

# 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](#)

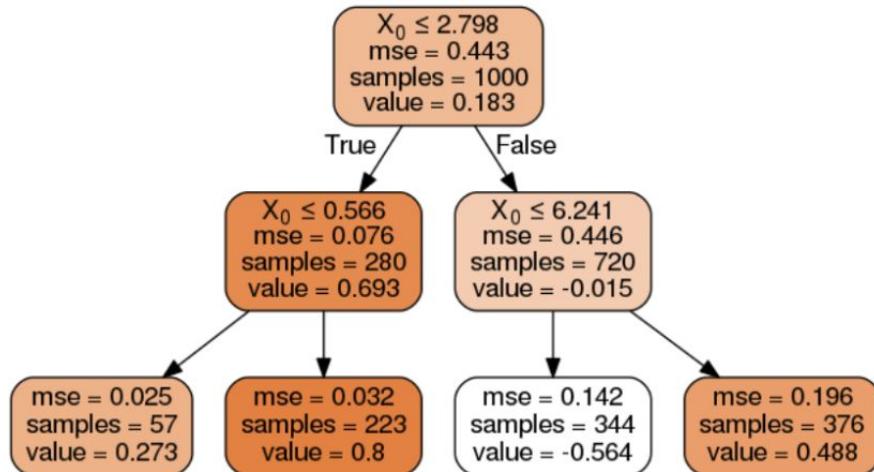
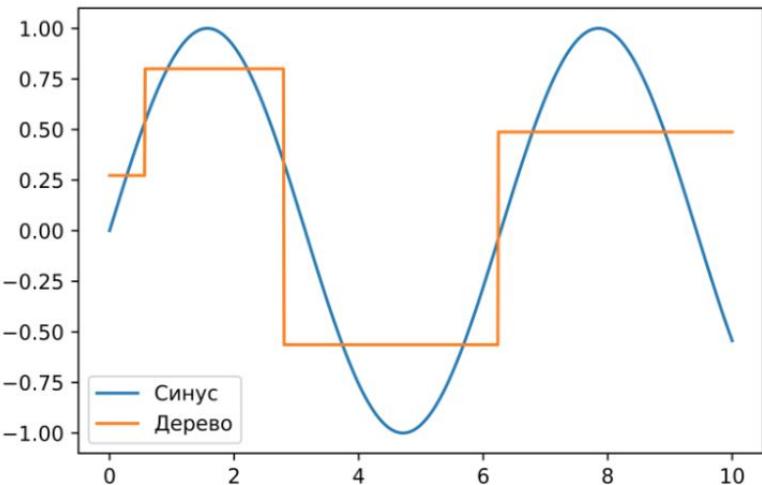


<http://www.shivambansal.com/blog/kaggle-bot/>

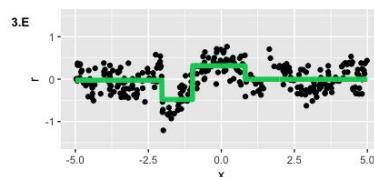
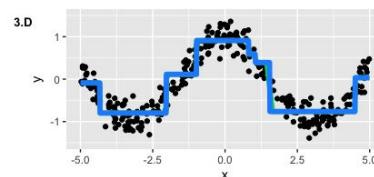
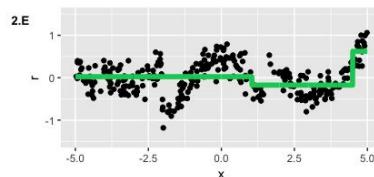
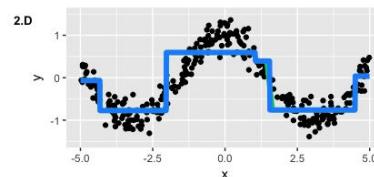
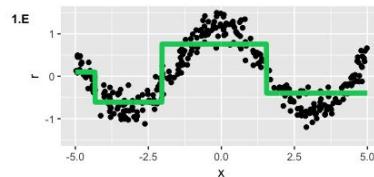
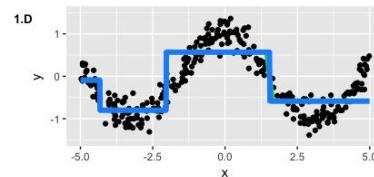
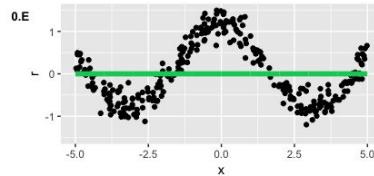
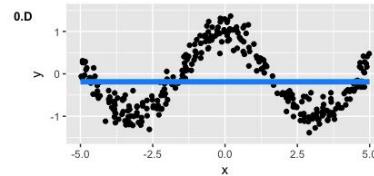
# План занятия

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

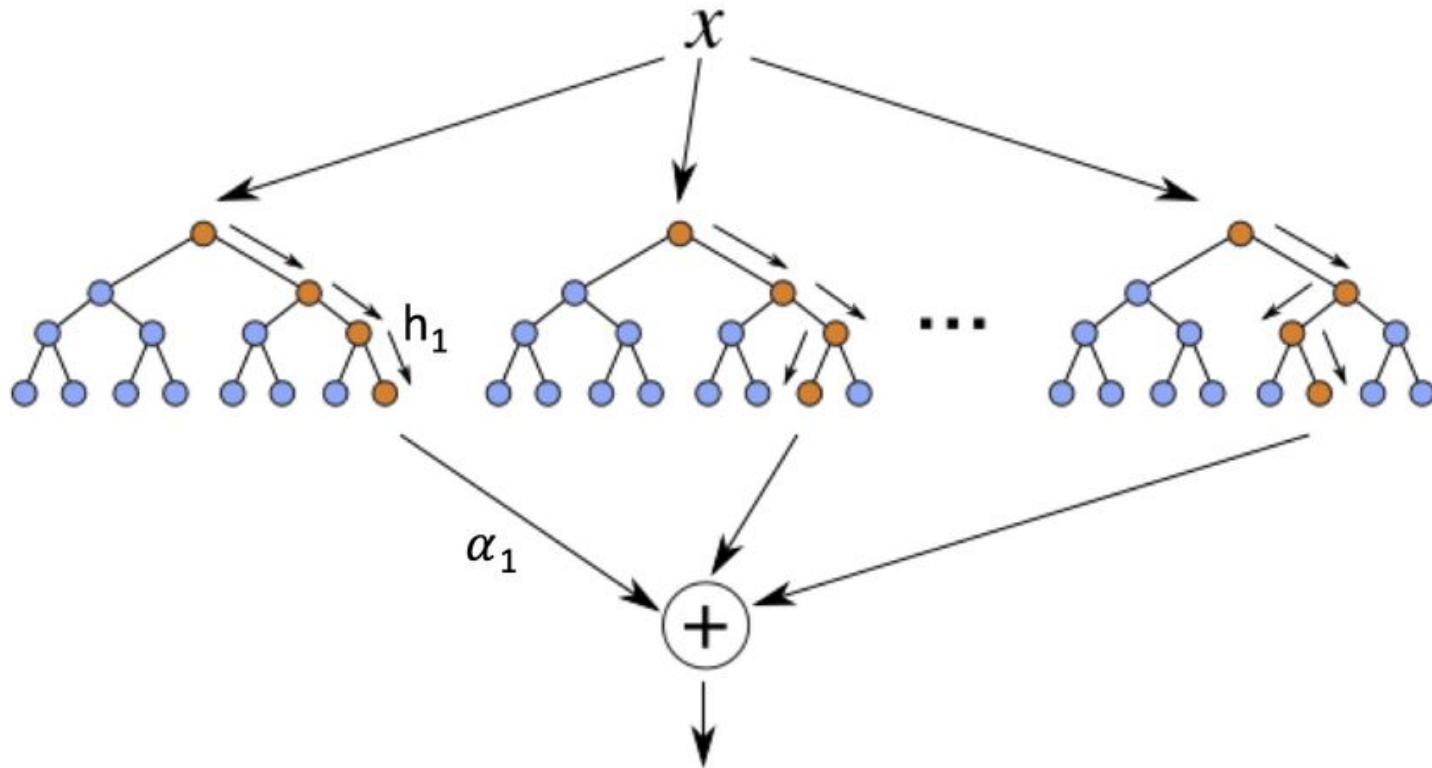
# Напоминание с прошлой лекции



# Напоминание с прошлой лекции



# Напоминание с прошлой лекции



# Ансамбли - МОТИВАЦИЯ

Предположим, вы хотите купить новые наушники.

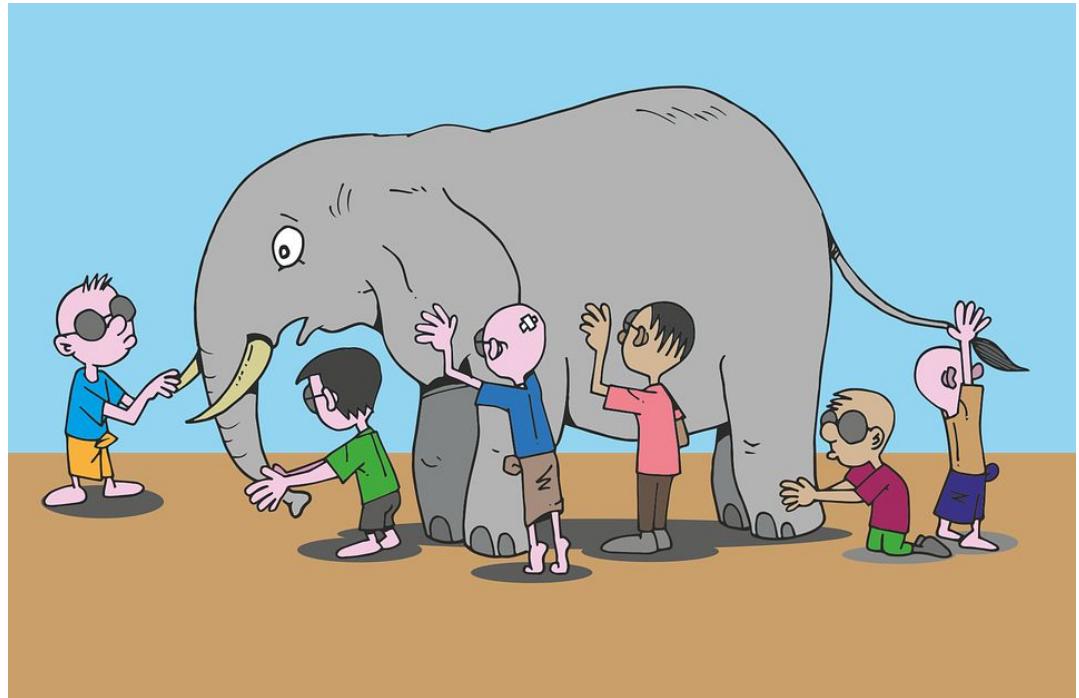
- Пойдёте ли вы в первый магазин и купите первые попавшиеся наушники?
- А, может, вы пойдёте в первый магазин, изучите ассортимент и купите понравившиеся наушники?
- А, может, вы сравните цены на Яндекс.Маркет, выберите самый выгодный вариант и купите понравившиеся наушники?
- А, может, вы проведёте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

# Ансамбли

**Зачем** нужны ансамбли?

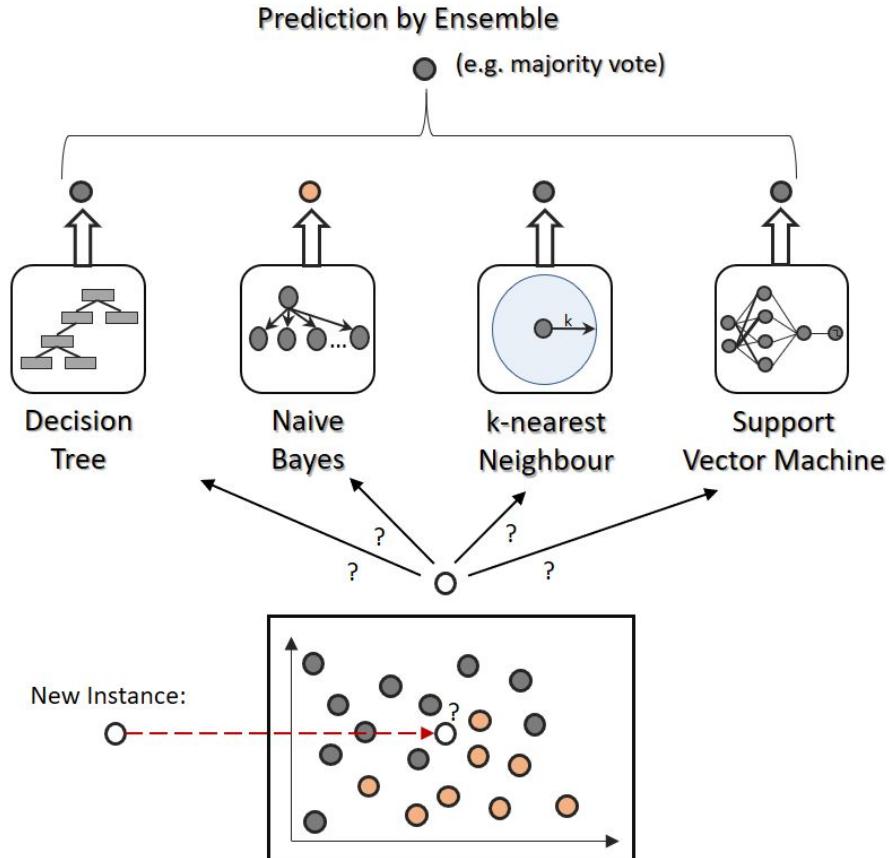
Ансамблевые методы позволяют объединить решения многих моделей для улучшения общего качества предсказания.

Bias, Variance, Noise



# Ансамбли - простые техники

- Мода предсказаний
- Среднее предсказаний
- Взвешенное среднее предсказаний



# Ансамбли - пример

1111111111

Пусть есть 3 классификатора А, В, С 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

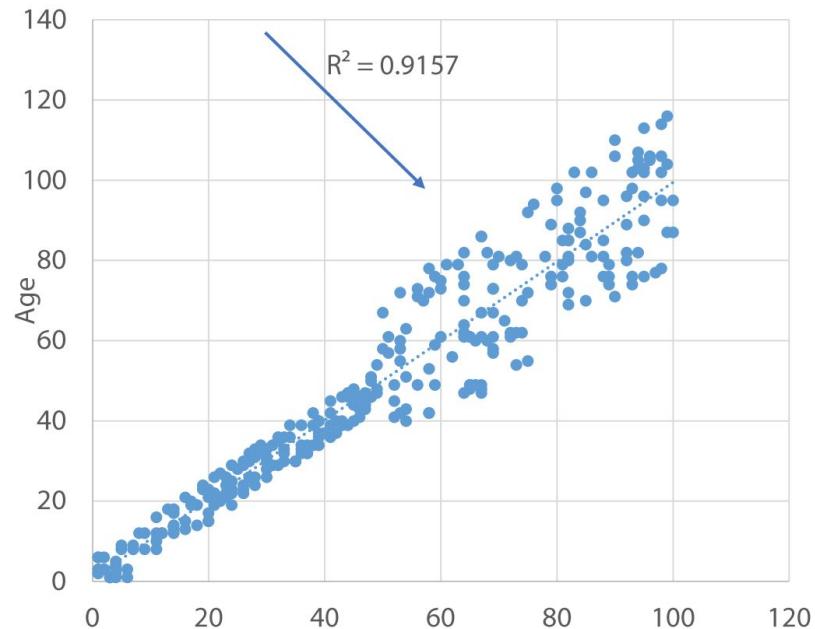
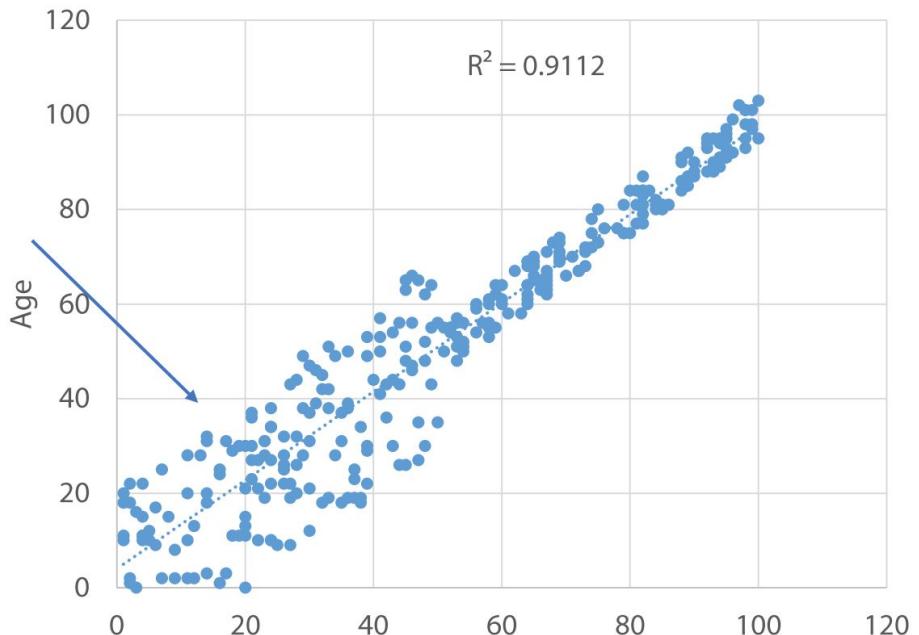
Для голосования среди 3 моделей можно ожидать 4 варианта:

- 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:  $0.3 * 0.3 * 0.3 = 0.027$

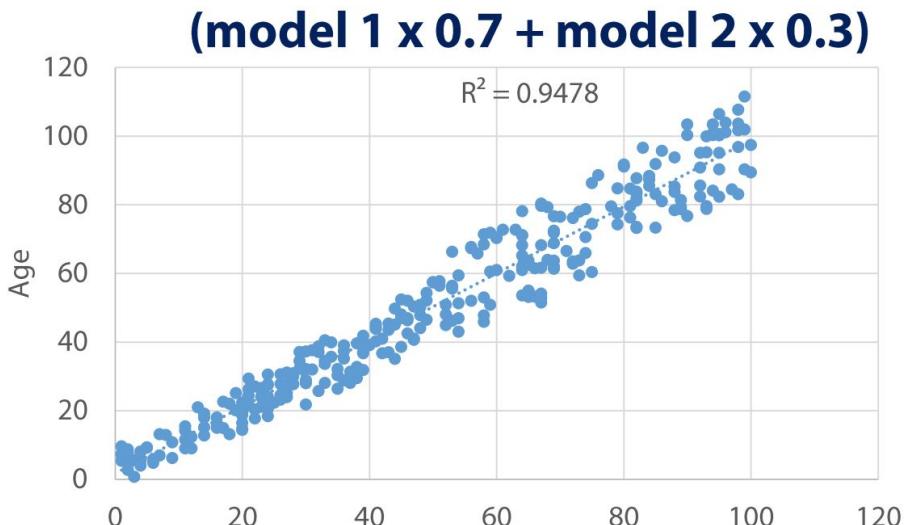
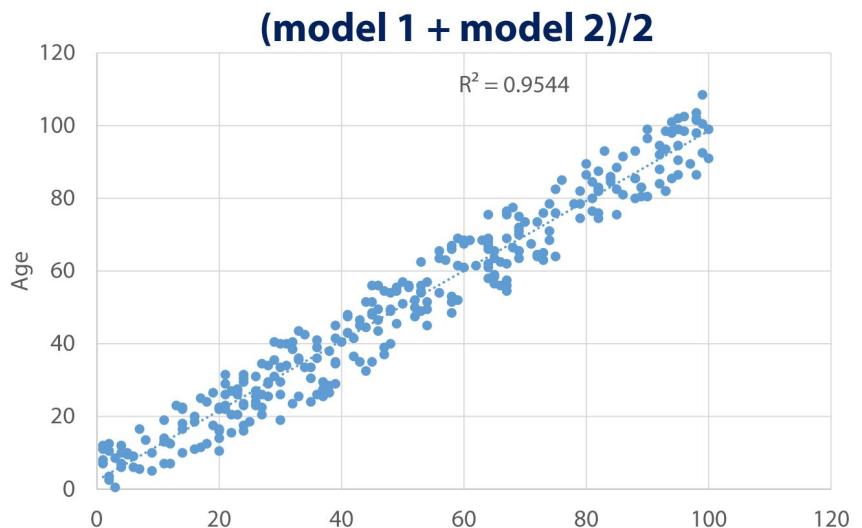
Итоговая точность ансамбля:  $0.3429 + 0.4409 = 0.7838$

Корреляция??

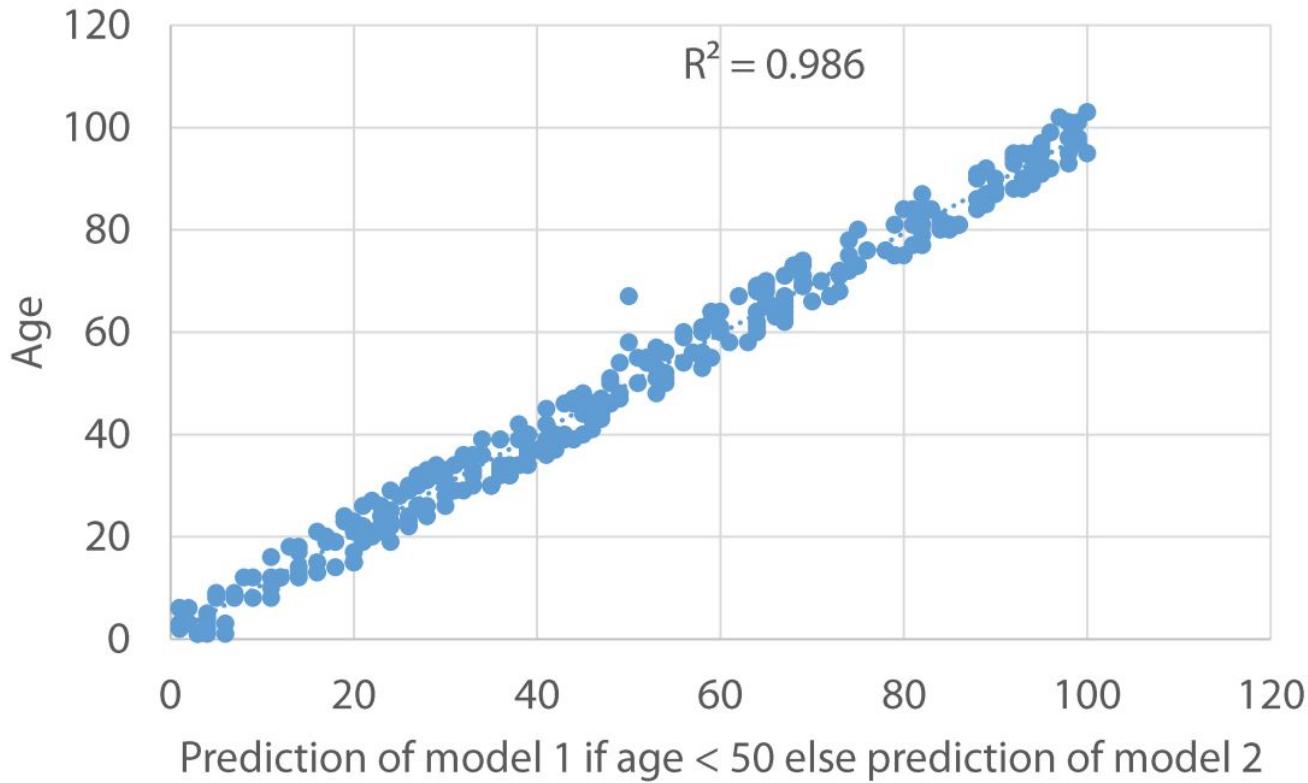
# Ансамбли - усреднение



# Ансамбли - усреднение



# Ансамбли - if



# 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
```

# Ансамбли - boosting

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

<b>Rownum</b>	<b>x0</b>	<b>x1</b>	<b>x2</b>	<b>x3</b>	<b>y</b>
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

<b>Rownum</b>	<b>x0</b>	<b>x1</b>	<b>x2</b>	<b>x3</b>	<b>y</b>	<b>pred</b>
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

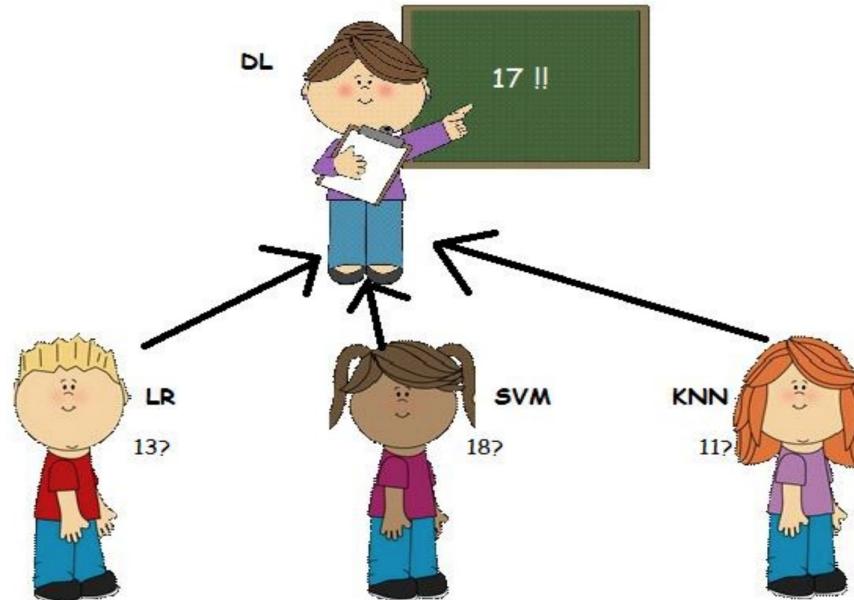
# Ансамбли - boosting

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

# Ансамбли - stacking

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



# 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**

pred0
0.24
0.95
0.64
0.89
0.11

B1

pred0
0.50
0.62
0.22
0.90
0.20

C1

# 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

# 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	?

# 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

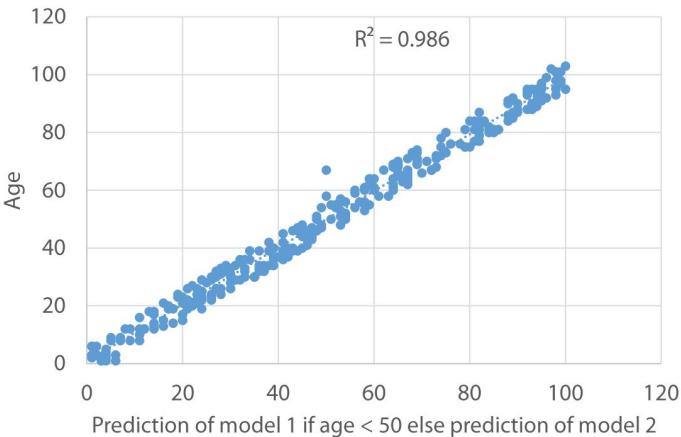
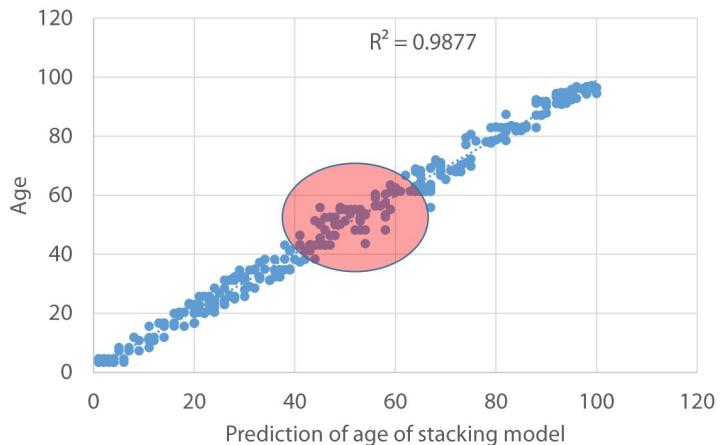
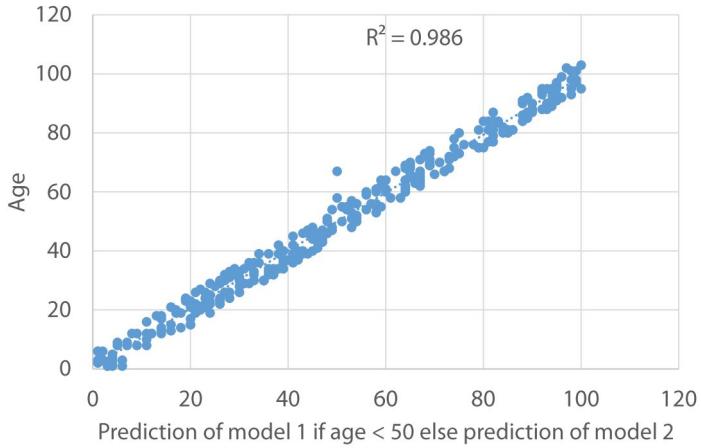
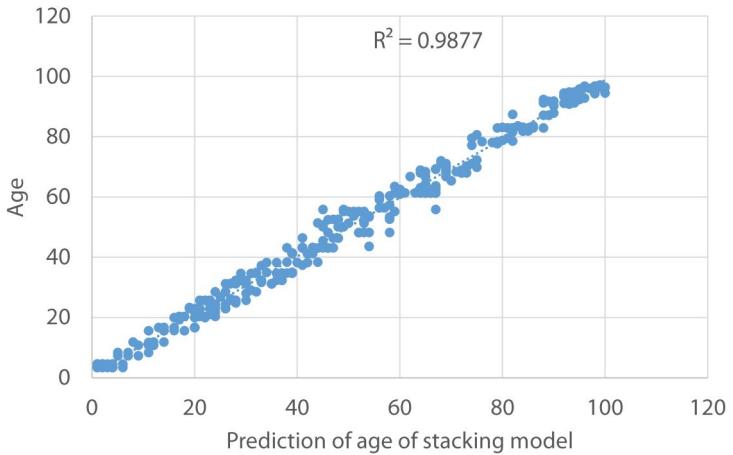
C1				
pred0	pred1	pred2	y	Preds3
0.50	0.50	0.39	?	0.45
0.62	0.59	0.46	?	0.23
0.22	0.31	0.54	?	0.99
0.90	0.47	0.09	?	0.34
0.20	0.09	0.61	?	0.05

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

## 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

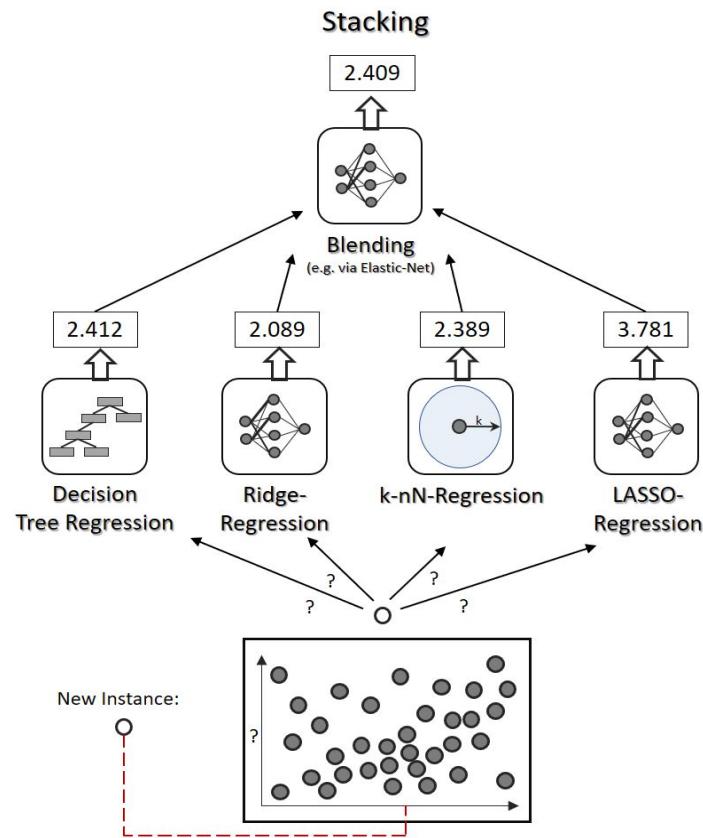
# train is the training data
# y is the target variable for the train data
# test is the test data
```

# Ансамбли - stacking

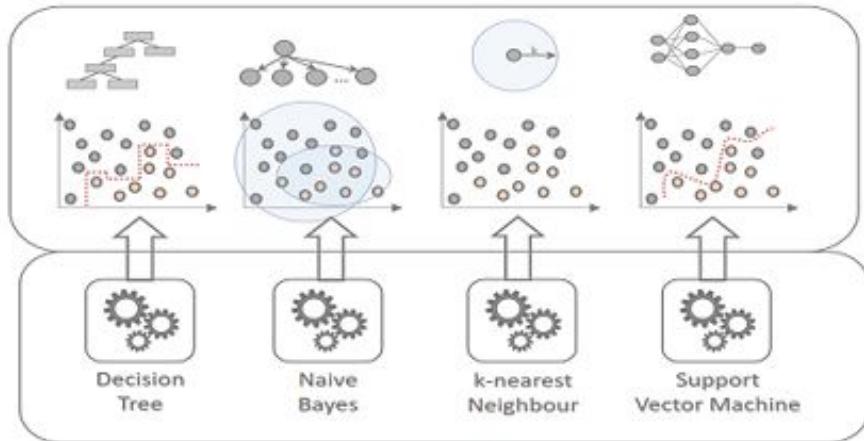
## Stacking example

```
#split train data in 2 parts, training and validation.
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



## Ensemble Learning



# Ансамбли - stacking

## 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

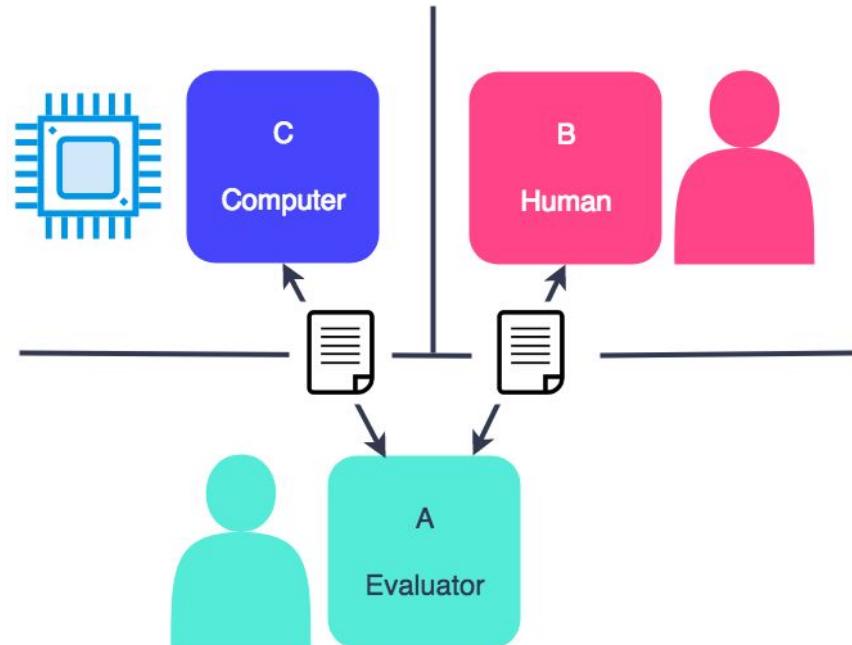
# Ансамбли: минусы

- 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.

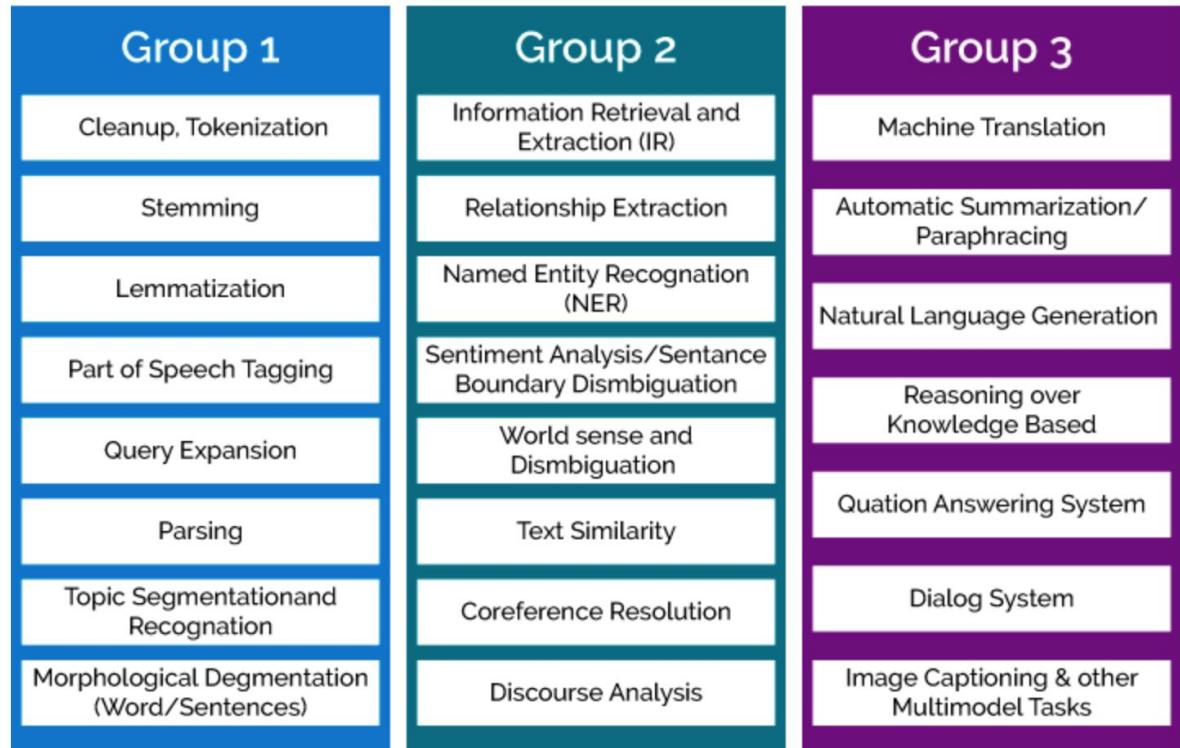
- 1950 - первые попытки, rule based
- 1980-2010 статистические методы + первые ML методы (Decision Trees)
- 2010s Neural networks + некоторые статистические методы



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

# NLP - tasks

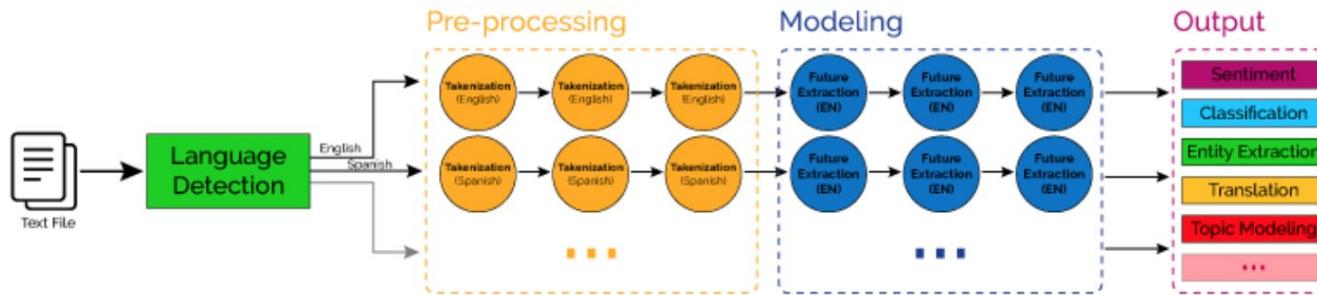
- Machine Translation
- Language modeling
- Part of speech tagging
- Parsing
- Summarization
- Sentiment Analysis
- Dialog Systems
- NER
- Question Answering
- Topic Modelling
- ...



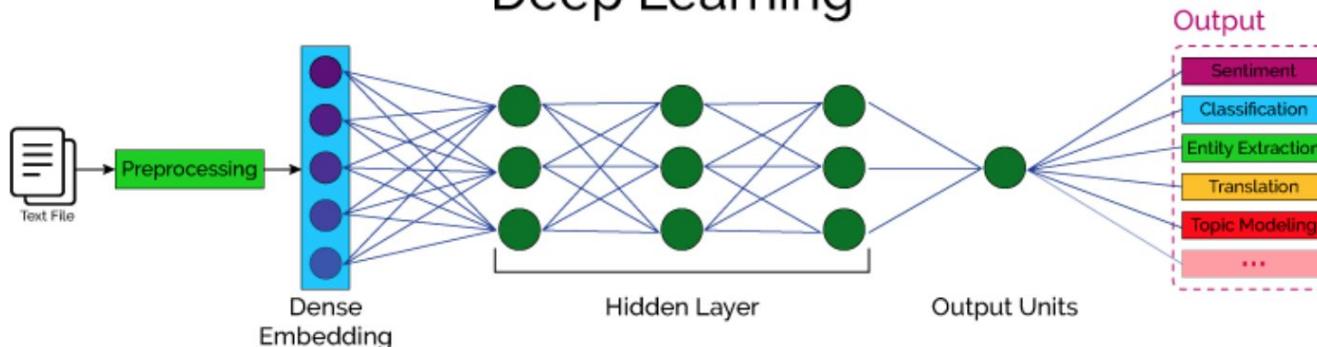
<https://www.upwork.com/hiring/for-clients/artificial-intelligence-and-natural-language-processing-in-big-data/>

# NLP - tasks

## Classical NLP



## Deep Learning



# 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 сравнивать 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	+	-	-

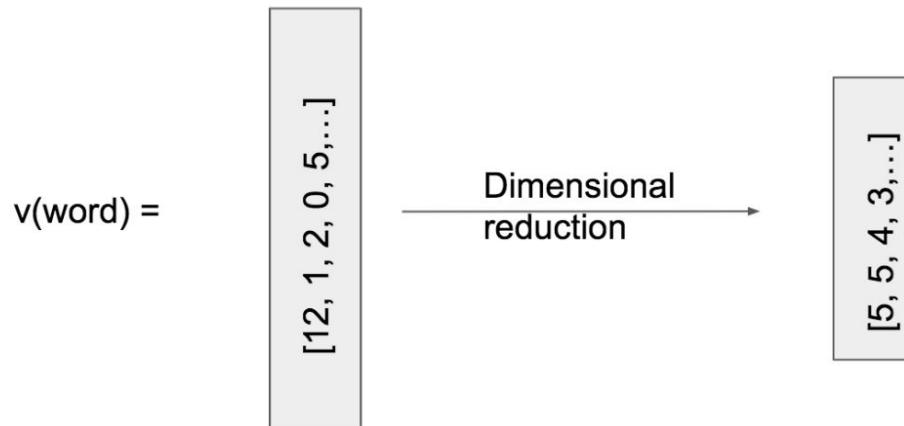
# NLP - context embeddings

$v(\text{word}_i)[j] = \text{count}(\text{co-occurrences word}_i \text{ with word}_j)$

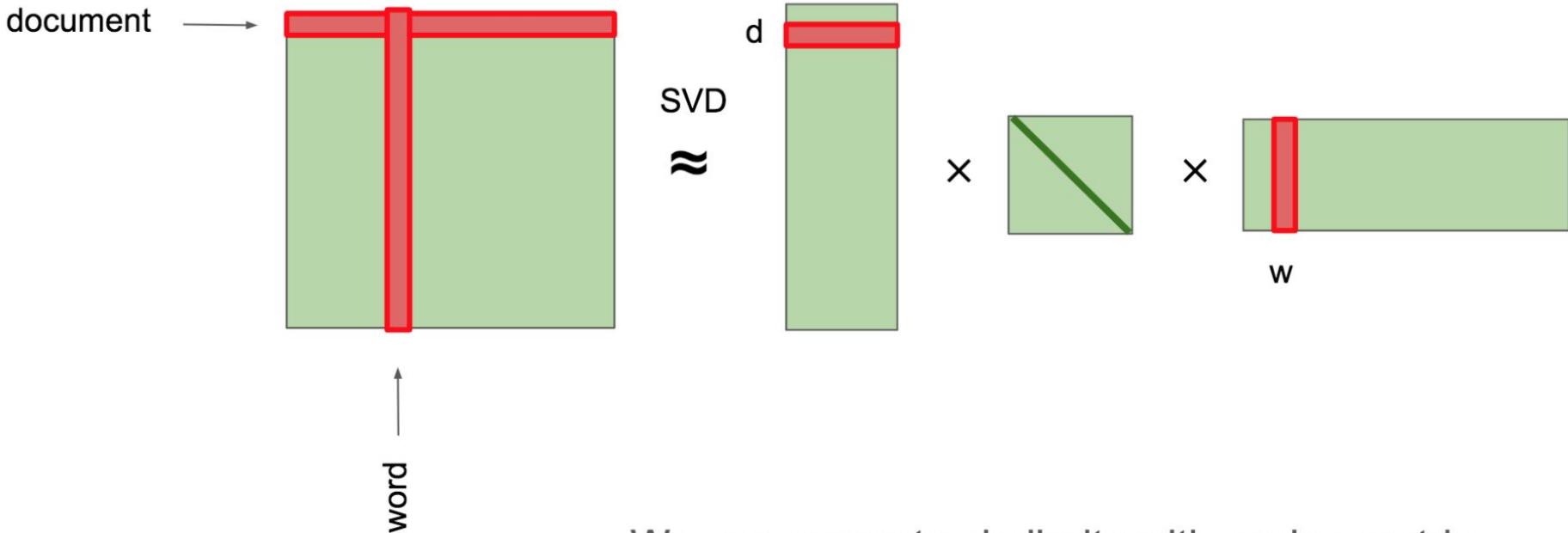
$v(\text{word}) = [12, 1, 2, 0, 5, \dots]$

Опять же большой вектор (размер словаря)

Можно сжать с помощью методов понижения размерности



# NLP - latent semantic analysis



# NLP - проблемы методов

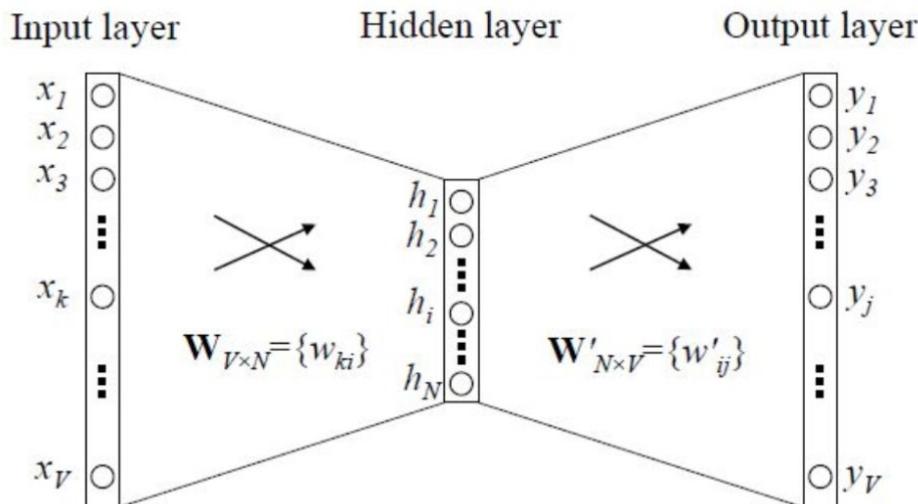
- Редкие слова
- Большое время для вычислений
- Неустойчивость к небольшим изменениям датасета

Что мы хотим?

Получить представления слов без правил, придуманных человеком.

# NLP - word embeddings

## Word2Vec

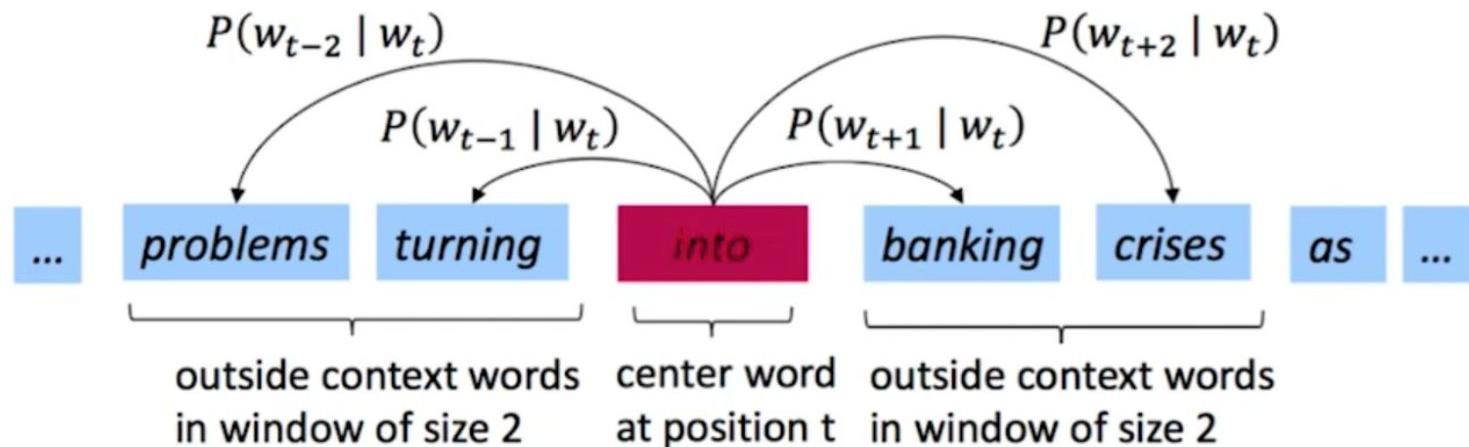


- Predict context words given a word (or vice versa)
- maximize probability of seeing word and its context together

# 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**):

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

# Word2Vec objective function

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

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

OR we want to minimize **negative log-likelihood**:

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

# 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”

## *Log-likelihood* maximization:

$$\mathcal{L} = \sum_{u \in W} \sum_{v \in W} n_{uv} \log p(u|v)$$

*word co-occurrence*

### Method:

- SGD, online by word pairs in the corpus

### Problem:

- *softmax* over vocabulary is slow!

Instead of predicting a word for another word,  
predict “yes” or “no” for word pairs:

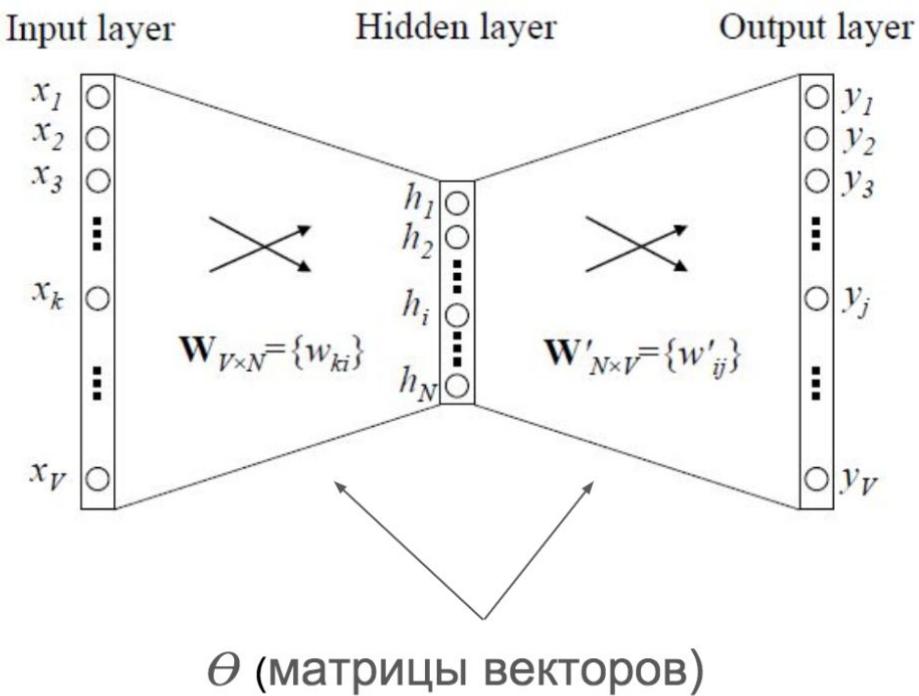
$$\sum_{u \in W} \sum_{v \in W} n_{uv} \log \sigma (\langle \phi_u, \theta_v \rangle) +$$

$$k \mathbb{E}_{\bar{v}} \log \sigma (-\langle \phi_u, \theta_{\bar{v}} \rangle) \rightarrow \max_{\phi_u, \theta_v}$$

- Use **positive examples** from data:  $v$  co-occurred with  $u$
- Sample **negative examples**:  $k$  random  $\bar{v}$  from the vocabulary

Train with SGD to find two matrices of parameters (as usual).

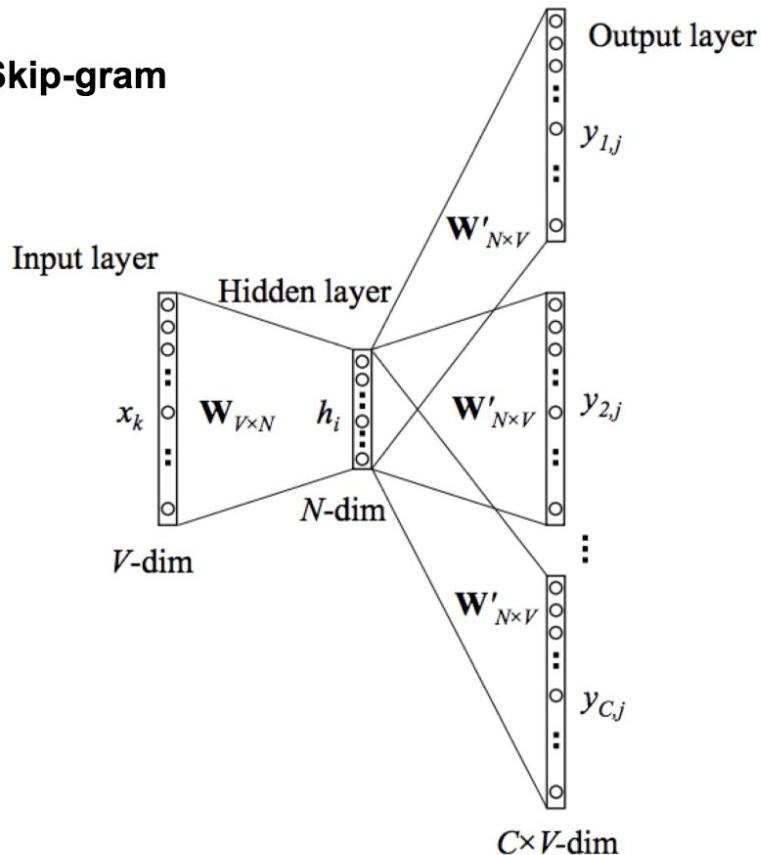
# Word2Vec



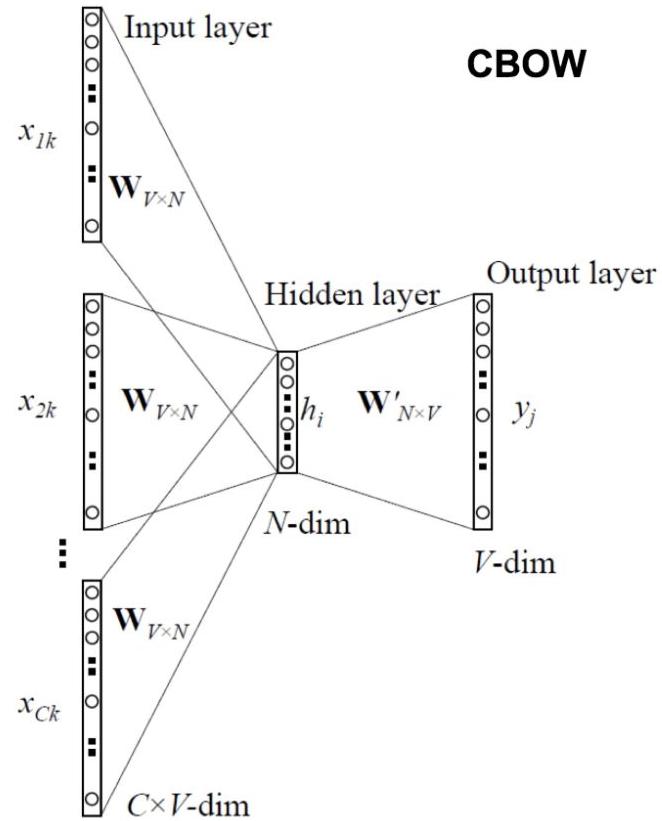
$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

# Word2Vec

**Skip-gram**

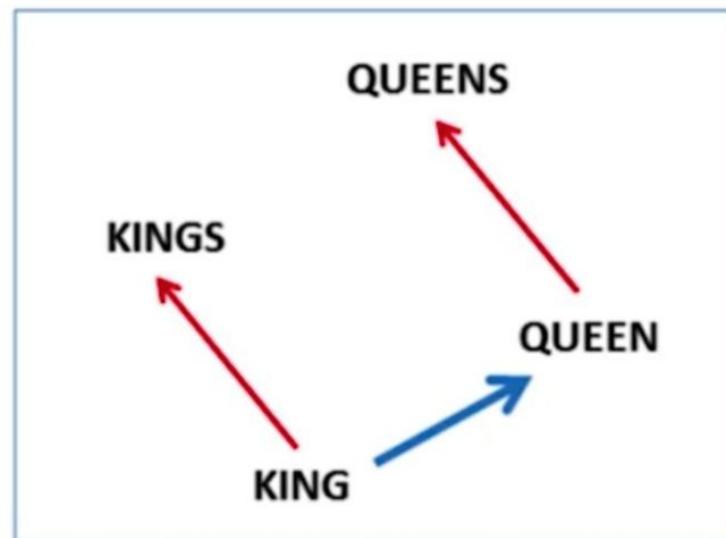
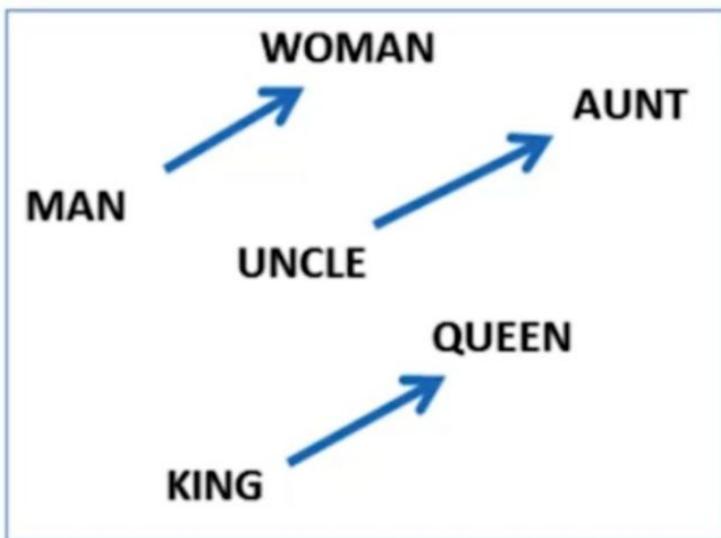


**CBOW**

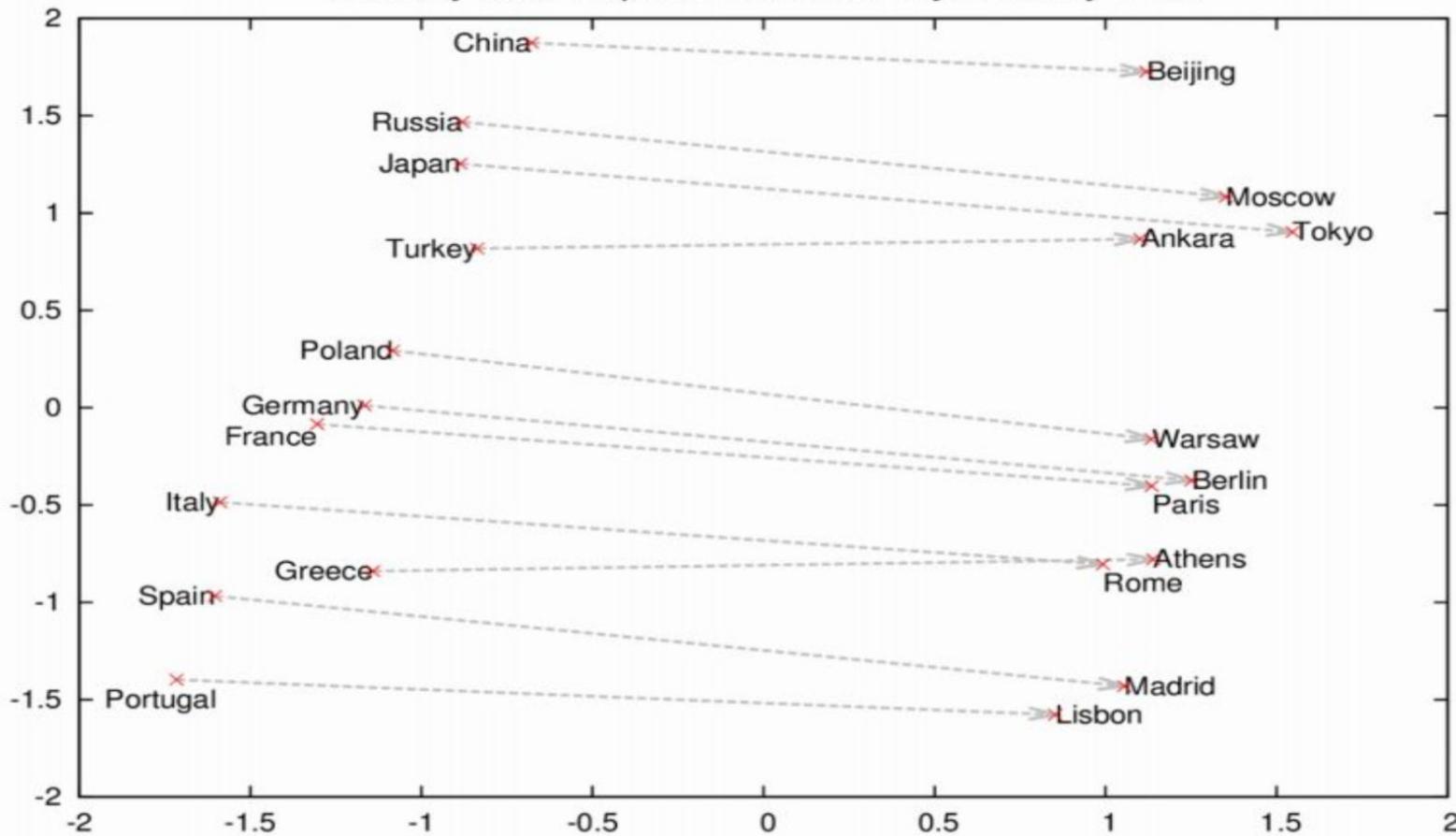


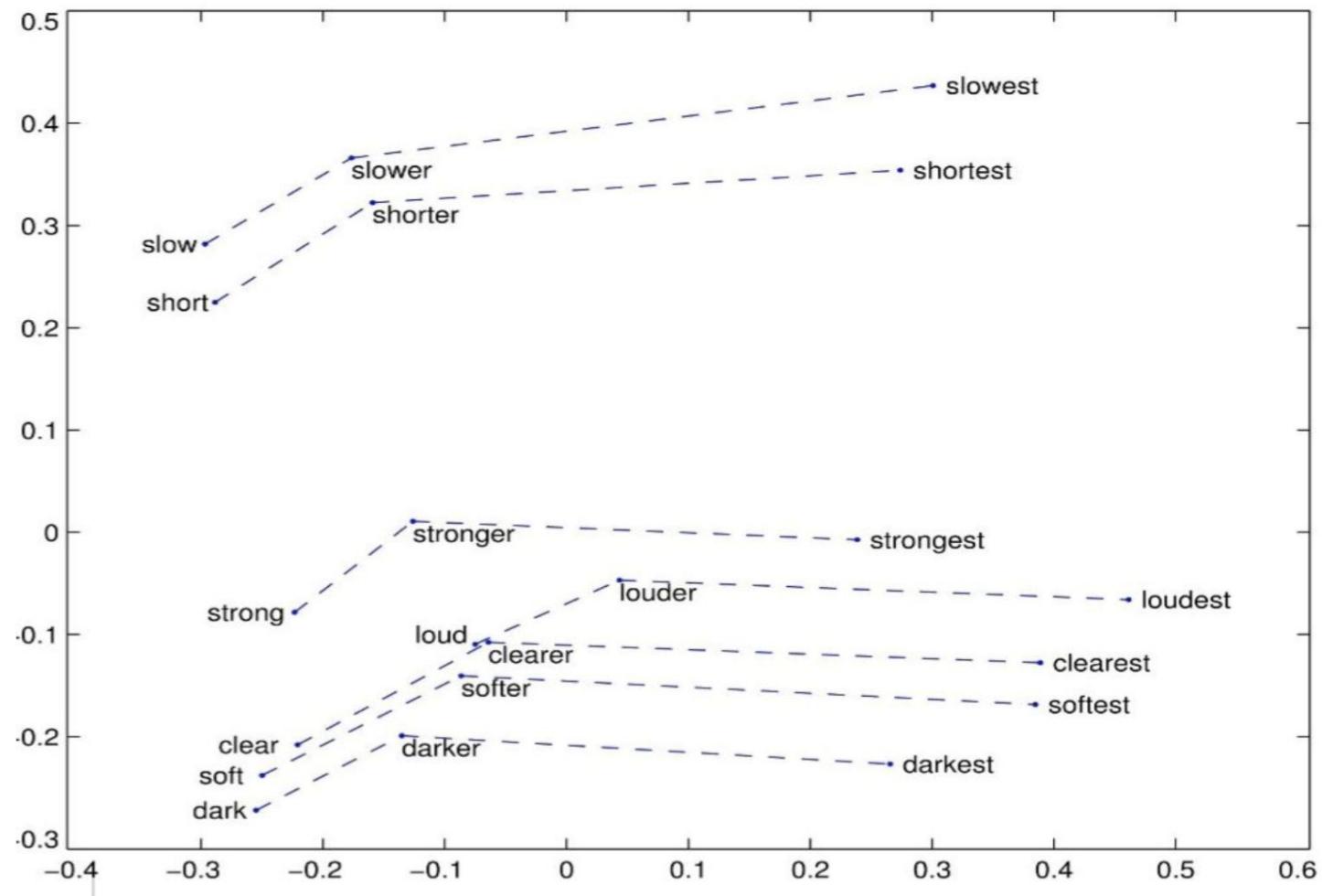
# Word2Vec

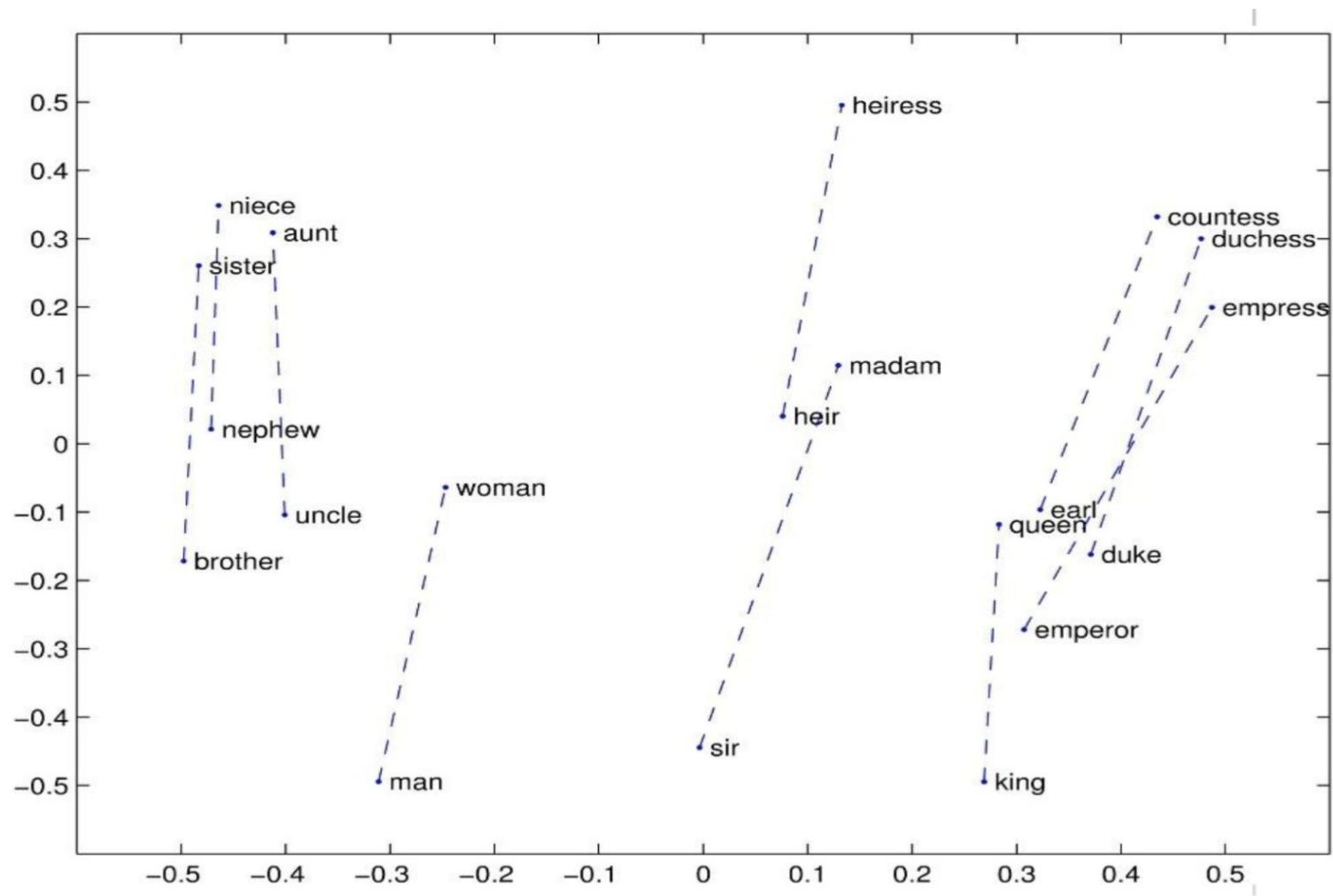
$$v(\text{king}) - v(\text{man}) + v(\text{woman}) \approx v(\text{queen})$$

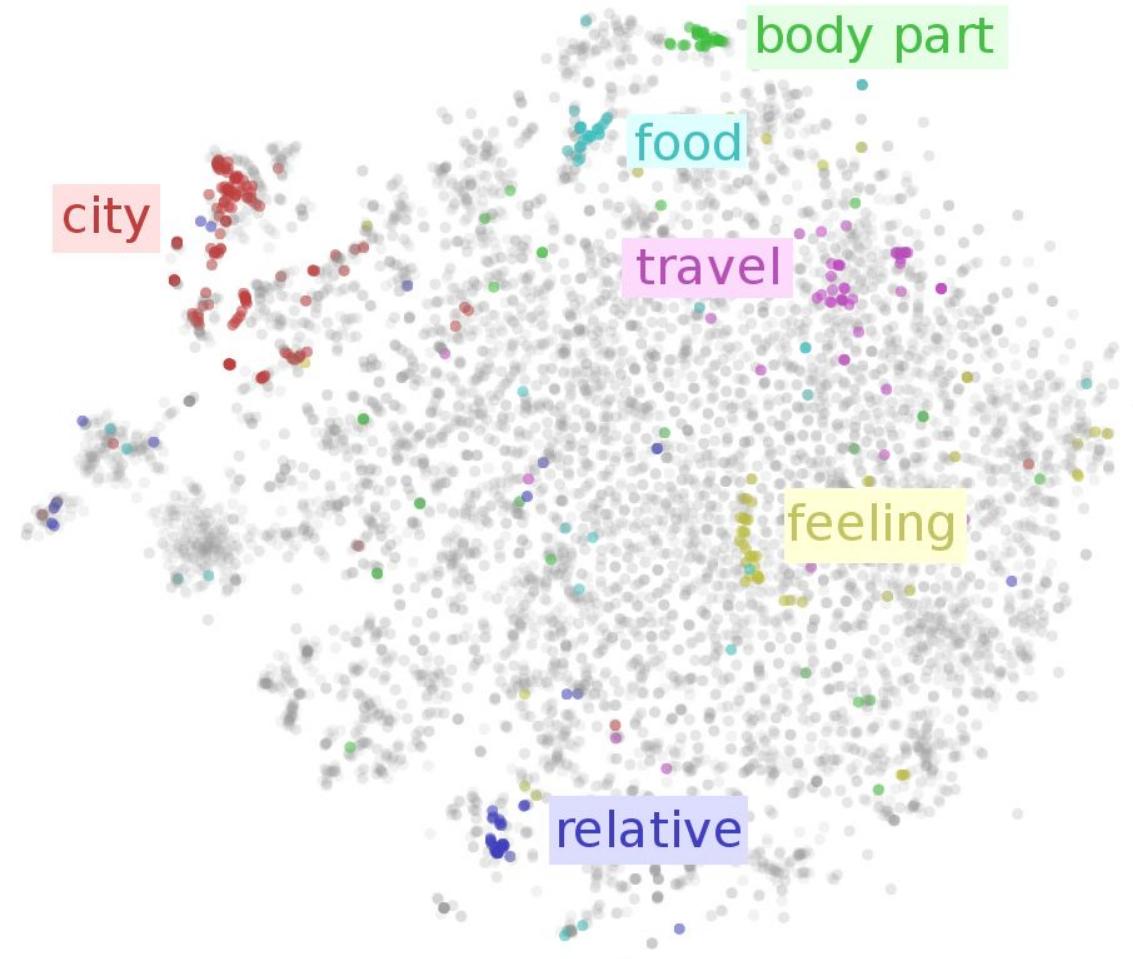


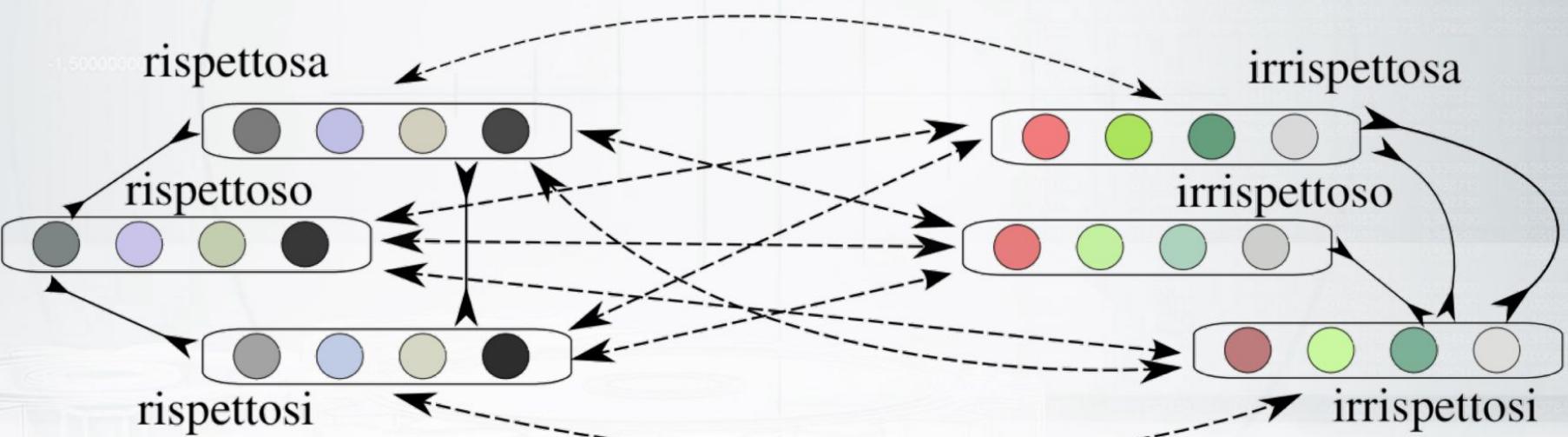
### Country and Capital Vectors Projected by PCA











Use morphology to improve word embeddings

# FastText

Represent a word as a bag of character n-grams, e.g. for n = 3:

$$G_{\text{where}} : \text{\_wh, whe, her, ere, re\_, where\_}$$

Model a word vector as a sum of sub-word vectors:

**SGNS:**

$$\text{sim}(u, v) = \langle \phi_u, \theta_v \rangle$$

**FastText:**

$$\text{sim}(u, v) = \sum_{g \in G_v} \langle \phi_u, \theta_g \rangle$$

Code and pre-trained embeddings: <https://fasttext.cc/>

# Sent2Vec

## First ideas:

- Average pre-trained word vectors (*word2vec*, *GloVe*, etc).
- Maybe use TF-IDF weights for averaging.

## Sent2vec:

- Learn sentence embedding as a sum of sub-sentence units:

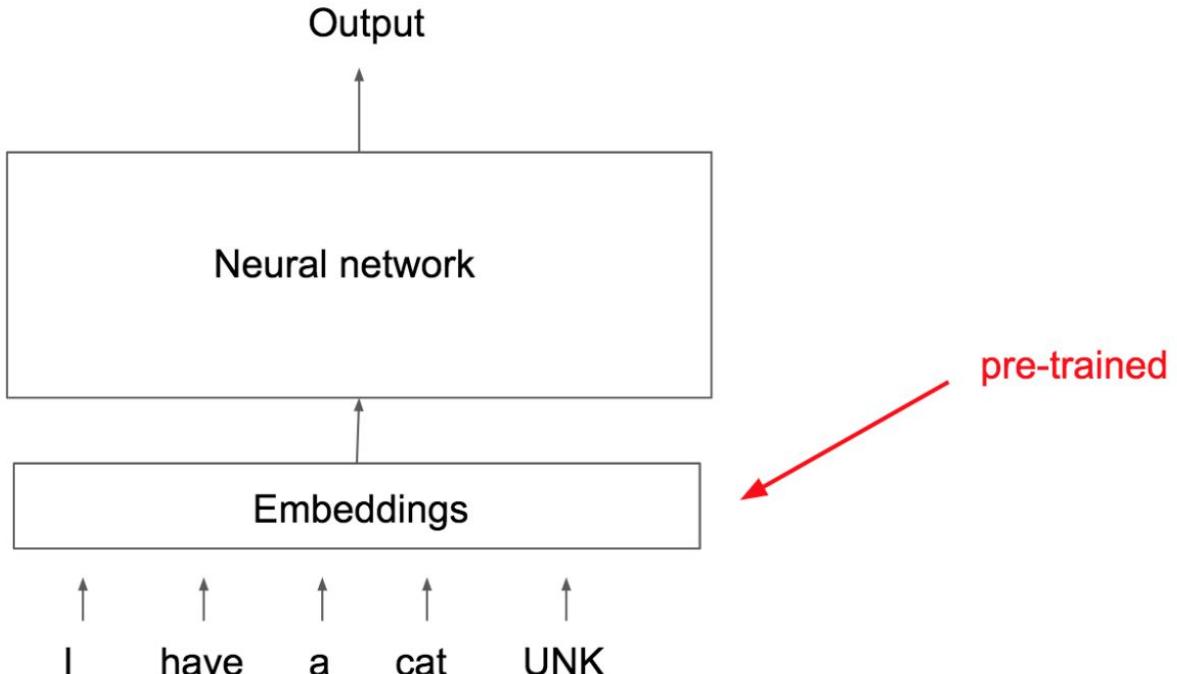
$$sim(u, s) = \frac{1}{|G_s|} \sum_{g \in G_s} \langle \phi_u, \theta_g \rangle$$

where  $G_s$  is a set of word n-grams for the sentence  $s$ .

**Code and embeddings:** <https://github.com/epfml/sent2vec>

# 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. ->  $P(\text{I.have.a.very.interesting.idea})$

- 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

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

$$\begin{aligned} P(\text{the cat slept quietly}) &= P(\text{quietly}|\text{the cat slept}) \cdot P(\text{the cat slept}) = \\ &P(\text{quietly}|\text{the cat slept}) \cdot P(\text{slept} | \text{the cat}) \cdot P(\text{cat}|\text{the}) \cdot P(\text{the}) \end{aligned}$$

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)

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

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

$$P(\text{a}|\text{cat seat on})P(\text{cat seat on}) = \dots \cong$$

$$P(\text{mat})P(\text{a})P(\text{on})P(\text{seat})P(\text{cat})$$

# Как это сделать с помощью Neural Networks?

Fixed size solution:

Sample probability of next token, given fixed size input:

$$\text{label} = \text{softmax}(Wx + b)$$

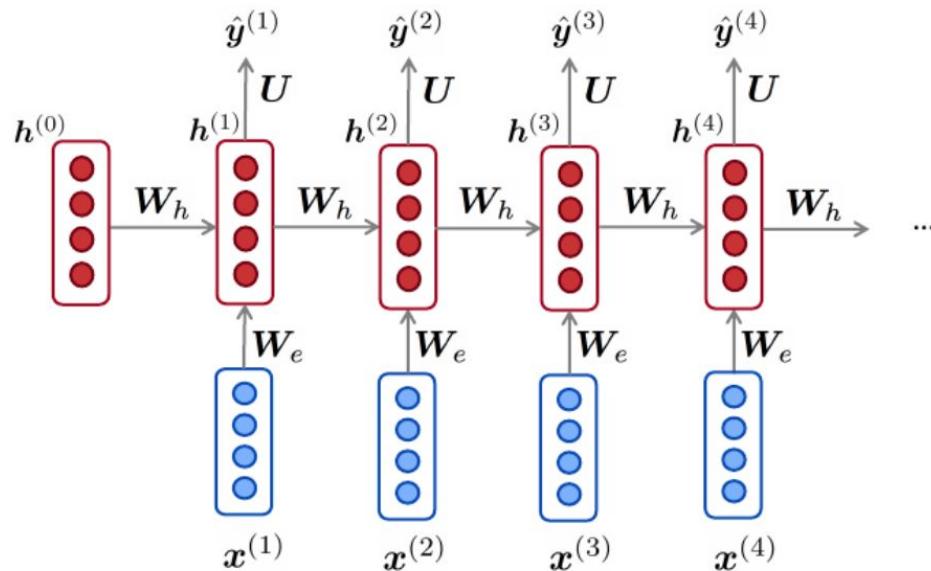
We can concatenate word embeddings:

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

We still have problems with fixed size model...

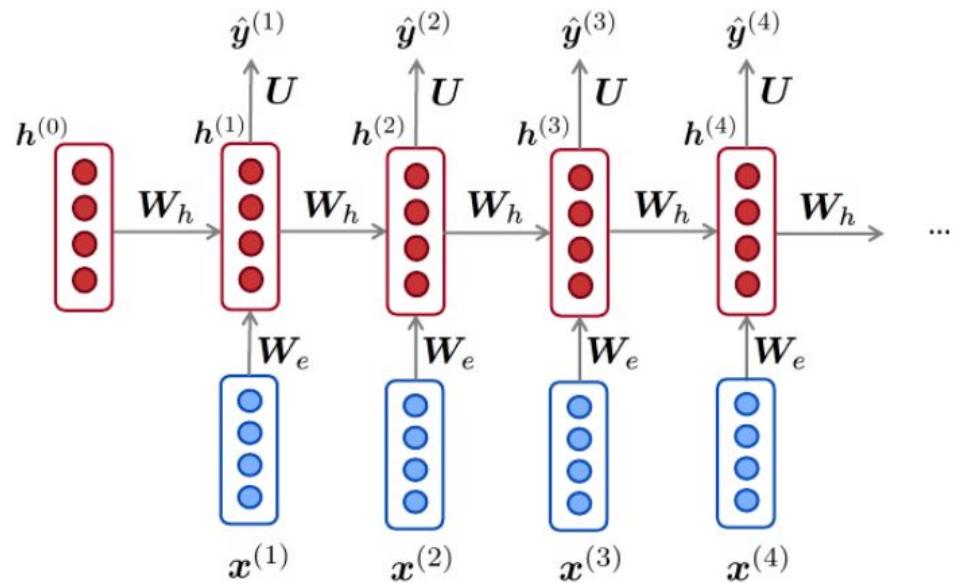
В общем случае мы хотим иметь доступ не только к n-граммам, но и ко всем предыдущим словам. Мы хотим каким-то образом запоминать информацию для предсказывания текущего токена.

## Recurrent neural networks (RNNs)



# Recurrent neural networks (RNNs)

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



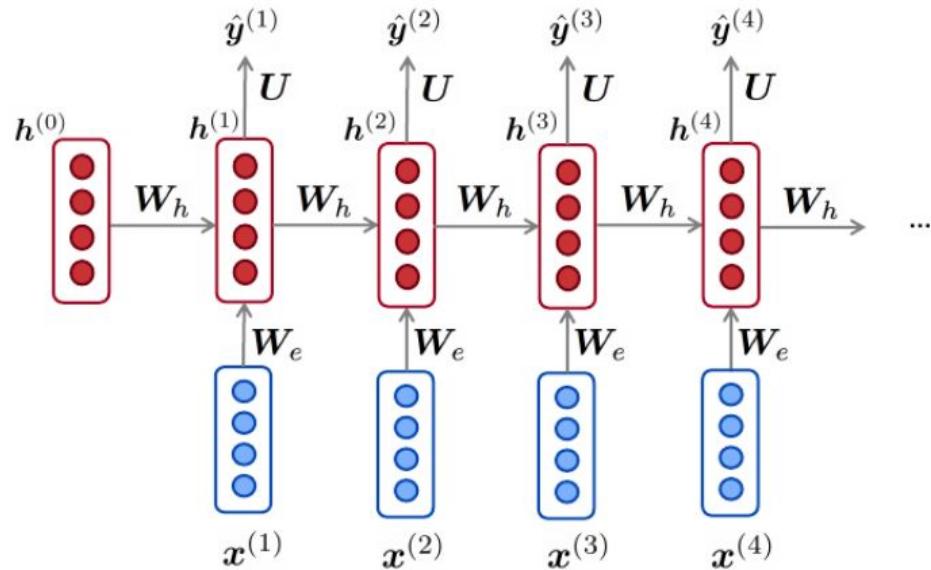
# Recurrent neural networks (RNNs)

$$h_t = Wf(h_{t-1}) + W^{(hx)}x_{[t]}$$

$$\hat{y}_t = W^{(S)}f(h_t)$$

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

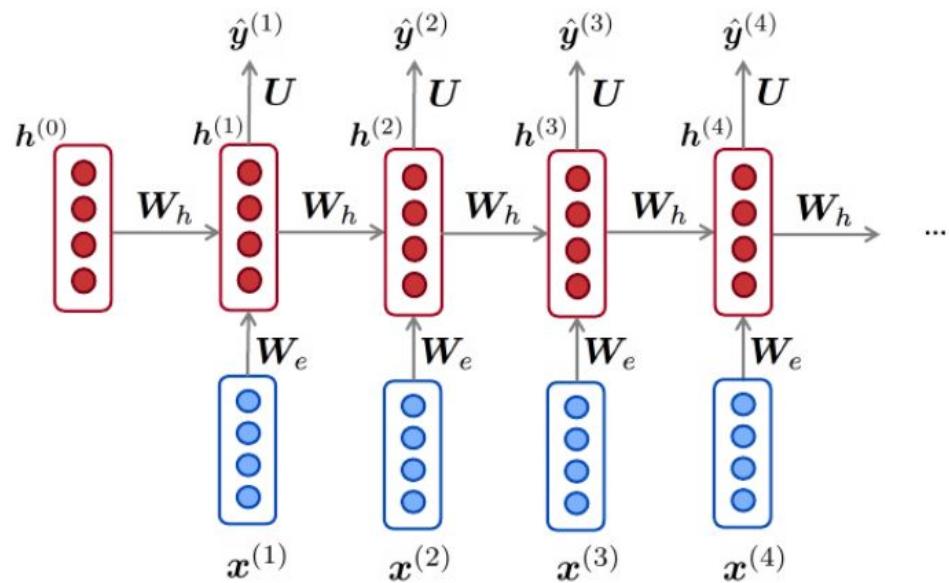
$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$



# 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

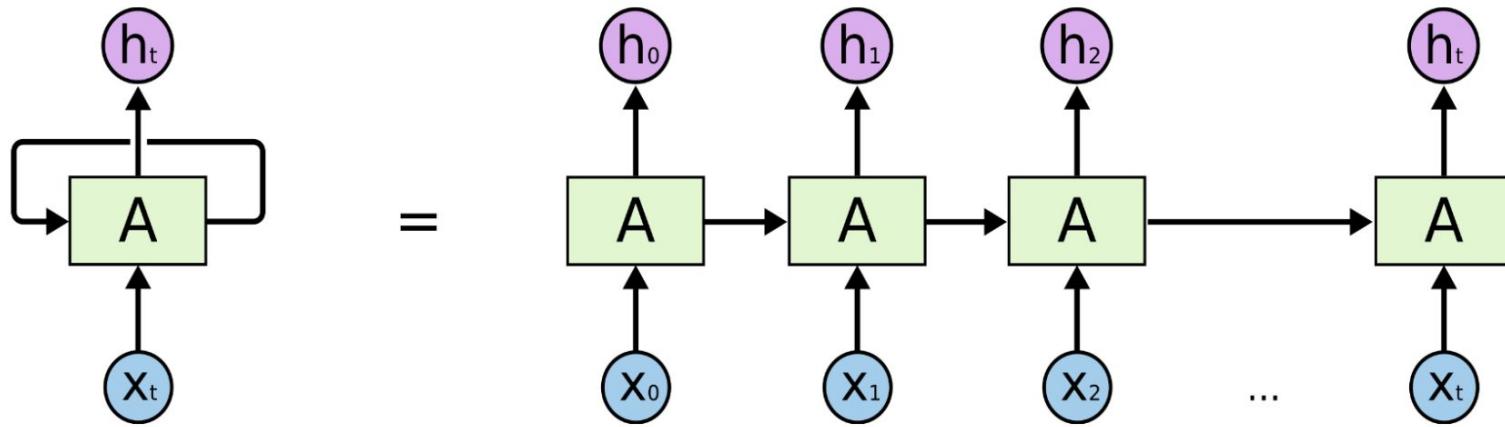
Benefits:

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

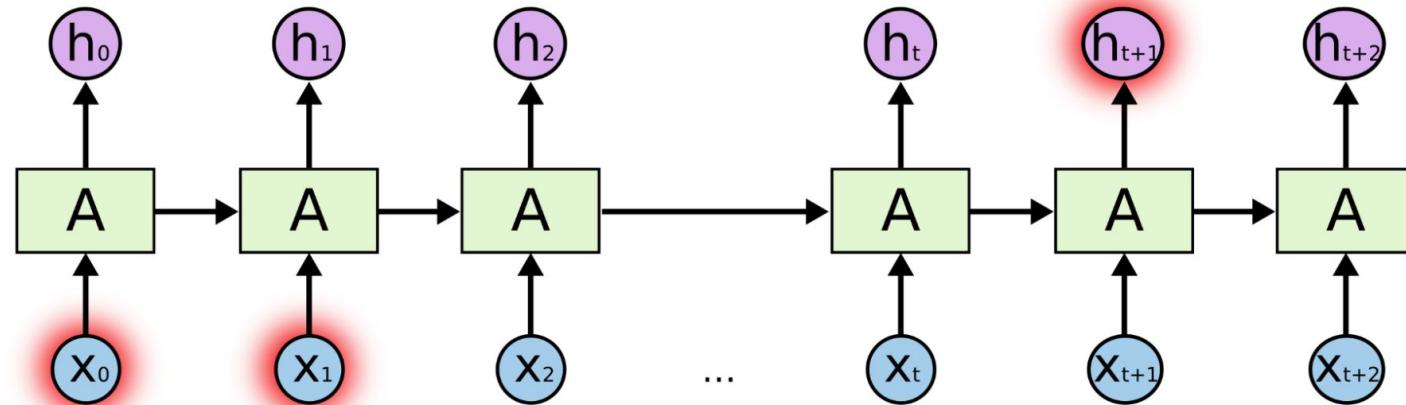
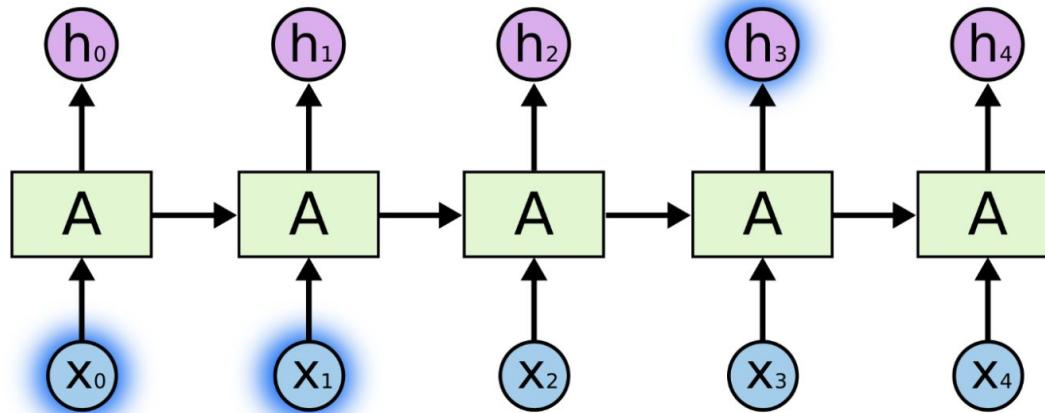
Problems:

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

# RNN



problems



What do we have:

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

What do we want

What to forget:

What to remember:

What to output:

What do we have:

$$s_{t-1} = (s_1, s_2, \dots, s_n) - \text{previous state}$$

$$h_{t-1} = (h_1, h_2, \dots, h_q) - \text{previous output}$$

$$x_{t-1} = (x_1, x_2, \dots, x_p) - \text{current input}$$

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

## 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:

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

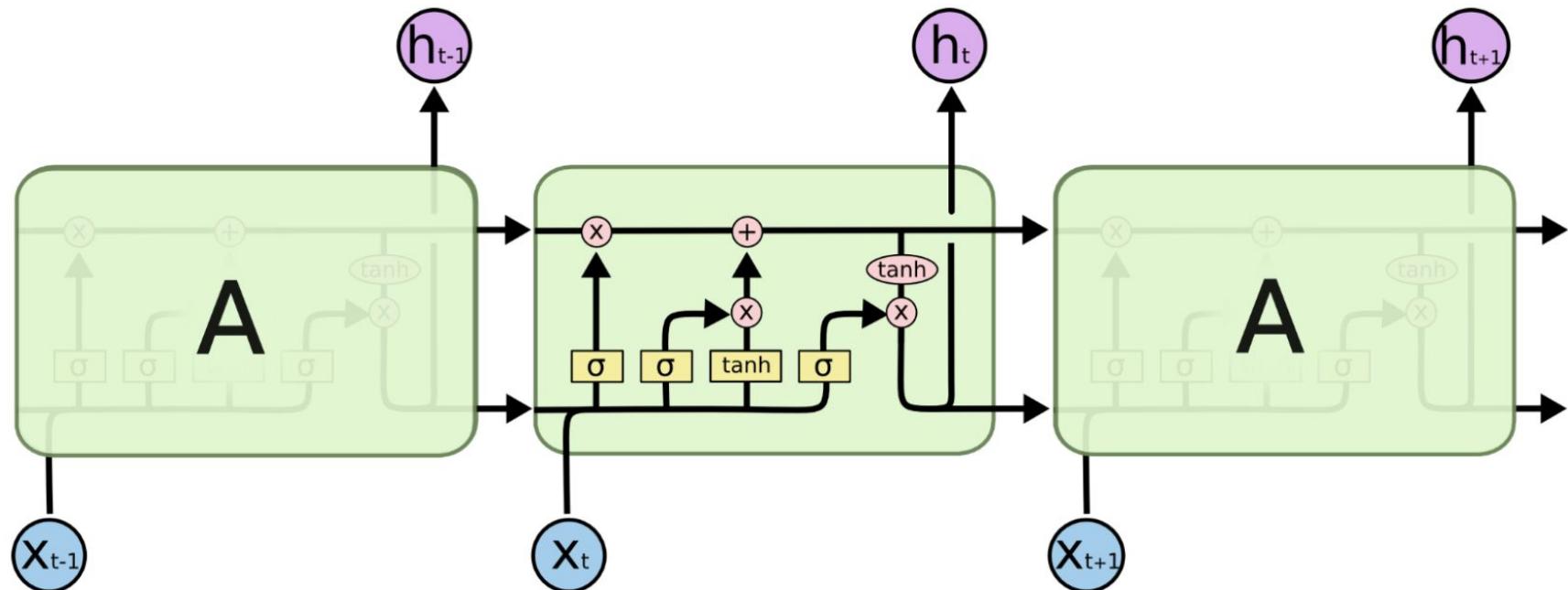
$$r \in [0, 1]^n, m \in [-1, 1]^n,$$

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

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

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

# LSTM



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

$$z = \sigma(W_z \cdot (x_t, h_{t-1})), \quad z \in [0, 1]^q$$

$$r = \sigma(W_r \cdot (x_t, h_{t-1})), \quad r \in [0, 1]^q$$

$$\tilde{h}_t = \tanh(W \cdot (x_t, r \cdot h_{t-1}))$$

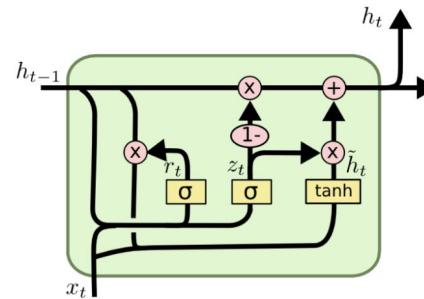
$$h_t = (1 - z) \cdot h_{t-1} + z * \tilde{h}_t$$

$$z = \sigma(W_z \cdot (x_t, h_{t-1})), \quad z \in [0, 1]^q$$

$$r = \sigma(W_r \cdot (x_t, h_{t-1})), \quad r \in [0, 1]^q$$

$$\tilde{h}_t = \tanh(W \cdot (x_t, r \cdot h_{t-1}))$$

$$h_t = (1 - z) \cdot h_{t-1} + z * \tilde{h}_t$$

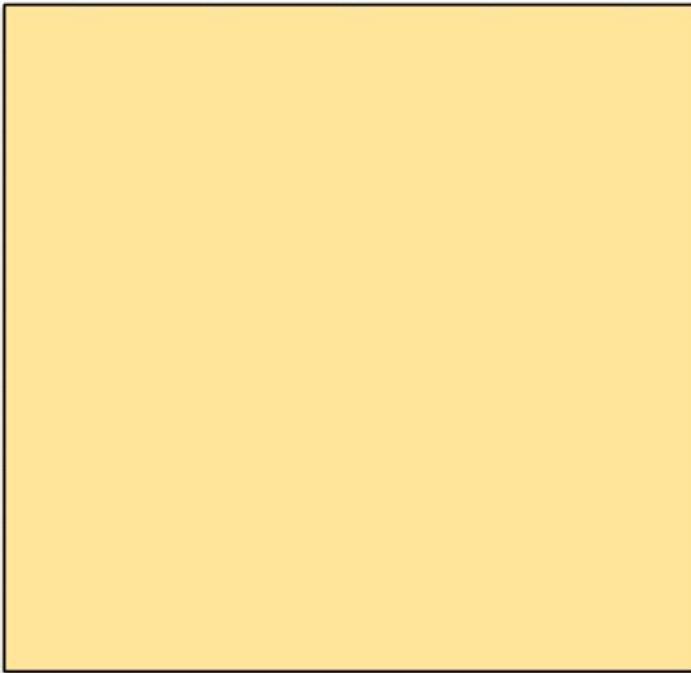


# RNN



$h_{t-1}$

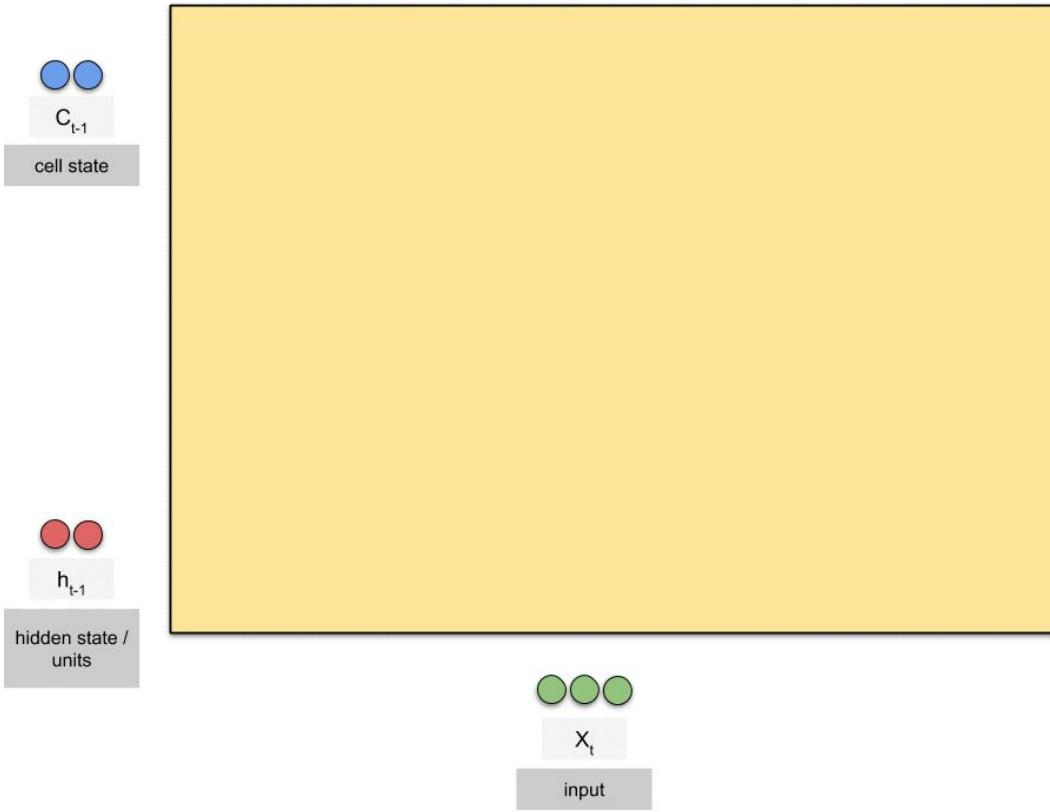
hidden units



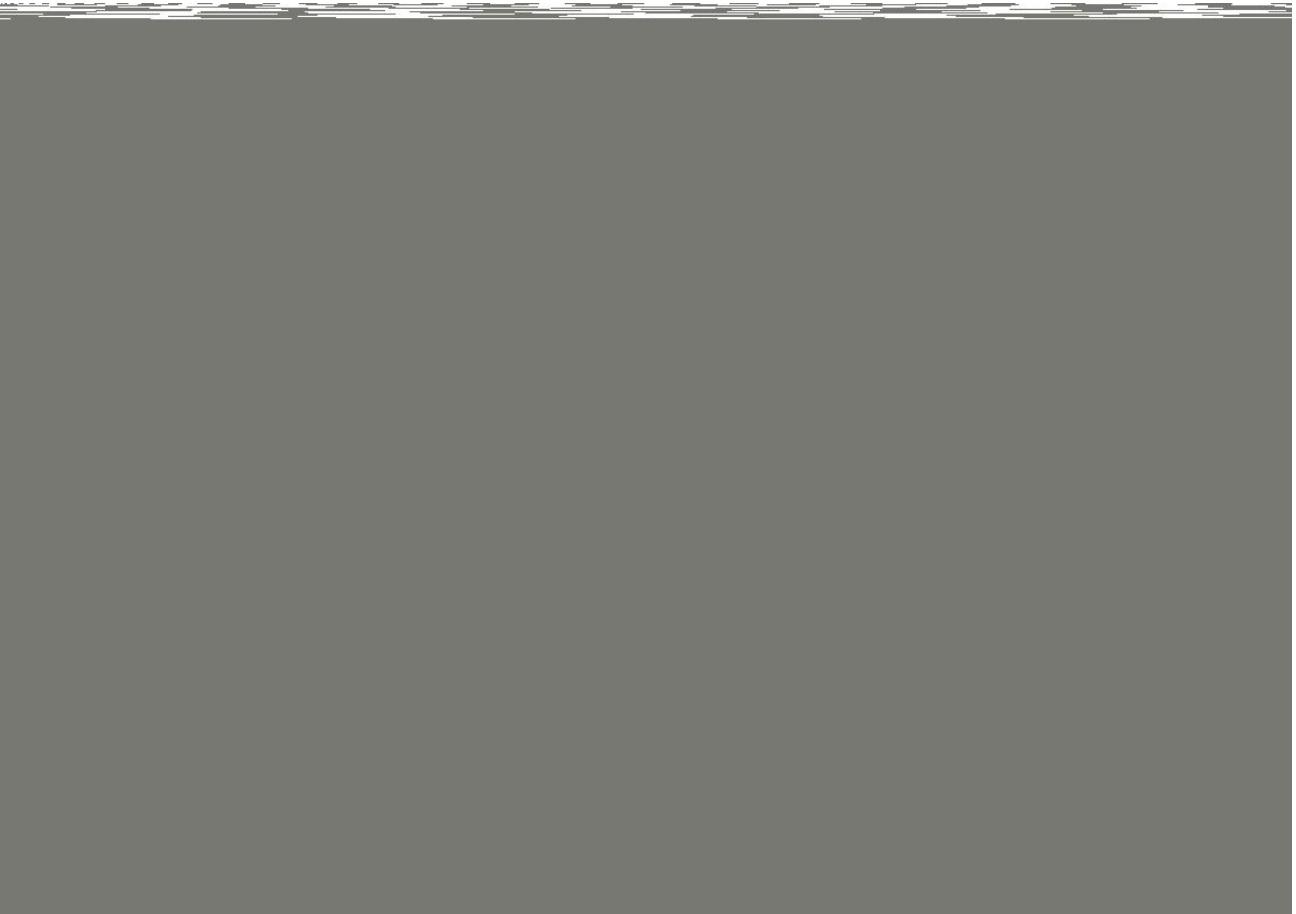
$x_t$

input

# LSTM



# GRU



# 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/>