Sequence handling
Text classification

NLP

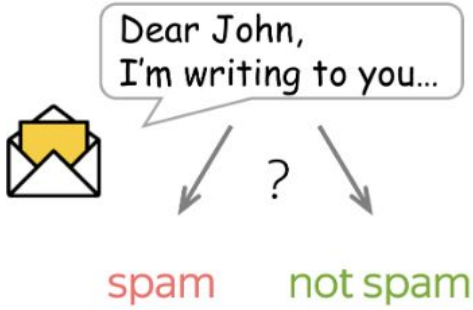
lecturer: Mollaev D. E.
Sber AI Lab

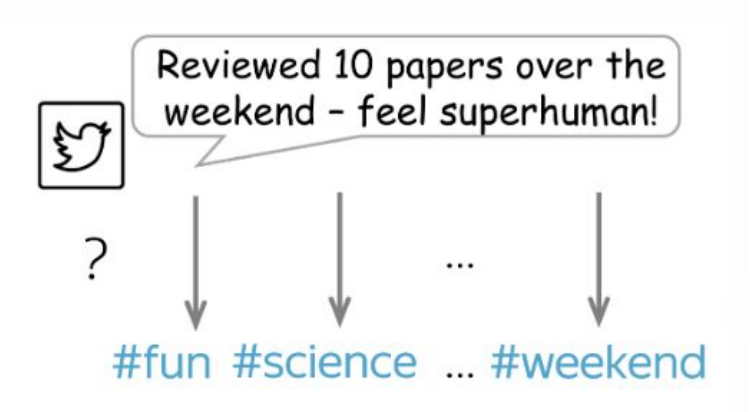
Lecture plan

- Examples of classification tasks
- General view
- Classical methods
- Neural networks

Text classification

- Binary-classification
- Multi-class classification
- Multi-label classification





Datasets for Text Classification

Dataset	Type	Number of labels	Size (train/test)	Avg. length (tokens)
SST	sentiment	5 or 2	8.5k / 1.1k	19
IMDb Review	sentiment	2	25k / 25k	271
Yelp Review	sentiment	5 or 2	650k / 50k	179
Amazon Review	sentiment	5 or 2	3m / 650k	79
TREC	question	6	5.5k / 0.5k	10
Yahoo! Answers	question	10	1.4m / 60k	131
AG's News	topic	4	120k / 7.6k	44
Sogou News	topic	6	54k / 6k	737
DBPedia	topic	14	560k / 70k	67

The specifics of datasets:

- Type
- Number of labels
- Size
- Average length

Lecture plan

- Examples of classification tasks
- General view
- Classical methods
- Neural networks

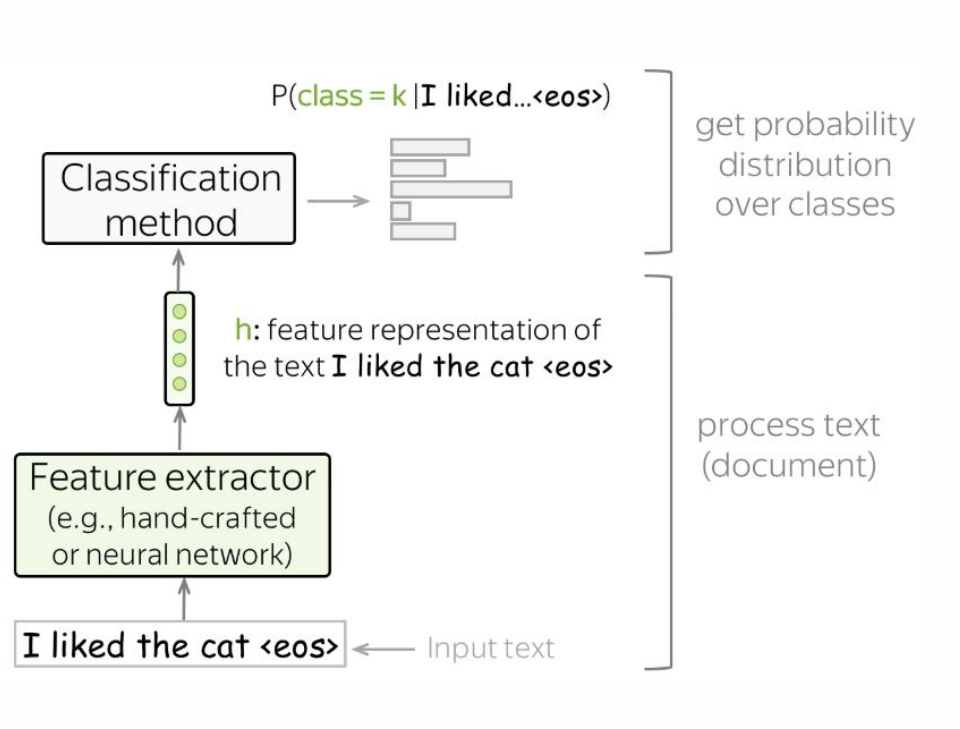
Feature extractor & Classification

- Feature extractor

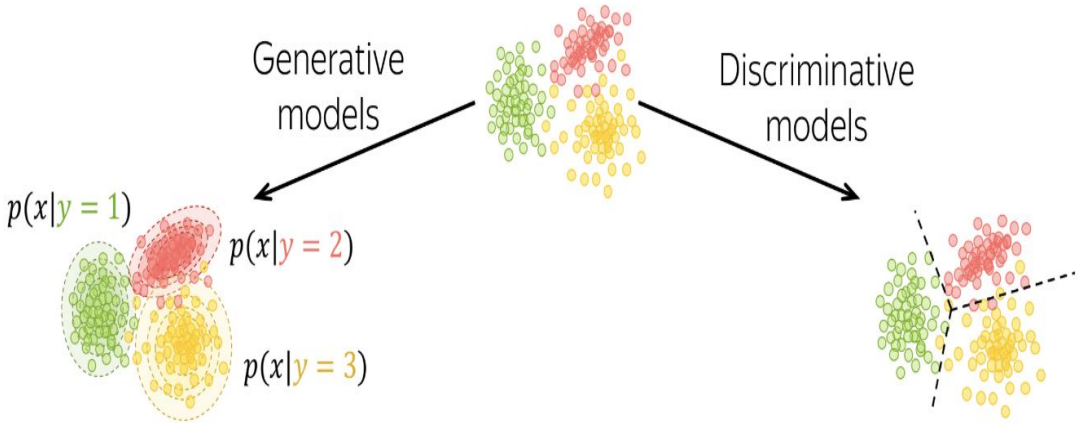
  Feature extractor can be either manually defined or learned

- Classification method

  Classifier has to assign class probabilities given feature representation of a text



Generative and Discriminative models



Learn: data distribution $p(x,y) = p(x|y) \cdot p(y)$

How predict: $y = \arg\max_k P(x, y = k) =$

$= \arg\max_k P(x|y = k) \cdot P(y = k)$

Learn: boundary between classes $p(y|x)$

How predict: $y = \arg\max_k P(y = k|x)$

- Generative models

Generative models learn joint probability distribution of data

- Discriminative models

Discriminative models are interested only in the conditional probability, i.e. they learn only the border between classes.

Lecture plan

- Examples of classification tasks
- General view
- Classical methods
  - Naive Bayes 
  - Logistic Regression
- Neural networks

Naive Bayes Classifier

Bayes' rule (hence Naïve **Bayes**)

Ignore P(x) – it does not influence the argmax

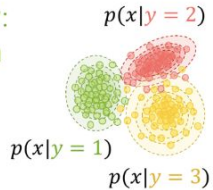
$$y^* = \arg\max_k P(y=k|x) = \arg\max_k \frac{P(x|y=k)\cdot P(y=k)}{P(x)} = \arg\max_k P(x|y=k)\cdot P(y=k)$$

need to define this

This is a generative model!

$$y^* = \arg\max_k P(y=k|x) = \arg\max_k P(x|y=k)\cdot P(y=k) = \arg\max_k P(x, y=k)$$

posterior probability: after looking at data (i.e., we know **x**)



prior probability: before looking at data (i.e., we don't know **x**)

joint probability (hence the model is **generative**)

How to define P(x|y=k) and P(y=k)?

P(y=k): count labels we can just evaluate the proportion of documents with the label k

(this is the maximum likelihood estimate, MLE)

$$P(y=k) = \frac{N(y=k)}{\sum_i N(y=i)}$$

How to define P(x|y=k) and P(y=k)?

P(x|y=k): use the "naive" assumptions

Here we assume that document $\boldsymbol{x}$ is represented as a set of features, e.g., a set of its words $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$:

$$P(\boldsymbol{x}|\boldsymbol{y} = \boldsymbol{k}) = P(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n|\boldsymbol{y} = \boldsymbol{k}).$$

The Naive Bayes assumptions are

- **Bag of Words** assumption: word order does not matter,
- **Conditional Independence** assumption: features (words) are independent given the class.

How to define P(x|y=k) and P(y=k)?

P(x|y=k): use the "naive" assumptions

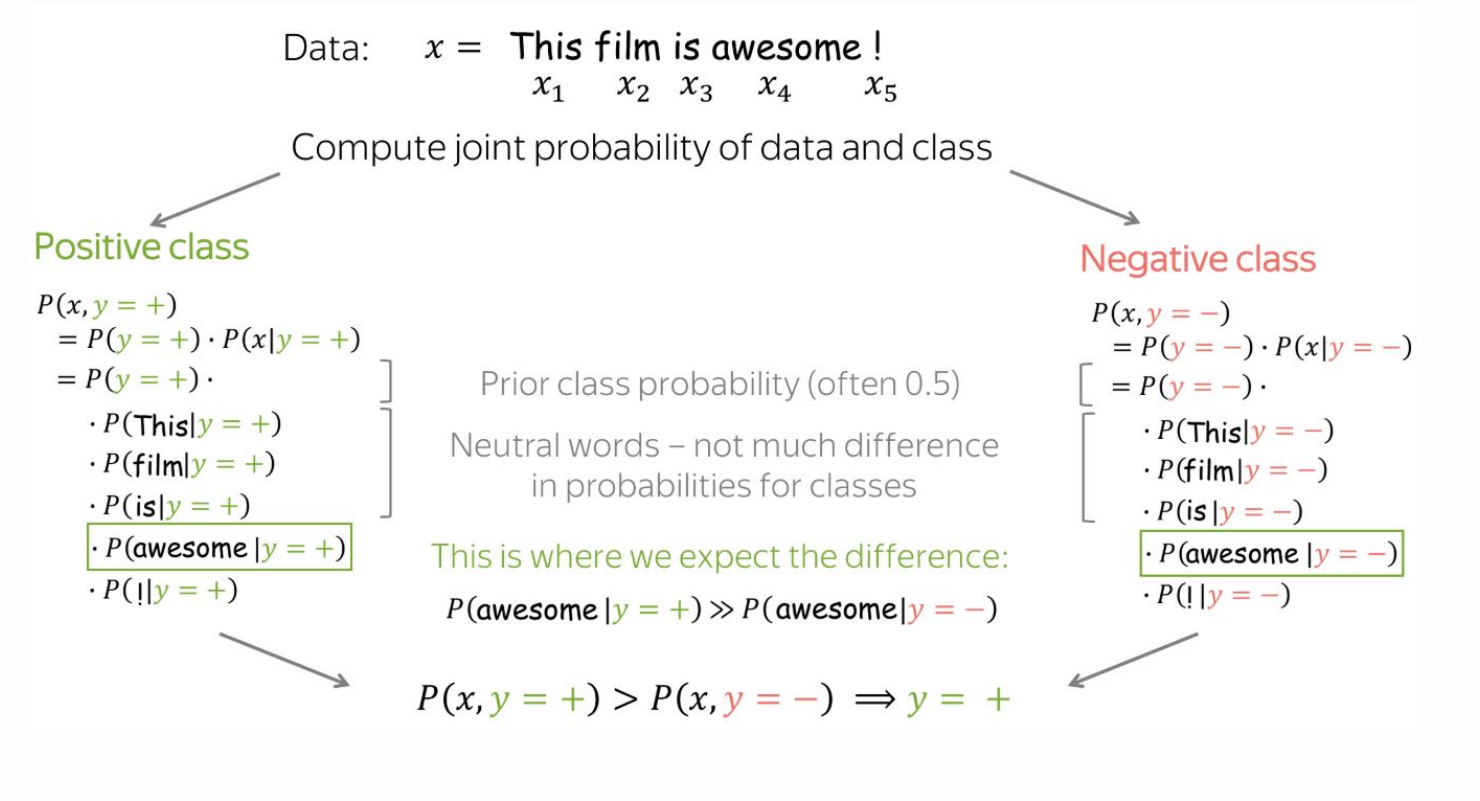
With these **"naive"** assumptions we get:

$$P(x|y=k) = P(x_1, \ldots, x_n|y=k) = \prod_{t=1}^{n} P(x_t|y=k).$$

The probabilities $P(x_i|y=k)$ are estimated as the proportion of times the word x_i appeared in documents of class k among all tokens in these documents:

$$P(x_i|y=k) = \frac{N(x_i, y=k)}{\sum_{t=1}^{|V|} N(x_t, y=k)},$$

Making a Prediction



What if $N(x_i, y = k) = 0$? Need to avoid this!

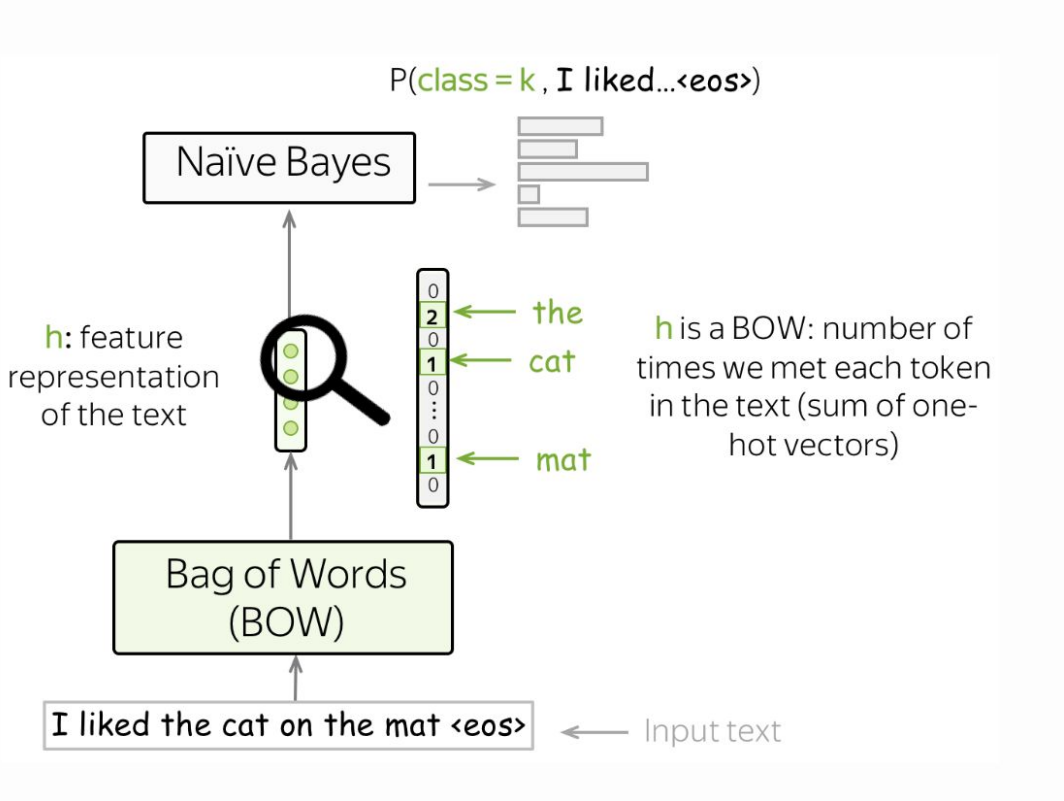
nulls out token prob. $\Longrightarrow$ nulls out document probability $\Longrightarrow$ Bad!

$$N(x_i, y = k) \Longrightarrow P(x_i|y = k) = \frac{N(x_i, y = k)}{\sum_{t=1}^{|V|} N(x_t, y = k)} = 0 \Longrightarrow P(x|y = k) = \prod_{i=1}^{n} P(x_i|y = k) = 0$$

In training data, haven't seen token x_i in documents of class k

$$P(x_i|y = k) = \frac{\delta + N(x_i, y = k)}{\sum_{t=1}^{|V|}(\delta + N(x_t, y = k))} = \frac{\delta + N(x_i, y = k)}{\delta \cdot |V| + \sum_{t=1}^{|V|} N(x_t, y = k)}$$

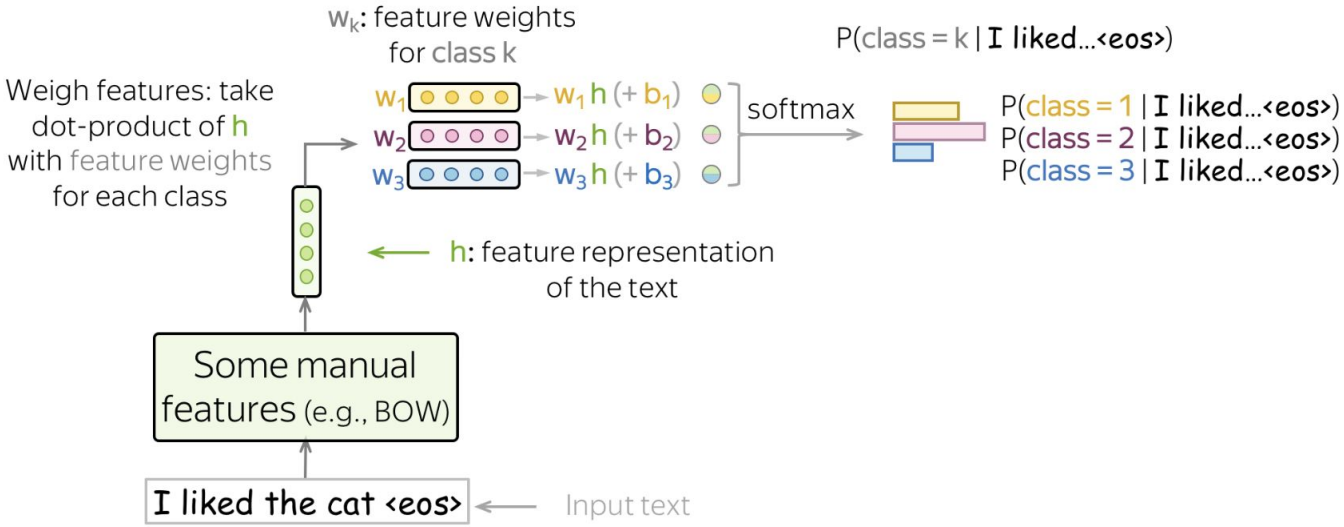
View in General Framework



Lecture plan

- Examples of classification tasks
- General view
- Classical methods
  - Naive Bayes
  - Logistic Regression 
- Neural networks

Maximum Entropy Classifier (Logistic Regression)



Maximum Entropy Classifier (Logistic Regression): Pipeline

- get $h = (f_1, f_2, \dots, f_n)$ - feature representation of the input text;
- take $w^{(i)} = (w_1^{(i)}, \dots, w_n^{(i)})$ - vectors with feature weights for each of the classes;
- for each class, weigh features, i.e. take the dot product of feature representation h with feature weights $w^{(k)}$:

$$w^{(k)}h = w_1^{(k)} \cdot f_1 + \dots + w_n^{(k)} \cdot f_n, \quad k = 1, \dots, K.$$

To get a bias term in the sum above, we define one of the features being 1 (e.g., $f_0 = 1$). Then

$$w^{(k)}h = w_0^{(k)} + w_1^{(k)} \cdot f_1 + \dots + w_n^{(k)} \cdot f_n, \quad k = 1, \dots, K.$$

- get class probabilities using softmax:

$$P(class = k|h) = \frac{\exp(w^{(k)}h)}{\sum_{i=1}^{K} \exp(w^{(i)}h)}.$$

Softmax normalizes the K values we got at the previous step to a probability distribution over output classes.

Training: Maximum Likelihood Estimate

Given training examples $x^1, \ldots, x^N$ with labels $y^1, \ldots, y^N$, $y^i \in \{1, \ldots, K\}$, we pick those weights $w^{(k)}, k = 1..K$ which maximize the probability of the training data:

$$w^* = \arg\max_w \sum_{i=1}^{N} \log P(y = y^i | x^i).$$

Equvalence to minimizing cross-entropy

Note that maximizing data log-likelihood is equivalent to minimizing cross entropy between the target probability distribution $p^* = (0, \ldots, 0, 1, 0, \ldots)$ (1 for the target label, 0 for the rest) and the predicted by the model distribution $p = (p_1, \ldots, p_K), p_i = p(i|x)$:

$$Loss(p^*, p) = -p^* \log(p) = -\sum_{i=1}^{K} p_i^* \log(p_i).$$

Since only one of p_i^* is non-zero (1 for the target label k, 0 for the rest), we will get $Loss(p^*, p) = -\log(p_k) = -\log(p(k|x))$.

Training: Maximum Likelihood Estimate

Training example: **I liked the cat on the mat <eos>**

Label: k

target

$$\log P(y = k|x) \rightarrow max$$

Maximizing log-likelihood of the correct class

$$\Updownarrow$$

$$-\log P(y = k|x) \rightarrow min$$

Minimizing negative log-likelihood of the correct class

$$\Updownarrow$$

$$-\sum_{i=1}^{K} p_i^* \cdot \log P(y = i|x) \rightarrow min$$

Minimizing cross-entropy loss

$$(p_k^* = 1, p_i^* = 0, i \neq k)$$

Model prediction:

P(class = i |**I liked…<eos>**)

Target:

p^*

Naive Bayes vs Logistic Regression

Naïve Bayes:

- very simple
- very fast
- interpretable
- assumes that features are conditionally independent
- text representation: manually defined (and often too simple, e.g. BOW)

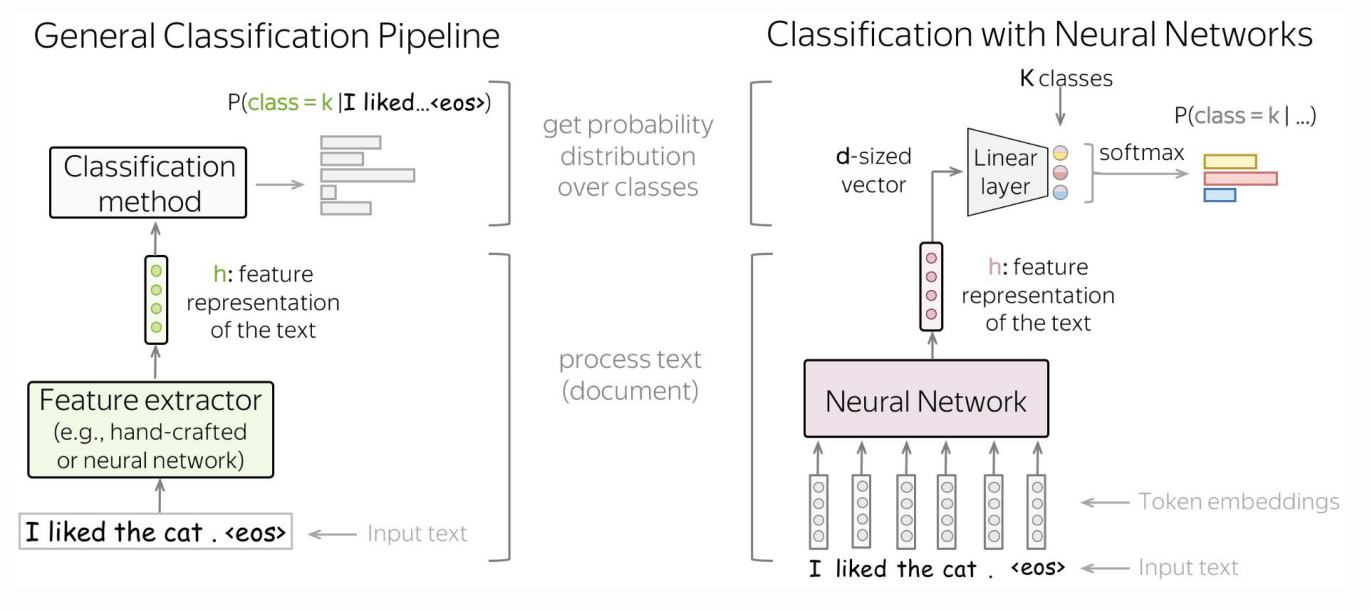
Logistic Regression:

- quite simple
- interpretable
- does **not** assume that features are conditionally independent
- not so fast (multiple iterations of gradient ascent)
- text representation: manually defined

Lecture plan

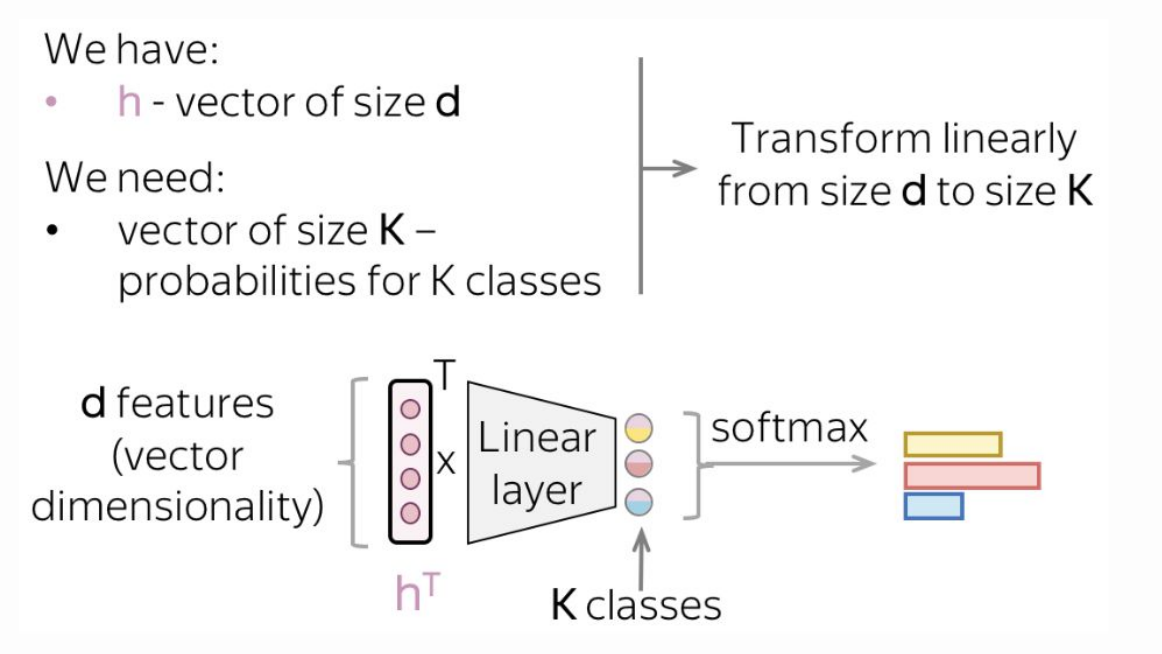
- Examples of classification tasks
- General view
- Classical methods
  - Naive Bayes
  - Logistic Regression
- Neural networks
  - General
  - Basic
  - RNN
  - CNN

Neural networks

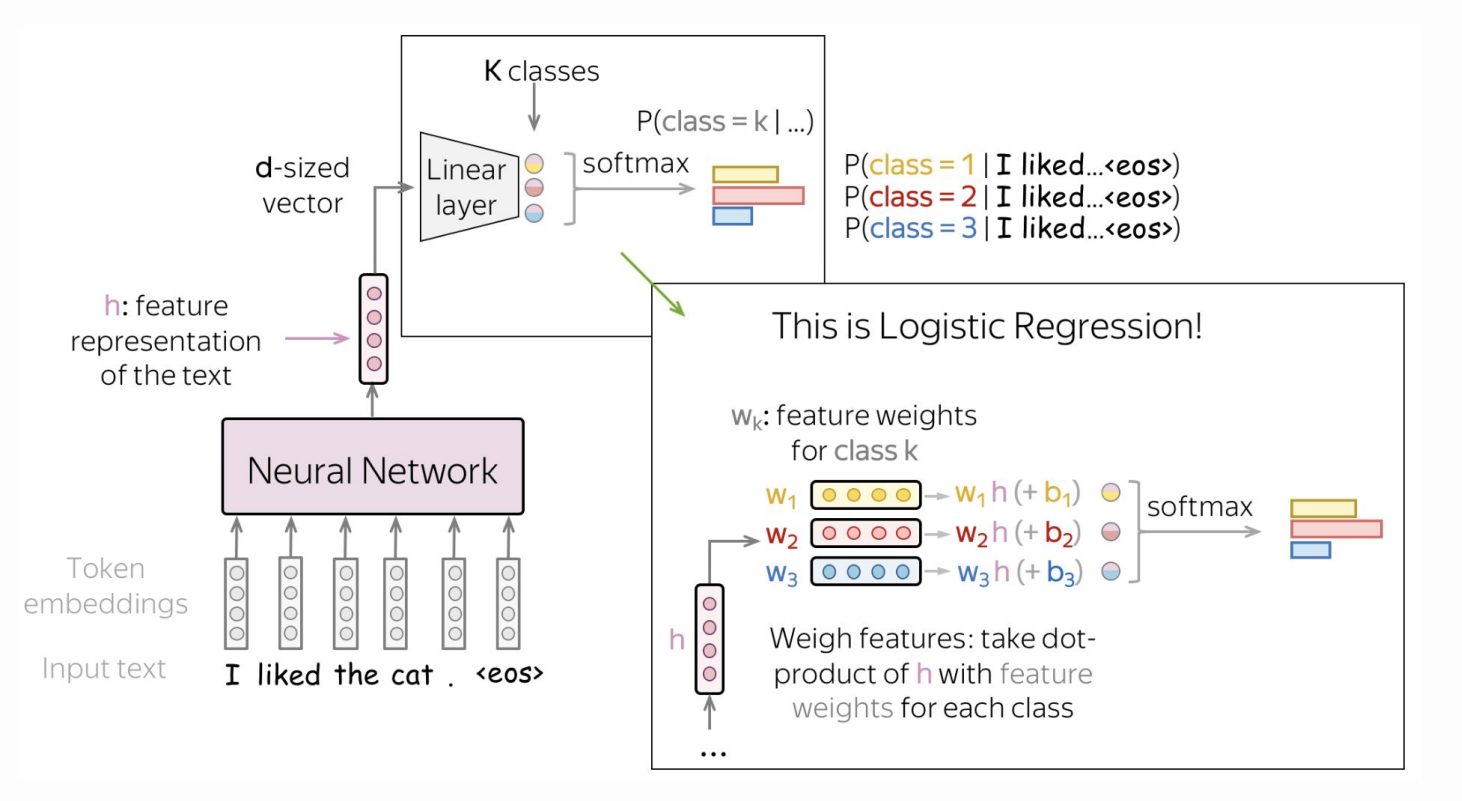


Instead of manually defined features, let a neural network to learn useful features.

Neural Networks: Head

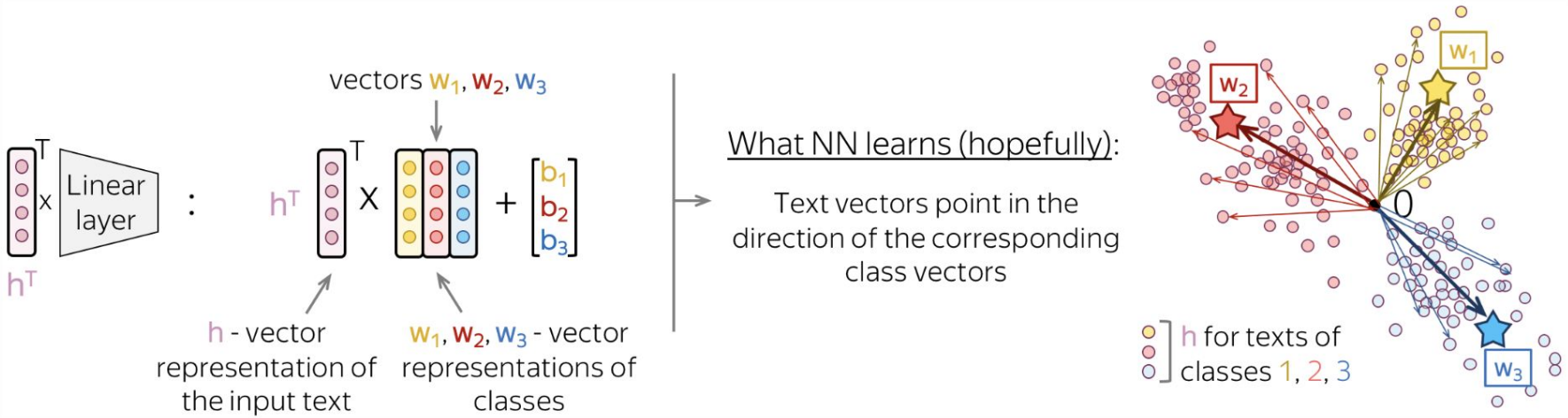


Classification Part: This is Logistic Regression!



Classification Part: This is Logistic Regression!

Intuition: Text Representation Points in the Direction of Class Representation



Training and the Cross-Entropy Loss

Training example: I liked the cat on the mat <eos>

Label: k

target

Model prediction:

P(class = i | I liked...<eos>)

Target:

p^*

Cross-entropy loss:

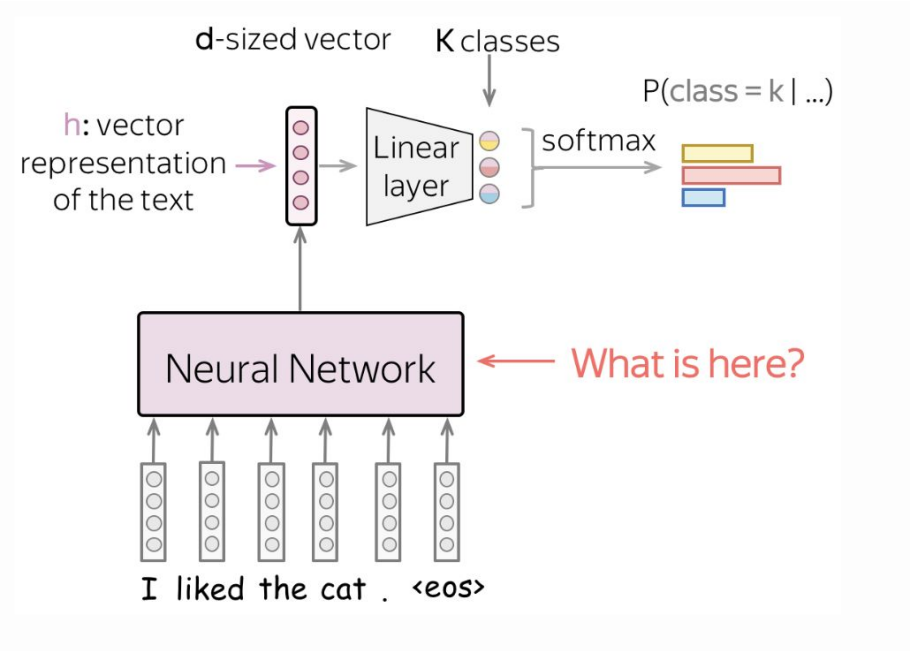
$$-\sum_{i=1}^{K} p_i^* \cdot \log P(y=i|x) \rightarrow min \quad (p_k^* = 1, p_i^* = 0, i \neq k)$$

For one-hot targets, this is equivalent to

$$-\log P(y=k|x) \rightarrow min$$

$$Loss(p^*, p) = -p^* \log(p) = -\sum_{i=1}^{K} p_i^* \log(p_i).$$

Models for Text Classification

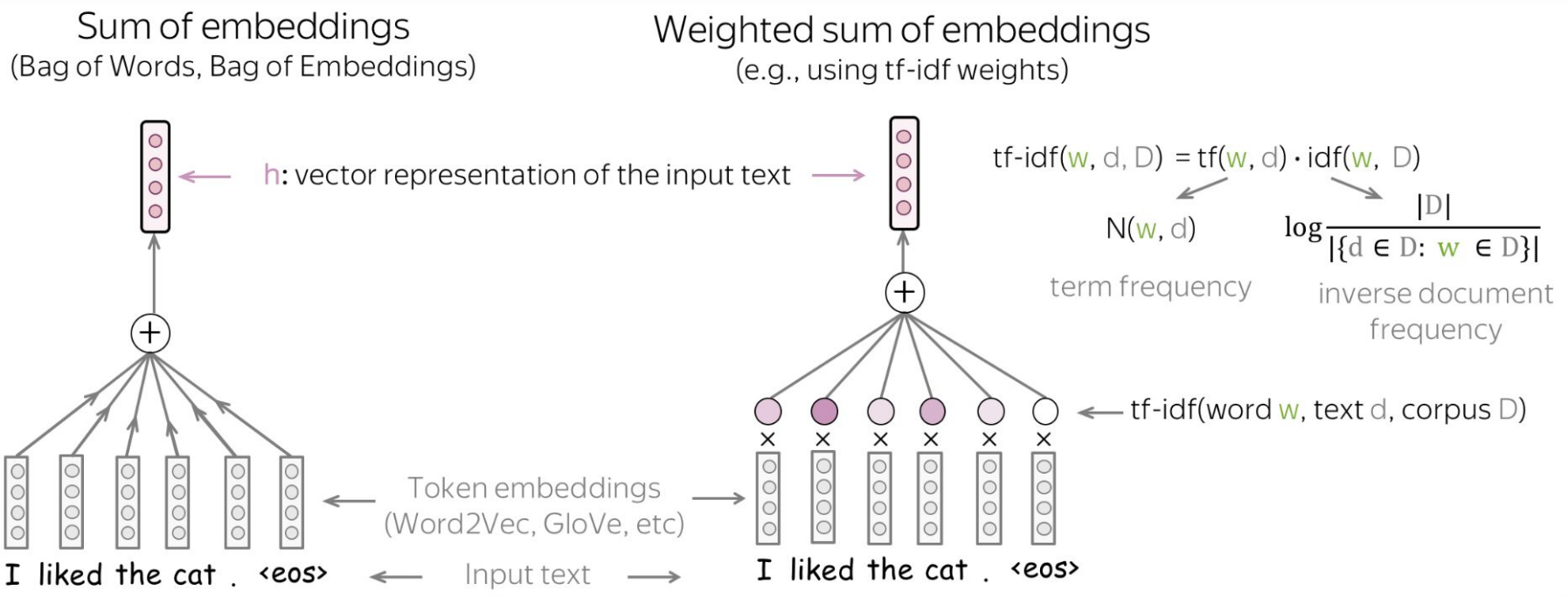


We need a model that can produce a fixed-sized vector for inputs of different lengths.

Lecture plan

- Examples of classification tasks
- General view
- Classical methods
  - Naive Bayes
  - Logistic Regression
- Neural networks
  - General
  - Basic
  - RNN
  - CNN

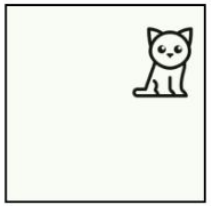
Basics: Bag of Embeddings (BOE), Weighted BOE, Word2Vec

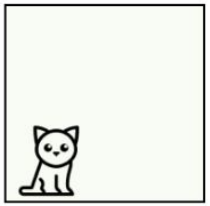


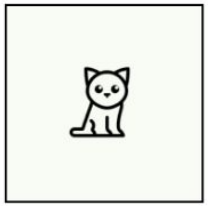
Lecture plan

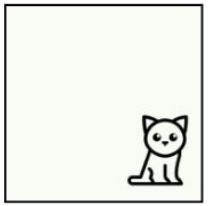
- Examples of classification tasks
- General view
- Classical methods
  - Naive Bayes
  - Logistic Regression
- Neural networks
  - General
  - Basic
  - CNN 
  - RNN

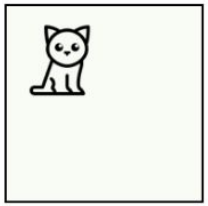
CNN: for image


Label: cat


Label: cat

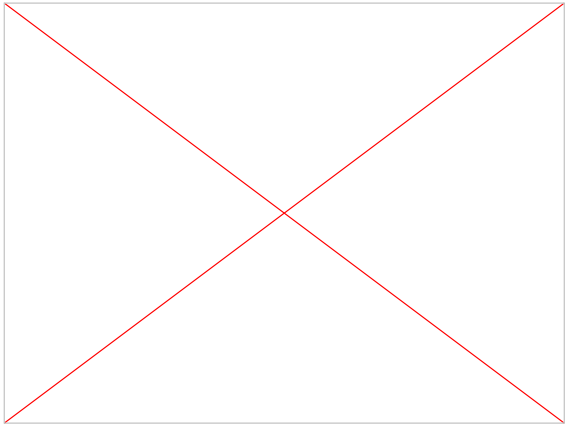

Label: cat

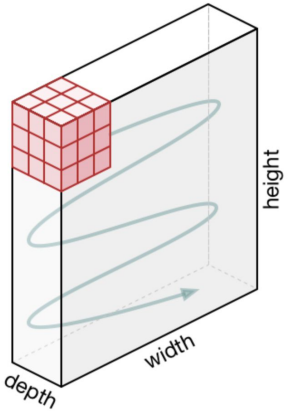

Label: cat


Label: cat

We don't care where the cat is, we care that it is somewhere.

Then why don't we process all these cats similarly?





- apply the same operation to small parts of an input
- find "matches" with patterns

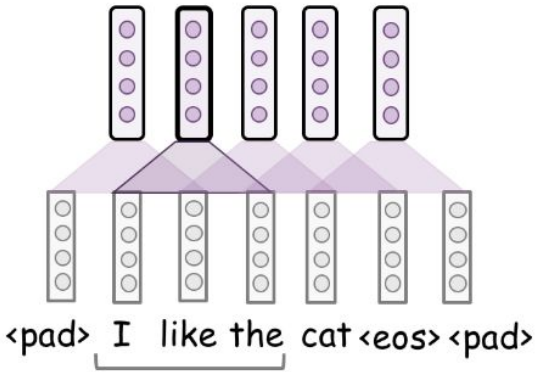
CNN: for text

An absolutely great movie! I watched the premiere with my friends.

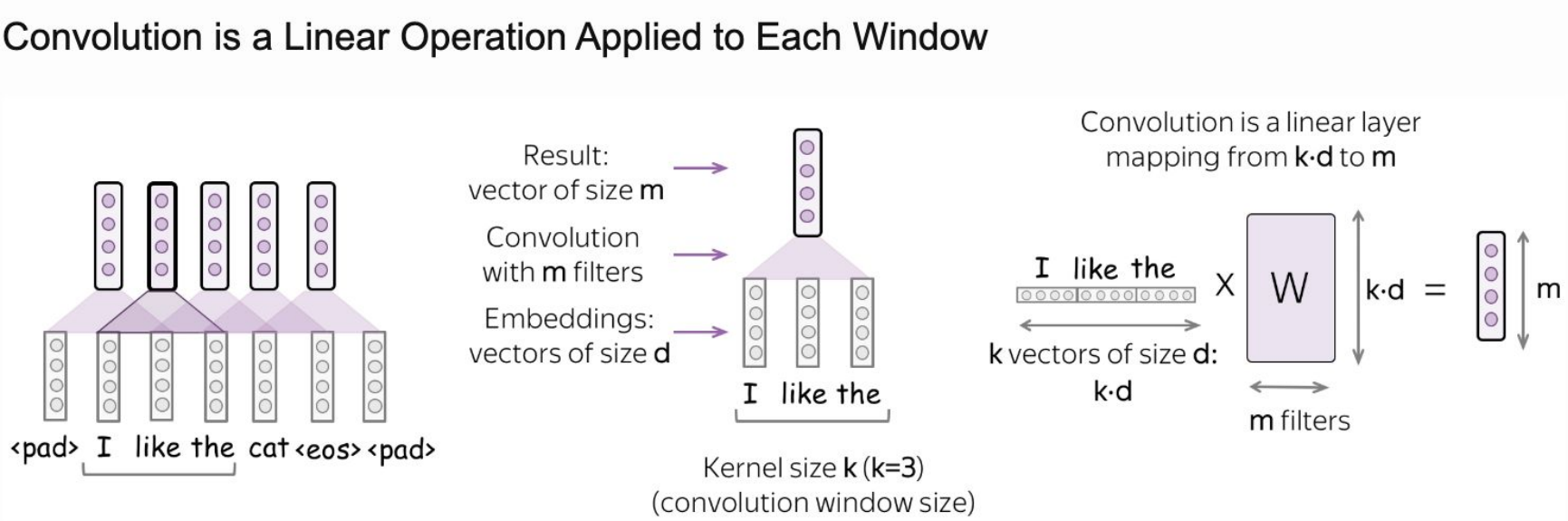
The movie about cats was absolutely great, and the cats were cute.

The movie is about cats running around, and it is absolutely great.

If a clue is very informative, maybe we don't care much where in a text it appears?



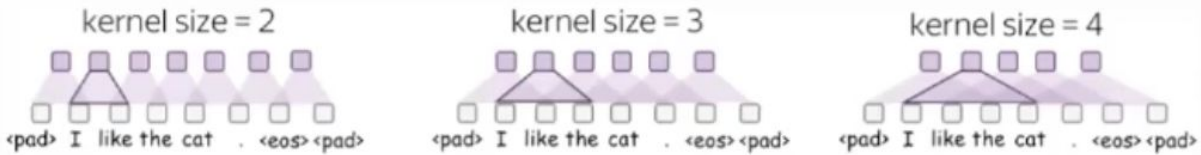
CNN


Convolution is a Linear Operation Applied to Each Window
<pad> I like the cat <eos> <pad>
Result:
vector of size m
Convolution
with m filters
Embeddings:
vectors of size d
I like the
Kernel size k (k=3)
(convolution window size)
Convolution is a linear layer
mapping from k·d to m
I like the
k vectors of size d:
k·d
X
W
k·d
=
m
m filters

Convolution: Parameters

- **Kernel size**: How far to look

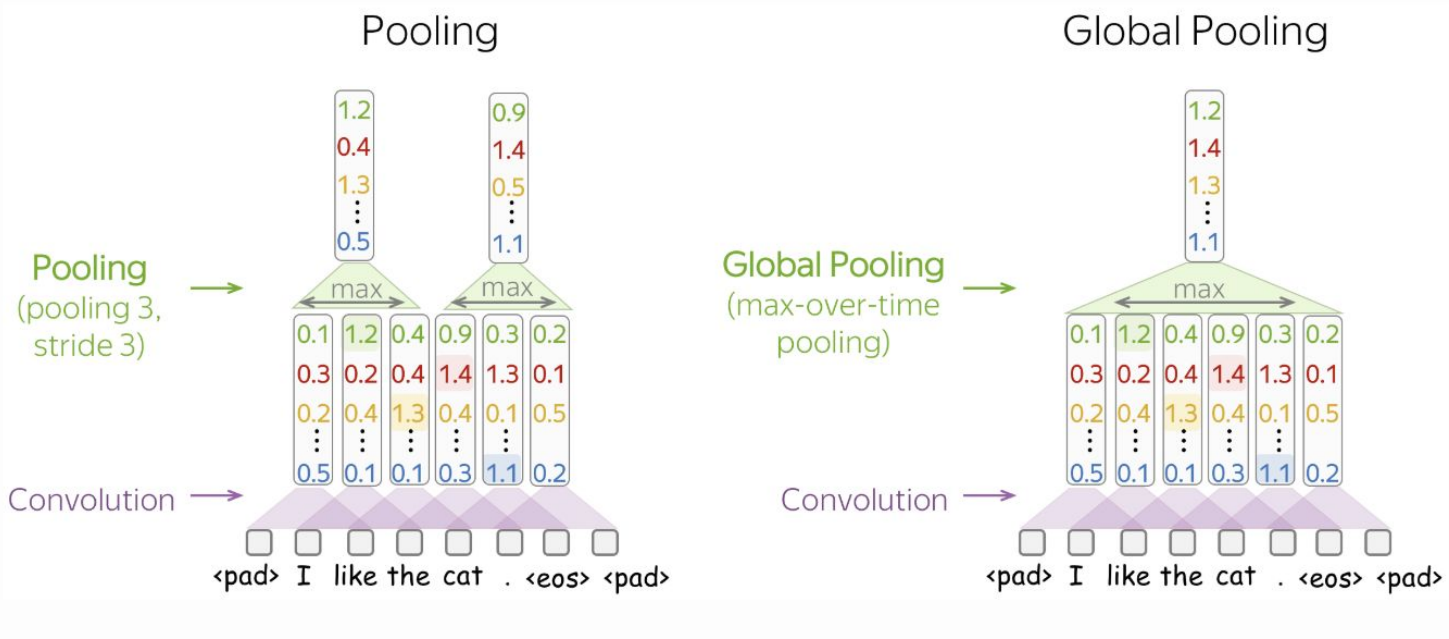
Kernel size is the number of input elements (tokens) a convolution looks at each step. For text, typical values are 2-5.



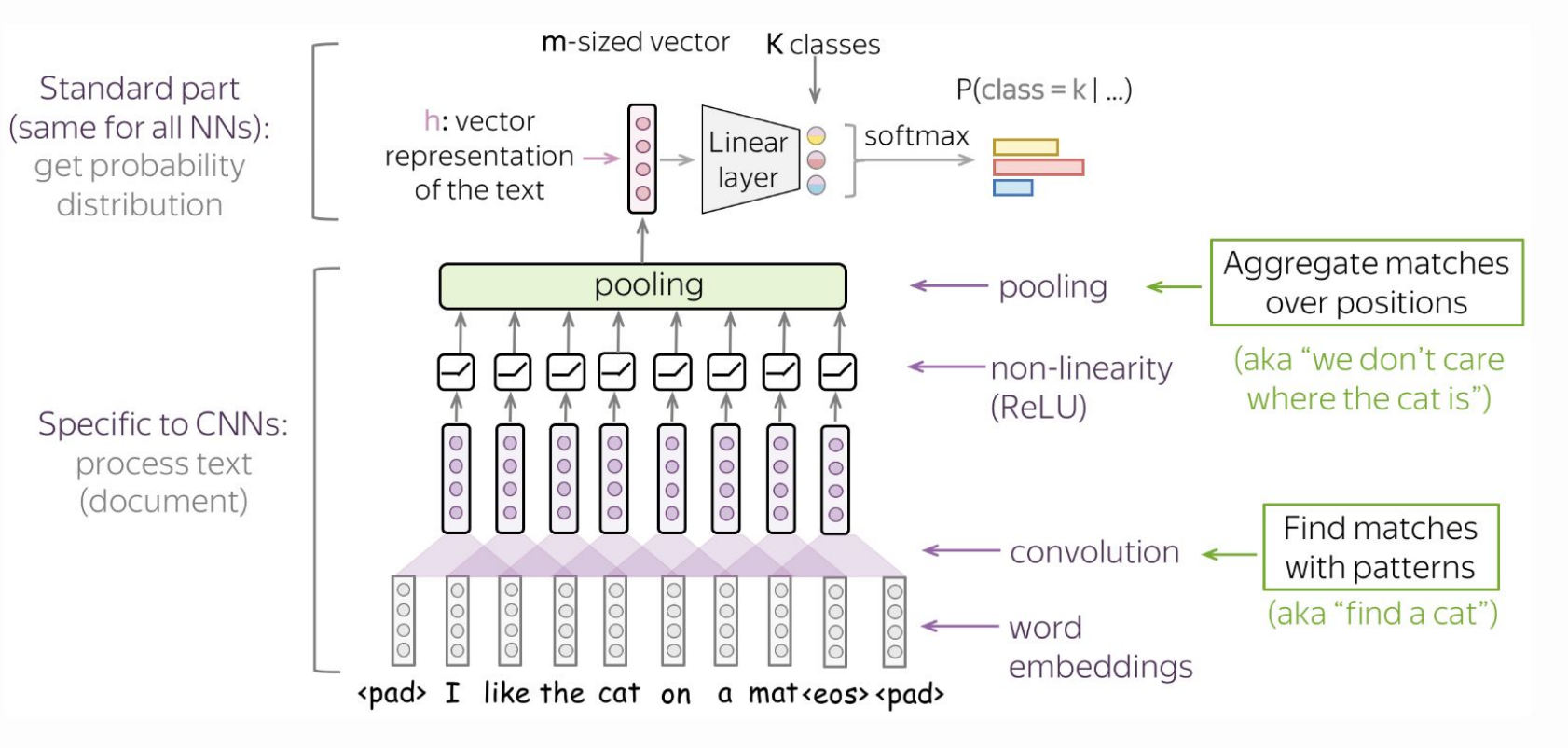
- **Stride**: How much move a filter at each step

Stride tells how much to move filter at each step. For example, stride equal to 1 means that we move the filter by 1 input element (pixel for images, token for texts) at each step.

CNN: Pooling


Pooling
Pooling
(pooling 3,
stride 3)
max
max
Convolution
<pad> I like the cat . <eos> <pad>
Global Pooling
Global Pooling
(max-over-time
pooling)
max
Convolution
<pad> I like the cat . <eos> <pad>

CNN: General framework

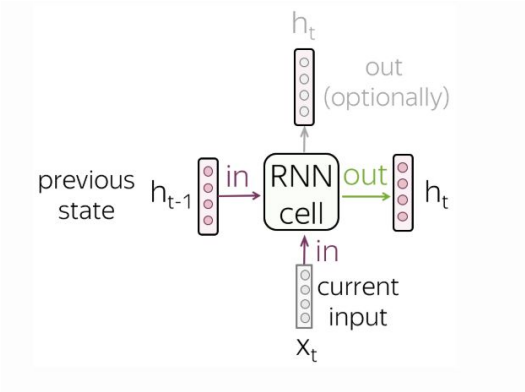


Lecture plan

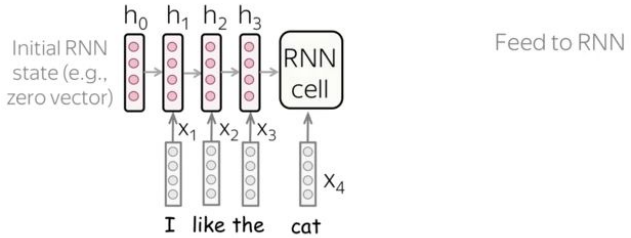
- Examples of classification tasks
- General view
- Classical methods
  - Naive Bayes
  - Logistic Regression
- Neural networks
  - General
  - Basic
  - CNN
  - RNN ⇦

Models: Recurrent Neural Networks (RNN)

- RNN cell



- RNN: sequence of tokens

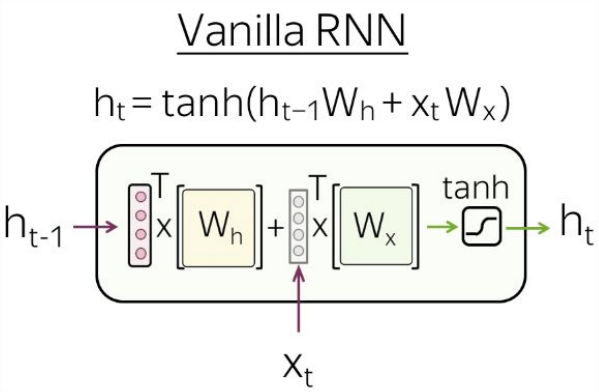


Models: Recurrent Neural Networks (RNN): Architectures

- **Vanilla RNN**

The simplest recurrent network, **Vanilla RNN**, transforms h_{t-1} and x_t linearly, then applies a non-linearity (most often, the $\tanh$ function):

$$h_t = \tanh(h_{t-1}W_h + x_t W_t).$$

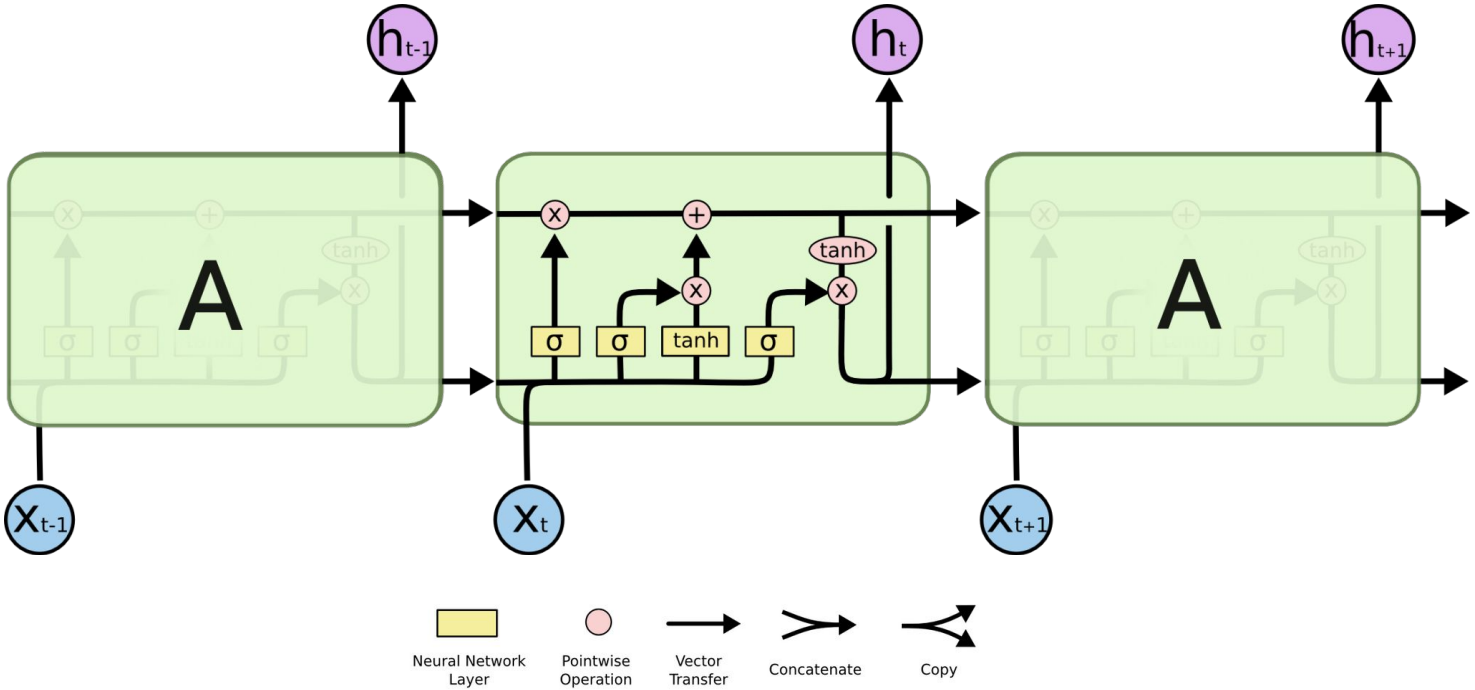


RNN: Architectures - LSTM

Gate:

- forget gate
- input gate
- output gate

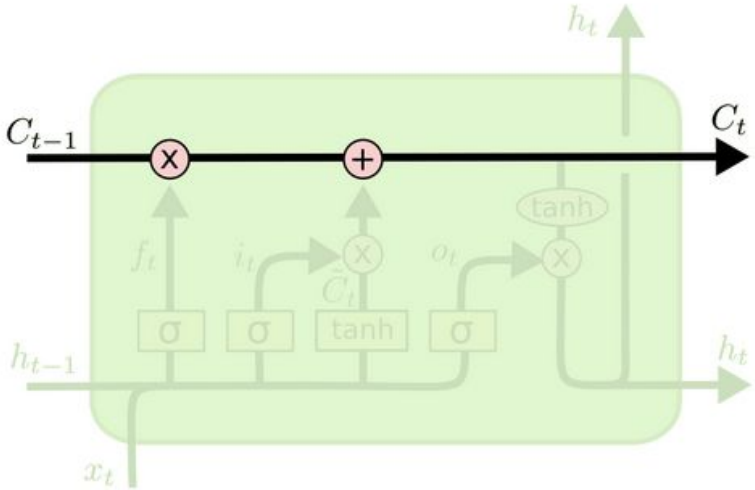
- LSTM



RNN: Architectures - LSTM

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

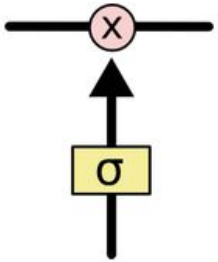
The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



RNN: Architectures - LSTM

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

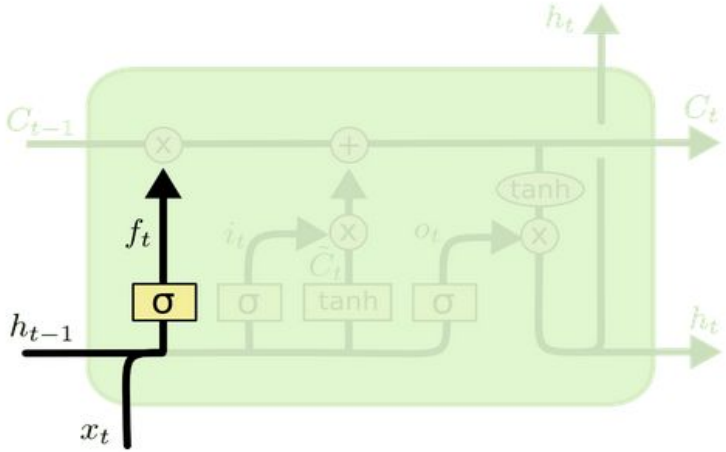
Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

RNN: Architectures - LSTM (Forget gate)

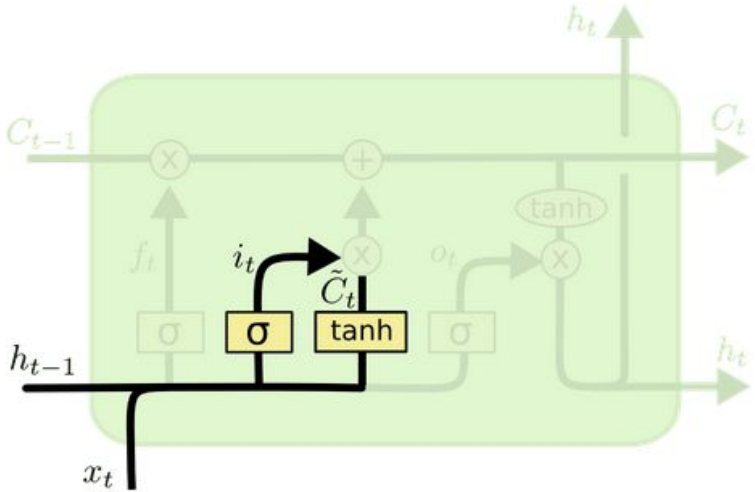
Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



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

RNN: Architectures - LSTM (Input gate)

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



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

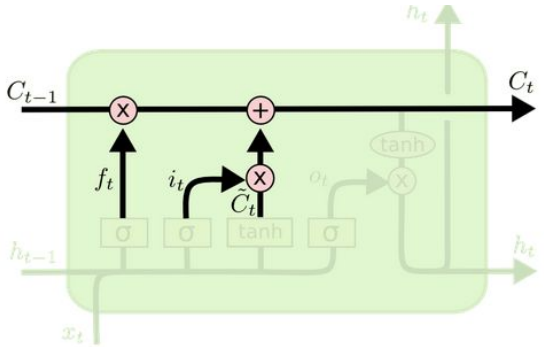
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

RNN: Architectures - LSTM

It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

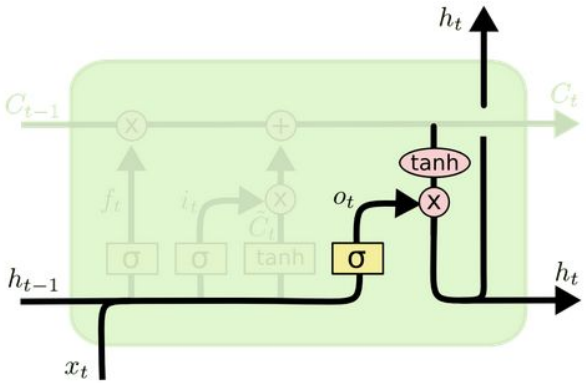
In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



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

RNN: Architectures - LSTM (Output gate)

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through **tanh** (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

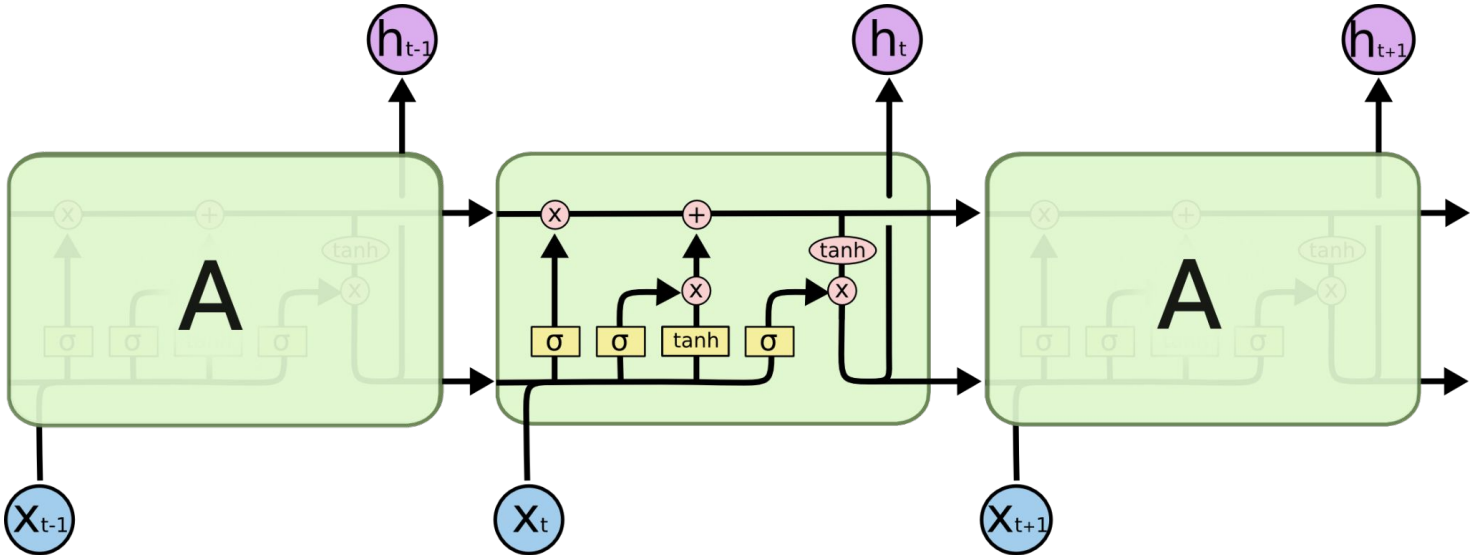


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

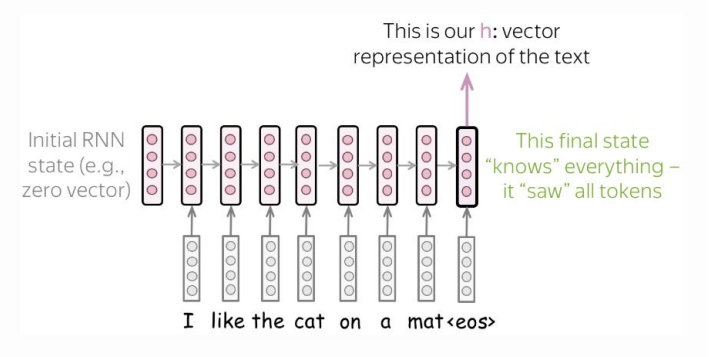
For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

LSTM: summary

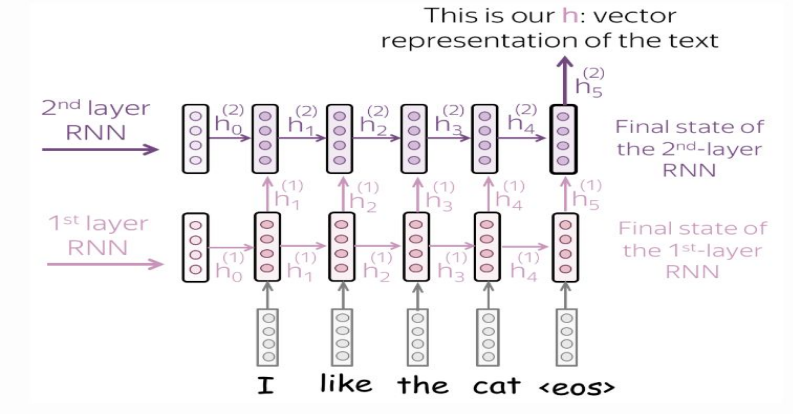


RNN: Settings for text classification

- **Simple**: read a text, take the final state

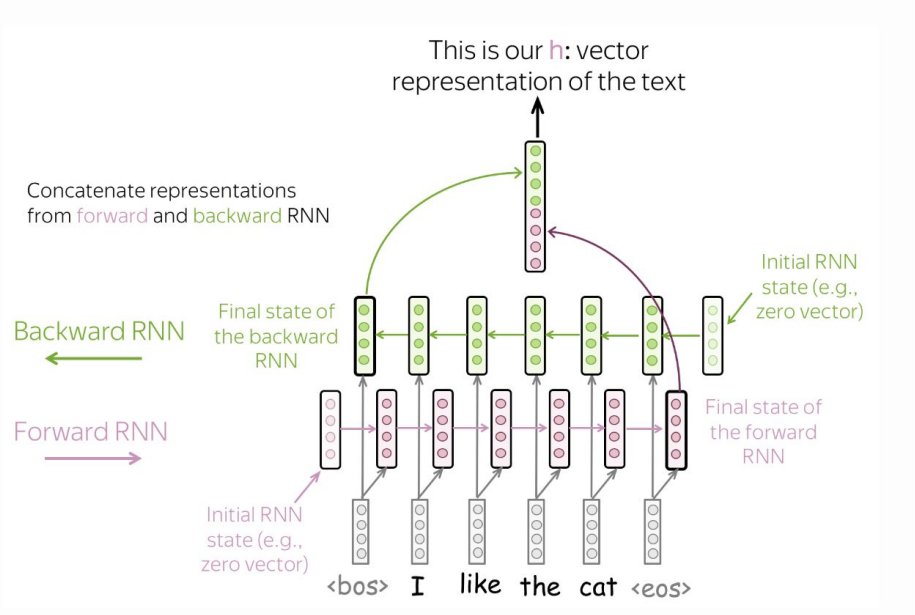


- **Multiple layers**: feed the states from one RNN to the next one



RNN: Settings for text classification

- **Bidirectional**: use final states from forward and backward RNNs.



Lecture plan

- Examples of classification tasks
- General view
- Classical methods
  - Naive Bayes
  - Logistic Regression
- Neural networks
  - General
  - Basic
  - CNN
  - RNN