Language models

Alena Fenogenova
Lecture 4. 27/09/2021

Today

- **Language models**: The main idea and applications
- **Probabilistic language models:** count-based language models, Markov language models, smoothing, hmm, crf
- **Neural-based language models:** Recurrent Neural Networks, LSTM and GRU networks
- **Text generation:** Generating texts with language models
- **Evaluation of language models:** Intrinsic and extrinsic methods
- **Sequence labelling task:** Named Entity Recognition

Language model

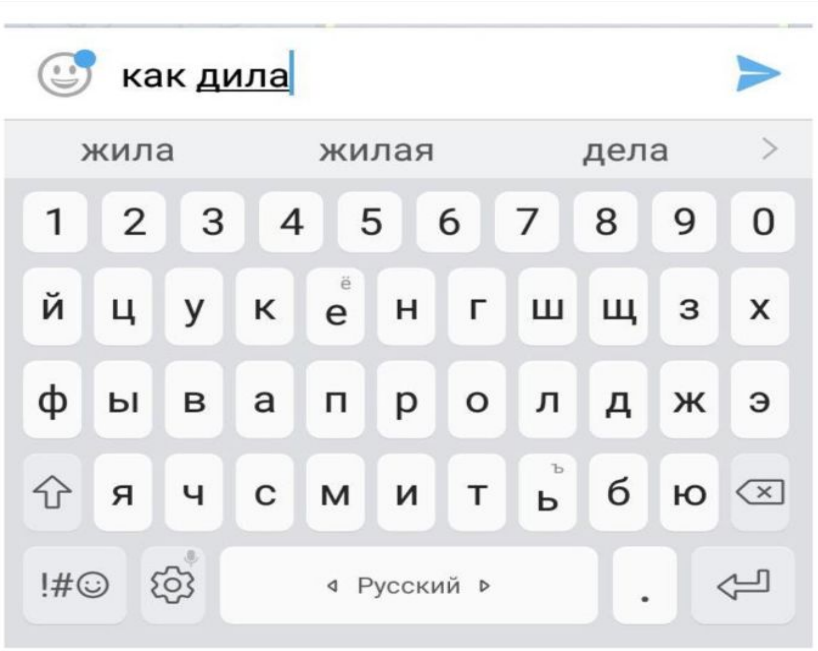
- **Language model** (LM) is a model that aims to estimate the probability of various linguistic units:
  - Characters or symbols
  - Tokens
  - Sentences
  - Documents
- The probabilities reflect the knowledge of language

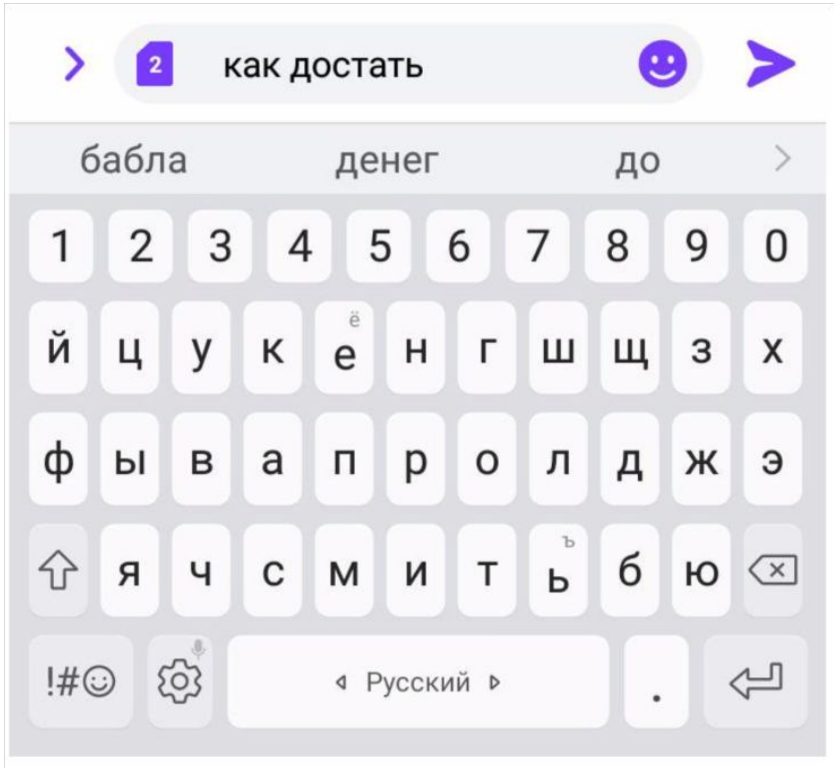
We try to model language

Applications

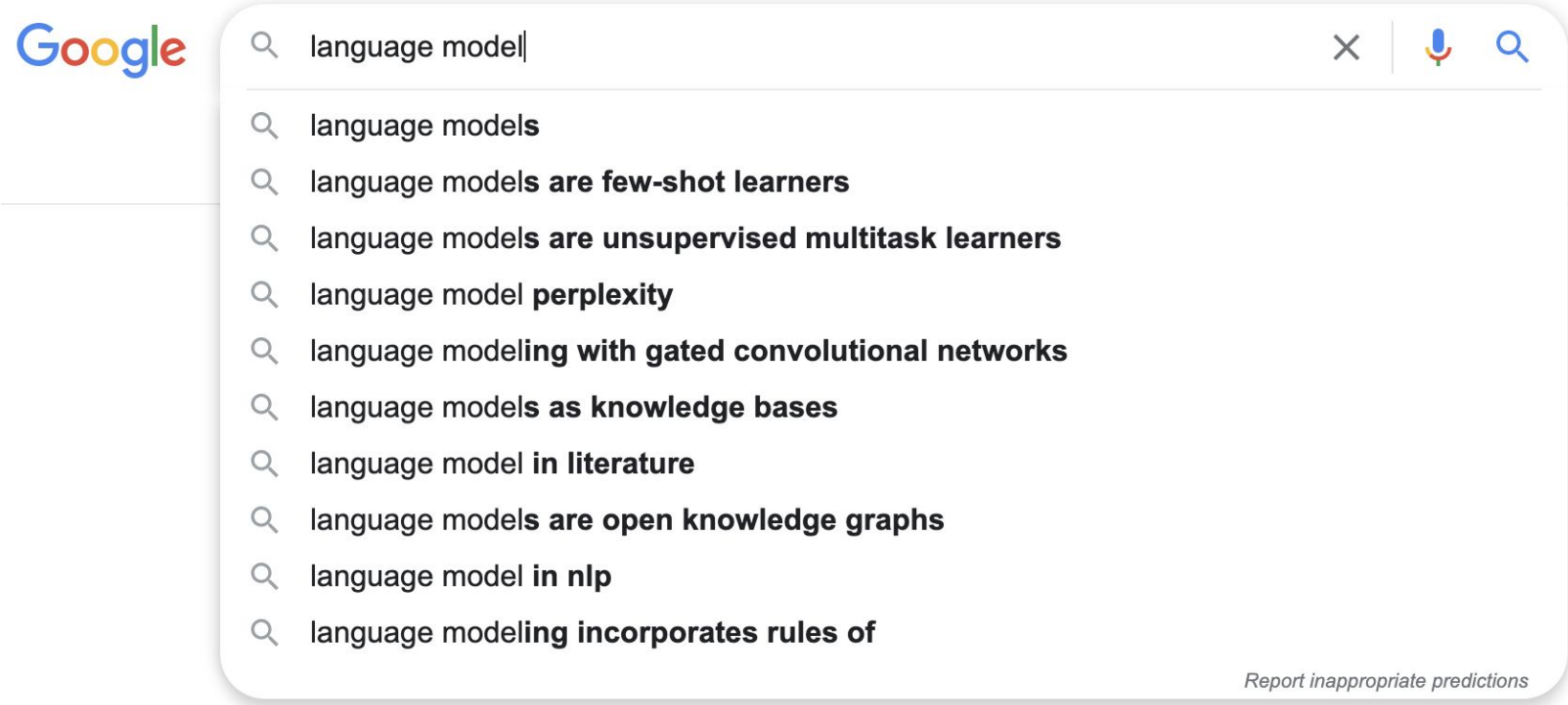
- Language detection
- Spell-checkers
- Keyboard autocomplete
- Email services
- Web search engines (Google, etc.)
- Machine translation
- Automatic speech recognition
- Text generation, chatbots etc.

Applications





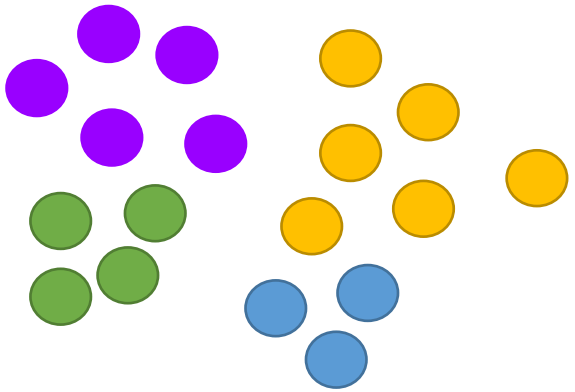
Applications



Probabilistic language models

Sentence probability

- We can calculate the probability of picking a green ball from the basket
- Can we do the same for words in sentences?



$$\frac{5}{5+6+4+3} = \frac{5}{18}$$

Sentence probability

- Formal task definitions:
  - $S = (w_1, w_2, \dots, w_n) - sentence, w_i - word, V - vocabulary$
- The probability of the sentence S:
  - $P(S) = P(w_1, w_2, \dots, w_n)$
- The probability of the next word in the word sequence:
  - $P(w_n | w_{n-1}, w_{n-2}, \dots, w_1)$

Sentence probability

- Chain rule:

$$P(S) = P(w_1, w_2, \dots, w_n)$$
$$= P(w_1)P(w_2|w_1)P(w_3|w_1w_2) \dots P(w_n|w_1, \dots w_{n-1})$$
$$= \prod_{i=1}^{n} P(w_i|w < i)$$

$$P(\text{«}We\ study\ NLP\text{»}) = P(\text{«}We\text{»}) \cdot P(\text{«}study\text{»}|\text{«}We\text{»}) \cdot P(\text{«}NLP\text{»}|\text{«}We\ study\text{»})$$

Left-to-right modeling

- Left-to-right modeling:

$$P(S) = \prod_{i=1}^{n} P(w_i | w < i)$$

- *Context* is a sequence of preceding words (tokens)

Left-to-right modeling

Maximum likelihood estimate of probabilities:

$$P(w_i|w_{1:i-1}) = \frac{counter(w_{1:i-1}w_i)}{counter(w_{1:i-1})}$$

Given a document collection *D*:

$$P(\text{«}We\ study\ NLP\text{»}) =$$

$$\frac{count(\text{«}We\ study\ NLP\text{»})}{count(\text{«}We\ study\text{»})} = \frac{2}{8} = \frac{1}{4}$$

Markov model

- **Andrei Markov (1856-1922)** is one of the most famous Russian mathematicians
- A primary research subjects are *Markov chains* and *Markov processes*

en.wikipedia.org

Markov model

- The independency assumption:
  - *"The probability of a word depends on a fixed number of previous words"*

$$P(w_i|w_1, \dots, w_{i-1}) = P(w_i|w_{i-n+1}, \dots, w_{i-1}).$$

For example:

- *N=1 (unigram model)* – $P(w_i|w_1, \dots, w_{i-1}) = P(w_i)$
- *N=2 (bigram model)* – $P(w_i|w_1, \dots, w_{i-1}) = P(w_i|w_{i-1})$
- *N=3 (trigram model)* – $P(w_i|w_1, \dots, w_{i-1}) = P(w_i|w_{i-2}, w_{i-1})$

Markov model

- Unigram model: $P(w_i)$

$$P(«We\ study\ NLP») = P(«We»)P(«study»)P(«NLP»)$$

- Bigram model: $P(w_i|w_{i-1})$

$$P(«We\ study\ NLP») = P(«We») \cdot P(«study»|«We») \cdot P(«NLP»|«study»)$$

N-grams models:
1) choose the most probable W_{i+1}
2) randomly select sample from this probability distribution of next words

Smoothing

- Smoothing allows to avoid the problem of zero probabilities

$$P(NLP \mid We\ study) = \frac{counter(We\ study\ NLP) = 0?}{counter(We\ study) = 0?}$$

Backoff - use bigrams instead of trigrams, unigrams instead of bigrams...

Linear interpolation

- Vector of positive weights or *lambdas:* $[\lambda_0, \lambda_1, \ldots, \lambda_{n-1}]$, $\sum_i \lambda_i = 1$
- Each probability gets weighted

$$P(NLP \mid We\ study)$$
$$\approx \lambda_2 P(NLP \mid We\ study) + \lambda_1 P(study \mid We) + \lambda_0 P(We)$$

https://web.stanford.edu/~jurafsky/slp3/3.pdf

Laplace smoothing

Add extra coefficients (for example one) to all the ngram counts, before we normalize them into probabilities. All the counts that used to be zero will now have a count of 1, the counts of 1 will be 2, and so on.

$$P(NLP \mid We\ study) = \frac{\delta + counter(We\ study\ NLP)}{\delta \cdot |V| + counter(We\ study)}$$

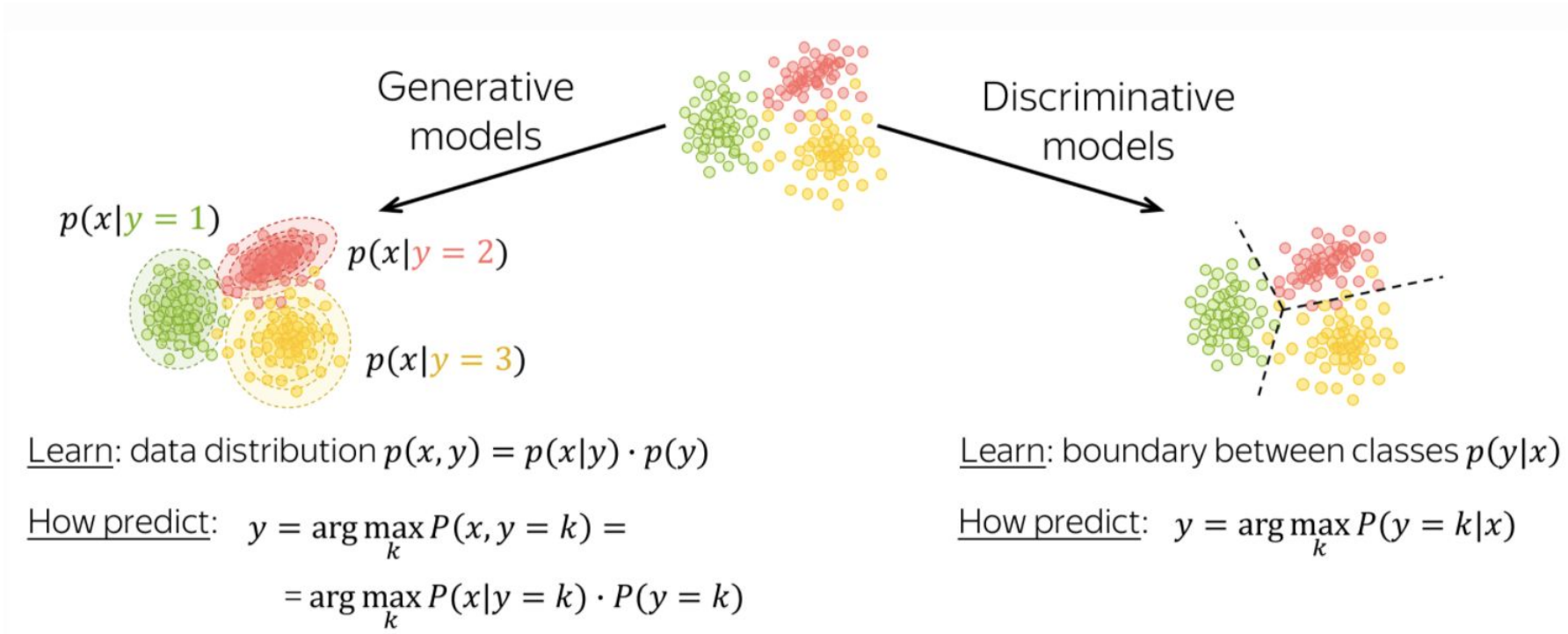
Document LM

Given the document D and query q, estimate the probability of generating the query text from a document language model:

- Rank document by the probability that the query could be generated by the document model;
- Calculate P(d|q) to rank the documents: P(d|q) α P(q|d)P(d)
- Assuming prior is uniform, unigram model: P(q|d) = $\prod_i$ (Pq_i|d)

$$\text{MLE: } P(q_i|d) = \frac{\text{count}(q_i,d)}{|d|}$$

Generative vs discriminative



Generative models learn joint probability distribution of data $p(x,y)=p(x|y) \cdot p(y)$. To make a prediction given an input x, these models pick a class with the highest joint probability.

Discriminative models are interested only in the conditional probability $p(y|x)$, i.e. they learn only the border between classes. To make a prediction given an input x, these models pick a class with the highest conditional probability

Generative sequence models

Part of speech tagging

Given a sentence or a sequence of words, predict its part of speech tag (Noun, Verb, Preposition, etc.)

the	cat	sat	on	a	mat
DET	NOUN	VERB	PREP	DET	NOUN

Generative model: P(y, x)

Hidden Markov model (HMM)

A Markov chain is useful when we need to compute a probability for a sequence of **observable** events.

A hidden Markov model (**HMM**) allows us to talk about:
- **observed** events hidden Markov model
- **hidden** events (like *part-of-speech* tags) that we think of as causal factors in our probabilistic model.

For part-of-speech tagging, the goal of HMM decoding is to choose the tag sequence $t_1 \ldots t_n$ that is most probable given the observation sequence of n words $w_1 \ldots w_n$

Hidden Markov model (HMM)

$$p(x,y) = p(x|y)p(y) \approx \prod_{t=1}^{T} p(x_t|y_t)p(y_t|y_{t-1})$$

observable

hiden

Markov assumption:

the probability of a particular state is dependent only on the previous state

$$p(y) \approx \prod_{t=1}^{T} p(y_t|y_{t-1})$$

Output independence:

$$p(x|y) \approx \prod_{t=1}^{T} p(x_t|y_t)$$

the probability of an output observation oi depends only on the state that produced the observation qi and not on any other states or any other observations

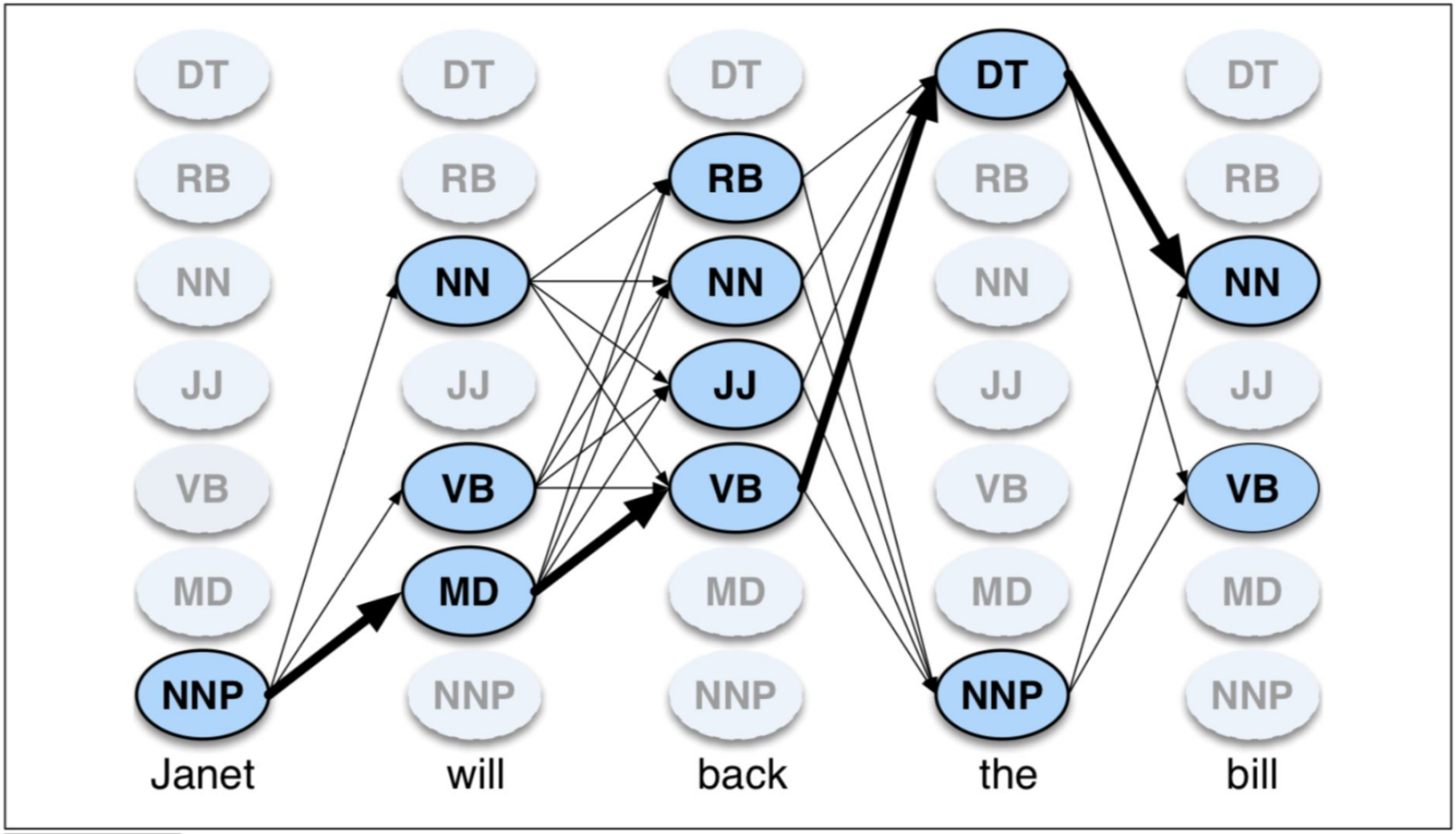
Hidden Markov model (HMM)

Three tasks of HMM

- **Likelihood**: given an observation sequence, estimate the likelihood of the observation sequence. *Forward-backword algorithm*.
- **Decoding**: given an observation sequence, discover the best hidden state sequence leading to these observations. *Viterbi*
- **Learning**: train HMM. *Baum-Welch*

https://web.stanford.edu/~jurafsky/slp3/A.pdf

Viterbi Algorithm



The algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence

Viterbi Algorithm

Compute path probabilities $V = |n \times T|$. v_{ij} represents the probability that the HMM is in state j after seeing the first i observations.

1. Intialize

$$v_{1j} = a_{0j} b(o_1), 1 \leq j \leq T$$

2. Recursion

$$v_{ij} = \max v_{i-1,k} a_{kj} b_j(o_i), 1 \leq i \leq n, 1 \leq j \leq T$$

3. End

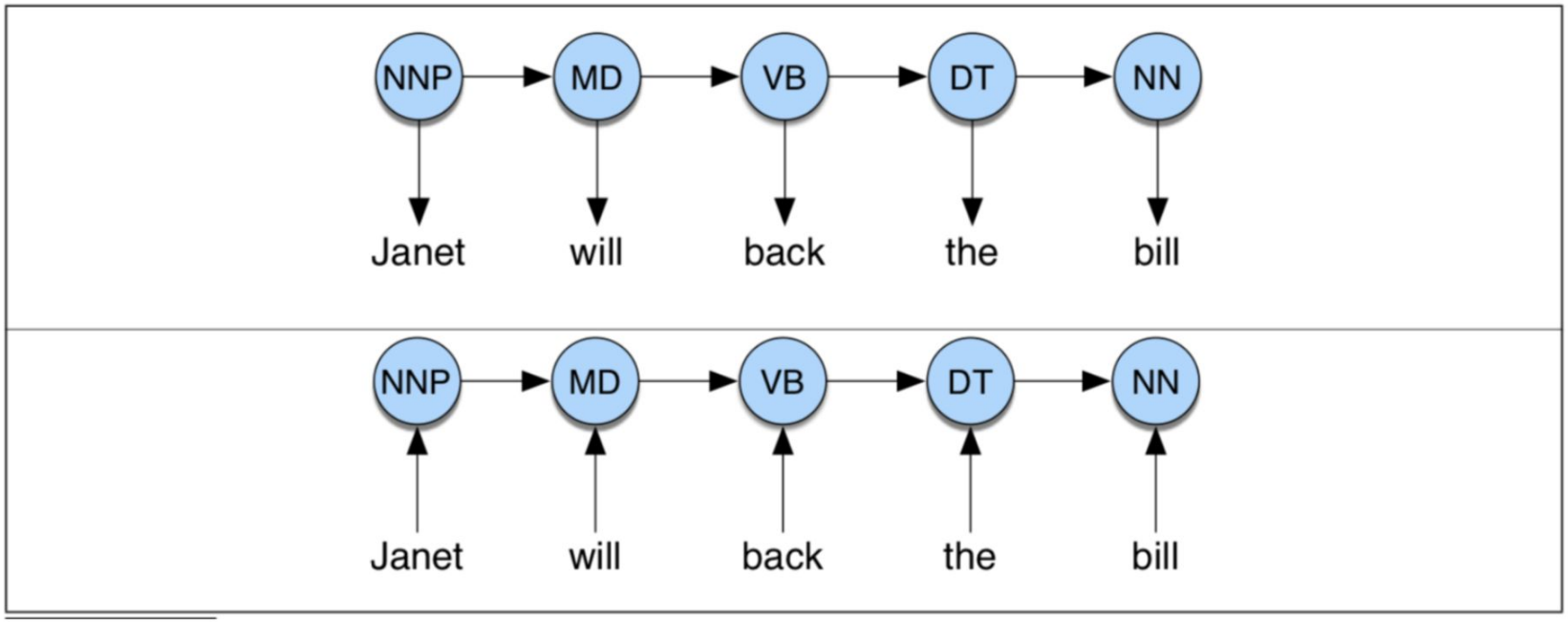
$$\max_{q \in Q^n} p(o, q) = \max_{1 \leq k \leq T} v_{nk} a_{kF}$$

https://www.davidsbatista.net/assets/documents/posts/2017-11-11-hmm_viterbi_mini_example.pdf

Maximum entropy Markov model (MEMM)

HMM: $\arg\max P(Y|X) = \arg\max_Y P(X|Y)P(Y)$

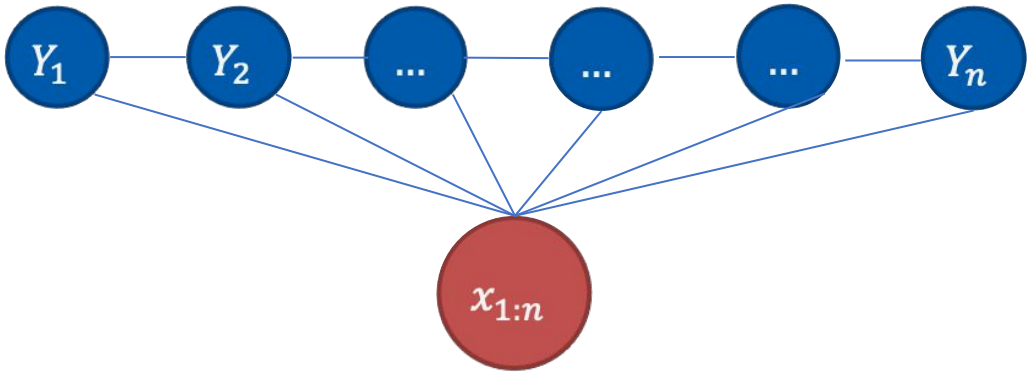
MEMM: $\arg\max_y P(Y|X) = \arg\max_Y P(y_i|y_{i-1}, x_i)$



In contrast to hidden Markov model (HMM), MEMM takes into account the dependencies between the neighboring states and the entire observe sequence

Conditional random field (CRF)

$$p(Y|X) = \frac{e^{\sum_{i=1}^{k} \lambda_i F_i(y,x)}}{\sum_{y' \in C^n} e^{\sum_{i=1}^{k} \lambda_i F_i(y',x)}}$$



- Conditional Random Field (CRF) is more complicated and can accommodate any context information by computing the conditional probabilities and considering the neighboring states as well.

Problems with count-based LMs

- Zero probabilities
- Scales up with the corpus size or the N-gram size
- The inability to use long contexts
- The texts end up generating the same phrase or token
- Incoherent and incohesive texts
- Corrupted agreement, grammar, etc.

Neural-based language models

Neural-based LMs

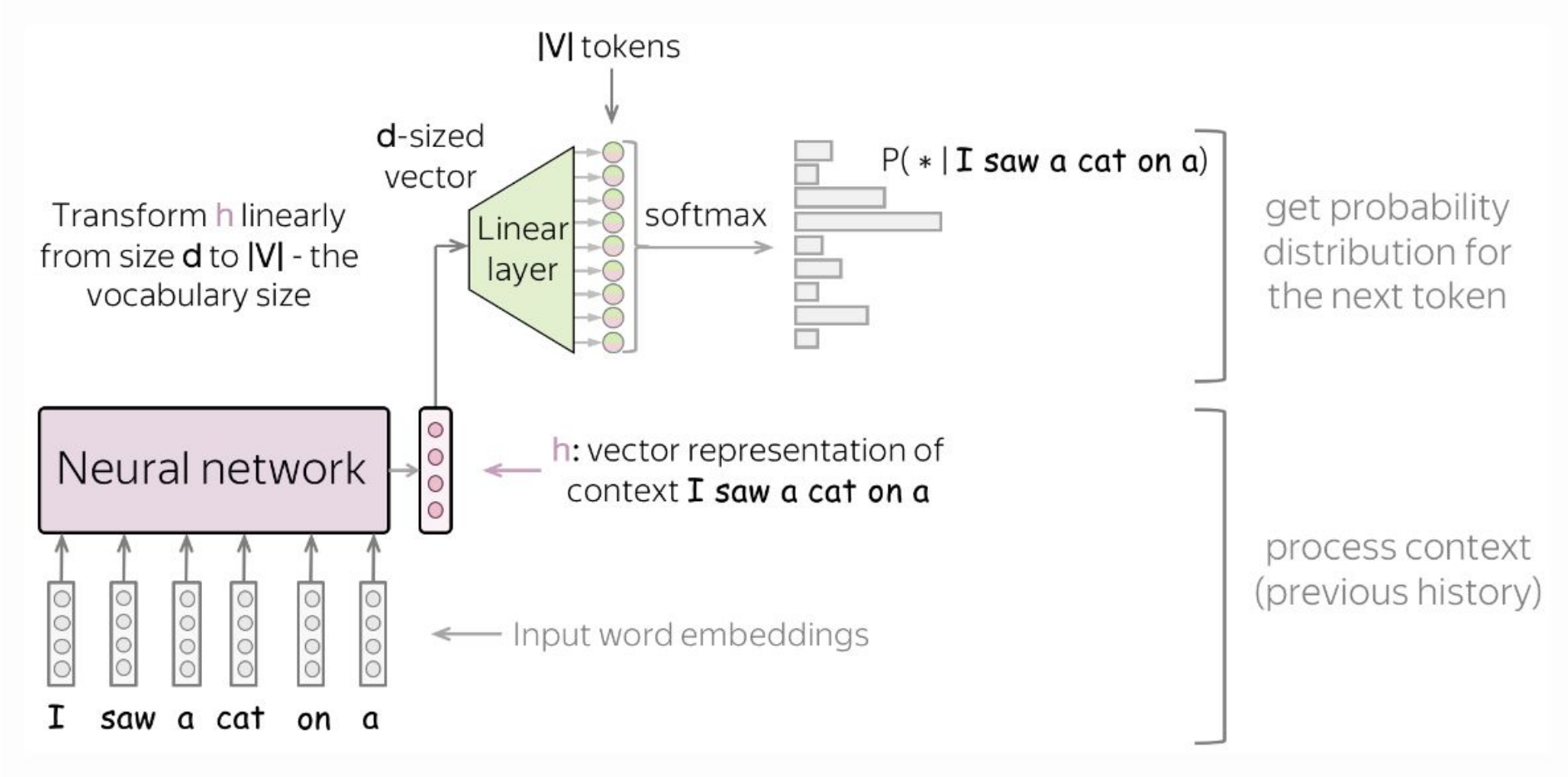
- How to predict the sentence probability?
- *Train a neural model*
- Vector representation of the context
- Using the context representation, predict the next word
- Does it remind you about multi-class classification?

General LM pipeline

- Split your document collection into N-grams
- Feed word embeddings for the context (preceding words) into your neural network
- Obtain *context representation* from the network
- Using this context representation, predict a probability distribution for the next word over |V| words, where |V| is the size of your vocabulary
- Loss function: Negative Log-likelihood

$$L = -\frac{1}{T} y \log(P(\hat{y}))$$

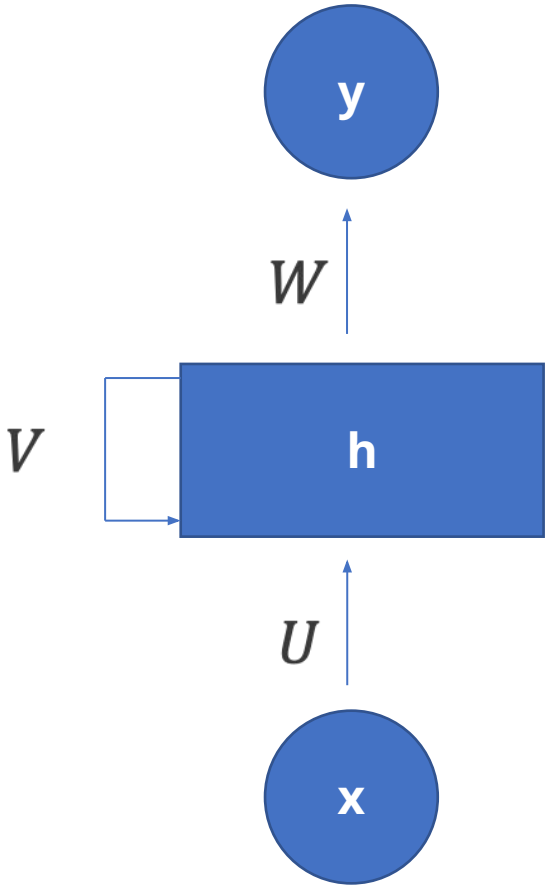
General LM pipeline



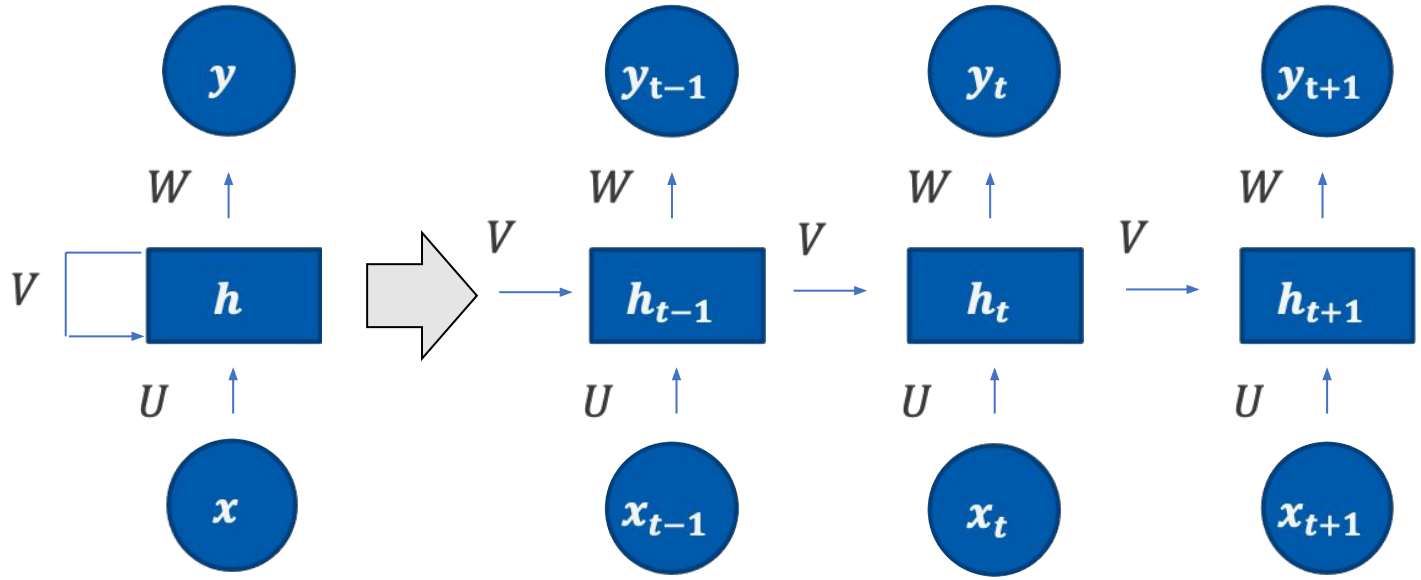
BUT: Generally slow; Expensive training; Does not need to store all the N-grams; Fixed context size

Recurrent neural network

- Allows to process sequences of unrestricted size
- Memory mechanism: the hidden state vector stores previously processed words (*context vector*)

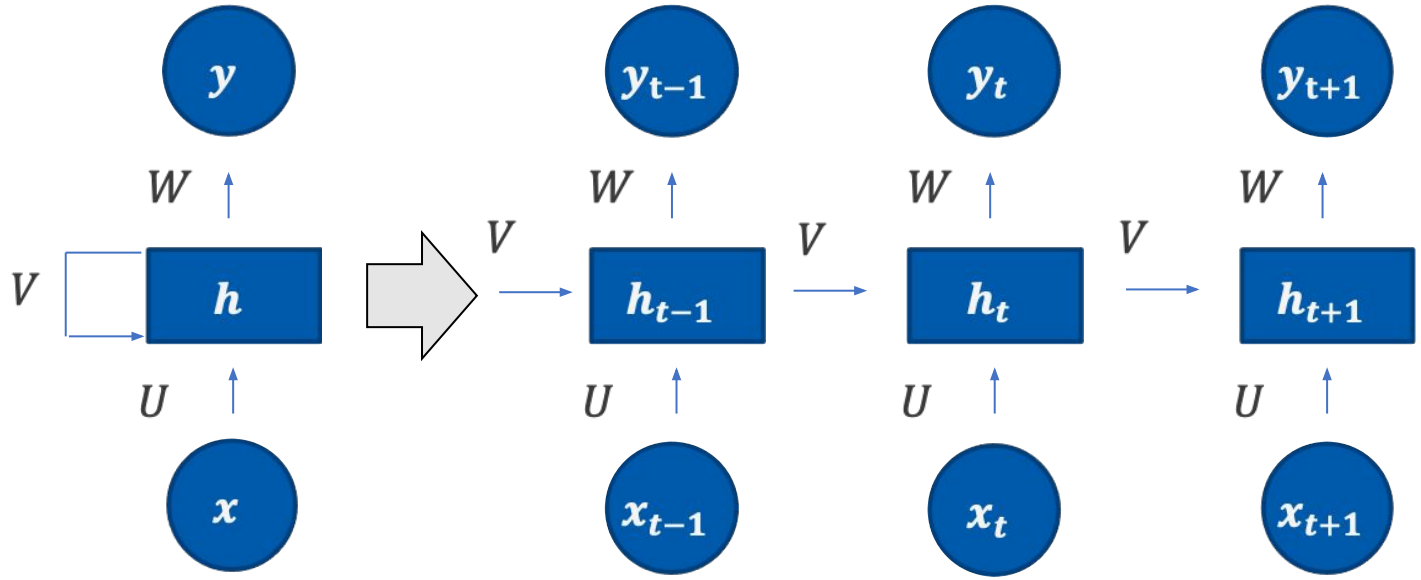


Recurrent neural network



- The input: word embeddings (***x***)
- The context vector: ***h***
- ***U, W, V*** are trainable weight matrices
- The output: ***y*** (words in your ***V***)

Recurrent neural network



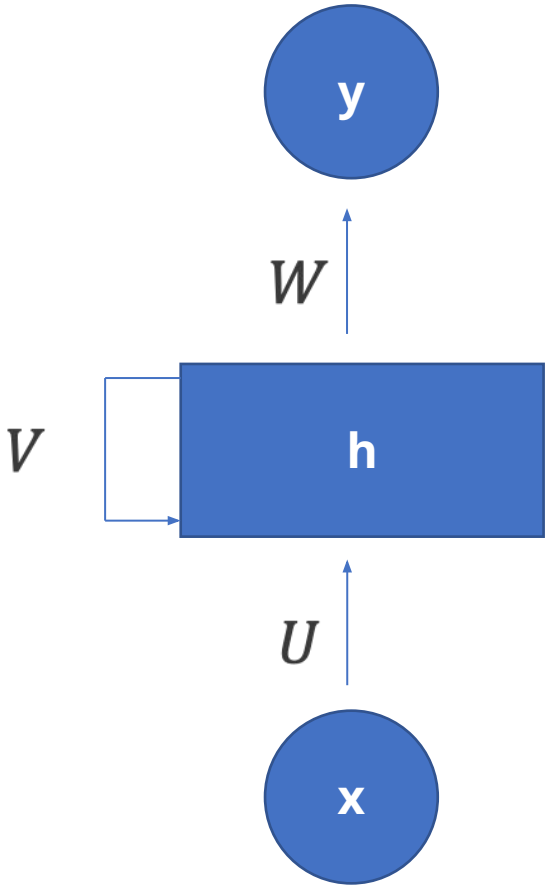
- $h_t = \sigma(V\, h_{t-1} + Ux_t + b_1)$ – updating the memory vector
- $\hat{y}_t = softmax(Wh_t + b_2) \in \mathbb{R}^{|V|}$ – prediction

Recurrent neural network. Training

- Tokenize texts in your document collection $\boldsymbol{w_1, w_2, \ldots, w_n}$
- You can use pre-trained word embeddings or train your own jointly with the network $\boldsymbol{x_i = E w_i}$
- Prepare the context

$$\boldsymbol{y_1 = w_2, y_{n-1} = w_n}$$

- Predict the next word $\boldsymbol{(x_i, y_i)}$



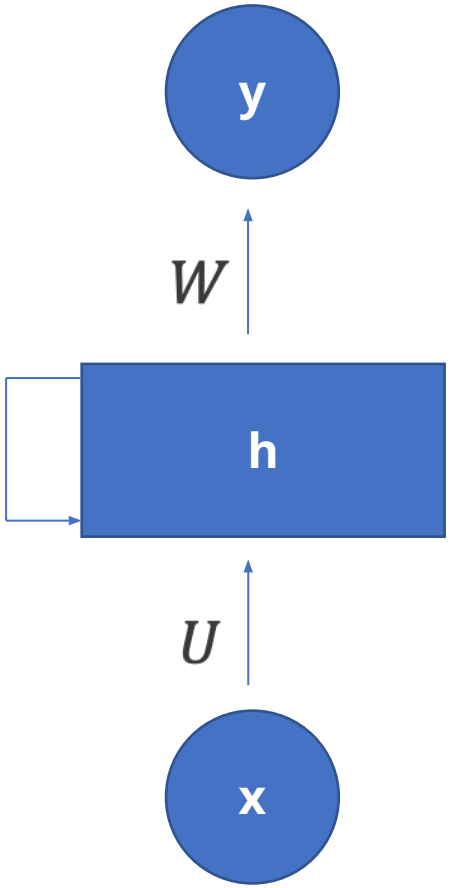
Recurrent neural network. Training

- At each time step ***t*** we compute the loss function, typically cross-entropy loss

$$loss_{local} = CE(y_t, \hat{y}_t)$$
$$= -\sum_{w \in V} y_t^w \log \hat{y}_t^w$$

- Global loss function:

$$loss_{global} = \sum loss_{local}$$



Vanishing gradient problem

Training a network works through tree major steps:

- forward pass and makes a prediction
- compare the prediction to the ground truth using a loss function
- back propagation (calculates the gradients for each node in the network)

Each node in a layer calculates it's gradient with respect to the effects of the gradients, in the layer before it. So if the adjustments to the layers before it is small, then adjustments to the current layer will be even smaller.

Gradients shrink as it back-propogates down

Easrliest layers are not learning =(

Summary

Pros:

- Do not depend on the size of the context
- Context vector, or memory vector
- The number of parameters does not scale with the size of your corpus

Cons:

- Generally slow
- Cannot encode long range relationships (vanishing gradient problem)
- Can “forget” relevant information (Short-term memory)

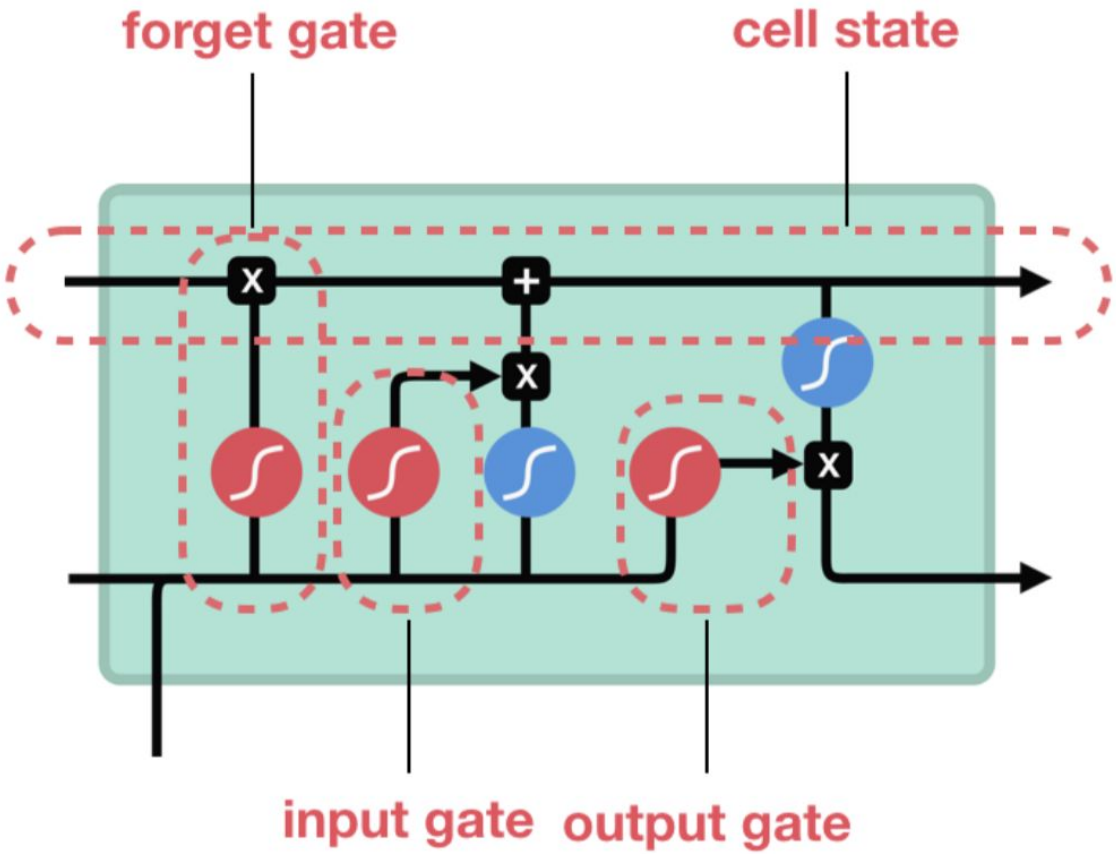
Improvements of RNN

- Long Short-Term Memory (LSTM) Network

  [Hochreiter and Schmidhuber, 1997]

- Gated Recurrent Unit (GRU) [Cho et al., 2014]

LSTM


forget gate
cell state
input gate
output gate


sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation

LSTM

Forget gate: decides what information should be thrown away or kept. Information from the previous hidden state and info from the current input => the sigmoid function (values between 0 and 1).

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

Input gate: decides what to update in the cell state. Previous hidden state and current input => into a sigmoid function (0 and 1 - important). That decides which values will be updated. Hidden state and current input go into the tanh function (-1 and 1) to help regulate the network. Multiply the tanh output with the sigmoid output.

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

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

LSTM

Update and get the new cell state. Cell gets pointwise multiplied by the forget vector. The output from the input gate => pointwise addition which updates the cell state to new values that the neural network finds relevant.

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

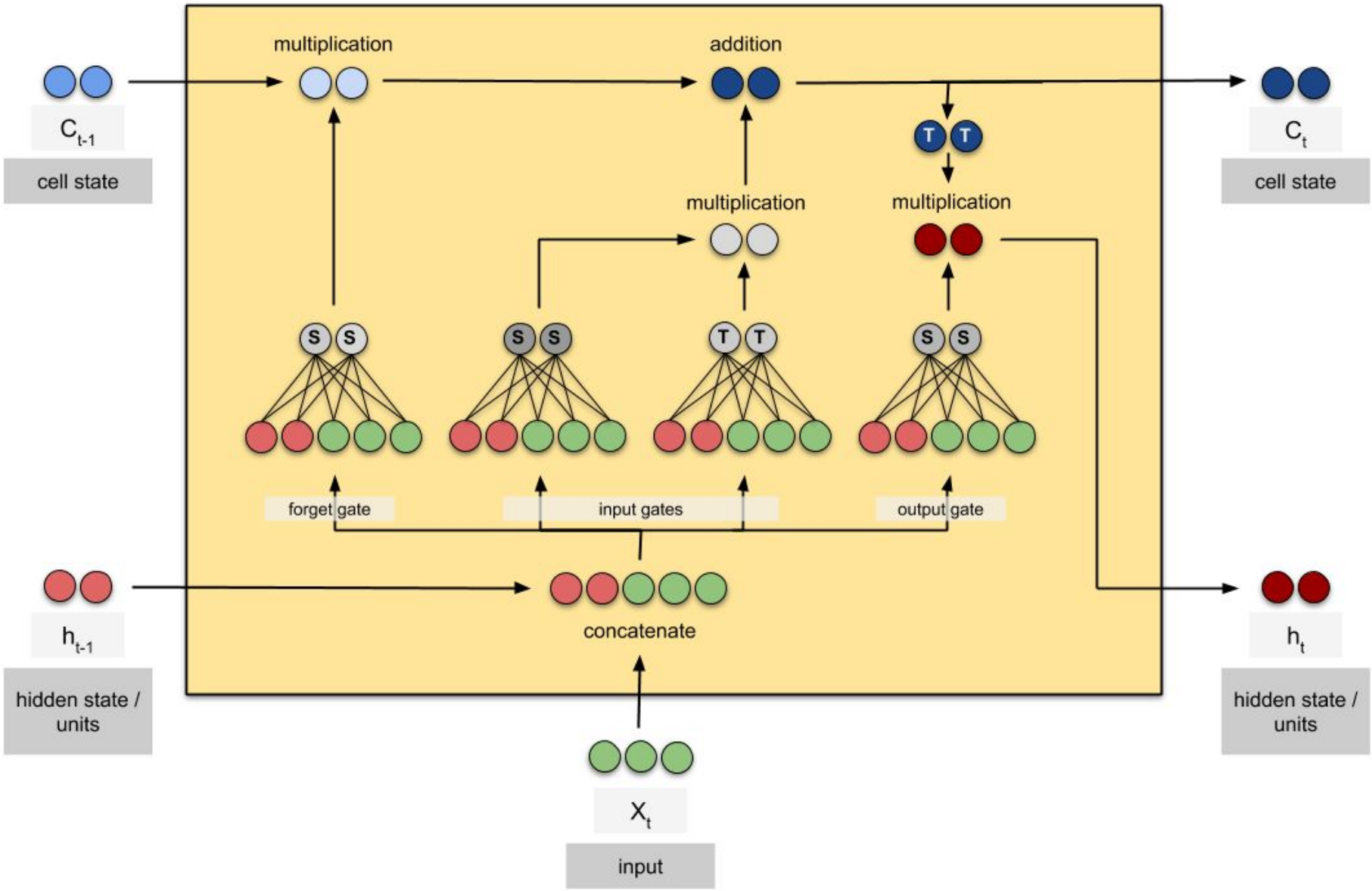
Output gate: decides what the next hidden state should be. Pass the previous hidden state and the current input into a sigmoid function. Modified cell state => to the tanh function. Multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden => to the next time step.

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

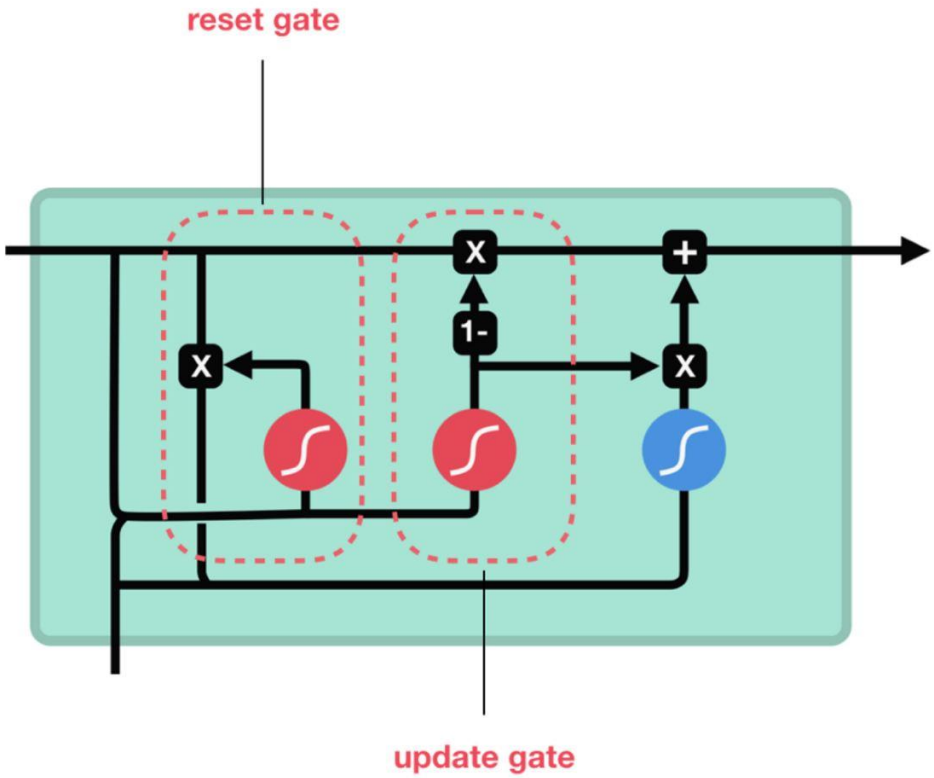
$$h_t = o_t * \tanh\left(C_t\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM


multiplication
addition
C_{t-1}
cell state
C_t
cell state
multiplication
multiplication
forget gate
input gates
output gate
h_{t-1}
hidden state / units
concatenate
h_t
hidden state / units
X_t
input

GRU



GRU cell and it's gates

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

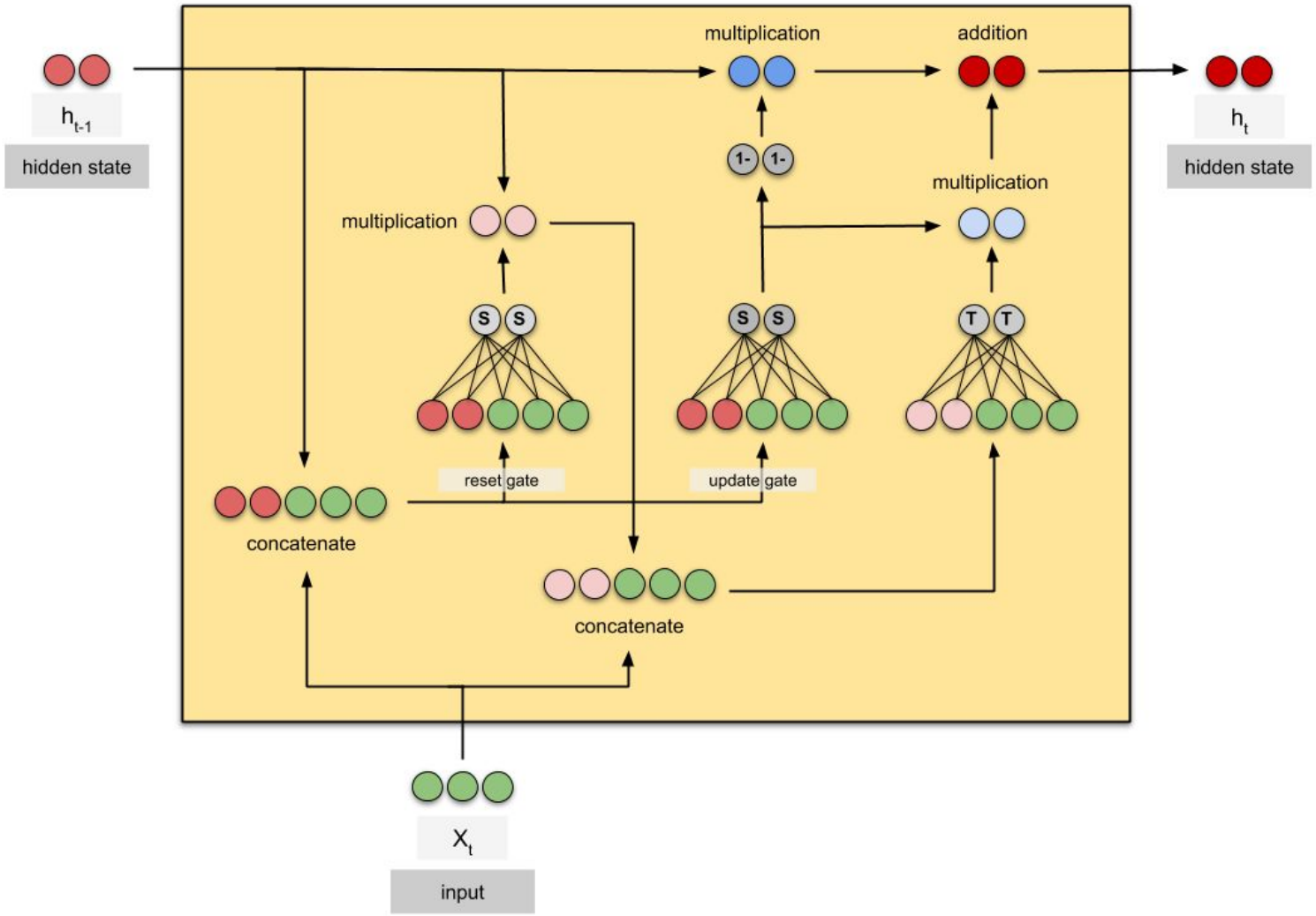
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

why?. You'll first read the review then determine if someone thought it was

GRU

The **Update gate** decides what information to throw away and what new information to add. The **Reset gate** is another gate is used to decide how much past information to forget.



Bidirectional networks

- We can process the input not only in the left-to-right manner, but also in the *bidirectional* manner (left-to-right, right-to-left)
- **Left-to-right:**

$$«We», «study», «NLP»$$

- **Right-to-left**

$$«NLP», «study», «We»$$

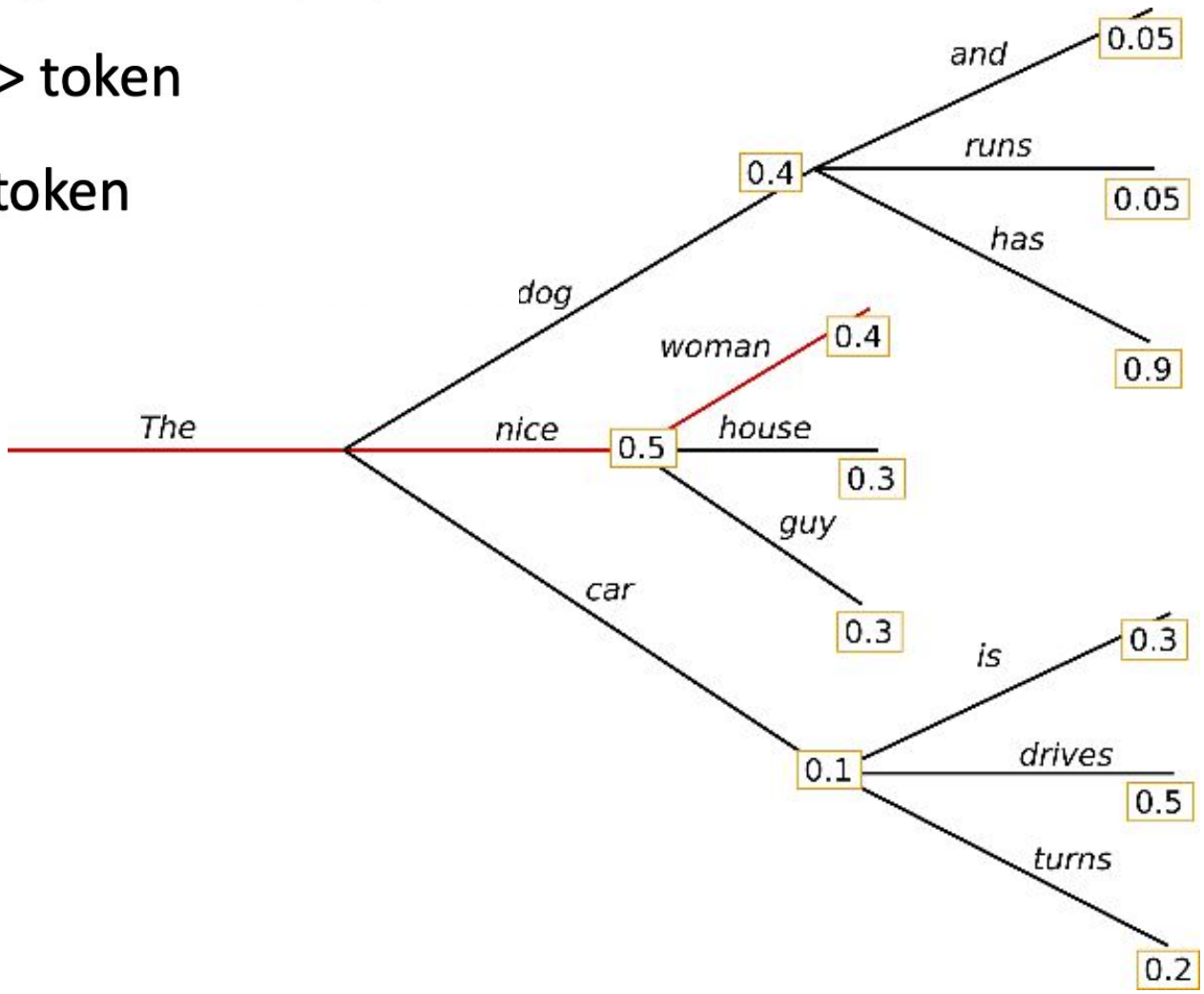
Text generation and evaluation

Task definition

- The input: the context $x_1, \ldots, x_m$
- The output: n next words $x_{m+1}, \ldots, x_{m+n}$
- The *generation strategy* determines the way LMs predict the next word:
  - Greed search

Greed search

- $w_t = argmax_w P(w \mid w_{1:t-1})$
- Context: <bos> token
- Finish: <eos> token



Greed search

- Deterministic algorithm
- Simple and fast
- Predicts the most optimal token at each time step
- Can miss high probability tokens that are “hidden” behind low probability tokens
- The generated texts are typically not versatile
- Other generation strategies: **See next weeks**

Text generation problems

- Text *degeneration* [Holzman et al., 2020]
- Repetitions of words, sentences, segments
- Incoherent and incohesive texts
- Factual incorrectness
- Grammar errors, etc.
- Evaluation of generated texts

[Holtzman et al., 2020] The Case of Neural Text Degeneration

Intrinsic evaluation

Perplexity (PPL)

- **inverse probability of the test set, normalised by the number of words**

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}}$$

$$\begin{aligned} PP(W) &= 2^{-\frac{1}{N}\log_2 P(w_1,w_2,\ldots,w_N)} \\ &= \left(2^{\log_2 P(w_1,w_2,\ldots,w_N)}\right)^{-\frac{1}{N}} \\ &= P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}} \end{aligned}$$

- **using the cross-entropy**

(it indicates the average number of bits needed to encode one word, and perplexity is the number of words that can be encoded with those bits)

$$PP(W) = 2^{H(W)} = 2^{-\frac{1}{N}\log_2 P(w_1,w_2,\ldots,w_N)}$$

The lower perplexity = the better

Values from 1 to |V|

https://towardsdatascience.com/perplexity-in-language-models-87a196019a94

Intrinsic evaluation

- LAnguage Model Analysis (LAMA) [Petroni et al., 2019]
- Top-k predictions:
  - *The cat is on the ___ .*

github.com/facebookresearch/LAMA

Extrinsic evaluation

- Downstream tasks:
  - spell checking
  - punctuation restoration
  - text summarization, etc.
- Natural Language Generation (NLG) benchmarks:
  - GEM [Gehrmann et al., 2021]

Extrinsic evaluation

- GEM tasks, e.g.:
  - Describe a restaurant, given all and only the attributes specified on the input
  - Summarize relevant points within a news article
  - Communicate the same information as the source sentence using simpler words and grammar
  - Generate a high-quality summary of an instructional article

Sequence labelling

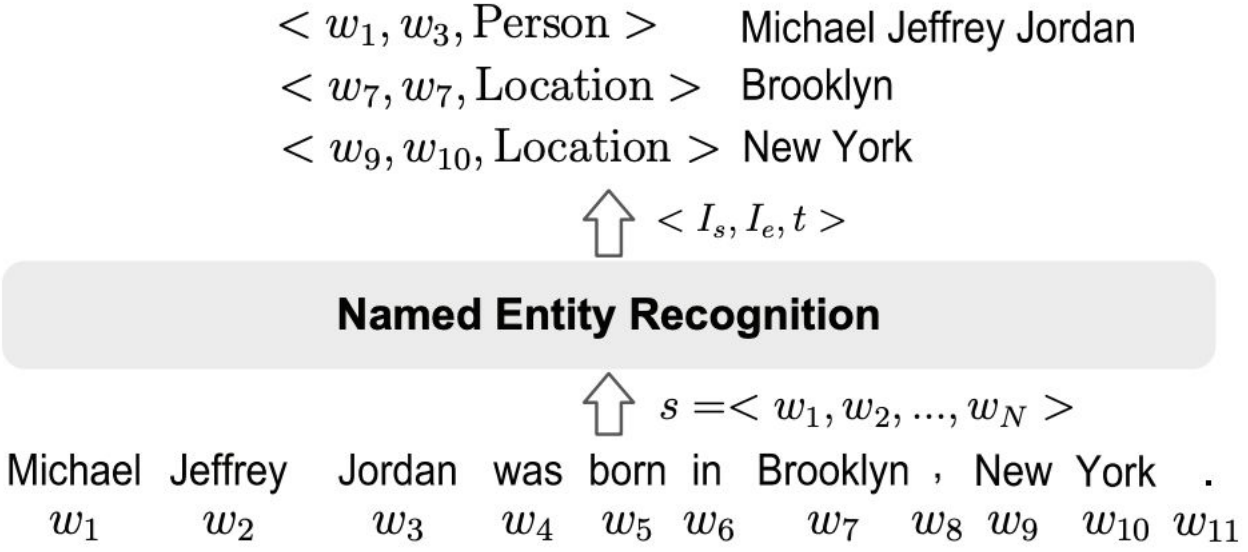
Named entity recognition

- Named Entity Recognition (NER) is one of the most popular sequence labelling tasks aimed at recognizing mentions of *named entities* in a text
- Named entities can be divided into:
  - generic (person, location, organization, etc.)
  - domain-specific (proteins, genes, etc.)

Applications

- Recognition of:
  - entities for question answering
  - entities in legal documents
  - medical terms in the patient records
  - the names of the products in the user queries, etc.

Task definition

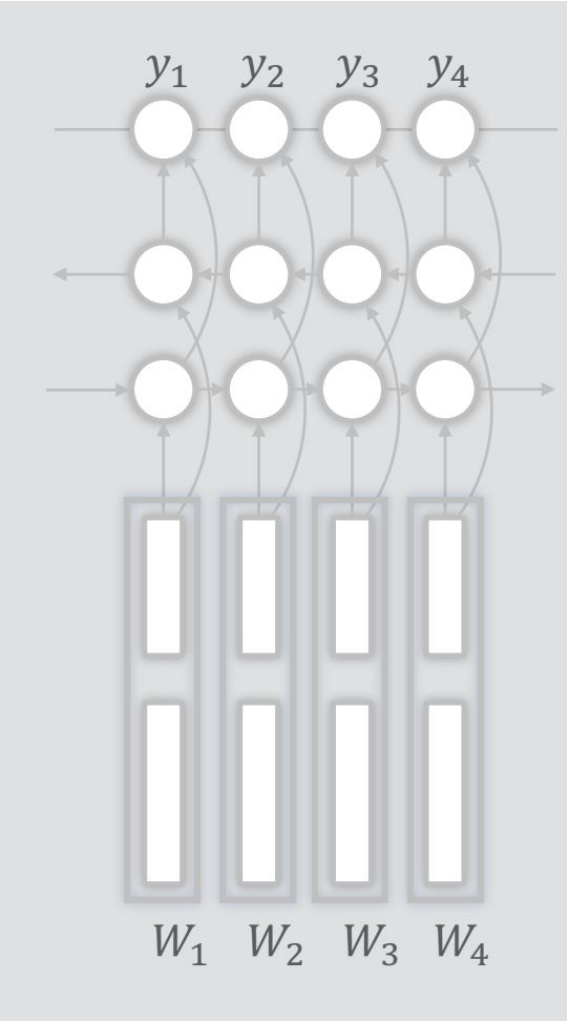


Token	Label
To	O
First	B-ORG
National	I-ORG
Bank	I-ORG

Tagging schema. IOB (Inside-outside-beginning)

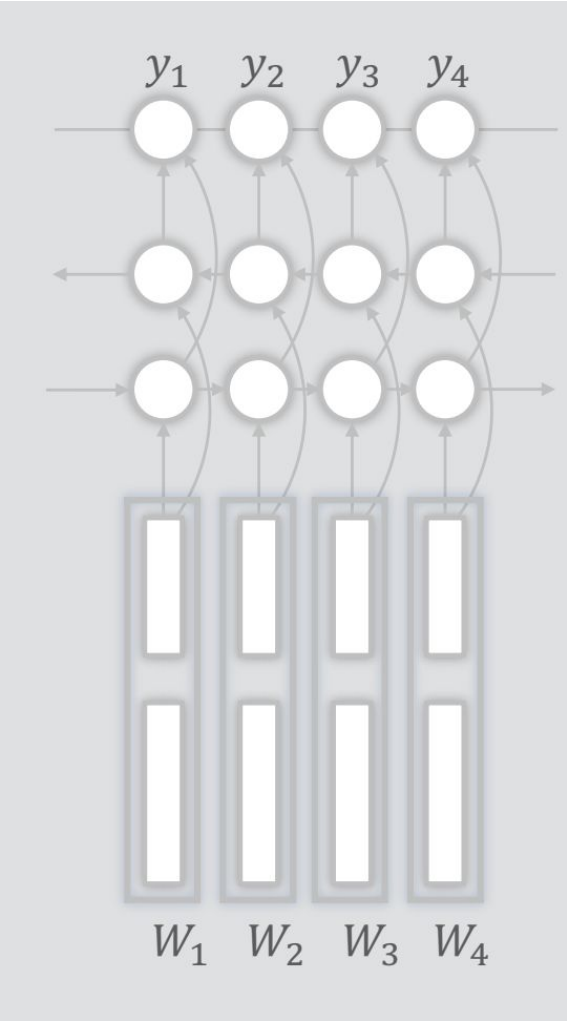
CNN + biLSTM + CRF

- Two types of the input representations:
  - Word-level
  - Character-level CNN
- You can use pre-trained word embeddings such as word2vec, fastText
- Character-level embeddings are trained jointly with the model



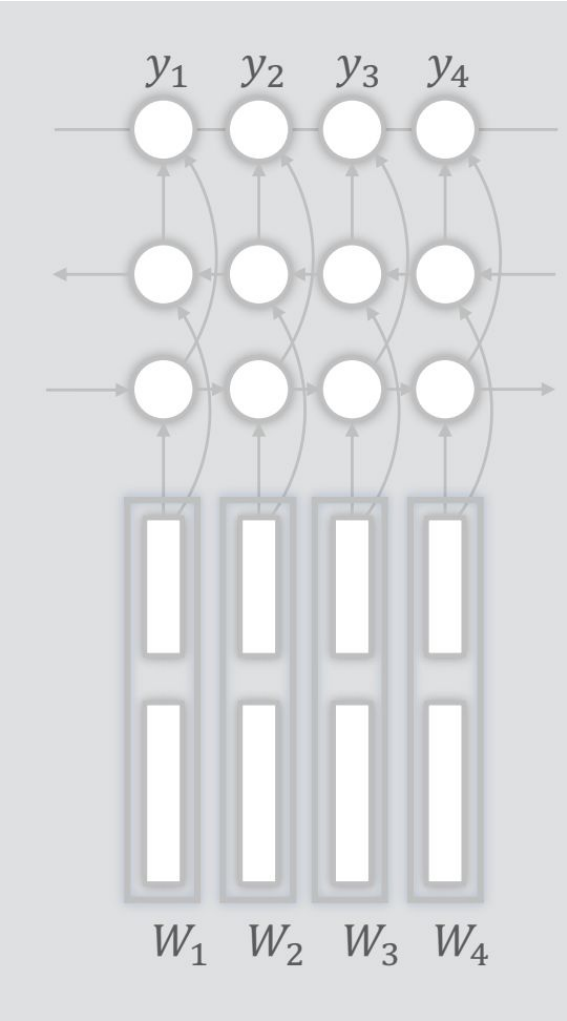
CNN + biLSTM + CRF

- Bidirectional LSTM:
  - Left-to-right
  - Right-to-left
- We get the *contextualized* representations



CNN + biLSTM + CRF

- The input to CRF is the output of the LSTM block
- Assigns the probabilities of the labels to each input token
- The output is the most probable combination of labels for each words considered together



NER evaluation

Precision, Recall, F-score (balanced, macro-average)

$$\text{Precision} = \frac{\#TP}{\#(TP + FP)} \qquad \text{Recall} = \frac{\#TP}{\#(TP + FN)}$$

$$\text{F-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

NER datasets

Corpus	Year	Text Source
MUC-6	1995	Wall Street Journal
MUC-6 Plus	1995	Additional news to MUC-6
MUC-7	1997	New York Times news
CoNLL03	2003	Reuters news
ACE	2000 - 2008	Transcripts, news
OntoNotes	2007 - 2012	Magazine, news, web, etc.
W-NUT	2015 - 2018	User-generated text
BBN	2005	Wall Street Journal
WikiGold	2009	Wikipedia
WiNER	2012	Wikipedia
WikiFiger	2012	Wikipedia
HYENA	2012	Wikipedia
N^3	2014	News
Gillick	2016	Magazine, news, web, etc.
FG-NER	2018	Various
NNE	2019	Newswire
GENIA	2004	Biology and clinical text
GENETAG	2005	MEDLINE
FSU-PRGE	2010	PubMed and MEDLINE
NCBI-Disease	2014	PubMed
BC5CDR	2015	PubMed
DFKI	2018	Business news and social media

For Russian:

Persons 1000
http://ai-center.botik.ru/Airec/index.php/ru/collections/28-persons-1000

Fact-RU-Eval
https://github.com/dialogue-evaluation/factRuEval-2016

BSNLP Shared task
http://bsnlp.cs.helsinki.fi/bsnlp-2019/shared_task.html

[Li et al., 2020] A survey on Deep Learning for Named Entity Recognition

NER open-source tools

NER System	URL
StanfordCoreNLP	https://stanfordnlp.github.io/CoreNLP/
OSU Twitter NLP	https://github.com/aritter/twitter_nlp
Illinois NLP	http://cogcomp.org/page/software/
NeuroNER	http://neuroner.com/
NERsuite	http://nersuite.nlplab.org/
Polyglot	https://polyglot.readthedocs.io
Gimli	http://bioinformatics.ua.pt/gimli
spaCy	https://spacy.io/api/entityrecognizer
NLTK	https://www.nltk.org
OpenNLP	https://opennlp.apache.org/
LingPipe	http://alias-i.com/lingpipe-3.9.3/
AllenNLP	https://demo.allennlp.org/
IBM Watson	https://natural-language-understanding-demo.ng.bluemix.net
FG-NER	https://fgner.alt.ai/extractor/
Intellexer	http://demo.intellexer.com/
Repustate	https://repustate.com/named-entity-recognition-api-demo
AYLIEN	https://developer.aylien.com/text-api-demo
Dandelion API	https://dandelion.eu/semantic-text/entity-extraction-demo
displaCy	https://explosion.ai/demos/displacy-ent
ParallelDots	https://www.paralleldots.com/named-entity-recognition
TextRazor	https://www.textrazor.com/named_entity_recognition

For Russian:

Natasha
https://github.com/natasha/natasha
https://github.com/natasha/slovnet

DeepPavlov NER
https://docs.deeppavlov.ai/en/0.1.5/components/ner.html

BSNLP solution
https://github.com/sberbank-ai/ner-bert

[Li et al., 2020] A survey on Deep Learning for Named Entity Recognition