NOTE: This PDF file only contains the first part of this 2-class presentation

NLP basics

Introduction to neural NLP

MVA - Speech and Language Processing #1+#2 (NLP 1+2)

Benoît Sagot & Robin Algayres
with many slides by Paul Michel

You shall know a word by the company it keeps.

J. R. Firth: ‘A Synopsis of Linguistic Theory’ (1957)

Outline

- Basic yet important linguistic concepts
  - Elementary processing units: sentences, tokens and subwords, “words”, named entities, and more
- Language models modelling vs. word vector representations
- Neural language modelling architectures
  - Recurrent neural networks
  - Attention and Transformers
- Pre-trained neural language models
  - ELMo
  - BERT and variants
  - GPT and descendents
  - BART and T5
- Transfer learning with pre-trained models
- Limitations in language models
- Sentence representations

Processing textual data

- A text is a **sequence of characters**
  - letters, ideograms, syllabograms…
  - punctuation marks
  - whitespace characters (not in all writing systems)
- Most NLP systems rely on a **double structuring** of such a sequence
  - Macroscopic units: “**sentences**” (utterances, speech turns)
  - Microscopic units: “**words**”
- Typically, sentences are processed individually as sequences of words
- This raises **two challenges**:

Processing textual data

- A text is a **sequence of characters**
  - letters, ideograms, syllabograms…
  - punctuation marks
  - whitespace characters (not in all writing systems)
- Most NLP systems rely on a **double structuring** of such a sequence
  - Macroscopic units: “**sentences**” (utterances, speech turns)
  - Microscopic units: “**words**”
- Typically, sentences are processed individually as sequences of words
- This raises **two challenges**:
  - How do we identify words and sentences in raw texts?
  - How do we represent words?

Sentences, words, tokens *et al.*

What is a sentence?

- Output of a **macroscopic segmentation**
  - **Self-contained syntactic structure**
  - Semantically related to other sentences at the "**discursive**" level
    - anaphora
    - discourse relations
  - (dubious) Prosodically marked (pauses, intonation)
  - Typographically marked
    - In the Latin script, the full stop (or period) is an ambiguous, sometimes overloaded symbol
    - The same holds for the uppercase first letter of a "sentence"

What is a sentence?

- This is an **idealised view**
  - Self-contained syntactic structures are not that frequent in speech
  - …and do not always match the “intuitive” notion of sentence
    “Malheur à toi si tu refuses”, le menaça-t-il.
  - Typography is sometimes misleading
    Best. Movie. Ever.
    Dès maintenant, la mobilisation est de mise. Pour l’amour des mots.
    The grocery sells cucumbers, lettuce, radishes, etc.
  - Nested structures, e.g. with quotes:
    "It is basically the perfect sort of tool to find objects like 'Oumuamua. We expect to find 100s of them with the LSST," Dr Fraser says.

What is a word?

- No linguist would dare define a unique notion “word”
- At least four notions must be distinguished:
  - The prosodic word
  - The typographic word, or **token** (or **pre-token**)
  - The morphosyntactic word, or **wordform** (or **form**)
  - The **semantic word**
- Let us review the last three concepts
  - And mismatches between these three notions

Tokens (today, sometimes “pre-tokens”)

- **A token is a purely typographic unit**
  - **Conventionally**, deterministically defined
- Starting point:
  - Many writing systems have “punctuation marks”
  - Some of them have a typographic separator (e.g. whitespace)
- In writing systems with a typographic separator, a token can be defined as:
  - A sequence of characters containing no separators and no punctuation marks,
  - or a punctuation mark (anything that is not a letter or a digit)
  - Ex.: *All of a sudden, he started playing table tennis with John Doe.*

Tokens (today, sometimes “pre-tokens”)

- **A token is a purely typographic unit**
  - **Conventionally**, deterministically defined
- Starting point:
  - Many writing systems have “punctuation marks”
  - Some of them have a typographic separator (e.g. whitespace)
- In writing systems with a typographic separator, a token can be defined as:
  - A maximal sequence of characters containing no separators and no punctuation marks (in a broad sense: . ; , ? / = + # @ & — and more!),
  - or a punctuation mark (anything that is not a letter or a digit)
  - Ex.: *All of a sudden , he started playing table tennis with John Doe .* (14 tokens)

Tokens (today, sometimes “pre-tokens”)

- In writing systems without a typographic separator, the simplest way to define a token is to consider each individual character as a token
  - Remember splitting a sentence into tokens must be **deterministic** by definition
  - Ex.: *我的漢語說得不太好。*

    ฉันฟังไม่เข้าใจ

    □□□□□

Tokens (today, sometimes “pre-tokens”)

- In writing systems without a typographic separator, the simplest way to define a token is to consider each individual character as a token
  - Remember splitting a sentence into tokens must be **deterministic** by definition
  - Ex.: 我 的 漢 語 說 得 不 太 好 。

    ฉัน ฟัง ไม่ เข้า ใจ

    □ □ □ □ □

“Subwords” or “tokens”

- In practice, we often need to deal with a **finite, fixed-size vocabulary**
  - E.g. certain classes of language models
- In such situations, an “optimal” vocabulary of character sequences is extracted
  - Vocabulary items are called BPEs (“byte pair encoding”), WordPieces, SentencePieces, etc., following the name of the vocabulary extraction tool/method
    Cf. for instance Sennrich et al. (2016), Kudo & Richardson (2018)
  - The generic terms “subword” or “token” are often used, despite their misleading surface relation with the word “word” and the confusion with the classical “token”
- In practice, subword boundaries are often represented differently whether they correspond or not to a whitespace (this information is relevant and must be preserved)
  - Ex.: *All _of _a _sud den _, _he _start ed _play ing _table _tennis _with _John Do e .*

Wordforms (or forms)

- A **wordform** is a syntactically atomic unit
  - It can receive annotations such as a part-of-speech (noun, adjective, etc…), morphological features (plural, dative…)
  - It corresponds to the leaves in syntactic structures
  - Not easy to identify! Ambiguities…
  - Sometimes, the token modifies the form (uppercase, errors…)
- **Mismatches** between tokens and forms are numerous:
  - 1 token corresponding to multiple forms = **amalgam**
    Ex.: aux (= à les), du (= de les OR du) won't (= will not) Sp. dámelo (= da me lo)
  - multiple tokens corresponding to 1 form = **compound word**
    Ex.: au fur et à mesure all of a sudden
  - multiple tokens corresponding to multiple forms
    Ex.: au fur et à mesure du (= au_fur_et_à_mesure_de le)

Named entities

- A **named entity** is a real-world object that can be denoted individually
  - Standard named entities: people, locations, organisations
  - Extended named entities: dates, addresses, URLs, e-mail addresses, numbers, etc.
- A named entity **mention** is an utterance denoting a named entity
  - Examples: Emmanuel Macron, Los Angeles, Apple Inc.
  - This denotation can be ambiguous…
- Named entity mentions have specific properties, apart from their specific, individual denotation:
  - Specific internal structure (**local grammar**), often culture-dependent more than language-dependent
    > **They can be viewed as atomic for the grammar of the language**, and therefore as **special forms**
- Ex.: *All of a sudden , he started playing table tennis with John Doe .*

Named entities

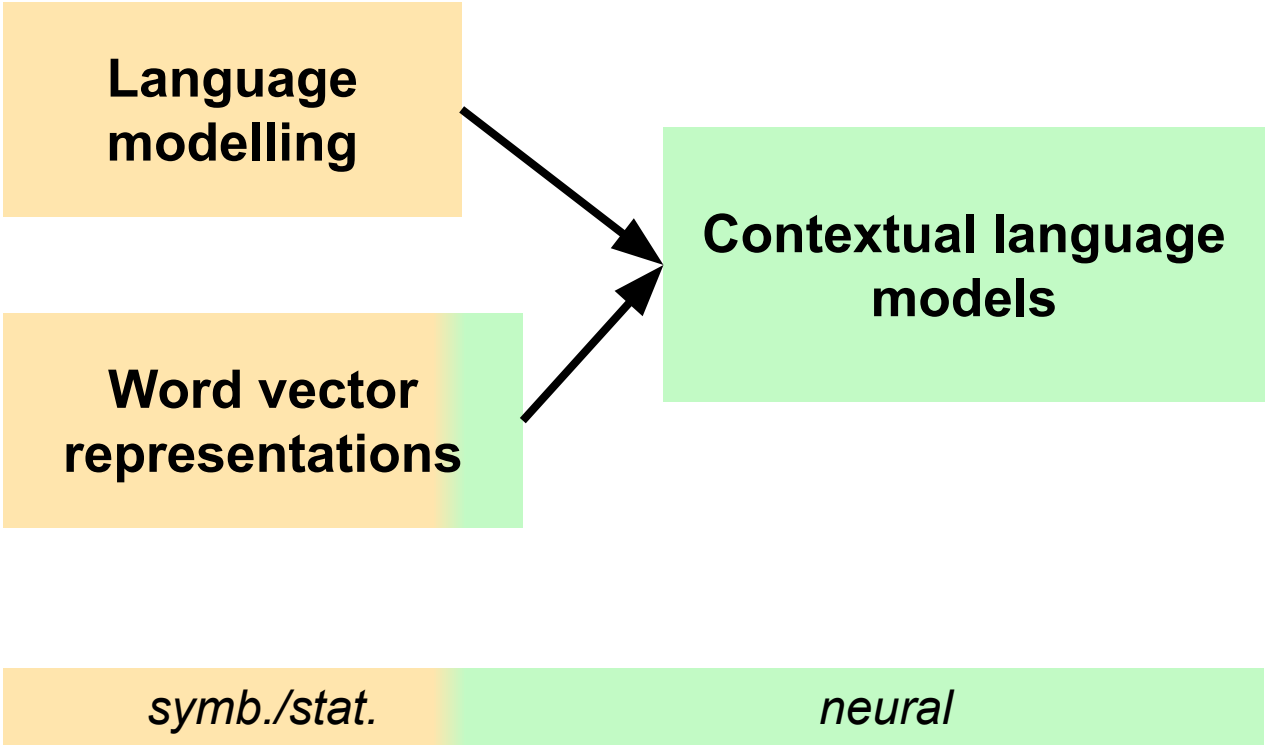
- A **named entity** is a real-world object that can be denoted individually
  - Standard named entities: people, locations, organisations
  - Extended named entities: dates, addresses, URLs, e-mail addresses, numbers, etc.
- A named entity **mention** is an utterance denoting a named entity
  - Examples: Emmanuel Macron, Los Angeles, Apple Inc.
  - This denotation can be ambiguous…
- Named entity mentions have specific properties, apart from their specific, individual denotation:
  - Specific internal structure (**local grammar**), often culture-dependent more than language-dependent
    > **They can be viewed as atomic for the grammar of the language**, and therefore as **special forms**
- Ex.: *all_of_a_sudden , he started playing table tennis with John_Doe .*

Semantic words

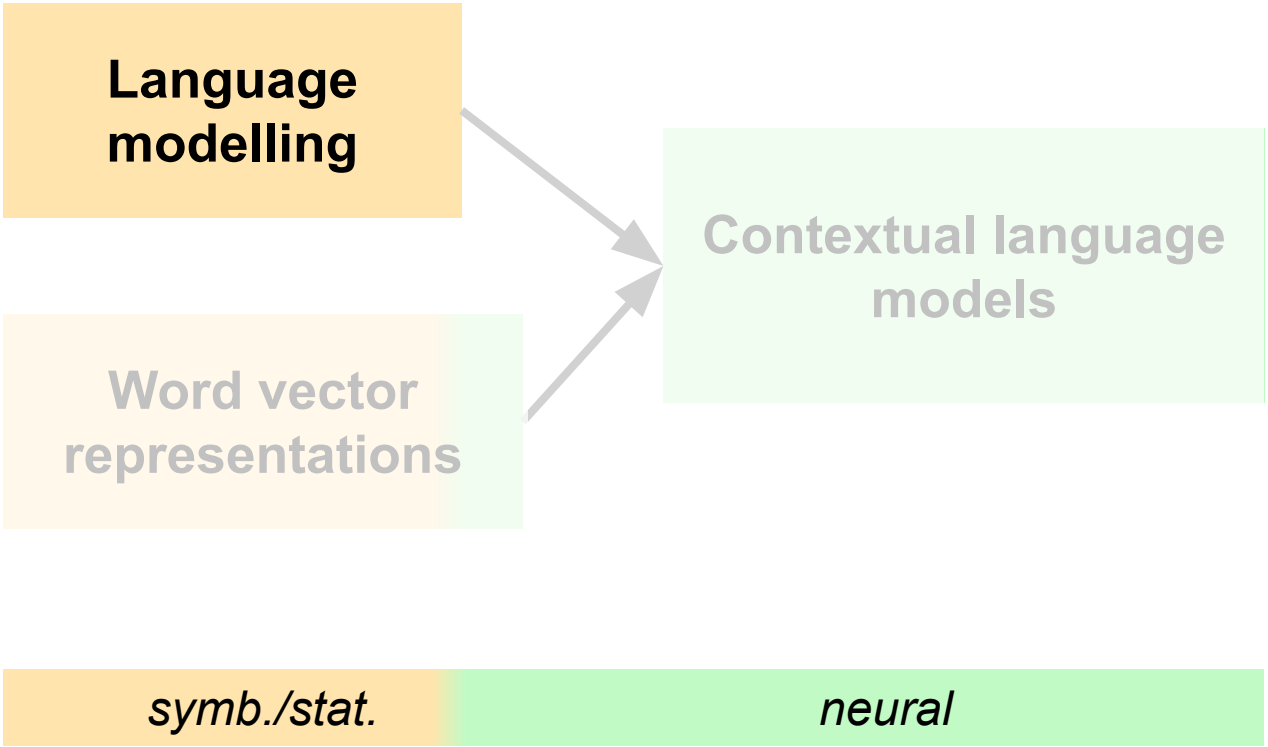
- A **semantic word** is a (sequence of) form(s) with non-compositional meaning
  - I.e. its meaning cannot be inferred from the meaning of its constitutive forms
    Ex.: *pomme de terre*, *red herring*
  - Often ambiguous!
    Ex.: *Il a sorti la pomme de terre / Il a modelé une pomme de terre cuite*
    W. C. wrote how he used red herring to lay a false trail, while training hunting dogs
  - Close but distinct from the notion of **term** (a conventional unit)
    Ex.: *machine à laver* (not **appareil à nettoyer* !)
- Ex.: *all_of_a_sudden , he started playing table_tennis with John_Doe .*

Language modelling
vs.
Word Vector Representations

Language modelling in NLP



Language modelling in NLP



What is a language model?

- **A language model is a probability distribution over word sequences**
  - It simply tells you that "*This is a meaningful sentence*" is more probable than "a *This llkdhfeee meaningful is is*"
- A language model can be used to
  - **Compare the probabilities of two word sequences** (e.g. multiple machine translation candidates)
  - Find the **most probable next word** following an input sequence
  - Find the **most probable word to fill a gap** in a sequence

p_{LM}("I like donuts") = 0.03874

p_{LM}("The green dog eats the moon") = 0.00648

p_{LM}("I donuts likes ") = 0.000035413

p_{LM}("J'aime les donuts ") = 0.0

p_{LM}("a6ca61 cvnon a58 ") = 0.0

Statistical (*n*-gram) language models

- Unigram models (non contextual) : $P_1(w_1w_2w_3) = P(w_1)P(w_2)P(w_3)$
- Bigram models (Markov) : $P_2(w_1w_2w_3) = P(w_1|\#)P(w_2|w_1)P(w_3|w_2)$
- *n*-gram models (Markov) : $P_n(w_k|\#w_1\ldots w_{k-2}w_{k-1}) = \ldots P(w_k|w_{k-n+1}\ldots w_{k-1})$
- Bidirectional bigram models:

$$P_{2,BD}(w_1w_2w_3) = P(w_1|\#)P(w_2|w_1)P(w_3|w_2)P'(w_3|\#)P'(w_2|w_3)P'(w_1|w_2)$$

- Probabilities can be computed in a number of ways:
  - The most common one is simple counting:

$$P(w_k|w_1\ldots w_{k-1}) = occ(w_1\ldots w_k) / occ(w_1\ldots w_{k-1})$$

What if an *n*-gram has never been seen?

Most of *n*-grams are unobserved:

- Most word sequences get 0 probability
- Problem grows exponentially with *n*
- Solutions:
  - Smoothing (e.g. Laplace)
  - Backoff, interpolation, clustering

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &\quad + \lambda_2 P(w_n|w_{n-1}) \\ &\quad + \lambda_3 P(w_n)\end{aligned}$$

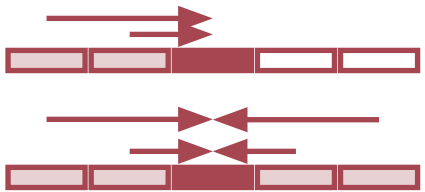
Advantages and drawbacks of *n*-gram language models

- Advantages
  - Very fast to train
  - Interpretable
- Drawbacks
  - Local context is not always enough

    Alice went to the vet to pick up [her/his] dog.
  - Increasing the size of the context is problematic (number of parameters, sparsity)

Predictive language models

- **Predicting** rather than **counting**
  - Given a position and a context, a model (e.g. neural) can be trained to predict the word that would best fit in this position
  - The probability of a word in a certain context is the probability associated to it by such a model
  - Two main types of context:
    - The left context
    - The full context



Evaluating Language Models: Perplexity

- The perplexity of a language model on a text, W, is the **inverted normalized probability** of predicting that text.

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

- Intuitively: How often do you need to guess a word before you make the correct prediction (ex: if *P* is a random, *PP*(*W*) = *N*)
- It is 2 to the power of the empirical cross entropy computed over a whole text

  => the perplexity heavily depends on the language model training data/loss:

  - Very sensitive to domain shift
  - Dependent on vocabulary size, tokenization, etc…

Evaluating Language Models: Perplexity

$$PP_\theta(W) = P_\theta(w_0 ... w_n)^{-\frac{1}{n}}$$

Bayes formula

$$= \prod_0^n P_\theta(w_i|w_{<i})^{-\frac{1}{n}}$$

$$= 2^{log_2(\prod_0^n P_\theta(w_i|w_{<i})^{-\frac{1}{n}})}$$

Empirical cross entropy of P or empirical mean of log(P) (**2 to the power of your training loss!**)

$$= 2^{-\frac{1}{n}\sum_0^n log_2(P_\theta(w_i|w_{<i}))}$$

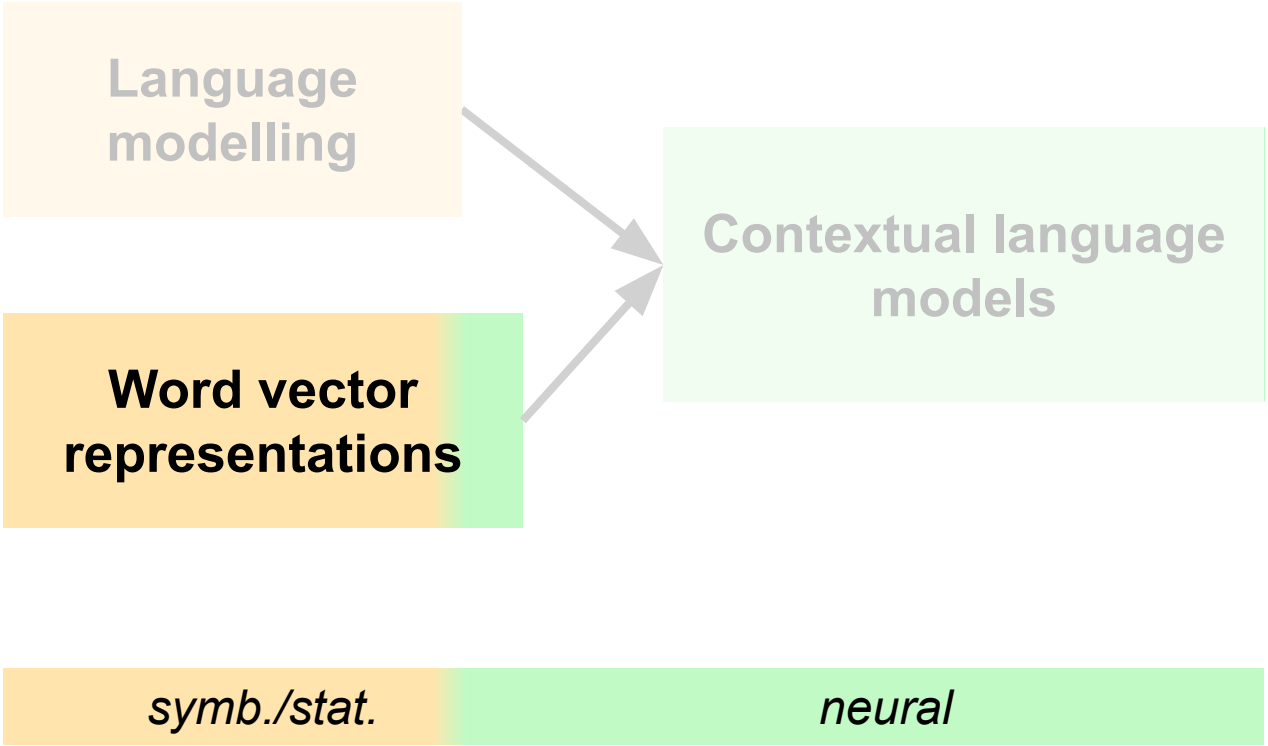
$$\rightarrow 2^{E_{P_{\theta*}}[log_2(P_\theta)]}$$

Weak law of large number: the empirical mean converges to the true mean

$$= 2^{CE(P_{\theta*}, P_\theta)}$$

Cross entropy

Language modelling in NLP



Word vector representations

- Each word, in each context it appears in, conveys a number of features
  - E.g. morphological, syntactic, semantic, etc.
- These features can be represented in 2 main ways:
  - Explicitly: *cat=[animal=+, human=-, size=average…]*
    - Cf. lexicons/dictionaries, including lexical resources widely used in NLP
  - Implicitly, using vector representations
    - Such approaches were notably put forward in early neural approaches to NLP (Rumelart, Hinton & Williams 1986)
    - A neural network is a specification of how to computationally transform an input into an output using operations on numbers

One-hot word vector representations?

- The simplest word representation consists in choosing a vocabulary of size *n* and represent the *i*-th word of the vocabulary as a vector of zeros except for the *i*-th coordinate, which is set to 1
  - **“1-hot” representation**
  - Main drawback: two words with similar features and behaviours have nothing more in common than completely different words
- If **two similar words could have similar representations**, models using such representations could be able to generalise

The distributional hypothesis for word vector representations

- Distributional hypothesis: **Two words are similar if the occur in the same (or similar) contexts**
    - Harris (1954) :
        - *oculist and eye-doctor … occur in almost the same environments*
        - *If A and B have almost identical environments we say that they are synonyms.*
    - Firth (1957) : *You shall know a word by the company it keeps!*
    - Example (modified by Lin (1998) from (Nida, 1975, page 167)): Suppose I asked you what is tesgüino?

      *A bottle of **tesgüino** is on the table*
      *Everybody likes **tesgüino***
      ***Tesgüino** makes you drunk*
      *We make **tesgüino** out of corn.*

⇒ We need context-based representations that go beyond 1-hot vectors

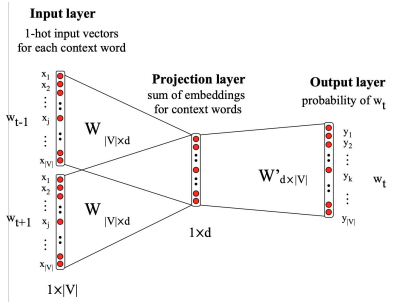
Non contextual embeddings

- Nowadays, word vector representations are generally called **word embeddings**
- First approach: given a word, associate it with a representation based on *all* its occurrences in a large corpus
  - Contextual information for each occurrence is replaced by 1-hot vectors
  - **Counting-based** approaches (raw, positive point-wise mutual information = PPMI) followed by dimensionality reduction to get dense vectors (SVD, LSA, LDA, etc.)
  - Predictive approaches based on a neural model
    Examples: *word2vec* (Mikolov et al. 2013),
    FastText (Bojanovski et al. 2017)

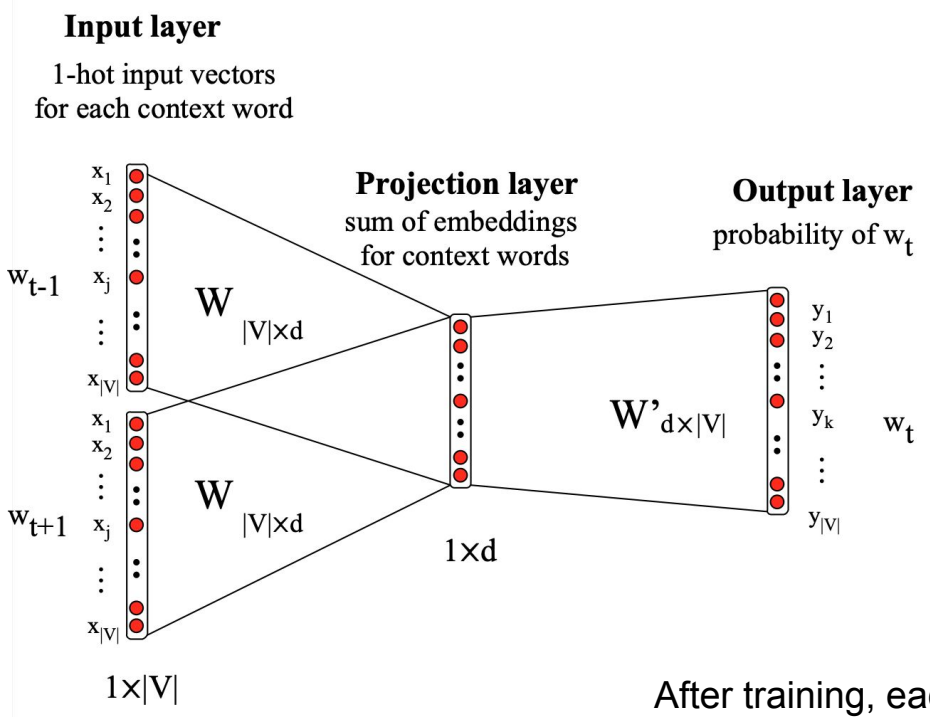
	aardvark	computer	data	pinch	result	sugar	…	…
apricot	0	0	0	1	0	1		
pineapple	0	0	0	1	0	1		
digital	0	2	1	0	1	0		
information	0	1	6	0	4	0		

Non contextual embeddings

- Nowadays, word vector representations are generally called **word embeddings**
- First approach: given a word, associate it with a representation based on *all* its occurrences in a large corpus
  - Contextual information for each occurrence is replaced by 1-hot vectors
  - Counting-based approaches (raw, positive point-wise mutual information = PPMI) followed by dimensionality reduction to get dense vectors (SVD, LSA, LDA, etc.)
  - **Predictive approaches** based on a **neural** model
    Examples: *word2vec* (Mikolov et al. 2013),
    FastText (Bojanovski et al. 2017)



Non contextual embeddings: Word2Vec



\+ Gradient descent and back-propagation

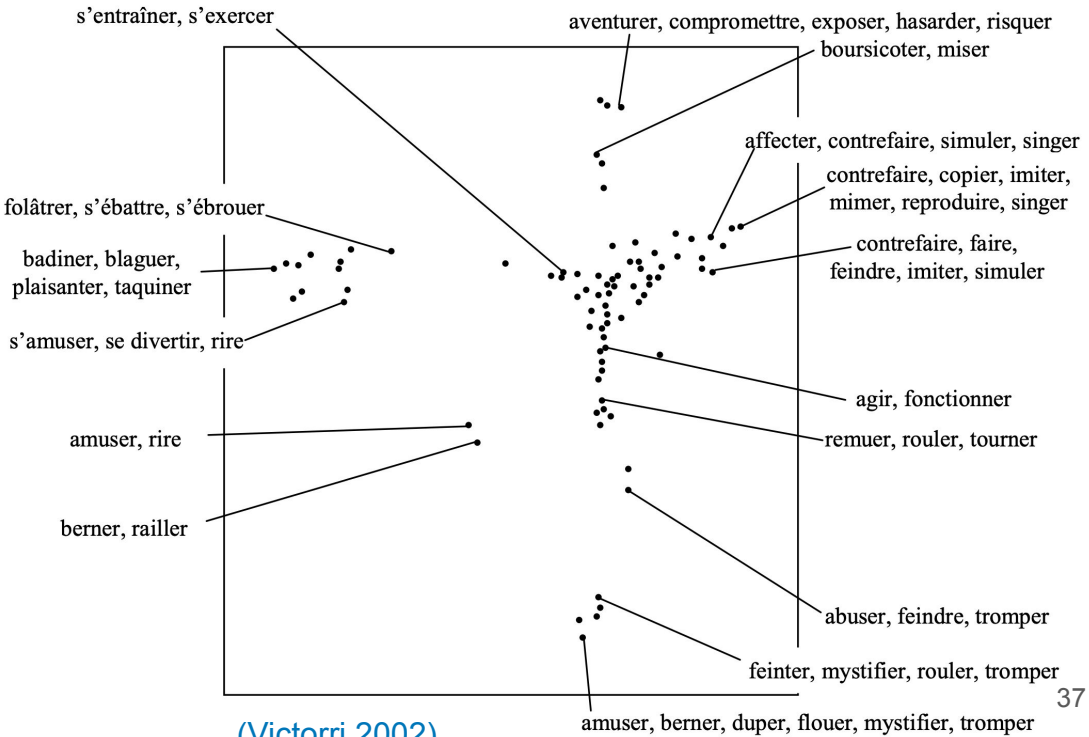
After training, each column in **W** is a word embedding

Non contextual embeddings

- The main limitation of these approaches is that they build **non contextual embeddings**:
  - e.g the word “*bank*” is represented by the same vector in all its occurrences/contexts/meanings

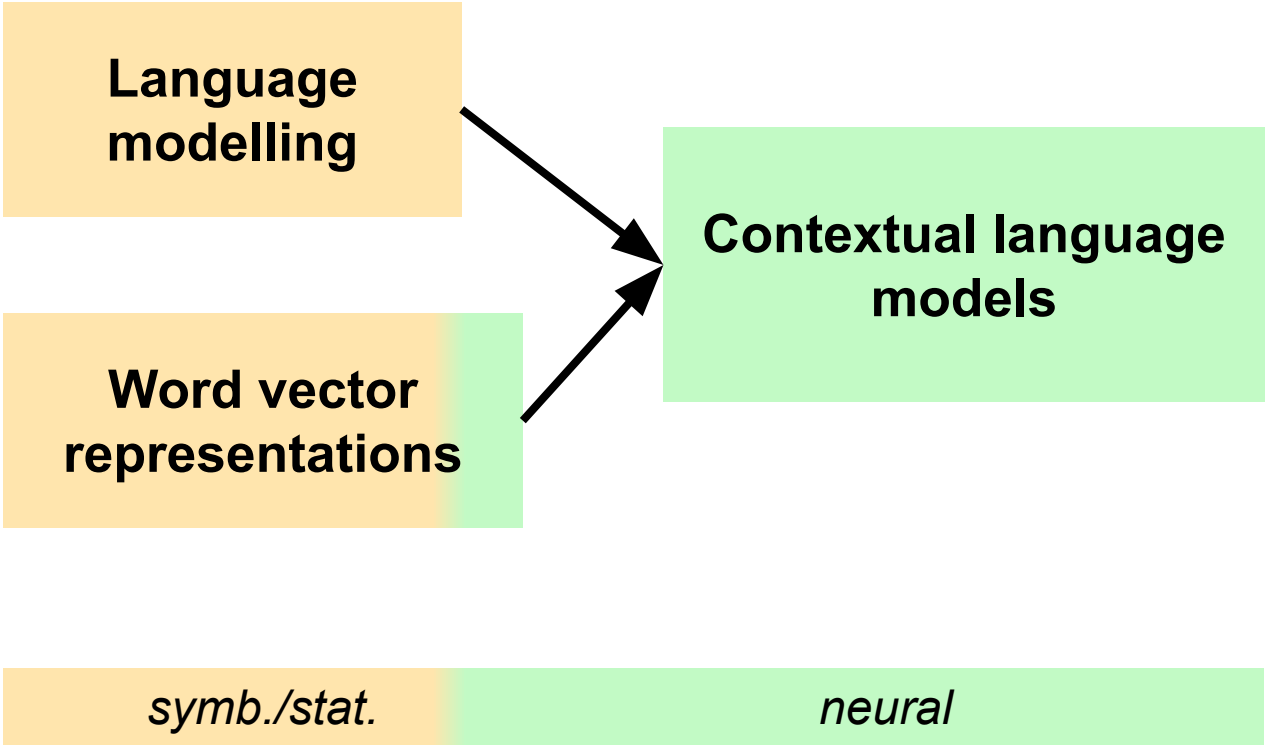
Early contextual word vector representations

- Work on the vector representation of word occurrences, i.e. on contextual embeddings, pre-date the spread of neural approaches in NLP
  - Example: work on the representation of lexical semantics in continuous spaces (Fuchs and Victorri 1996, Victorri 2002)
  - The goal was to analyse the lexical semantic space at the lemma level, not to produce representations to be used as such



(Victorri 2002)

Language modelling in NLP



Word embeddings meet language models

- Our ideal vector representations:
  - Take each occurrence's context into account
  - Similar occurrences of similar words, i.e. similar (word, context) pairs, should be represented by similar vectors, to maximise generalisation
- This can be achieved using… a language model!
  - Representations produced by neural predictive language models (e.g. on the last layer) can be seen as contextual word embeddings
  - Both the neural architecture, the training task(s) and the loss function must be chosen carefully

Word embeddings meet language models

- A **generative language model** is mostly used for what it was trained for: generating the next word given an input (the prompt) — often iteratively
  - Some tasks can be modelled as text generation tasks via prompting
  - E.g. by giving a prompt such as "*Translate the following into English: <sentence>*"

GPT, GPT-2,3

generative

Word embeddings meet language models

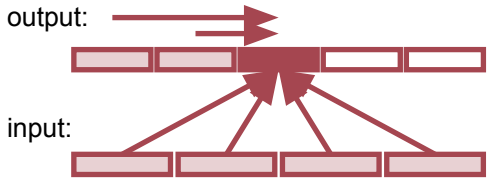
- Language models based on a **masked language modelling (MLM)** architecture (e.g. BERT) produce better representations — they see the full context, not only the left context
  - The language model can be used to do what it was trained for: **filling a gap**
  - The representations they produce can be used as contextual embeddings: this is the **embedding provider scenario**
  - A language model can also be used to initialise part of a task-specific architecture, part of which is isomorphic to the language model architecture: this is the **fine-tuning** scenario, i.e. an instance of transfer learning

ELMo, BERT, RoBERTa

MLM

Word embeddings meet language models

- A **sequence-to-sequence language models** is a generative language model conditioned on a given input
  - The input can be a “noised” version of the expected output (e.g. masked, transformed, written in another language, and more)
  - Such models are **generally fine-tuned** to perform conditioned text generation tasks, including but not limited to text transformation tasks. Examples:
    - Text transformation: machine translation, text simplification, etc.
    - Other conditioned text generation tasks: question answering, general chatbot



BART, T5

seq2seq

Neural Language Modelling Architectures

Recurrent Neural Networks for Generative LM

Recurrent Neural Networks for Generative LM

Recurrent Neural Networks

RNN generative

- Parameterize next word probability with neural network

$$p_{LM}(w_n \mid w_{n-1}, w_{n-2}, \ldots, w_1) \propto f(w_n, w_{n-1}, w_{n-2}, \ldots, w_1)$$

Recurrent Neural Networks

RNN | generative

- Parameterize next word probability with neural network

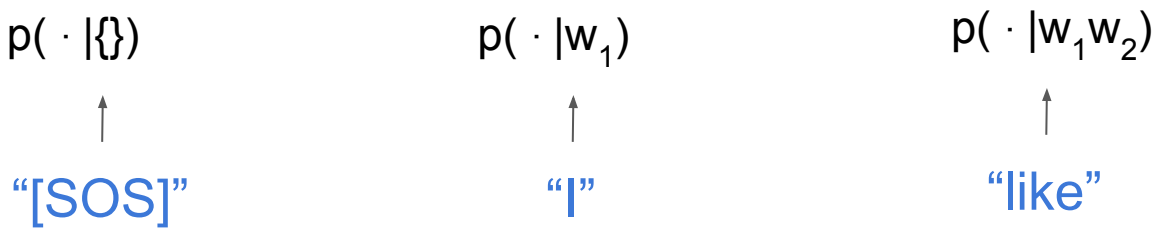
$$p_{LM}(w_n \mid w_{n-1}, w_{n-2}, \ldots, w_1) \propto f(w_n, w_{n-1}, w_{n-2}, \ldots, w_1)$$

$$\log p_{LM}(w_n \mid w_{n-1}, w_{n-2}, \ldots, w_1) \propto \underbrace{e[w_n]^T}_{\text{Next word embedding}} \underbrace{h(w_{n-1}, w_{n-2}, \ldots, w_1)}_{\text{Context encoder (neural network)}}$$

Recurrent Neural Networks

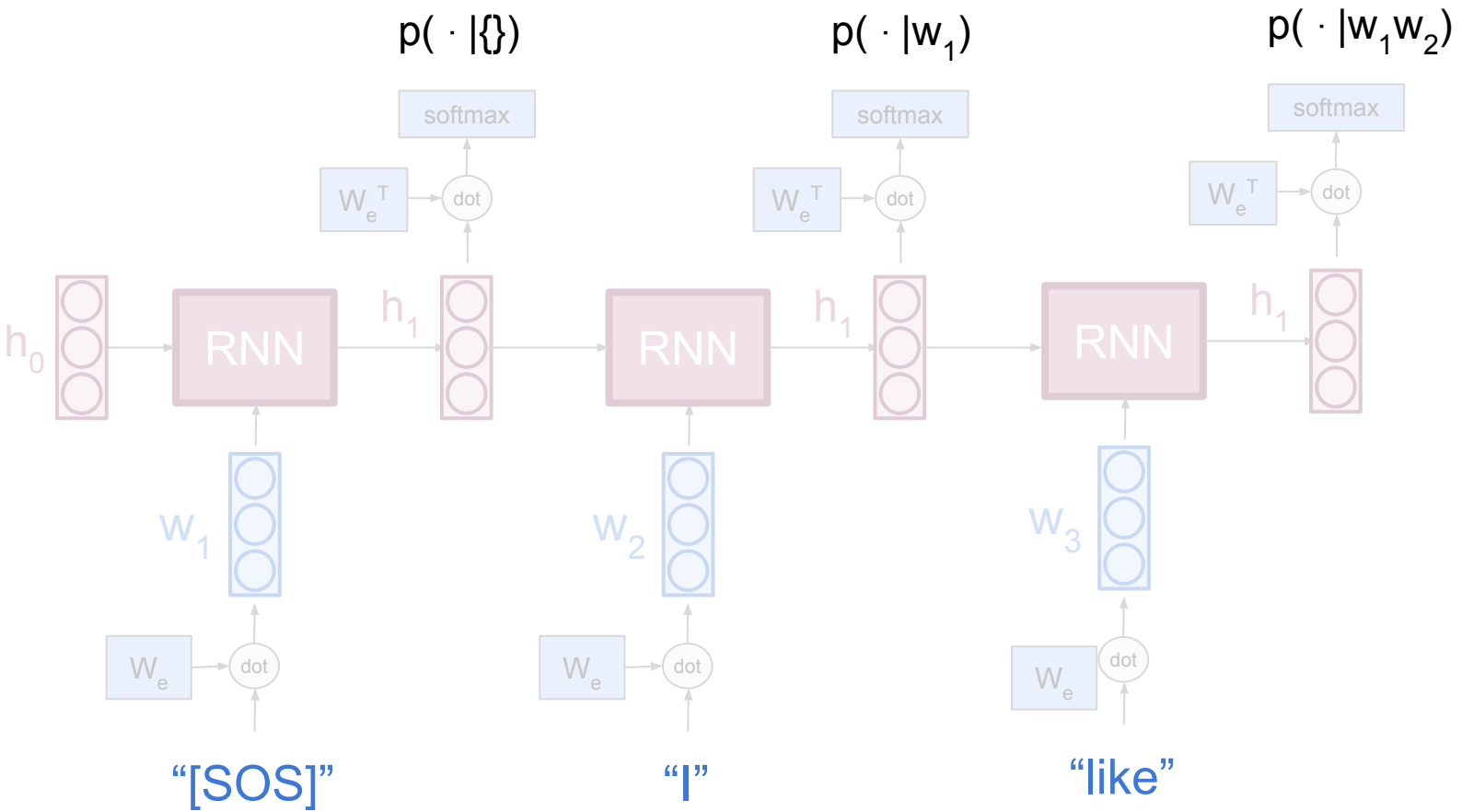
RNN | generative

For each input token, a probability distribution over the whole vocabulary!



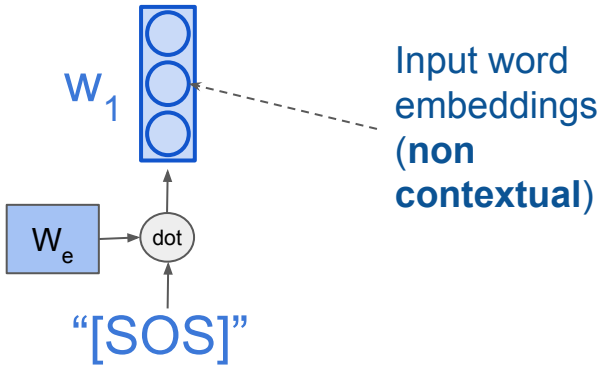
Recurrent Neural Networks

RNN | generative



Recurrent Neural Networks

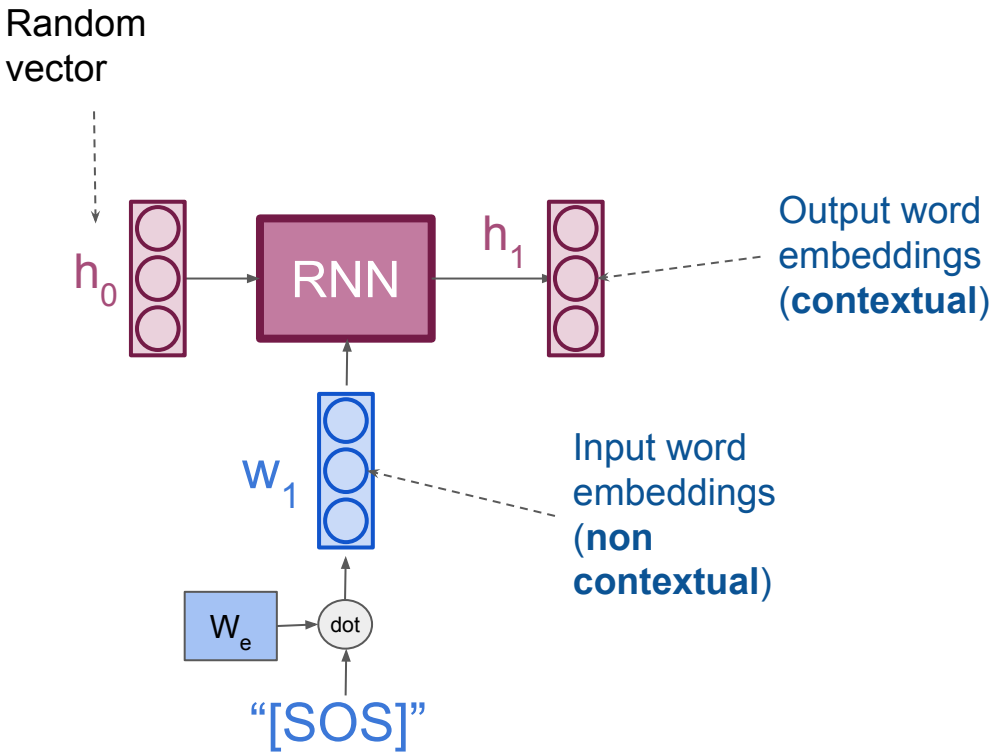
RNN | generative



W_e is a matrix of **BIG** random non contextual word embeddings (50k vectors) !

Recurrent Neural Networks

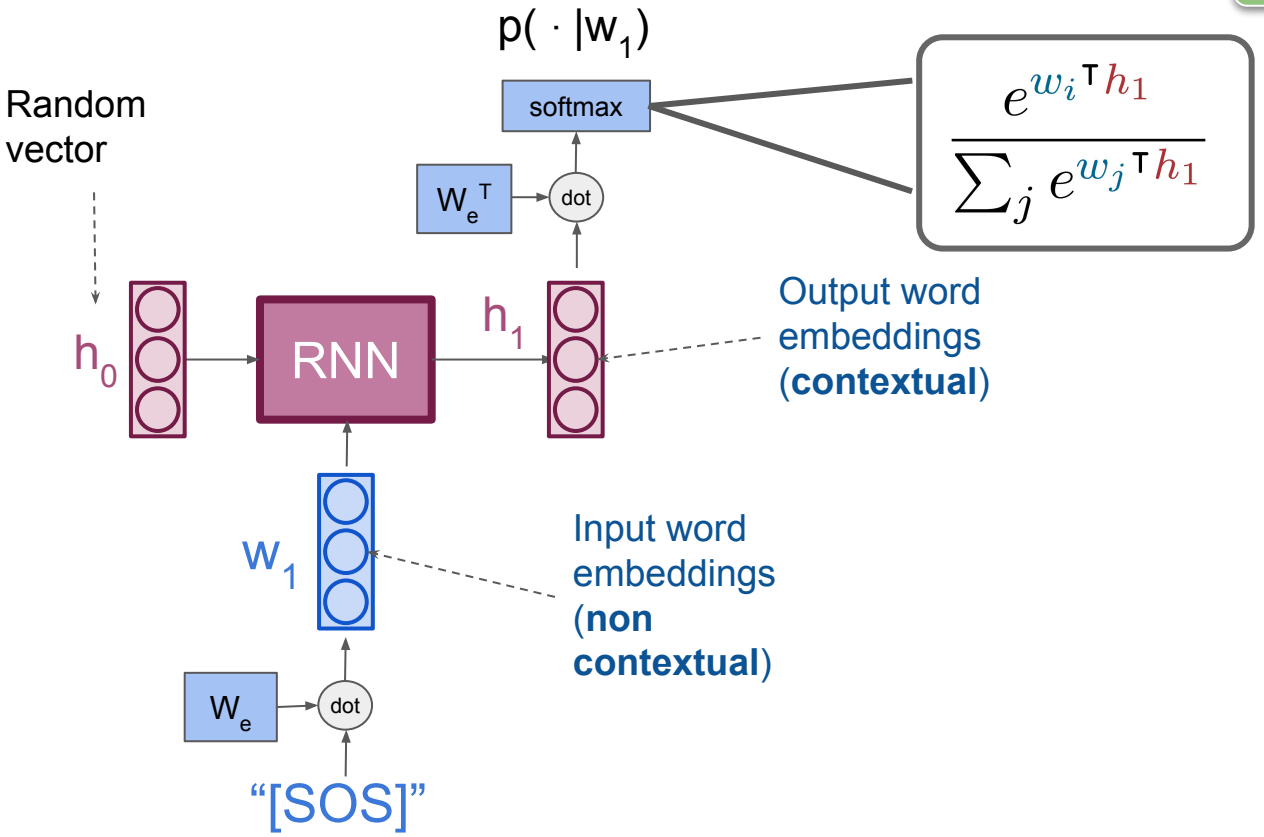
RNN | generative



W_e is a matrix of **BIG** random non contextual word embeddings (50k vectors) !

Recurrent Neural Networks

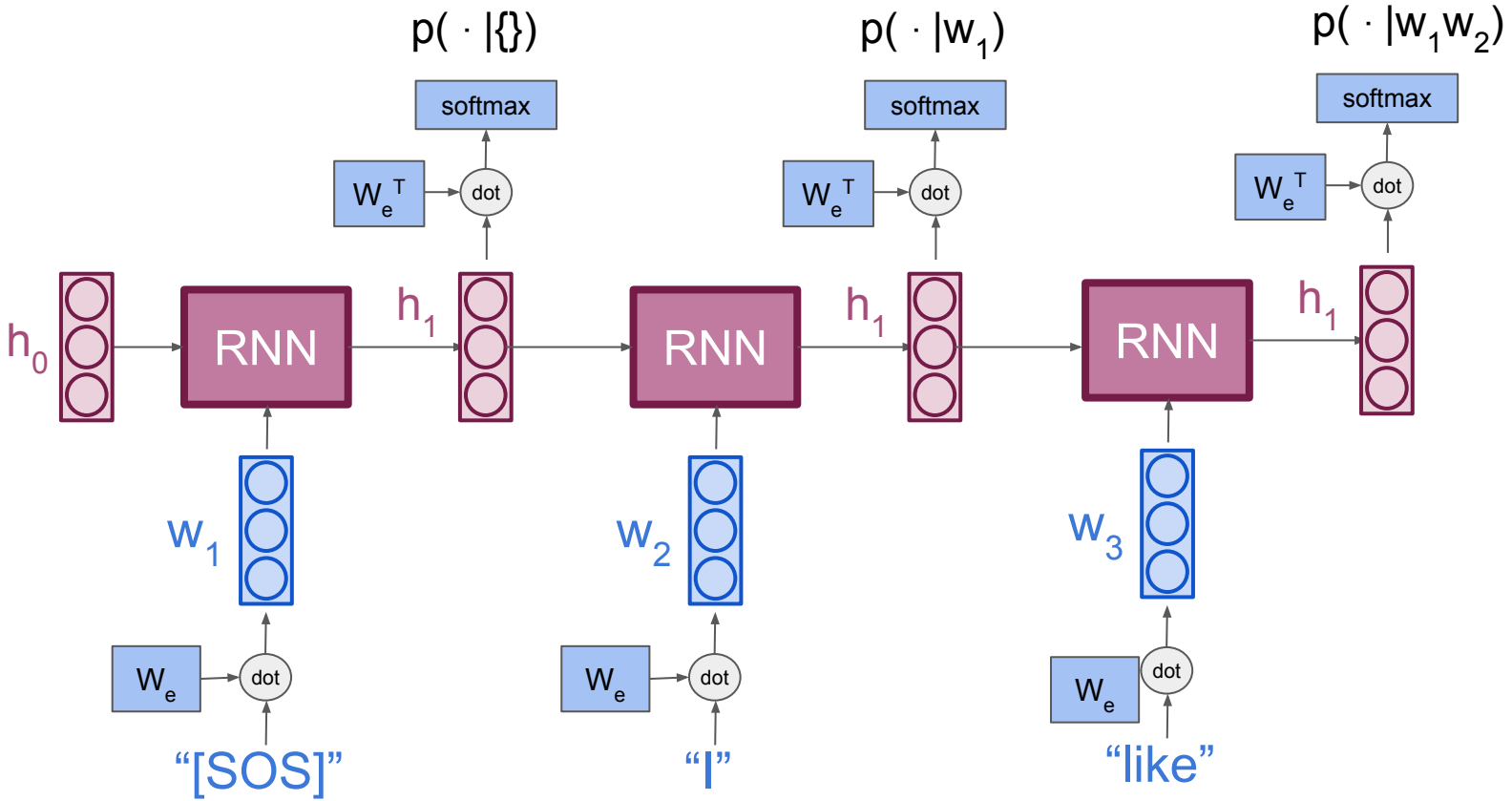
RNN | generative



Probability distribution over the whole vocabulary

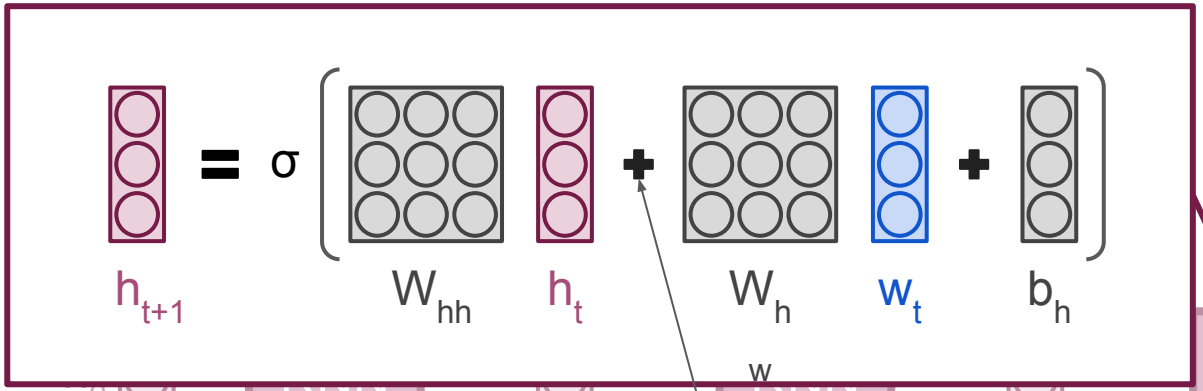
Recurrent Neural Networks

RNN | generative



Recurrent Neural Networks

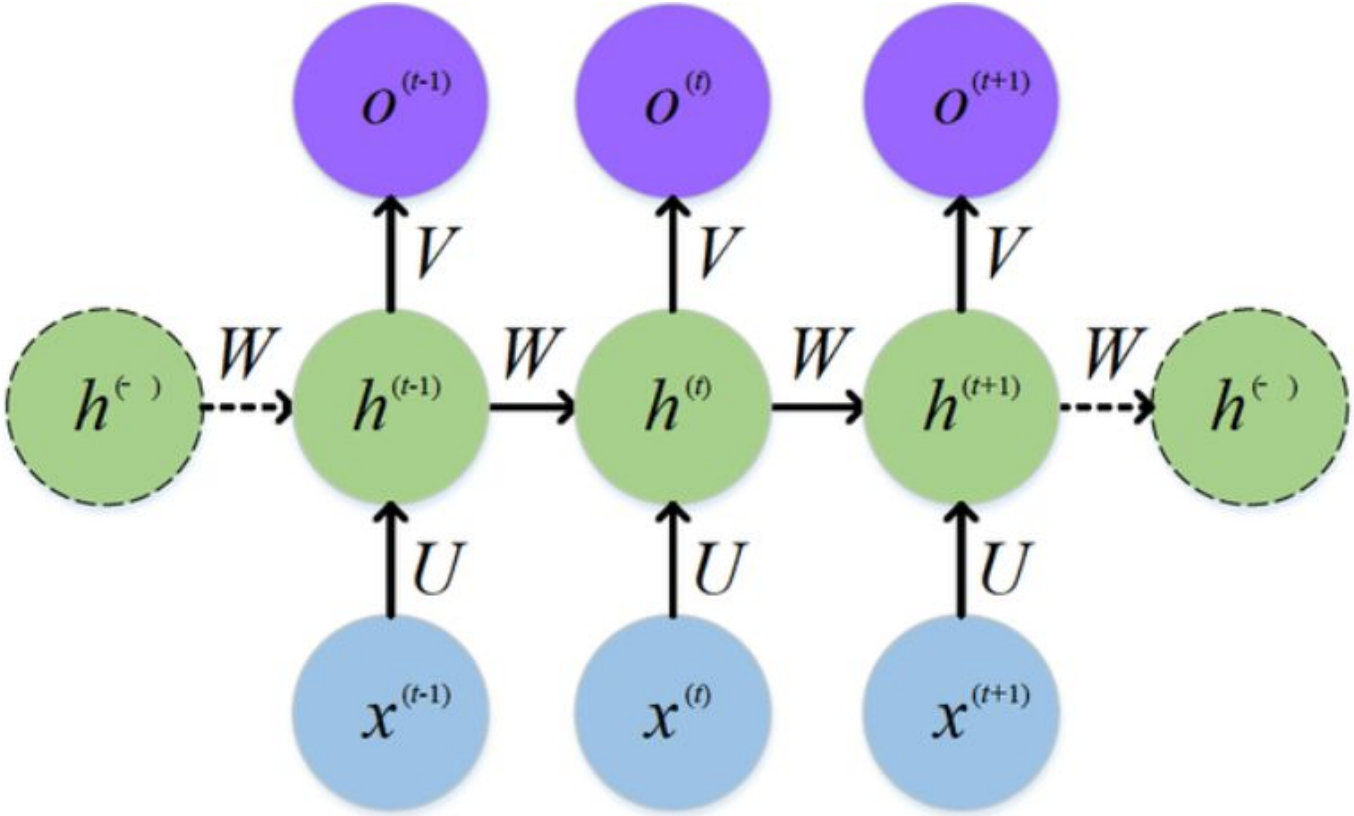
RNN | generative



indeed , it is an addition… a nonsense for feature-based machine learning…

This RNN is a total of **2 FCs + 1 embedding layer** (at least, there are plenty of definitions…)

Another one (3 FC+1emb layer)



Another one: minimal RNN (1 FC+1emb layer)

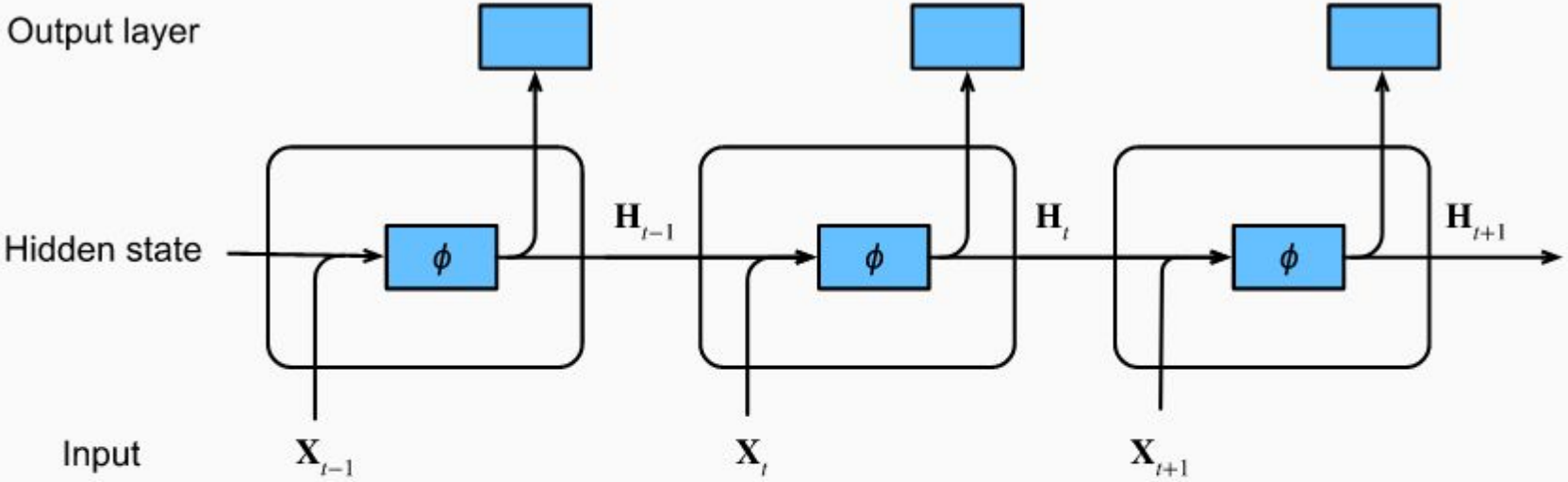
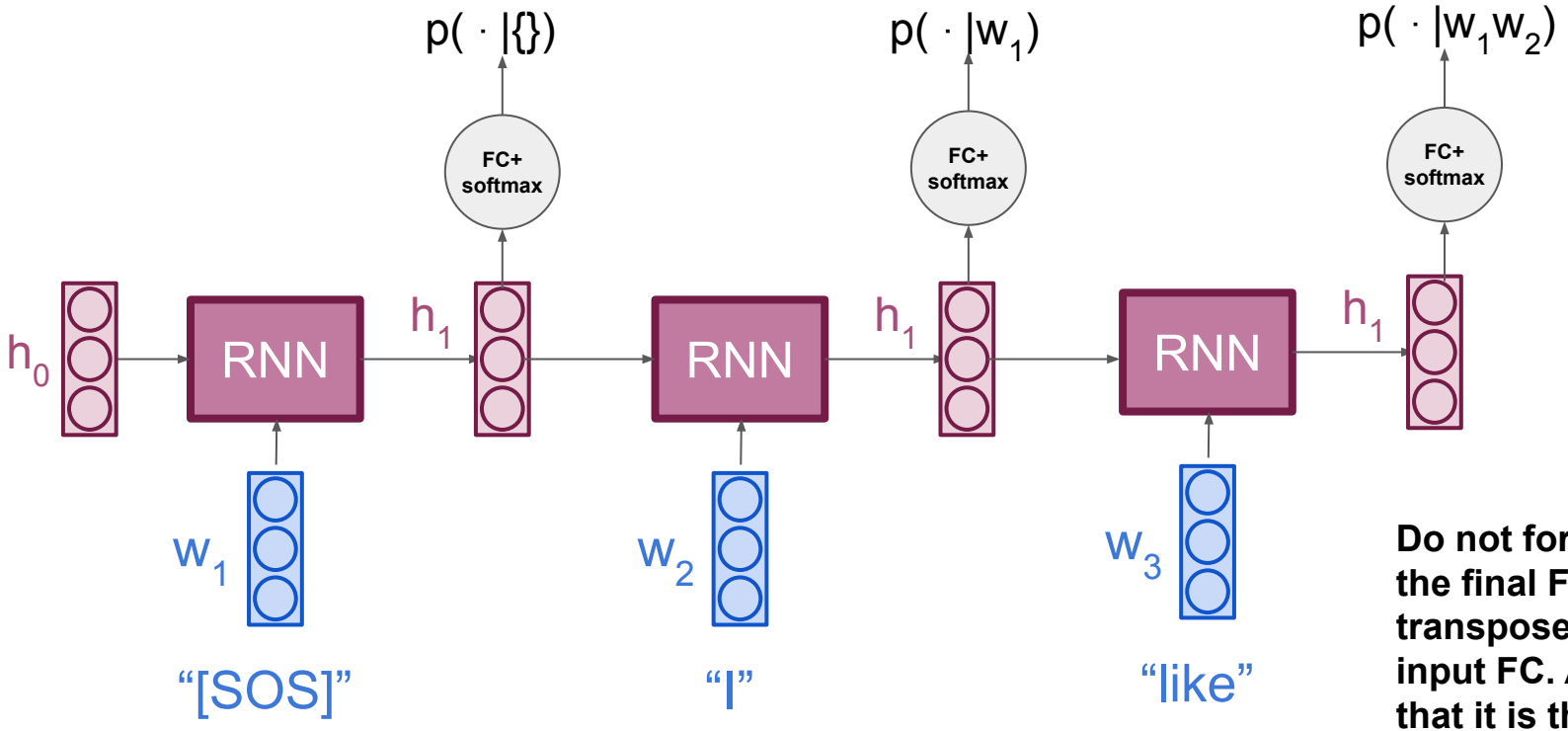


Fig. 9.4.1 An RNN with a hidden state.

RNN (compact representation)

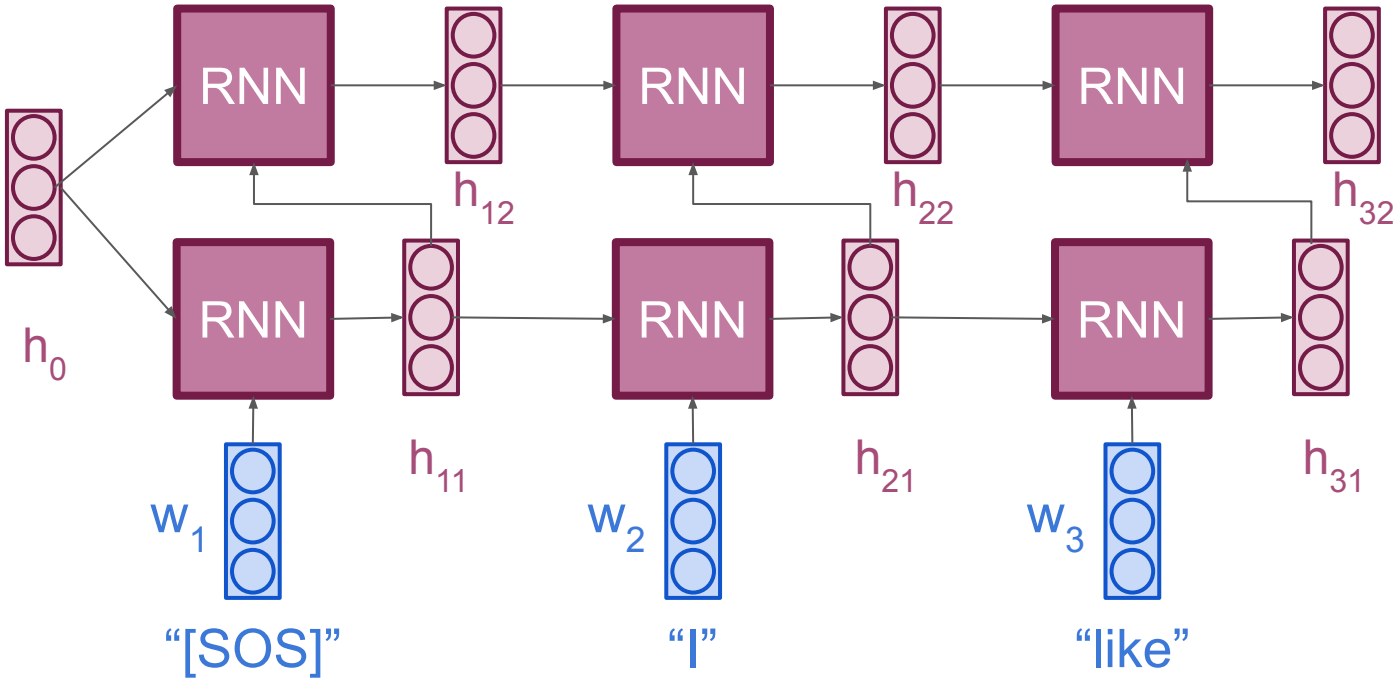
RNN | generative



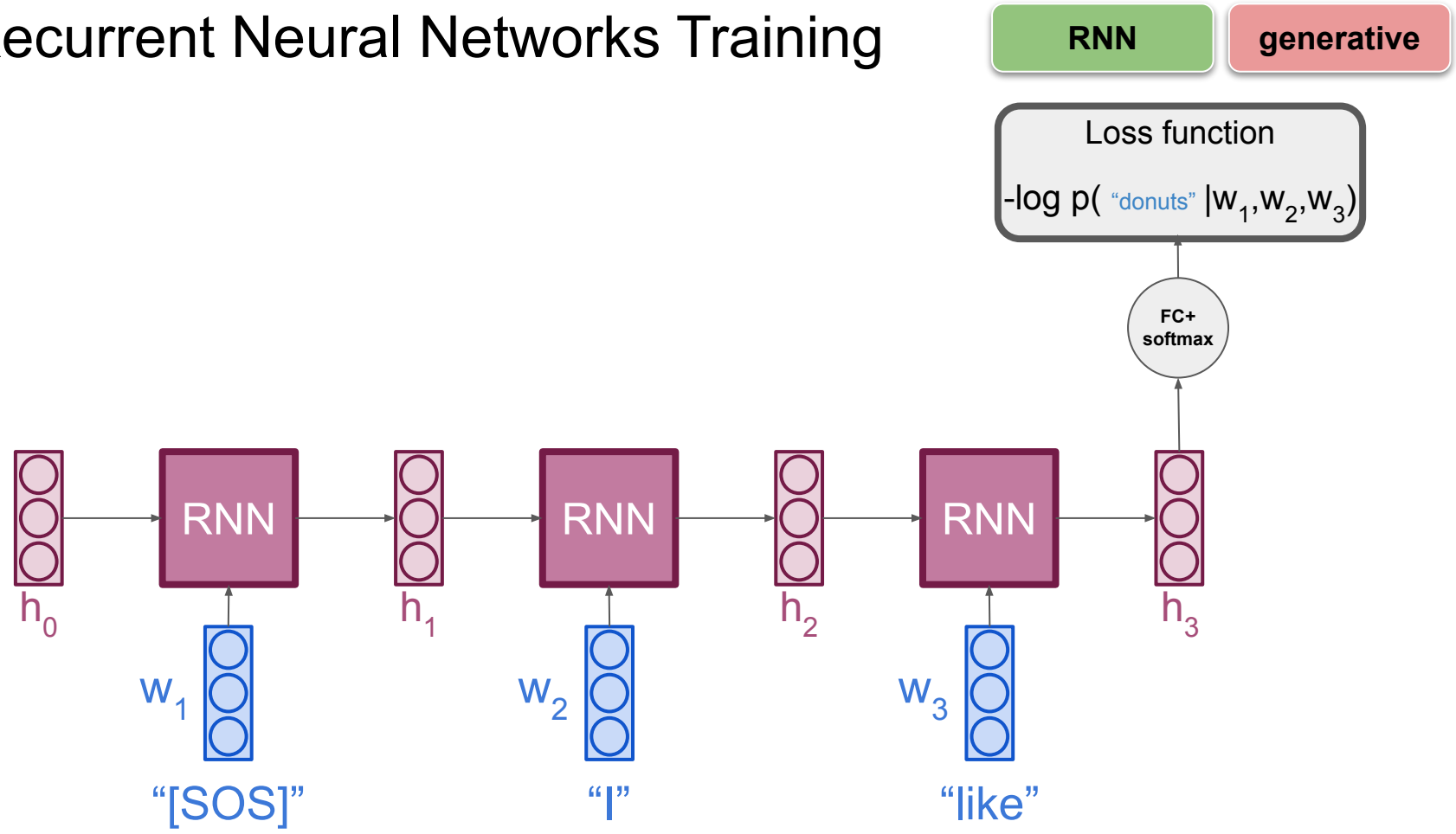
Do not forget that the final FC is the transpose of the input FC. And that it is the biggest layer.

RNN generative

More layers ?

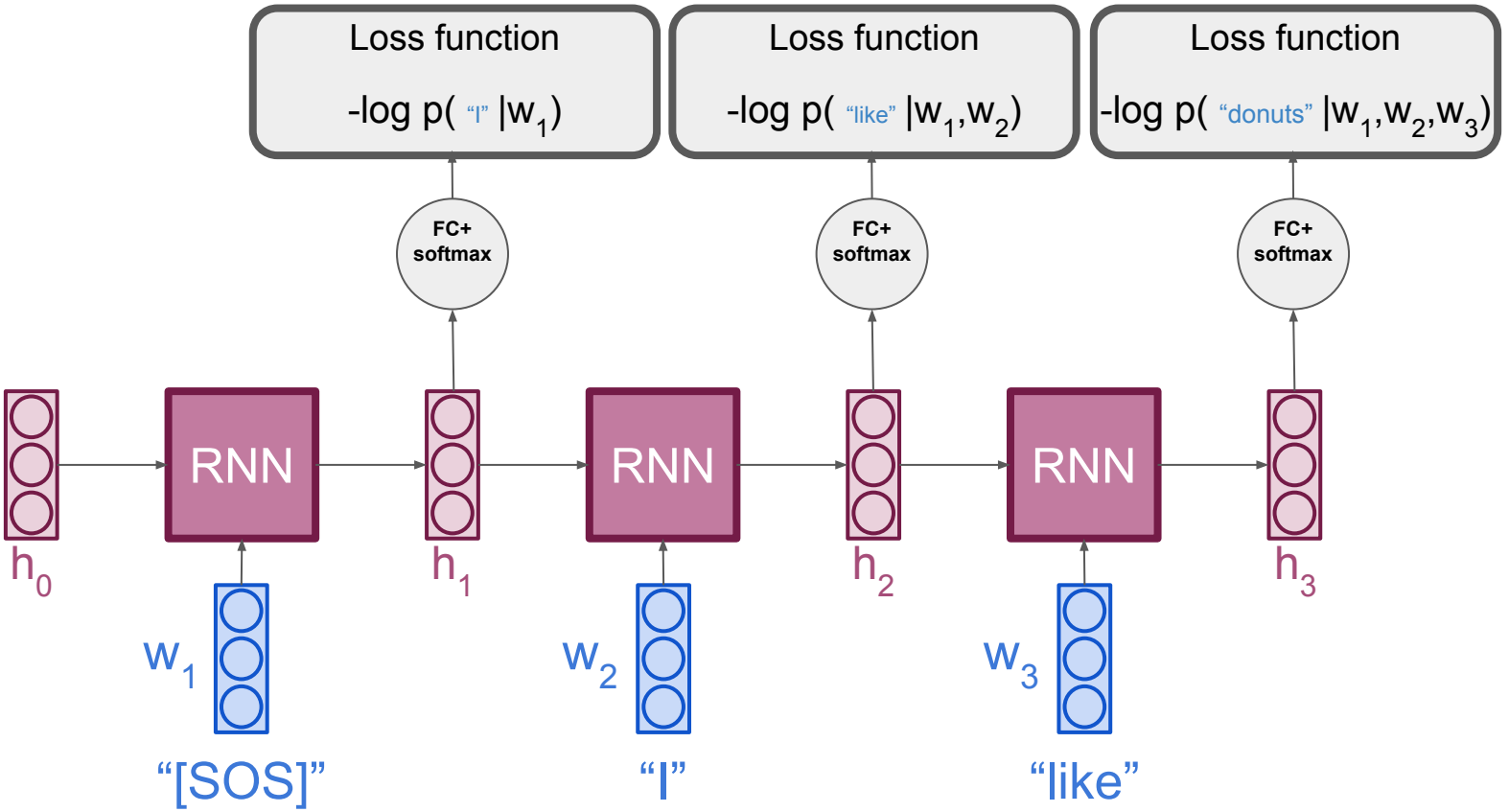


Recurrent Neural Networks Training



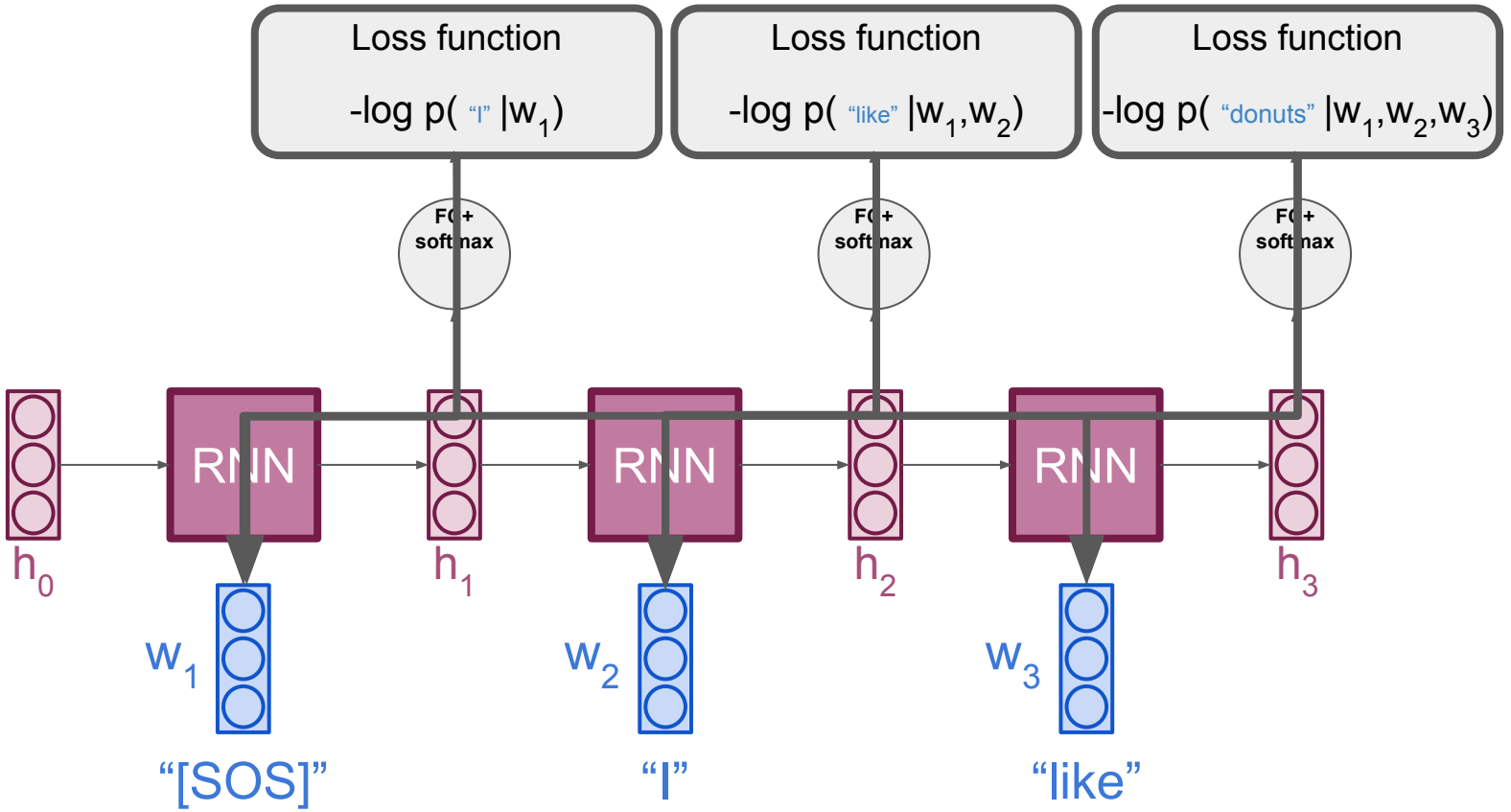
Recurrent Neural Networks Training

RNN | generative



Backpropagation through time (BPTT)

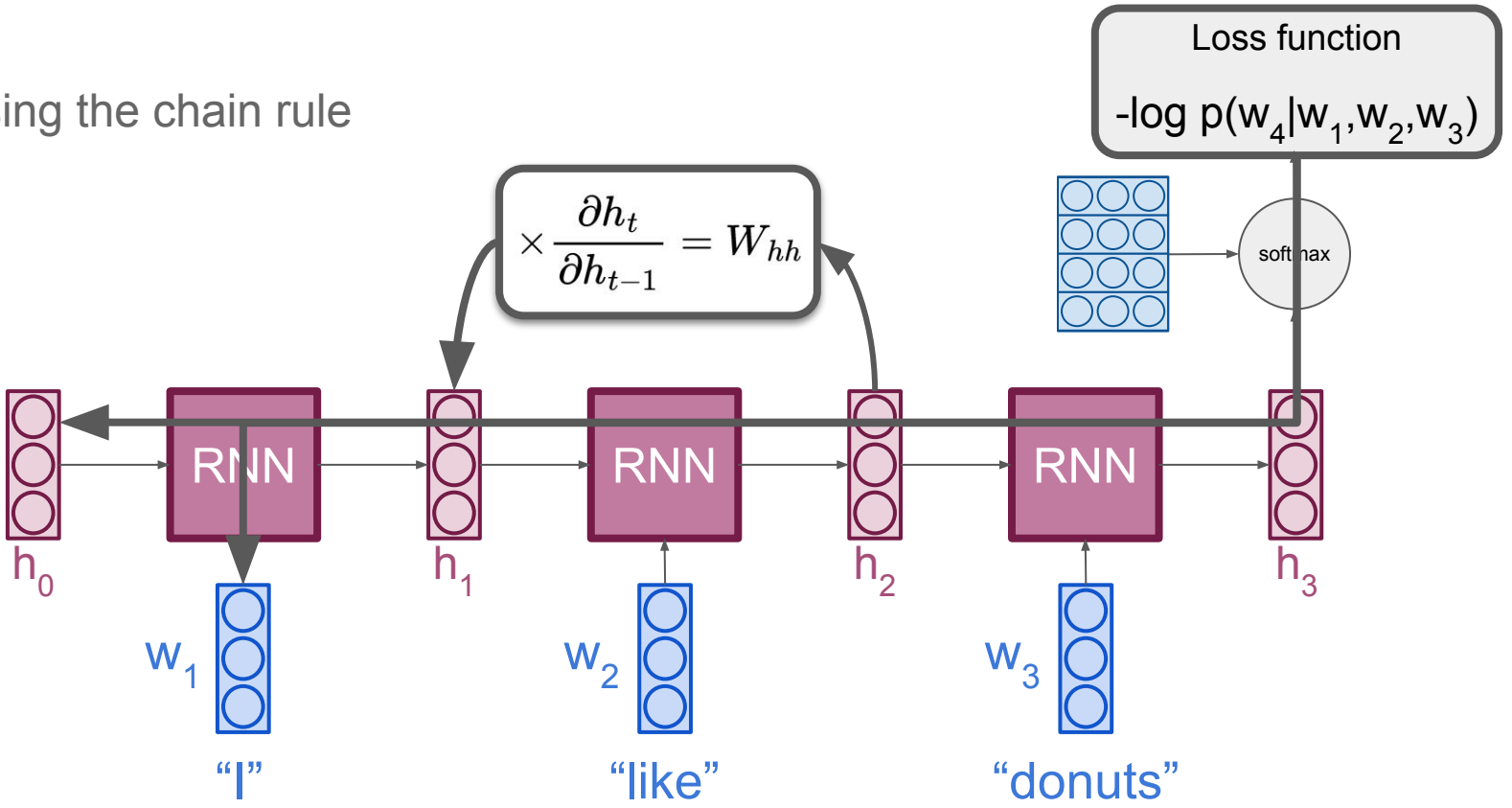
RNN | generative



BPTT

RNN generative

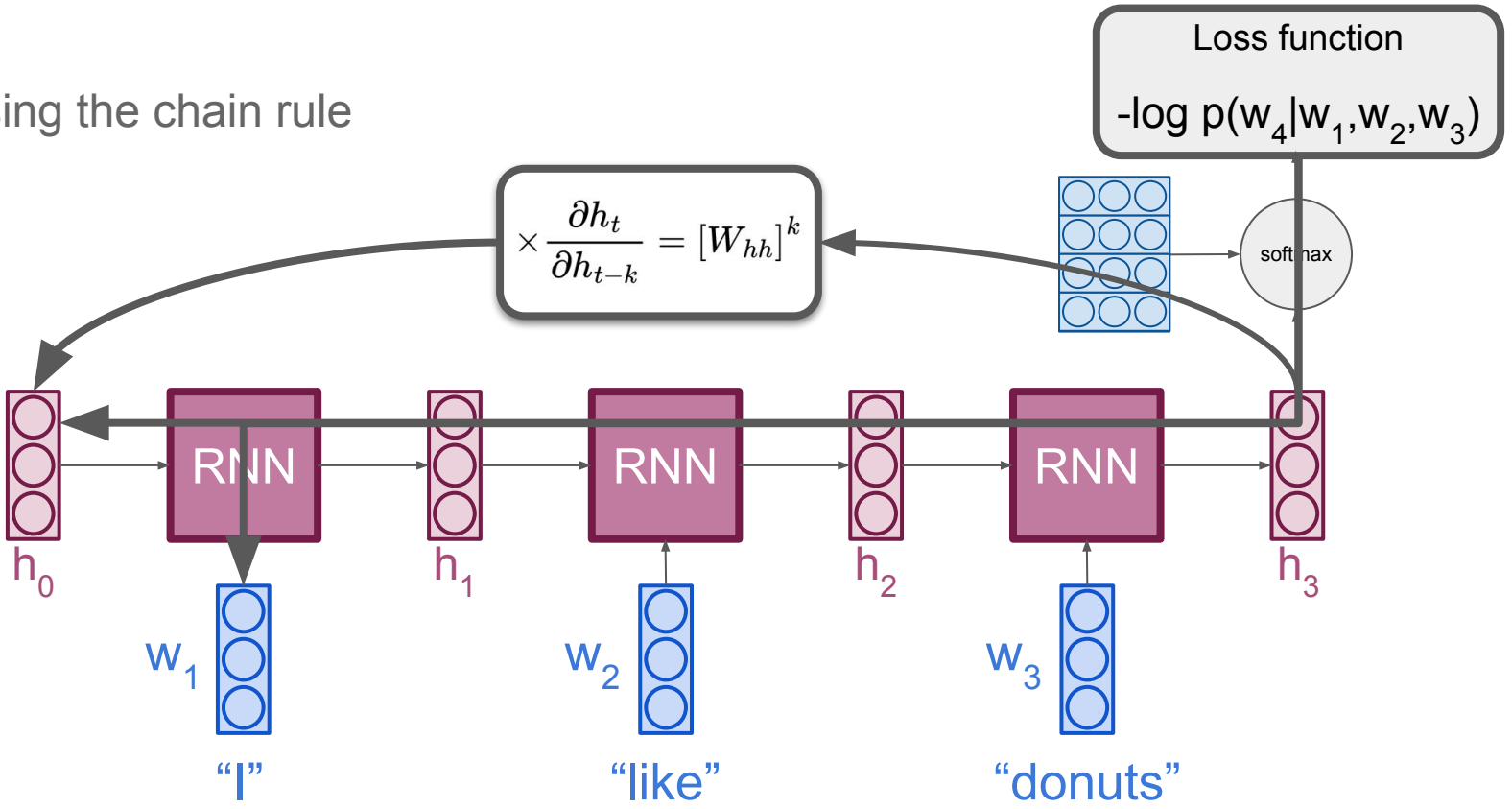
Using the chain rule



Vanishing Gradients

RNN | generative

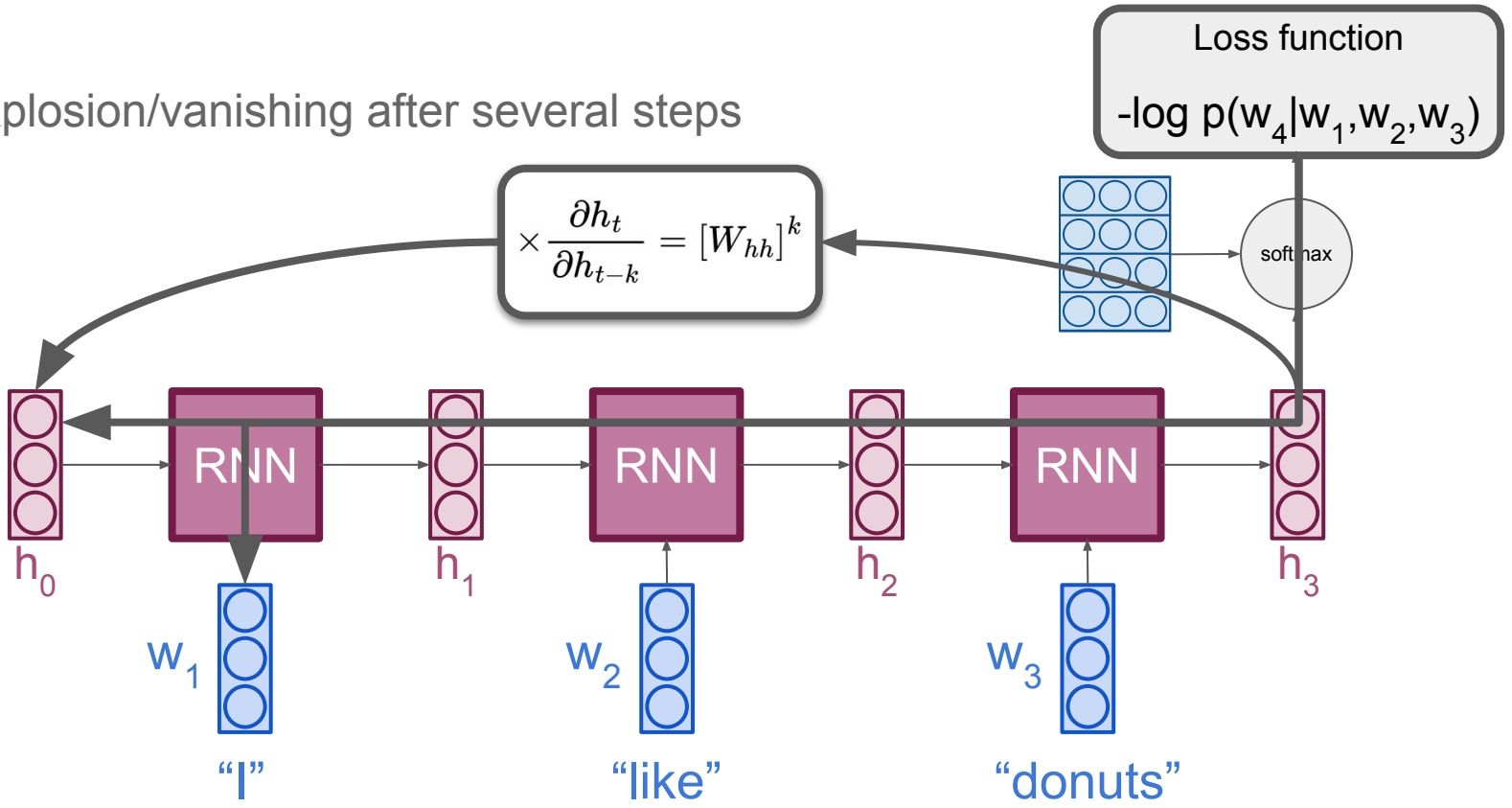
Using the chain rule



Vanishing Gradients

RNN generative

Explosion/vanishing after several steps



Backpropagation through time (BPTT)

For a simple RNN function f, with a recurrent weights matrix w_h, applied on a sentence of T word embeddings x_t, word labels y_t, hidden states h_t.

$$h_t = f(x_t, h_{t-1}, w_h),$$

$$L(x_1, \ldots, x_T, y_1, \ldots, y_T, w_h) = \frac{1}{T} \sum_{t=1}^{T} l(y_t, h_t)$$

$$\frac{\partial L}{\partial w_h} = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial l(y_t, h_t)}{\partial w_h} = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial l(y_t, h_t)}{\partial h_t} \frac{\partial h_t}{\partial w_h}$$

Chain rule

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}.$$

Classic derivative of multivariable function $h_t = f(x_t, h_{t-1}(w_h), w_h)$

Backpropagation through time (BPTT)

For a simple RNN function f, with a recurrent weights matrix w_h, applied on a sentence of T word embeddings x_t, word labels y_t, hidden states h_t.

$$h_t = f(x_t, h_{t-1}, w_h),$$

$$L(x_1, \ldots, x_T, y_1, \ldots, y_T, w_h) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, h_t)$$

$$\frac{\partial L}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, h_t)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, h_t)}{\partial h_t} \frac{\partial h_t}{\partial w_h}$$

Chain rule

Classic derivative of multivariable function $h_t = f(x_t, h_{t-1}(w_h), w_h)$

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}.$$

$= w_h^{t-j} * \text{activation}(h_j)$

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}.$$

Modified equations from https://d2l.ai/chapter_recurrent-neural-networks/bptt.html

Vanishing and exploding gradient

The update of W_h is a sum of products of W_h^{t-j}

Vanishing gradient: **some** long term dependencies will have a gradient of 0

Exploding gradient: the total gradient update will overflow (i.e unstable training , NaN in your loss function)

When does these products go to 0 or diverge ?

“It is sufficient for $\rho < 1$, where ρ is the spectral radius (i.e the largest eigenvalue) of the recurrent weight matrix (w_h), for long term components to vanish (as $t \rightarrow \infty$) and necessary for $\rho > 1$ for them to explode”
Pascanu et al. On the difficulty of training recurrent neural networks

How to deal with vanishing and exploding gradients, in RNNs?

Vanishing?

Use LSTM/GRU (i.e RNNs with *skip connections*)

Exploding?

Use gradient clipping

Truncated BPTT (i.e remove long term dependencies)

Both?

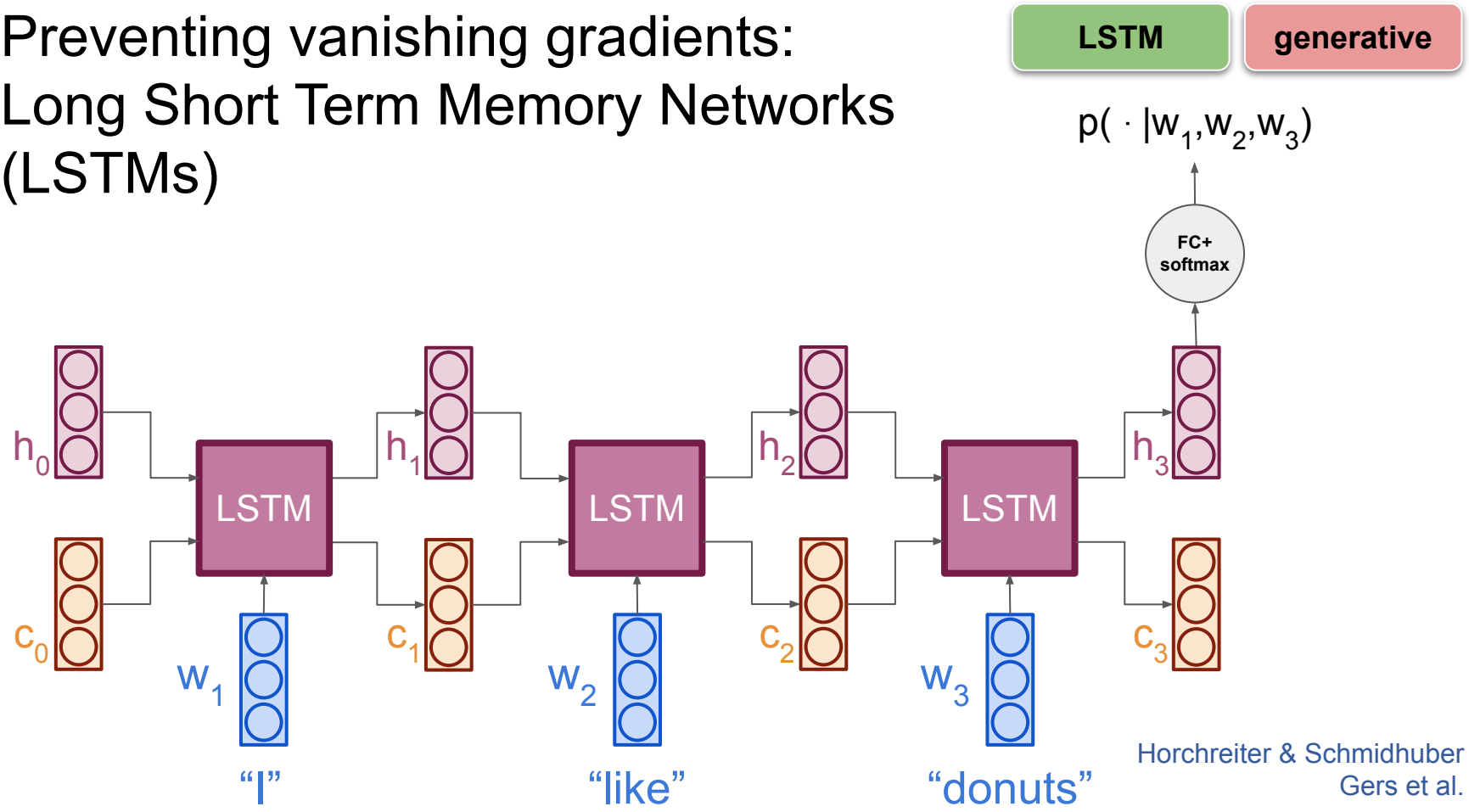
Good weights initialisation: Xavier Initialisation

LayerNorm:

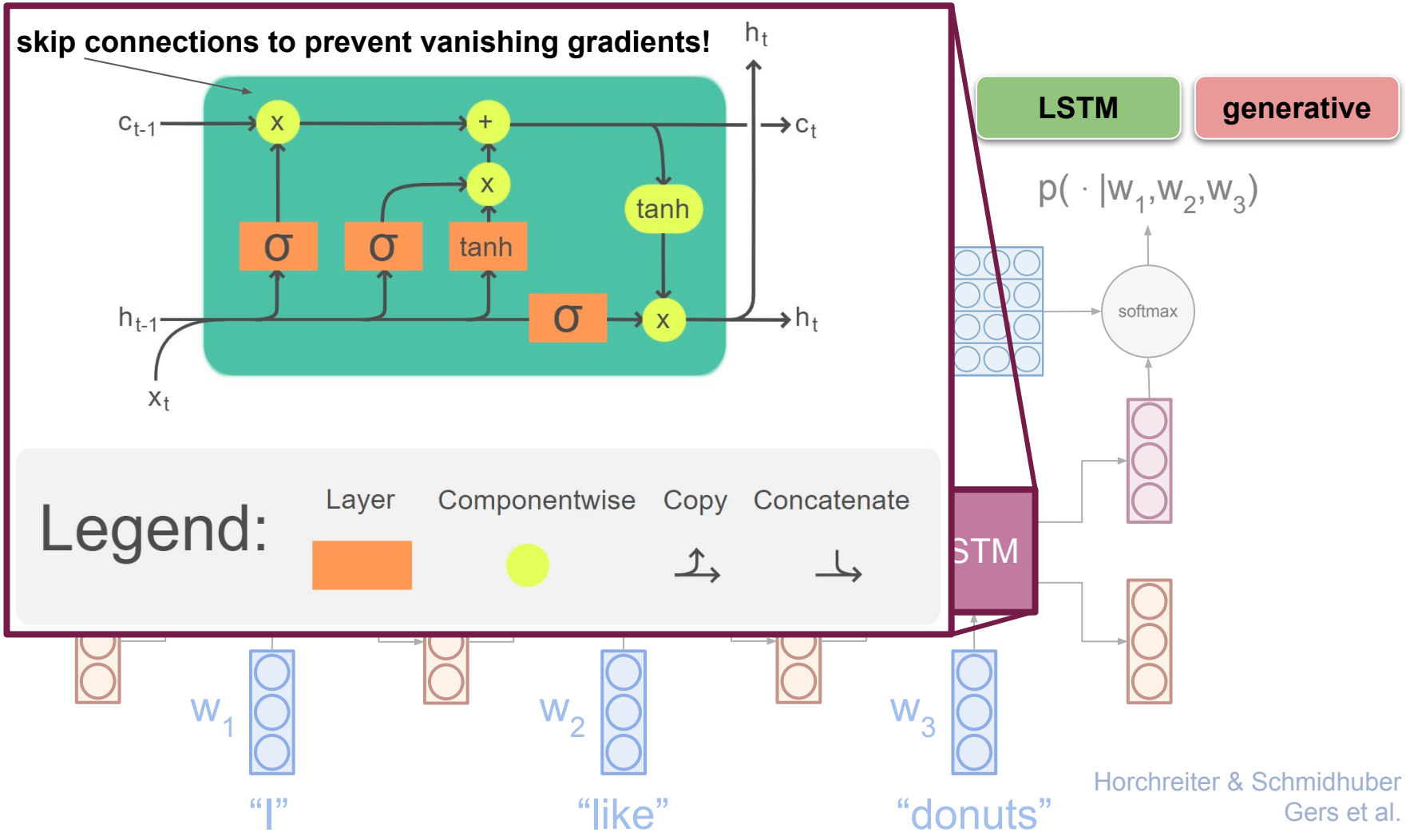
to prevent saturating sigmoid units(i.e vanishing grads)

to prevent high values in for ReLU activations (i.e exploding grads)

Preventing vanishing gradients: Long Short Term Memory Networks (LSTMs)



Horchreiter & Schmidhuber (1997)
Gers et al. (2000)

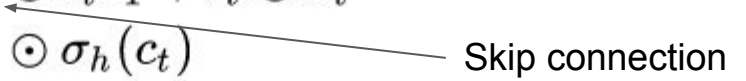


Horchreiter & Schmidhuber (1997)
Gers et al. (2000)

Preventing vanishing gradients: LSTM and GRU formulas

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma_h(c_t)
\end{aligned}
$$

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \hat{h}_t
\end{aligned}
$$



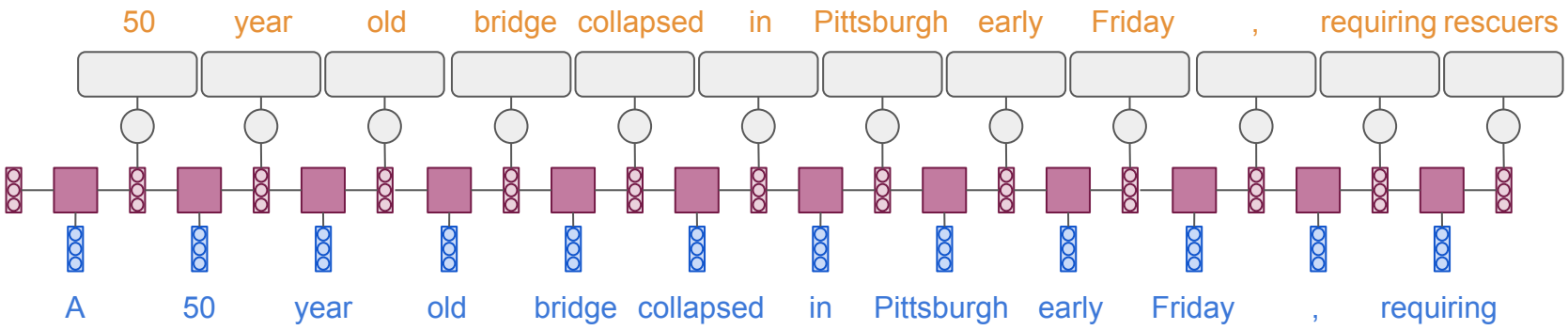
LSTM: 8 matrices, one skip connection

GRU: 6 matrices, one skip connection

Preventing exploding gradients: Truncated BPTT

RNN generative

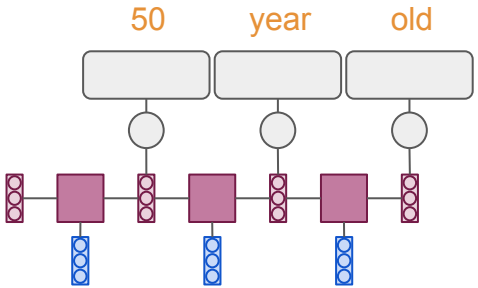
Each update requires a lot of compute and exploding gradients problem



Truncated BPTT

RNN generative

Compute gradient on shorter sub-sequence
Update parameters

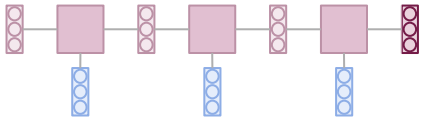


A 50 year old bridge collapsed in Pittsburgh early Friday , requiring

Truncated BPTT

RNN generative

Keep last hidden state

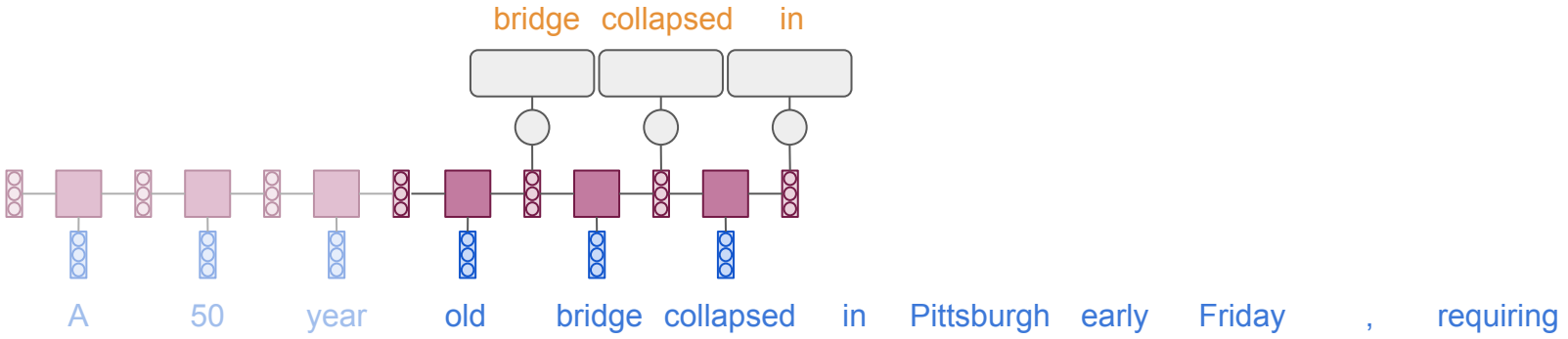


A 50 year old bridge collapsed in Pittsburgh early Friday , requiring

Truncated BPTT

RNN generative

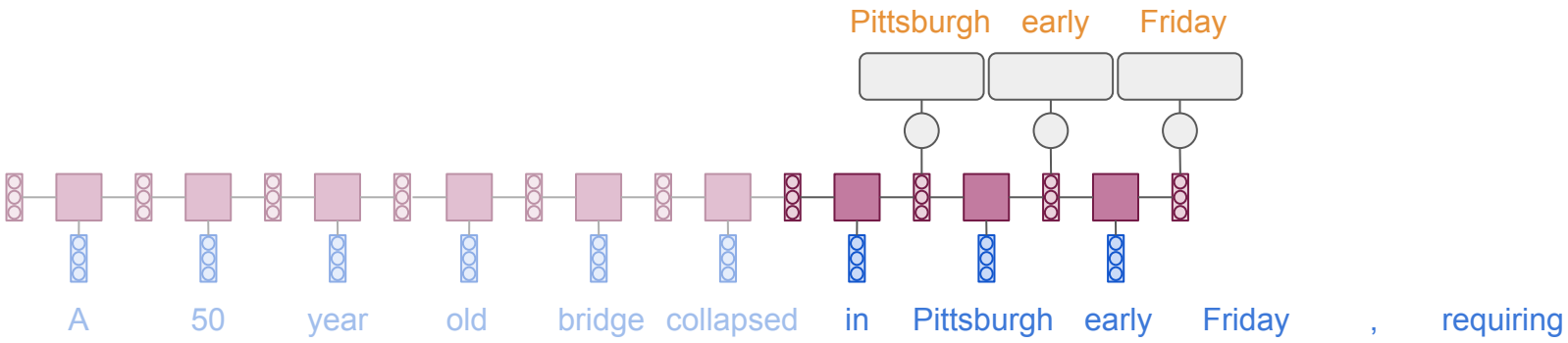
Compute gradient over next sub-sequence



Truncated BPTT

RNN generative

Etc…

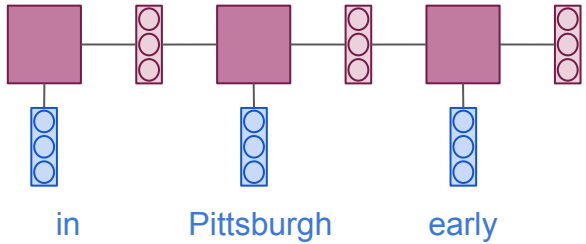


Vanishing and exploding gradients: Layer Norm

RNN | generative

Each dimension of each layer of the hidden states is normalised over the whole sentence. Gamma and Beta are learned

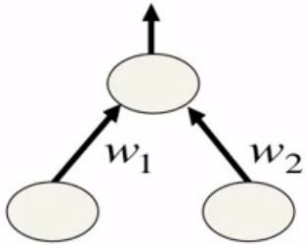
$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$



Why not batch norm?
For text data, batches are very uneven in length.

Layer Norm prevents sigmoids from saturating (i.e vanishing gradients) and ReLU from diverging (exploding gradients)

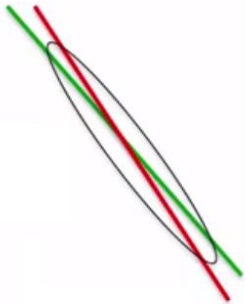
why layer norm reduces numerical instability?



color indicates training case

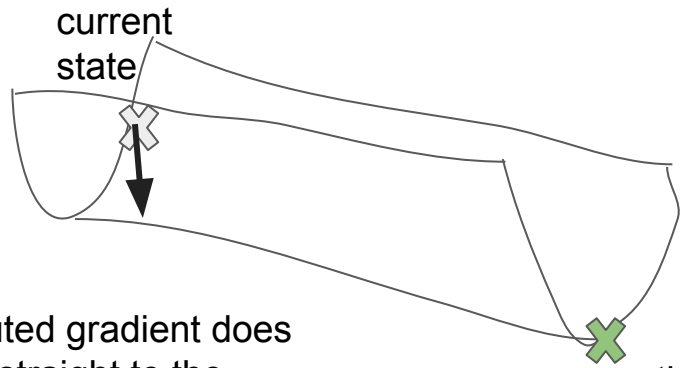
101, 101 → 2
101, 99 → 0

gives error surface



1, 1 → 2
1, -1 → 0

gives error surface



Computed gradient does not go straight to the optimal: need to normalise layer inputs

Recurrent Models in a Nutshell

LSTM

generative

- Better language models than n-grams

Recurrent Models in a Nutshell

LSTM

generative

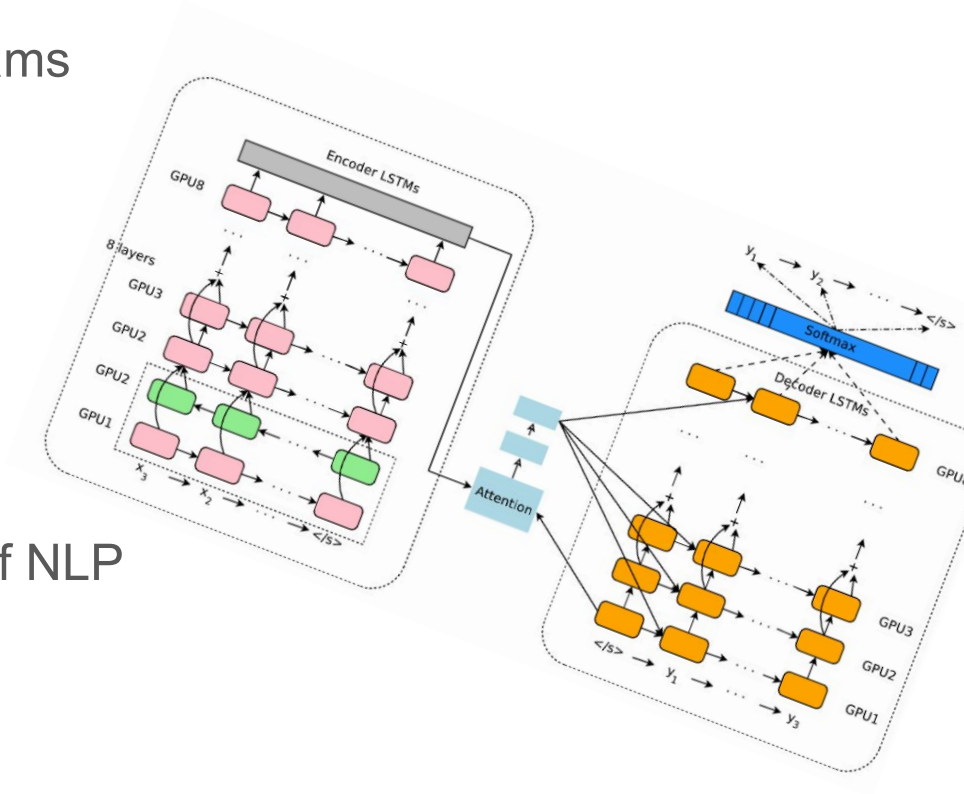
- Better language models than n-grams

- Highly tuned
  - Fast implementations
  - Good default hyper-parameters

Recurrent Models in a Nutshell

LSTM | generative

- Better language models than n-grams
- Highly tuned
  - Fast implementations
  - Good default hyper-parameters
- LSTM & variants were backbone of NLP at some point
  - Google translate ran on LSTM stack

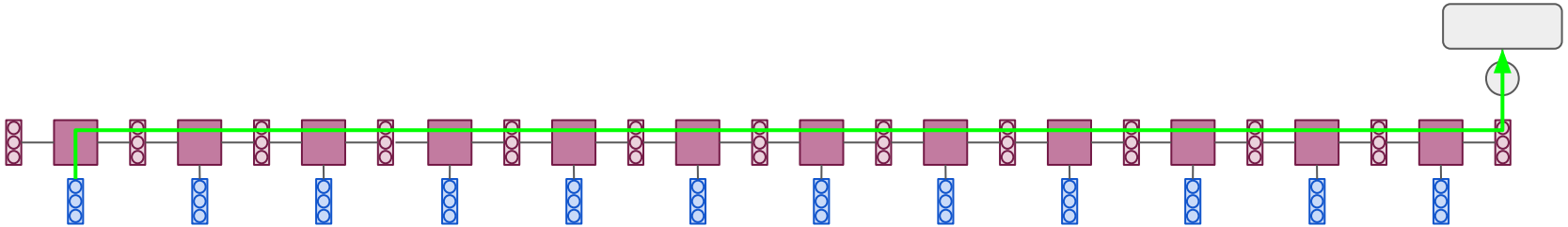


Downsides of Recurrent Models

LSTM

generative

- How much context do they really use?
  - Theoretically unbounded



Downsides of Recurrent Models

LSTM generative

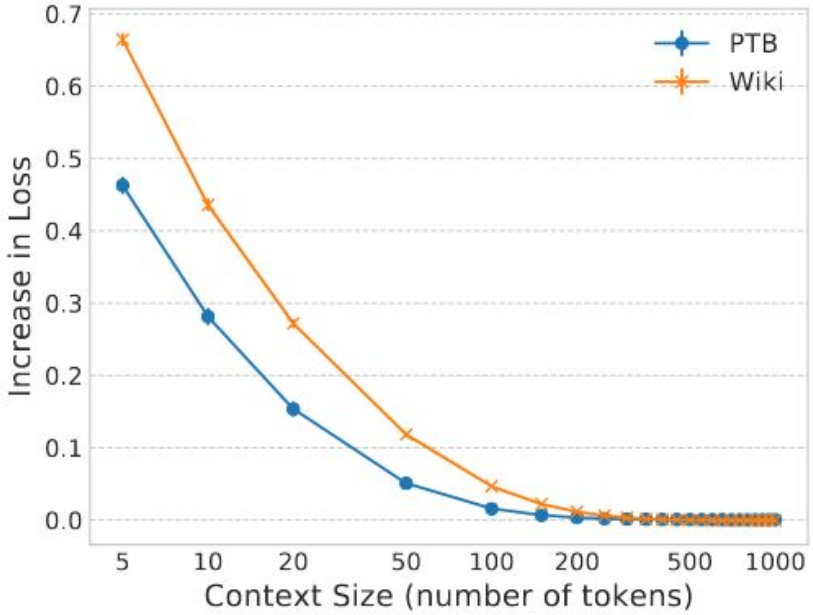
- How much context do they really use?
  - Theoretically unbounded
  - In practice: ~200 tokens

From Khandelwal et al. (2018)

Downsides of Recurrent Models

LSTM generative

- How much context do they really use?
  - Theoretically unbounded
  - In practice: ~200 tokens



From Khandelwal et al. (2018)

Downsides of Recurrent Models

LSTM generative

- How much context do they really use?
  - Theoretically unbounded
  - In practice: ~200 tokens

- Scalability issues
  - Need to encode each token **sequentially**
  - Hard to parallelize during training
  - Difficult to scale to >8 layers (vanishing gradients, etc..)

A word on Out-of-Vocabulary words

LSTM

generative

A word on Out-of-Vocabulary words

LSTM generative

How to use a trained LM on a sentence that has a new word-type ?

Byte-Pair-Encoding (BPE):

- Any sequence of letters => N known smaller subwords
- Example:

  INPUT: All of a sudden, he started playing table tennis with John Doe
  OUTPUT: All _of _a _sud_den _, _he _start_ed _play_ing _table _tennis _with _John_Do_e _.

In practice 50k units are used in current language models

A word on Out-of-Vocabulary words

LSTM generative

How does it work?

Recursively get the byte pair that occurs most often and substitute it.

Example (wikipedia):

Aaabdaaabac

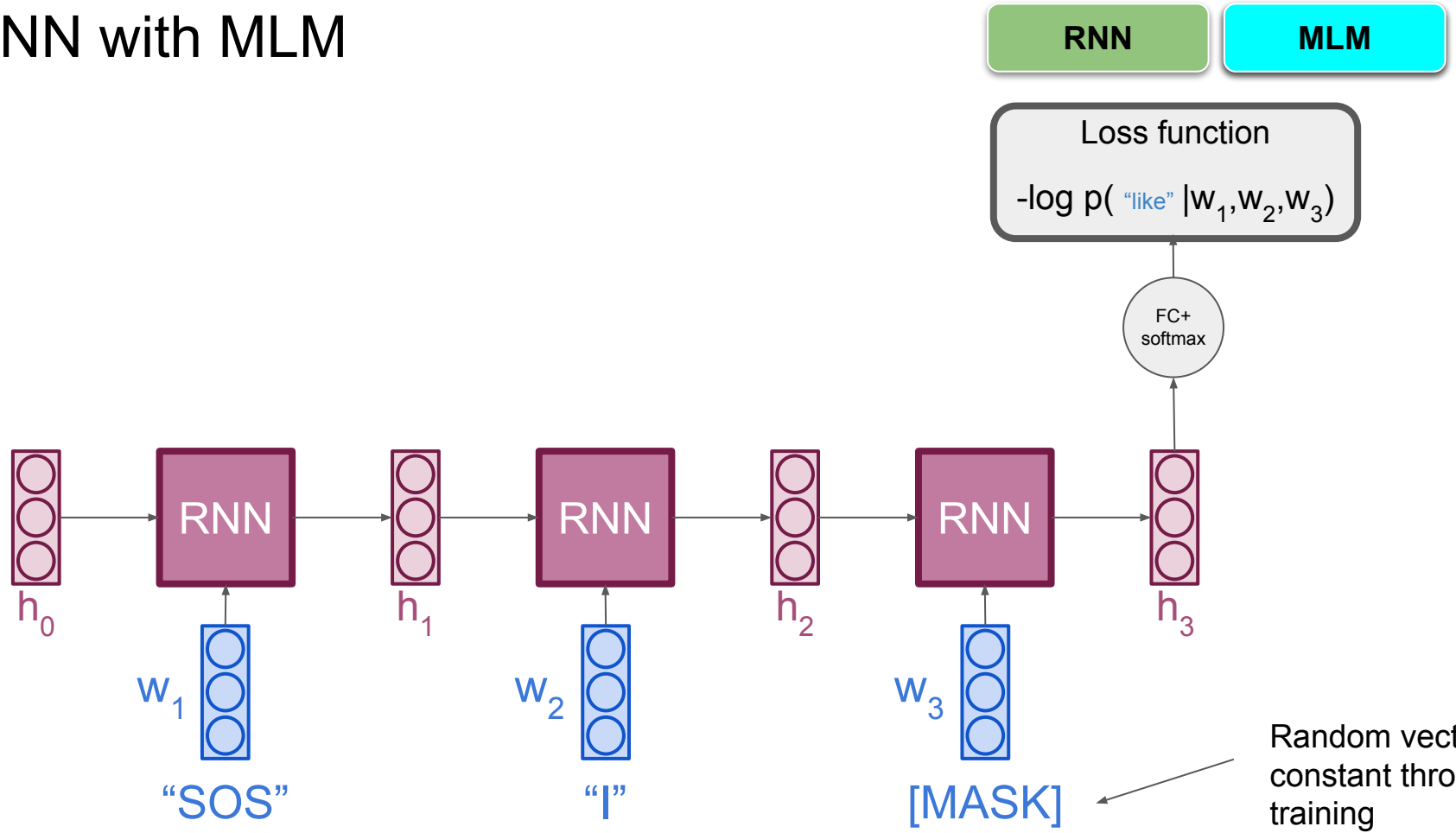
ZabdZabac (Z=aa)

ZYdZYac (Z=aa, Y=ab)

XdXac (X=ZY, Y=ab, Z=aa)

Bidirectional Recurrent Neural Networks For Masked Language Modelling

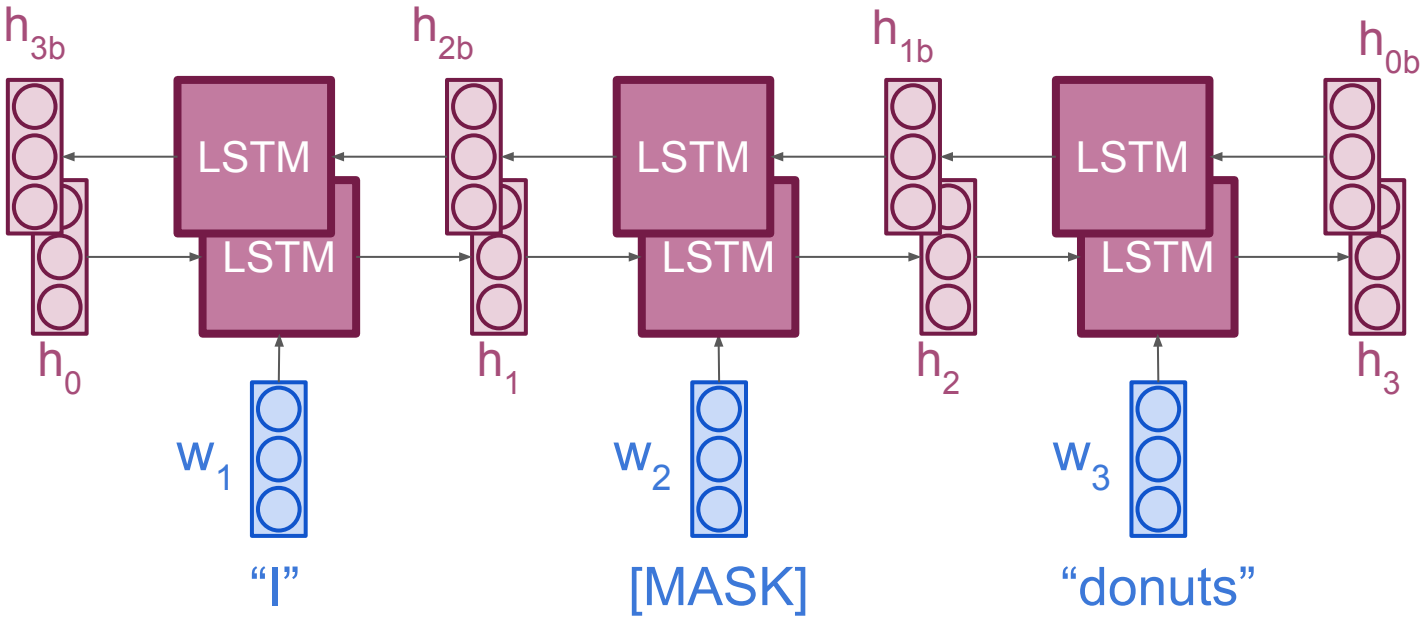
RNN with MLM


RNN
MLM
Loss function
-log p("like" |w1,w2,w3)
FC+ softmax
RNN
RNN
RNN
h0
h1
h2
h3
w1
w2
w3
"SOS"
"I"
[MASK]
Random vector, constant through training

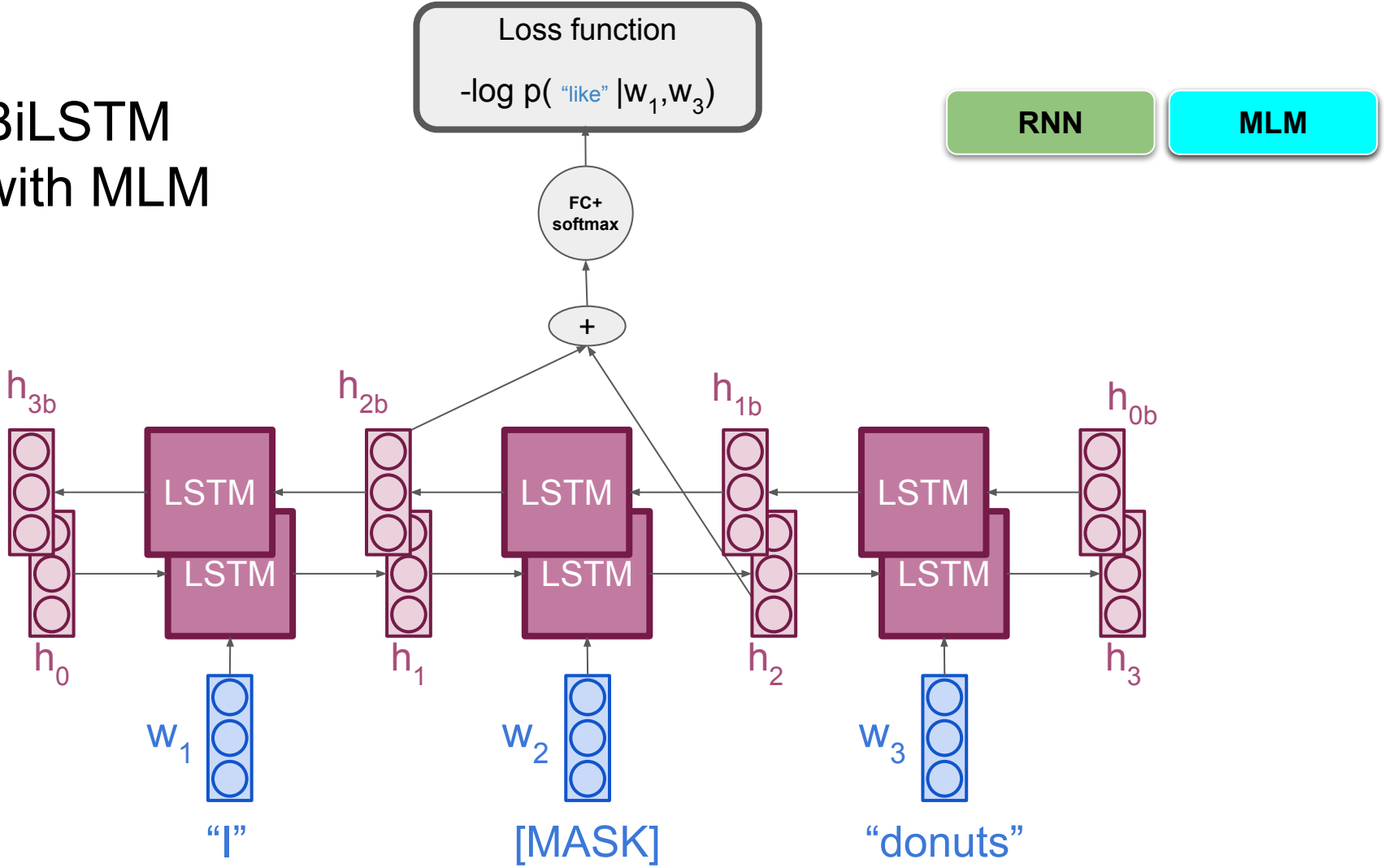
BiLSTM with MLM

RNN MLM

BiLSTM are two **independent** LSTM models whose outputs are concatenated after their last layer



BiLSTM with MLM

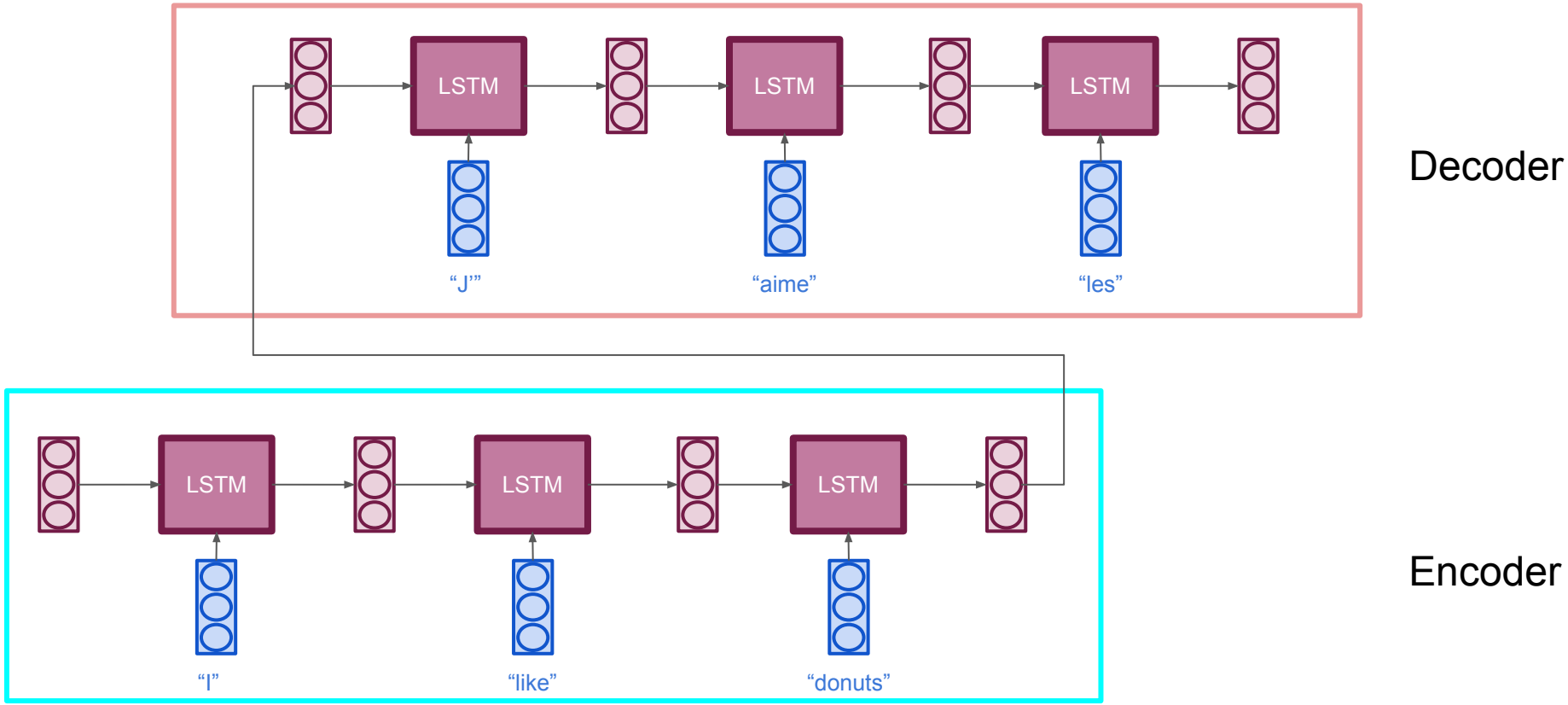


Recurrent Neural Networks

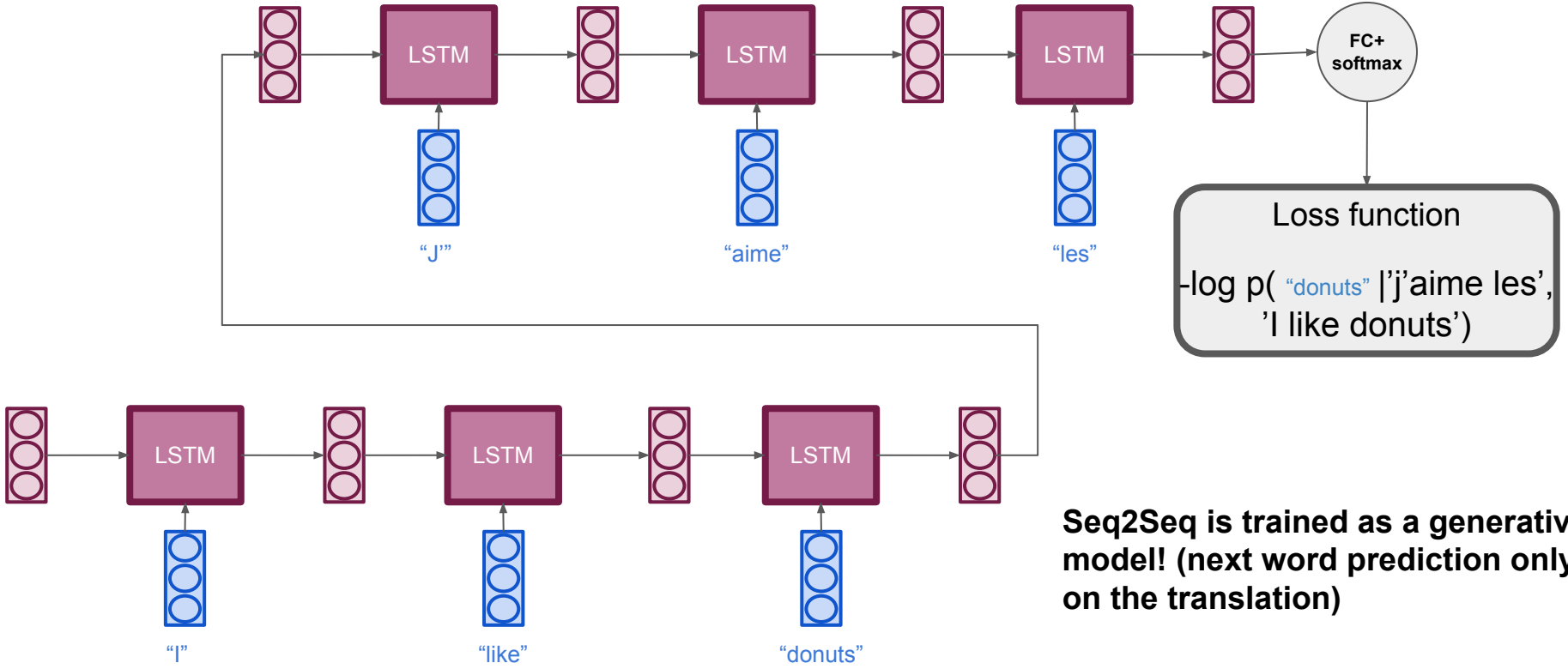
For Sequence to Sequence

Vanilla Seq2seq RNN

RNN seq2seq

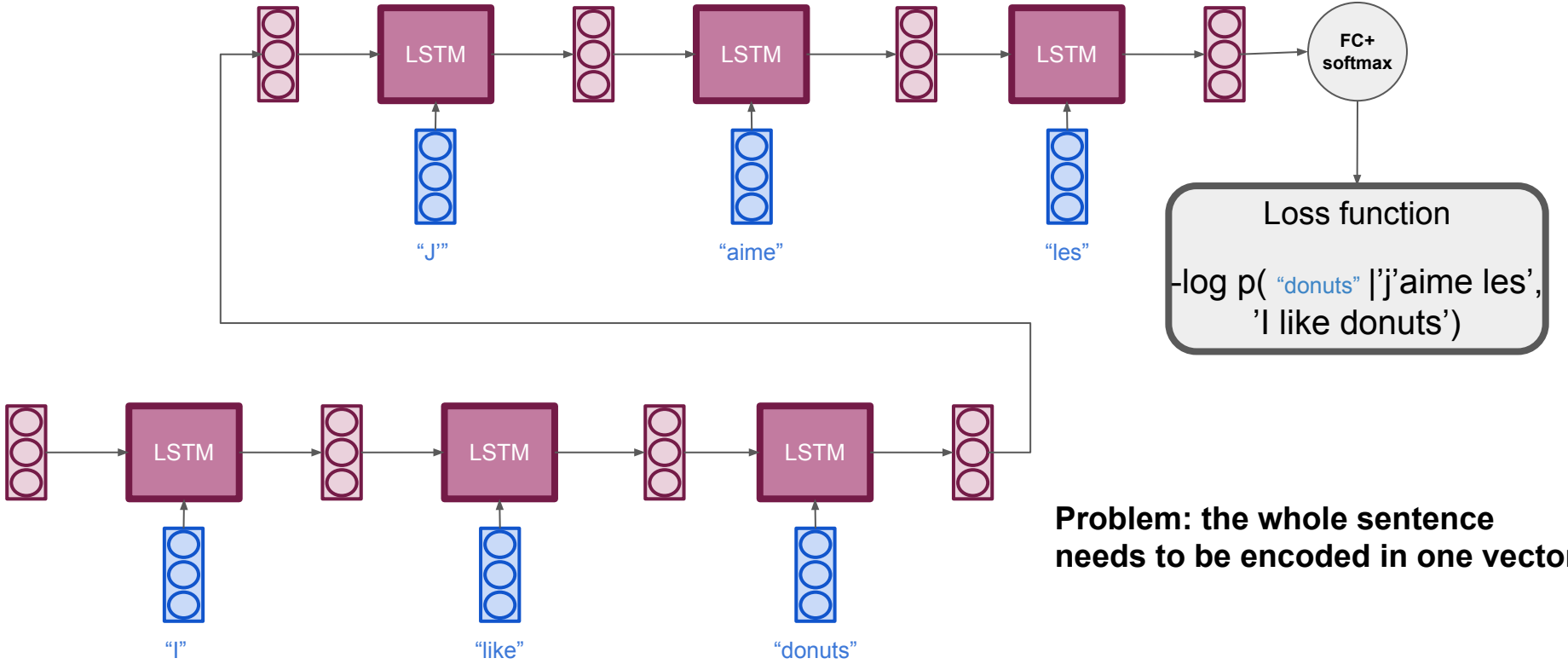


Vanilla Seq2seq RNN: training



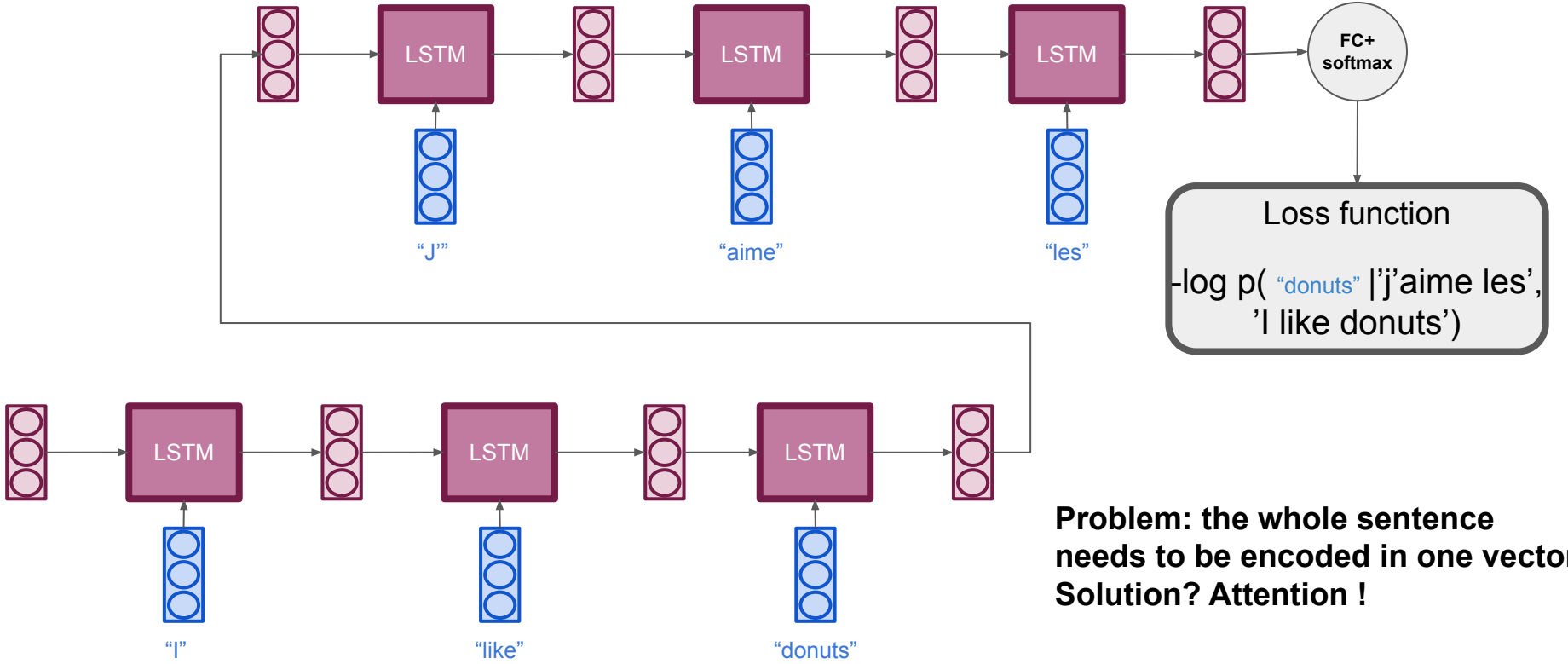
Seq2Seq is trained as a generative model! (next word prediction only on the translation)

Vanilla Seq2seq RNN: training (generative)



Problem: the whole sentence needs to be encoded in one vector

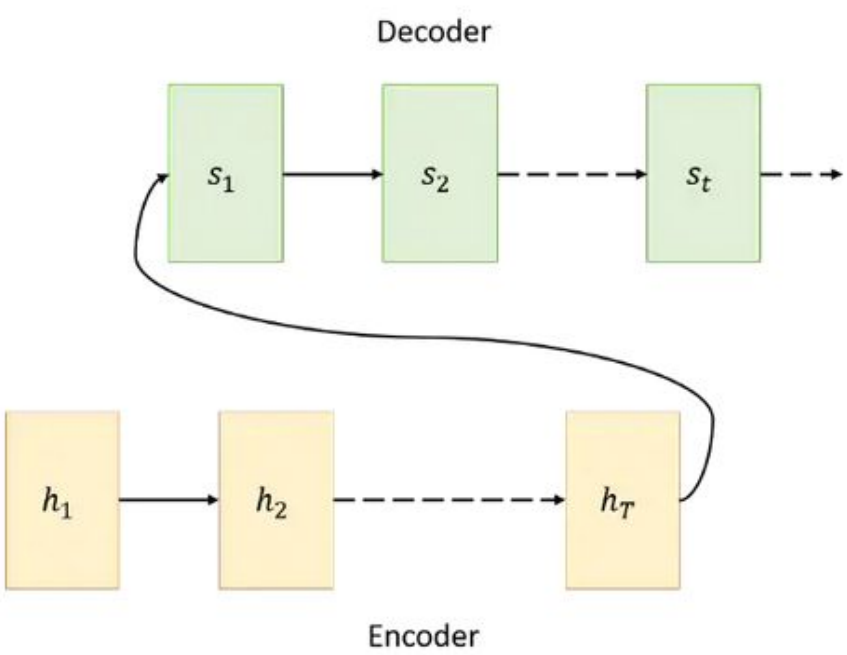
Vanilla Seq2seq RNN: training (generative)



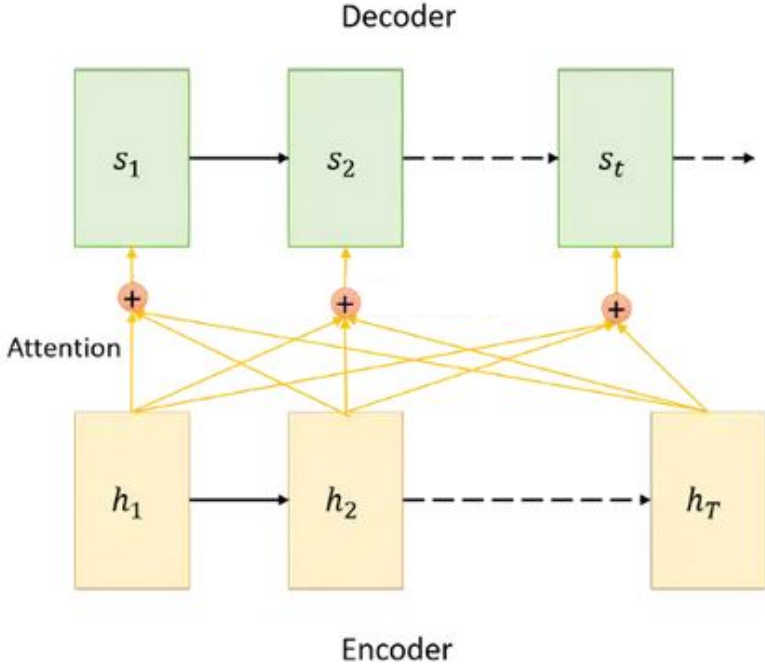
Problem: the whole sentence needs to be encoded in one vector
Solution? Attention !

Attention Seq2seq RNN

RNN seq2seq



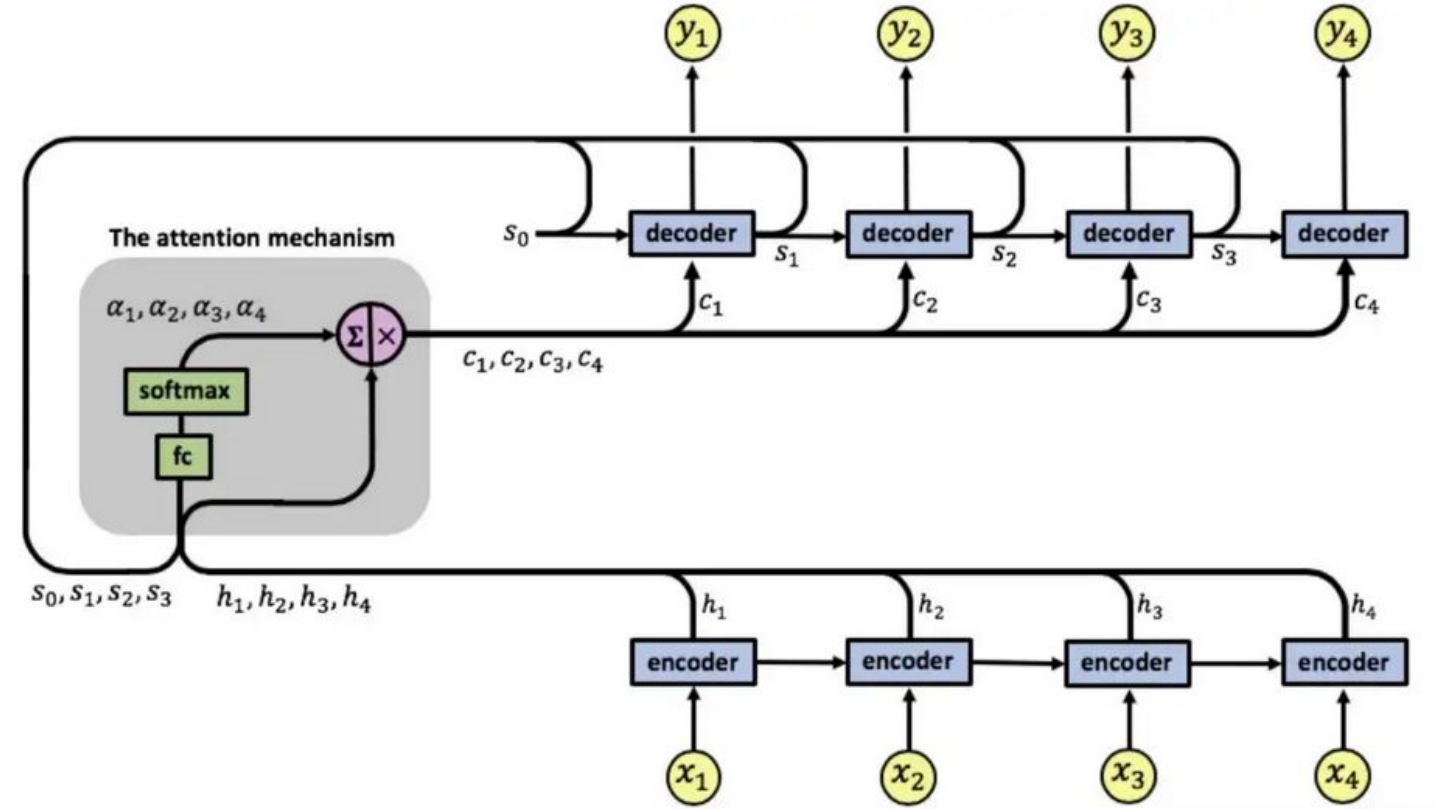
(a) Vanilla Encoder Decoder Architecture



(b) Attention Mechanism

Attention Seq2seq RNN

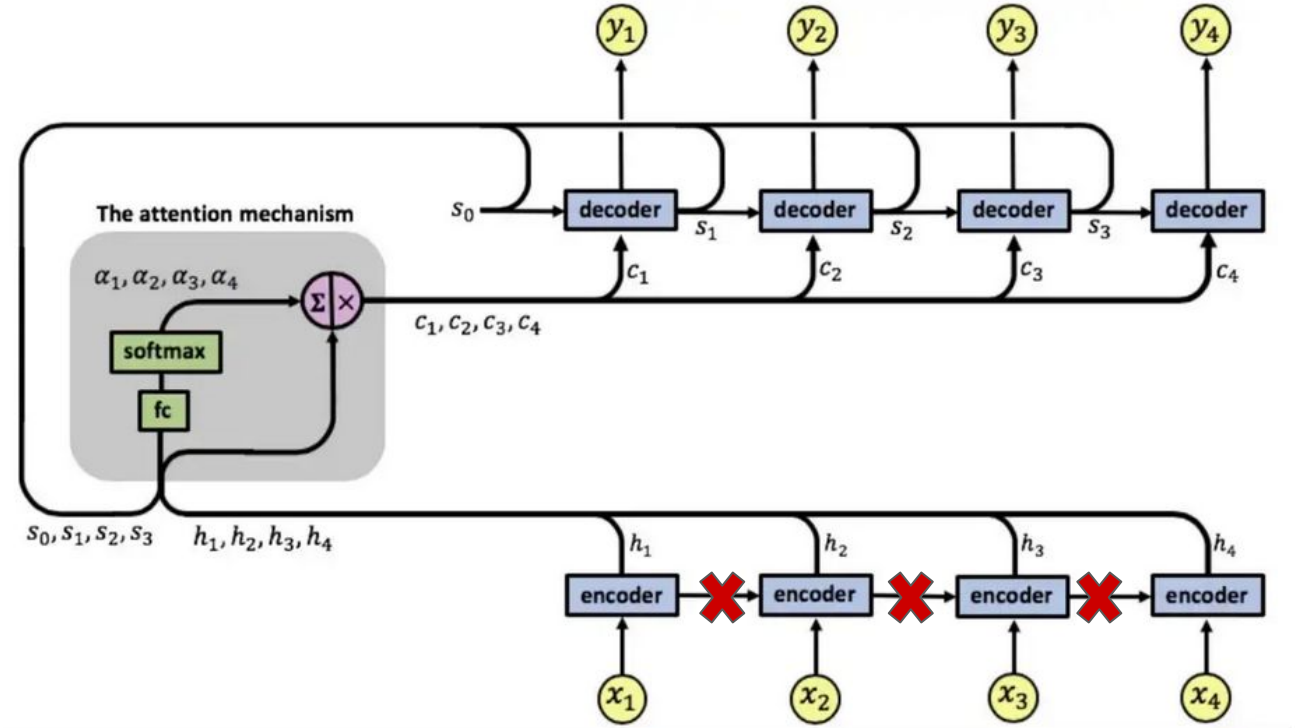
RNN seq2seq



Attention is all you need?

RNN

seq2seq

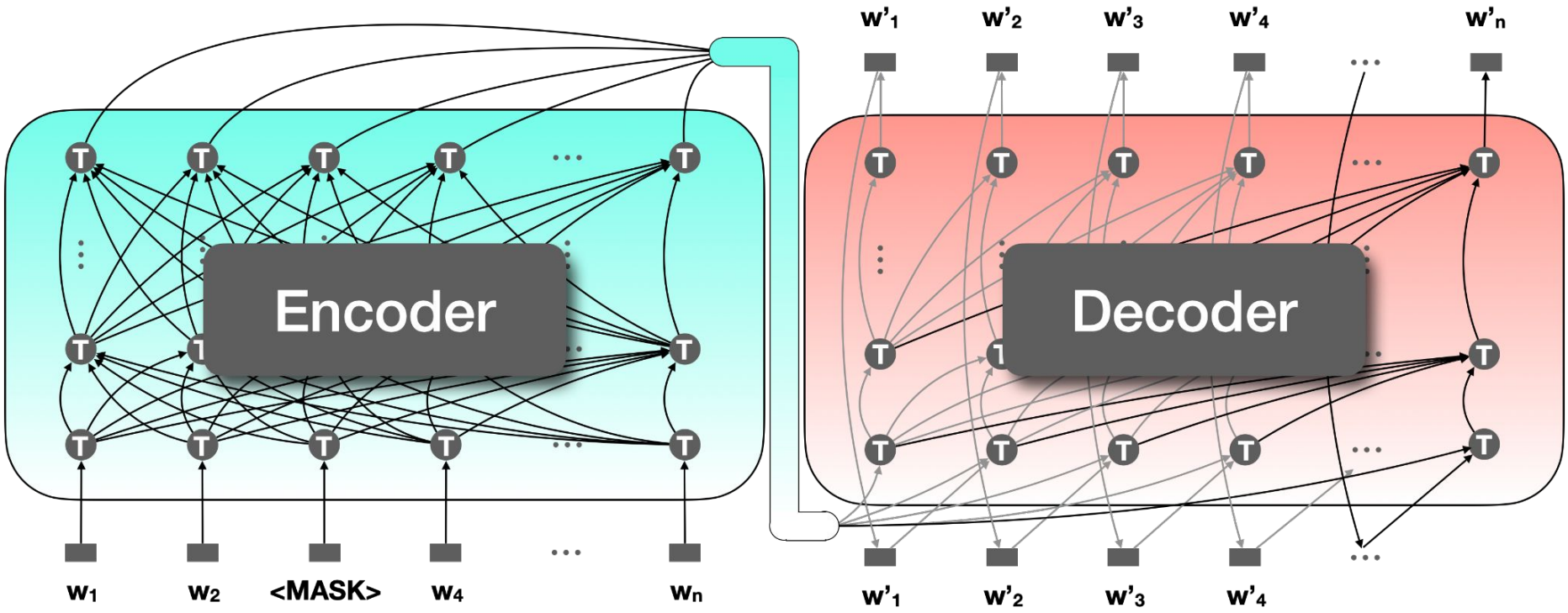


Attention works really well and causality is **very slow**. What if we remove the temporal flow in the encoder and use only attention?

Transformer

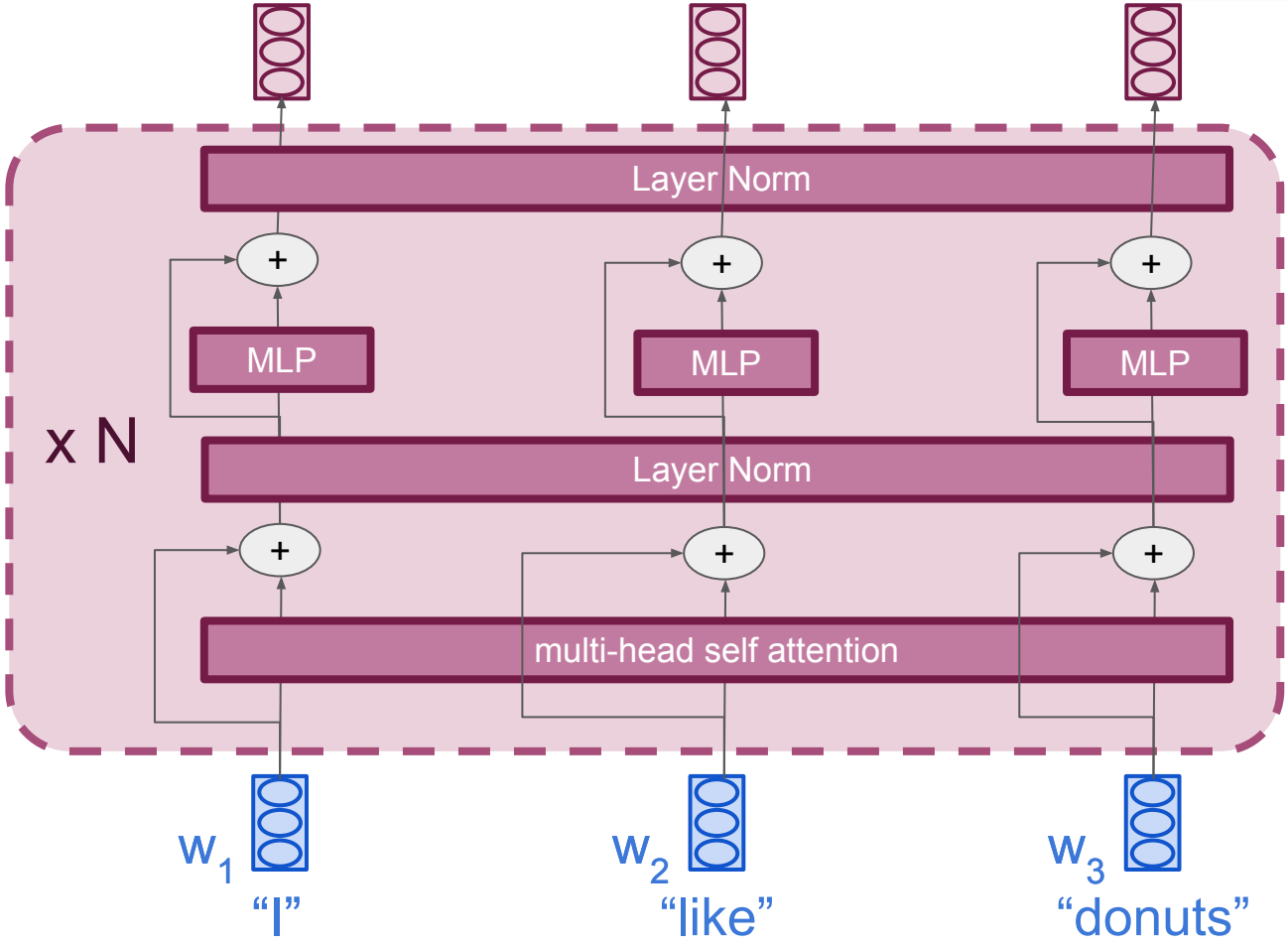
The full Transformer architecture

Transformer seq2seq



Transformer Encoder

Transformer MLM

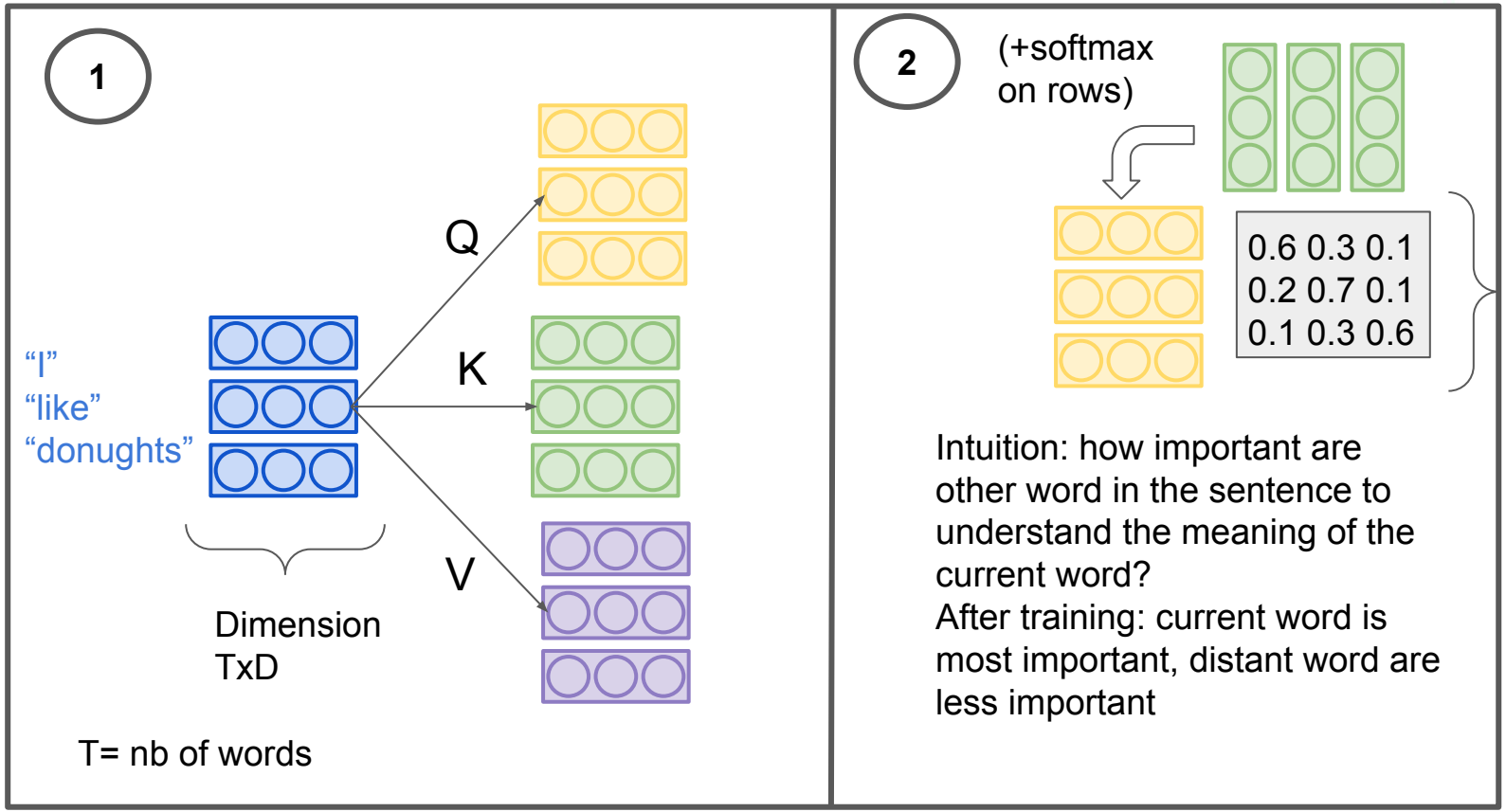


No more causal architecture
The whole sentence is processed at once!

Not shown here: as for RNNs, there is a big embedding matrix used to create the $w_1 w_2 w_3$ and to compute the final softmax.

Self-Attention* in 3 steps

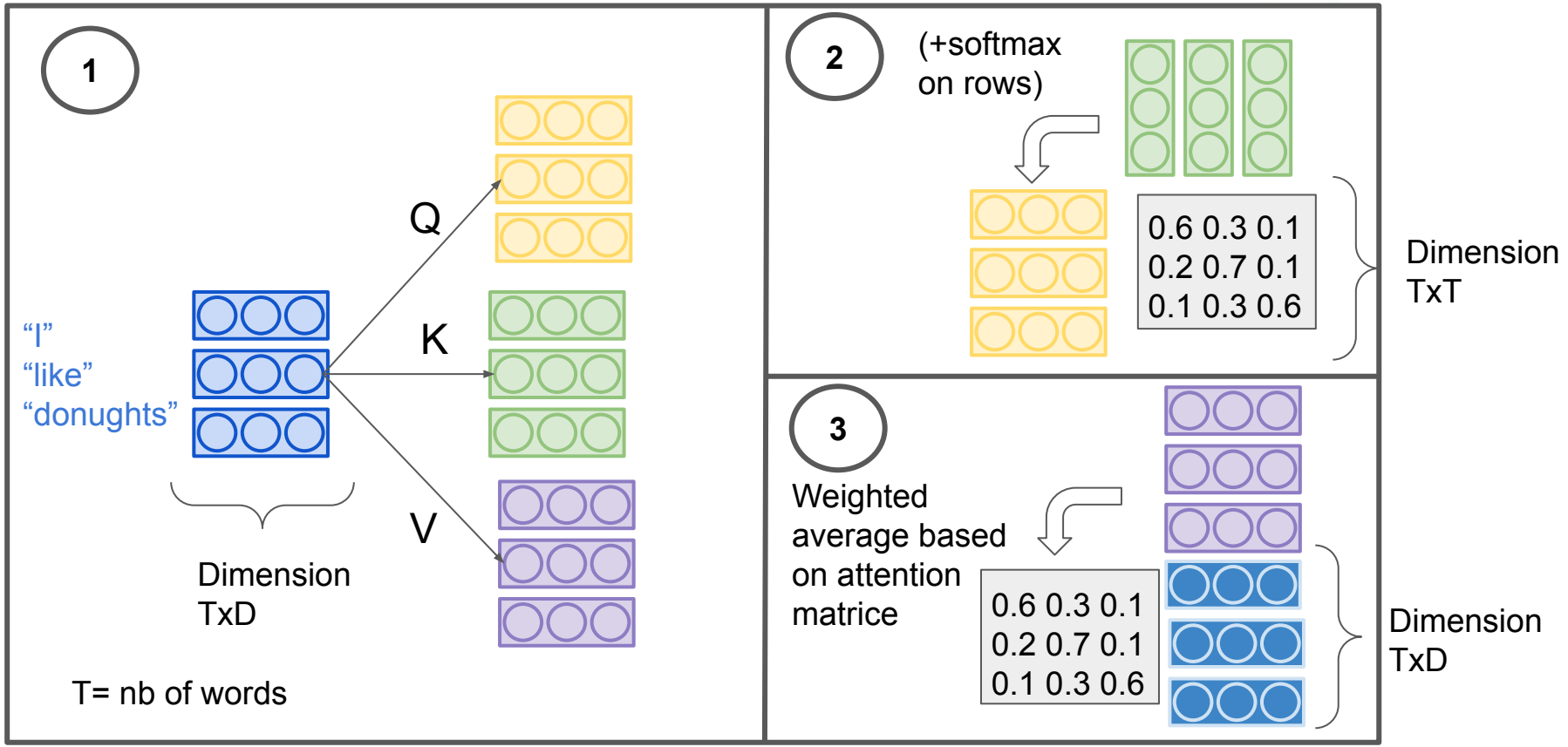
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



*Vaswani et al. Attention is all you need.

Self-Attention* in 3 steps

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



*Vaswani et al. Attention is all you need.

Self-Attention* in 3 steps

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

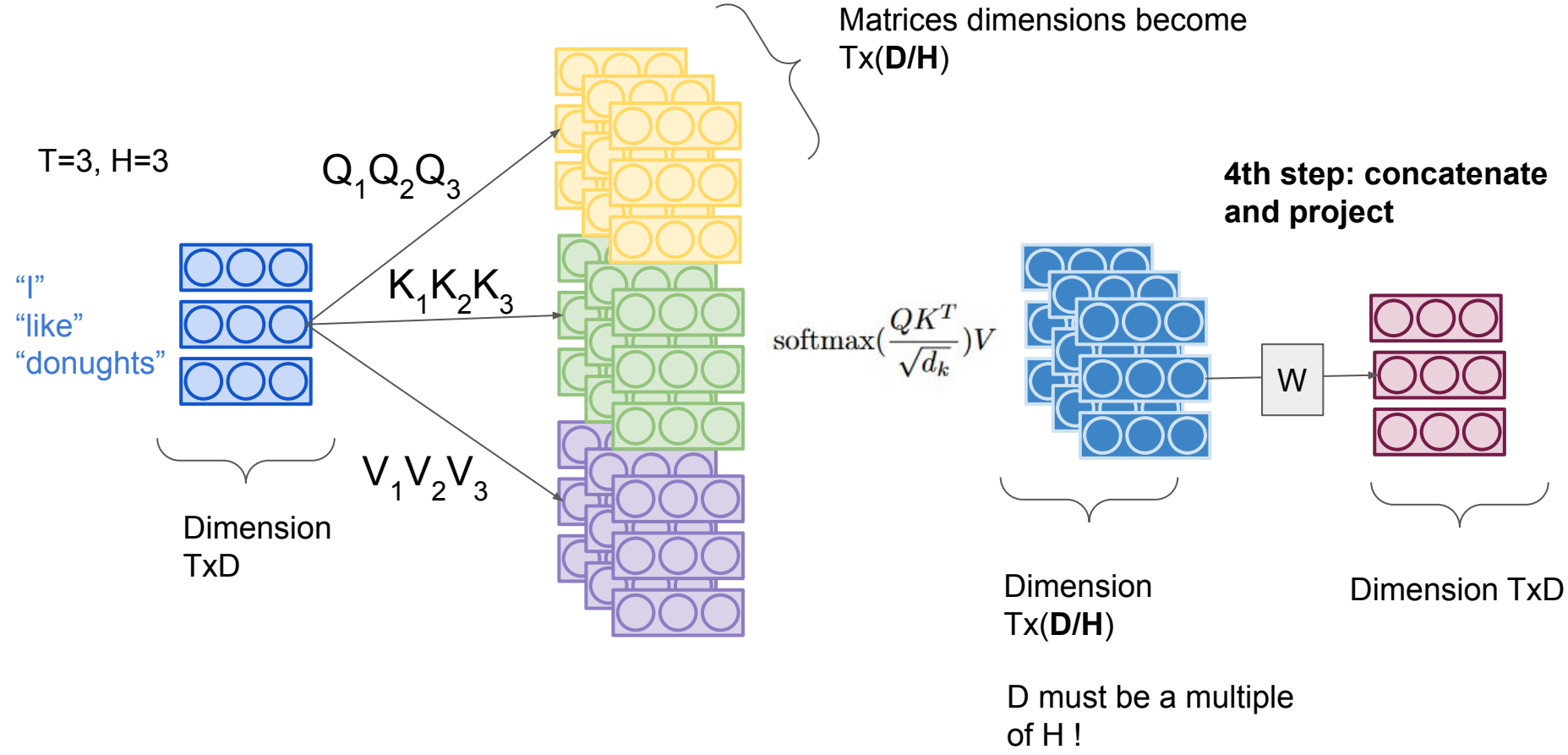
Why division in the softmax?

The dot product creates very large values for large weight matrices and results in NaN values. (it still does even with the division but less)

Why ?
If the output of Q and K are normalised (mean 0 and variance 1)
Then the attention values

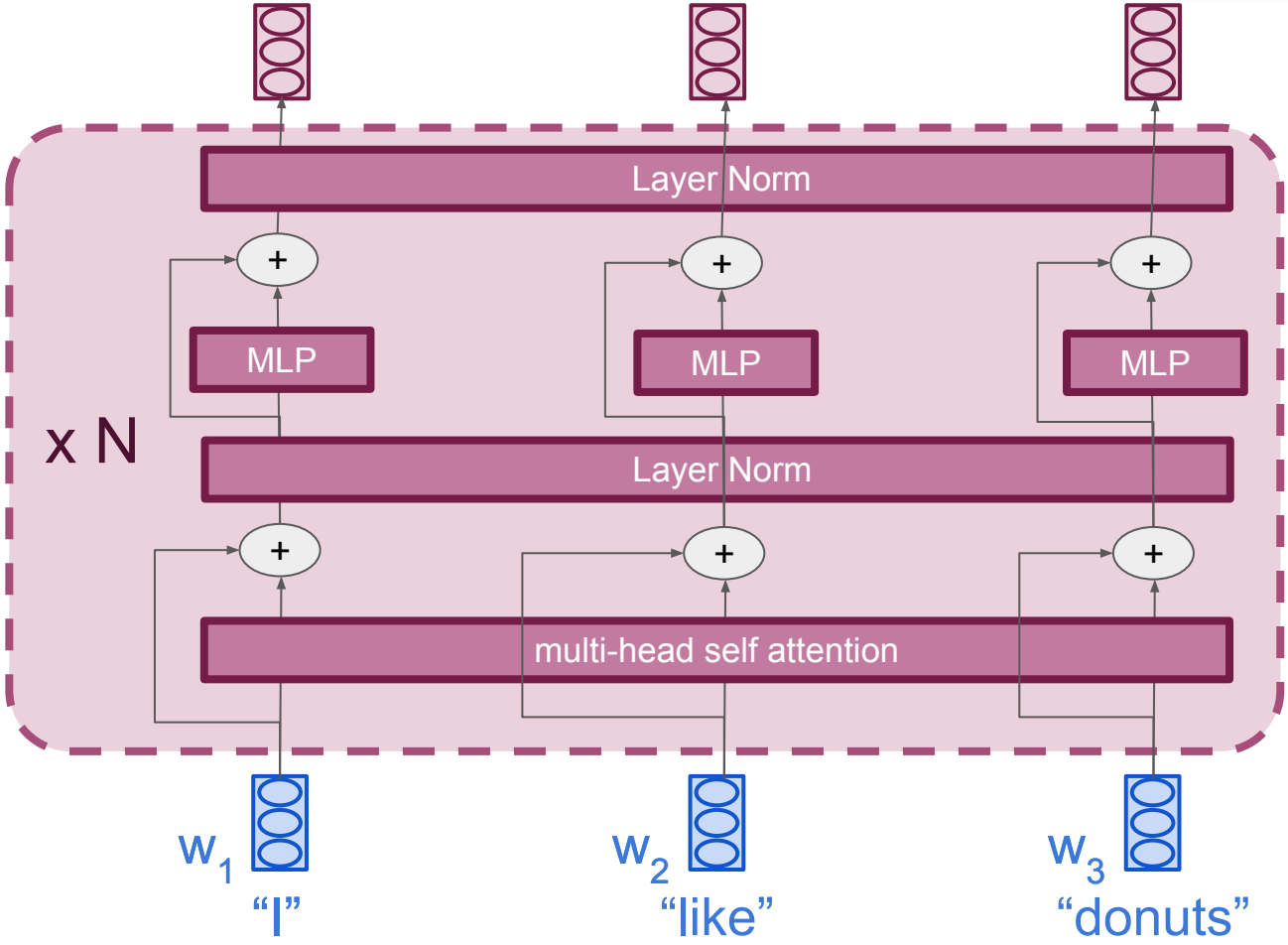
$q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance d_k.

Multi-head self-attention (4 steps)



Transformer Encoder

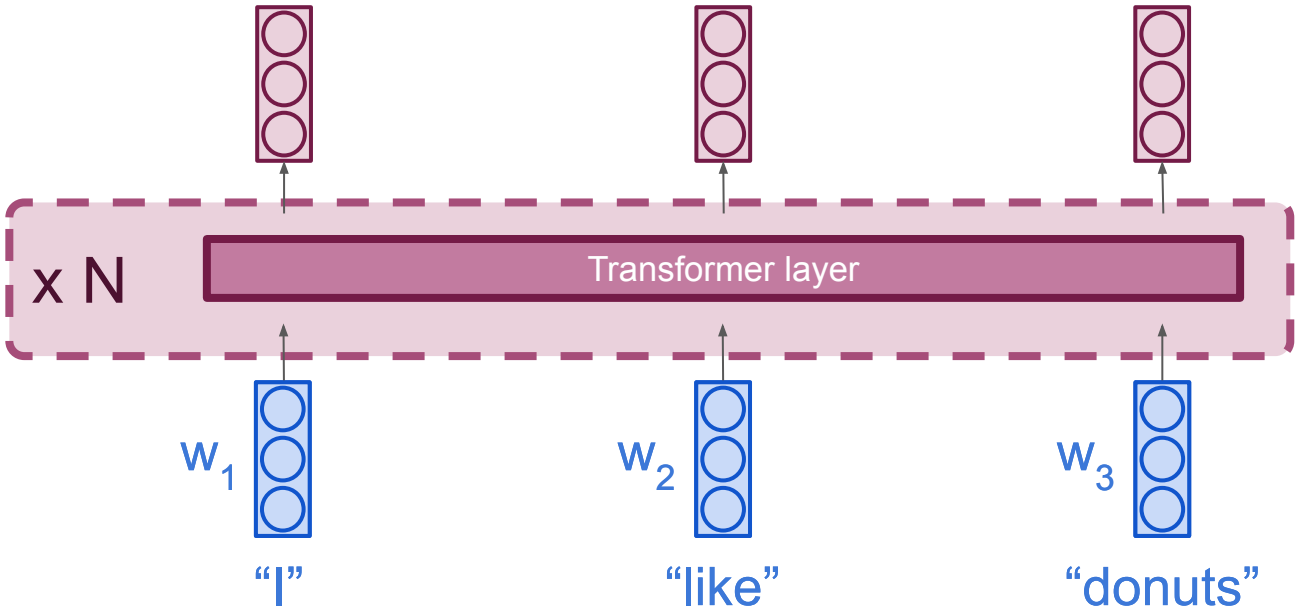
Transformer | MLM



Residual connection: prevent vanishing gradient. Necessary in all deep architectures.

How is word order encoded?

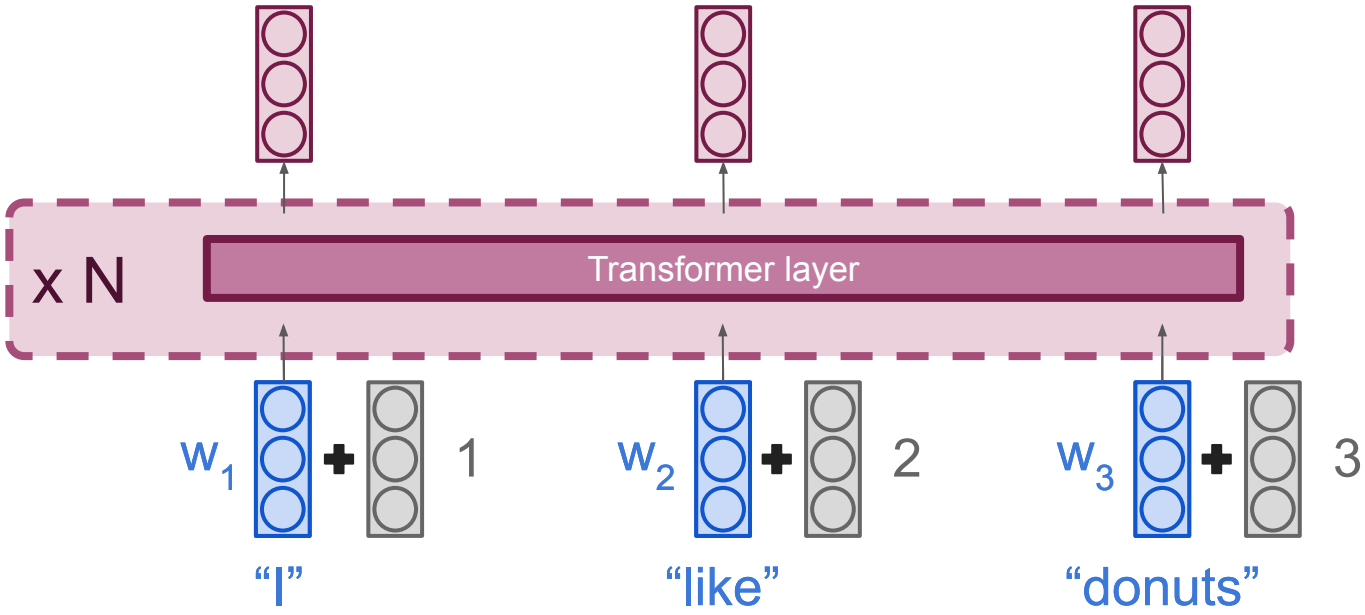
Transformer | MLM



How is word order encoded?

Transformer | MLM

Positional encoding !



Positional encodings: not that easy

- Unique encoding for each time-step
- Distance between two time-steps should be consistent across sentences with different lengths (cannot use a onehot)
- Should generalize to longer sentences than seen during training
- It’s values should be bounded (no integer)

Example: sinusoidal positional encoding

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

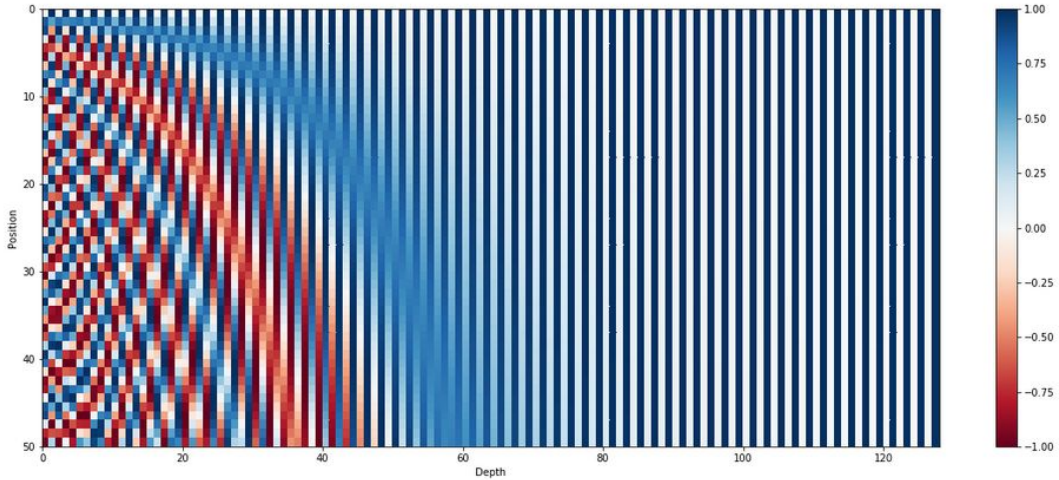


Figure 2 - The 128-dimensional positonal encoding for a sentence with the maximum lenght of 50. Each row represents the embedding vector $\overrightarrow{p_t}$

Example: sinusoidal positional encoding

$$\omega_k = \frac{1}{10000^{2k/d}}$$

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

Required: $t << T_{max}$ (here $T_{max} = 10000$)

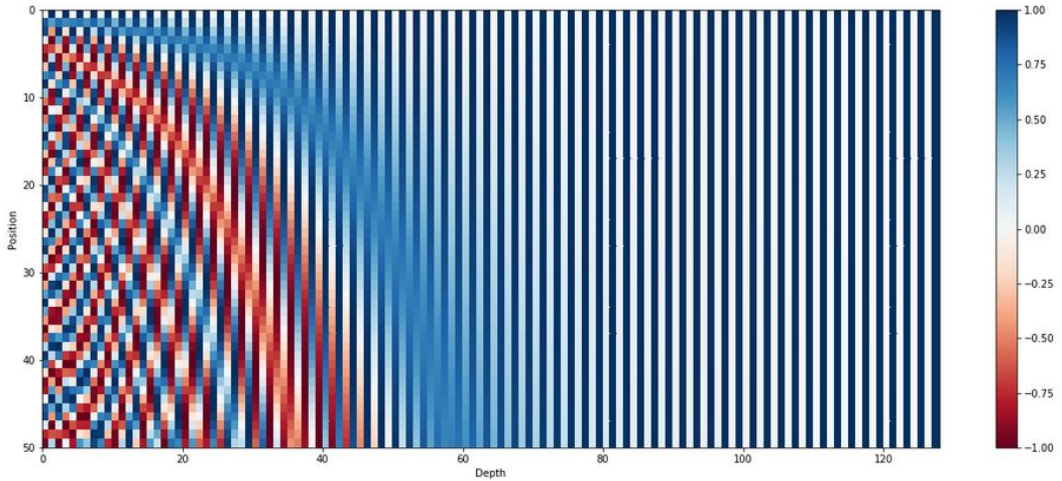
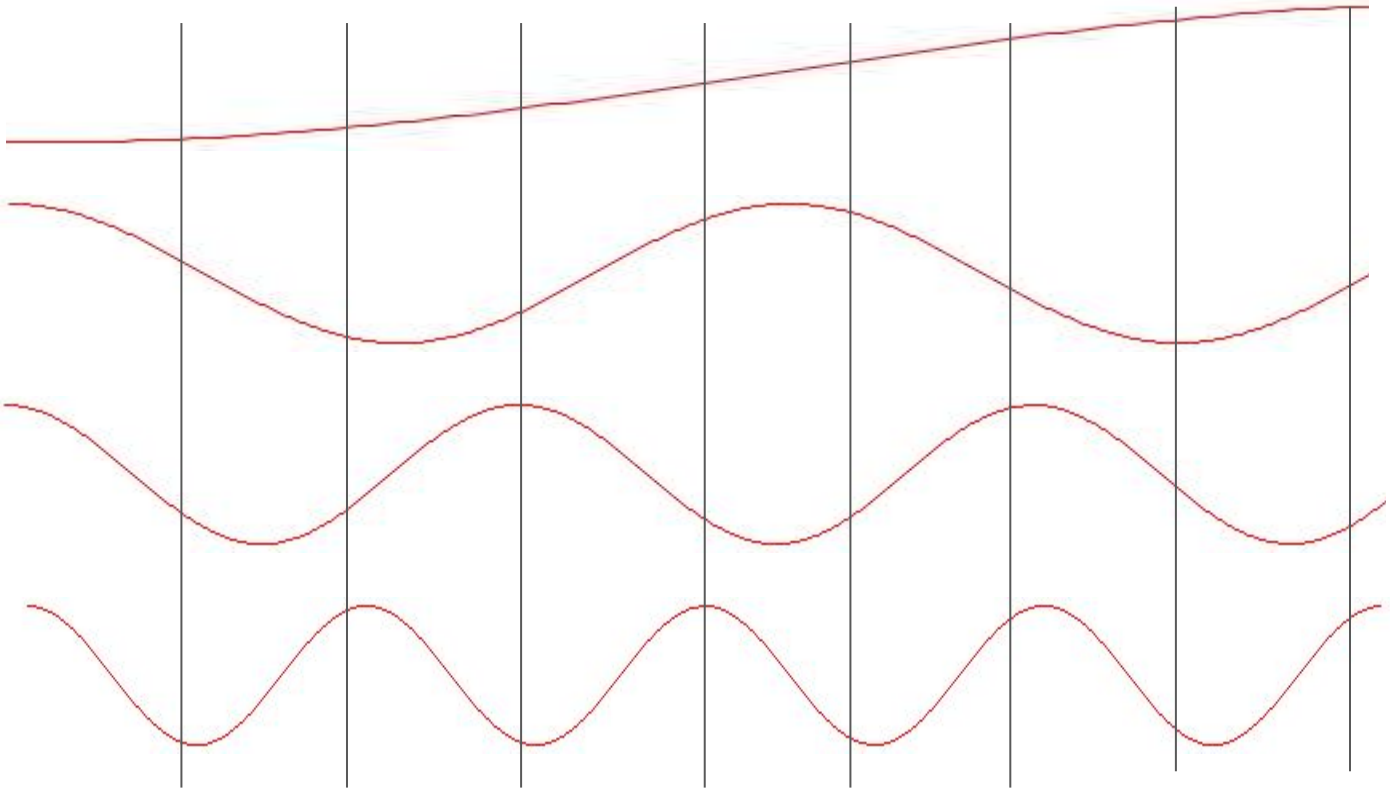
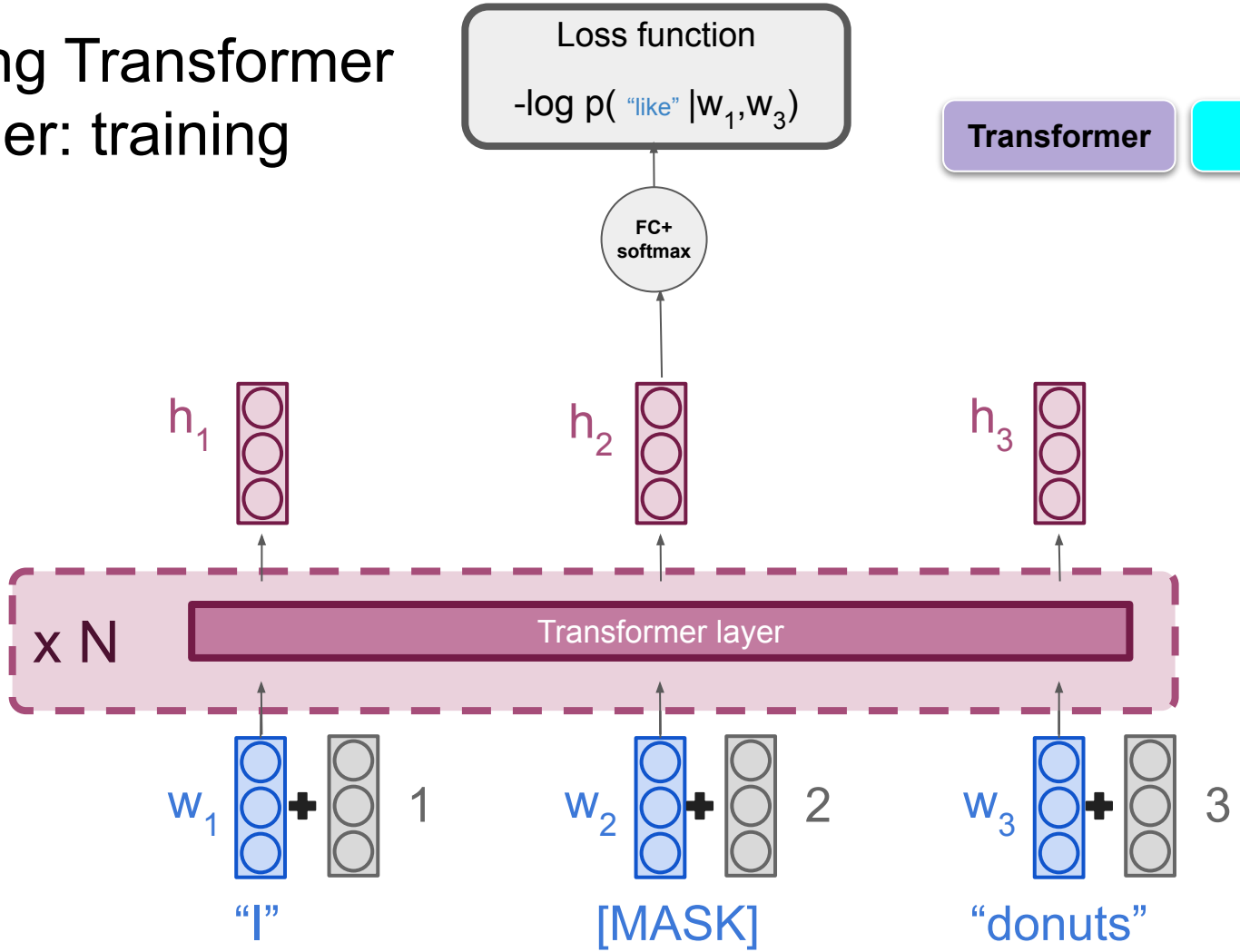


Figure 2 - The 128-dimensional positonal encoding for a sentence with the maximum lenght of 50. Each row represents the embedding vector $\overrightarrow{p_t}$

Visual interpretation



Training Transformer Encoder: training



Number of operations in a one layer transformer encoder ?

$$\begin{aligned}\phi_{MSA}(n,d) &= \phi_{qkv}(n,d) + \phi_A(n,d) + \phi_O(n,d) + \phi_{proj}(n,d) \\ &= 3nd^2 + n^2d + n^2d + nd^2 \\ &= 4nd^2 + 2n^2d.\end{aligned}$$

(multi-head self-attention is computed in four steps, see previous slide)

$$\phi_{MLP}(n,d) = \phi_{fc1}(n,d) + \phi_{fc2}(n,d) = 4nd^2 + 4nd^2 = 8nd^2.$$

(Fc1: D->4D and Fc2: 4D->D)

$$\phi_{BLK}(n,d) = \phi_{MSA}(n,d) + \phi_{MLP}(n,d) = 12nd^2 + 2n^2d.$$

(total number of operation)

So where is the bottleneck ? sentence length or embedding dimension?

Number of operations in a one layer transformer encoder ?

$$\begin{aligned}\phi_{MSA}(n,d) &= \phi_{qkv}(n,d) + \phi_A(n,d) + \phi_O(n,d) + \phi_{proj}(n,d) \\ &= 3nd^2 + n^2d + n^2d + nd^2 \\ &= 4nd^2 + 2n^2d.\end{aligned}$$

(multi-head self-attention is computed in four steps, see previous slide)

$$\phi_{MLP}(n,d) = \phi_{fc1}(n,d) + \phi_{fc2}(n,d) = 4nd^2 + 4nd^2 = 8nd^2.$$

(Fc1: D->4D and Fc2: 4D->D)

$$\phi_{BLK}(n,d) = \phi_{MSA}(n,d) + \phi_{MLP}(n,d) = 12nd^2 + 2n^2d.$$

(total number of operation)

So where is the bottleneck ? sentence length or embedding dimension?
It depends! Very long sentences are a no go but large LM have d>>n, so d is the bottleneck

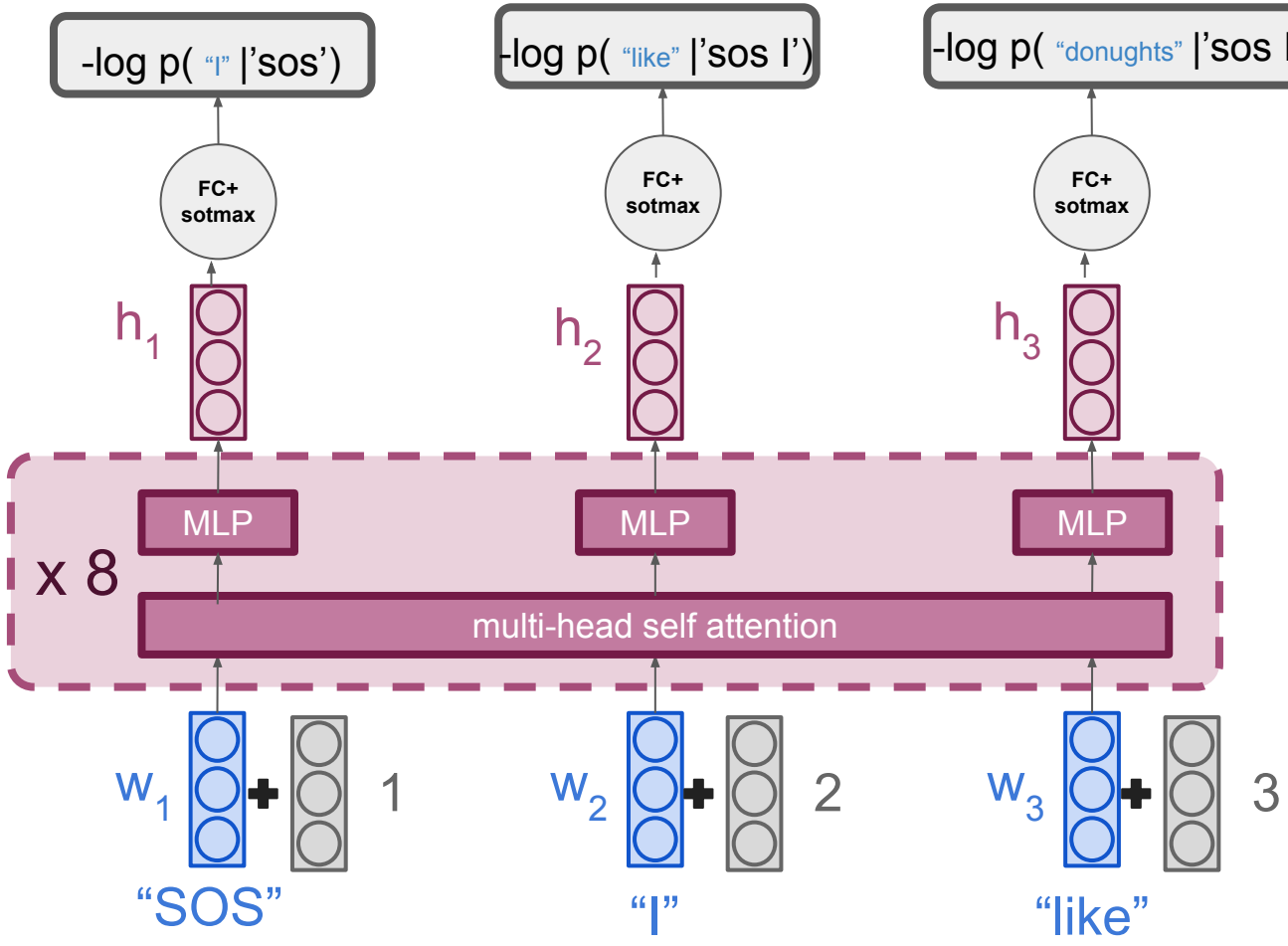
Transformer generative

Transformer Decoder

Same architecture as encoder, as for RNN, with one **small** yet **crucial** change: the causal mask on the attention matrices.

Transformer Decoder Training

Transformer | generative

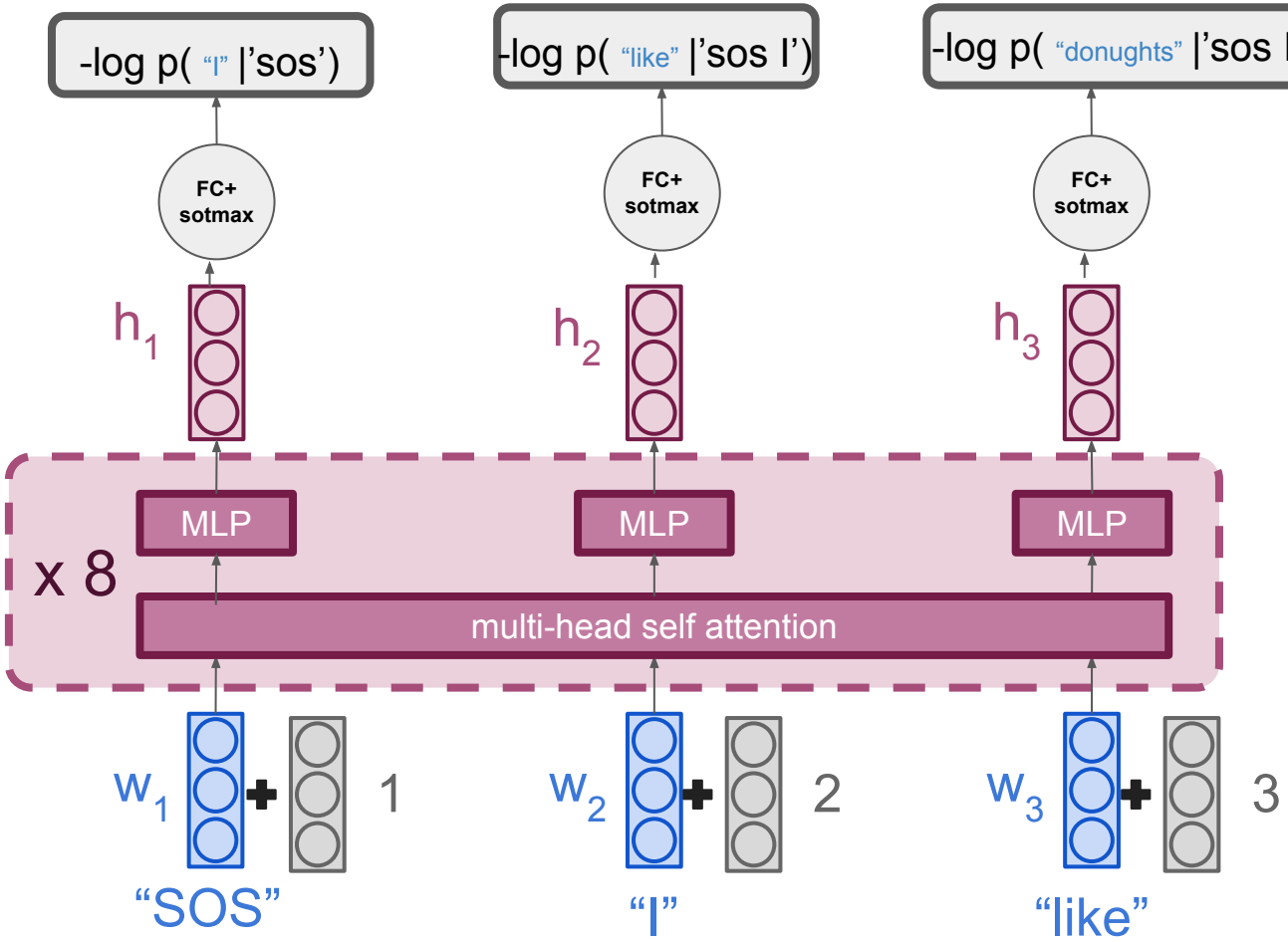


Trained to predict the next word

"doughnuts"

Transformer Decoder Training

Transformer | generative

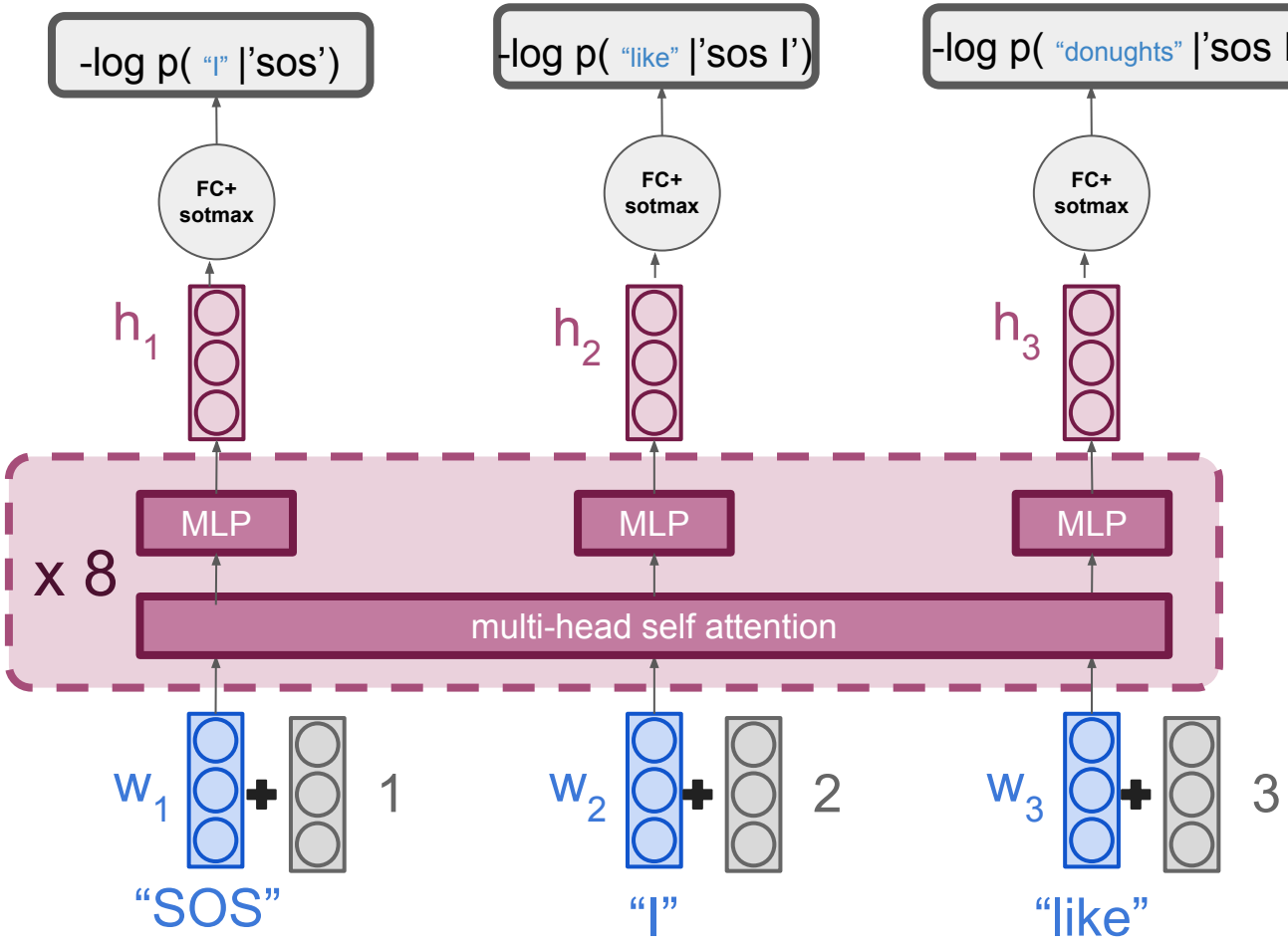


Yet, we cannot do this, why?

"doughnuts"

Transformer Decoder Training

Transformer | generative

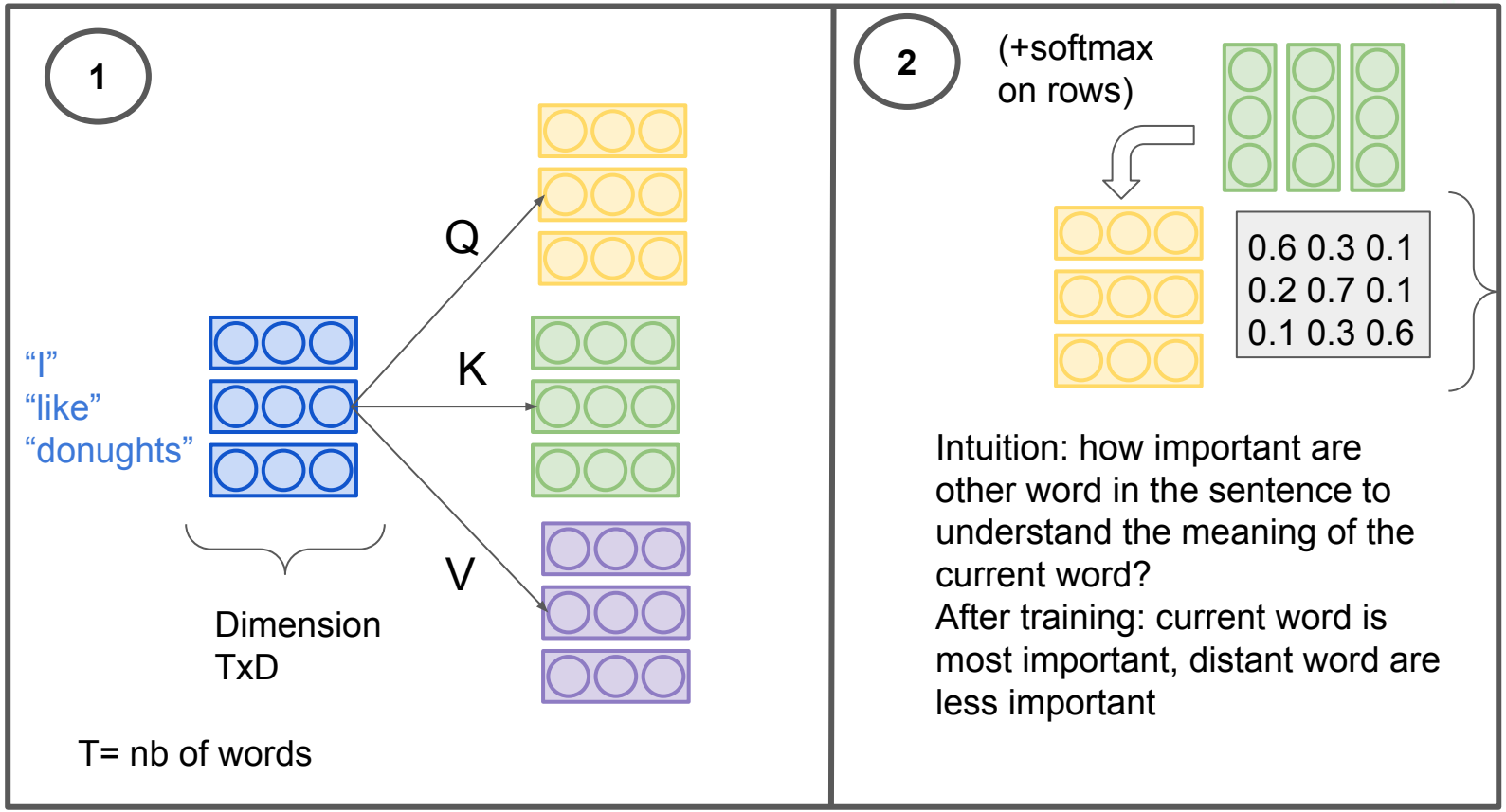


Yet, we cannot do this, why?
MSA goes both ways! We need to prevent the transformer from attending futur tokens

"doughnuts"

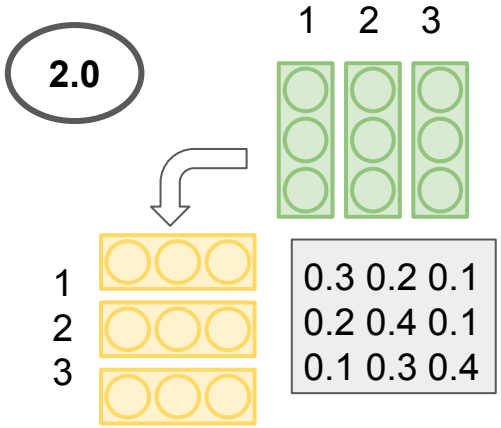
Self-Attention* in 3 steps

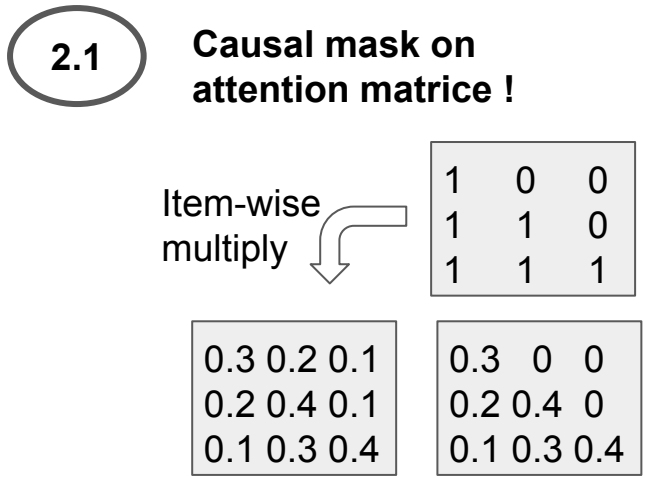
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

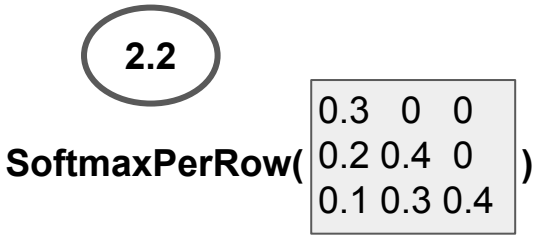


*Vaswani et al. Attention is all you need.

How to make a transformer causal ? Modify step 2

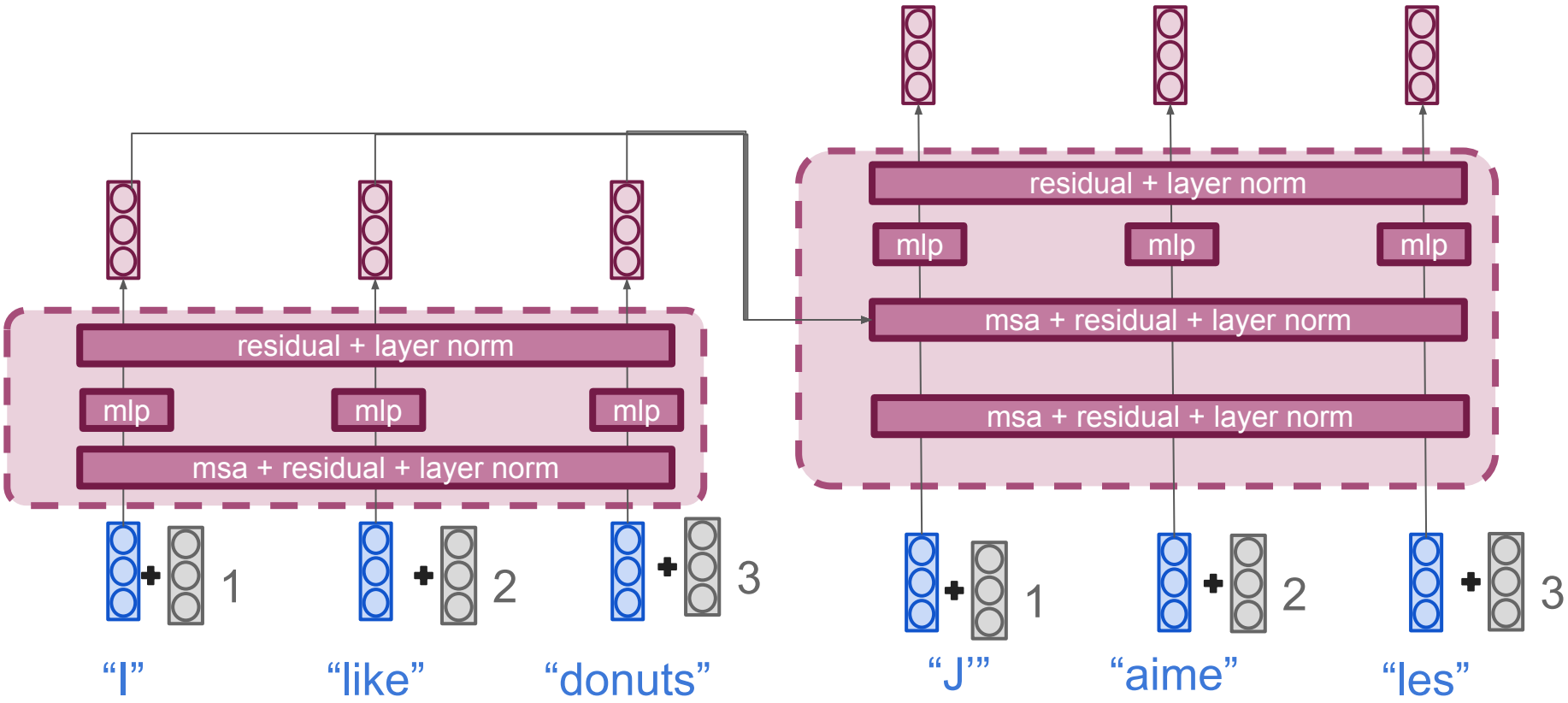






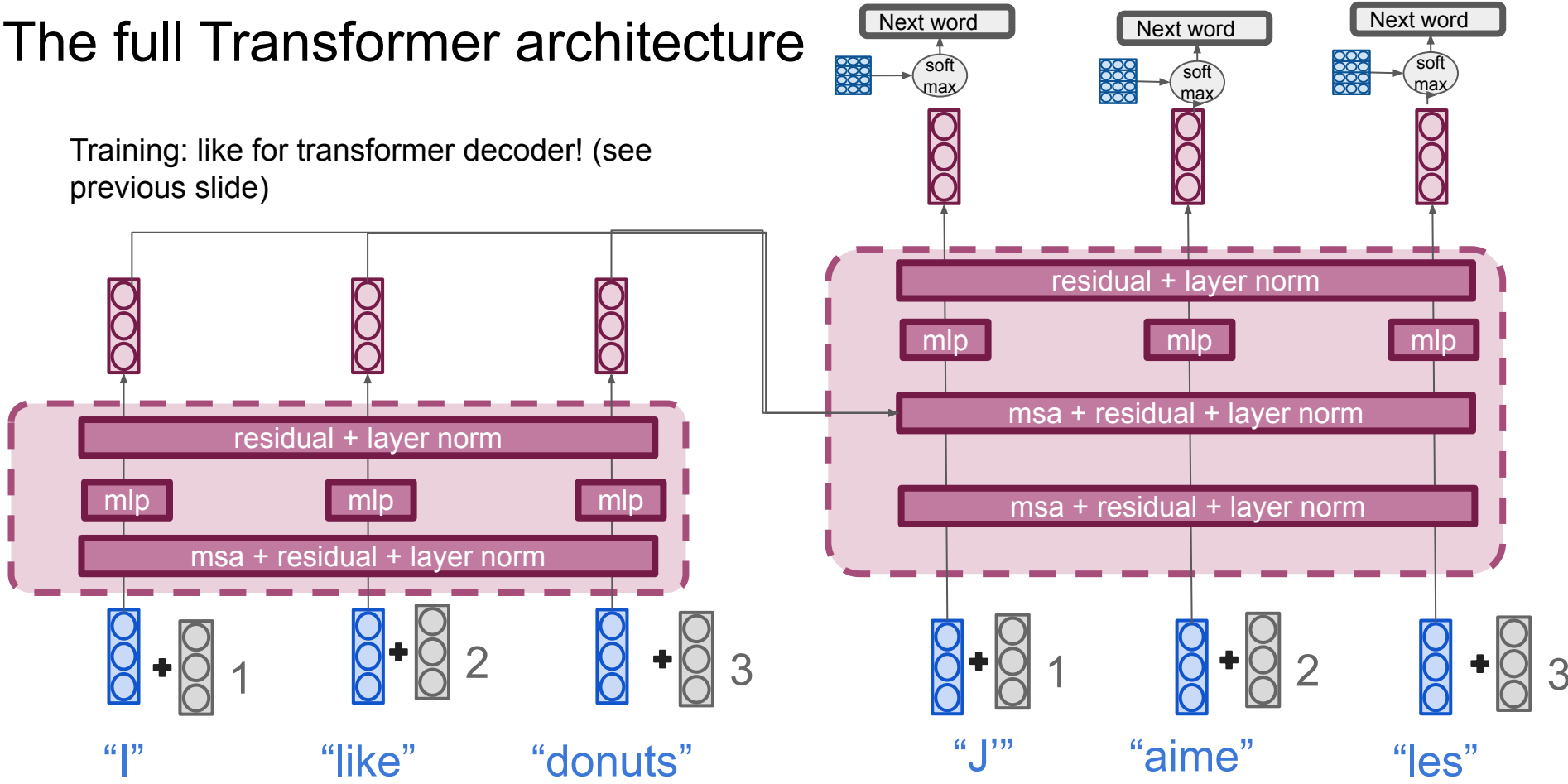
The full Transformer architecture

Transformer seq2seq



The full Transformer architecture

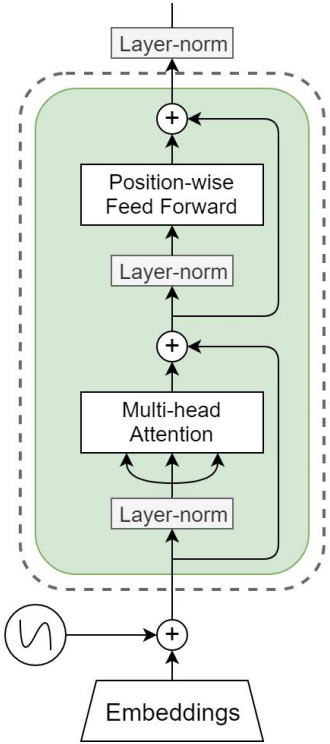
Training: like for transformer decoder! (see previous slide)



Transformers In a Nutshell

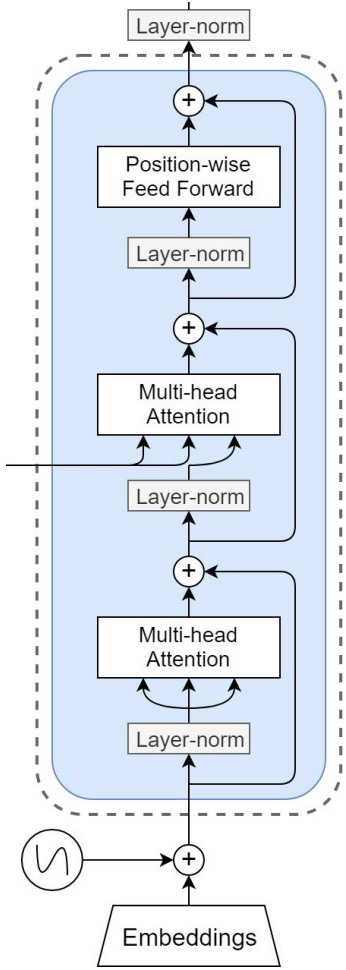
Transformer

- Easy parallelism
  - Entire sequences can be processed at once
  - Good for GPU
- Easier to attend to distant context
  - Direct path from one position to the other
- Several architectural details left out
  - Residual connections
  - Layer normalization
  - Decoder architecture in the full Transformer architecture (Encoder+Decoder)



Transformers In a Nutshell

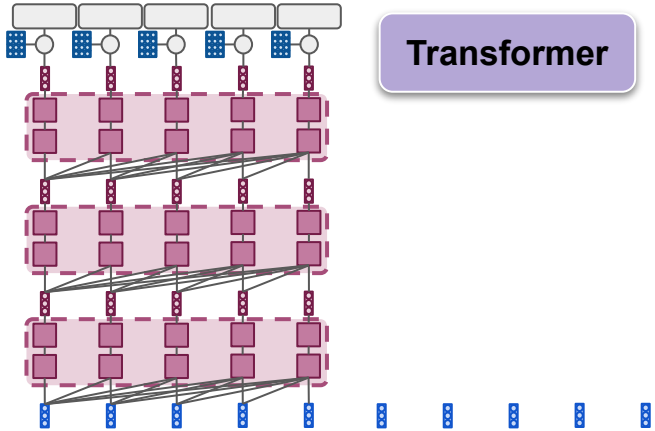
- Easy parallelism
  - Entire sequences can be processed at once
  - Good for GPU
- Easier to attend to distant context
  - Direct path from one position to the other
- Several architectural details left out
  - Residual connections
  - Layer normalization
  - Decoder architecture in the full Transformer architecture (Encoder+Decoder)



Transformer

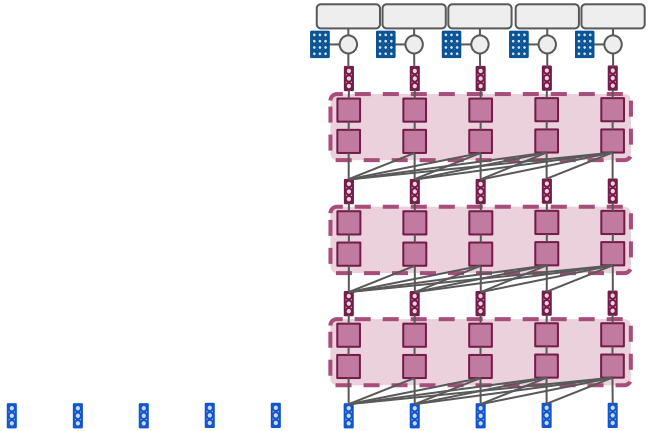
Downsides of Transformers

- Fixed context window
  - Back to the Markov assumption?



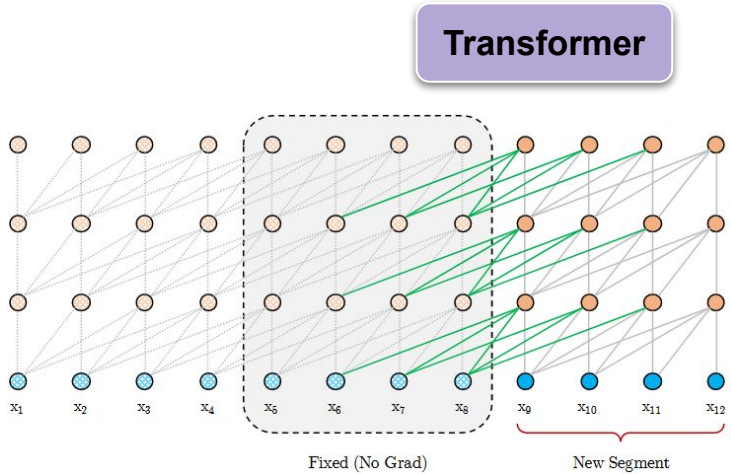
Downsides of Transformers

- Fixed context window
  - Back to the Markov assumption?



Downsides of Transformers

- Fixed context window
  - Back to the Markov assumption?
  - But there are workarounds (Transformer-XL)

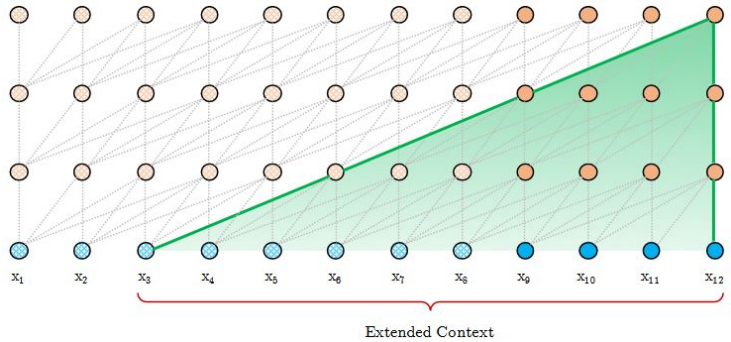


From Dai et al. (2019)

Downsides of Transformers

Transformer

- Fixed context window
  - Back to the markov assumption?
  - But there are workarounds (Transformer-XL)
  - Effective context-length ~700

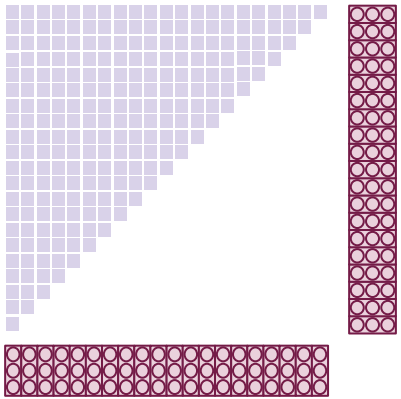


From Dai et al. (2019)

Downsides of Transformers

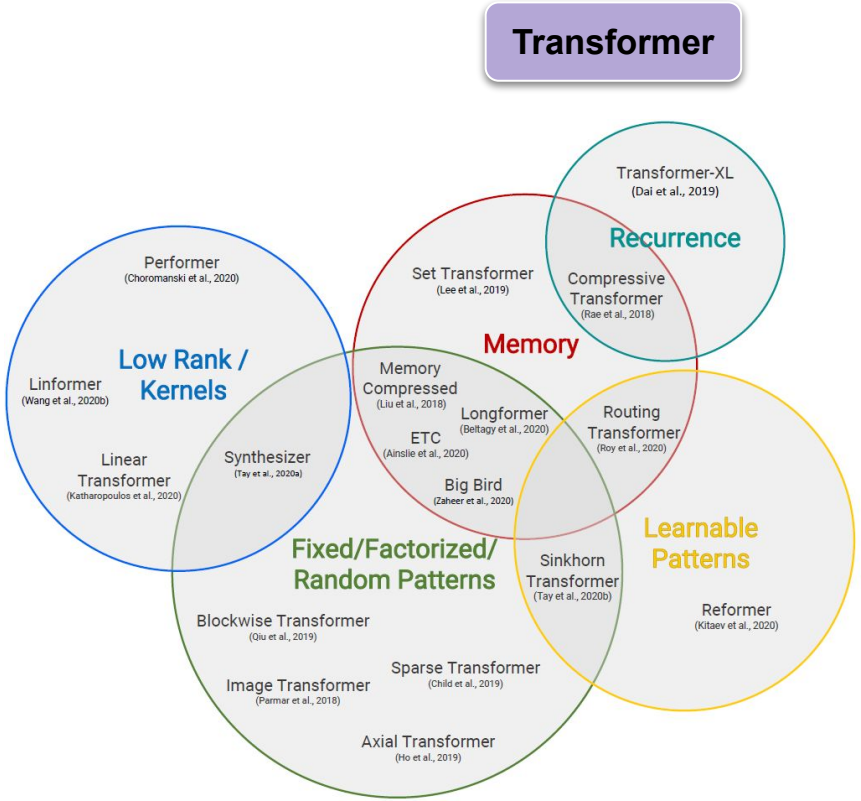
Transformer

- Fixed context window
  - Back to the Markov assumption?
  - But there are workarounds (Transformer-XL)
  - Effective context-length ~700
- Compute-intensive for long sequences
  - Self-attention is quadratic in sequence length



Downsides of Transformers

- Fixed context window
  - Back to the Markov assumption?
  - But there are workarounds (Transformer-XL)
  - Effective context-length ~700
- Compute-intensive for long sequences
  - Self-attention is quadratic in sequence length
  - Lots of work on making attention more efficient



From Tay et al. (2020)

Downsides of Transformers

Transformer

- Fixed context window
  - Back to the Markov assumption?
  - But there are workarounds (Transformer-XL)
  - Effective context-length ~700
- Compute-intensive for long sequences
  - Self-attention is quadratic in sequence length
  - Lots of work on making attention more efficient
- Tricky to train, especially on small corpora

Downsides of Transformers

Transformer

- Fixed context window
  - Back to the Markov assumption?
  - But there are workarounds (Transformer-XL)
  - Effective context-length ~700
- Compute-intensive for long sequences
  - Self-attention is quadratic in sequence length
  - Lots of work on making attention more efficient
- Tricky to train, especially on small corpora
  - Requires large batch size, tuned optimizer, learning rate schedule…
  - Doesn't work very well below <4 layers

Cosine learning rate schedule with warmup