Large Scale Machine Learning

Natural Language Processing

Adeline Fermanian
adeline.fermanian@mines-paristech.fr

March 2023

Mines ParisTech - PSL

Acknowledgement

Slides inspired by

- Édouard Grave
- Claire Boyer
- fidle-cnrs
- Charles Deledalle's lectures

Motivation

- Why NLP ?

Motivation

- Why NLP ?
  - Process, analyze and/or produce natural language

Motivation

- Why NLP ?
  - Process, analyze and/or produce natural language
  - Interact with computers using natural language

Motivation

- Why NLP ?
  - Process, analyze and/or produce natural language
  - Interact with computers using natural language
- Many applications (lot of information in text):

Motivation

- Why NLP ?
  - Process, analyze and/or produce natural language
  - Interact with computers using natural language
- Many applications (lot of information in text):
  - text classification, spam detection, topic identification

Motivation

- Why NLP ?
  - Process, analyze and/or produce natural language
  - Interact with computers using natural language
- Many applications (lot of information in text):
  - text classification, spam detection, topic identification
  - machine translation

Motivation

- Why NLP ?
  - Process, analyze and/or produce natural language
  - Interact with computers using natural language
- Many applications (lot of information in text):
  - text classification, spam detection, topic identification
  - machine translation
  - information retrieval, web search

Motivation

- Why NLP ?
  - Process, analyze and/or produce natural language
  - Interact with computers using natural language
- Many applications (lot of information in text):
  - text classification, spam detection, topic identification
  - machine translation
  - information retrieval, web search
  - medical records, scientific articles

Motivation

- Why NLP ?
  - Process, analyze and/or produce natural language
  - Interact with computers using natural language
- Many applications (lot of information in text):
  - text classification, spam detection, topic identification
  - machine translation
  - information retrieval, web search
  - medical records, scientific articles
- Large scale? Wikipedia: 3B words, Common Crawl: 24TB

Text classification: is this spam?

Text classification: is this review positive?

Text classification: what is this article about?

Enriching Word Vectors with Subword Information

Piotr Bojanowski* and **Edouard Grave*** and **Armand Joulin** and **Tomas Mikolov**
Facebook AI Research
{bojanowski,egrave,ajoulin,tmikolov}@fb.com

Abstract

Continuous word representations, trained on large unlabeled corpora are useful for many natural language processing tasks. Popular models that learn such representations ignore the morphology of words, by assigning a distinct vector to each word. This is a limitation, especially for languages with large vocabularies and many rare words. In this paper, we propose a new approach based on the skipgram

et al., 2010; Baroni and Lenci, 2010). In the neural network community, Collobert and Weston (2008) proposed to learn word embeddings using a feed-forward neural network, by predicting a word based on the two words on the left and two words on the right. More recently, Mikolov et al. (2013b) proposed simple log-bilinear models to learn continuous representations of words on very large corpora efficiently.

Large Language Models: ChatGPT

AD

Hello ChatGPT! What are you useful for?

Hello! As a language model, I am designed to be useful for a wide variety of tasks that involve natural language processing, such as:

1. Answering questions: I can provide information on a broad range of topics, from science and history to entertainment and current events.
2. Generating text: I can help generate text for a variety of purposes, such as writing essays, creating social media posts, or even composing poetry.
3. Translation: I can translate text from one language to another, although my proficiency may vary depending on the language pair.
4. Personalized Assistance: I can provide personalized assistance such as setting up reminders, scheduling appointments and more.
5. Learning: I can help people learn new things through answering questions, providing explanations, and suggesting resources.

Overall, I am designed to be a helpful resource for anyone who needs assistance with tasks that involve natural language processing.

Sommaire

Word representation

Recurrent NN

Transformers

Tokenization

- How to represent words as input?

“I’ve never seen a movie like this before”

Tokenization

- How to represent words as input?

  “I’ve never seen a movie like this before”

- Split into tokens:

  [“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

Tokenization

- How to represent words as input?

  “I’ve never seen a movie like this before”

- Split into tokens:

  [“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

- Issues for tokenization:

Tokenization

- How to represent words as input?

  “I’ve never seen a movie like this before”

- Split into tokens:

  [“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

- Issues for tokenization:
  - *I’ve* $\rightarrow$ [I’ve] or [I] [have] or [I] [’ve] ?

Tokenization

- How to represent words as input?

  “I’ve never seen a movie like this before”

- Split into tokens:

  [“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

- Issues for tokenization:
  - *I’ve* $\rightarrow$ [I’ve] or [I] [have] or [I] [’ve] ?
  - *low-frequency* $\rightarrow$ [low-frequency] or [low] [frequency] ?

Tokenization

- How to represent words as input?

  “I’ve never seen a movie like this before”

- Split into tokens:

  [“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

- Issues for tokenization:
  - *I’ve* $\rightarrow$ [I’ve] or [I] [have] or [I] [’ve] ?
  - *low-frequency* $\rightarrow$ [low-frequency] or [low] [frequency] ?
  - Some arbitrary choices: be consistent!

Tokenization

- How to represent words as input?

  “I’ve never seen a movie like this before”

- Split into tokens:

  [“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

- Issues for tokenization:
  - *I’ve* $\rightarrow$ [I’ve] or [I] [have] or [I] [’ve] ?
  - *low-frequency* $\rightarrow$ [low-frequency] or [low] [frequency] ?
  - Some arbitrary choices: be consistent!
  - Language-dependant!

One-hot encoding

Dictionary

0	a
1	before
2	fantastic
3	i’ve
4	is
5	like
6	movie
7	never
8	seen
9	this

[“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

One-hot encoding

Dictionary

0	a
1	before
2	fantastic
3	i've
4	is
5	like
6	movie
7	never
8	seen
9	this

["I've", "never","seen", "a","movie", "like", "this", "before"]

[3, 7, 8, 0, 6, 5, 9, 1]

One-hot encoding

Dictionary

0	a
1	before
2	fantastic
3	i've
4	is
5	like
6	movie
7	never
8	seen
9	this

["I've", "never","seen", "a","movie", "like", "this", "before"]

[3, 7, 8, 0, 6, 5, 9, 1]

- The values associated with each word are meaningless: words with a contiguous subscript are typically unrelated.

One-hot encoding

Dictionary

0	a
1	before
2	fantastic
3	i've
4	is
5	like
6	movie
7	never
8	seen
9	this

["I've", "never","seen", "a","movie", "like", "this", "before"]

[3, 7, 8, 0, 6, 5, 9, 1]

- The values associated with each word are meaningless: words with a contiguous subscript are typically unrelated.
- Solution: vectorize!

One-hot encoding

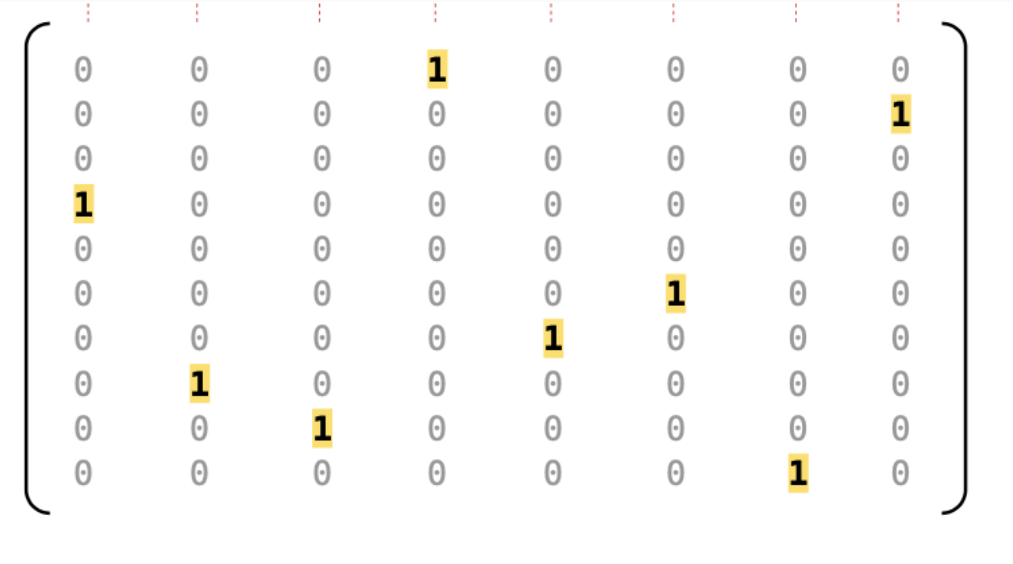
[“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

[3, 7, 8, 0, 6, 5, 9, 1]

One-hot encoding

[“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

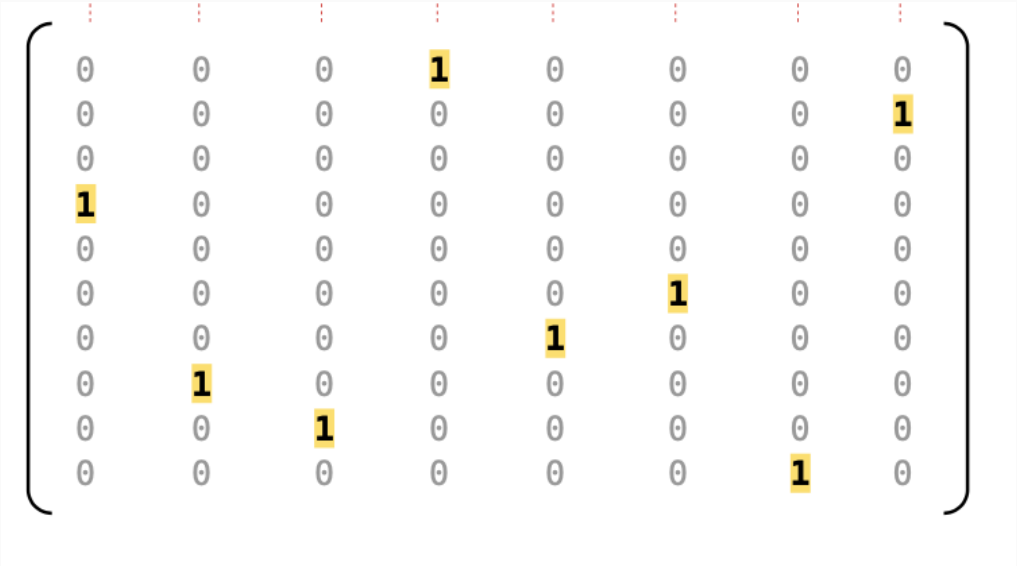
[3, 7, 8, 0, 6, 5, 9, 1]



One-hot encoding

[“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

[3, 7, 8, 0, 6, 5, 9, 1]

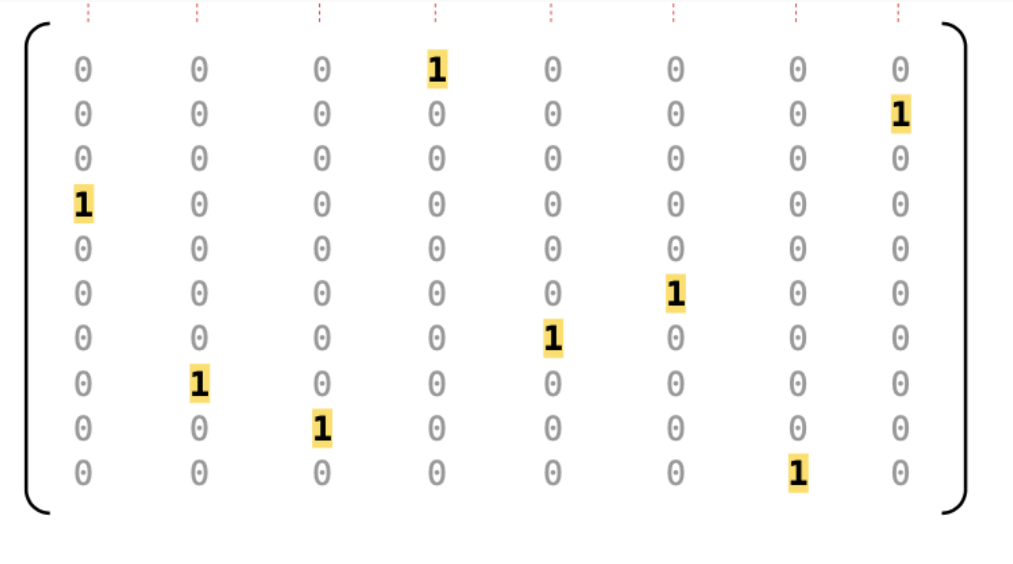


- Each word has its own dimension. Limits: size!

One-hot encoding

[“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

[3, 7, 8, 0, 6, 5, 9, 1]



- Each word has its own dimension. Limits: size!

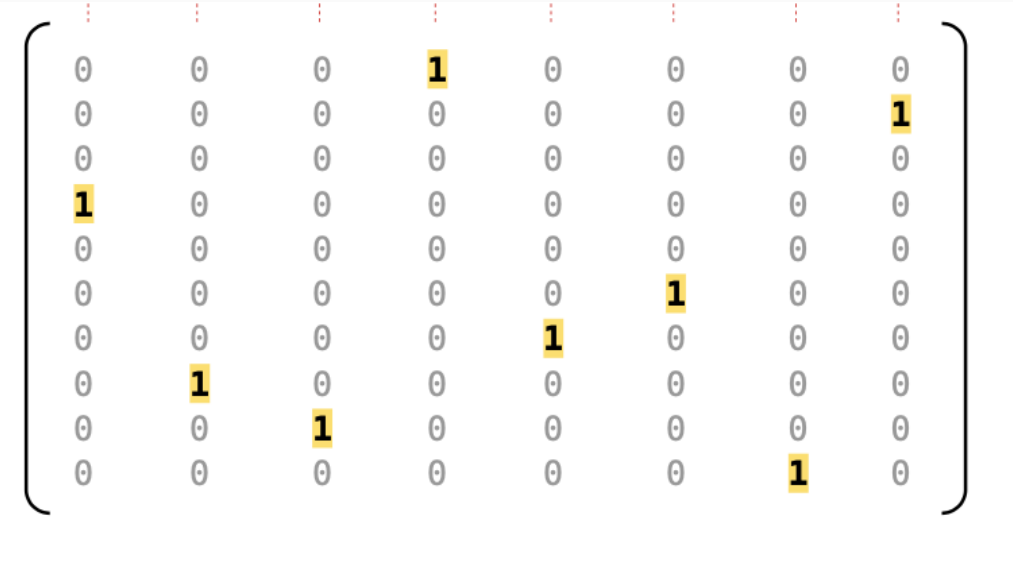
Example

- Dictionary of 80 000 words, text of 300 words

One-hot encoding

[“I’ve”, “never”,“seen”, “a”,“movie”, “like”, “this”, “before”]

[3, 7, 8, 0, 6, 5, 9, 1]

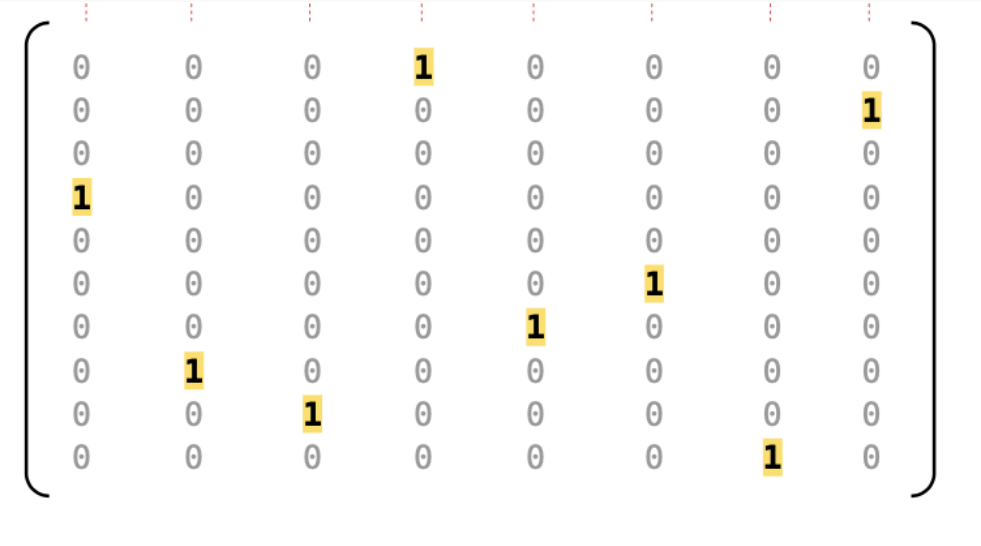


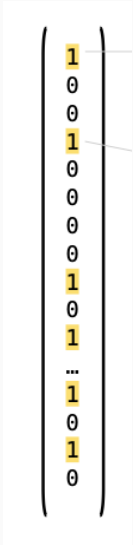
- Each word has its own dimension. Limits: size!

Example

- Dictionary of 80 000 words, text of 300 words
- Memory: 24.10^6 parameters to store the text!

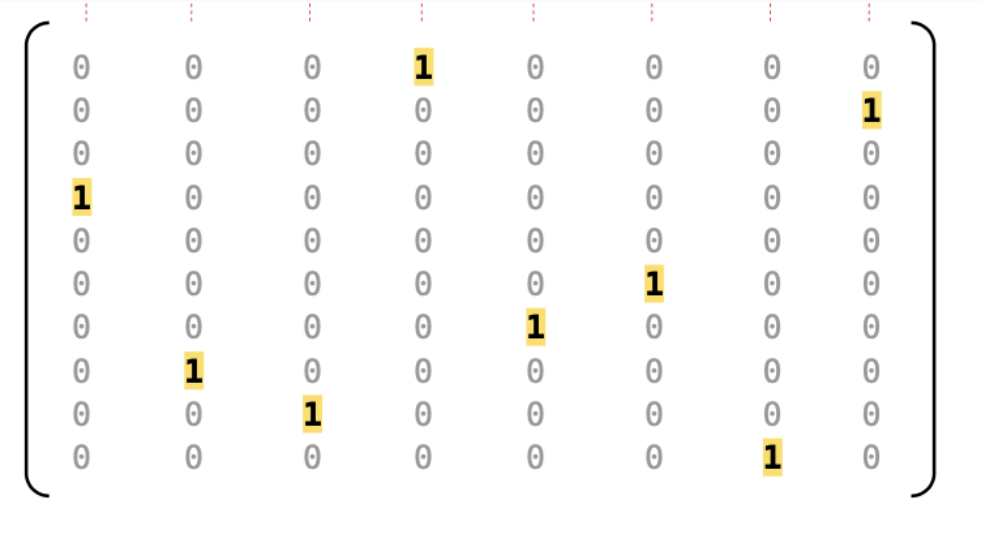
One-hot vector

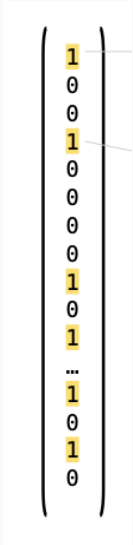




- 1 if word is present in the sentence

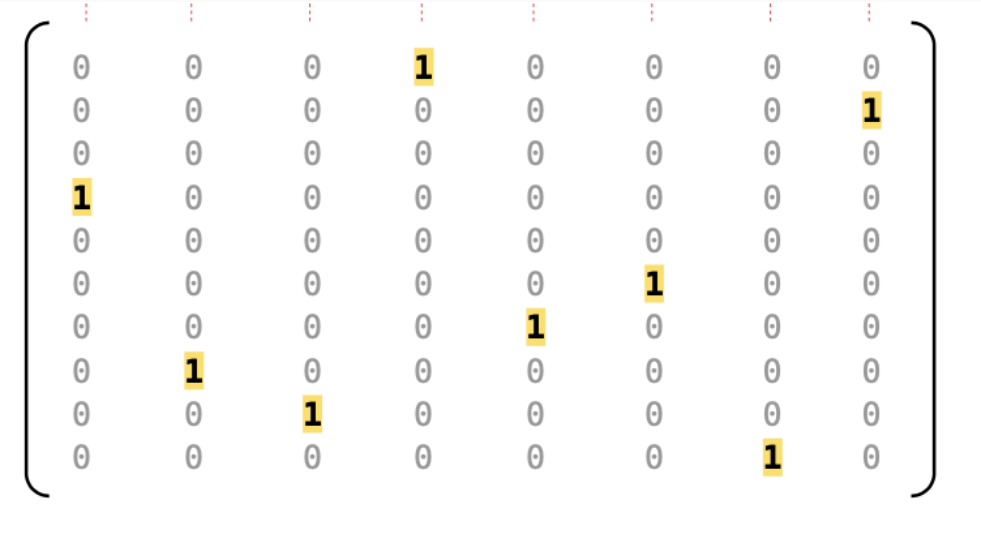
One-hot vector

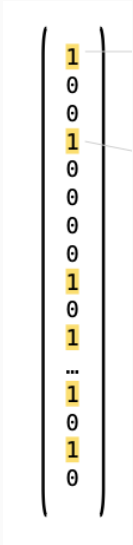




- 1 if word is present in the sentence
- Size does not depend on length of the sentence

One-hot vector





- 1 if word is present in the sentence
- Size does not depend on length of the sentence
- But: lose the words order

Embedding layer

- Goal: sparse word vector $\rightarrow$ short dense vector.

Embedding layer

- Goal: sparse word vector $\rightarrow$ short dense vector.
- Learned for a classification task: one layer of a neural network.

Embedding layer

- Goal: sparse word vector $\rightarrow$ short dense vector.
- Learned for a classification task: one layer of a neural network.
- Size of the embedding: hyperparameter

Embedding layer

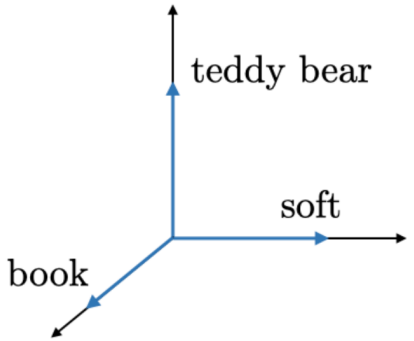
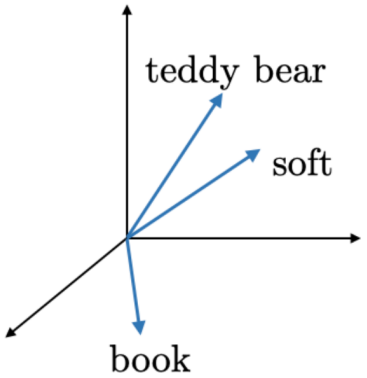
- Goal: sparse word vector $\rightarrow$ short dense vector.
- Learned for a classification task: one layer of a neural network.
- Size of the embedding: hyperparameter

Example

- Text of 300 words
- Embedding size: 200
- Memory: 60 000 parameters to store the text!

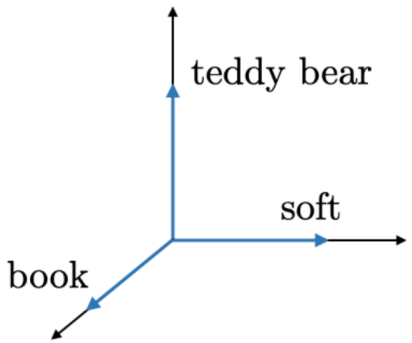
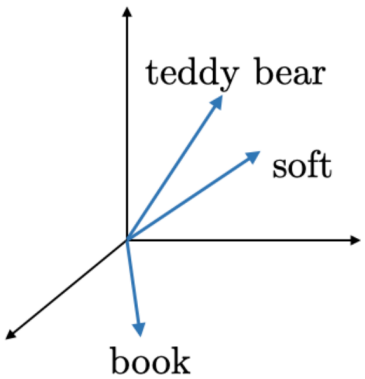
Word embedding

- Goal: obtain a word embedding with semantic meaning (and not on a classification task)

1-hot representation	Word embedding
	

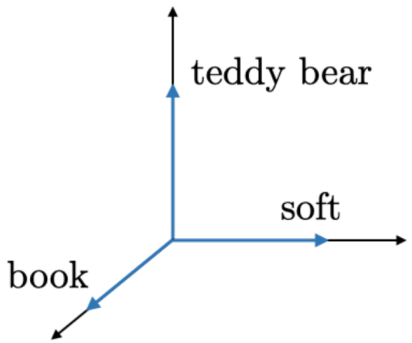
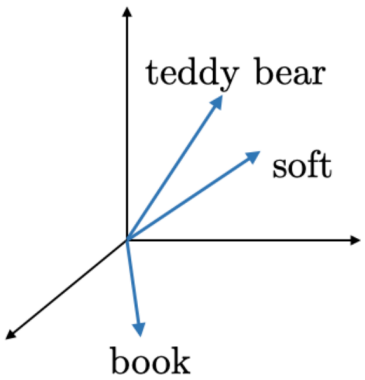
Word embedding

- Goal: obtain a word embedding with semantic meaning (and not on a classification task)
- Word2Vec (Mikolov et al., 2013)

1-hot representation	Word embedding
	

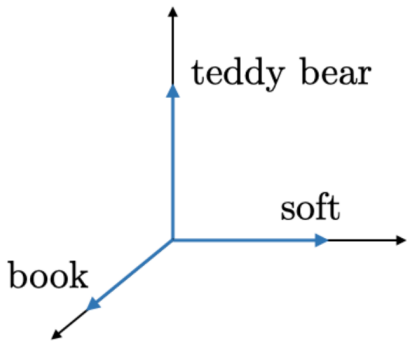
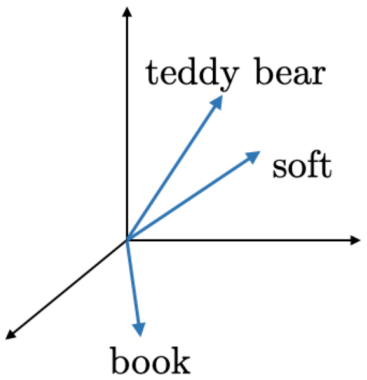
Word embedding

- Goal: obtain a word embedding with semantic meaning (and not on a classification task)
- Word2Vec (Mikolov et al., 2013)
- Dictionaries built from large corpora are open-source:

1-hot representation	Word embedding
	

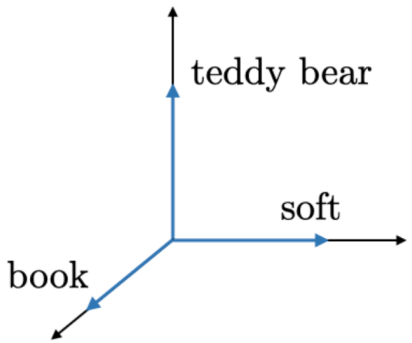
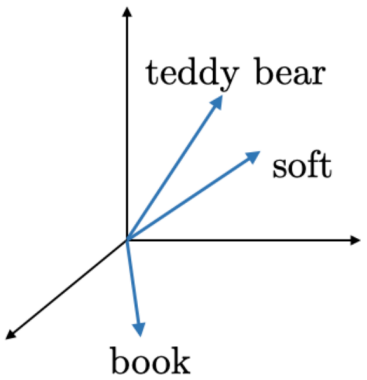
Word embedding

- Goal: obtain a word embedding with semantic meaning (and not on a classification task)
- Word2Vec (Mikolov et al., 2013)
- Dictionaries built from large corpora are open-source:
  - Continuous Bag-of-Words (CBOW)

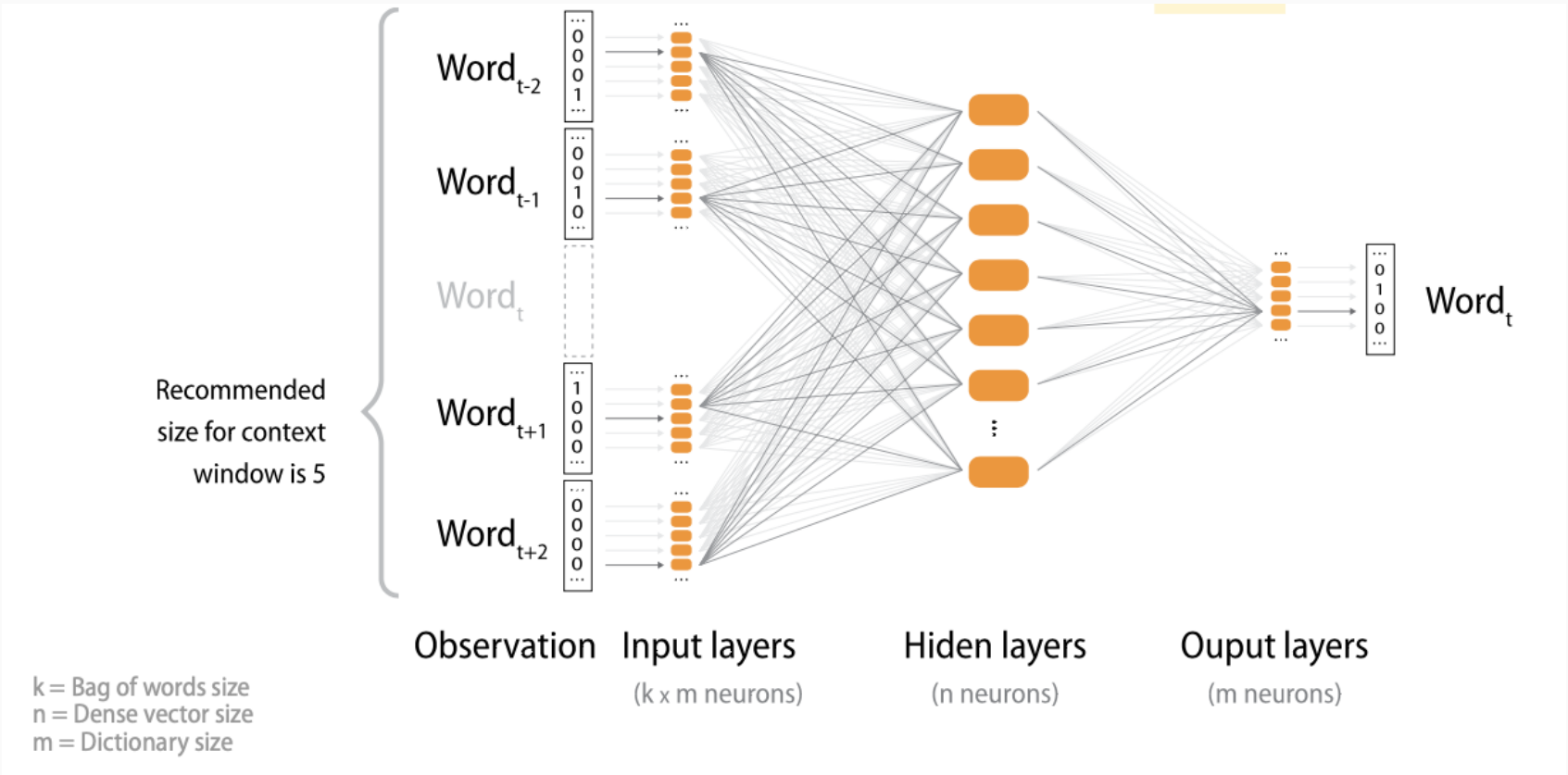
1-hot representation	Word embedding
	

Word embedding

- Goal: obtain a word embedding with semantic meaning (and not on a classification task)
- Word2Vec (Mikolov et al., 2013)
- Dictionaries built from large corpora are open-source:
  - Continuous Bag-of-Words (CBOW)
  - Skip-Gram (SG)

1-hot representation	Word embedding
	

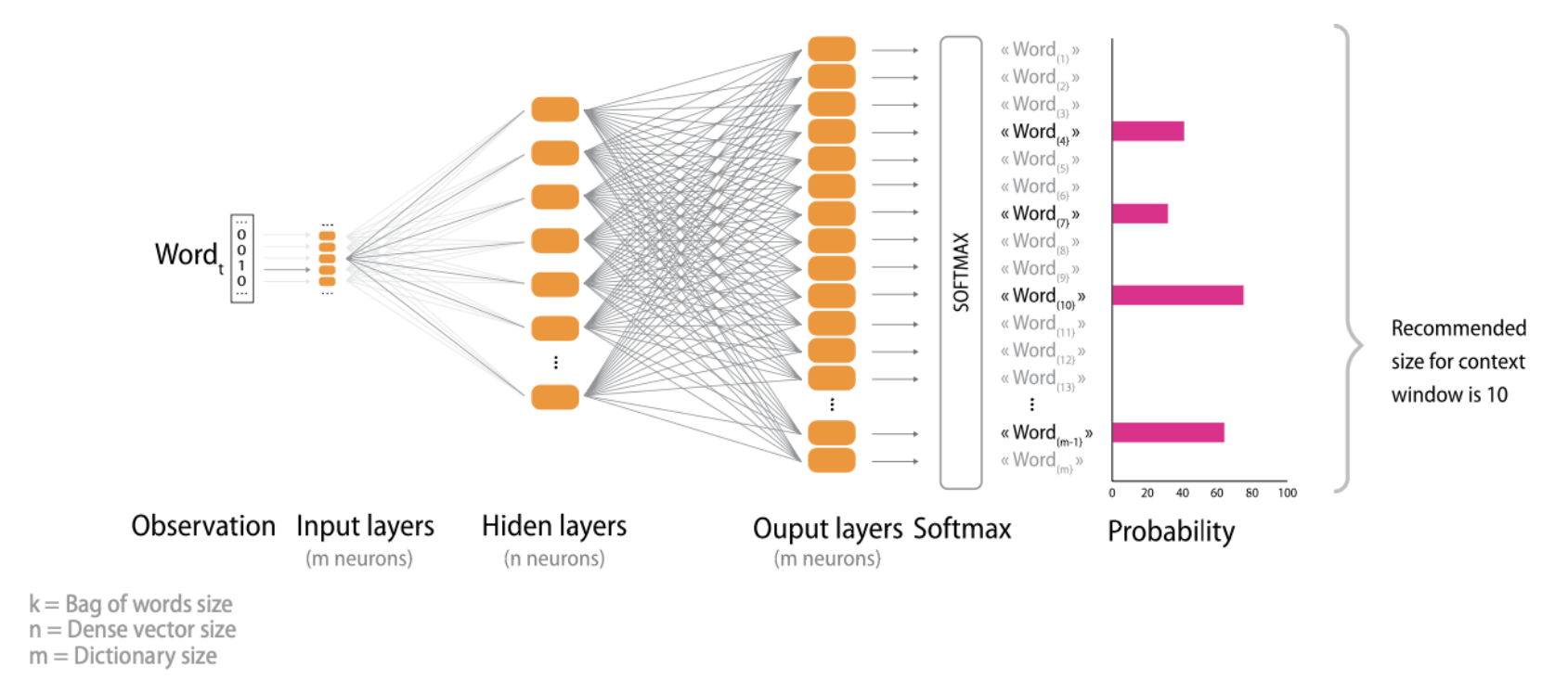
CBOW



source: fidle-cnrs

- Objective: find a word from its context

Skip-Gram



source: fidle-cnrs

- Objective: find the context from a word

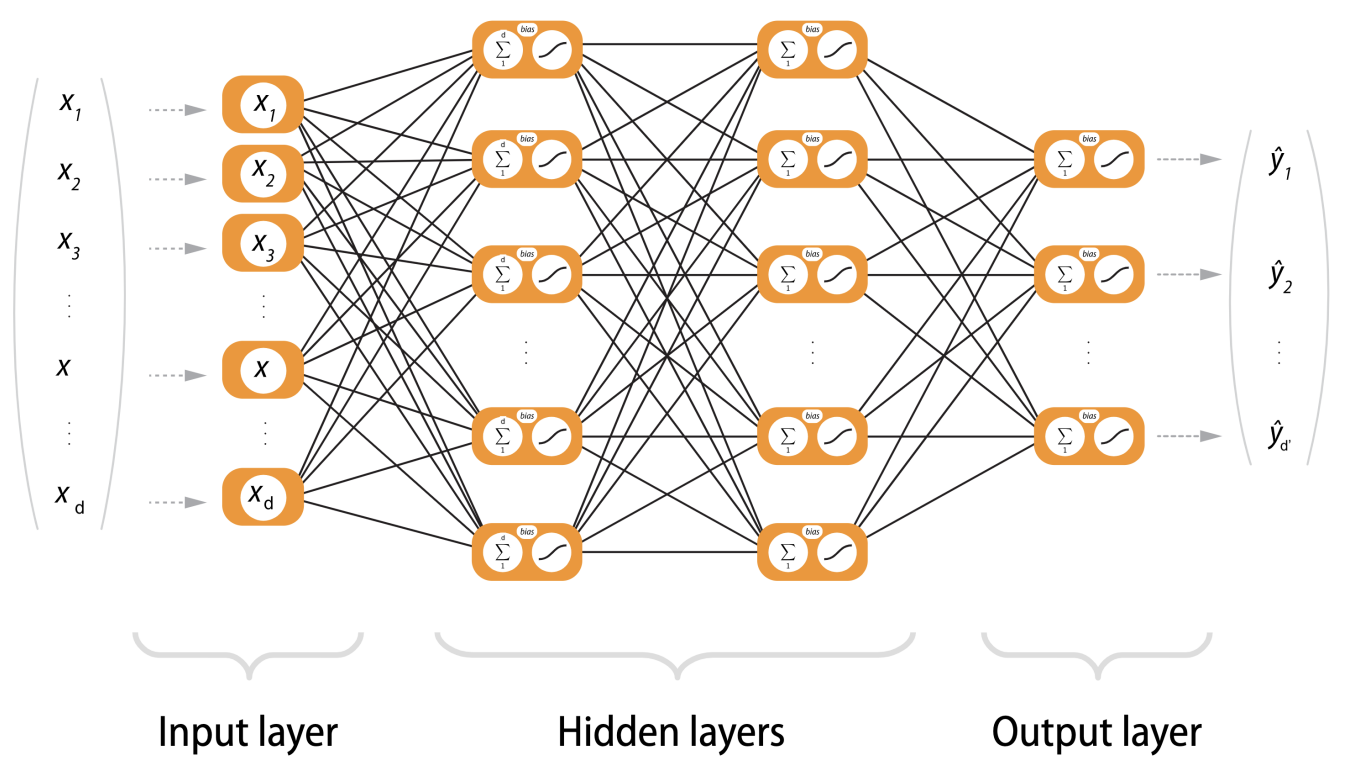
Sommaire

Word representation

Recurrent NN

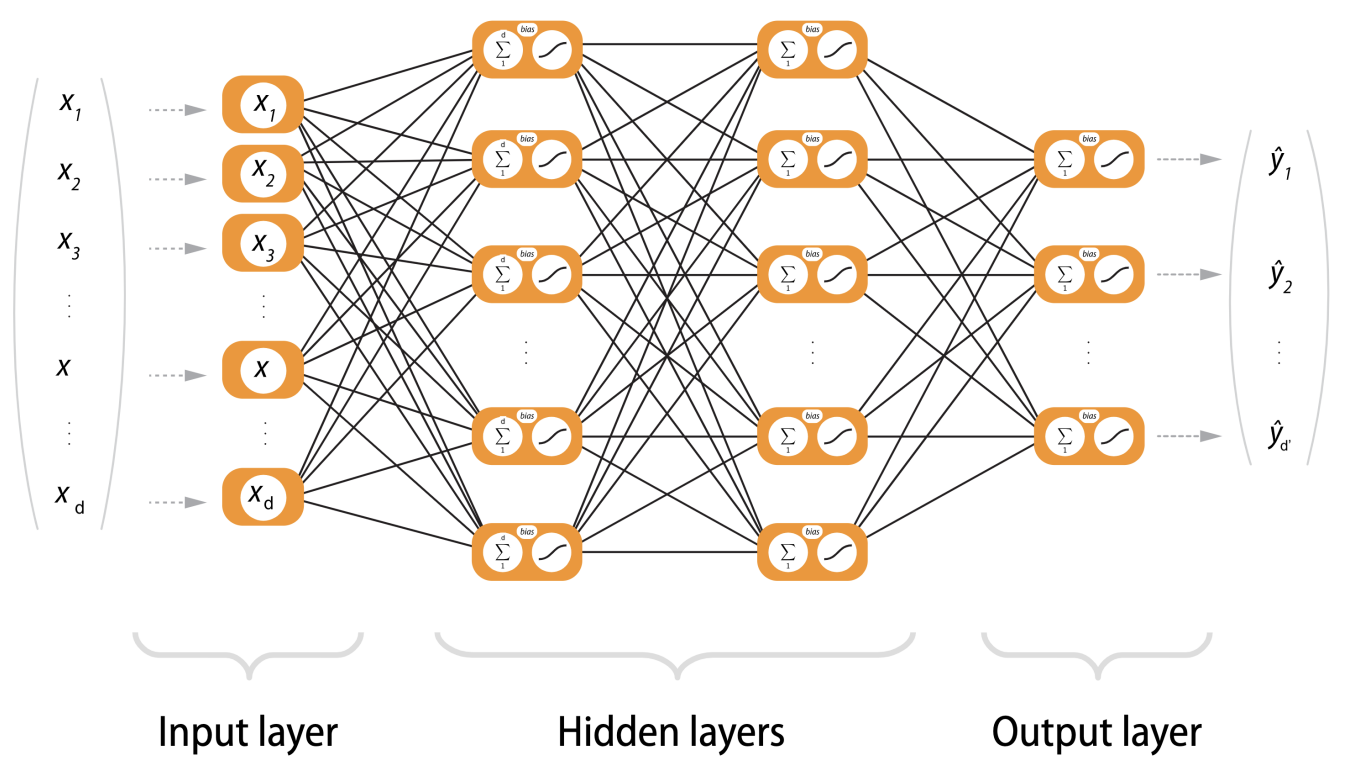
Transformers

Recall: neural networks



source: fidle-cnrs

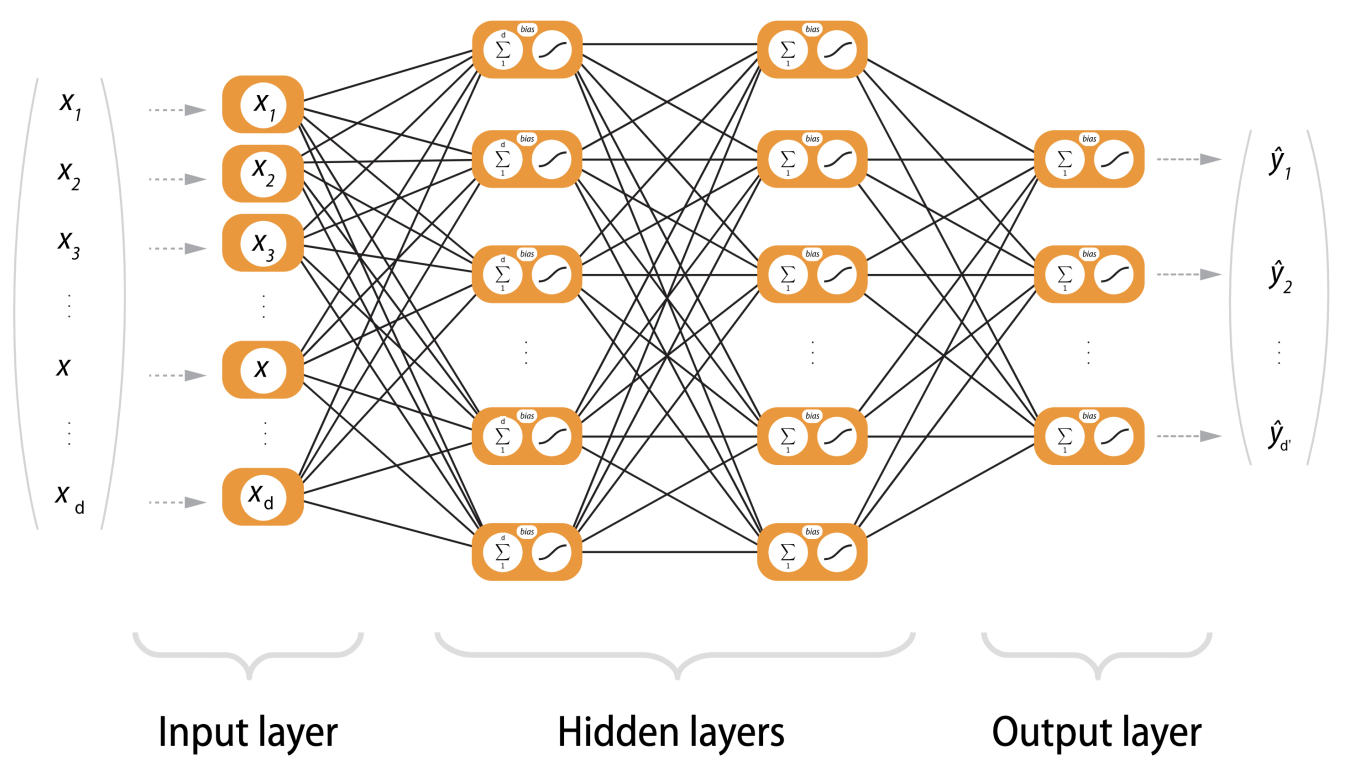
Recall: neural networks



source: fidle-cnrs

- Cascade of linear and nonlinear functions

Recall: neural networks



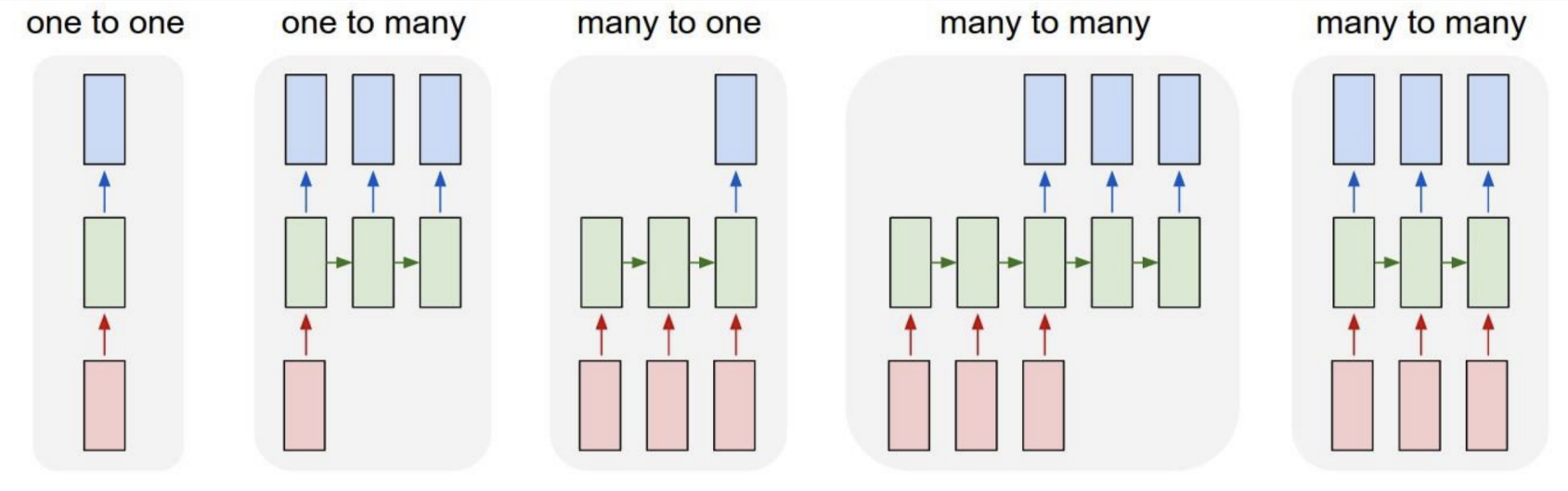
source: fidle-cnrs

- Cascade of linear and nonlinear functions
- Formally

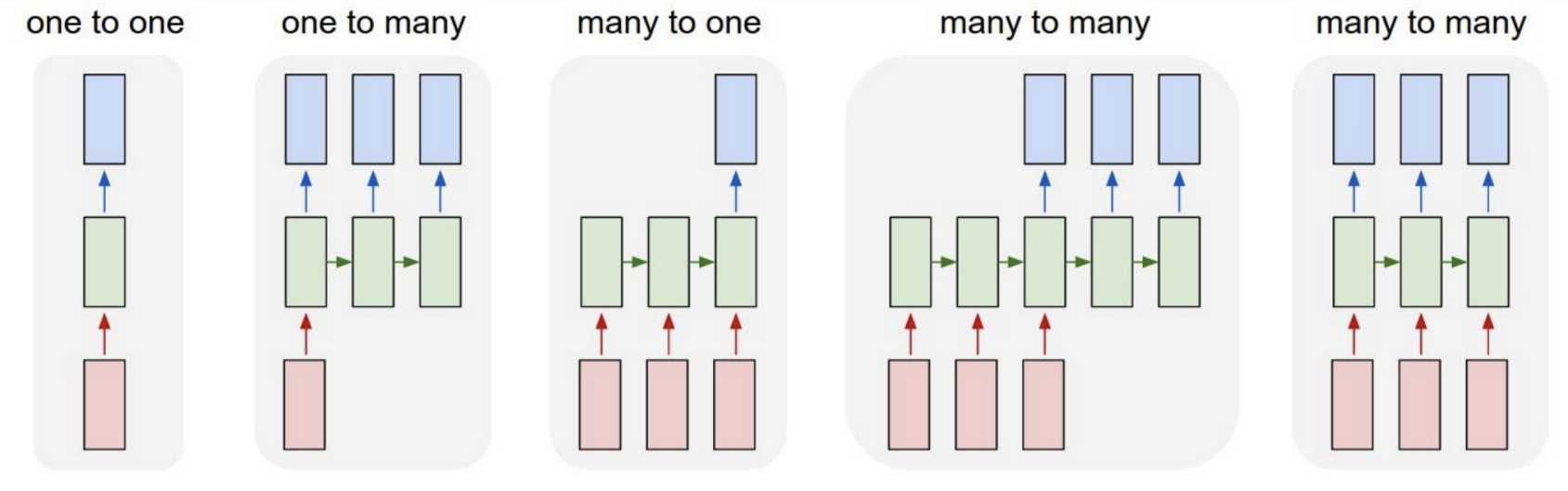
$$\hat{y} = \sigma\left(W_L \sigma\left(W_{L-1} \sigma\left(\ldots \sigma\left(W_1 x\right)\right)\right)\right)$$

Recurrent neural networks

- Recurrent Neural Networks (RNNs) are Artificial Neural Networks that can deal with sequences of variable size.

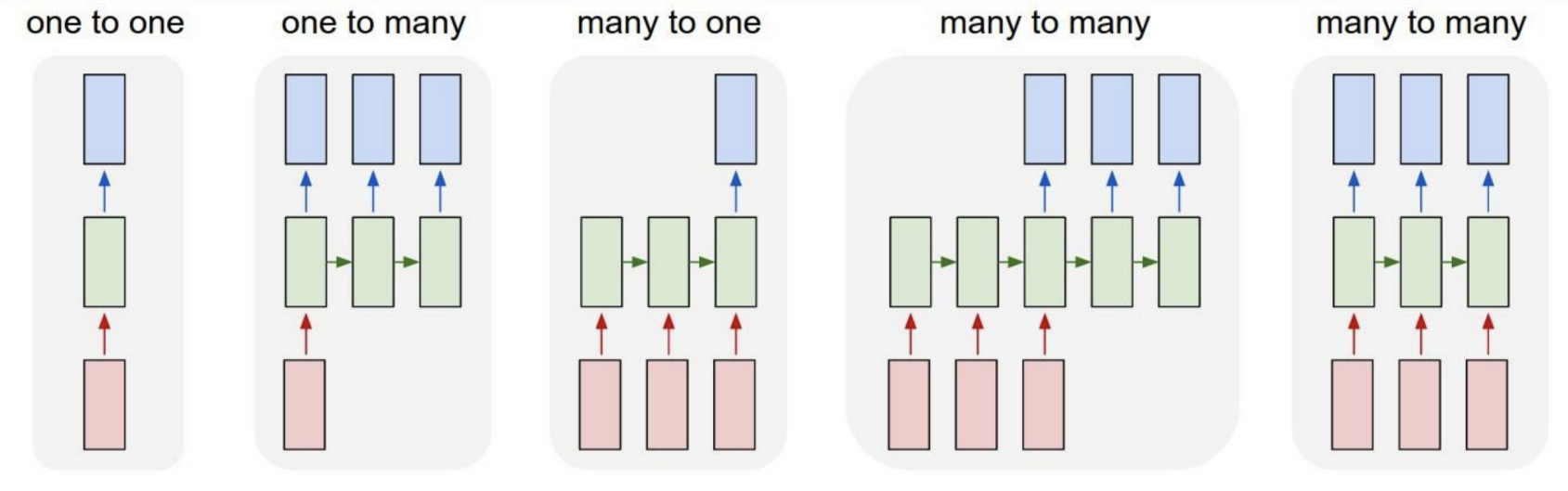


Different uses of recurrent neural networks



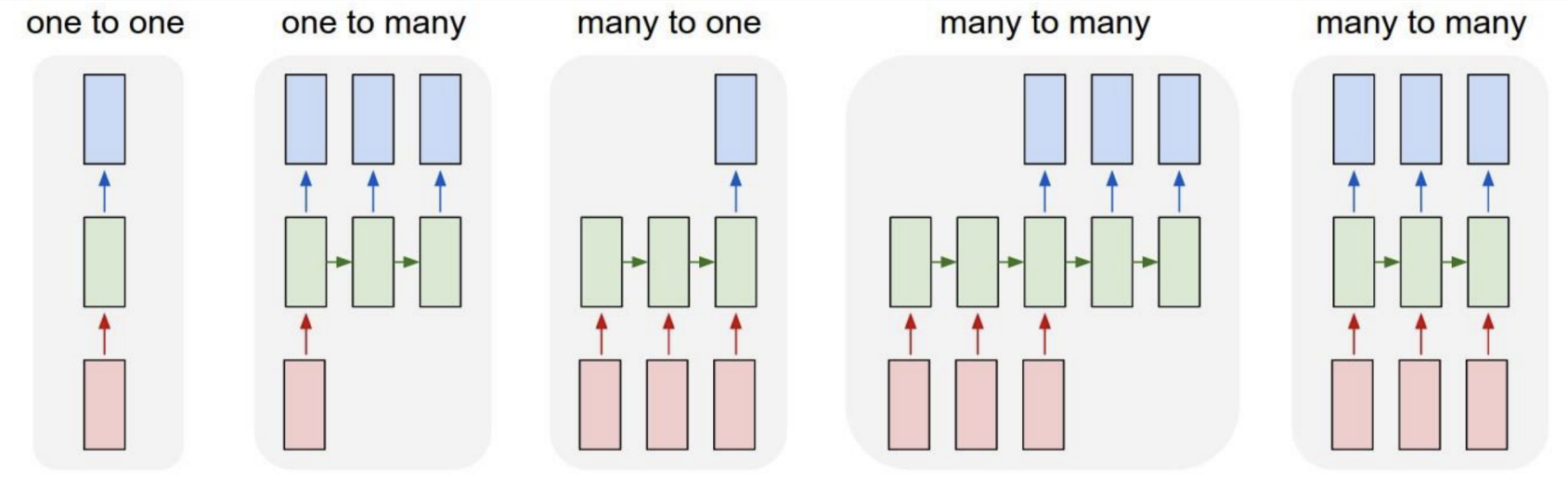
- Image classification (one-to-one)

Different uses of recurrent neural networks



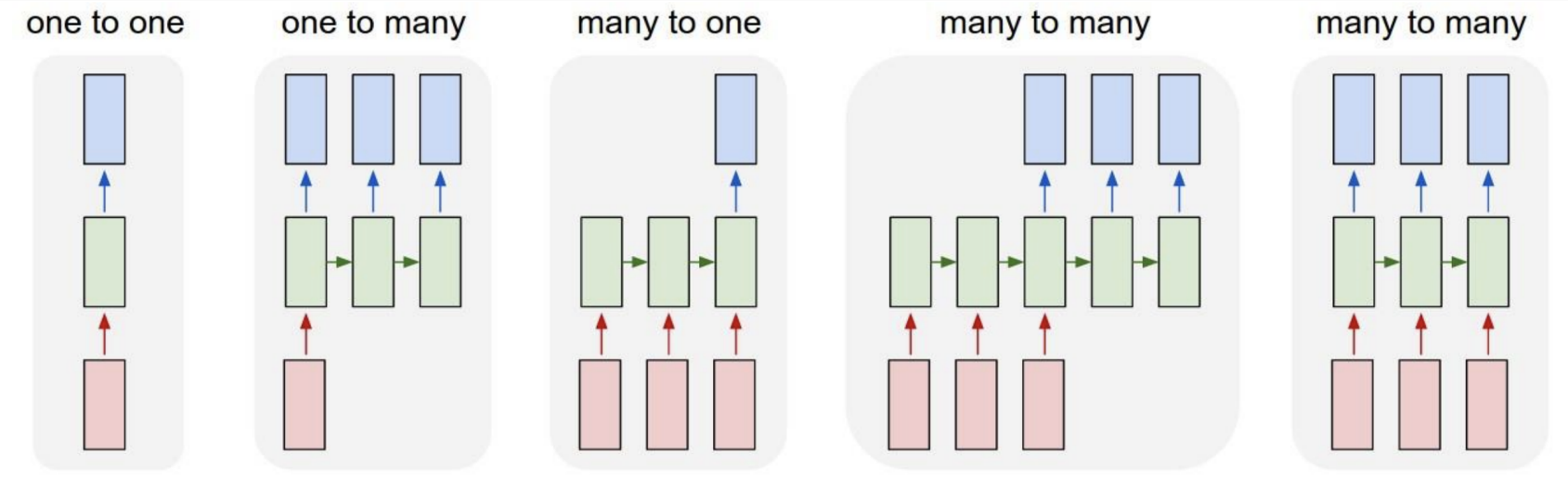
- Image classification (one-to-one)
- Image Captioning (one-to-many): image/sequence of words

Different uses of recurrent neural networks



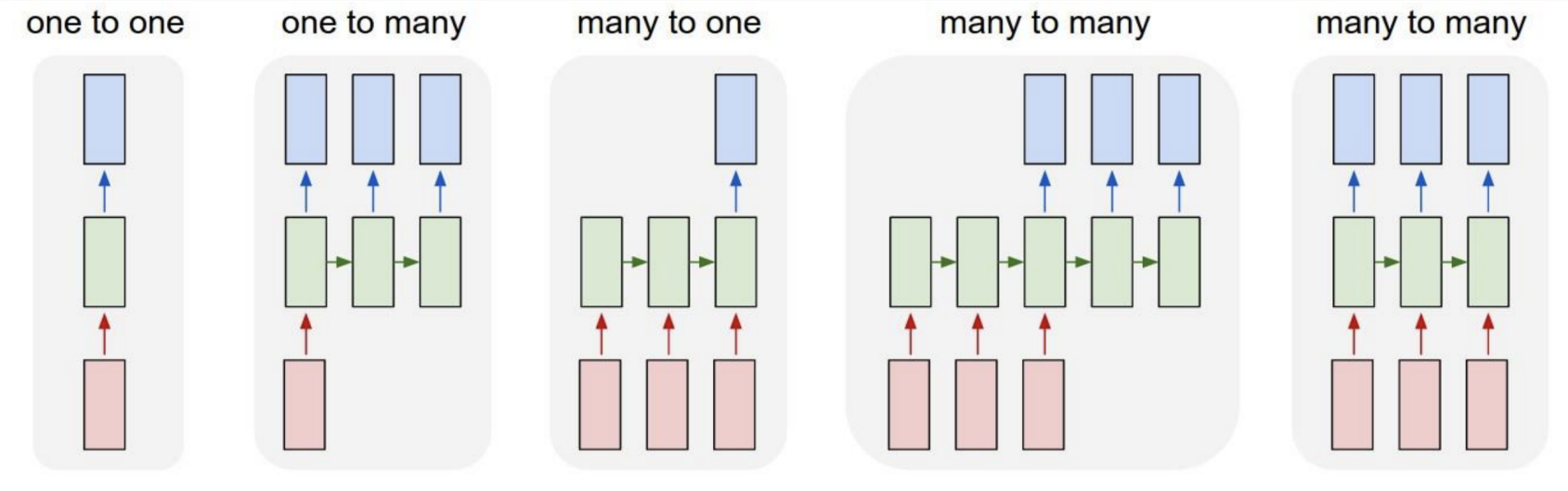
- Image classification (one-to-one)
- Image Captioning (one-to-many): image/sequence of words
- Sentiment classification (many-to-one): sequence of words/sentiment

Different uses of recurrent neural networks



- Image classification (one-to-one)
- Image Captioning (one-to-many): image/sequence of words
- Sentiment classification (many-to-one): sequence of words/sentiment
- Translation (many-to-many): sequence of words/sequence of words

Different uses of recurrent neural networks



- Image classification (one-to-one)
- Image Captioning (one-to-many): image/sequence of words
- Sentiment classification (many-to-one): sequence of words/sentiment
- Translation (many-to-many): sequence of words/sequence of words
- Video classification on frame level (many-to-many): sequence of image/sequence of label

Language generating NN: training

How to learn “`The cat is in the kitchen drinking milk.`”?

Language generating NN: training

How to learn "`The cat is in the kitchen drinking milk.`"?

- Learn: $\mathbb{P}$ (next word|current word and past)

Language generating NN: training

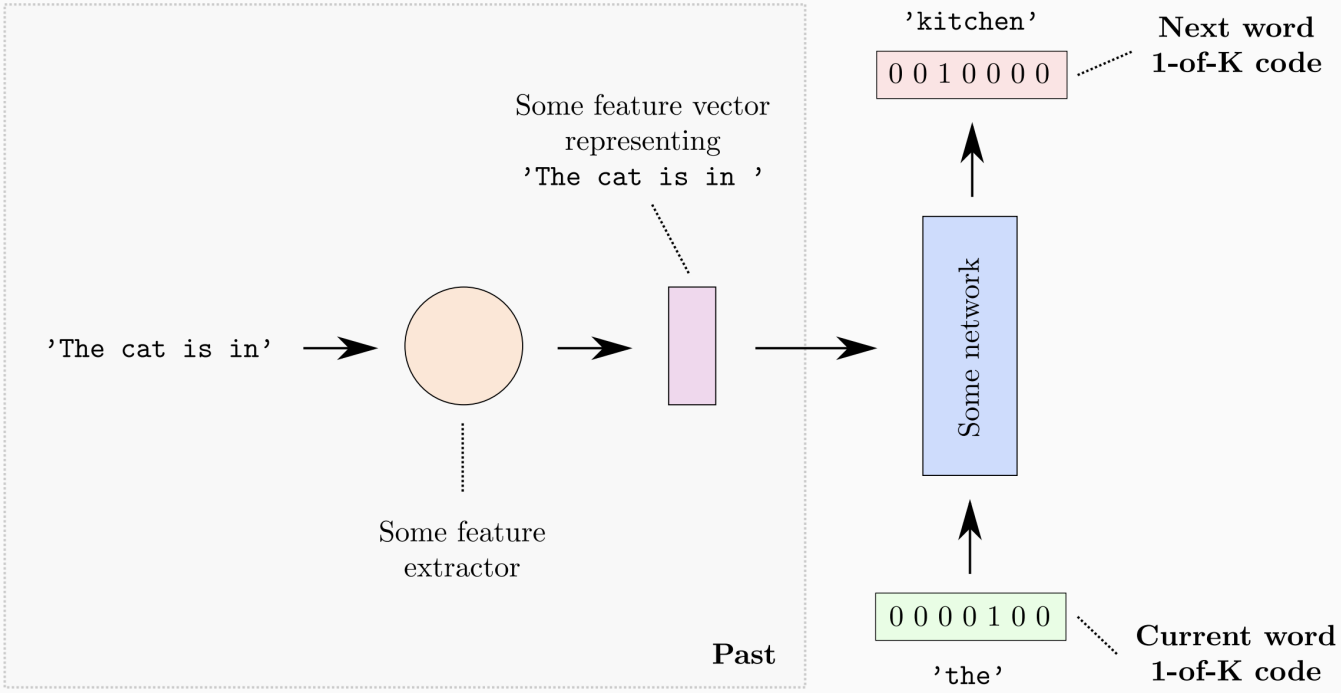
How to learn "`The cat is in the kitchen drinking milk.`"?

- Learn: $\mathbb{P}$ (next word|current word and past)
- Represent the past as a feature vector

Language generating NN: training

How to learn "`The cat is in the kitchen drinking milk.`"?

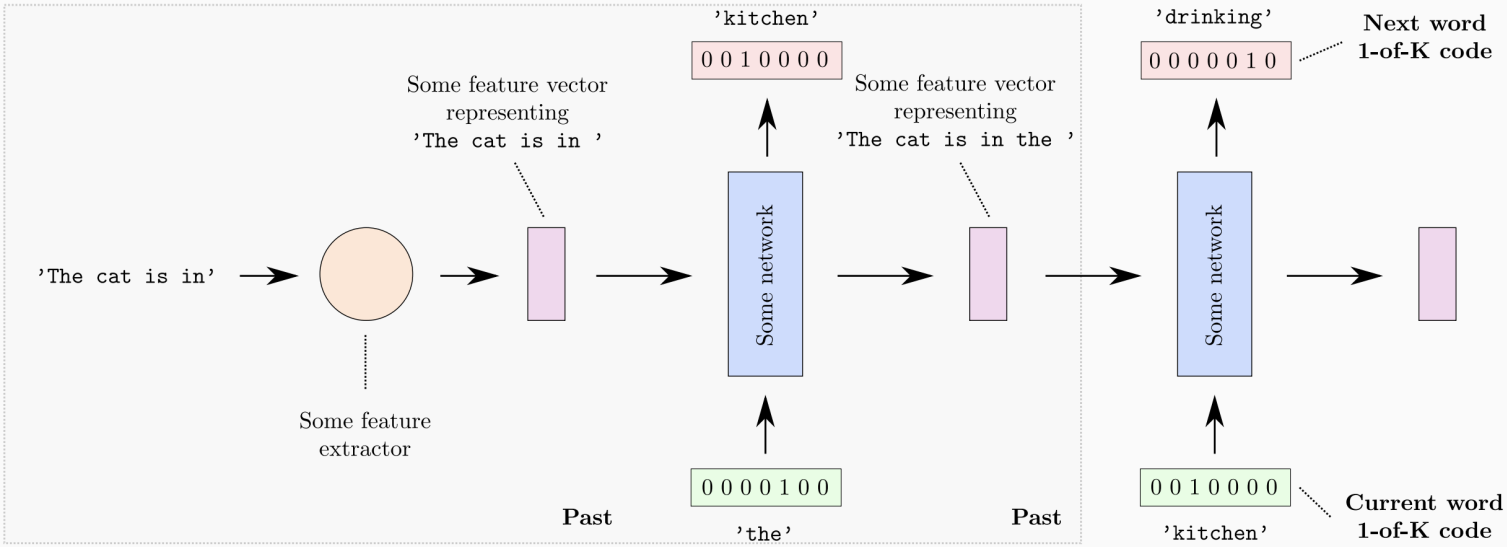
- Learn: $\mathbb{P}$ (next word|current word and past)
- Represent the past as a feature vector



Language generating NN: training

How to learn "`The cat is in the kitchen drinking milk.`"?

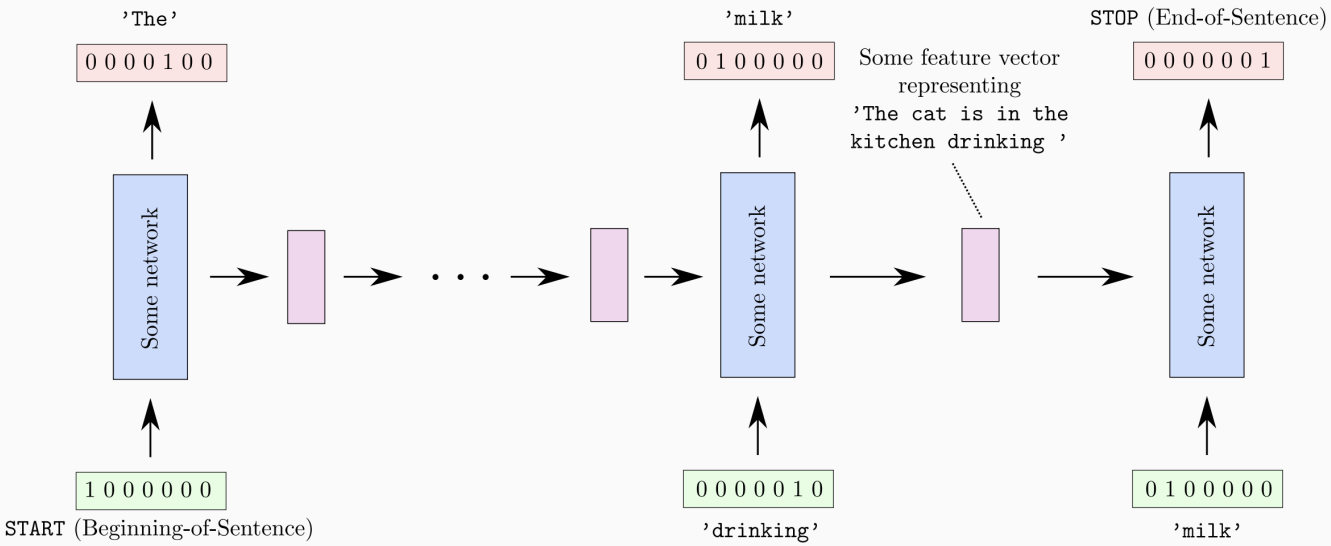
- Learn: $\mathbb{P}$ (next word|current word and past)
- Represent the past as a feature vector



- Learn also how to represent the current sentence
- Repeat for the next word

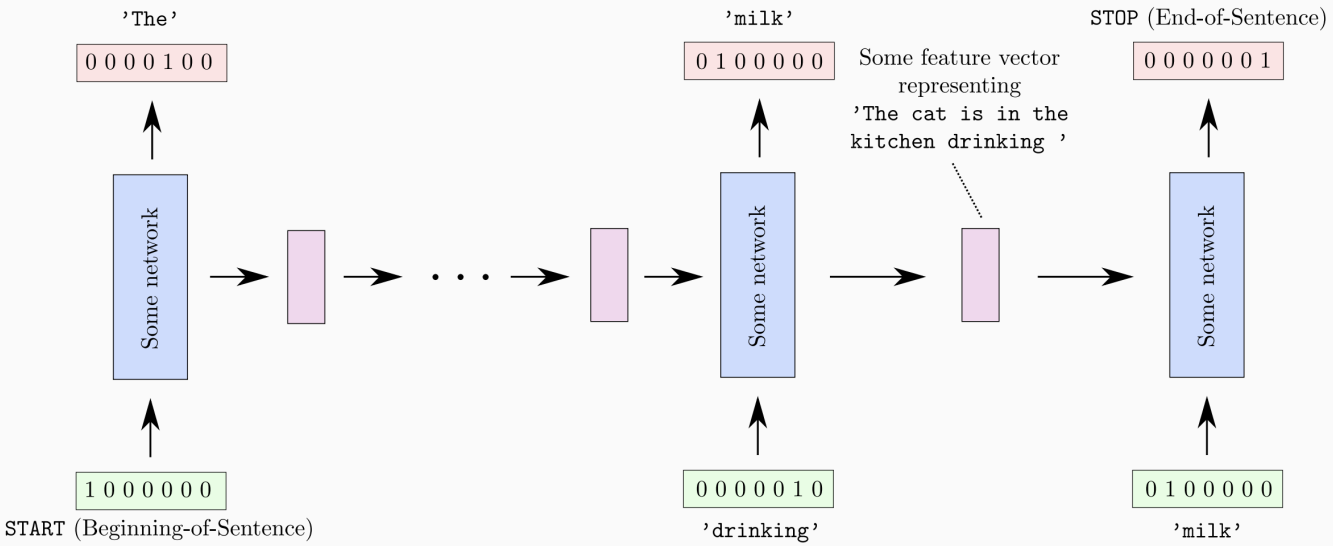
Language generating NN: training

- Add two words: START and STOP to delimitate the sentence



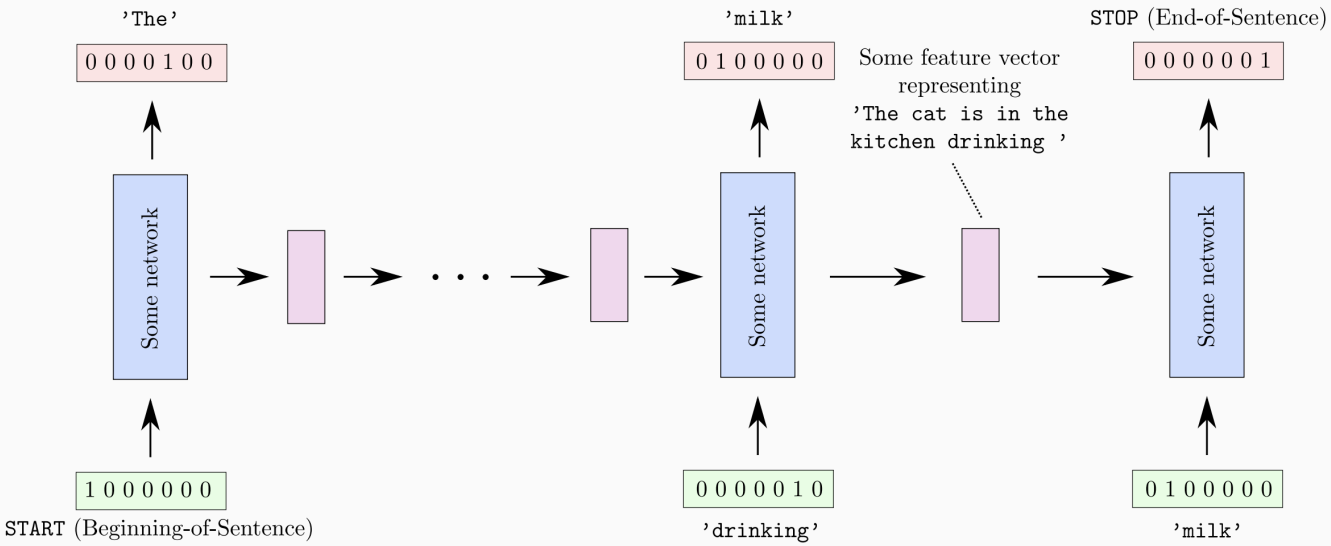
Language generating NN: training

- Add two words: START and STOP to delimitate the sentence
- Learn everything end-to-end on a large corpus of sentences



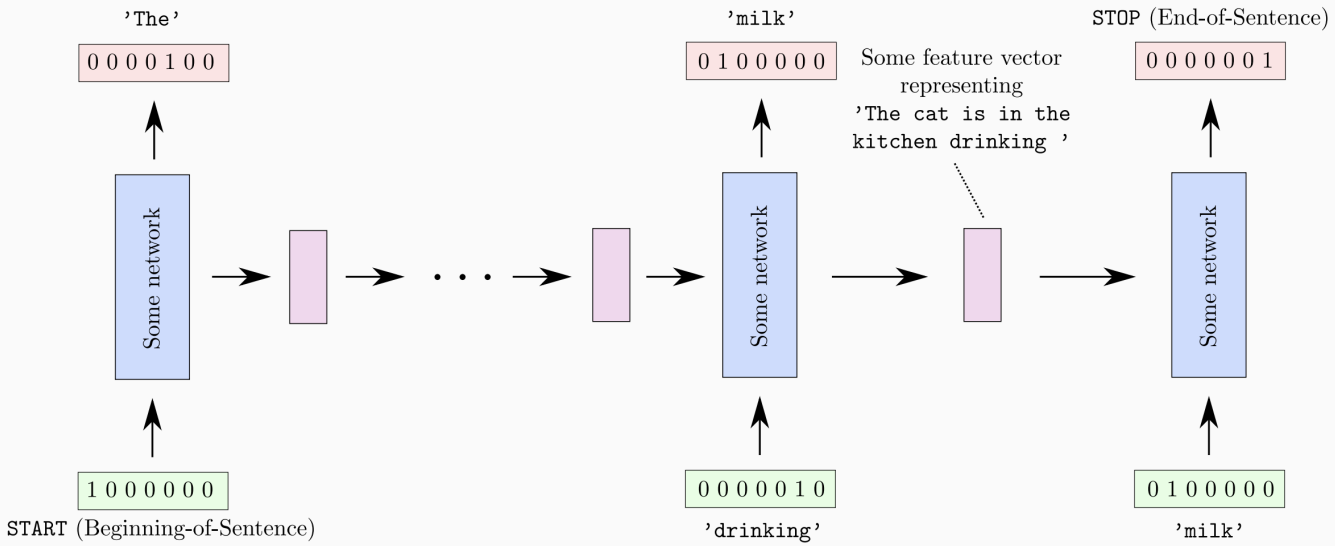
Language generating NN: training

- Add two words: START and STOP to delimitate the sentence
- Learn everything end-to-end on a large corpus of sentences
- Minimize the sum of the cross-entropy of each word



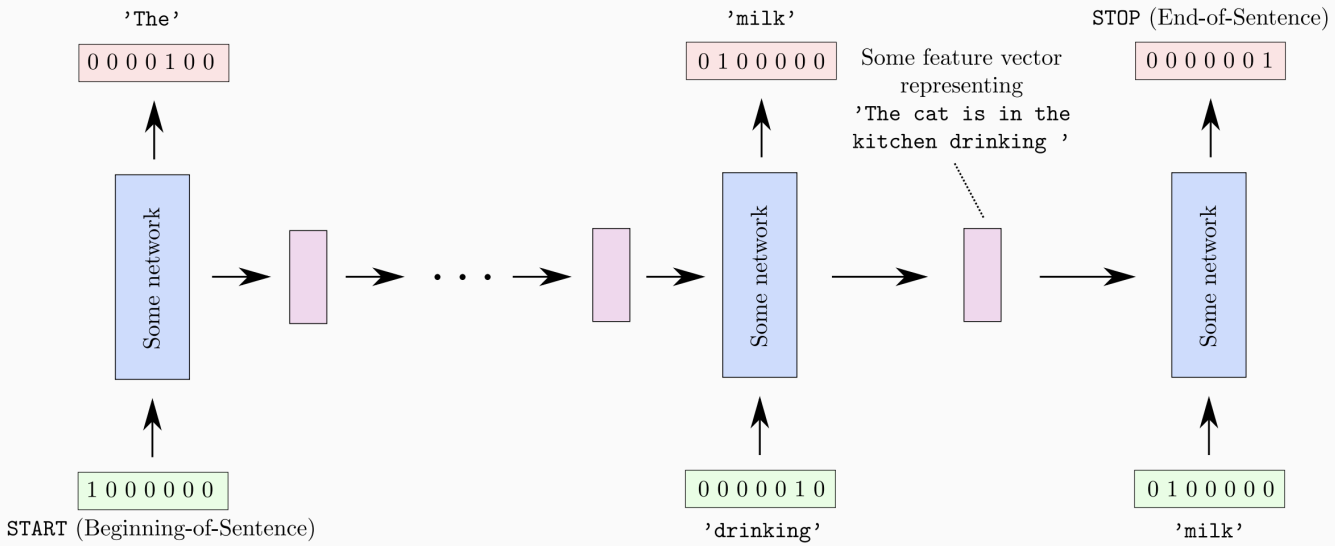
Language generating NN: training

- Add two words: START and STOP to delimitate the sentence
- Learn everything end-to-end on a large corpus of sentences
- Minimize the sum of the cross-entropy of each word
- Intermediate features will learn how to memorize the past/context/state



Language generating NN: training

- Add two words: START and STOP to delimitate the sentence
- Learn everything end-to-end on a large corpus of sentences
- Minimize the sum of the cross-entropy of each word
- Intermediate features will learn how to memorize the past/context/state

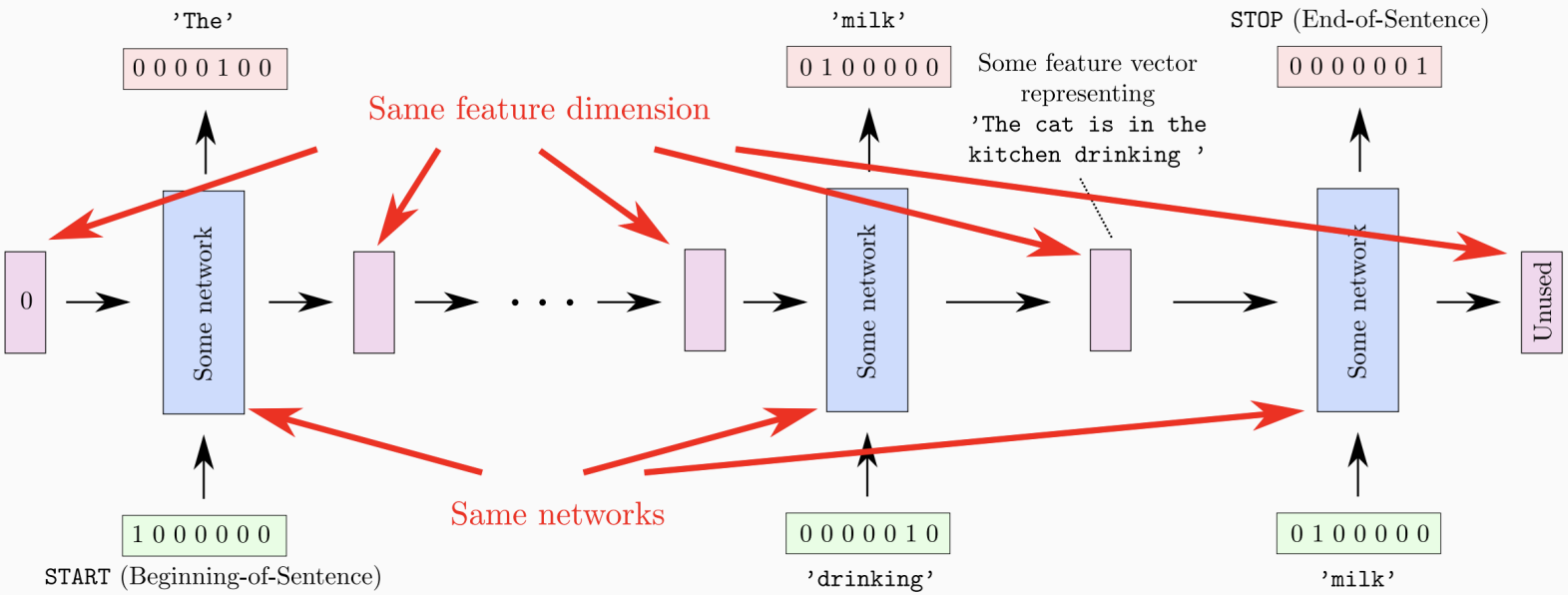


▷ *How should the network architecture and size of intermediate features evolve with the location in the sequence?*

Language generating NN: training

from Charles Deledalle's lectures

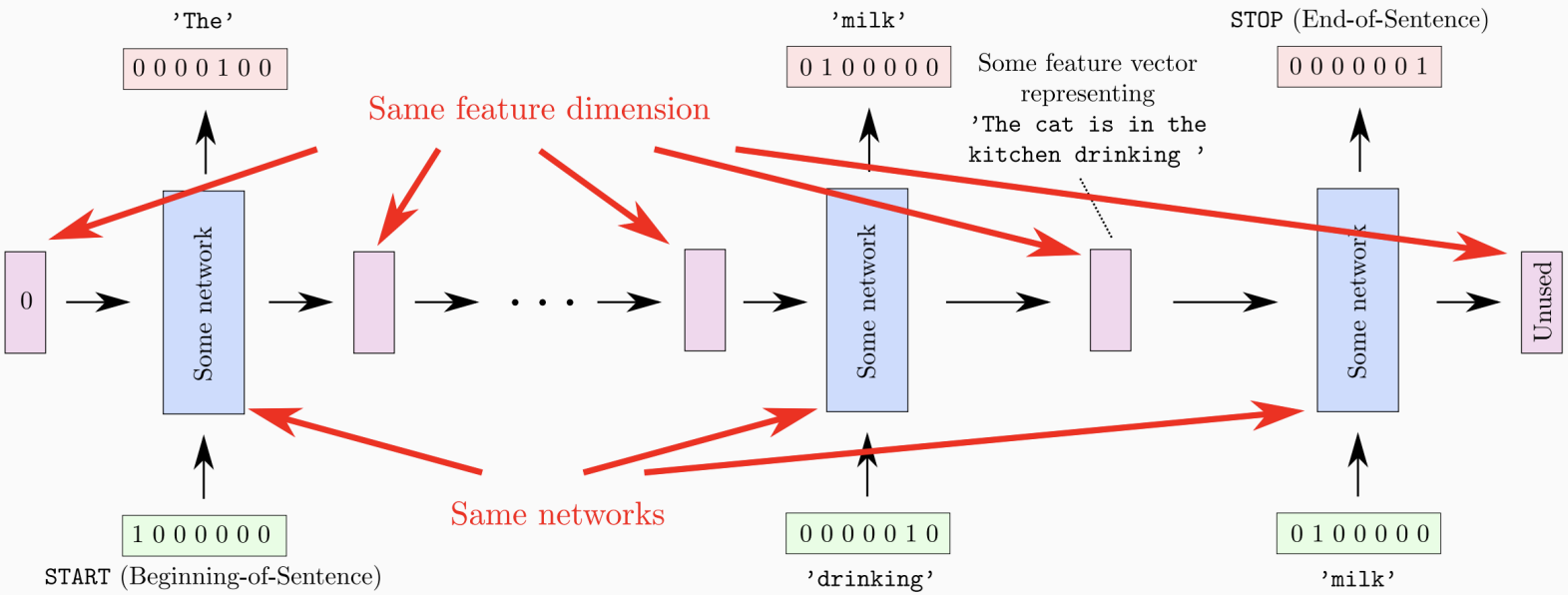
- Use the same networks and the same feature dimension



Language generating NN: training

from Charles Deledalle's lectures

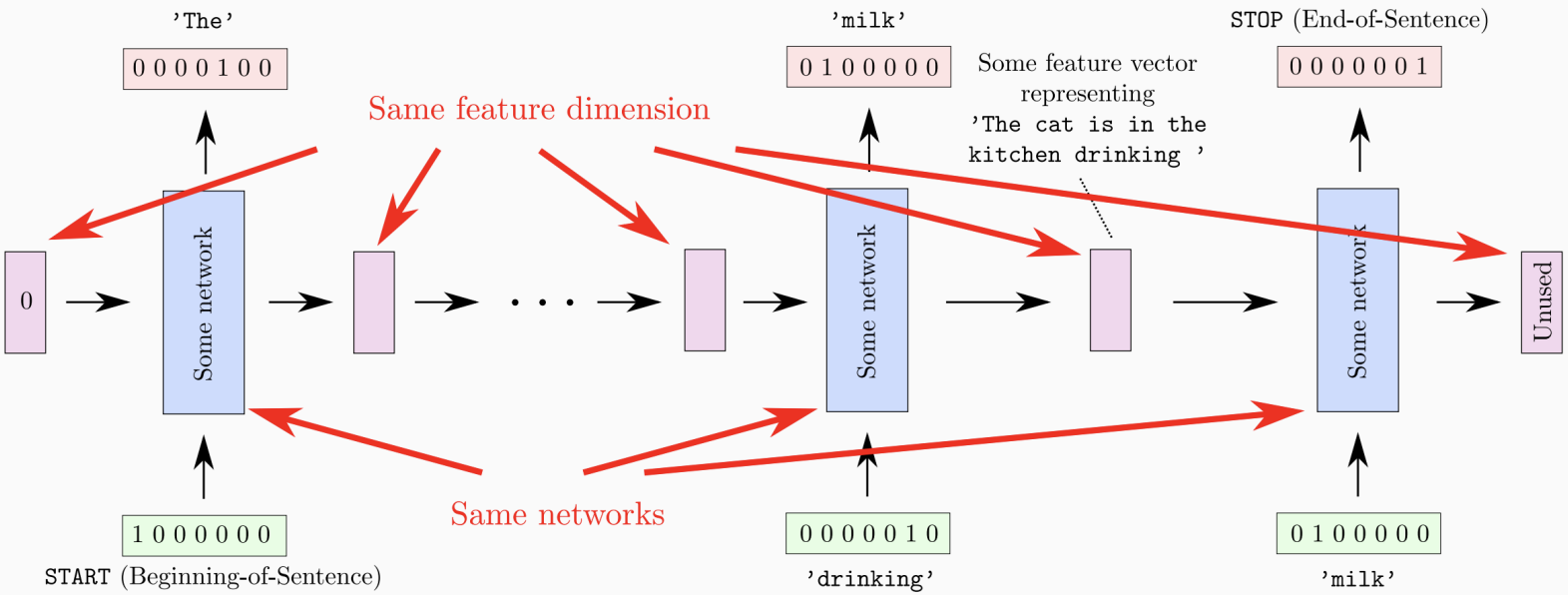
- Use the same networks and the same feature dimension
- The past is always embedded in a fix-sized feature



Language generating NN: training

from Charles Deledalle's lectures

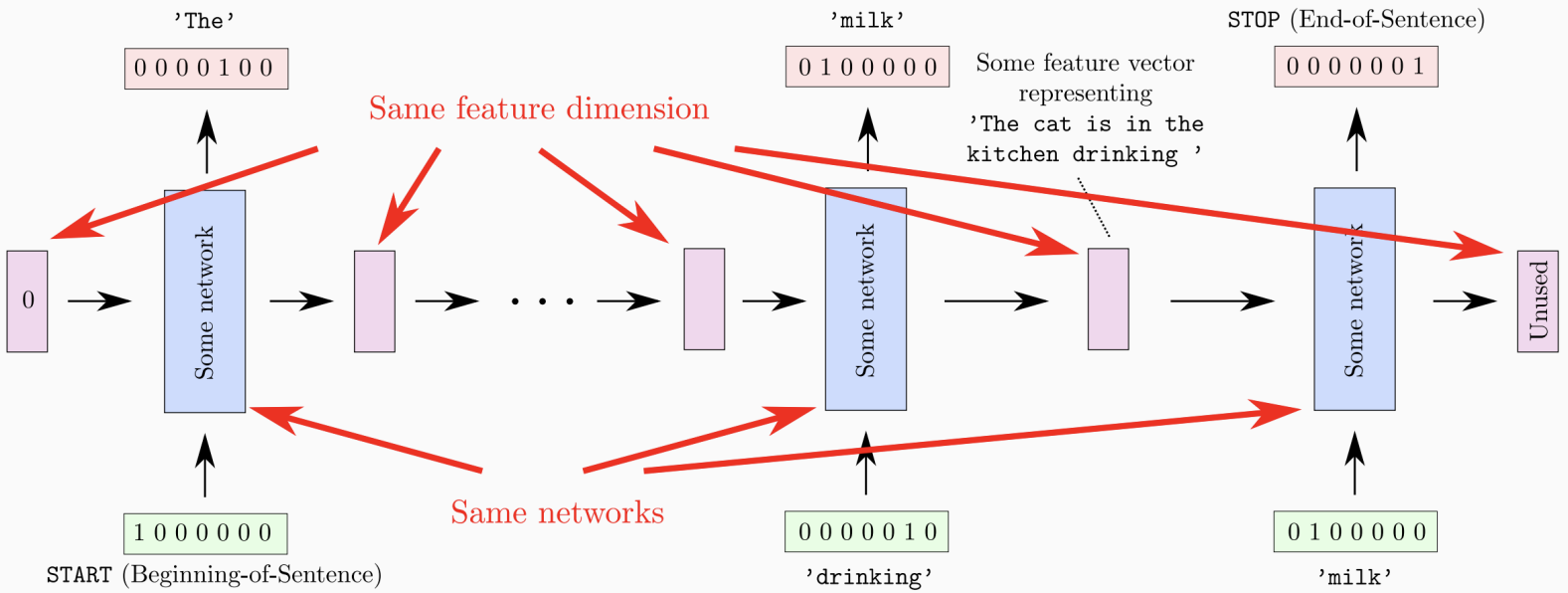
- Use the same networks and the same feature dimension
- The past is always embedded in a fix-sized feature
- Set the first feature as a zero tensor



Language generating NN: training

from Charles Deledalle's lectures

- Use the same networks and the same feature dimension
- The past is always embedded in a fix-sized feature
- Set the first feature as a zero tensor

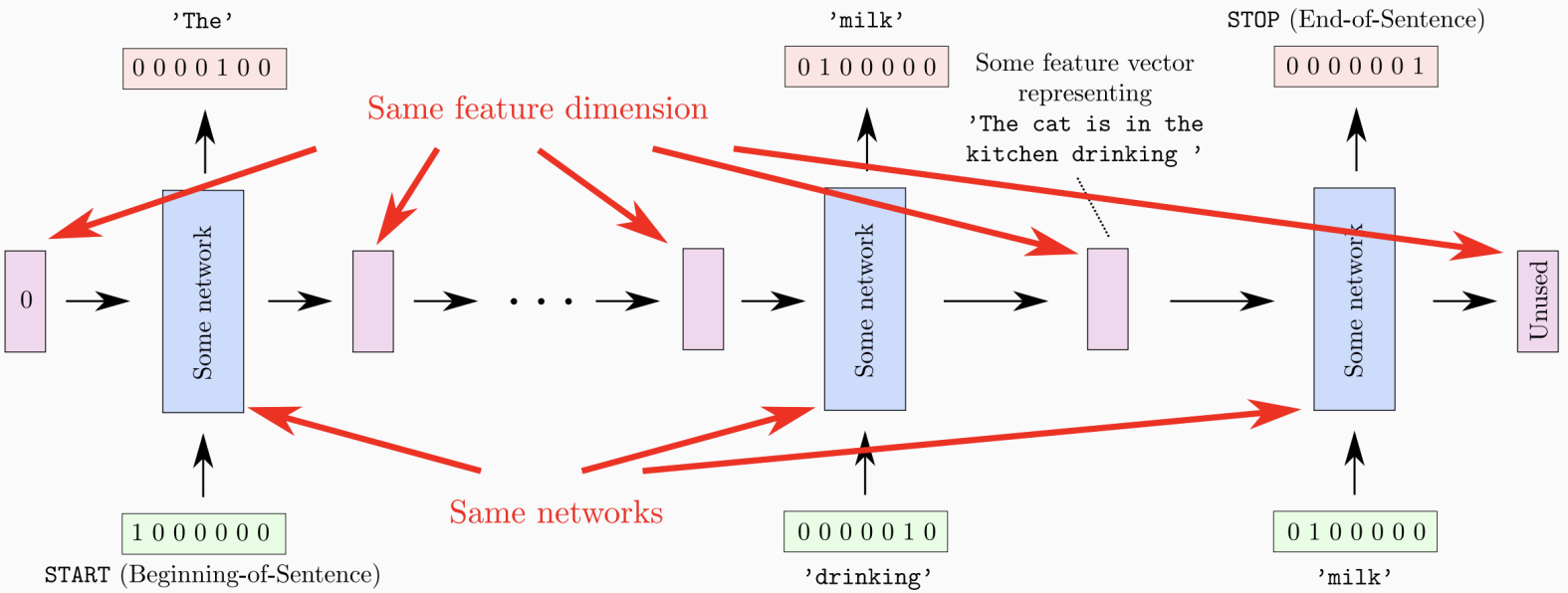


- ▷ Allows you to learn from arbitrarily long sequences

Language generating NN: training

from Charles Deledalle's lectures

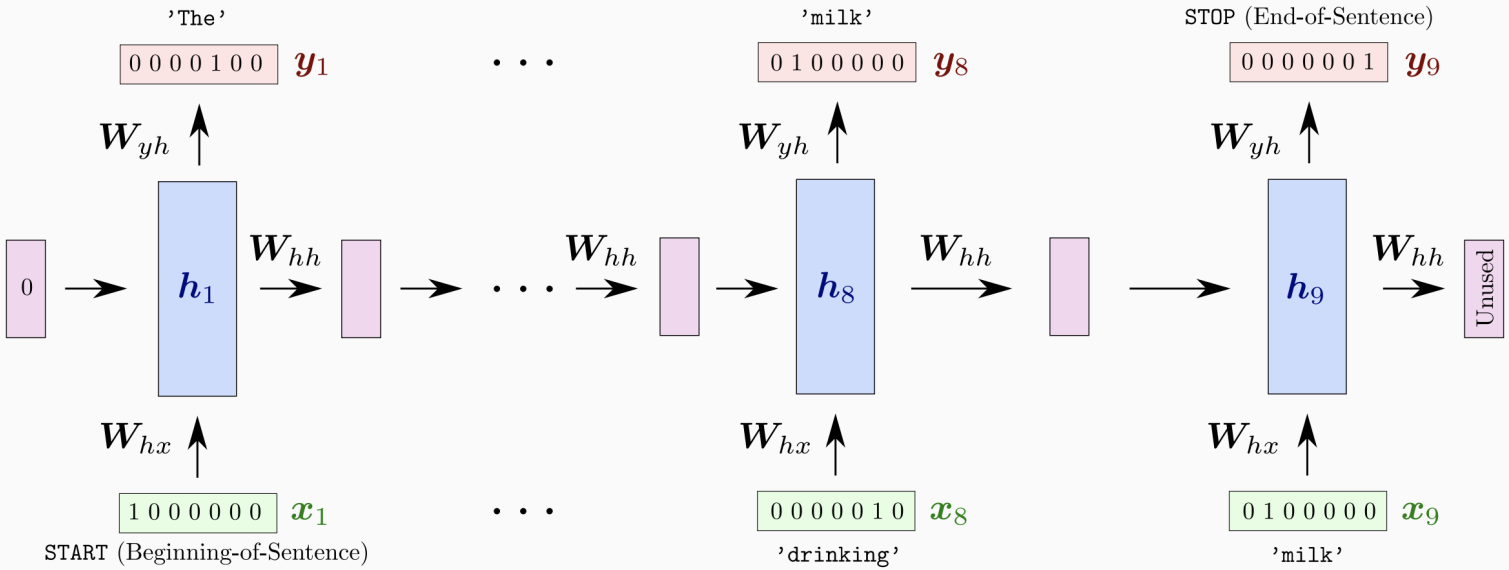
- Use the same networks and the same feature dimension
- The past is always embedded in a fix-sized feature
- Set the first feature as a zero tensor



- Allows you to learn from arbitrarily long sequences
- Sharing the architecture $\Rightarrow$ fewer parameters $\Rightarrow$ training requires less data and the final prediction can be expected to be more accurate

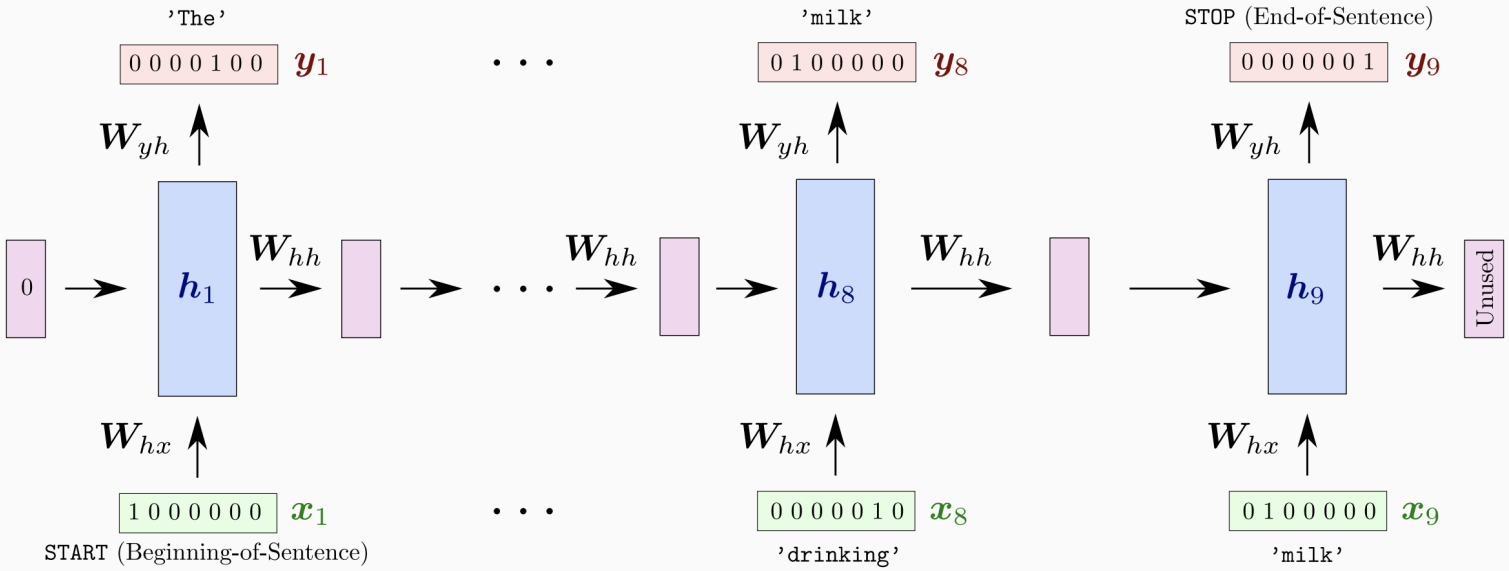
A simple shallow RNN for sentence generation

- This is an unfolded representation of an RNN



A simple shallow RNN for sentence generation

- This is an unfolded representation of an RNN



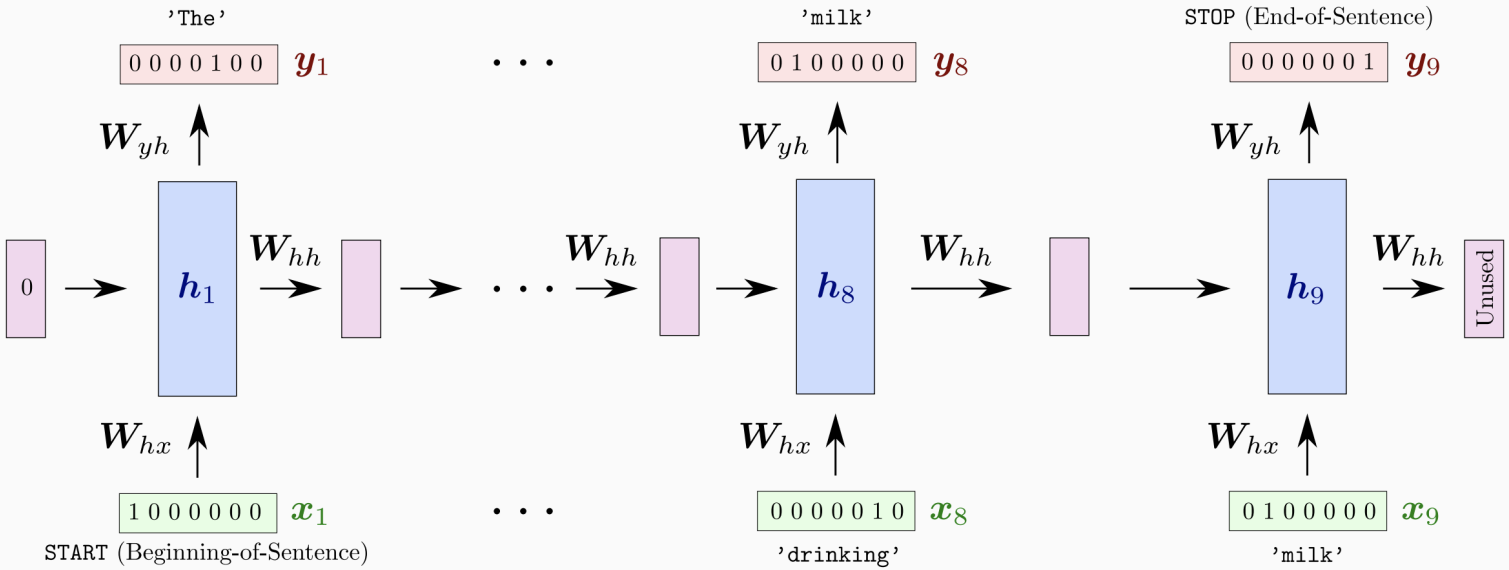
Vanilla RNN

$$h_t = g\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = \text{softmax}\left(W_{yh}h_t + b_y\right)$$

A simple shallow RNN for sentence generation

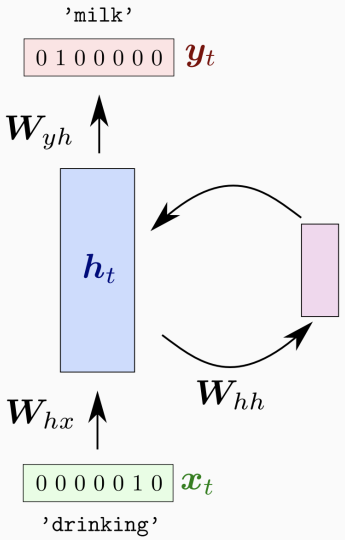
- This is an unfolded representation of an RNN



Vanilla RNN

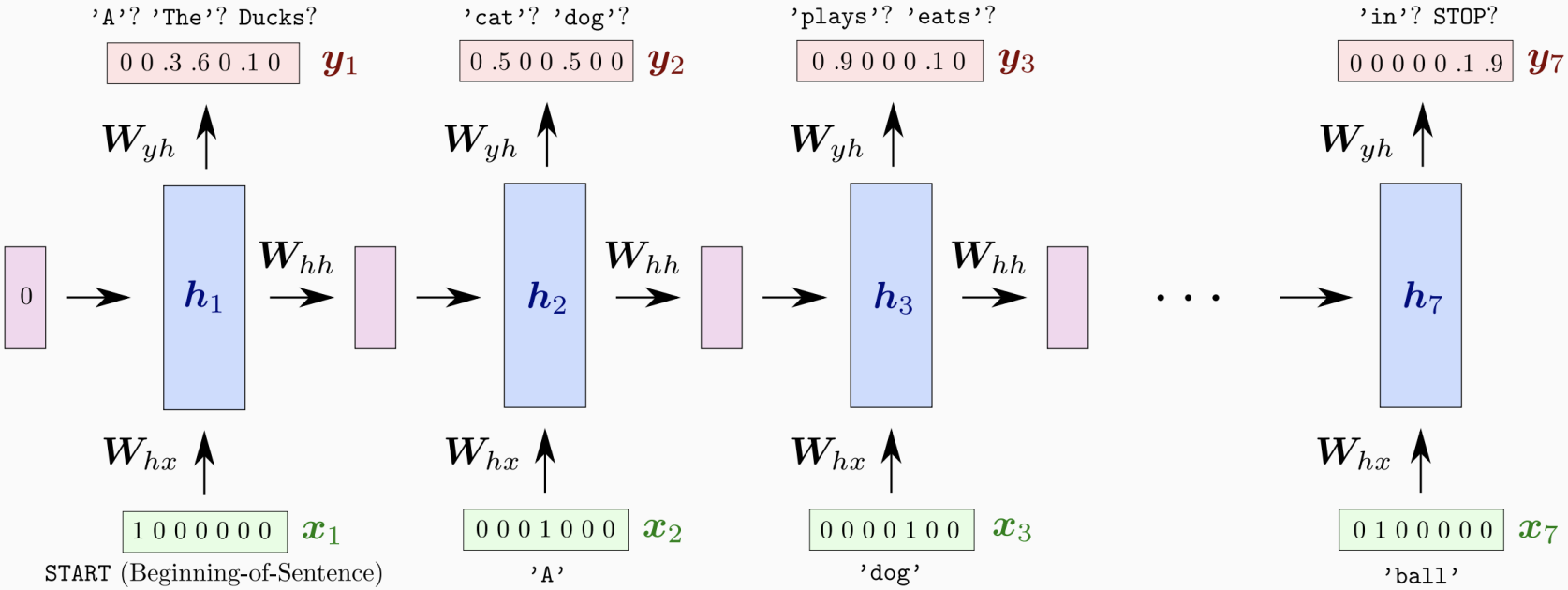
$$h_t = g\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = \text{softmax}\left(W_{yh}h_t + b_y\right)$$



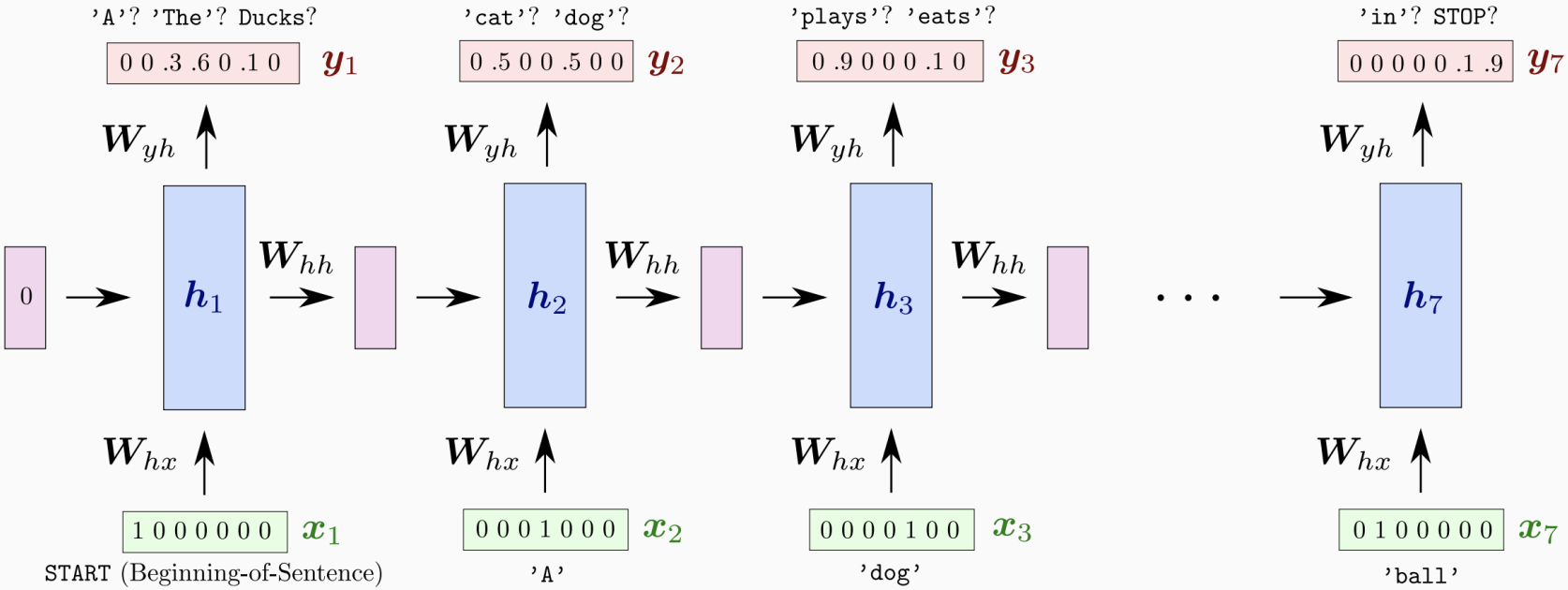
- Folded representation: RNN ≈ ANN with loops

Generate a sentence in practice



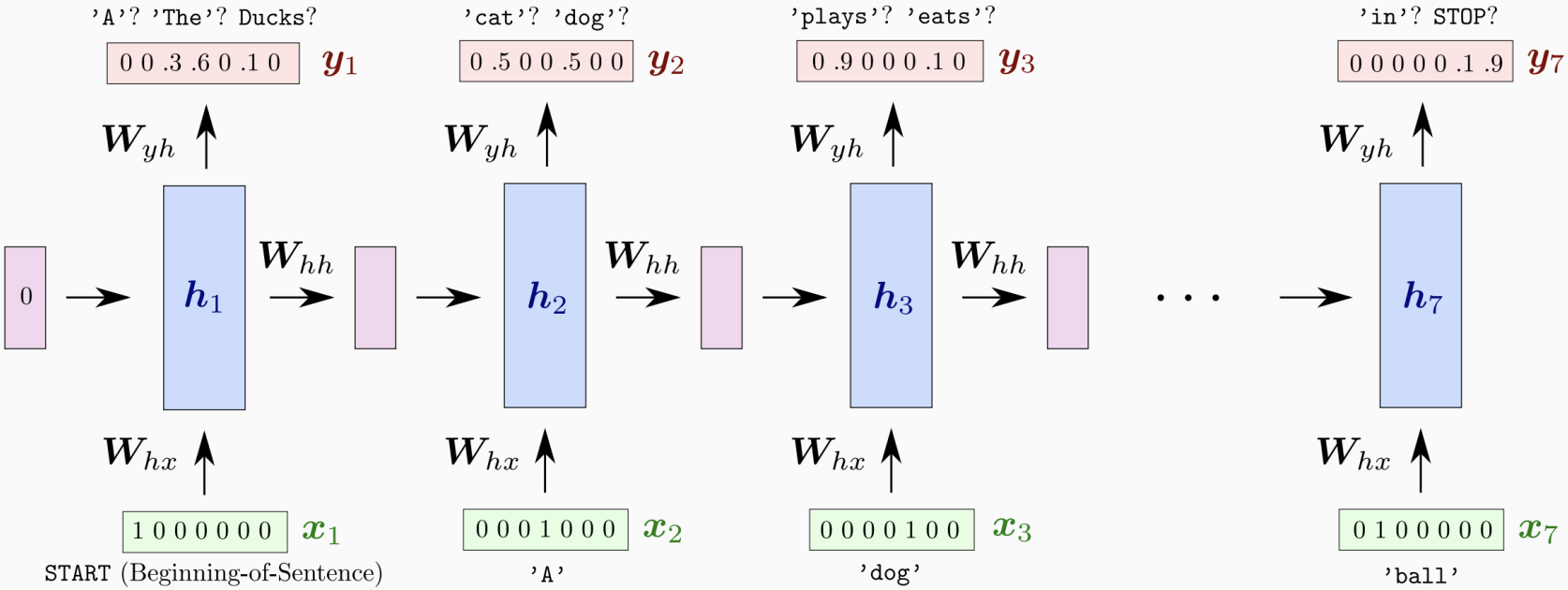
- Provide START, get all the probabilities
  $\mathbb{P}(\text{next word}|\text{current word} = \text{START})$

Generate a sentence in practice



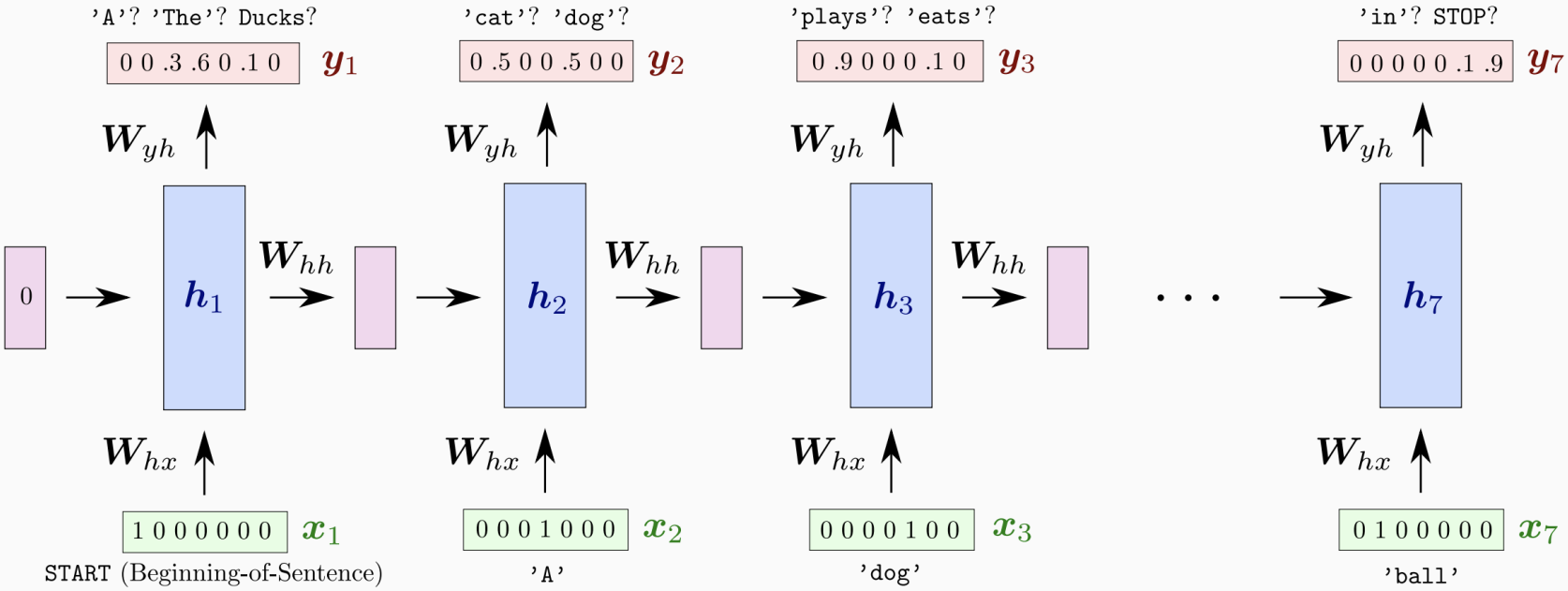
- Provide START, get all the probabilities
  $\mathbb{P}(\text{next word}|\text{current word} = \text{START})$
- Select one of these words according to their probabilities, let say 'A',

Generate a sentence in practice



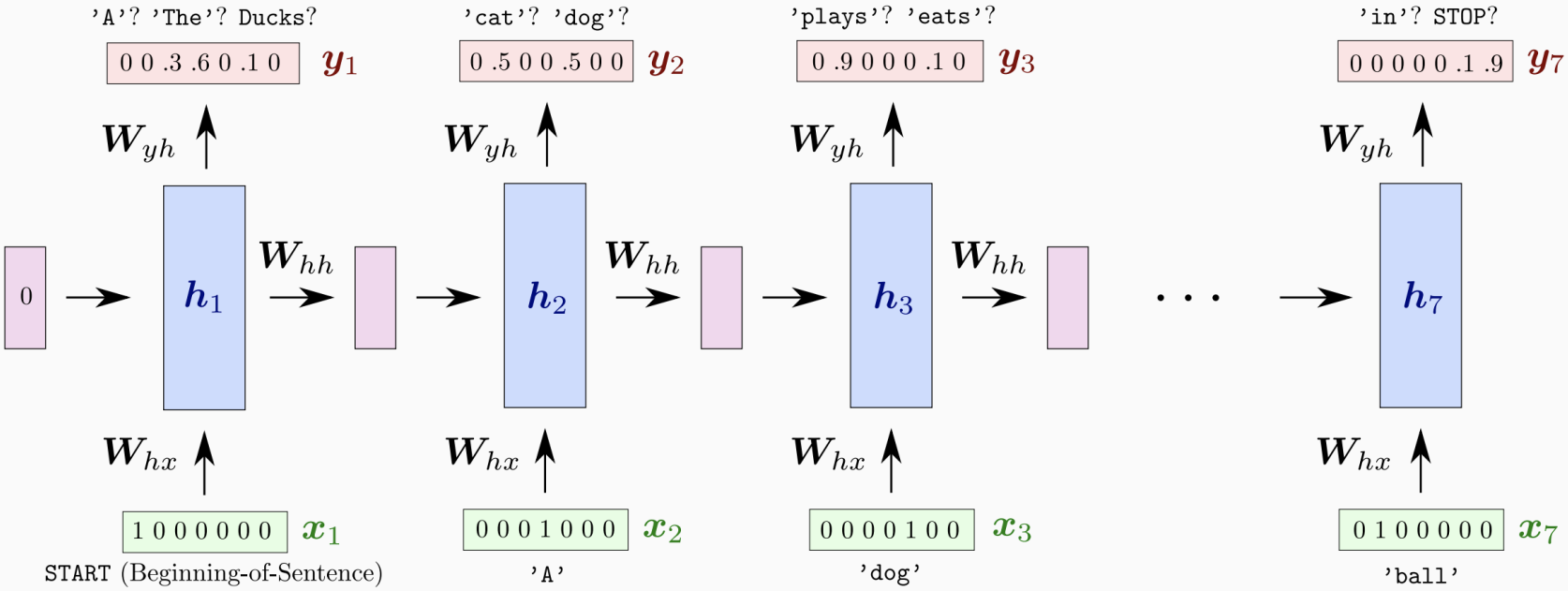
- Provide START, get all the probabilities
  $\mathbb{P}$(next word|current word = START)
- Select one of these words according to their probabilities, let say 'A',
- Provide 'A' and the past, and get $\mathbb{P}$(next word|current word = A)

Generate a sentence in practice



- Provide START, get all the probabilities
  $\mathbb{P}(\text{next word}|\text{current word} = \text{START})$
- Select one of these words according to their probabilities, let say 'A',
- Provide 'A' and the past, and get $\mathbb{P}(\text{next word}|\text{current word} = \text{A})$
- Repeat while generating the sentence 'A dog plays with a ball'

Generate a sentence in practice



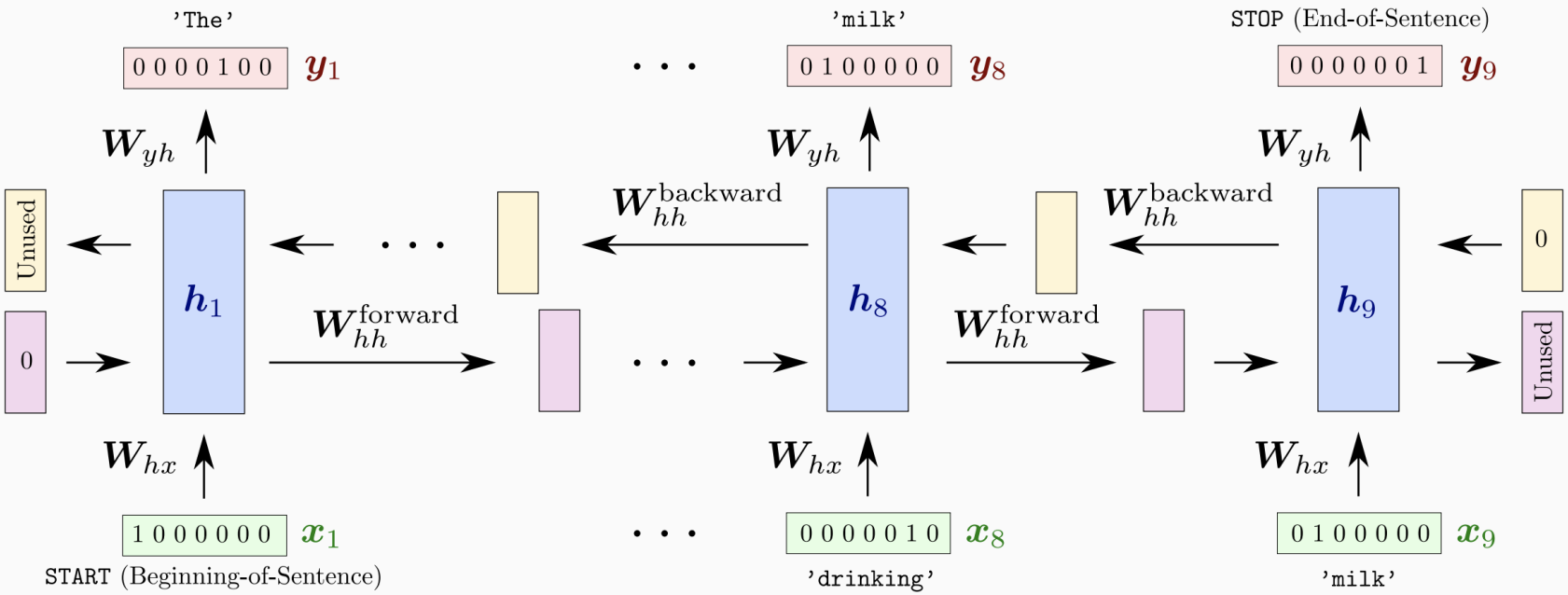
- Provide START, get all the probabilities
  $\mathbb{P}(\text{next word}|\text{current word} = \text{START})$
- Select one of these words according to their probabilities, let say 'A',
- Provide 'A' and the past, and get $\mathbb{P}(\text{next word}|\text{current word} = \text{A})$
- Repeat while generating the sentence 'A dog plays with a ball'
- Stop as soon as you have picked STOP.

Bidirectional RNN

- Output at time t may not only depend on the previous elements, but also on future elements

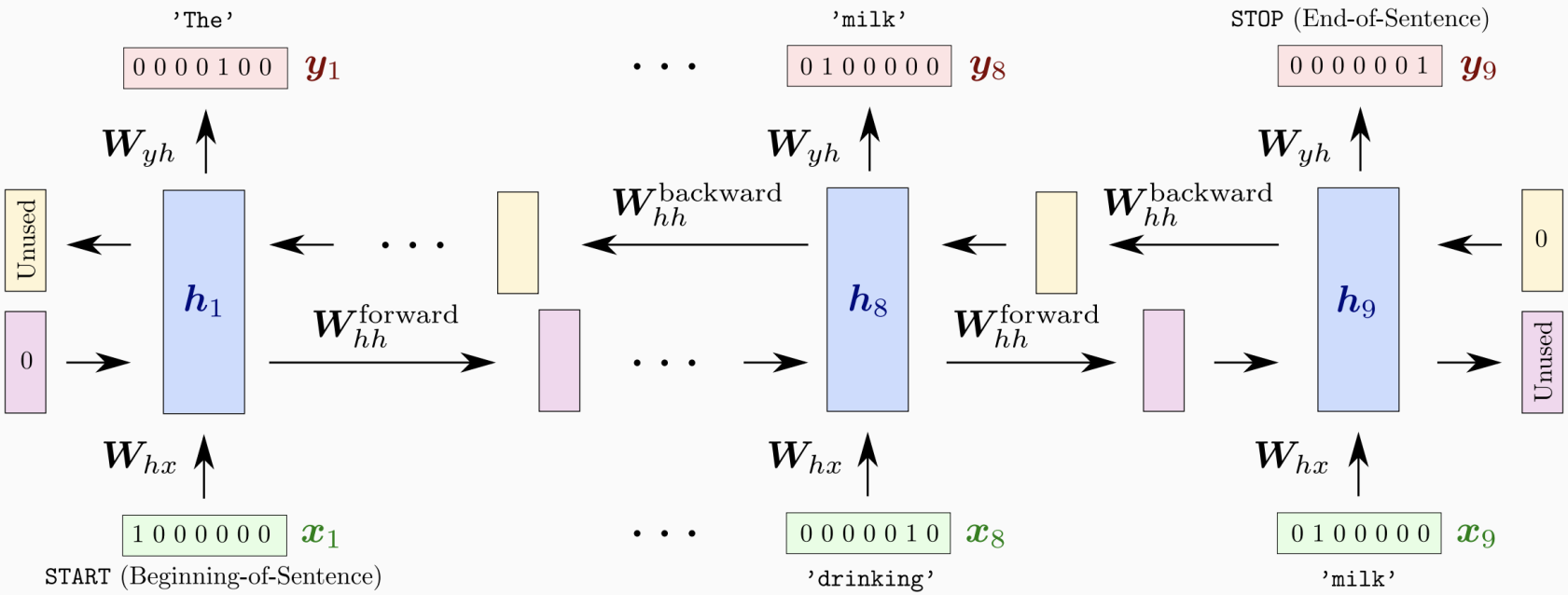
Bidirectional RNN

- Output at time t may not only depend on the previous elements, but also on future elements



Bidirectional RNN

- Output at time t may not only depend on the previous elements, but also on future elements



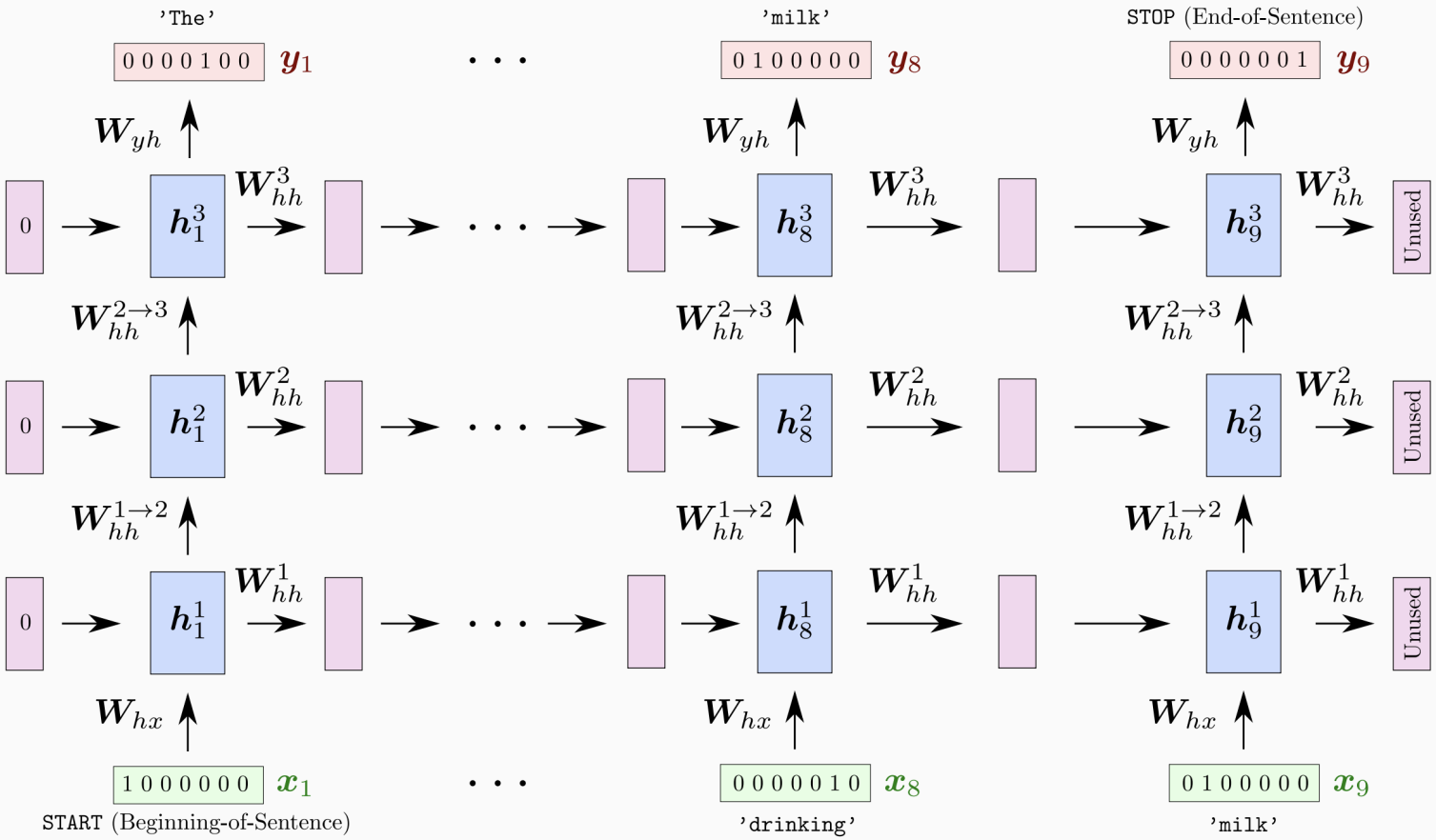
Bidirectional RNN

$$h_t = g\left(W_{hx}x_t + W_{hh}^{\text{forward}}h_{t-1} + W_{hh}^{\text{backward}}h_{t+1} + b_h\right)$$

$$y_t = \text{softmax}\left(W_{yh}h_t + b_y\right)$$

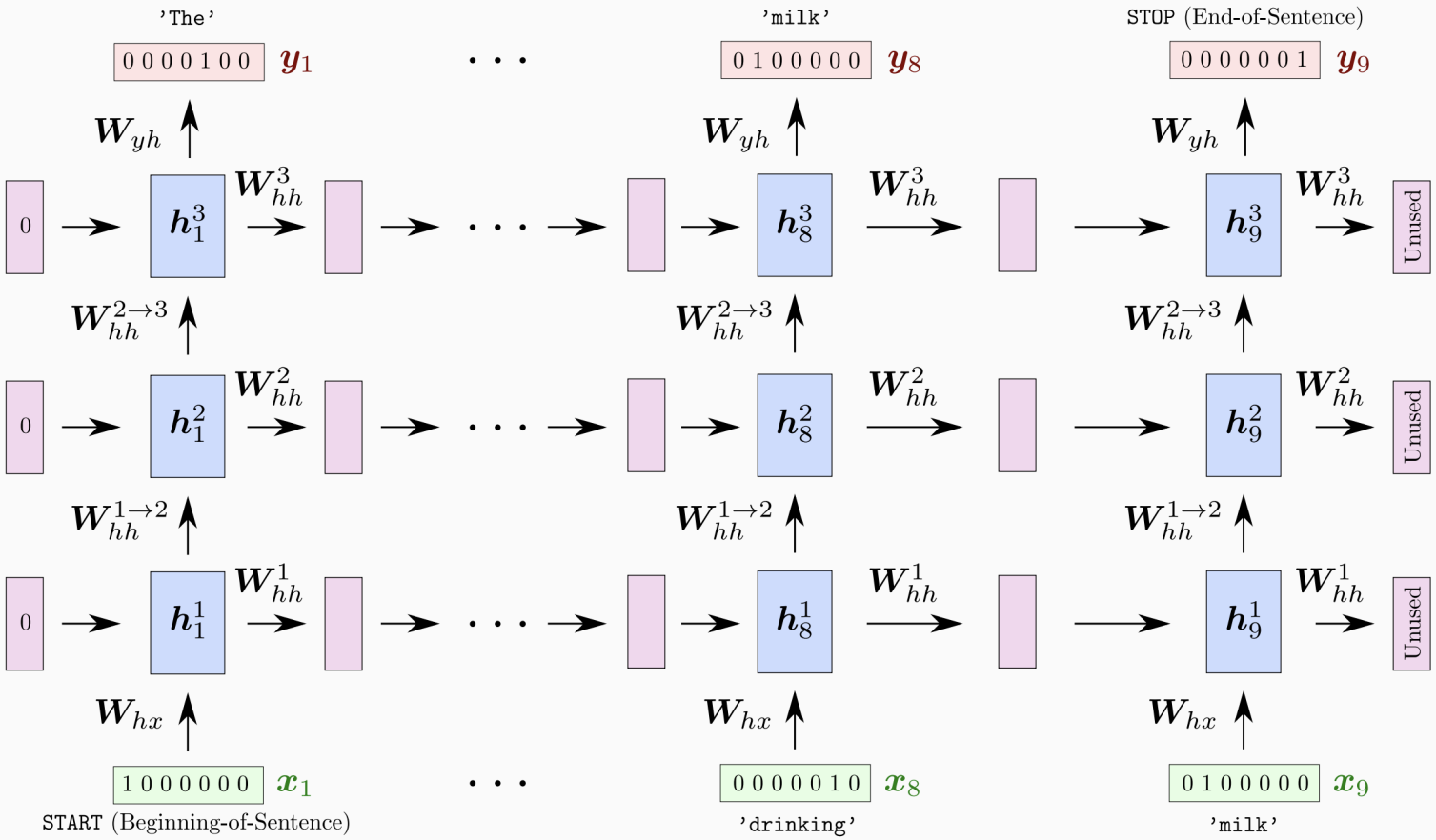
Deep RNN

- Multiple layers per time step (a feature hierarchy)



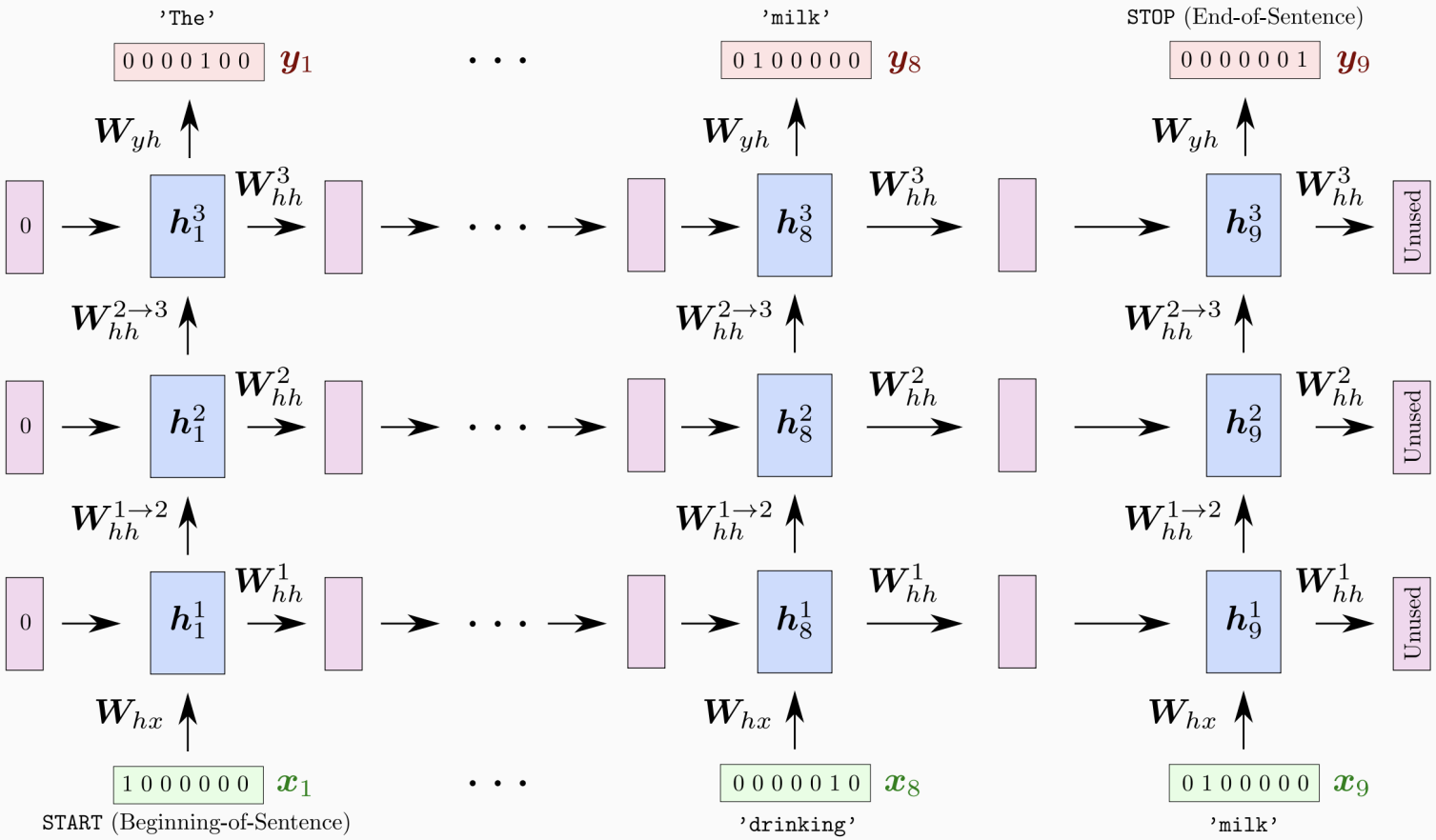
Deep RNN

- Multiple layers per time step (a feature hierarchy)
- Higher learning capacity

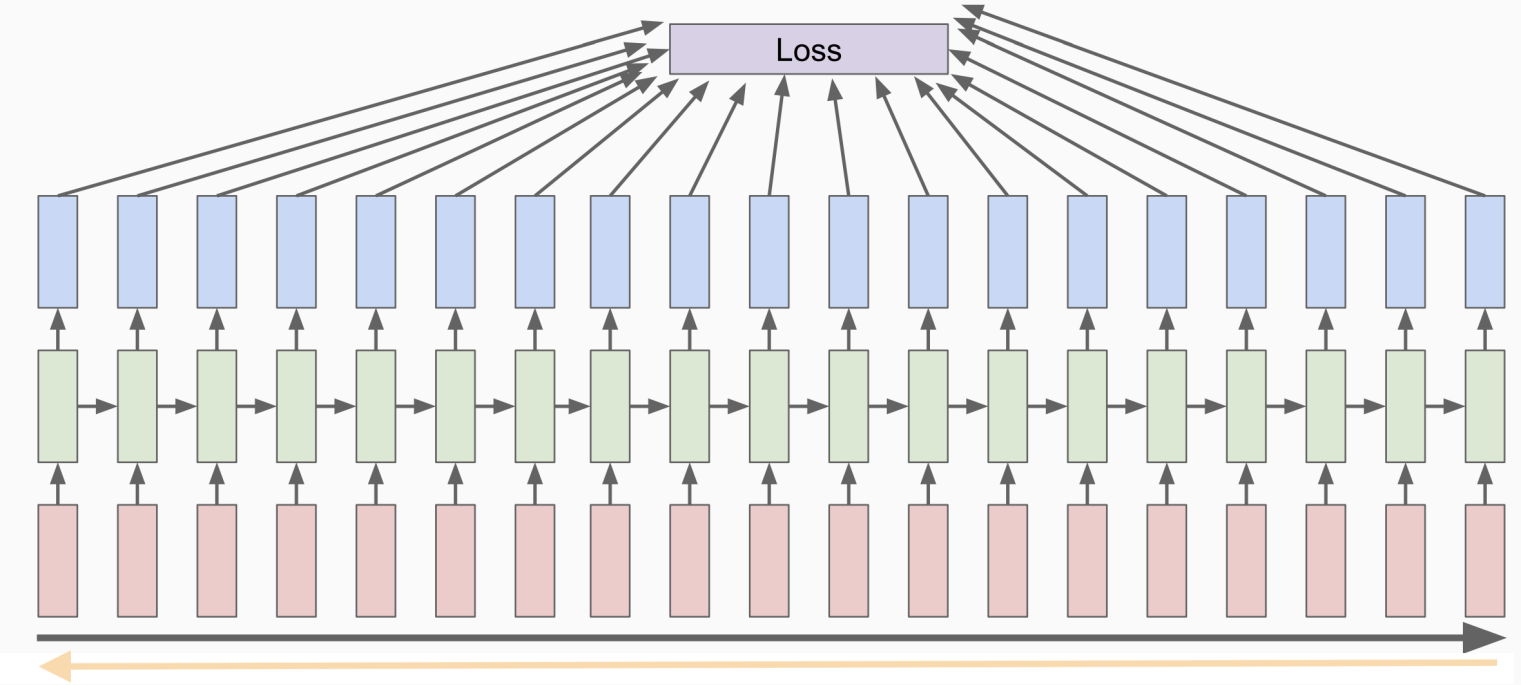


Deep RNN

- Multiple layers per time step (a feature hierarchy)
- Higher learning capacity
- Requires a lot more training data

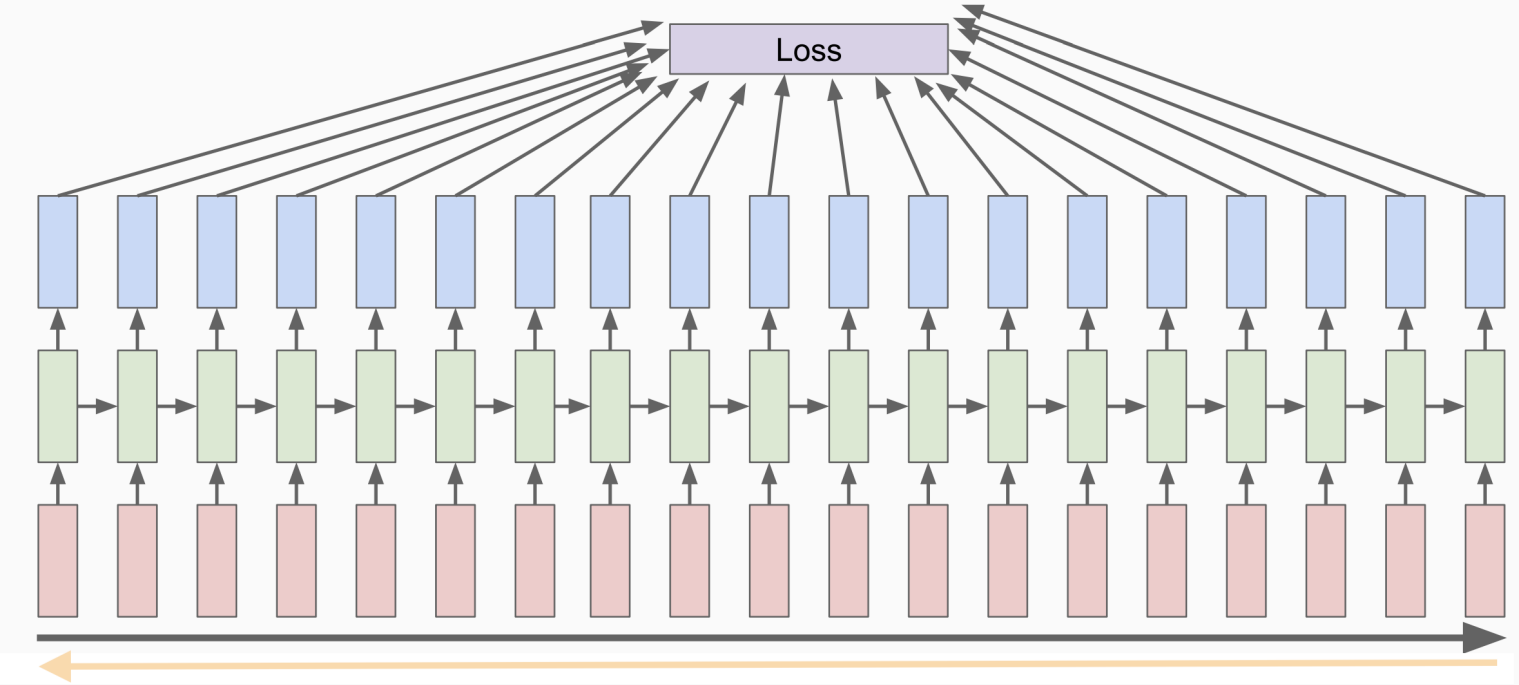


Learning phase for RNN



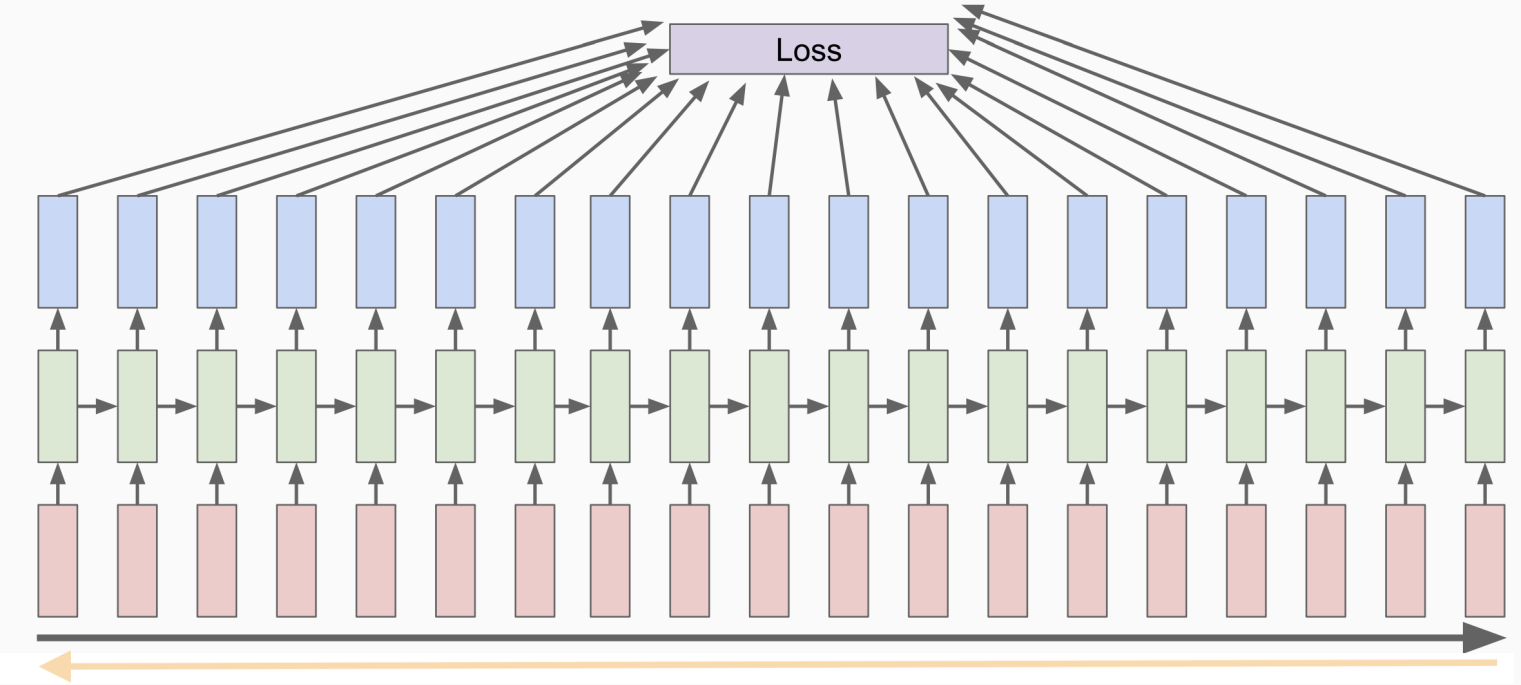
- Similar to standard backprop for training a traditional NN

Learning phase for RNN



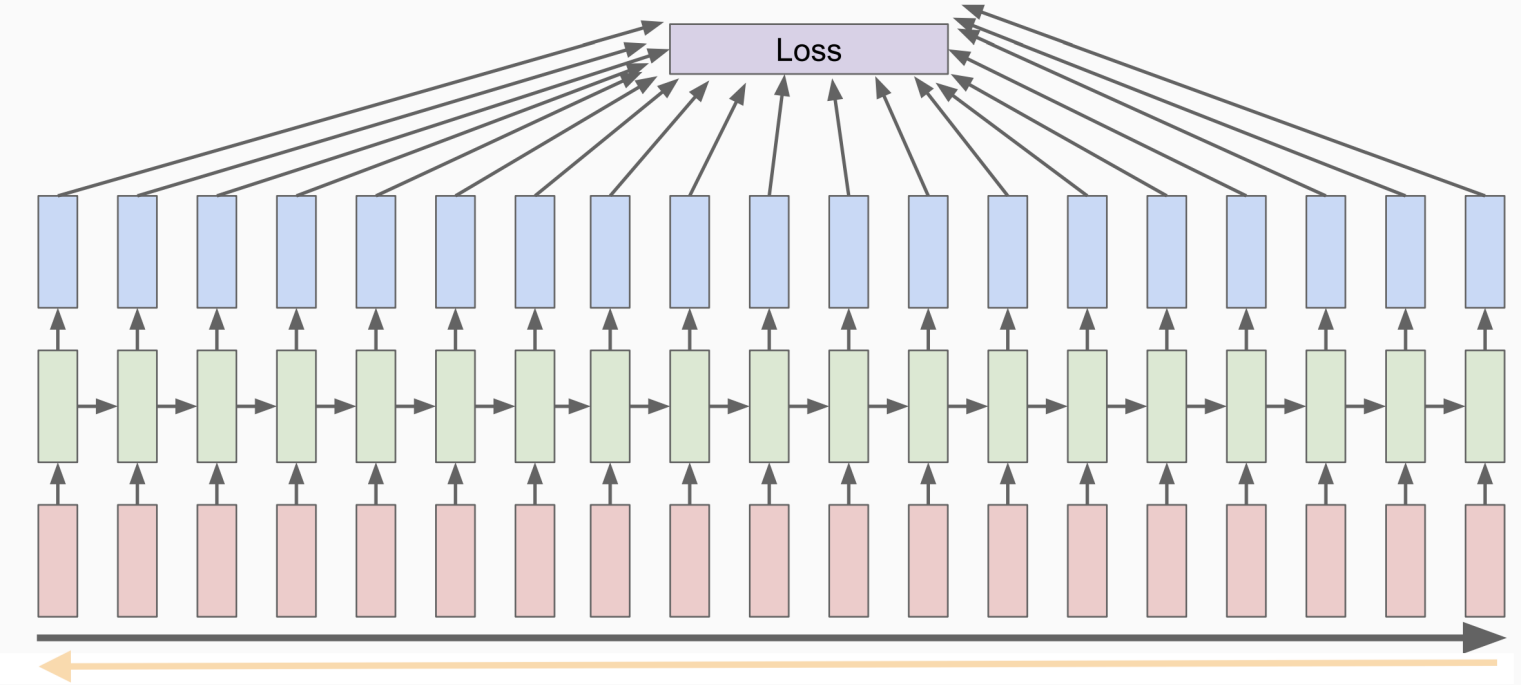
- Similar to standard backprop for training a traditional NN
- Take into account that parameters are shared by all steps in the network

Learning phase for RNN



- Similar to standard backprop for training a traditional NN
- Take into account that parameters are shared by all steps in the network
- Forward through the entire sequence to compute the loss

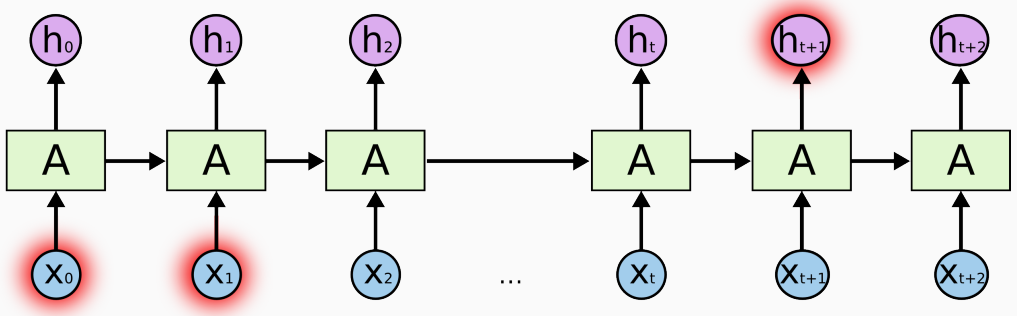
Learning phase for RNN



- Similar to standard backprop for training a traditional NN
- Take into account that parameters are shared by all steps in the network
- Forward through the entire sequence to compute the loss
- Backward through the entire sequence to compute gradients

Language generating RNN: limitations

- Vanilla RNN have difficulties learning long-term dependencies

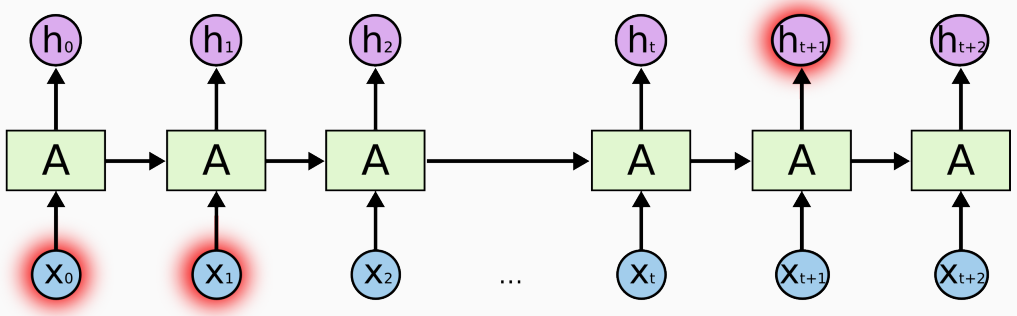


`I grew up in France ... I speak fluent ???`

(we need the context of France from further back)

Language generating RNN: limitations

- Vanilla RNN have difficulties learning long-term dependencies



`I grew up in France ... I speak fluent ???`

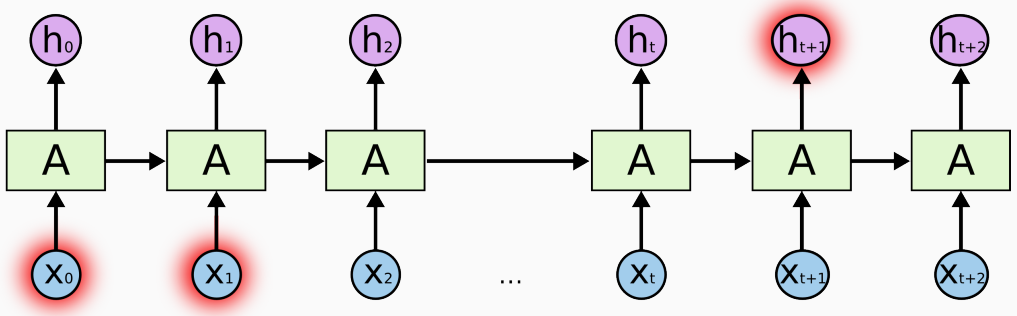
(we need the context of France from further back)

- Vanishing/exploding gradient problem

$$\underbrace{\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|}_{\| W_{hh}^\top \text{diag}(\sigma'(W_{hh} h_{t-1} + W_{xh} x_t)) \|} \sim \eta \Rightarrow \left\| \frac{\partial h_T}{\partial h_k} \right\| = \left\| \prod_{t=k+1}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right\| \sim \eta^{T-k}$$

Language generating RNN: limitations

- Vanilla RNN have difficulties learning long-term dependencies



`I grew up in France ... I speak fluent ???`

(we need the context of France from further back)

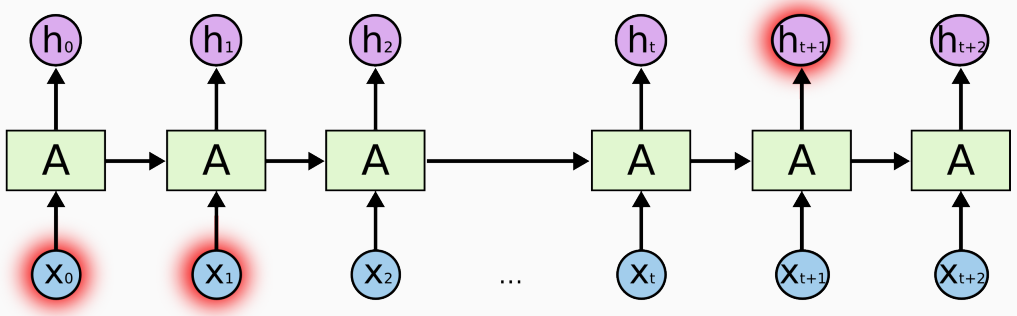
- Vanishing/exploding gradient problem

$$\underbrace{\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|}_{\| W_{hh}^\top \text{diag}(\sigma'(W_{hh} h_{t-1} + W_{xh} x_t)) \|} \sim \eta \Rightarrow \left\| \frac{\partial h_T}{\partial h_k} \right\| = \left\| \prod_{t=k+1}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right\| \sim \eta^{T-k}$$

- As $T - k$ increases, the contribution of the k-th term to the gradient decreases exponentially fast

Language generating RNN: limitations

- Vanilla RNN have difficulties learning long-term dependencies



I grew up in France ... I speak fluent ???

(we need the context of France from further back)

- Vanishing/exploding gradient problem

$$\underbrace{\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|}_{\| W_{hh}^\top \text{diag}(\sigma'(W_{hh} h_{t-1} + W_{xh} x_t)) \|} \sim \eta \Rightarrow \left\| \frac{\partial h_T}{\partial h_k} \right\| = \left\| \prod_{t=k+1}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right\| \sim \eta^{T-k}$$

- As $T - k$ increases, the contribution of the k-th term to the gradient decreases exponentially fast
- Certain types of RNNs are specifically designed to get around them

GRU (Gated Recurrent Unit)

- Interpreting the hidden state as the memory of a recurrent unit, decide whether certain units are worth memorizing (in which case the state is updated), and others are worth forgetting (in which case the state is reset)

GRU (Gated Recurrent Unit)

- Interpreting the hidden state as the memory of a recurrent unit, decide whether certain units are worth memorizing (in which case the state is updated), and others are worth forgetting (in which case the state is reset)

GRU (Gated Recurrent Unit)

- Interpreting the hidden state as the memory of a recurrent unit, decide whether certain units are worth memorizing (in which case the state is updated), and others are worth forgetting (in which case the state is reset)
- Define two gating operations, called "reset" and "update":

$$r_t = \sigma\left(W_{rx}x_t + W_{rh}h_{t-1}\right) \qquad z_t = \sigma\left(W_{zx}x_t + W_{zh}h_{t-1}\right)$$

GRU (Gated Recurrent Unit)

- Interpreting the hidden state as the memory of a recurrent unit, decide whether certain units are worth memorizing (in which case the state is updated), and others are worth forgetting (in which case the state is reset)
- Define two gating operations, called "reset" and "update":

$$r_t = \sigma\left(W_{rx}x_t + W_{rh}h_{t-1}\right) \qquad z_t = \sigma\left(W_{zx}x_t + W_{zh}h_{t-1}\right)$$

- Instead of $h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1}\right)$, consider

$$\tilde{h}_t = \sigma\left(W_{hx}x_t + W_{hh}\left(h_{t-1} \odot r_t\right)\right)$$

GRU (Gated Recurrent Unit)

- Interpreting the hidden state as the memory of a recurrent unit, decide whether certain units are worth memorizing (in which case the state is updated), and others are worth forgetting (in which case the state is reset)
- Define two gating operations, called "reset" and "update":

$$r_t = \sigma\left(W_{rx}x_t + W_{rh}h_{t-1}\right) \qquad z_t = \sigma\left(W_{zx}x_t + W_{zh}h_{t-1}\right)$$

- Instead of $h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1}\right)$, consider

$$\tilde{h}_t = \sigma\left(W_{hx}x_t + W_{hh}\left(h_{t-1} \odot r_t\right)\right)$$

  - If the reset gate $\simeq 1$, then this looks like a regular RNN unit (i.e., we retain memory)

GRU (Gated Recurrent Unit)

- Interpreting the hidden state as the memory of a recurrent unit, decide whether certain units are worth memorizing (in which case the state is updated), and others are worth forgetting (in which case the state is reset)
- Define two gating operations, called "reset" and "update":

$$r_t = \sigma\left(W_{rx}x_t + W_{rh}h_{t-1}\right) \qquad z_t = \sigma\left(W_{zx}x_t + W_{zh}h_{t-1}\right)$$

- Instead of $h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1}\right)$, consider

$$\tilde{h}_t = \sigma\left(W_{hx}x_t + W_{hh}\left(h_{t-1} \odot r_t\right)\right)$$

  - If the reset gate $\simeq 1$, then this looks like a regular RNN unit (i.e., we retain memory)
  - If the reset gate $\simeq 0$, then this looks like a regular perceptron/dense layer (i.e., we forget)

GRU (Gated Recurrent Unit)

- Interpreting the hidden state as the memory of a recurrent unit, decide whether certain units are worth memorizing (in which case the state is updated), and others are worth forgetting (in which case the state is reset)
- Define two gating operations, called "reset" and "update":

$$r_t = \sigma\left(W_{rx}x_t + W_{rh}h_{t-1}\right) \qquad z_t = \sigma\left(W_{zx}x_t + W_{zh}h_{t-1}\right)$$

- Instead of $h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1}\right)$, consider

$$\tilde{h}_t = \sigma\left(W_{hx}x_t + W_{hh}\left(h_{t-1} \odot r_t\right)\right)$$

  - If the reset gate $\simeq 1$, then this looks like a regular RNN unit (i.e., we retain memory)
  - If the reset gate $\simeq 0$, then this looks like a regular perceptron/dense layer (i.e., we forget)
- the update gate tells us how much memory retention versus forgetting needs to happen

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

Long-Short Term Memory

$$h_t = g\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right) \qquad \text{(memory)}$$

$$y_t = \text{softmax}\left(W_{yh}h_t + b_y\right) \qquad \text{(used as feature for prediction)}$$

Long-Short Term Memory

$$g_t = g\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right) \quad \text{(input modulation gate)}$$

$$c_t = g_t \quad \text{(place memory in a cell unit c)}$$

$$h_t = c_t$$

$$y_t = \text{softmax}\left(W_{yh}h_t + b_y\right) \quad \text{(use } h_t \text{ for prediction)}$$

Long-Short Term Memory

$$g_t = g\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right) \qquad \text{(input modulation gate)}$$

$$c_t = c_{t-1} + g_t \qquad \text{(the cell keeps track of long term)}$$

$$h_t = c_t$$

$$y_t = \text{softmax}\left(W_{yh}h_t + b_y\right)$$

Long-Short Term Memory

$$f_t = \text{sigm}\,(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \qquad \text{(forget gate)}$$

$$g_t = g\,(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \qquad \text{(input modulation gate)}$$

$$c_t = f_t \odot c_{t-1} + g_t \qquad \text{(but can forget some of its memories)}$$

$$h_t = c_t$$

$$y_t = \text{softmax}\,(W_{yh}h_t + b_y)$$

Long-Short Term Memory

$$i_t = \text{sigm}\,(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \quad \text{(input gate)}$$

$$f_t = \text{sigm}\,(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \quad \text{(forget gate)}$$

$$g_t = g\,(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \quad \text{(input modulation gate)}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad \text{(but can forget some of its memories)}$$

$$h_t = c_t$$

$$y_t = \text{softmax}\,(W_{yh}h_t + b_y)$$

Long-Short Term Memory

$$o_t = \text{sigm}\,(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \quad \text{(output gate)}$$

$$i_t = \text{sigm}\,(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \quad \text{(input gate)}$$

$$f_t = \text{sigm}\,(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \quad \text{(forget gate)}$$

$$g_t = g\,(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \quad \text{(input modulation gate)}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad \text{(but can forget some of its memories)}$$

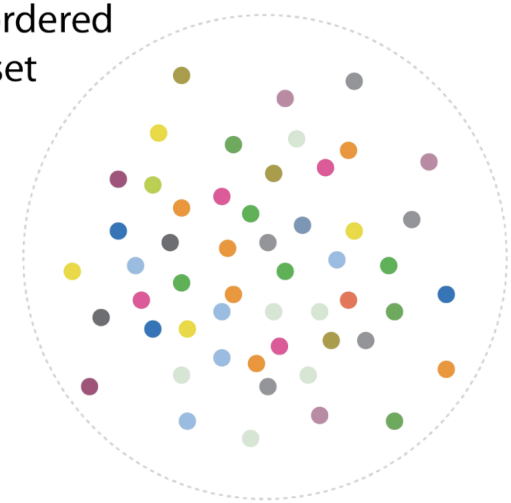
$$h_t = o_t \odot c_t \quad \text{(weight memory for generating feature)}$$

$$y_t = \text{softmax}\,(W_{yh}h_t + b_y)$$

- There are many variants, but this is the general idea

Preparation of sequence data

Not ordered
Dataset



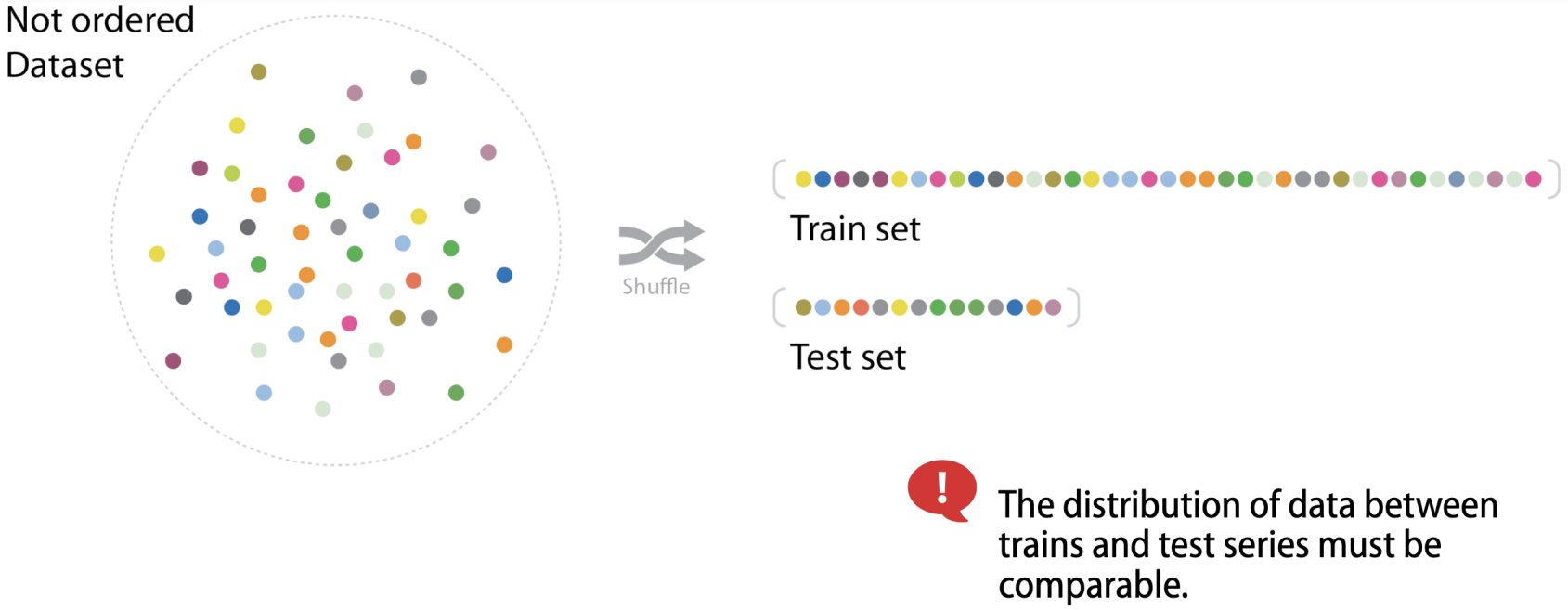
Shuffle

Train set

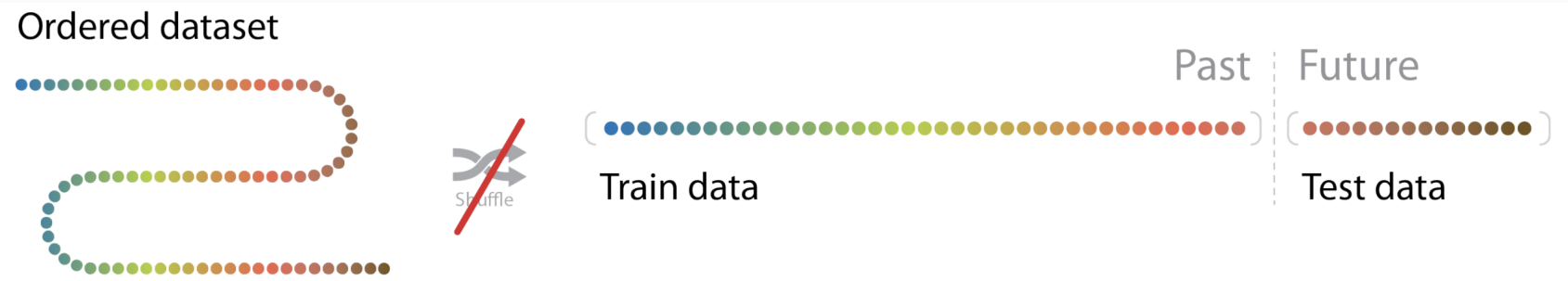
Test set

The distribution of data between trains and test series must be comparable.

Preparation of sequence data

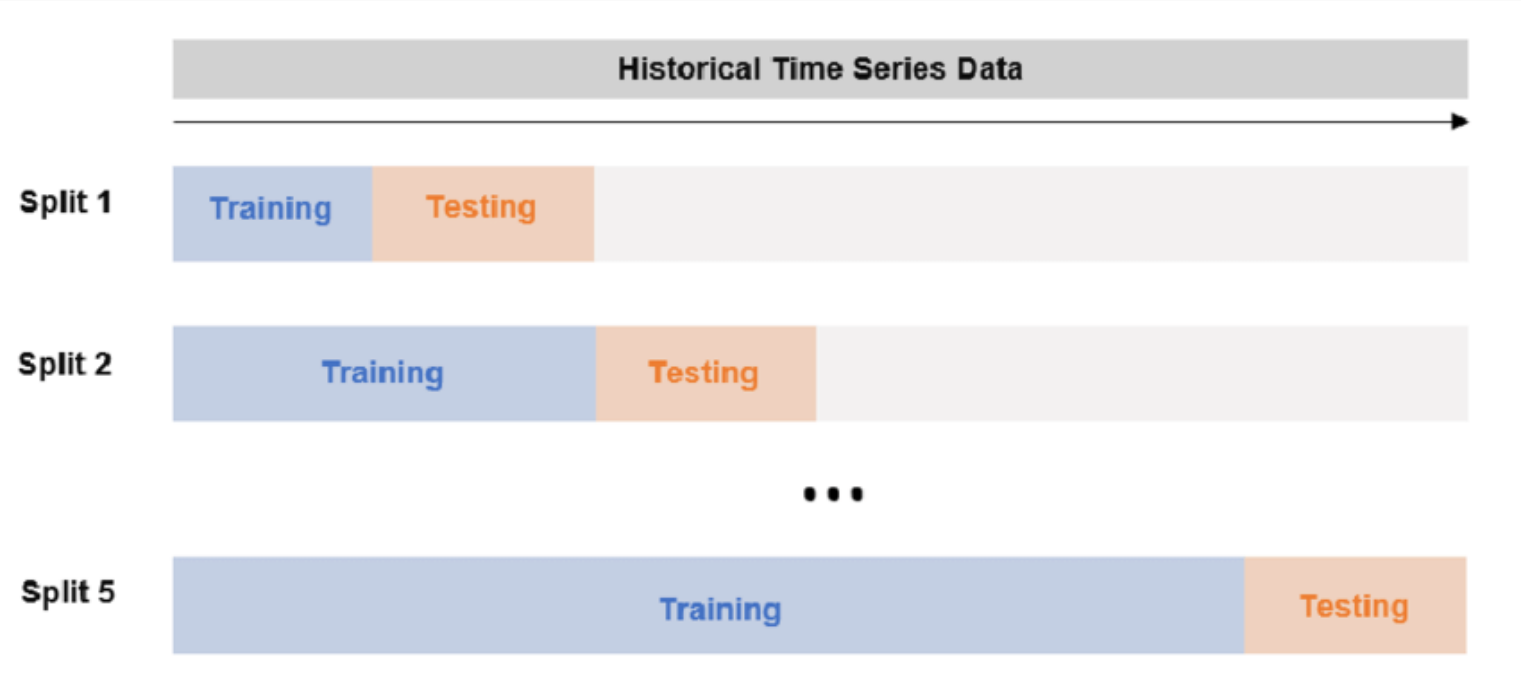


- Can you predict the past with the future? Beware of splitting time series!



source: fidle-cnrs

Cross-validation for time series



Sommaire

Word representation

Recurrent NN

Transformers

Self-attention for what?

So far

- RNN maps a sequence to a single output or a sequence

Self-attention for what?

So far

- RNN maps a sequence to a single output or a sequence
- Self-attention maps a set of inputs $\{x_1, \ldots, x_N\}$ to a set of outputs $\{y_1, \ldots, y_N\}$

Self-attention for what?

So far

- RNN maps a sequence to a single output or a sequence
- Self-attention maps a set of inputs $\{x_1, \ldots, x_N\}$ to a set of outputs $\{y_1, \ldots, y_N\}$
- This is an embedding

A preliminary version of self-attention

$$y_i = \sum_{j=1}^{N} w_{ij} x_j$$

- Each output is a weighted average of all inputs where the weights w_{ij} are row-normalized such that they sum to 1

A preliminary version of self-attention

$$y_i = \sum_{j=1}^{N} w_{ij} x_j$$

- Each output is a weighted average of all inputs where the weights w_{ij} are row-normalized such that they sum to 1
- The weights are directly derived from the inputs, e.g.

$$w'_{ij} = x_i^\top x_j \qquad \left. w_{ij} = \frac{\exp(w'_{ij})}{\sum_{j'} \exp(w'_{ij'})} \right\} = \text{softmax}\left((w'_{ij})_j\right)$$

A preliminary version of self-attention

$$y_i = \sum_{j=1}^{N} w_{ij} x_j$$

- Each output is a weighted average of all inputs where the weights w_{ij} are row-normalized such that they sum to 1
- The weights are directly derived from the inputs, e.g.

$$w'_{ij} = x_i^\top x_j \qquad w_{ij} = \left.\frac{\exp(w'_{ij})}{\sum_{j'} \exp(w'_{ij'})}\right\} = \text{softmax}\left((w'_{ij})_j\right)$$

  - ▷ *Here, everything is deterministic, for now nothing is learned*

A preliminary version of self-attention

$$y_i = \sum_{j=1}^{N} w_{ij} x_j$$

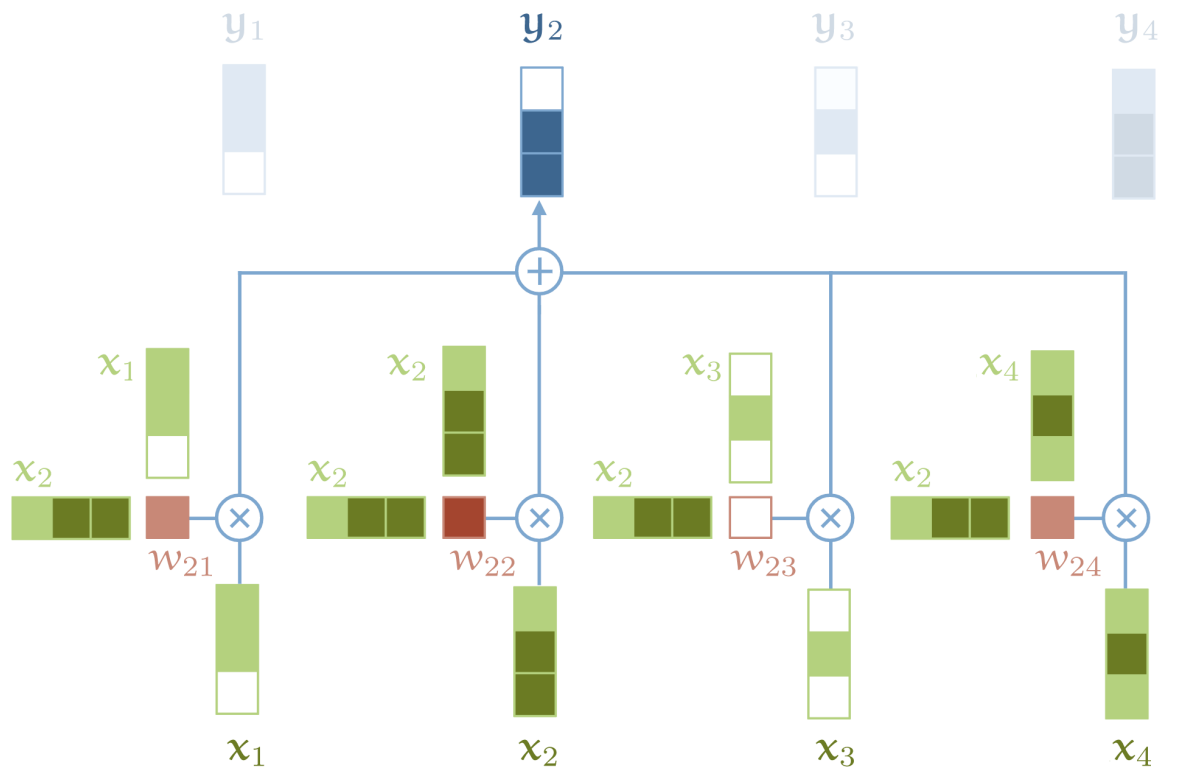
- Each output is a weighted average of all inputs where the weights w_{ij} are row-normalized such that they sum to 1
- The weights are directly derived from the inputs, e.g.

$$w'_{ij} = x_i^\top x_j \qquad w_{ij} = \frac{\exp(w'_{ij})}{\sum_{j'} \exp(w'_{ij'})} \Bigg\} = \text{softmax}\left((w'_{ij})_j\right)$$

  - *Here, everything is deterministic, for now nothing is learned*
  - *The operation is permutation-invariant (but this can be fixed, see later)*

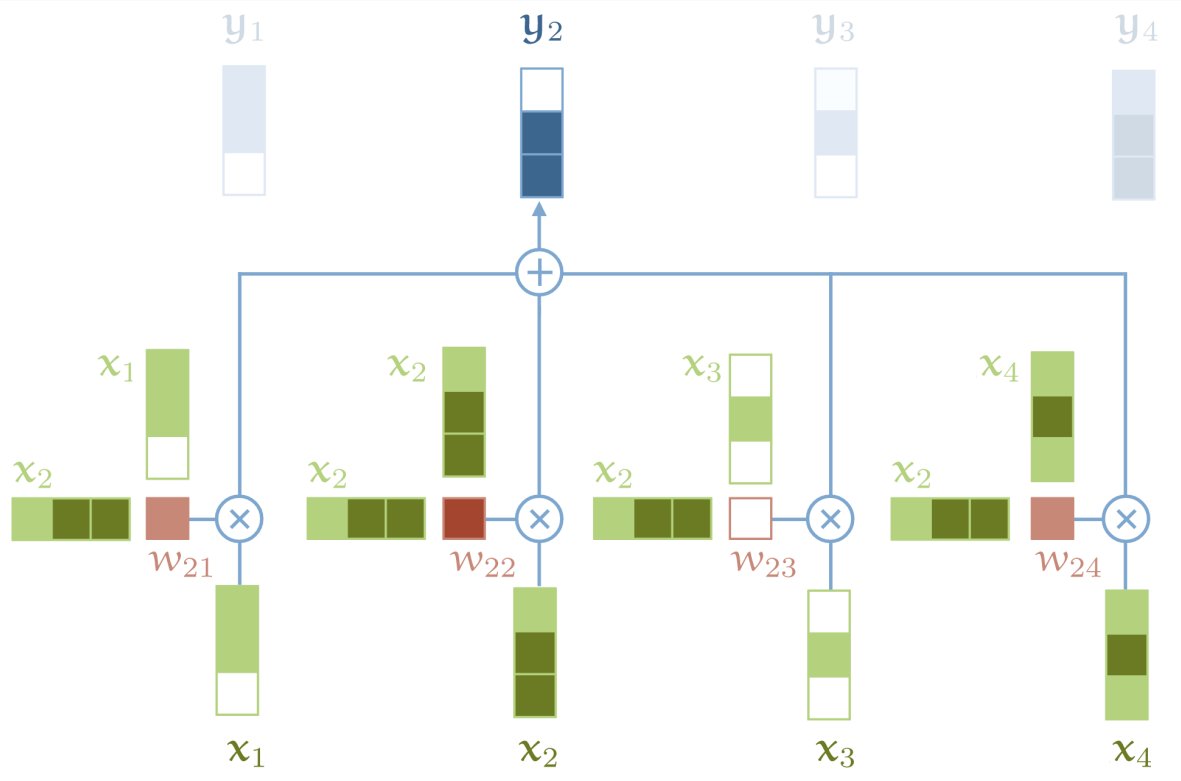
A preliminary version of self-attention

from http://peterbloem.nl/blog/transformers



A preliminary version of self-attention

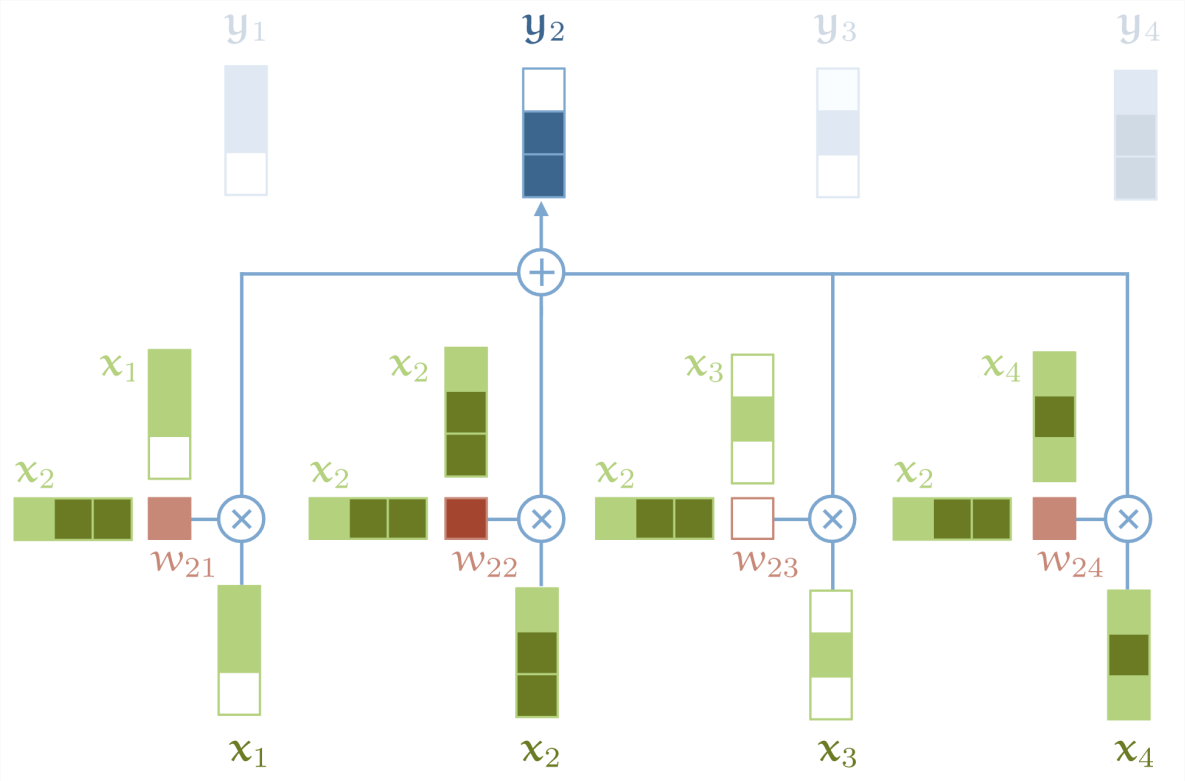
from http://peterbloem.nl/blog/transformers



- A few other ingredients are needed for a complete transformer

A preliminary version of self-attention

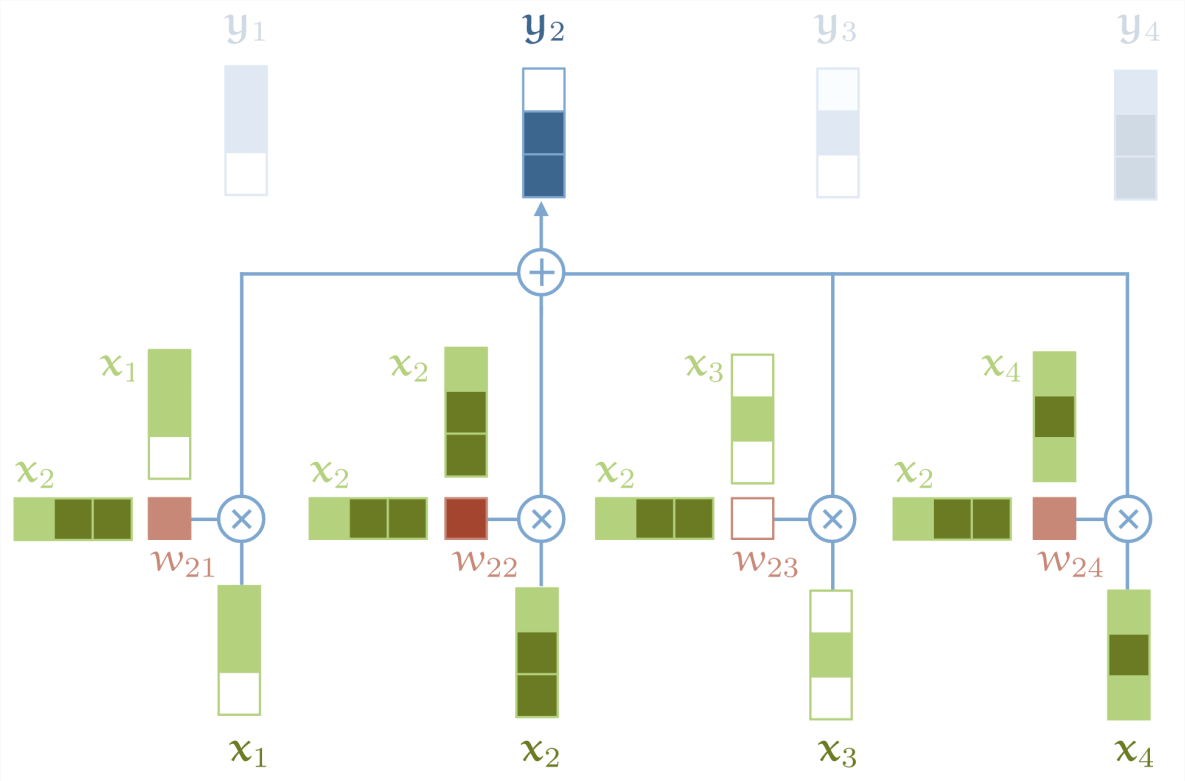
from http://peterbloem.nl/blog/transformers



- A few other ingredients are needed for a complete transformer
- But this is the only operation in the whole architecture that propagates information between vectors

A preliminary version of self-attention

from http://peterbloem.nl/blog/transformers



- A few other ingredients are needed for a complete transformer
- But this is the only operation in the whole architecture that propagates information between vectors
  - ▷ *Every other operation in the transformer is applied to each vector in the input sequence without interactions between vectors*

A preliminary version of self-attention

What's the point?

- Restriction of self-attention to linear models

A preliminary version of self-attention

What's the point?

- Restriction of self-attention to linear models
- Example of Neural Machine Translation (NMT)

A preliminary version of self-attention

What's the point?

- Restriction of self-attention to linear models
- Example of Neural Machine Translation (NMT)
- Task: translate "`the dog sat on the couch`" from English to French

A preliminary version of self-attention

What's the point?

- Restriction of self-attention to linear models
- Example of Neural Machine Translation (NMT)
- Task: translate "`the dog sat on the couch`" from English to French
  - A lot of redundancy in natural languages

A preliminary version of self-attention

What's the point?

- Restriction of self-attention to linear models
- Example of Neural Machine Translation (NMT)
- Task: translate "`the dog sat on the couch`" from English to French
  - A lot of redundancy in natural languages
  - '`the`' '`on`' are common, not informative, not correlated

A preliminary version of self-attention

What's the point?

- Restriction of self-attention to linear models
- Example of Neural Machine Translation (NMT)
- Task: translate "`the dog sat on the couch`" from English to French
  - A lot of redundancy in natural languages
  - '`the`' '`on`' are common, not informative, not correlated
  - '`dog`' '`couch`' are similar, both nouns, can be grouped according to subject-object relationships or subject-predicate relationships

A preliminary version of self-attention

What's the point?

- Restriction of self-attention to linear models
- Example of Neural Machine Translation (NMT)
- Task: translate "`the dog sat on the couch`" from English to French
  - A lot of redundancy in natural languages
  - '`the`' '`on`' are common, not informative, not correlated
  - '`dog`' '`couch`' are similar, both nouns, can be grouped according to subject-object relationships or subject-predicate relationships
- It would be useful if the model automatically "grouped" similar words together

A preliminary version of self-attention

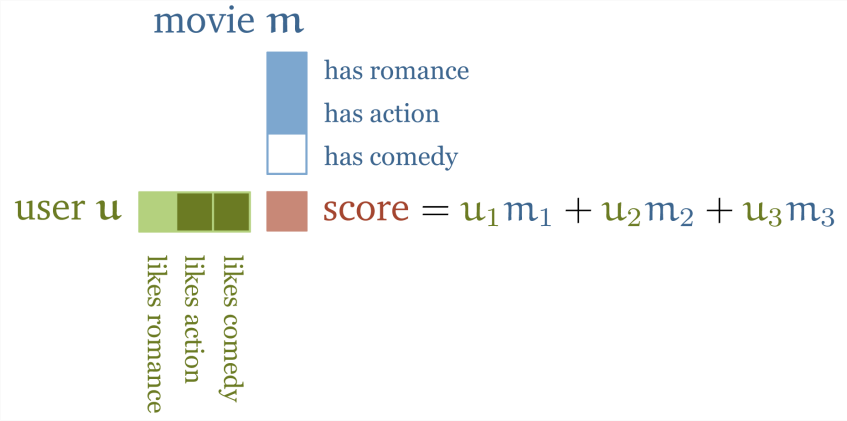
What's the point?

- Restriction of self-attention to linear models
- Example of Neural Machine Translation (NMT)
- Task: translate "`the dog sat on the couch`" from English to French
  - A lot of redundancy in natural languages
  - '`the`' '`on`' are common, not informative, not correlated
  - '`dog`' '`couch`' are similar, both nouns, can be grouped according to subject-object relationships or subject-predicate relationships
- It would be useful if the model automatically "grouped" similar words together
- Possible by the scalar products

A preliminary version of self-attention

Another example: movie recommendation

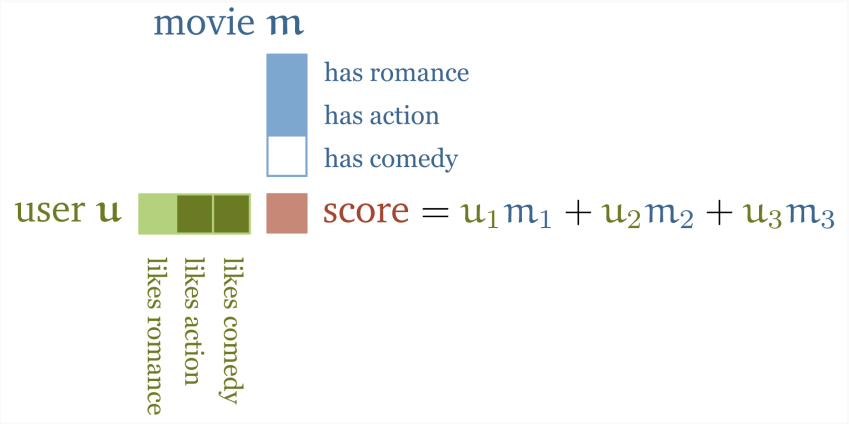
1. create manual features for movies and for users



A preliminary version of self-attention

Another example: movie recommendation

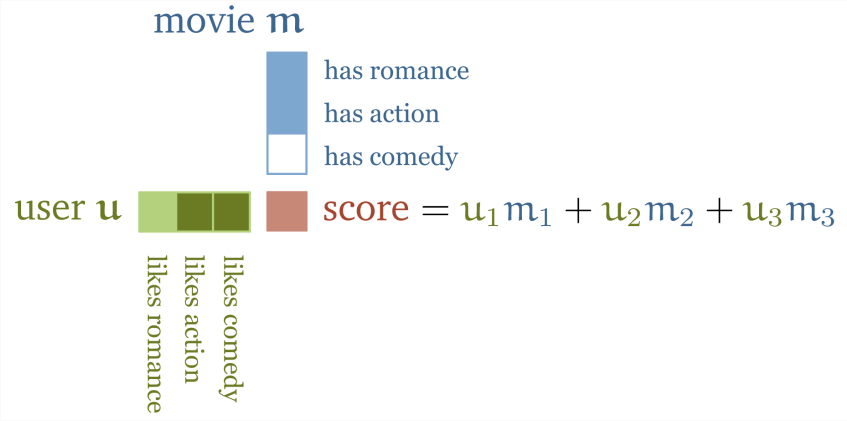
1. create manual features for movies and for users
   - how much romance there is in the movie, and how much action,



A preliminary version of self-attention

Another example: movie recommendation

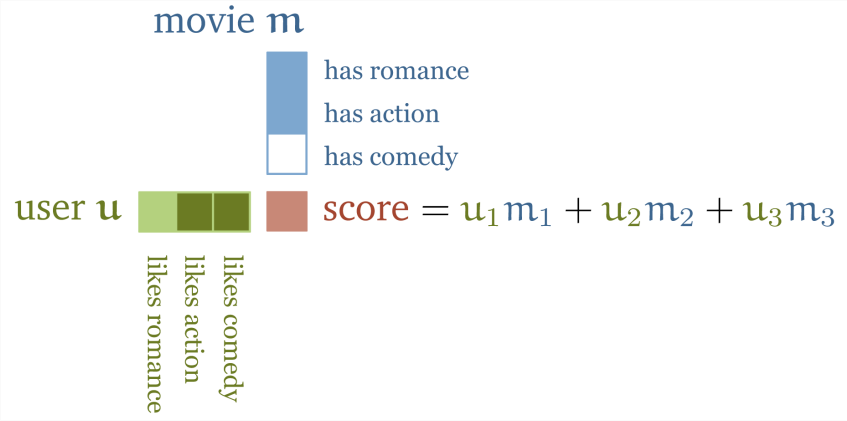
1. create manual features for movies and for users
   - how much romance there is in the movie, and how much action,
   - how much they enjoy romantic movies and how much they enjoy action-based movies



A preliminary version of self-attention

Another example: movie recommendation

1. create manual features for movies and for users
   - how much romance there is in the movie, and how much action,
   - how much they enjoy romantic movies and how much they enjoy action-based movies

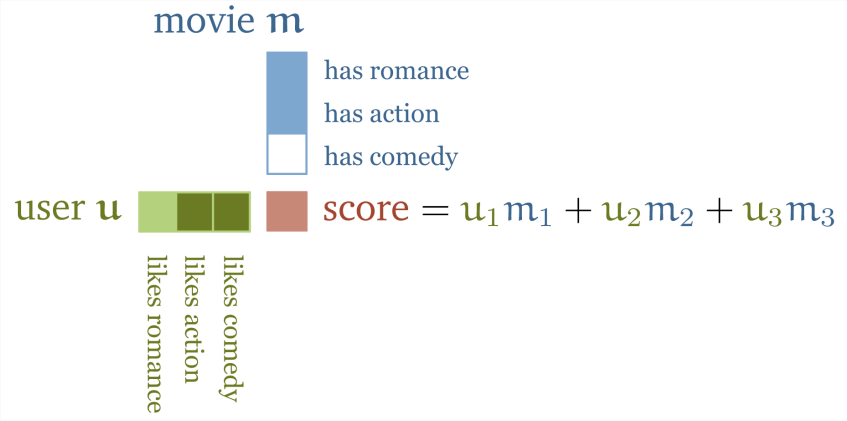


2. The dot product between the two feature vectors gives a score for how well the attributes of the movie match what the user enjoys

A preliminary version of self-attention

Another example: movie recommendation

1. create manual features for movies and for users
   - how much romance there is in the movie, and how much action,
   - how much they enjoy romantic movies and how much they enjoy action-based movies

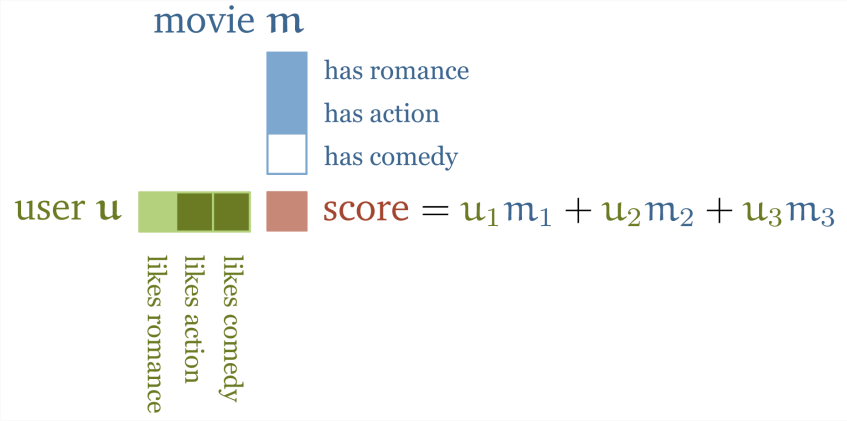


2. The dot product between the two feature vectors gives a score for how well the attributes of the movie match what the user enjoys
3. Replace user feature u by movies that she liked

A preliminary version of self-attention

Another example: movie recommendation

1. create manual features for movies and for users
   - how much romance there is in the movie, and how much action,
   - how much they enjoy romantic movies and how much they enjoy action-based movies



2. The dot product between the two feature vectors gives a score for how well the attributes of the movie match what the user enjoys
3. Replace user feature u by movies that she liked

Dot product $\approx$ relations between objects

A step further

Going back to the NMT example:

- Input: a sequence of words $x_1, \ldots x_N$

A step further

Going back to the NMT example:

- Input: a sequence of words $x_1, \ldots x_N$
- Embedding layer: apply to each word x_i an embedding v_i (the values that we will learn)

A step further

Going back to the NMT example:

- Input: a sequence of words $x_1, \dots x_N$
- Embedding layer: apply to each word x_i an embedding v_i (the values that we will learn)
  - ▷ Learning the values v_i is learning how "related" two words are

A step further

Going back to the NMT example:

- Input: a sequence of words $x_1, \dots x_N$
- Embedding layer: apply to each word x_i an embedding v_i (the values that we will learn)
  - ▷ Learning the values v_i is learning how "related" two words are
  - ▷ Entirely determined by the learning task

A step further

Going back to the NMT example:

- Input: a sequence of words $x_1, \dots x_N$
- Embedding layer: apply to each word x_i an embedding v_i (the values that we will learn)
  - ▷ Learning the values v_i is learning how "related" two words are
  - ▷ Entirely determined by the learning task

Example: "The dog sleeps on the couch"

- 'The': not very relevant to the interpretation of the other words in the sentence

A step further

Going back to the NMT example:

- Input: a sequence of words $x_1, \dots x_N$
- Embedding layer: apply to each word x_i an embedding v_i (the values that we will learn)
  - ▷ Learning the values v_i is learning how "related" two words are
  - ▷ Entirely determined by the learning task

Example: "The dog sleeps on the couch"

- 'The': not very relevant to the interpretation of the other words in the sentence

▷ Desire 1: the embedding v_{The} should have a zero or negative scalar product with the other words

A step further

Going back to the NMT example:

- Input: a sequence of words $x_1, \dots x_N$
- Embedding layer: apply to each word x_i an embedding v_i (the values that we will learn)
  - ▷ Learning the values v_i is learning how "related" two words are
  - ▷ Entirely determined by the learning task

Example: "The dog sleeps on the couch"

- 'The': not very relevant to the interpretation of the other words in the sentence

▷ Desire 1: the embedding v_{The} should have a zero or negative scalar product with the other words

- Helpful to interpret who sleeps

A step further

Going back to the NMT example:

- Input: a sequence of words $x_1, \dots x_N$
- Embedding layer: apply to each word x_i an embedding v_i (the values that we will learn)
  - ▷ Learning the values v_i is learning how "related" two words are
  - ▷ Entirely determined by the learning task

Example: "`The dog sleeps on the couch`"

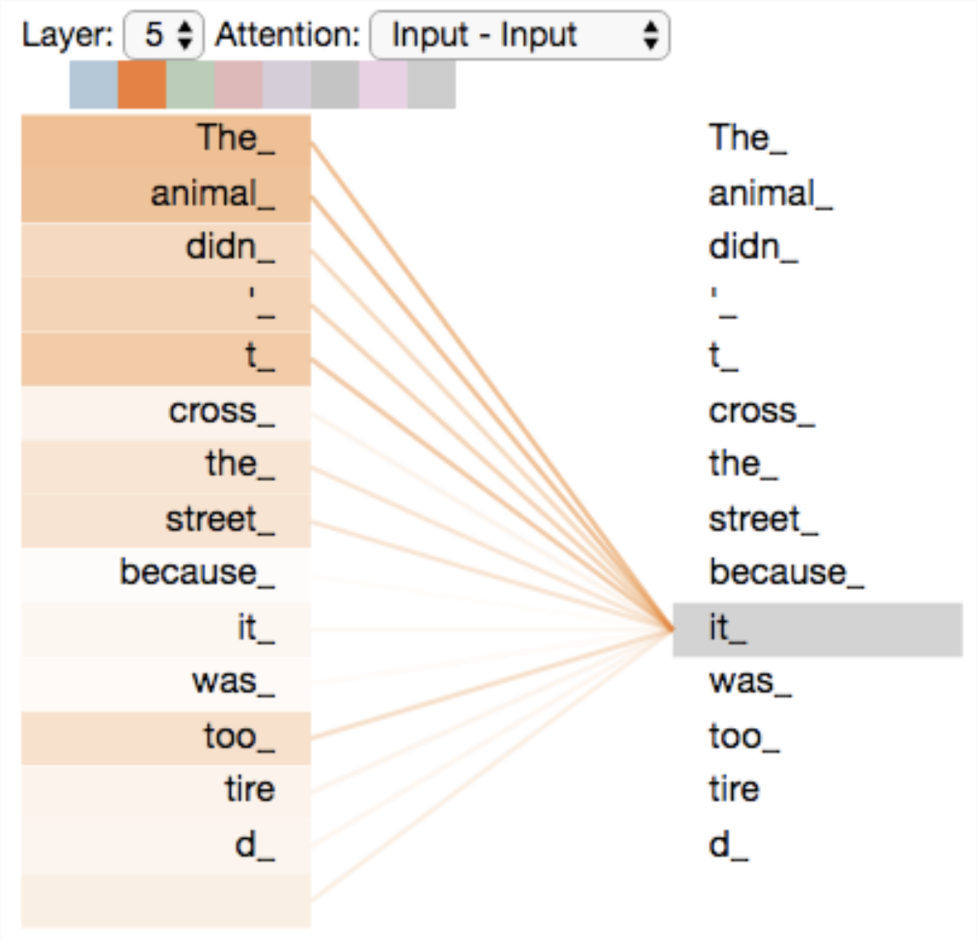
- '`The`': not very relevant to the interpretation of the other words in the sentence

▷ Desire 1: the embedding $v_{\texttt{The}}$ should have a zero or negative scalar product with the other words

- Helpful to interpret who sleeps

▷ Desire 2: for nouns like '`dog`' and verbs like '`sleeps`', learn an embedding $v_{\texttt{dog}}$ and $v_{\texttt{sleeps}}$ that have a high, positive dot product

Learning the embedding: attention weights



- Showing the scalar products between the learned embedding v
- As we are encoding the word "`it`", part of the attention mechanism was focusing on "`the animal`"

Towards a real self-attention layer

In the toy self-attention version, every input vector x_i is used in three different ways in the self attention operation

$$w'_{ij} = x_i{}^\top x_j, \quad w_{ij} = \text{softmax}((w'_{ij})_j), \quad y_i = \sum_{j=1}^{N} w_{ij} x_j$$

Towards a real self-attention layer

In the toy self-attention version, every input vector x_i is used in three different ways in the self attention operation

$$w'_{ij} = x_i{}^\top x_j, \quad w_{ij} = \text{softmax}((w'_{ij})_j), \quad y_i = \sum_{j=1}^{N} w_{ij} x_j$$

- **(Query)** x_i is compared to every other vector to establish the weights for its own output y_i

Towards a real self-attention layer

In the toy self-attention version, every input vector x_i is used in three different ways in the self attention operation

$$w'_{ij} = x_i{}^\top x_j, \quad w_{ij} = \text{softmax}((w'_{ij})_j), \quad y_i = \sum_{j=1}^{N} w_{ij} x_j$$

- **(Query)** x_i is compared to every other vector to establish the weights for its own output y_i
- **(Key)** x_i is compared to every other vector to establish the weights for the output of the j-th vector y_j

Towards a real self-attention layer

In the toy self-attention version, every input vector x_i is used in three different ways in the self attention operation

$$w'_{ij} = x_i{}^\top x_j, \quad w_{ij} = \text{softmax}((w'_{ij})_j), \quad y_i = \sum_{j=1}^{N} w_{ij} x_j$$

- (Query) x_i is compared to every other vector to establish the weights for its own output y_i
- (Key) x_i is compared to every other vector to establish the weights for the output of the j-th vector y_j
- (Value) x_i is used as part of the weighted sum to compute each output vector once the weights have been established.

Towards a real self-attention layer

In the toy self-attention version, every input vector x_i is used in three different ways in the self attention operation

$$w'_{ij} = {x_i}^\top x_j, \quad w_{ij} = \text{softmax}((w'_{ij})_j), \quad y_i = \sum_{j=1}^{N} w_{ij} x_j$$

- **(Query)** x_i is compared to every other vector to establish the weights for its own output y_i
- **(Key)** x_i is compared to every other vector to establish the weights for the output of the j-th vector y_j
- **(Value)** x_i is used as part of the weighted sum to compute each output vector once the weights have been established.

These three roles are called the query, key, and value.

Towards a real self-attention layer

Make these roles distinct by adding a few dummy variables:

$$q_i = x_i \qquad \text{(Query)}$$
$$k_i = x_i \qquad \text{(Key)}$$
$$v_i = x_i \qquad \text{(Value)}$$

Towards a real self-attention layer

Make these roles distinct by adding a few dummy variables:

$$q_i = x_i \qquad \text{(Query)}$$
$$k_i = x_i \qquad \text{(Key)}$$
$$v_i = x_i \qquad \text{(Value)}$$

and then write out the output as:

$$w'_{ij} = q_i^\top k_j \qquad w_{ij} = \text{softmax}((w'_{ij})_j) \qquad y_i = \sum_{j=1}^{N} w_{ij} v_j$$

Towards a real self-attention layer

Make these roles distinct by adding a few dummy variables:

$$q_i = x_i \qquad \text{(Query)}$$
$$k_i = x_i \qquad \text{(Key)}$$
$$v_i = x_i \qquad \text{(Value)}$$

and then write out the output as:

$$w'_{ij} = q_i^\top k_j \qquad w_{ij} = \text{softmax}((w'_{ij})_j) \qquad y_i = \sum_{j=1}^{N} w_{ij} v_j$$

Then, we can use learnable parameters for each of these roles, for instance:

$$q_i = W_q x_i \qquad \text{(Query)}$$
$$k_i = W_k x_i \qquad \text{(Key)}$$
$$v_i = W_v x_i \qquad \text{(Value)}$$

where W_q, W_k, W_v are learnable projection matrices that defines the roles of each data point

An attention head

from http://peterbloem.nl/blog/transformers

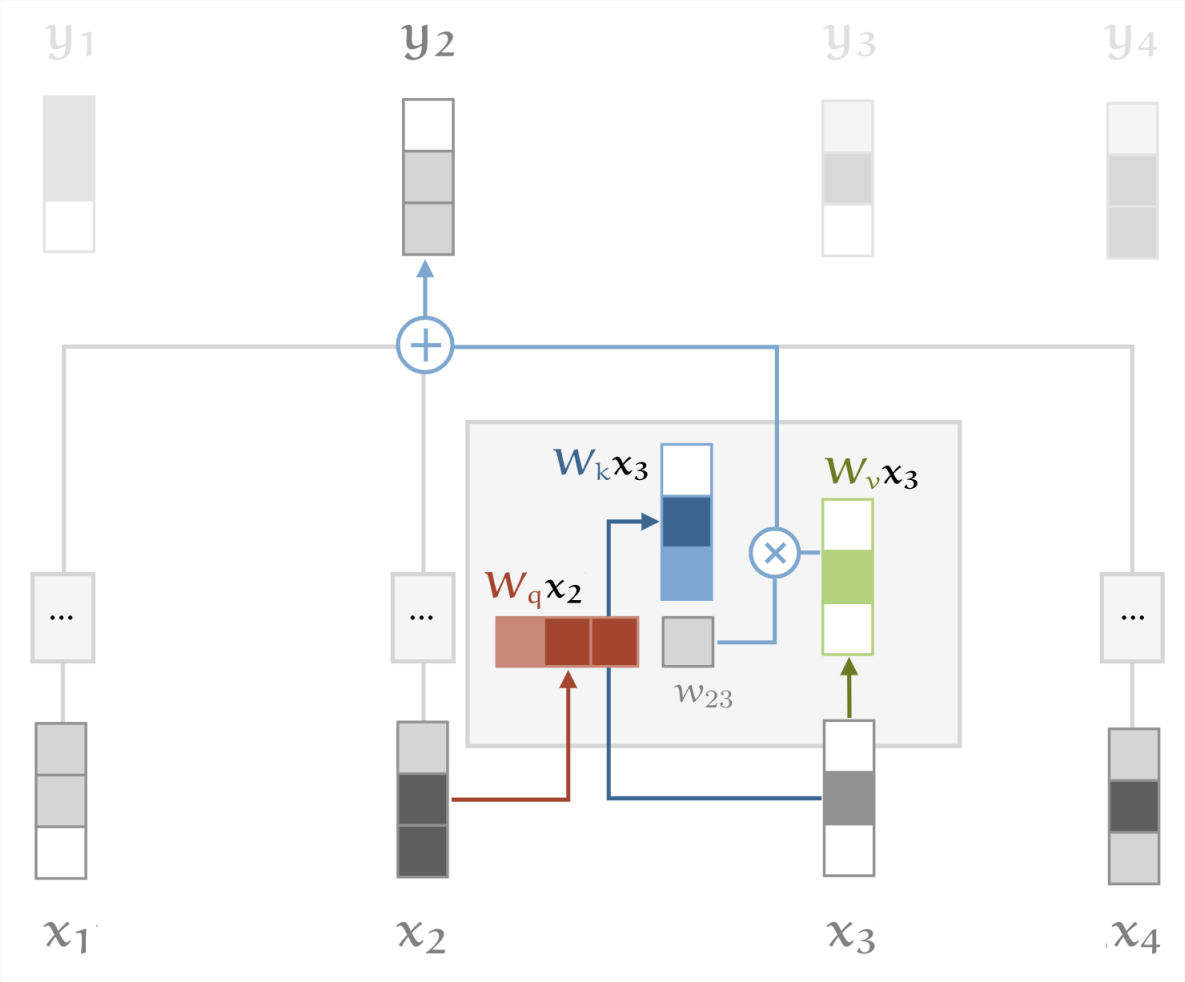


Figure 1: Illustration of the self-attention with key, query and value transformations

Scaling the dot product

- The dot product in attention weights is usually scaled

$$w'_{ij} = \frac{1}{\sqrt{\text{dimension of the embedding}}} q_i^\top k_j$$

where dimension of the embedding = size of q_i, k_i, v_i

Scaling the dot product

- The dot product in attention weights is usually scaled

$$w'_{ij} = \frac{1}{\sqrt{\text{dimension of the embedding}}} q_i^\top k_j$$

  where dimension of the embedding = size of q_i, k_i, v_i

- The softmax function can be sensitive to very large input values

  $\rightsquigarrow$ vanishing gradient / slow training

Scaling the dot product

- The dot product in attention weights is usually scaled

$$w'_{ij} = \frac{1}{\sqrt{\text{dimension of the embedding}}} q_i^\top k_j$$

  where dimension of the embedding = size of q_i, k_i, v_i

- The softmax function can be sensitive to very large input values

  $\rightsquigarrow$ vanishing gradient / slow training

- The average value of the dot product grows with the embedding dimension

Multi-head self attention layer

- Concatenate different self-attention mechanisms to give it more flexibility

Multi-head self attention layer

- Concatenate different self-attention mechanisms to give it more flexibility
- Index each head with $r = 1, 2, \ldots$

$$q_i^r = W_q^r x_i \qquad k_i^r = W_k^r x_i \qquad v_i^r = W_v^r x_i$$

$$(w')_{ij}^r = (q_i^r)^\top k_j^r \qquad w_{ij}^r = \text{softmax}((w')_{ij}^r)) \qquad y_i^r = \sum_{j=1}^{N} w_{ij}^r v_j^r$$

$$y_i = W_y \text{concat}\left(y_i^1, y_i^2, \ldots\right)$$

Multi-head self attention layer

- Concatenate different self-attention mechanisms to give it more flexibility
- Index each head with $r = 1, 2, \ldots$

$$q_i^r = W_q^r x_i \qquad k_i^r = W_k^r x_i \qquad v_i^r = W_v^r x_i$$

$$(w')_{ij}^r = (q_i^r)^\top k_j^r \qquad w_{ij}^r = \text{softmax}((w')_{ij}^r)) \qquad y_i^r = \sum_{j=1}^{N} w_{ij}^r v_j^r$$

$$y_i = W_y \text{concat}\left(y_i^1, y_i^2, \ldots\right)$$

$$(y_1, \ldots, y_N) = \textbf{Attn}\,(x_1, \ldots, x_N)$$

Multi-head self attention layer

- Concatenate different self-attention mechanisms to give it more flexibility
- Index each head with $r = 1, 2, \ldots$

$$q_i^r = W_q^r x_i \qquad k_i^r = W_k^r x_i \qquad v_i^r = W_v^r x_i$$

$$(w')_{ij}^r = (q_i^r)^\top k_j^r \qquad w_{ij}^r = \text{softmax}((w')_{ij}^r) \qquad y_i^r = \sum_{j=1}^{N} w_{ij}^r v_j^r$$

$$y_i = W_y \text{concat}\left(y_i^1, y_i^2, \ldots\right)$$

$$(y_1, \ldots, y_N) = \textbf{Attn}\,(x_1, \ldots, x_N)$$

- Trick to reduce dimension: use lower-dimensional matrices W_q^r, W_k^r and W_v^r: $\dim(x) \times h$ intead of $\dim(x) \times \dim(x)$

A multi-head attention

from http://peterbloem.nl/blog/transformers

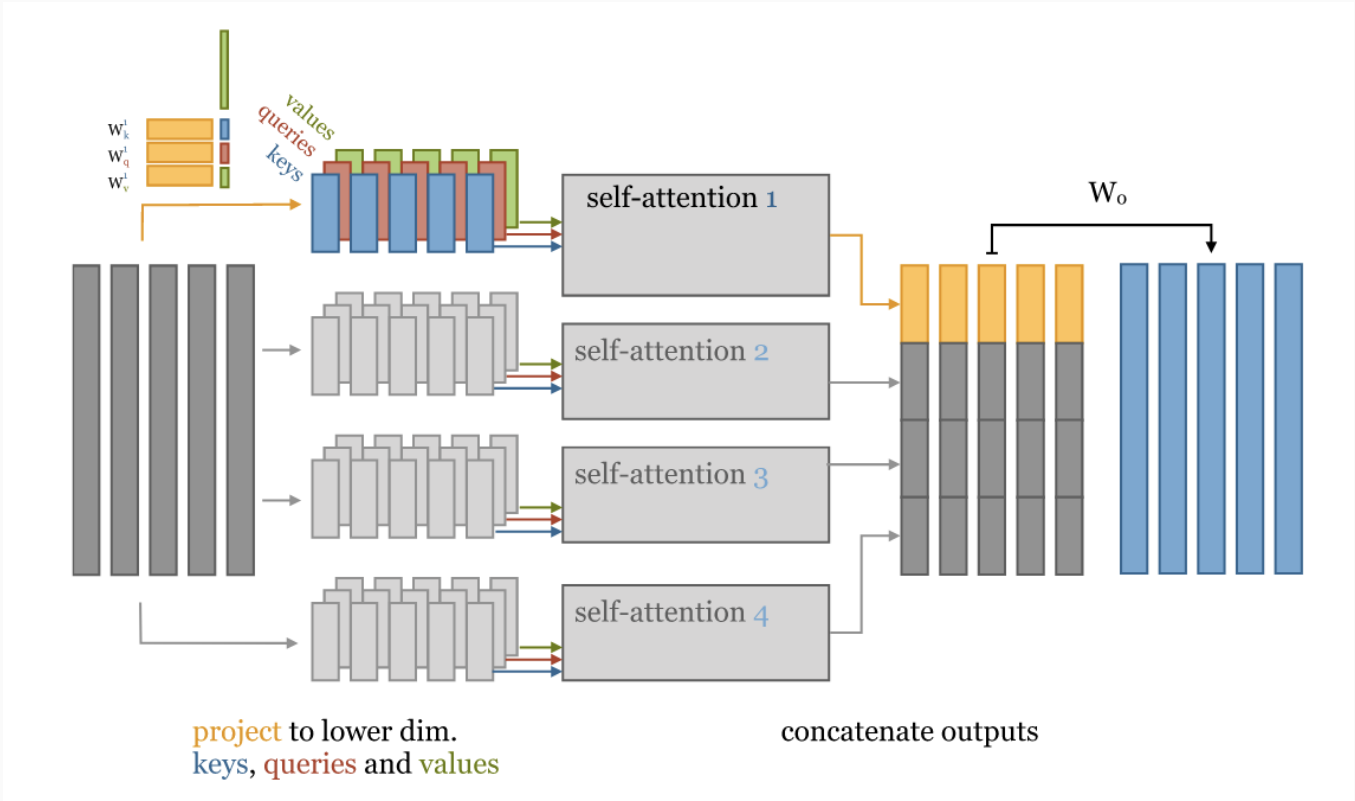


Figure 2: Illustration of multi-head self-attention with 4 heads. To get our keys, queries and values, we project the input down to vector sequences of smaller dimension.

On the vocabulary

- ‘key’, ‘query’, ‘value’ come from a key-value data structure (search engine)

On the vocabulary

- ‘key’, ‘query’, ‘value’ come from a key-value data structure (search engine)

  If we give a query key and match it to a database of available keys, then the data structure returns the corresponding matched value

On the vocabulary

- ‘key’, ‘query’, ‘value’ come from a key-value data structure (search engine)

  If we give a query key and match it to a database of available keys, then the data structure returns the corresponding matched value
- Similar here
  - matching done by scalar products
  - softmax ensures a soft-matching
  - keys are matched to queries in some extent

On the vocabulary

- ‘key’, ‘query’, ‘value’ come from a key-value data structure (search engine)

  If we give a query key and match it to a database of available keys, then the data structure returns the corresponding matched value
- Similar here
  - matching done by scalar products
  - softmax ensures a soft-matching
  - keys are matched to queries in some extent

- ”Self-attention”?

On the vocabulary

- ‘key’, ‘query’, ‘value’ come from a key-value data structure (search engine)

  If we give a query key and match it to a database of available keys, then the data structure returns the corresponding matched value
- Similar here
  - matching done by scalar products
  - softmax ensures a soft-matching
  - keys are matched to queries in some extent

- ”Self-attention”? The self-attention mechanism allows the inputs
  1. to interact with each other (“self”)
  2. to find out who they should pay more attention to (“attention”)

On the vocabulary

- ‘key’, ‘query’, ‘value’ come from a key-value data structure (search engine)

  If we give a query key and match it to a database of available keys, then the data structure returns the corresponding matched value

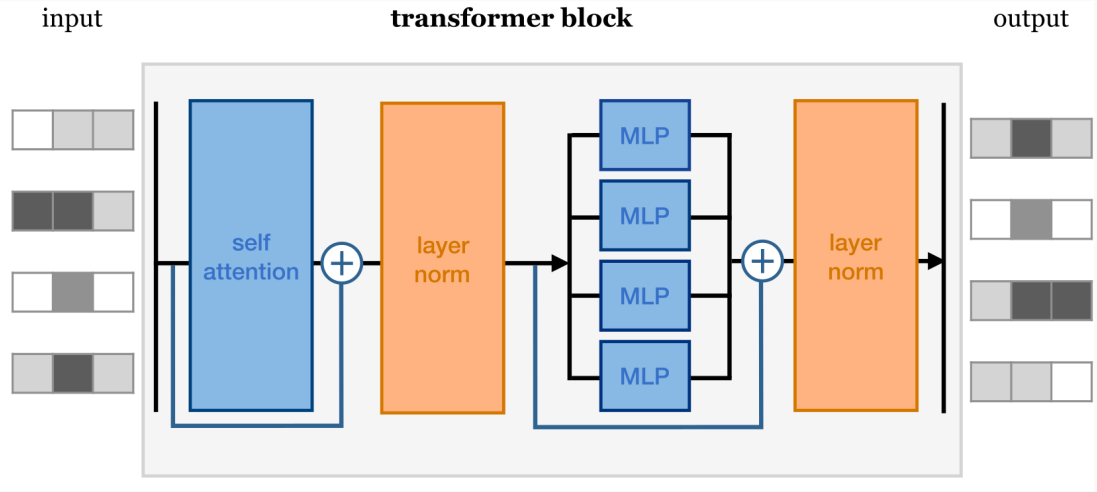
- Similar here
  - matching done by scalar products
  - softmax ensures a soft-matching
  - keys are matched to queries in some extent

- ”Self-attention”? The self-attention mechanism allows the inputs
  1. to interact with each other (“self”)
  2. to find out who they should pay more attention to (“attention”)

  The outputs are aggregates of these interactions and attention scores.

Transformer?

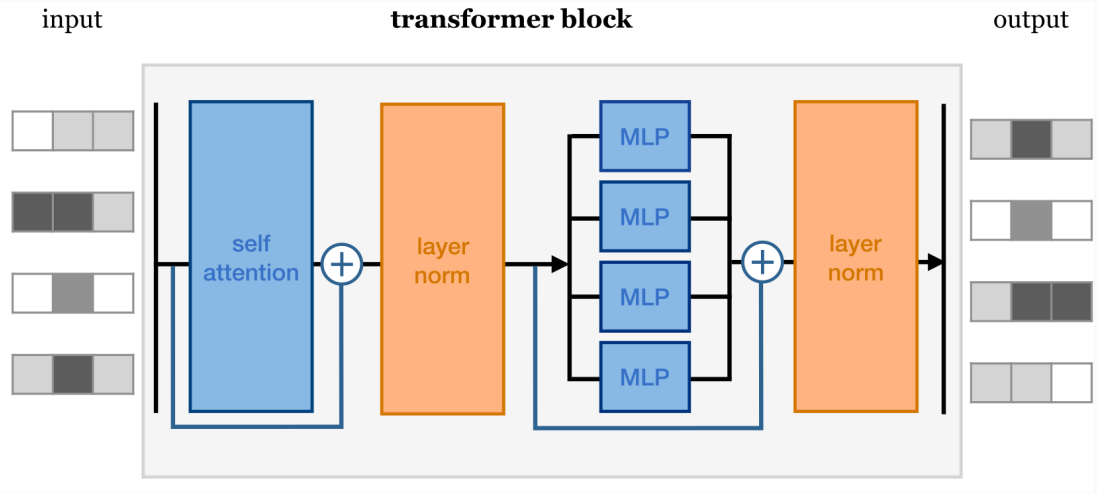
- This is an architecture



from http://peterbloem.nl/blog/transformers

Transformer?

- This is an architecture

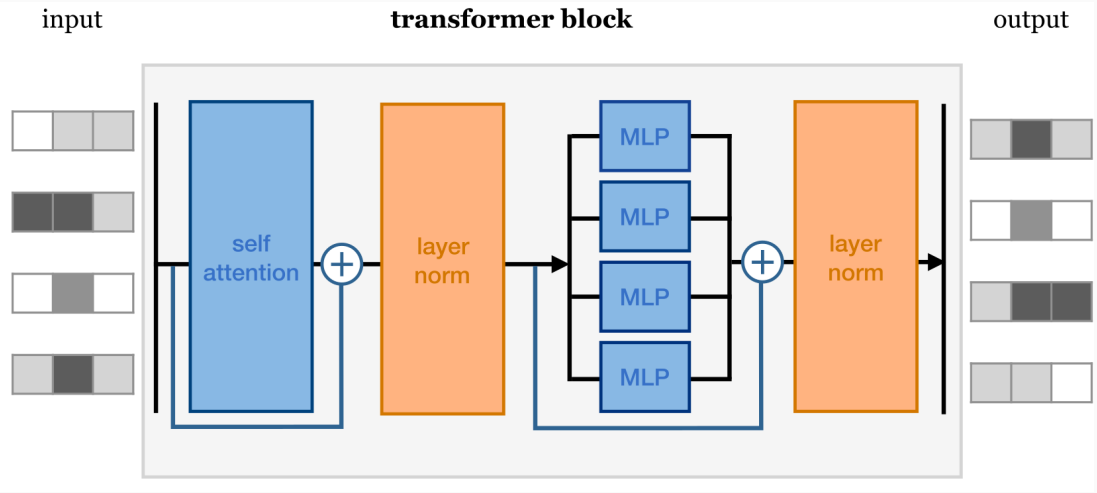


from http://peterbloem.nl/blog/transformers

- Combining self-attention, residual connections, layer normalizations and standard MLPs

Transformer?

- This is an architecture

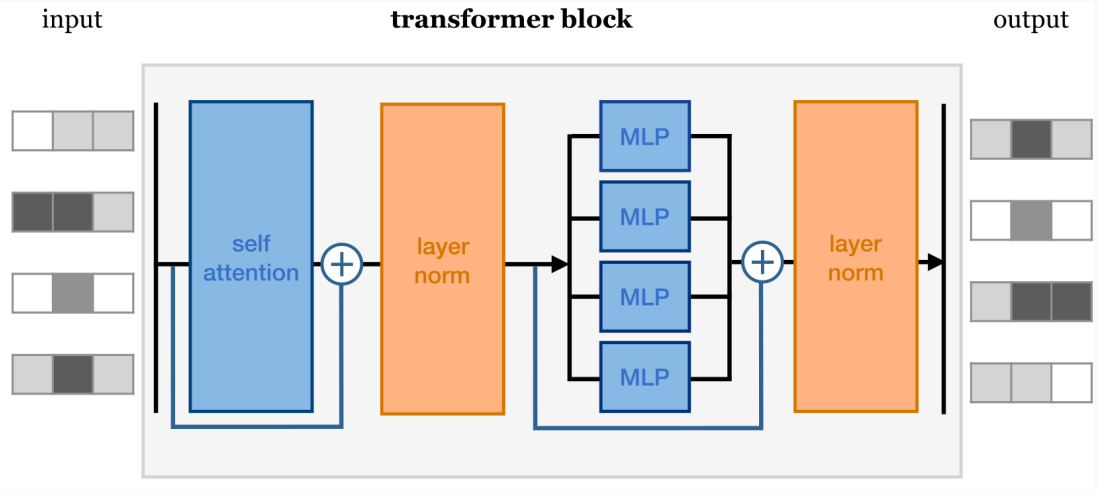


from http://peterbloem.nl/blog/transformers

- Combining self-attention, residual connections, layer normalizations and standard MLPs
- Normalization and residual connections are standard tricks used to help deep neural networks train faster and more accurately

Transformer?

- This is an architecture



from http://peterbloem.nl/blog/transformers

- Combining self-attention, residual connections, layer normalizations and standard MLPs
- Normalization and residual connections are standard tricks used to help deep neural networks train faster and more accurately
- The layer normalization is applied over the embedding dimension only

Positional encoding

- Unlike sequence models (such as RNNs or LSTMs), self-attention layers are permutation-equivariant

Positional encoding

- Unlike sequence models (such as RNNs or LSTMs), self-attention layers are permutation-equivariant
- Meaning that

$$\begin{cases} \texttt{`The dog chases the cat'} \\ \texttt{`The cat chases the dog'} \end{cases}$$

will learn the same features

Positional encoding

- Unlike sequence models (such as RNNs or LSTMs), self-attention layers are permutation-equivariant
- Meaning that

$$\begin{cases} \texttt{`The dog chases the cat'} \\ \texttt{`The cat chases the dog'} \end{cases}$$

  will learn the same features
- Solution: positional embedding/encoding
  - One-hot encoding
  - Sinusoidal encoding

$$\text{position } t \to (\sin(\omega_1 t), \sin(\omega_2 t), \ldots, \sin(\omega_d t))$$

with $\omega_k = \frac{1}{10000^{k/d}}$ (float continuous counterparts of binary values)

Positional encoding

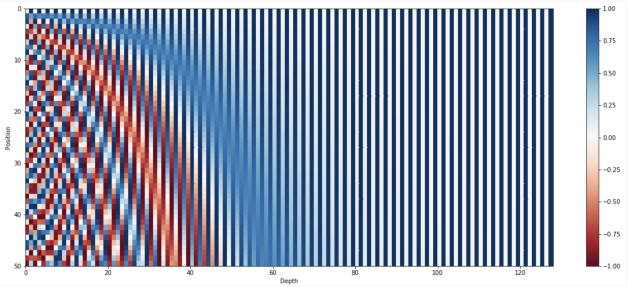
- Unlike sequence models (such as RNNs or LSTMs), self-attention layers are permutation-equivariant
- Meaning that

$$\begin{cases} \texttt{`The dog chases the cat'} \\ \texttt{`The cat chases the dog'} \end{cases}$$

  will learn the same features
- Solution: positional embedding/encoding
  - One-hot encoding
  - Sinusoidal encoding

$$\text{position } t \to (\sin(\omega_1 t), \sin(\omega_2 t), \ldots, \sin(\omega_d t))$$

with $\omega_k = \frac{1}{10000^{k/d}}$ (float continuous counterparts of binary values)



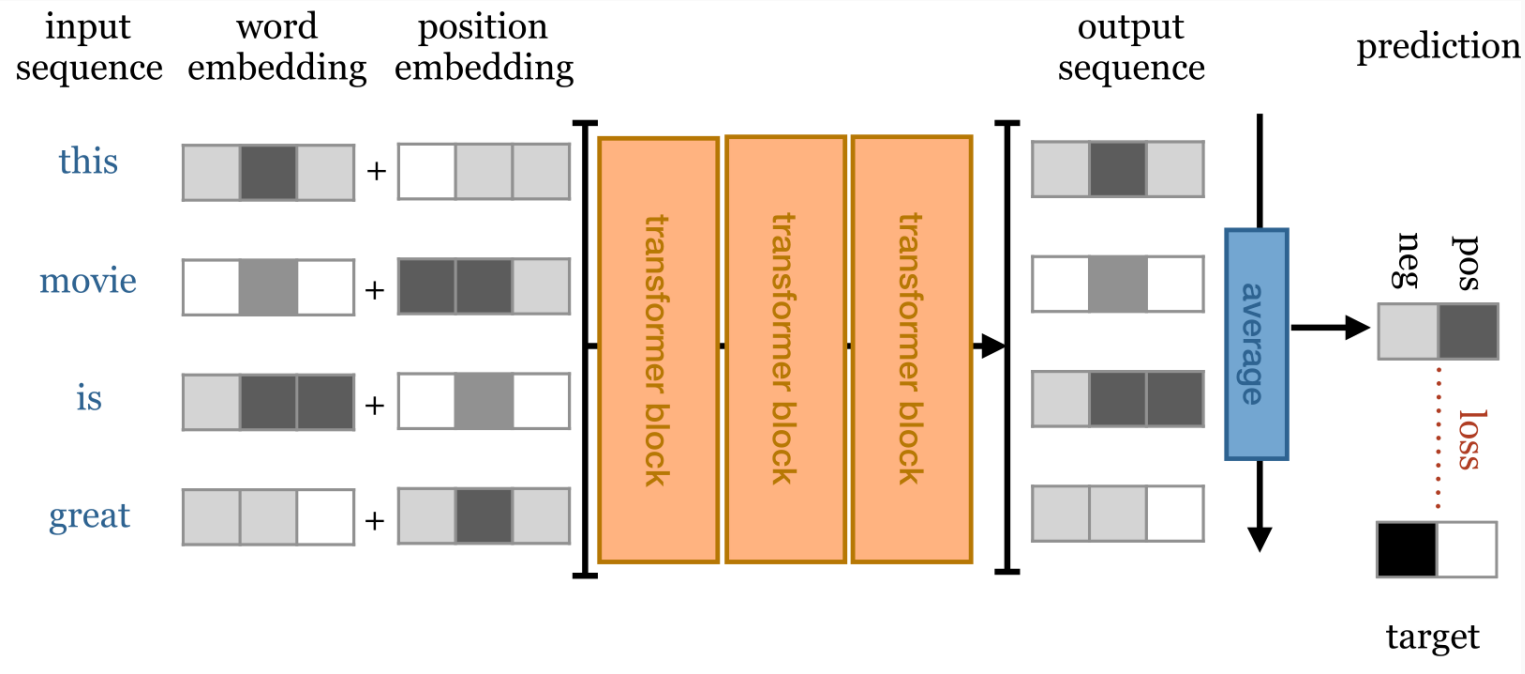
The 128-dimensional positional encoding for a sentence with a maximum length of 50. Each row represents the encoding vector.

Simple sequence classification transformer

- Goal: build a sequence classifier for sentiment analysis
- IMDb sentiment classification dataset
  - (input) movie reviews (sequences of words)
  - (output) classification labels: positive or negative

Simple sequence classification transformer

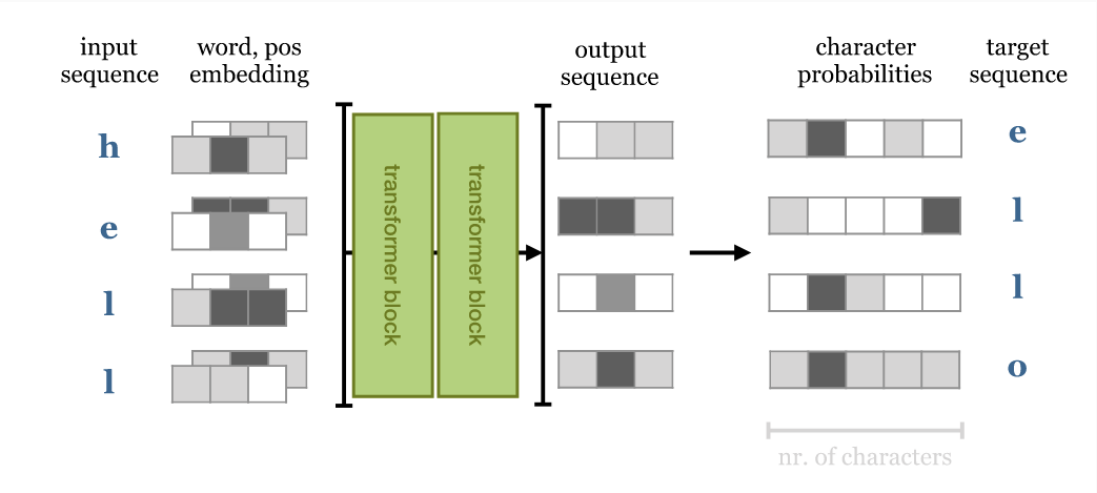
- Goal: build a sequence classifier for sentiment analysis
- IMDb sentiment classification dataset
  - (input) movie reviews (sequences of words)
  - (output) classification labels: positive or negative



from http://peterbloem.nl/blog/transformers

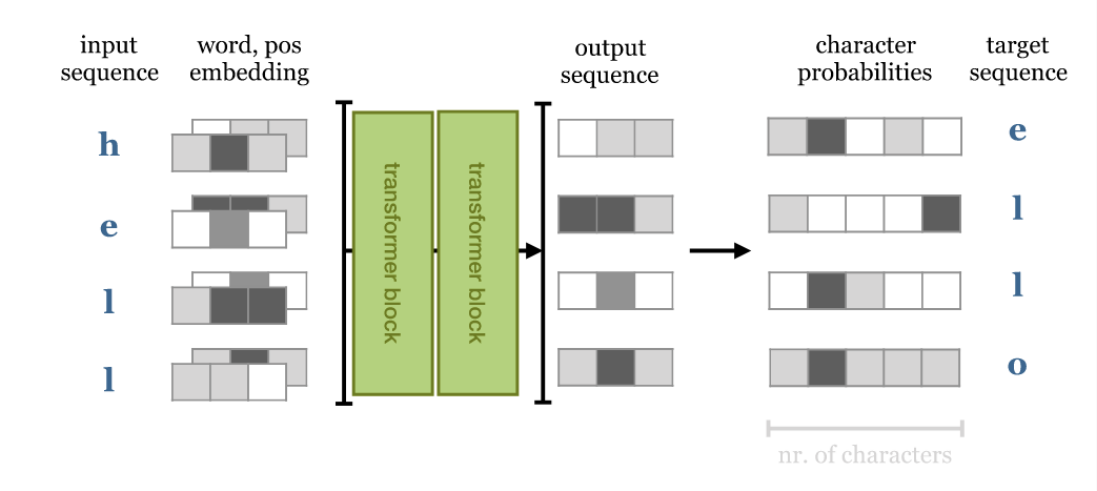
Text generation transformer

- Goal: predict the next character in a sequence



Text generation transformer

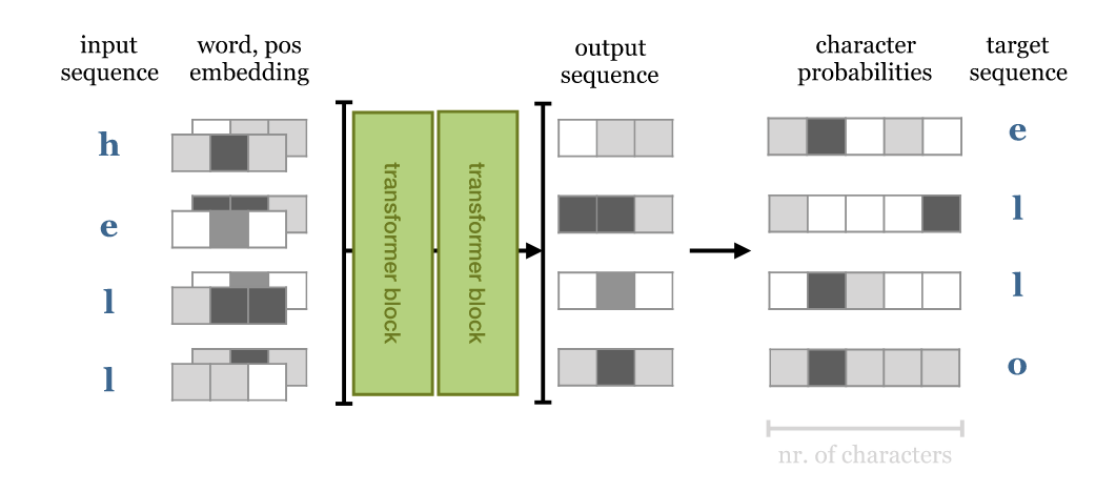
- Goal: predict the next character in a sequence



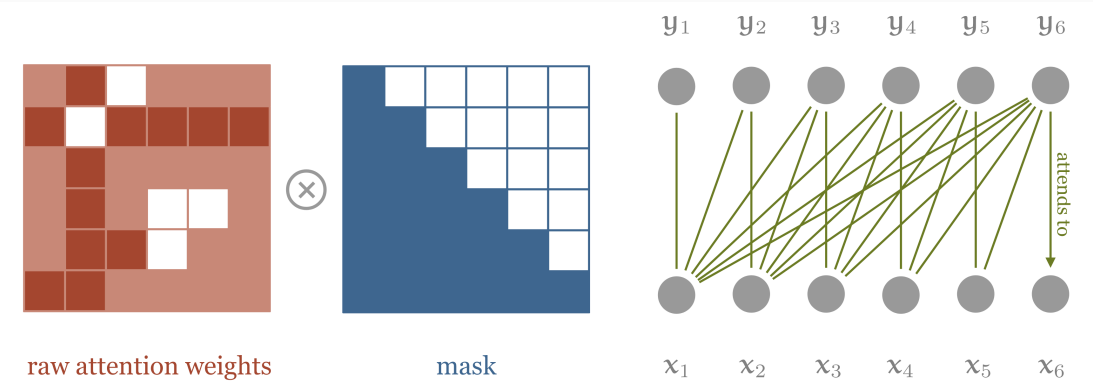
- With a transformer, the output depends on the entire input sequence: vacuously easy task!

Text generation transformer

- Goal: predict the next character in a sequence

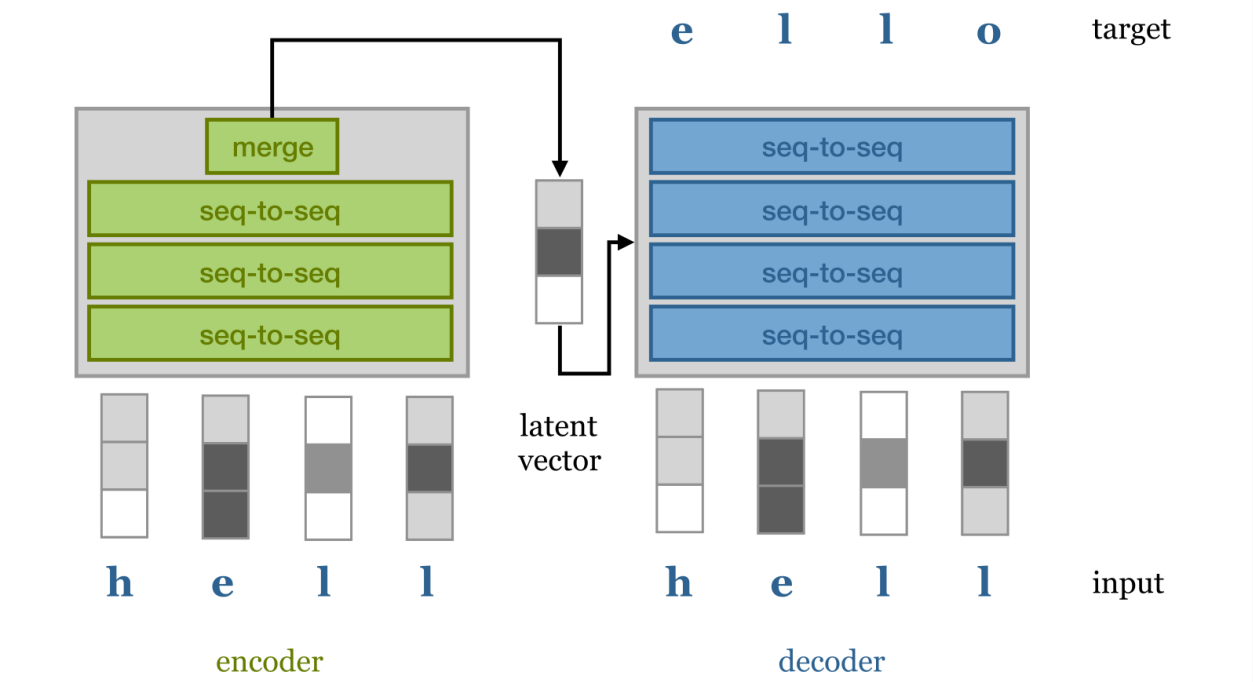


- With a transformer, the output depends on the entire input sequence: vacuously easy task!
- Solution: apply a mask to ensure that it cannot look forward into the sequence



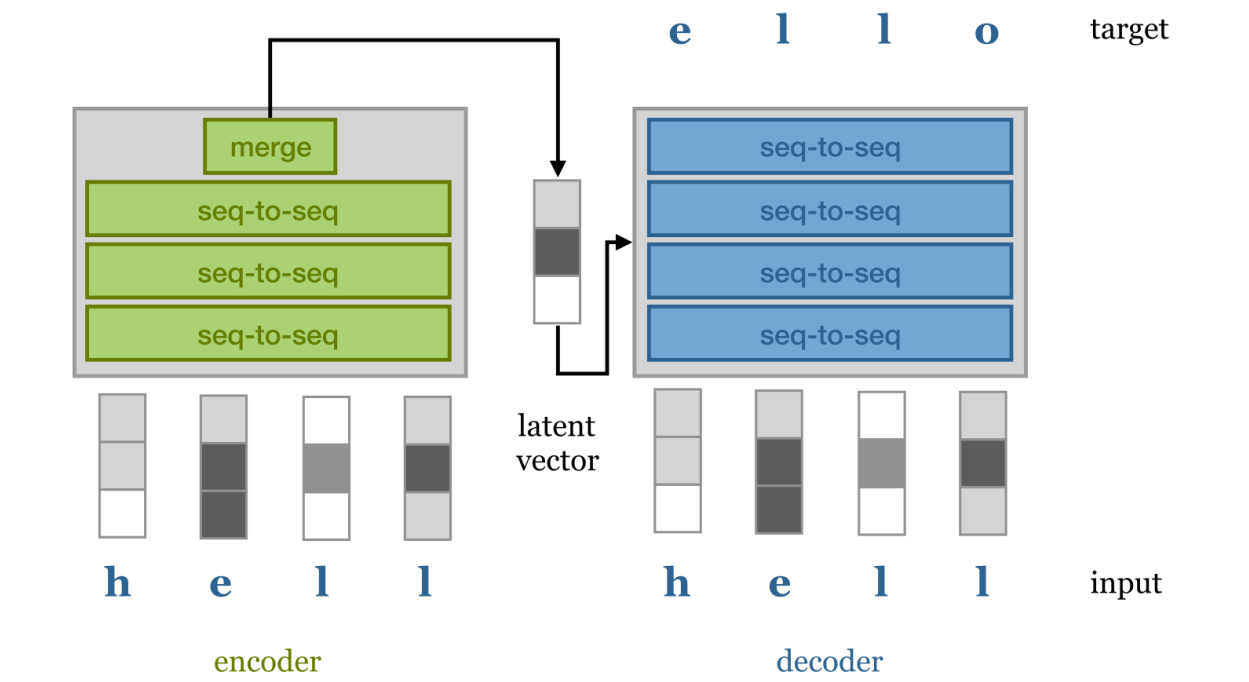
The original transformer

- Vaswani et al. (2017)



The original transformer

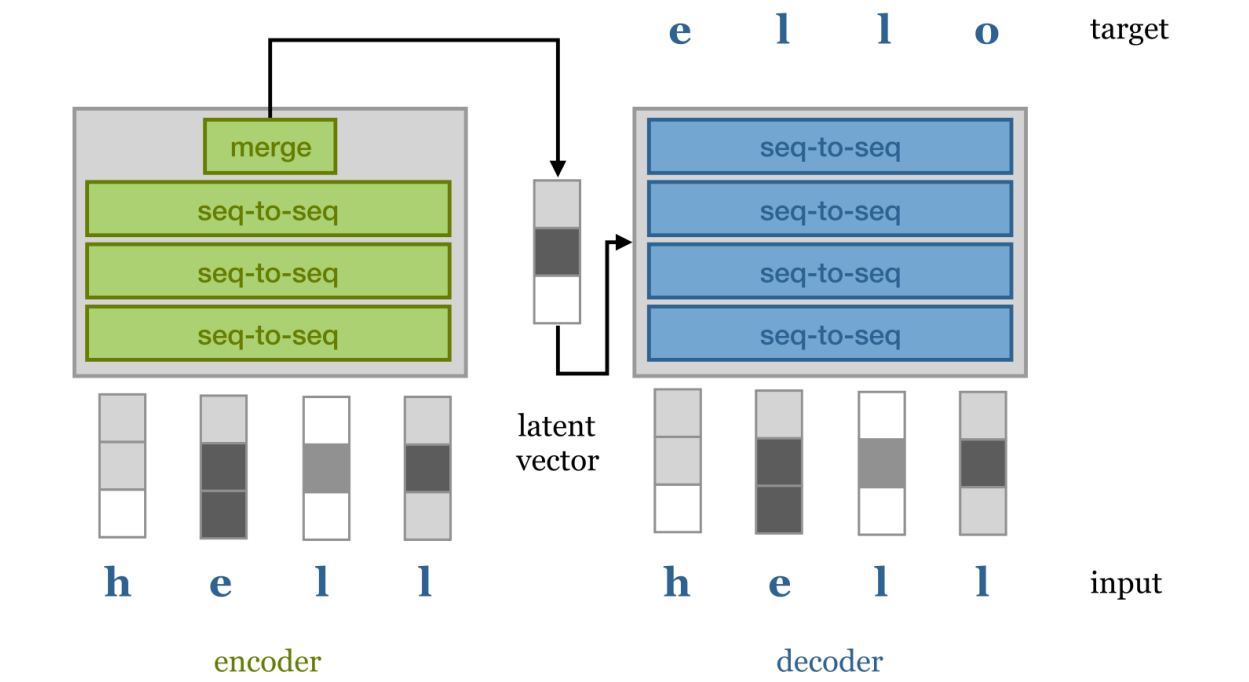
- Vaswani et al. (2017)



- A sequence-to-sequence structure by encoder-decoder architecture with teacher forcing

The original transformer

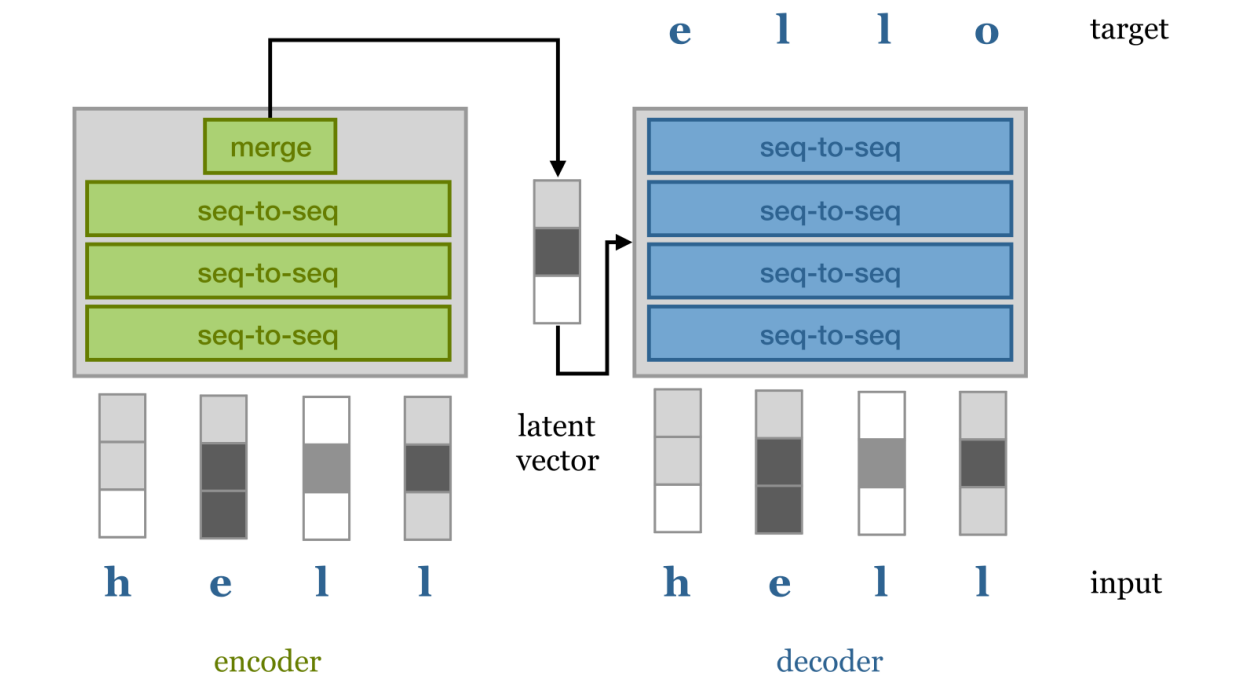
- Vaswani et al. (2017)



- A sequence-to-sequence structure by encoder-decoder architecture with teacher forcing
  - encoder: takes the input sequence and maps it to a latent representation

The original transformer

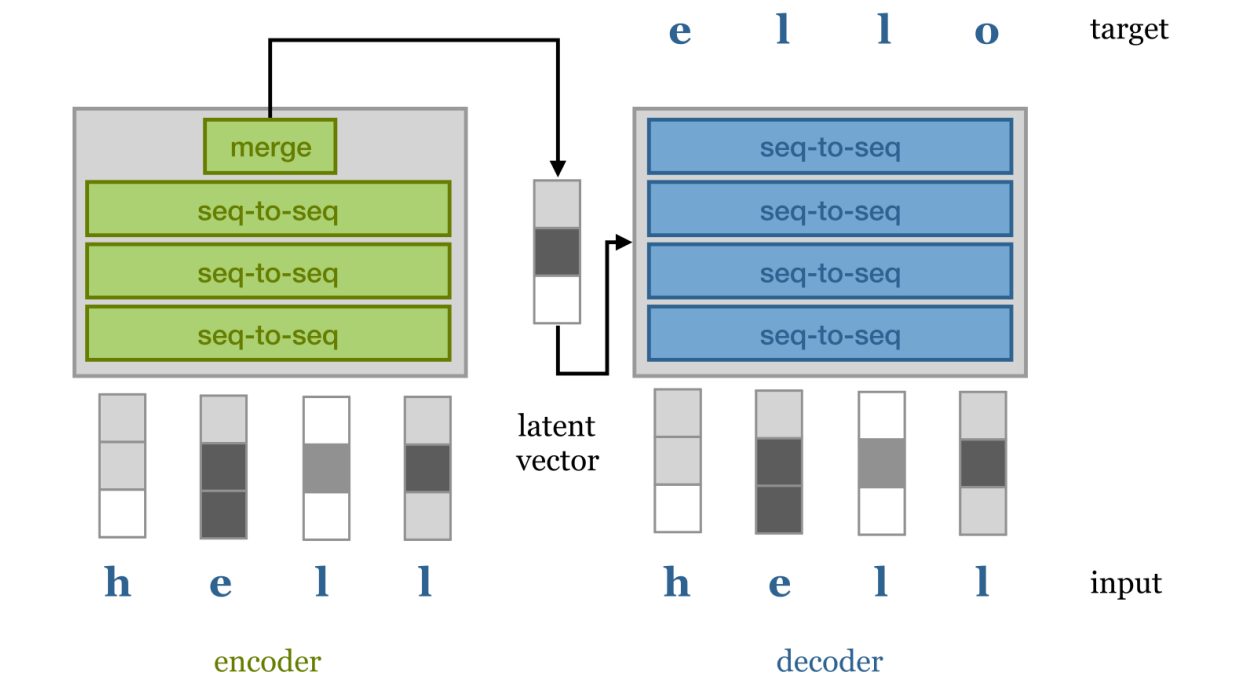
- Vaswani et al. (2017)



- A sequence-to-sequence structure by encoder-decoder architecture with teacher forcing
  - encoder: takes the input sequence and maps it to a latent representation
  - decoder: unpacks it to the desired target sequence (for instance, language translation)

The original transformer

- Vaswani et al. (2017)



- A sequence-to-sequence structure by encoder-decoder architecture with teacher forcing
  - encoder: takes the input sequence and maps it to a latent representation
  - decoder: unpacks it to the desired target sequence (for instance, language translation)
  - teacher forcing: the decoder also has access to the input sequence

Focus on the decoder

- The decoder also has access to the input sequence in an autoregressive manner: access to the words it has already generated

Focus on the decoder

- The decoder also has access to the input sequence in an autoregressive manner: access to the words it has already generated
- The decoder can use
  - word-for-word sampling to take care of the low-level structure like syntax and grammar
  - the latent vector to capture more high-level semantic structure

Modern transformers

- BERT (Bidirectional Encoder Representations from Transformers): reaches human-level performance on a variety of language based tasks: question answering, sentiment classification or classifying whether two sentences naturally follow one another

Modern transformers

- BERT (Bidirectional Encoder Representations from Transformers): reaches human-level performance on a variety of language based tasks: question answering, sentiment classification or classifying whether two sentences naturally follow one another
  - simple stack of transformer blocks

Modern transformers

- BERT (Bidirectional Encoder Representations from Transformers): reaches human-level performance on a variety of language based tasks: question answering, sentiment classification or classifying whether two sentences naturally follow one another
  - simple stack of transformer blocks
  - pre-trained on a large general-domain corpus (English books and wikipedia)

Modern transformers

- BERT (Bidirectional Encoder Representations from Transformers): reaches human-level performance on a variety of language based tasks: question answering, sentiment classification or classifying whether two sentences naturally follow one another
  - simple stack of transformer blocks
  - pre-trained on a large general-domain corpus (English books and wikipedia)
  - pre-training possible through masking or next-sequence classification

Modern transformers

- BERT (Bidirectional Encoder Representations from Transformers): reaches human-level performance on a variety of language based tasks: question answering, sentiment classification or classifying whether two sentences naturally follow one another
  - simple stack of transformer blocks
  - pre-trained on a large general-domain corpus (English books and wikipedia)
  - pre-training possible through masking or next-sequence classification
- GPT-2: prediction of the next word

Modern transformers

- BERT (Bidirectional Encoder Representations from Transformers): reaches human-level performance on a variety of language based tasks: question answering, sentiment classification or classifying whether two sentences naturally follow one another
  - simple stack of transformer blocks
  - pre-trained on a large general-domain corpus (English books and wikipedia)
  - pre-training possible through masking or next-sequence classification
- GPT-2: prediction of the next word
- Transformer-XL: for long sequence of text

Modern transformers

- BERT (Bidirectional Encoder Representations from Transformers): reaches human-level performance on a variety of language based tasks: question answering, sentiment classification or classifying whether two sentences naturally follow one another
  - simple stack of transformer blocks
  - pre-trained on a large general-domain corpus (English books and wikipedia)
  - pre-training possible through masking or next-sequence classification
- GPT-2: prediction of the next word
- Transformer-XL: for long sequence of text
- Sparse transformers: uses sparse attention matrices

Wrapping up

- 2 NN network architecture paradigms: recurrent networks and attention

Wrapping up

- 2 NN network architecture paradigms: recurrent networks and attention
- Ideas behind attention surprisingly simple!

Wrapping up

- 2 NN network architecture paradigms: recurrent networks and attention
- Ideas behind attention surprisingly simple!
- Back-propagation in all of them: this is the learning phase

References i

 Mikolov, Tomas et al. (2013). “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems* 26.

 Vaswani, Ashish et al. (2017). “Attention is all you need”. In: *Advances in neural information processing systems* 30.