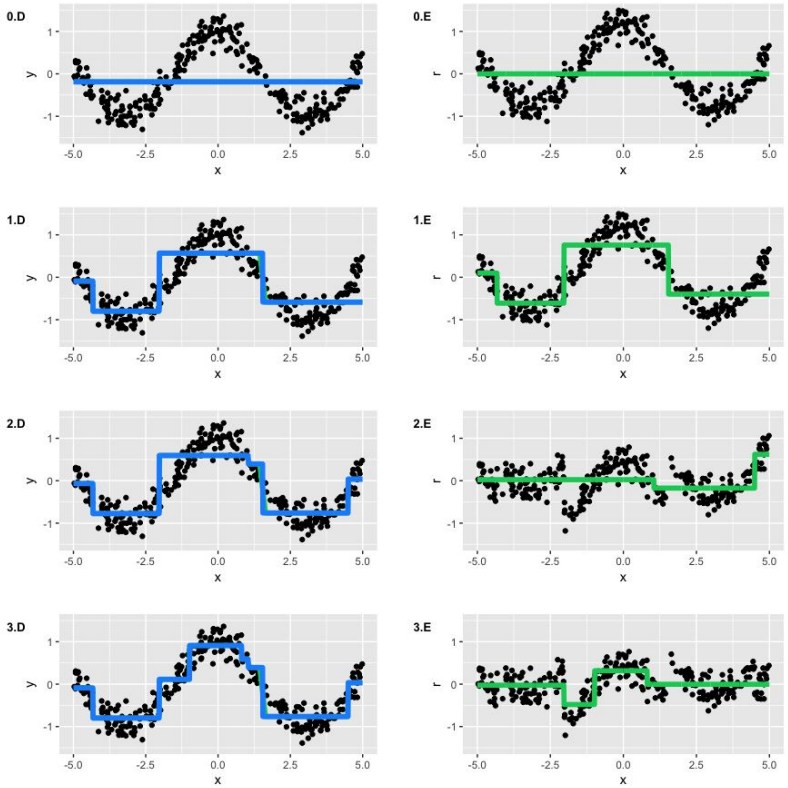
- Напоминание с прошлой лекции
- Ансамбли произвольных моделей
- Основы NLP: Word Embeddings, Language Model, RNN, LSTM, GRU

Напоминание с прошлой лекции



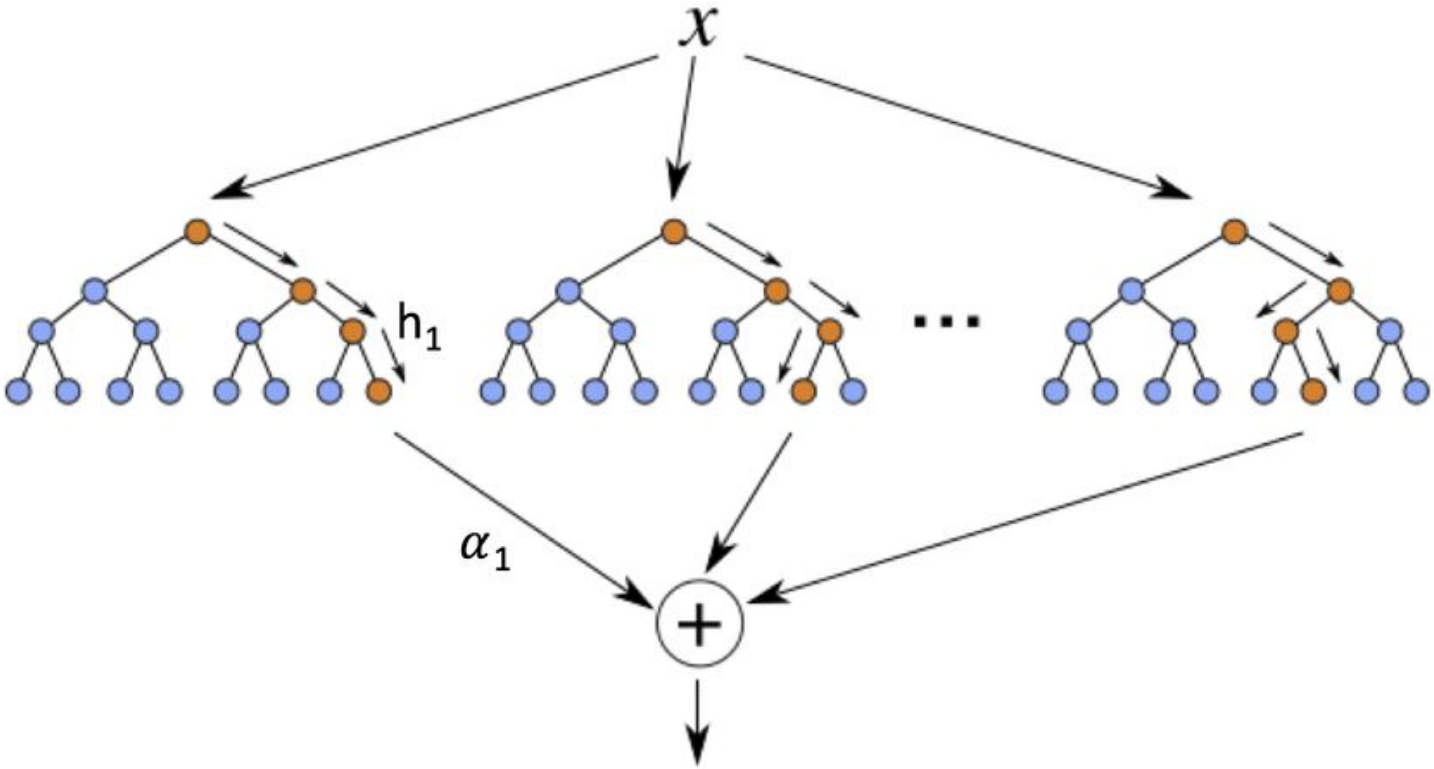


Напоминание с прошлой лекции



https://www.pvsm.ru/matematika/255559

Напоминание с прошлой лекции



https://www.vaishalilambe.com/blog/data-science-algorithms-random-forest

Ансамбли - мотивация

Предположим, вы хотите купить новые наушники.

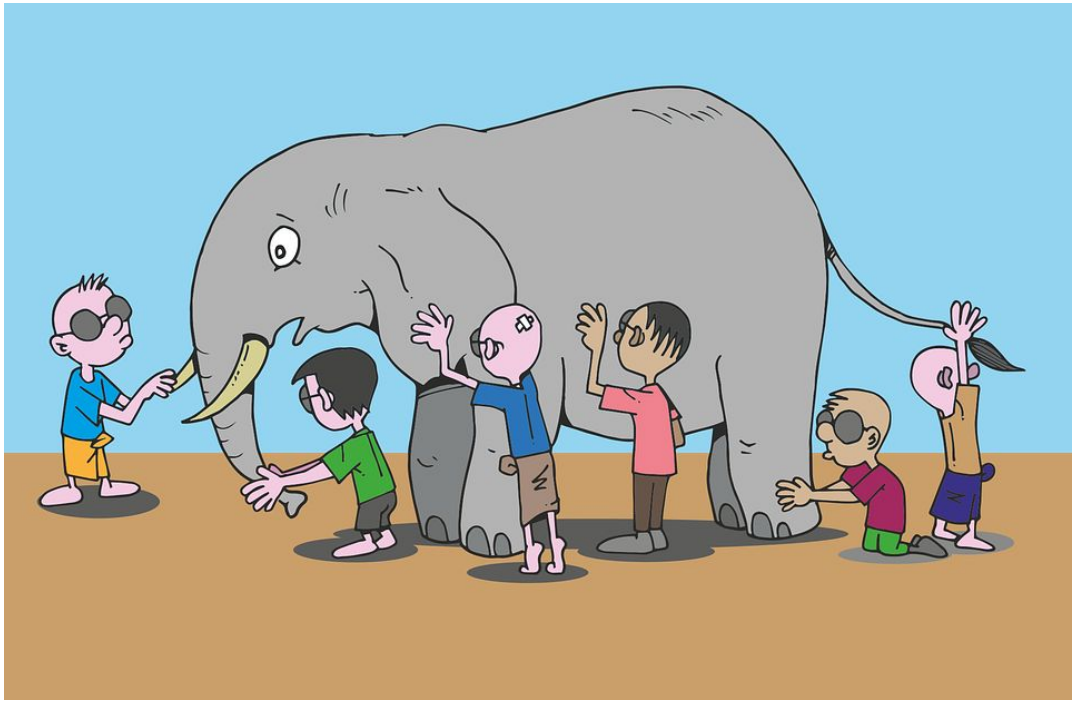
- Пойдёте ли вы в первый магазин и купите первые попавшиеся наушники?
- А, может, вы пойдёте в первый магазин, изучите ассортимент и купите понравившиеся наушники?
- А, может, вы сравните цены на Яндекс.Маркет, выберите самый выгодный вариант и купите понравившиеся наушники?
- А, может, вы проведёте “исследование”, пообщаетесь с коллегами/друзьями, разбирающимися в наушниках, выберите самый выгодный вариант и купите понравившиеся наушники?

Ансамбли

Зачем нужны ансамбли?

Ансамблевые методы позволяют объединить решения многих моделей для улучшения общего качества предсказания.

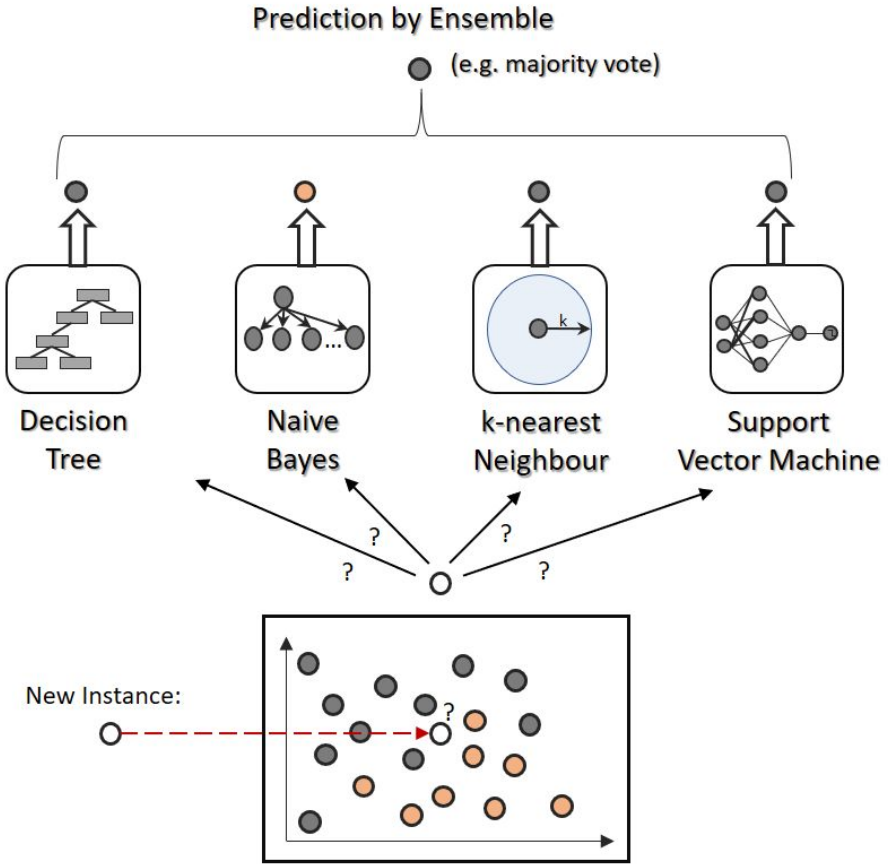
Bias, Variance, Noise



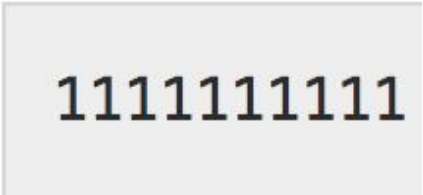
https://towardsdatascience.com/simple-guide-for-ensemble-learning-methods-d87cc68705a2

Ансамбли - простые техники

- Мода предсказаний
- Среднее предсказаний
- Взвешенное среднее предсказаний



Ансамбли - пример



Пусть есть 3 классификатора A, B, C с равными точностями 70%. Можно рассматривать, как псевдослучайные классификаторы, которые с p=0.7 выдают класс 1, с p=0.3 выдают класс 0.

Давайте покажем, как с помощью простого голосования можно получить итоговую ансамбль-модель с точностью 78%.

Ансамбли - пример

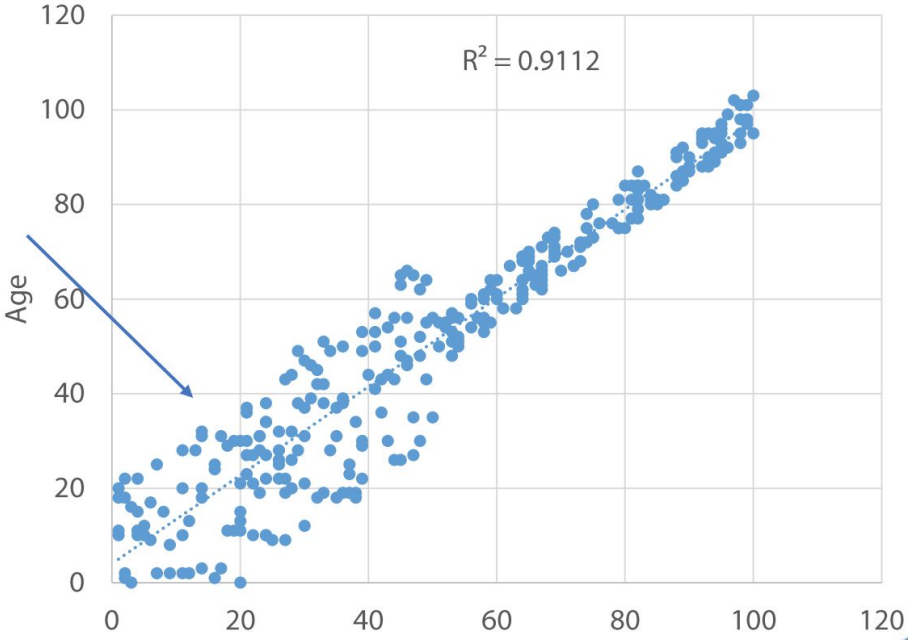
Для голосования среди 3 моделей можно ожидать 4 варианта:

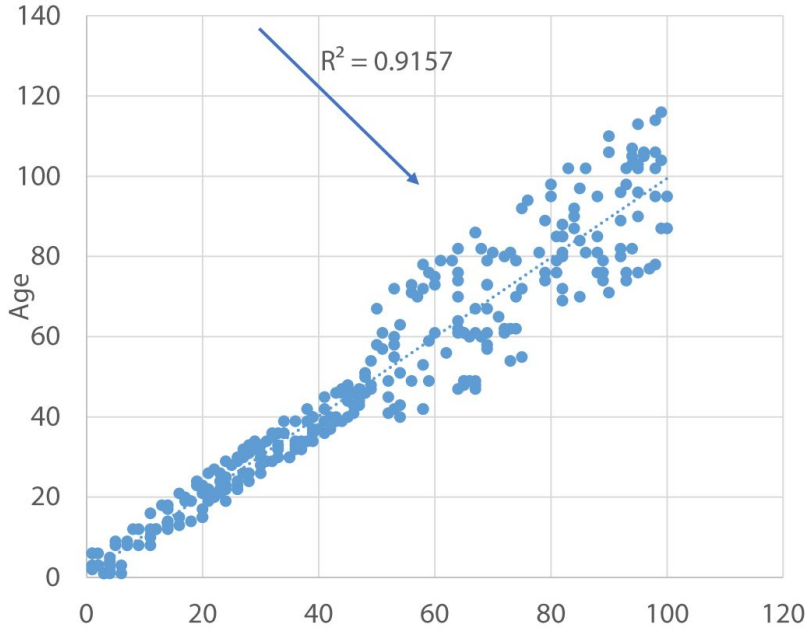
1) Все три верно: 0.7*0.7*0.7 = 0.3429
2) Два из трёх верно: 3 * (0.7*0.7*0.3) = 0.4409
3) Один из трёх верно: 3 * (0.7*0.3*0.3) = 0.189
4) Никто не ответил верно: 0.3*0.3*0.3 = 0.027

Итоговая точность ансамбля: 0.3429 + 0.4409 = 0.7838

Корреляция??

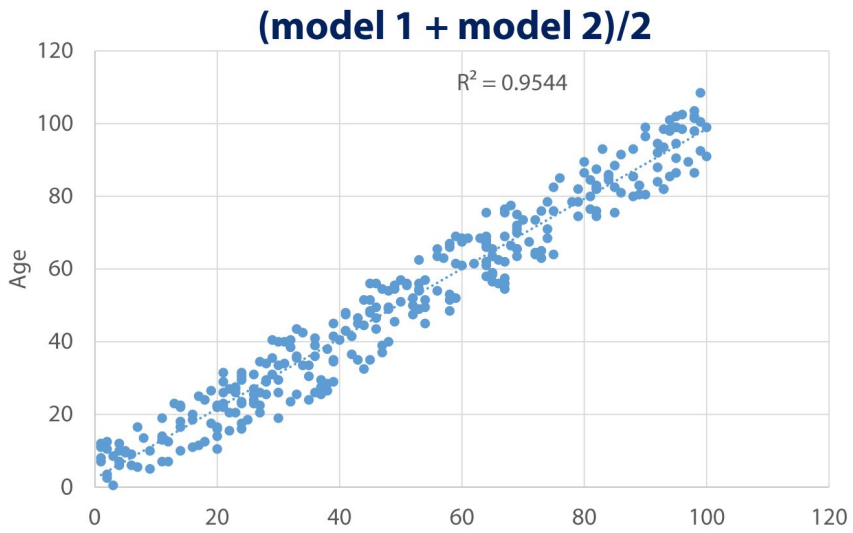
Ансамбли - усреднение

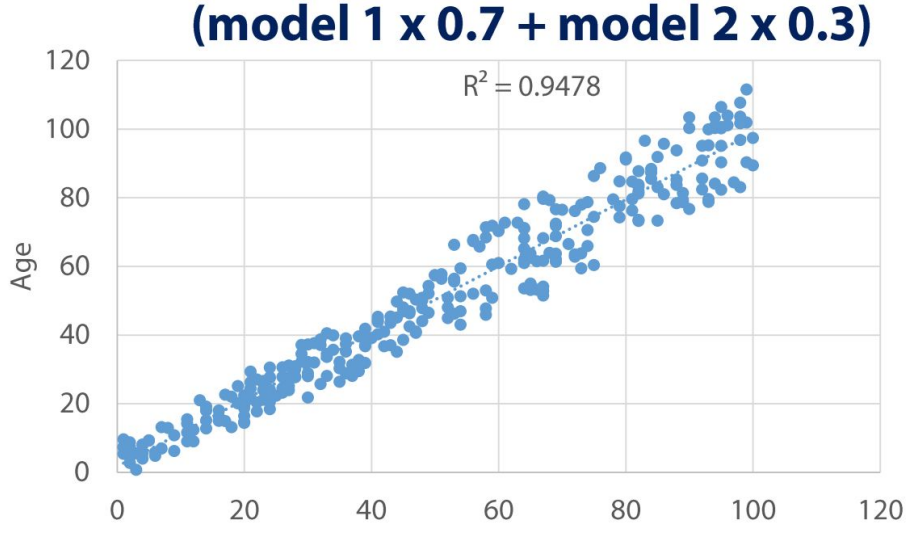




https://www.coursera.org/learn/competitive-data-science

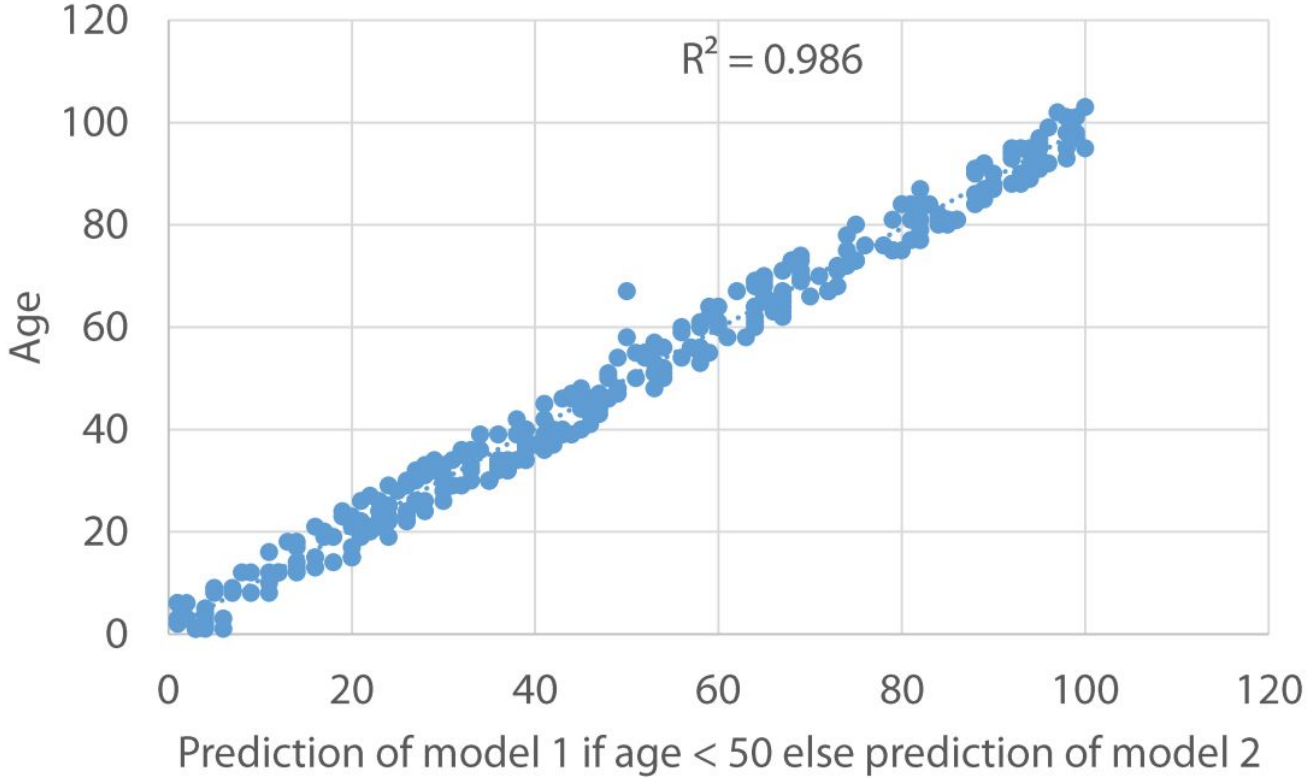
Ансамбли - усреднение





https://www.coursera.org/learn/competitive-data-science

Ансамбли - if



https://www.coursera.org/learn/competitive-data-science

Ансамбли - bagging

Используем немного отличающиеся версии одной модели для увеличения точности предсказания.

Параметры для Bagging:

- random seed
- bootstrap
- shuffling
- model specific parameters

Ансамбли - bagging

BaggingClassifier and **BaggingRegressor** from Sklearn

```
# train is the training data
# test is the test data
# y is the target variable
model=RandomForestRegressor()
bags=10
seed=1
# create array object to hold bagged predictions
bagged_prediction=np.zeros(test.shape[0])
#loop for as many times as we want bags
for n in range (0, bags):
     model.set_params(random_state=seed + n)# update seed
     model.fit(train,y) # fit model
     preds=model.predict(test) # predict on test data
     bagged_prediction+=preds # add predictions to bagged predictions
#take average of predictions
bagged_prediction/= bags
```

https://www.coursera.org/learn/competitive-data-science

Ансамбли - boosting

Вид взвешенного среднего ряда моделей, где все модели строятся последовательно, учитывая качество всех предыдущих.

Rownum	x0	x1	x2	x3	y
0	0.94	0.27	0.80	0.34	1
1	0.84	0.79	0.89	0.05	1
2	0.83	0.11	0.23	0.42	1
3	0.74	0.26	0.03	0.41	0
4	0.08	0.29	0.76	0.37	0
5	0.71	0.76	0.43	0.95	1
6	0.08	0.72	0.97	0.04	0

Rownum	x0	x1	x2	x3	y	pred
0	0.94	0.27	0.80	0.34	1	0.80
1	0.84	0.79	0.89	0.05	1	0.75
2	0.83	0.11	0.23	0.42	1	0.65
3	0.74	0.26	0.03	0.41	0	0.40
4	0.08	0.29	0.76	0.37	0	0.55
5	0.71	0.76	0.43	0.95	1	0.34
6	0.08	0.72	0.97	0.04	0	0.02

https://www.coursera.org/learn/competitive-data-science

Ансамбли - boosting

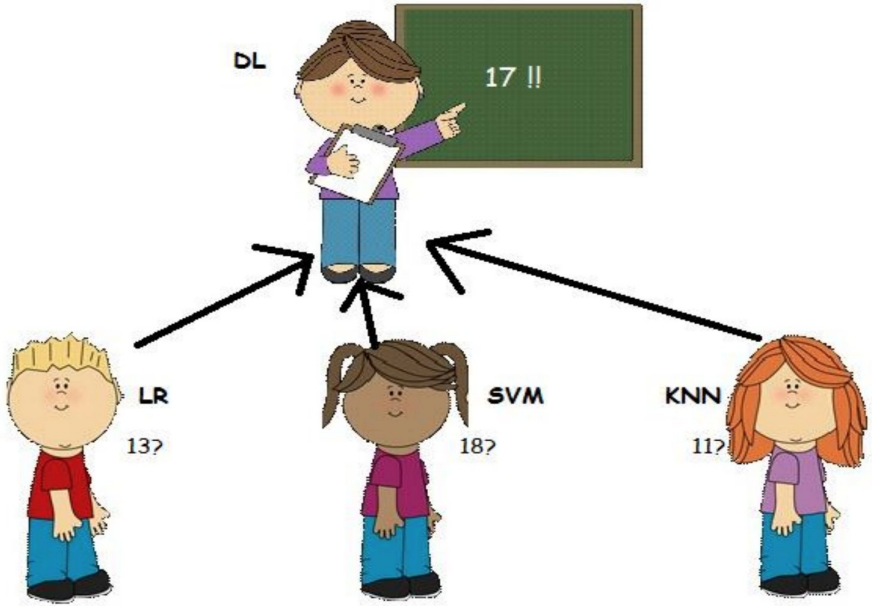
Rownum	x0	x1	x2	x3	y	pred	error
0	0.94	0.27	0.80	0.34	1	0.80	0.20
1	0.84	0.79	0.89	0.05	1	0.75	0.25
2	0.83	0.11	0.23	0.42	1	0.65	0.35
3	0.74	0.26	0.03	0.41	0	0.40	-0.40
4	0.08	0.29	0.76	0.37	0	0.55	-0.55
5	0.71	0.76	0.43	0.95	1	0.34	0.66
6	0.08	0.72	0.97	0.04	0	0.02	-0.02

Rownum	x0	x1	x2	x3	y	new pred
0	0.94	0.27	0.80	0.34	0.2	0.15
1	0.84	0.79	0.89	0.05	0.25	0.20
2	0.83	0.11	0.23	0.42	0.35	0.40
3	0.74	0.26	0.03	0.41	-0.4	-0.30
4	0.08	0.29	0.76	0.37	-0.55	-0.20
5	0.71	0.76	0.43	0.95	0.66	0.24
6	0.08	0.72	0.97	0.04	-0.02	-0.01

https://www.coursera.org/learn/competitive-data-science

Ансамбли - stacking

Получение предсказаний ряда моделей для hold-out датасета, и использование мета-моделей для обучения на их предсказаниях.



https://www.coursera.org/learn/competitive-data-science

Ансамбли - stacking

Алгоритм простого stacking:

1) делим train на 2 части
2) обучаем base-модели на 1-ой части
3) делаем предсказание base-моделями для 2-ой части
4) используем предсказания из 3) для обучения модели высшего уровня

Ансамбли - stacking

A

X0	x1	x2	xn	y
0.17	0.25	0.93	0.79	1
0.35	0.61	0.93	0.57	0
0.44	0.59	0.56	0.46	0
0.37	0.43	0.74	0.28	1
0.96	0.07	0.57	0.01	1

B

X0	x1	x2	xn	y
0.89	0.72	0.50	0.66	0
0.58	0.71	0.92	0.27	1
0.10	0.35	0.27	0.37	0
0.47	0.68	0.30	0.98	0
0.39	0.53	0.59	0.18	1

C

X0	x1	x2	xn	y
0.29	0.77	0.05	0.09	?
0.38	0.66	0.42	0.91	?
0.72	0.66	0.92	0.11	?
0.70	0.37	0.91	0.17	?
0.59	0.98	0.93	0.65	?

Train algorithm **0** on A and make predictions for B and C and save to **B1**, **C1**

B1

pred0
0.24
0.95
0.64
0.89
0.11

C1

pred0
0.50
0.62
0.22
0.90
0.20

https://www.coursera.org/learn/competitive-data-science

Ансамбли - stacking

A

X0	x1	x2	xn	y
0.17	0.25	0.93	0.79	1
0.35	0.61	0.93	0.57	0
0.44	0.59	0.56	0.46	0
0.37	0.43	0.74	0.28	1
0.96	0.07	0.57	0.01	1

B

X0	x1	x2	xn	y
0.89	0.72	0.50	0.66	0
0.58	0.71	0.92	0.27	1
0.10	0.35	0.27	0.37	0
0.47	0.68	0.30	0.98	0
0.39	0.53	0.59	0.18	1

C

X0	x1	x2	xn	y
0.29	0.77	0.05	0.09	?
0.38	0.66	0.42	0.91	?
0.72	0.66	0.92	0.11	?
0.70	0.37	0.91	0.17	?
0.59	0.98	0.93	0.65	?

Train algorithm **0** on A and make predictions for B and C and save to **B1**, **C1**
Train algorithm **1** on A and make predictions for B and C and save to **B1**, **C1**

B1

pred0	pred1
0.24	0.72
0.95	0.25
0.64	0.80
0.89	0.58
0.11	0.20

C1

pred0	pred1
0.50	0.50
0.62	0.59
0.22	0.31
0.90	0.47
0.20	0.09

https://www.coursera.org/learn/competitive-data-science

Ансамбли - stacking

A

X0	x1	x2	xn	y
0.17	0.25	0.93	0.79	1
0.35	0.61	0.93	0.57	0
0.44	0.59	0.56	0.46	0
0.37	0.43	0.74	0.28	1
0.96	0.07	0.57	0.01	1

B

X0	x1	x2	xn	y
0.89	0.72	0.50	0.66	0
0.58	0.71	0.92	0.27	1
0.10	0.35	0.27	0.37	0
0.47	0.68	0.30	0.98	0
0.39	0.53	0.59	0.18	1

C

X0	x1	x2	xn	y
0.29	0.77	0.05	0.09	?
0.38	0.66	0.42	0.91	?
0.72	0.66	0.92	0.11	?
0.70	0.37	0.91	0.17	?
0.59	0.98	0.93	0.65	?

Train algorithm **0** on A and make predictions for B and C and save to **B1**, **C1**
Train algorithm **1** on A and make predictions for B and C and save to **B1**, **C1**
Train algorithm **2** on A and make predictions for B and C and save to **B1**, **C1**

B1

pred0	pred1	pred2	y
0.24	0.72	0.70	0
0.95	0.25	0.22	1
0.64	0.80	0.96	0
0.89	0.58	0.52	0
0.11	0.20	0.93	1

C1

pred0	pred1	pred2	y
0.50	0.50	0.39	?
0.62	0.59	0.46	?
0.22	0.31	0.54	?
0.90	0.47	0.09	?
0.20	0.09	0.61	?

https://www.coursera.org/learn/competitive-data-science

Ансамбли - stacking

A

X0	x1	x2	xn	y
0.17	0.25	0.93	0.79	1
0.35	0.61	0.93	0.57	0
0.44	0.59	0.56	0.46	0
0.37	0.43	0.74	0.28	1
0.96	0.07	0.57	0.01	1

B

X0	x1	x2	xn	y
0.89	0.72	0.50	0.66	0
0.58	0.71	0.92	0.27	1
0.10	0.35	0.27	0.37	0
0.47	0.68	0.30	0.98	0
0.39	0.53	0.59	0.18	1

C

X0	x1	x2	xn	y
0.29	0.77	0.05	0.09	?
0.38	0.66	0.42	0.91	?
0.72	0.66	0.92	0.11	?
0.70	0.37	0.91	0.17	?
0.59	0.98	0.93	0.65	?

Train algorithm **0** on A and make predictions for B and C and save to **B1**, **C1**
Train algorithm **1** on A and make predictions for B and C and save to **B1**, **C1**
Train algorithm **2** on A and make predictions for B and C and save to **B1**, **C1**

B1

pred0	pred1	pred2	y
0.24	0.72	0.70	0
0.95	0.25	0.22	1
0.64	0.80	0.96	0
0.89	0.58	0.52	0
0.11	0.20	0.93	1

C1

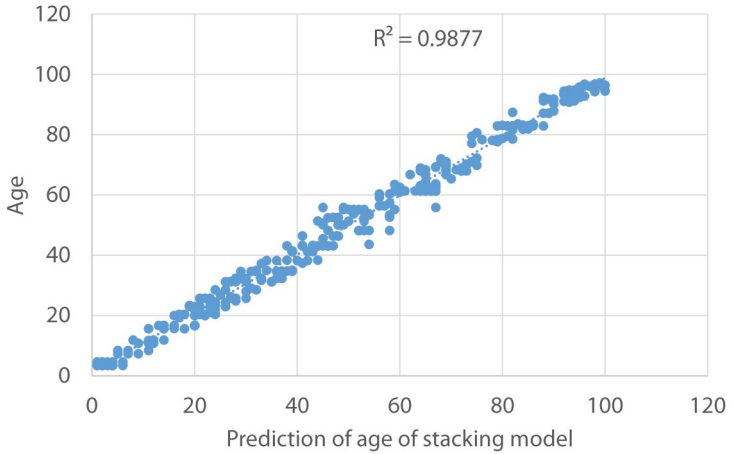
pred0	pred1	pred2	y	Preds3
0.50	0.50	0.39	?	0.45
0.62	0.59	0.46	?	0.23
0.22	0.31	0.54	?	0.99
0.90	0.47	0.09	?	0.34
0.20	0.09	0.61	?	0.05

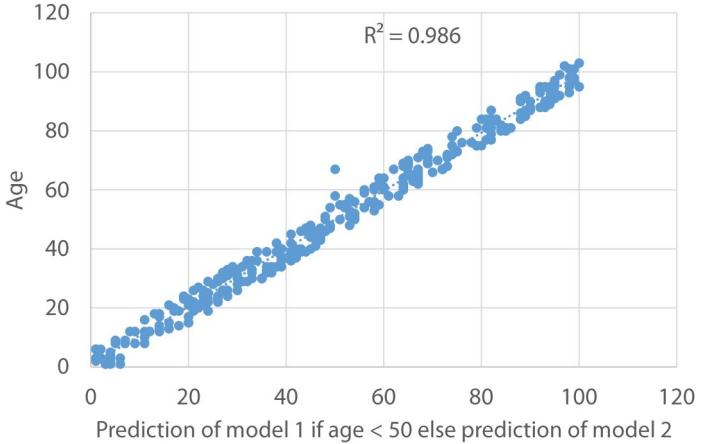
Train algorithm **3** on B1 and make predictions for C1

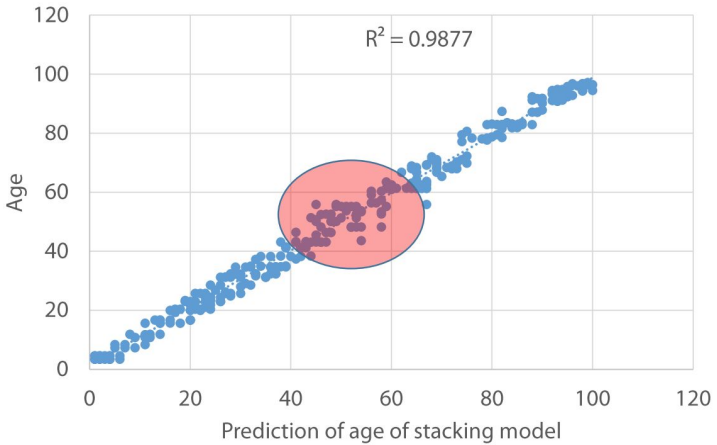
https://www.coursera.org/learn/competitive-data-science

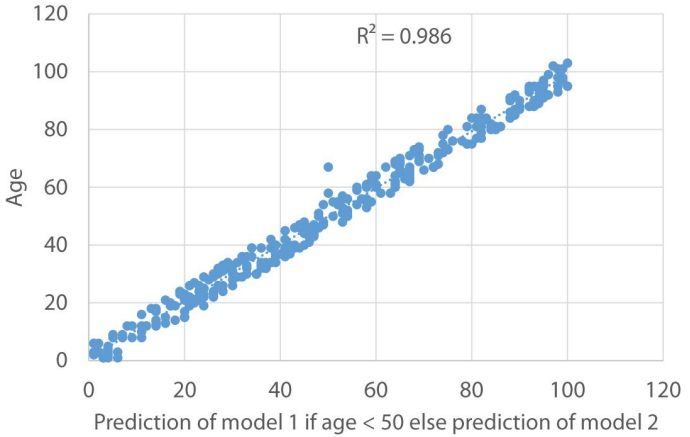
Ансамбли - stacking

https://www.coursera.org/learn/competitive-data-science









Ансамбли - stacking

https://www.coursera.org/learn/competitive-data-science

Stacking example

```
from sklearn.ensemble import RandomForestRegressor #import model
from sklearn.linear_model import LinearRegression #import model
import numpy as np #import numpy for stats
from sklearn.model_selection import train_test_split # split the training data

# train is the training data
# y is the target variable for the train data
# test is the test data
```

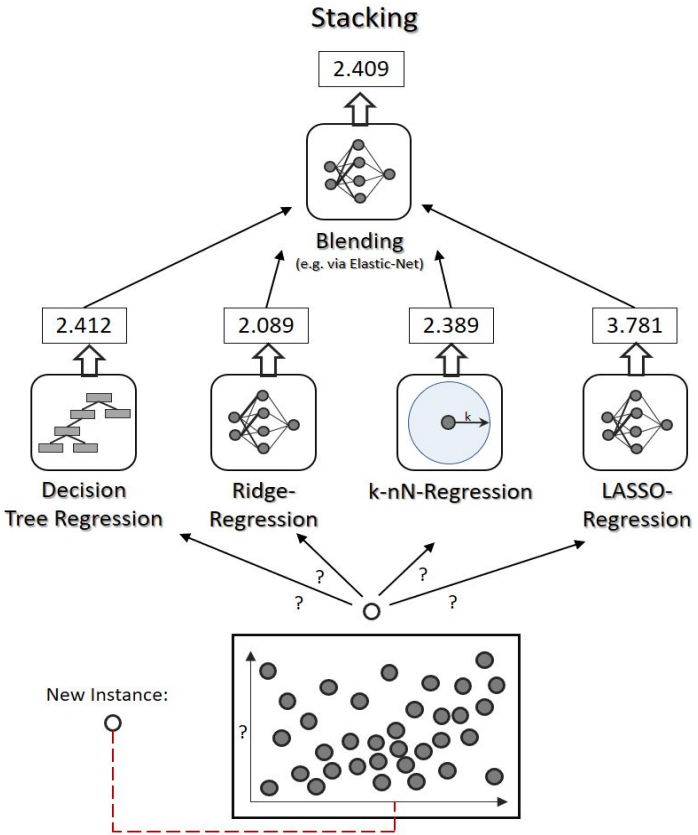
Ансамбли - stacking

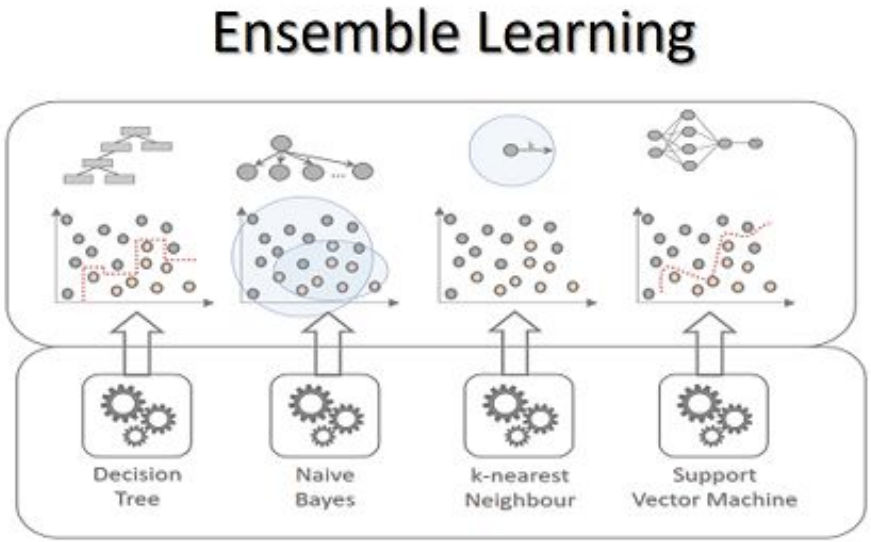
Stacking example

https://www.coursera.org/learn/competitive-data-science

```
#split train data in 2 parts, training and  valdiation.
training,valid,ytraining,yvalid = train_test_split(train,y,test_size=0.5)
#specify models
model1=RandomForestRegressor()
model2=LinearRegression()
#fit models
model1.fit(training,ytraining)
model2.fit(training,ytraining)
#make predictions for validation
preds1=model1.predict(valid)
preds2=model2.predict(valid)
#make predictions for test data
test_preds1=model1.predict(test)
test_preds2=model2.predict(test)
#Form a new dataset for valid and test via stacking the predictions
stacked_predictions=np.column_stack((preds1,preds2))
stacked_test_predictions=np.column_stack((test_preds1,test_preds2))
#specify meta model
meta_model=LinearRegression()
#fit meta model on stacked predictions
meta_model.fit(stacked_predictions,yvalid)
#make predictions on the stacked predictions of the test data
final_predictions=meta_model.predict(stacked_test_predictions)
```

Ансамбли - stacking





Ансамбли - stacking

https://www.coursera.org/learn/competitive-data-science

Модели 1-го уровня

- ❑ 2-3 gradient boosted trees (lightgb, xgboost, H2O, catboost)
- ❑ 2-3 Neural nets (keras, pytorch)
- ❑ 1-2 ExtraTrees/Random Forest (sklearn)
- ❑ 1-2 linear models as in logistic/ridge regression, linear svm (sklearn)
- ❑ 1-2 knn models (sklearn)
- ❑ 1 Factorization machine (libfm)
- ❑ 1 svm with nonlinear kernel if size/memory allows (sklearn)

Модели 2-го уровня

- ❑ gradient boosted trees with small depth (like 2 or 3)
- ❑ Linear models with high regularization
- ❑ Extra Trees

https://gist.github.com/geffy/5a534f9deb80775f0f497a4133a79d7b

Ансамбли: плюсы

- Улучшение качества предсказаний засчёт учёта преимуществ каждого отдельного алгоритма в итоговой ансамбль-модели
- Робастность и устойчивость
- Линейные + нелинейные зависимости в ансамбле от двух моделей

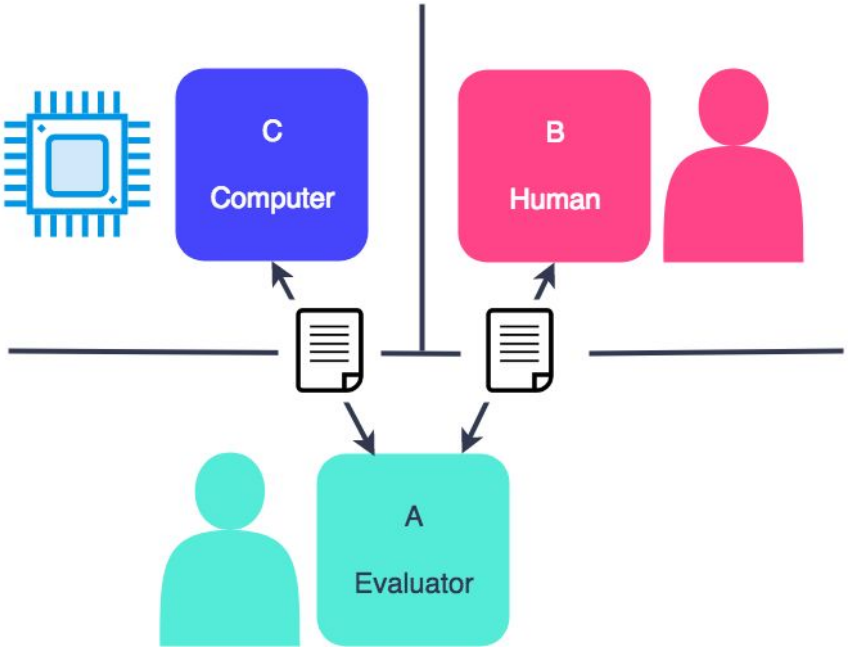
Ансамбли: минусы

- Ухудшение интерпретации предсказаний модели
- Вычислительная сложность, долгое построение
- Выбор подходящих моделей - искусство

Natural Language Processing

Задачи, связанные с обработкой текстов, люди пытались решать ещё с самого начала появления компьютеров.

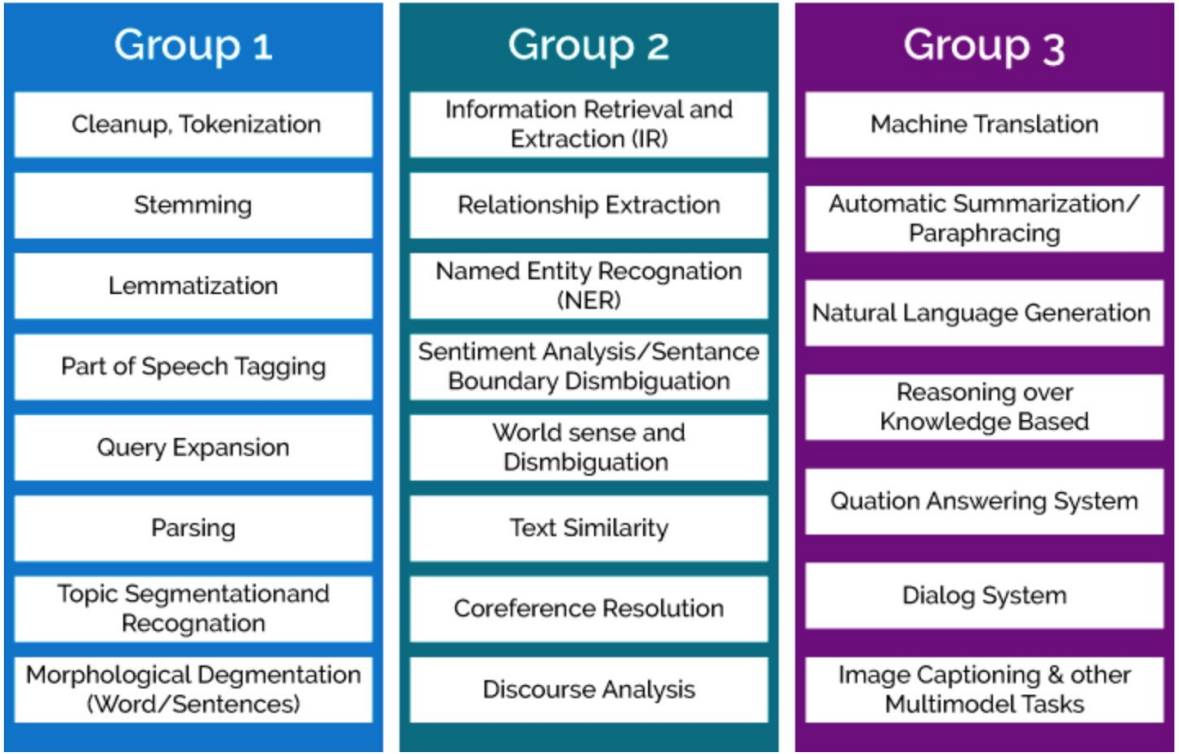
- 1950 - первые попытки, rule based
- 1980-2010 статистические методы + первые ML методы (Decision Trees)
- 2010s Neural networks + некоторые статические методы



https://botsociety.io/blog/2018/03/the-turing-test/

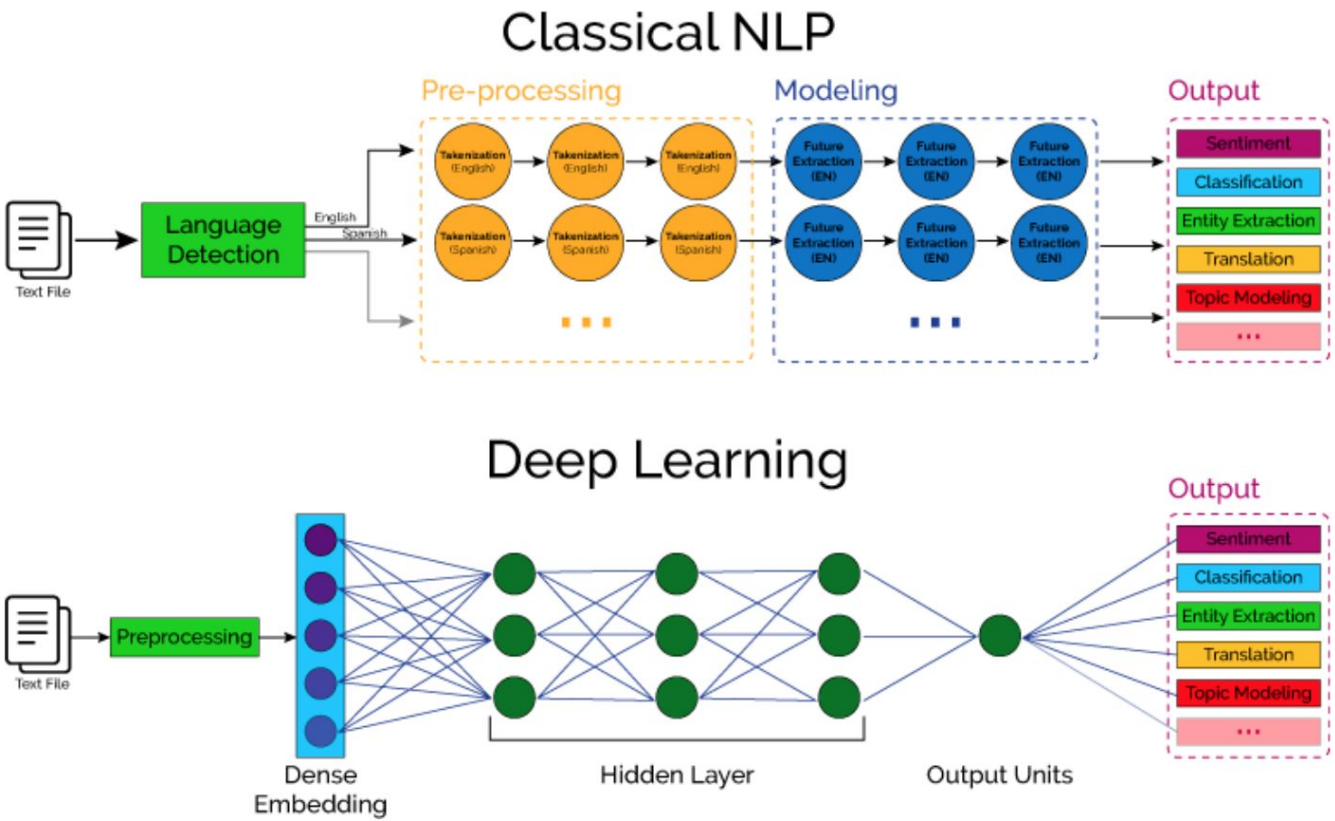
NLP - tasks

- Machine Translation
- Language modeling
- Part of speech tagging
- Parsing
- Summarization
- Sentiment Analysis
- Dialog Systems
- NER
- Question Answering
- Topic Modelling
- ...



https://www.upwork.com/hiring/for-clients/artificial-intelligence-and-natural-language-processing-in-big-data/

NLP - tasks



https://www.upwork.com/hiring/for-clients/artificial-intelligence-and-natural-language-processing-in-big-data/

NLP - one hot

Можем закодировать каждое слово своим числом, но тогда будут one-hot vectors размера словаря (500,000).

mother = [0...0,1,0...0]

cat = [1,0...0]

НО:

- Вектора не содержат в себе никакого “смысла”
- Вектора никак нельзя сравнить разумным образом

NLP - context embeddings

What other words fit these examples?

1. Marie rode a _____
2. _____ wheel was punctured
3. The _____ has a beautiful white frame.

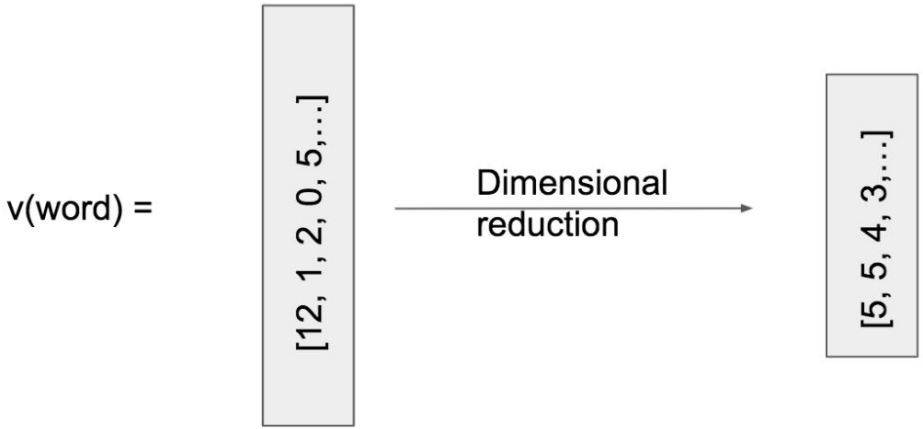
Word	1	2	3
Bicycle	+	+	+
Bike	+	+	+
Car	+	+	-
Horse	+	-	-

NLP - context embeddings

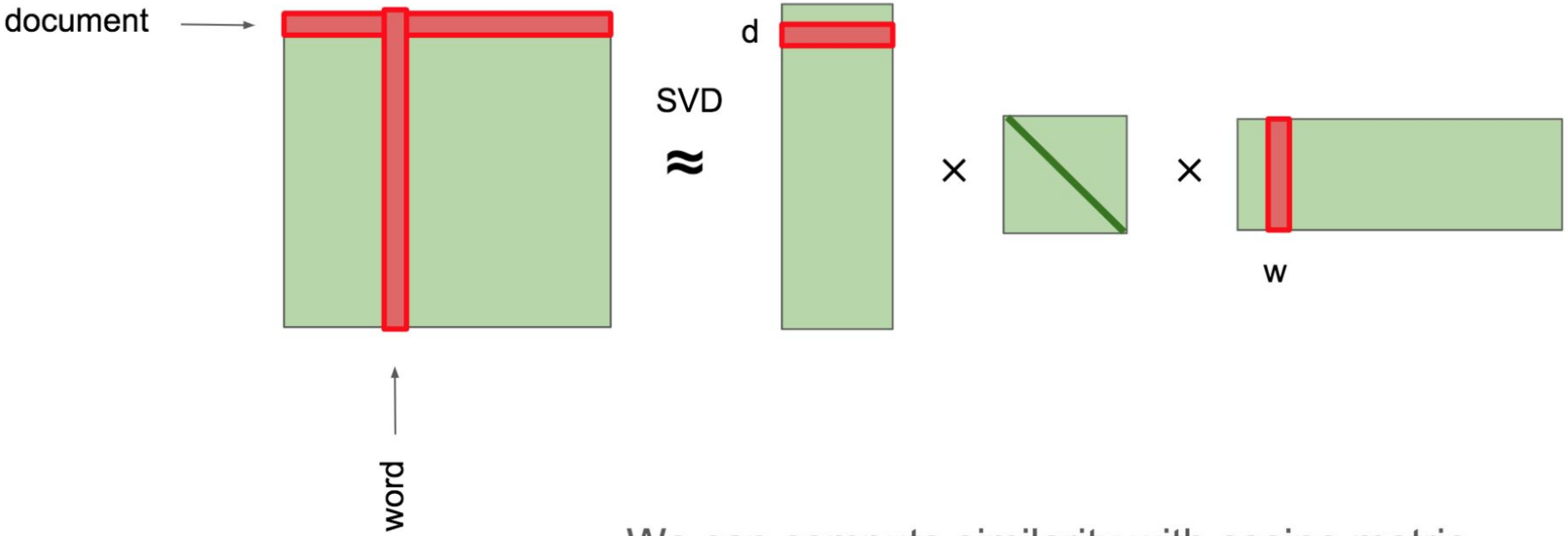
v(word_i)[j] = count(co-occurrences word_i with word_j)
v(word) = [12,1,2,0,5,...]

Опять же большой вектор (размер словаря)

Можно сжать с помощью методов понижения размерности



NLP - latent semantic analysis



We can compute similarity with cosine metric.

NLP - проблемы методов

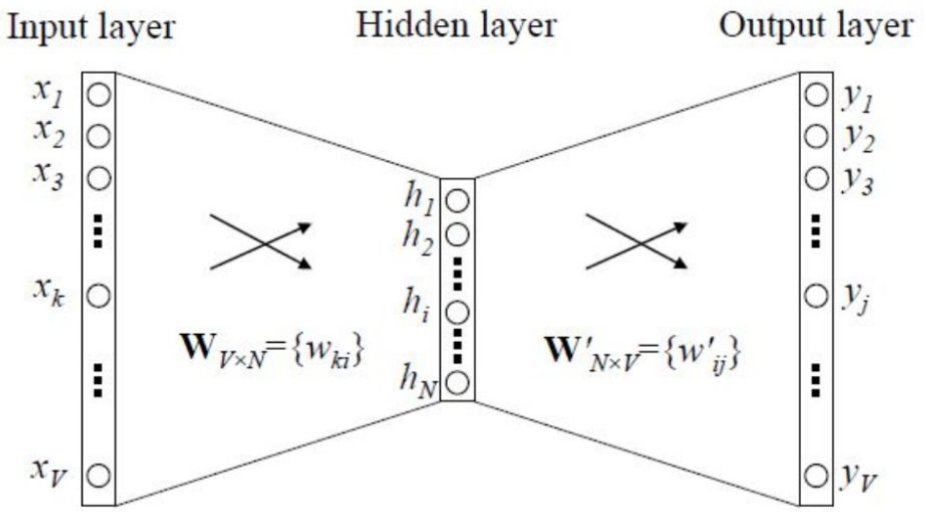
- Редкие слова
- Большое время для вычислений
- Неустойчивость к небольшим изменениям датасета

Что мы хотим?

Получить представления слов без правил, придуманных человеком.

NLP - word embeddings

Word2Vec

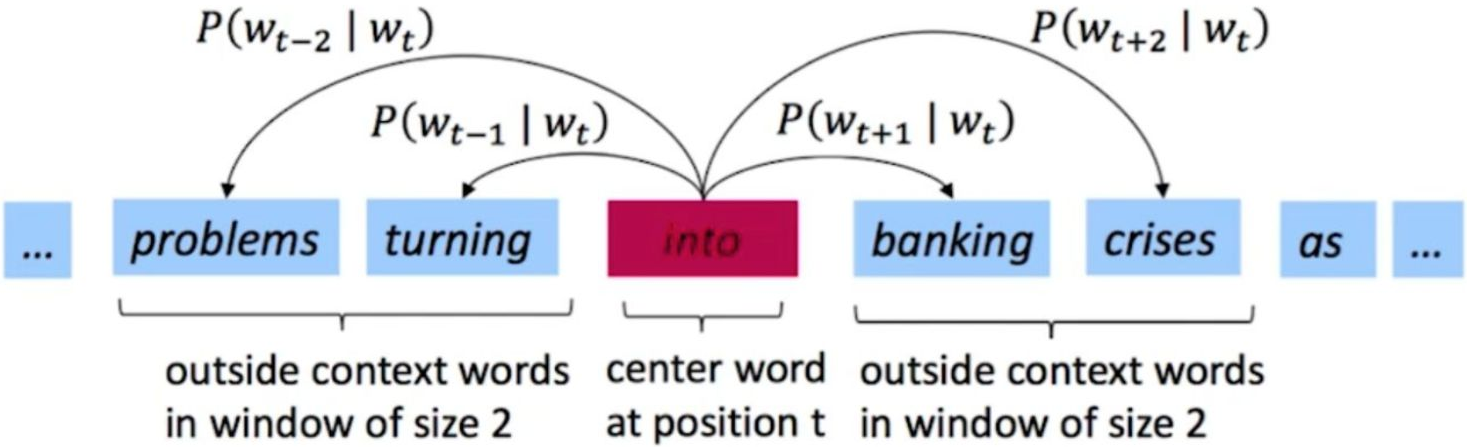


- ➔ Predict context words given a word (or vice versa)
- ➔ maximize probability of seeing word and its context together

NLP - word embeddings

Word2Vec

Going through text corpus by sliding windows



NLP - word embeddings

Word2Vec objective function

We want to maximize probability of context words given the center word (log-likelihood):

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P\left(w_{t+j} \mid w_t; \theta\right)$$

Word2Vec objective function

We want to maximize probability of context words given the center word (log-likelihood):

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

OR we want to minimize negative log-likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Word2Vec objective function

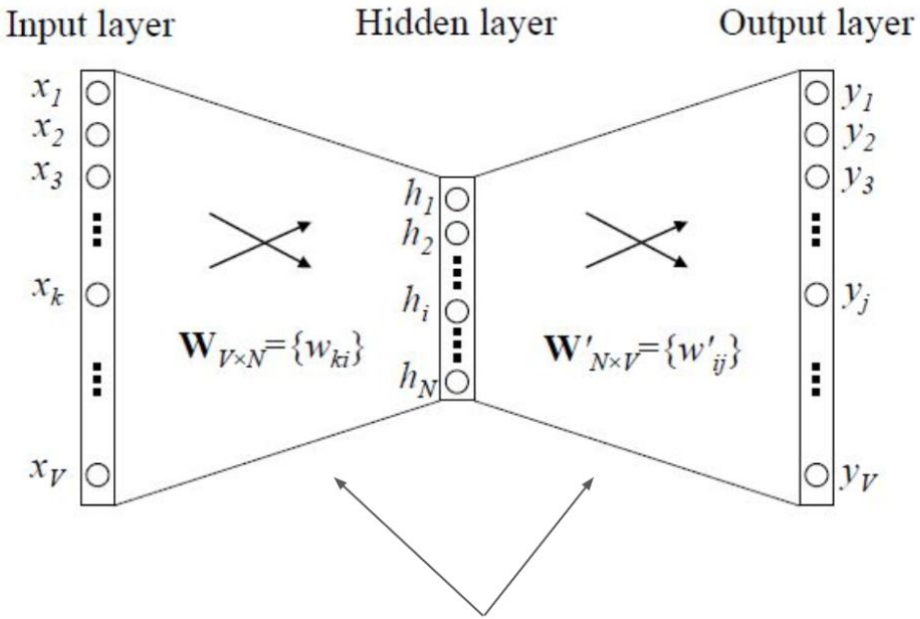
Given a center word **c** and a context word **o**:

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

- u is a context word vector
- v is a center word vector

“Scalar product between me and my neighbour must be as big as possible”

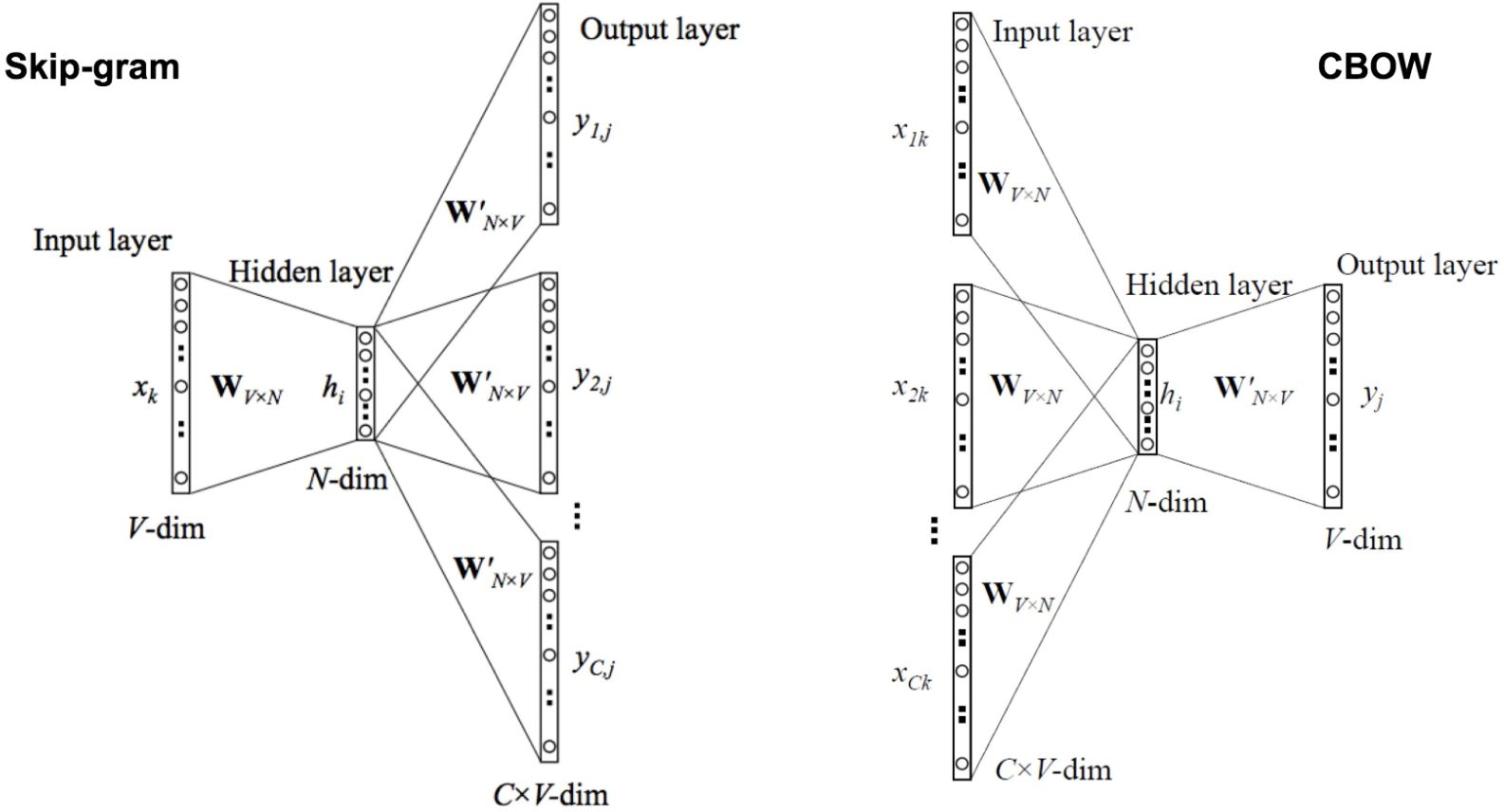
Word2Vec



$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

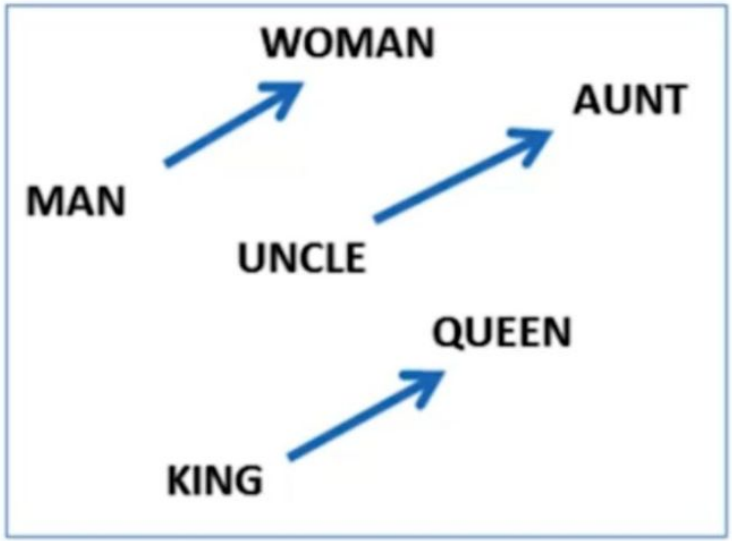
θ (матрицы векторов)

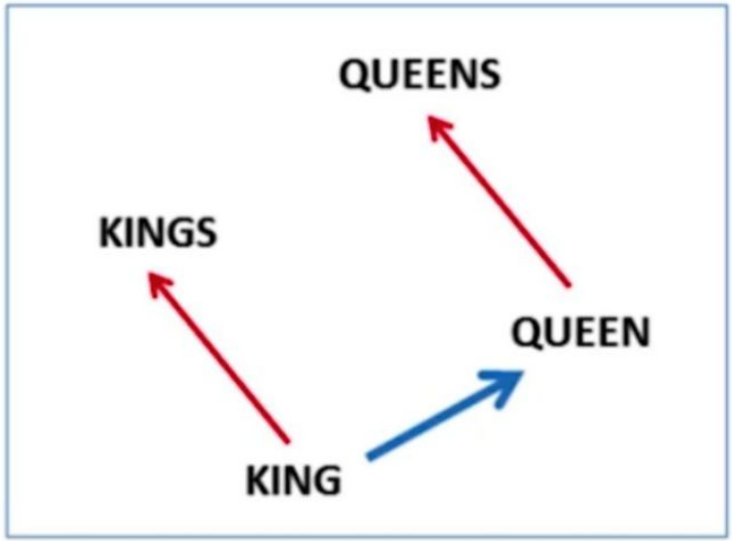
Word2Vec

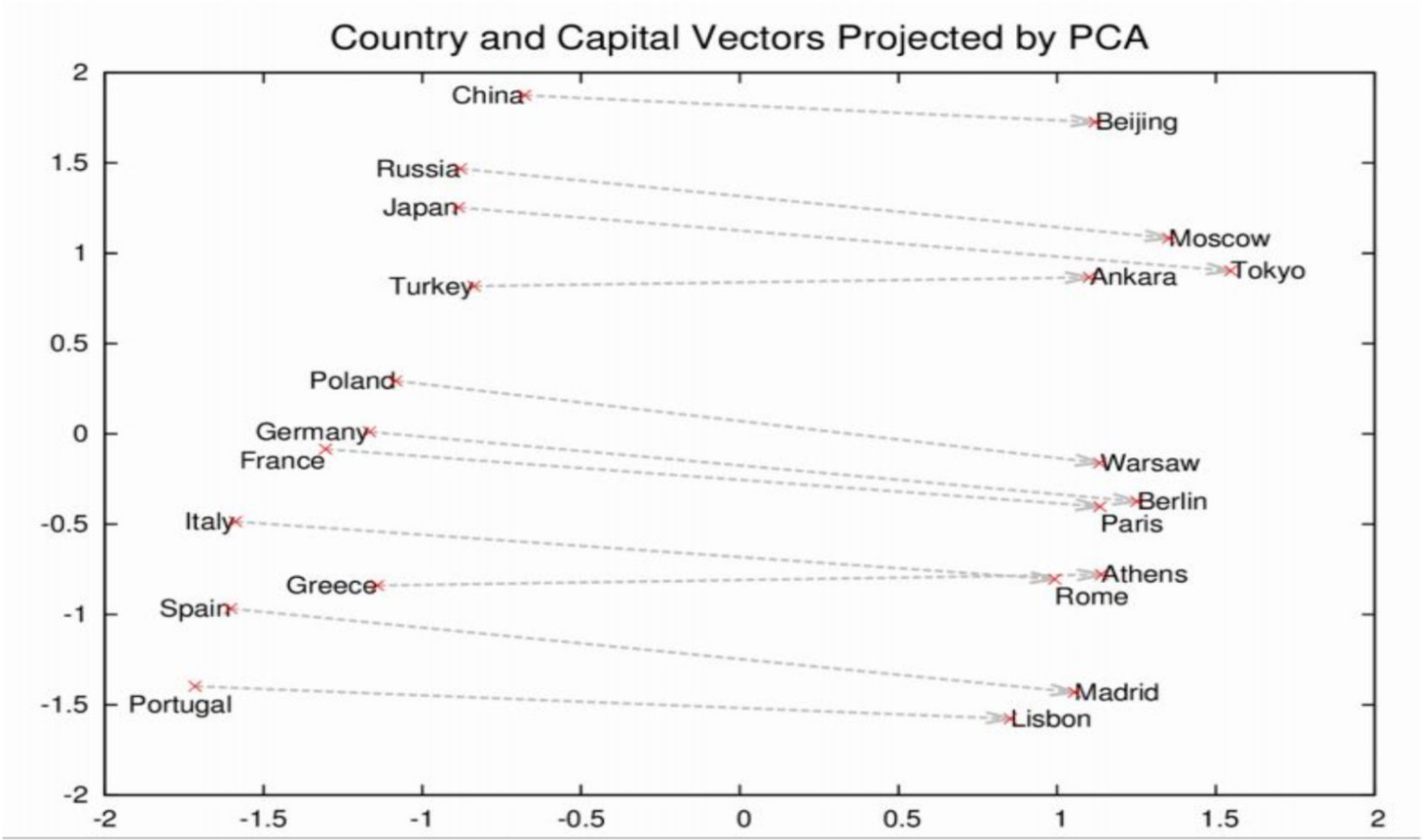

Skip-gram
Input layer
Hidden layer
Output layer
x_k
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W}'_{N\times V}$
$y_{1,j}$
$y_{2,j}$
$y_{C,j}$
V-dim
N-dim
$C\times V$-dim
CBOW
Input layer
Hidden layer
Output layer
x_{1k}
x_{2k}
x_{Ck}
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W}'_{N\times V}$
y_j
N-dim
V-dim
$C\times V$-dim

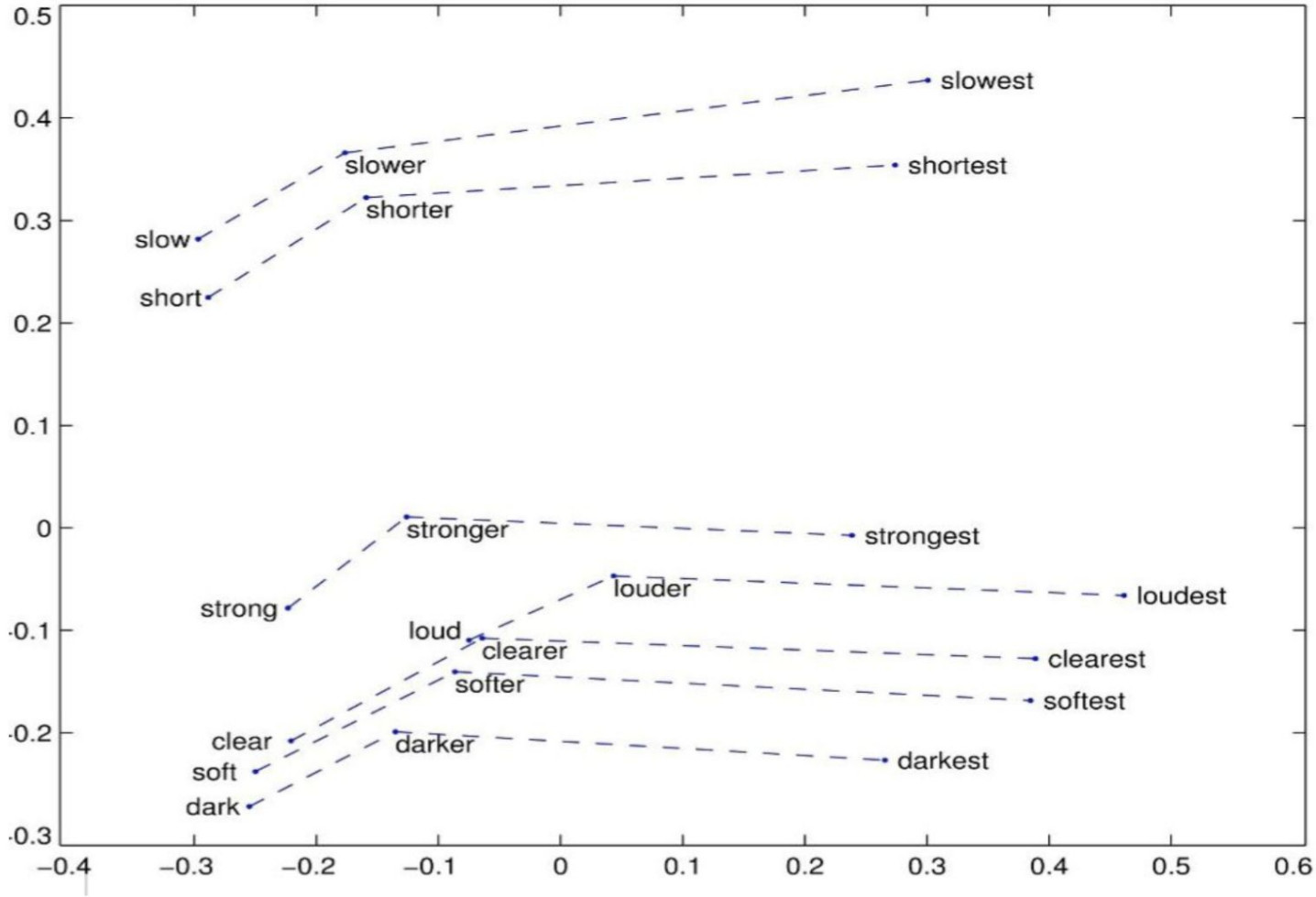
Word2Vec

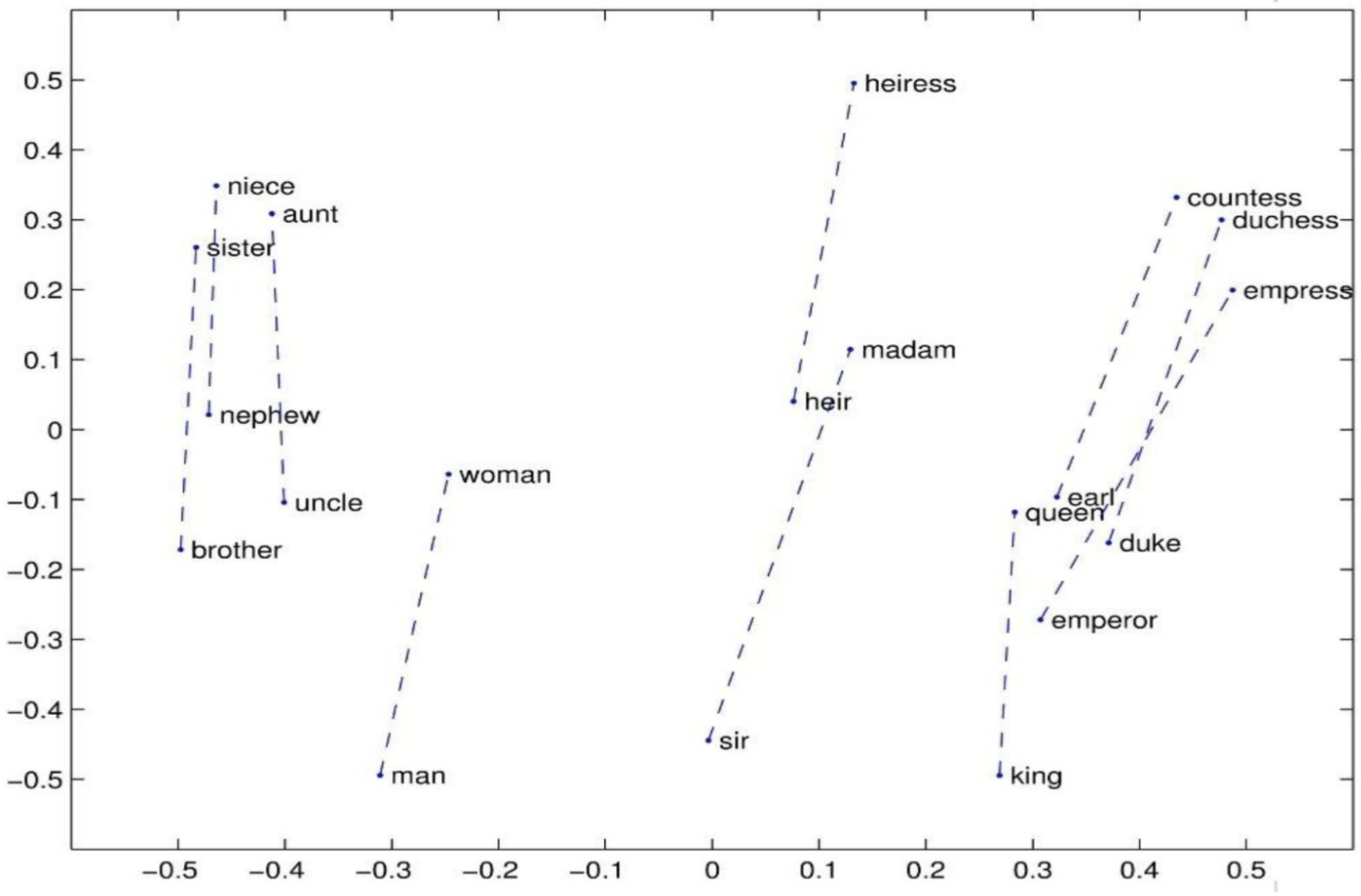
v(king) - v(man) + v(woman) ≈ v(queen)

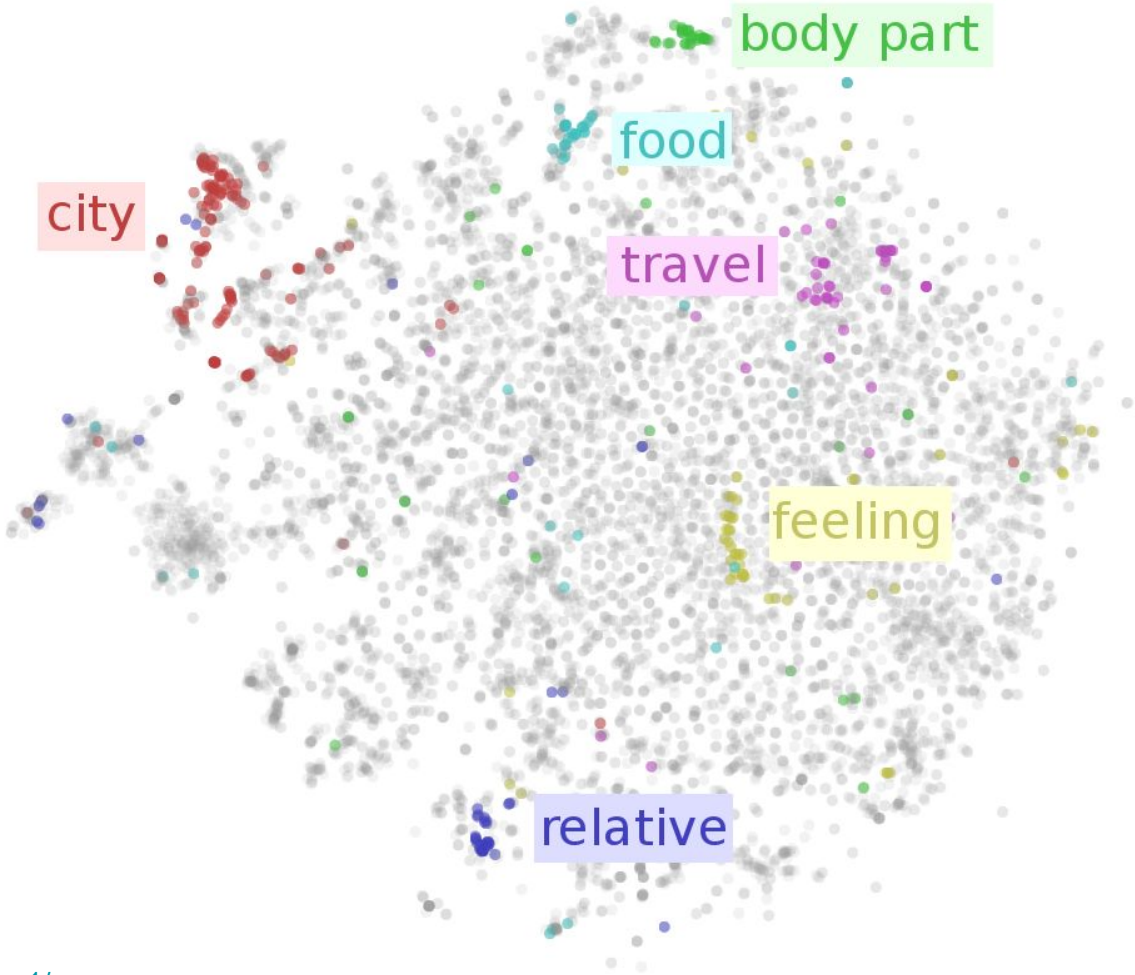





Country and Capital Vectors Projected by PCA
China
Beijing
Russia
Japan
Moscow
Tokyo
Turkey
Ankara
Poland
Warsaw
Germany
France
Berlin
Paris
Italy
Athens
Greece
Rome
Spain
Madrid
Portugal
Lisbon
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark


0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
niece
aunt
sister
nephew
uncle
brother
woman
man
heiress
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king



http://ruder.io/word-embeddings-1/

FastText

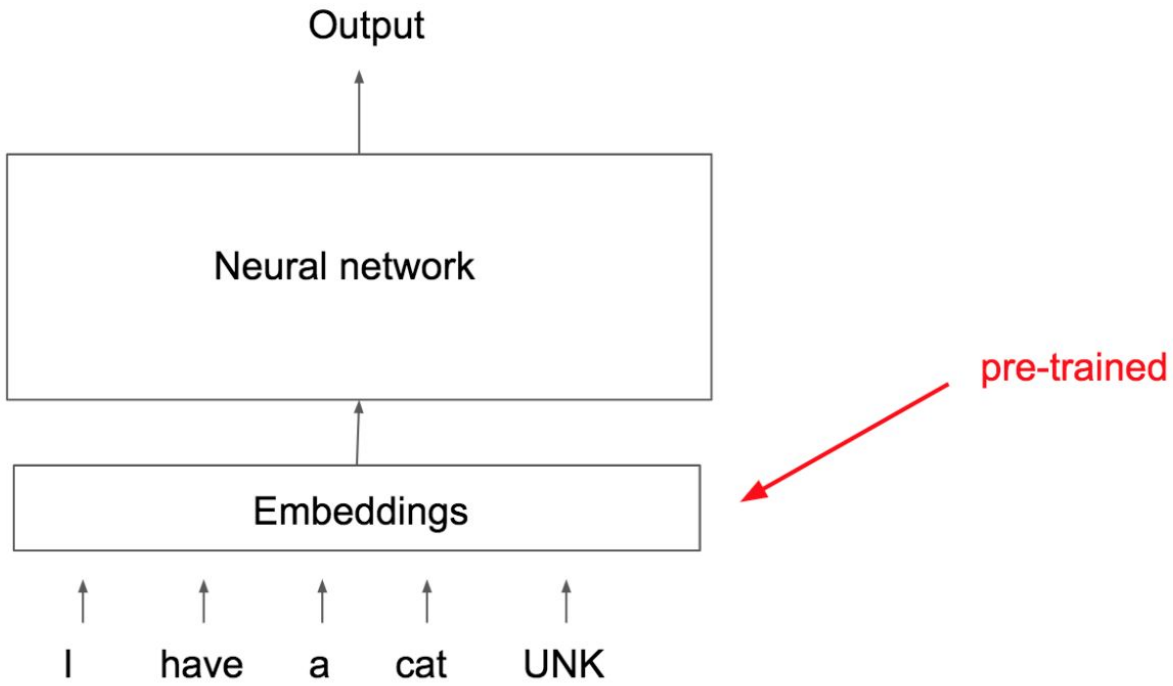
- Divide word into bag of n-grams: apple = <ap, ppl, ple, le>
- Compute vector for each n-gram
- Vector for a word = sum of vectors for word n-grams

Advantages:

- Reasonable embeddings for rare words and words with mistakes
- Model is the same as before, we can even use model trained on words to train it further on n-grams!

Зачем нам нужны эмбеддинги?

When you have small text data for your task



Language model

Given sequence of words LM can predict probability of this sequence.

I have a very interesting idea. -> P(I,have,a,very,interesting,idea)

- It is difficult to know the true probability of an arbitrary sequence of words (at least they are changing in time)
- We need to approximate probability with some model
- As usual this model will be better in one bunch of things and worse in another

- We can do language model statistically
- We can use ML algorithms

Language model

Given sequence of words LM can predict probability of this sequence.

I have a very interesting idea. -> P(I,have,a,very,interesting,idea)

- It is difficult to know the true probability of an arbitrary sequence of words (at least they are changing in time)
- We need to approximate probability with some model
- As usual this model will be better in one bunch of things and worse in another

- We can do language model statistically
- We can use ML algorithms

N-grams

$$P(X, Y) = P(X|Y)P(Y)$$

$$P(the\ cat\ slept\ quietly) = P(quietly|the\ cat\ slept) \cdot P(the\ cat\ slept) = P(quietly|the\ cat\ slept) \cdot P(slept \mid the\ cat) \cdot P(cat|the) \cdot P(the)$$

In N-gram model we state the independence assumption: the probability of word only depends on the fixed number of previous words.

Example: unigram (1-gram)

$$P(\text{cat seat on a mat}) = P(\text{mat}|\text{cat seat on a}) \cdot P(\text{cat seat on a}) = P(\text{mat}|\text{cat seat on a}) \cdot P(\text{a}|\text{cat seat on})P(\text{cat seat on}) = \ldots \cong P(\text{mat})P(\text{a})P(\text{on})P(\text{seat})P(\text{cat})$$

Как это сделать с помощью Neural Networks?

Fixed size solution:

Sample probability of next token, given fixed size input:

$$label = softmax(Wx + b)$$

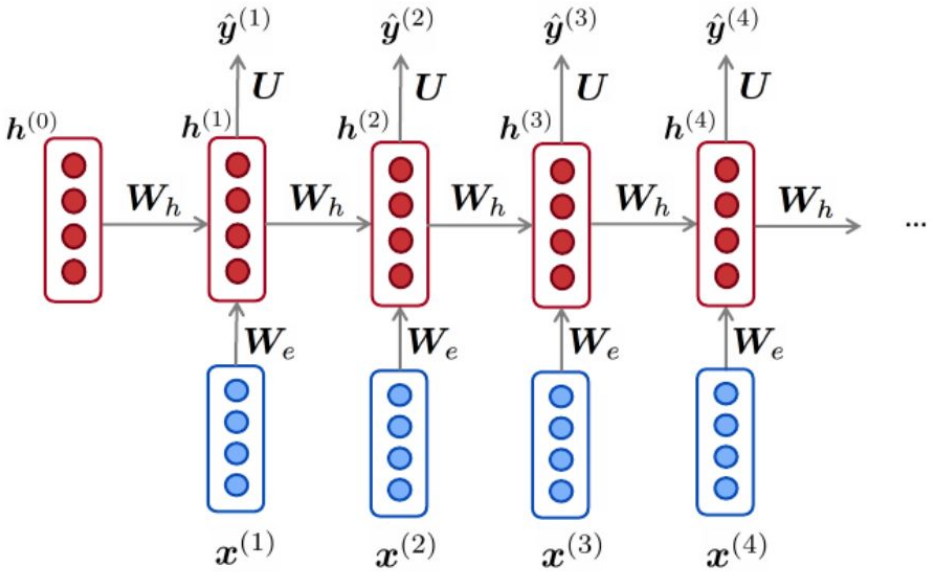
We can concatenate word embeddings:

$$label = softmax(W[\text{cat},\text{seat},\text{on}] + b)$$

We still have problems with fixed size model...

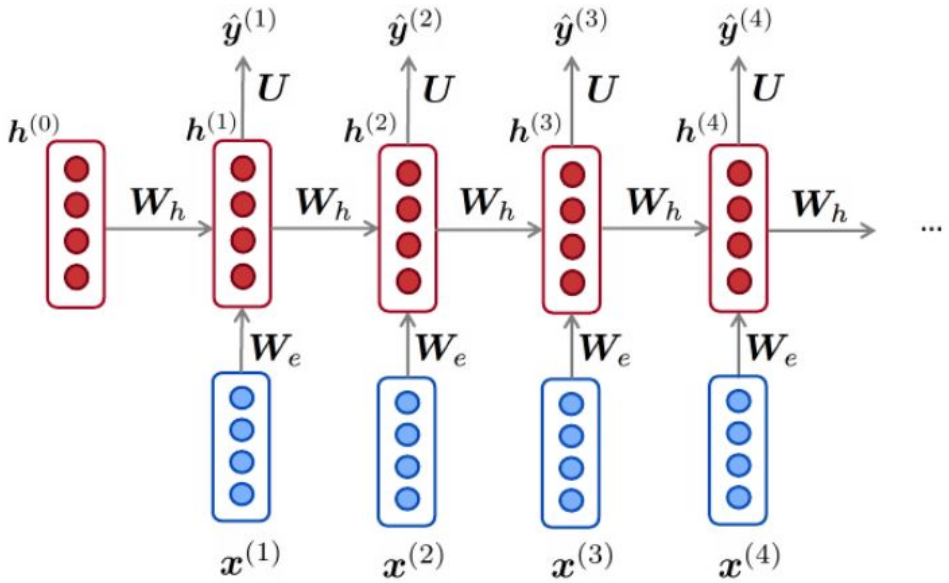
В общем случае мы хотим иметь доступ не только к n-граммам, но и ко всем предыдущим словам. Мы хотим каким-то образом запоминать информацию для предсказывания текущего токена.

Recurrent neural networks (RNNs)



Recurrent neural networks (RNNs)

$$
\begin{aligned}
h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\
\hat{y}_t &= W^{(S)} f(h_t)
\end{aligned}
$$



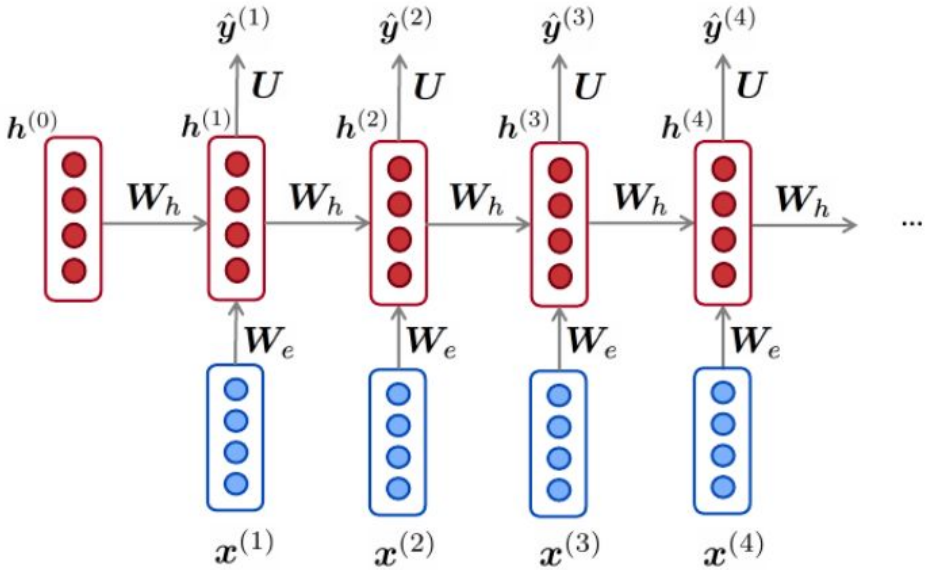
Recurrent neural networks (RNNs)

$$
\begin{aligned}
h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\
\hat{y}_t &= W^{(S)} f(h_t)
\end{aligned}
$$

$$
\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}
$$

E = error

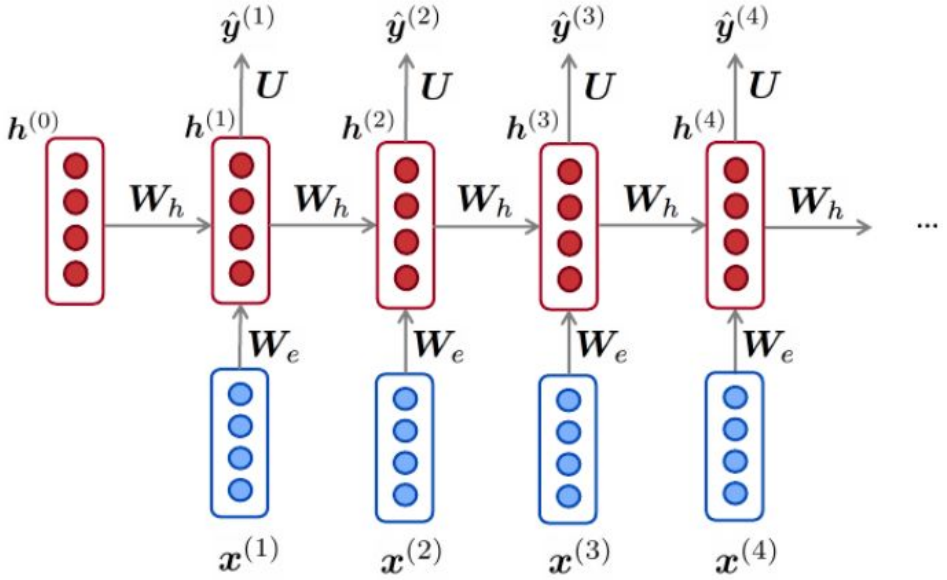
$$
\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}
$$



More chain rule

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$



RNNs have a lot of benefits, but also have some problems.

- Vanishing & Exploding gradients
- One vector for all information

RNN Language model

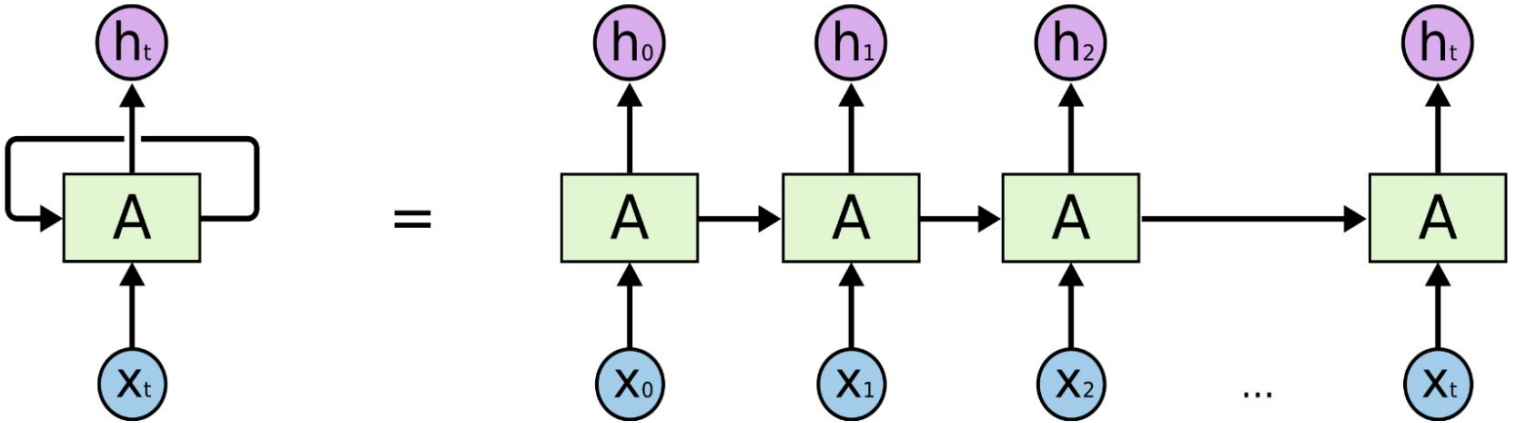
Benefits:

- Can process any length input with the same model size
- Can use information occured many steps before
- Share representations

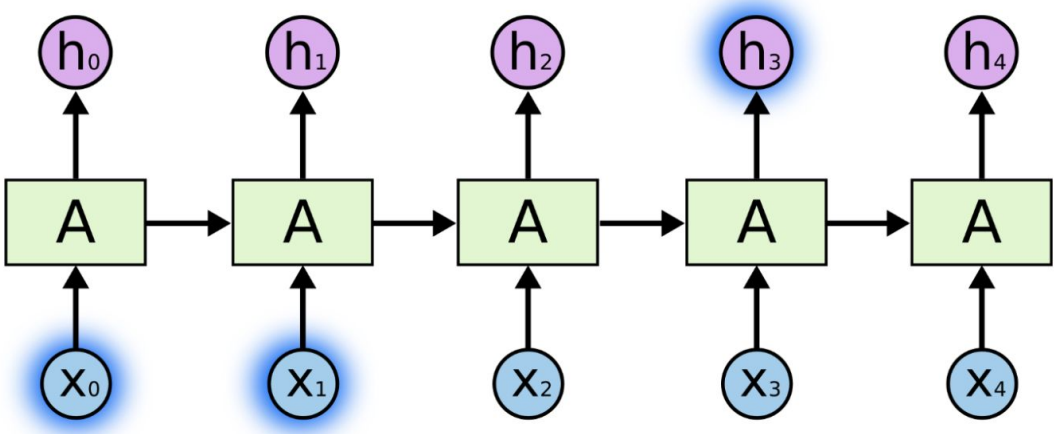
Problems:

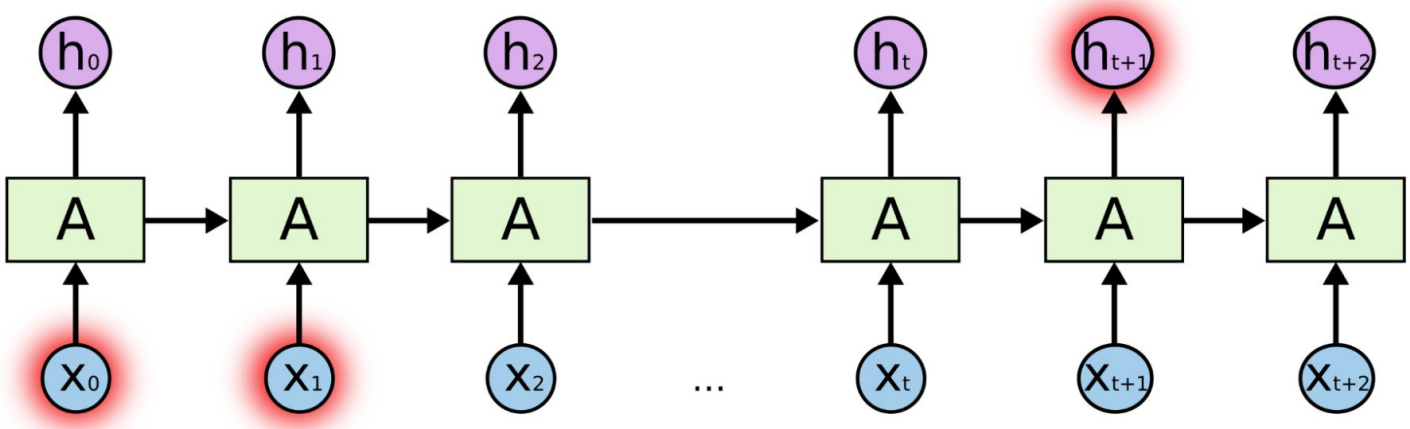
- Works quite slow
- In fact it is difficult to access remote information

RNN



problems


h0
h1
h2
h3
h4
A
A
A
A
A
x0
x1
x2
x3
x4


h0
h1
h2
ht
ht+1
ht+2
A
A
A
A
A
A
x0
x1
x2
...
xt
xt+1
xt+2

dream recurrent cell

What do we have:

$$s_{t-1} = (s_1, s_2, ..., s_n) - \textit{previous state}$$
$$h_{t-1} = (h_1, h_2, ..., h_q) - \textit{previous output}$$
$$x_{t-1} = (x_1, x_2, ..., x_p) - \textit{current input}$$

What do we want

What to forget:

What to remember:

What to output:

dream recurrent cell

What do we have:

$$s_{t-1} = (s_1, s_2, ..., s_n) - \textit{previous state}$$
$$h_{t-1} = (h_1, h_2, ..., h_q) - \textit{previous output}$$
$$x_{t-1} = (x_1, x_2, ..., x_p) - \textit{current input}$$

What do we want:

What to forget: $\tilde{s}_{t-1} = f(x_t, h_{t-1}) \cdot s_{t-1}, F \in [0,1]^n$

What to remember: $\tilde{s}_t = r(x_t, h_{t-1}) \cdot m(x_t, h_{t-1})$

$r \in [0,1]^n, \ m \in [-1,1]^n$

new state: $s_t = \tilde{s}_{t-1} + \tilde{s}_t$

What to output: $h_t = act(W \cdot (x_t, h_{t-1}) + b) \cdot i(s_t)$

dream recurrent cell

What do we want:

What to forget: $\tilde{s}_{t-1} = f(x_t, h_{t-1}) \cdot s_{t-1}, F \in [0,1]^n$

What to remember: $\tilde{s}_t = r(x_t, h_{t-1}) \cdot m(x_t, h_{t-1})$

$r \in [0,1]^n, \ m \in [-1,1]^n$

new state: $s_t = \tilde{s}_{t-1} + \tilde{s}_t$

What to output: $h_t = act(W \cdot (x_t, h_{t-1}) + b) \cdot i(s_t)$

What we do not have:

$f \in [0,1]^n -$ *forgetting function*

$r \in [0,1]^n, \ m \in [-1,1]^n,$ *remembering function and memories itself*

$i \in [-1,1]^n -$ *impact function*

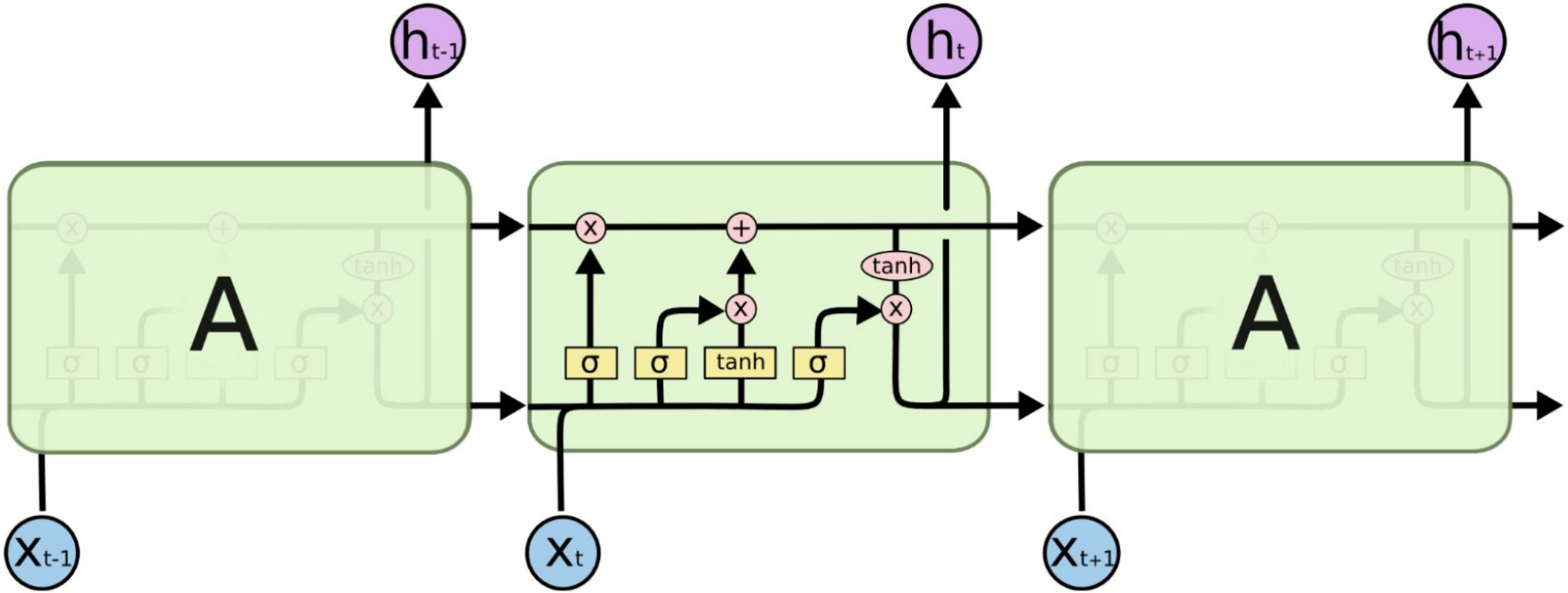
What we can have:

$$f \in [0,1]^n F = \sigma(W_f \cdot (x_t, h_{t-1}) + b_f)$$
$$r \in [0,1]^n, \ \ m \in [-1,1]^n,$$

$$r = \sigma(W_r \cdot (x_t, h_{t-1}) + b_r),$$
$$m = tanh(W_m \cdot (x_t, h_{t-1}) + b_m)$$

$$i \in [-1,1]^n, \ \ i = tanh(s_t)$$

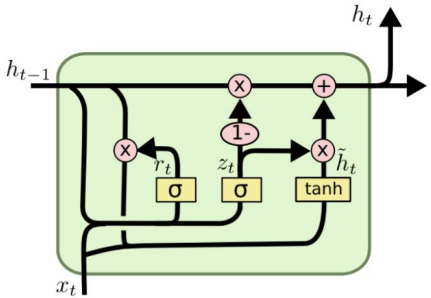
LSTM



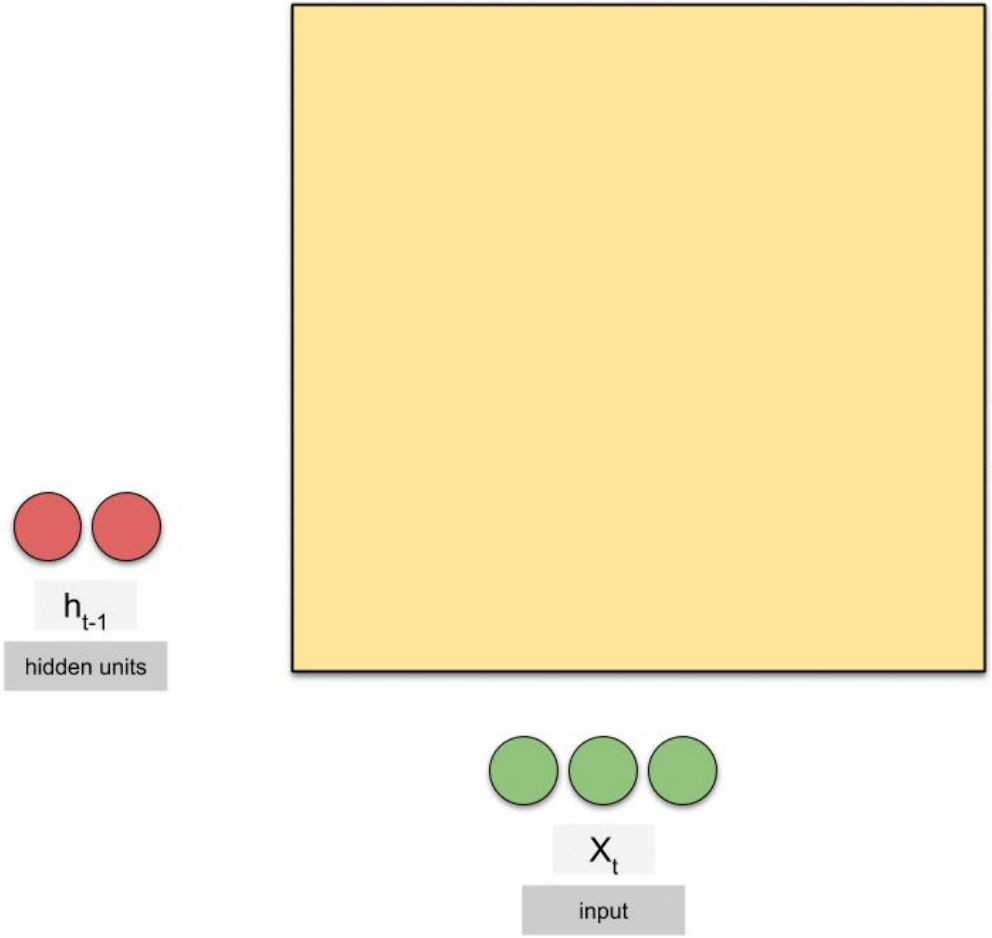
Let's make LSTM little bit simpler by removing it's state. It's role will be performed by previous output.

$$z = \sigma(W_z \cdot (x_t, h_{t-1})),\ z \in [0,1]^q$$
$$r = \sigma(W_r \cdot (x_t, h_{t-1})),\ r \in [0,1]^q$$
$$\tilde{h_t} = tanh(W \cdot (x_t, r \cdot h_{t-1}))$$
$$h_t = (1-z) \cdot h_{t-1} + z * \tilde{h_t}$$

$$z = \sigma(W_z \cdot (x_t, h_{t-1})),\ z \in [0,1]^q$$
$$r = \sigma(W_r \cdot (x_t, h_{t-1})),\ r \in [0,1]^q$$
$$\tilde{h_t} = tanh(W \cdot (x_t, r \cdot h_{t-1}))$$
$$h_t = (1-z) \cdot h_{t-1} + z * \tilde{h_t}$$

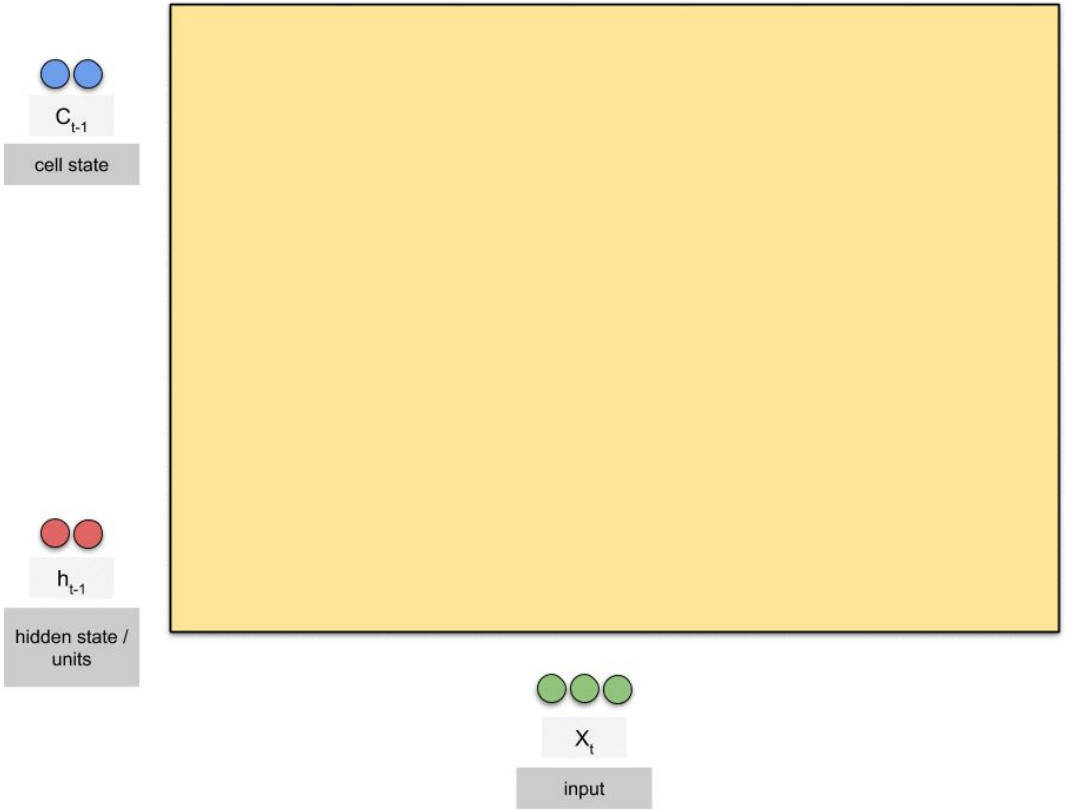


RNN



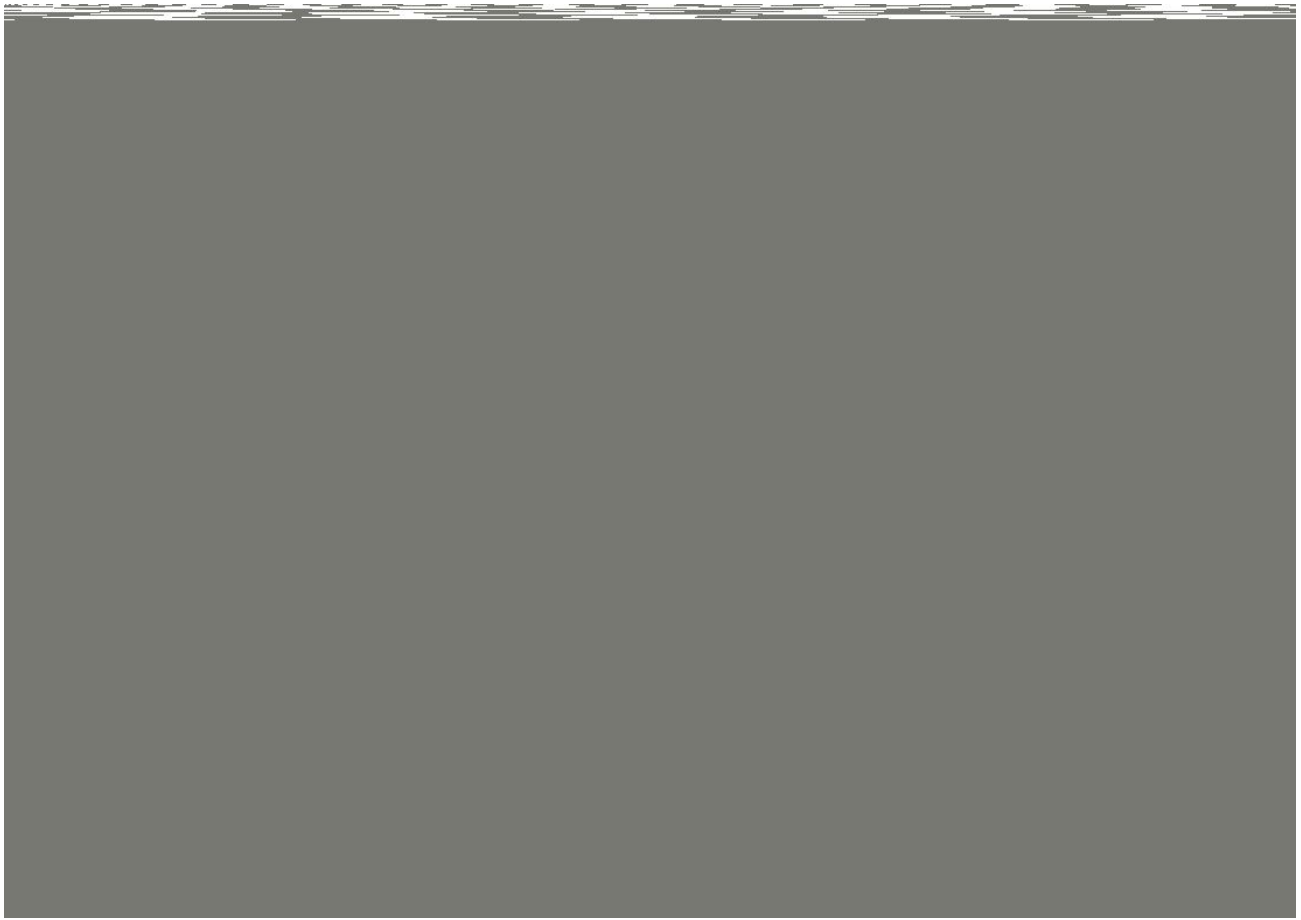
https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45

LSTM



https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45

GRU



https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45

Links

- https://towardsdatascience.com/simple-guide-for-ensemble-learning-methods-d87cc68705a2
- https://habr.com/ru/company/yandex/blog/349372/
- https://habr.com/ru/company/yandex/blog/339638/
- https://habr.com/ru/company/mailru/blog/358736/
- https://habr.com/ru/company/Voximplant/blog/446738/
- https://arxiv.org/abs/1810.04805
- https://arxiv.org/abs/1409.3215
- https://data-science-blog.com/blog/2017/12/03/ensemble-learning/

Links

- https://towardsdatascience.com/simple-guide-for-ensemble-learning-methods-d87cc68705a2
- https://botsociety.io/blog/2018/03/the-turing-test/
- https://medium.com/jatana/unsupervised-text-summarization-using-sentence-embeddings-adb15ce83db1
- https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45
- https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- https://habr.com/ru/company/wunderfund/blog/331310/