Large Scale Machine Learning

Natural Language Processing

Antoine Recanati
antoine.recanati@sancare.fr
March, 5th 2025

Mines ParisTech - PSL

Acknowledgement

Slides adapted from those by Adeline Fermanian, herself inspired by

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

I also stole excalidraw drawings on word embeddings, from Romain Brand (Sancare), with his consent.

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
  - conversational agent (chatbot), virtual assistant

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
  - conversational agent (chatbot), virtual assistant
  - text summarization (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
  - conversational agent (chatbot), virtual assistant
  - text summarization (medical records, scientific articles)
  - generation of slides for a NLP course at Mines ParisTech.

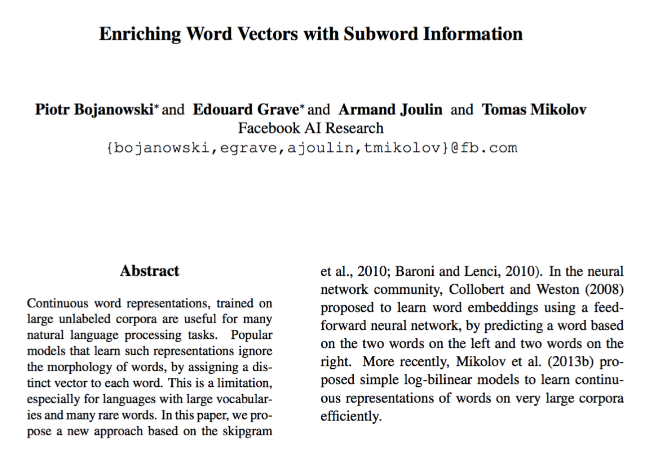
Motivation

- 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
  - conversational agent (chatbot), virtual assistant
  - text summarization (medical records, scientific articles)
  - generation of slides for a NLP course at Mines ParisTech.
- 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

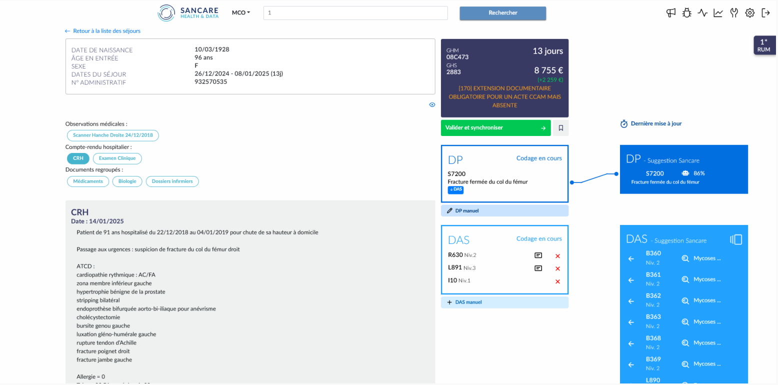
Piotr Bojanowski* and **Edouard Grave*** and **Armand Joulin** and **Tomas Mikolov**
Facebook AI Research
{bojanowski,egrave,ajoulin,tmikolov}@fb.com

Abstract

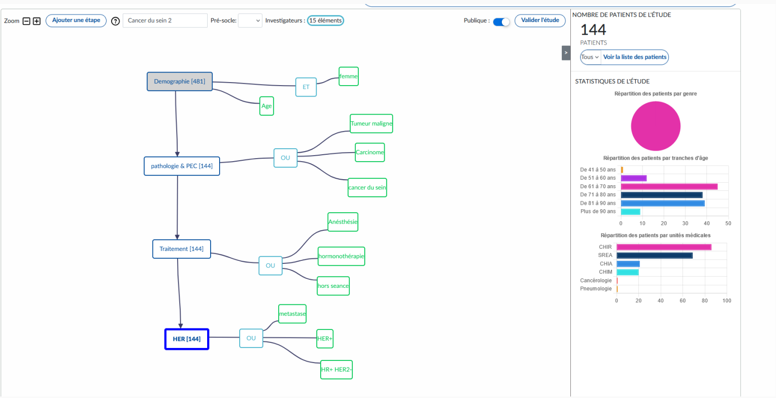
Continuous word representations, trained on large unlabeled corpora are useful for many natural language processing tasks. Popular models that learn such representations ignore the morphology of words, by assigning a distinct vector to each word. This is a limitation, especially for languages with large 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.

Text classification: find main diagnosis of patient record ? (advertisement!)



Information extraction: find patients patients with HR+/HER2- BC



Large Language Models: generate text (e.g., ChatGPT)

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

Representing Numeric Data

Numeric data is straightforward to represent in vector form. Each numeric feature is directly used as a dimension in the vector.

Example: Consider a dataset of houses with features: *Size (sqft)*, *Number of Bedrooms*, and *Age (years)*.

Size (sqft)	Bedrooms	Age (years)
2000	3	20
1500	2	30
2500	4	10

This can be represented as vectors:

$$\begin{bmatrix} 2000 & 3 & 20 \\ 1500 & 2 & 30 \\ 2500 & 4 & 10 \end{bmatrix}$$

Each row is a vector representing a house in our dataset.

Representing Categorical Data: The Challenge

Categorical data needs to be represented numerically for machine learning models.

Example: Consider a dataset with a categorical feature *Color* having three possible values: *Red*, *Green*, *Blue*.

How to represent these categories as numbers?

Color
Red
Green
Blue
Red

Representing Categorical Data: The Challenge

Categorical data needs to be represented numerically for machine learning models.

Example: Consider a dataset with a categorical feature *Color* having three possible values: *Red*, *Green*, *Blue*.

Color
Red
Green
Blue
Red

One simple approach: assign numbers to categories

Category	**Numeric Value**
Red	1
Green	2
Blue	3

Problem: This implies an ordering (Blue ¿ Green ¿ Red) and distance between categories that doesn't exist!

Representing Categorical Data with One-Hot Encoding

Categorical data can be represented using one-hot encoding, where each category is transformed into a binary vector.

Example: Consider a dataset with a categorical feature *Color* having three possible values: *Red*, *Green*, *Blue*.

One-hot encoding for *Color*:

Color
Red
Green
Blue
Red

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Columns represent *Red*, *Green*, and *Blue* respectively. Each row is a binary vector representing the color of an item.

This encoding method allows us to convert categorical data into a numeric form that can be used by machine learning models.

Representing Text - Tokenization

- How to represent words as input?

"I've never seen a movie like this before"

Representing Text - 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"]

Representing Text - 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:

Representing Text - 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* → [I've] or [I] [have] or [I] ['ve] ?

Representing Text - 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* → [I've] or [I] [have] or [I] ['ve] ?
  - *low-frequency* → [low-frequency] or [low] [frequency] ?

Representing Text - 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* → [I've] or [I] [have] or [I] ['ve] ?
  - *low-frequency* → [low-frequency] or [low] [frequency] ?
  - Some arbitrary choices: be consistent!

Representing Text - 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!

Representing Text - 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”]

Representing Text - 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]

Representing Text - 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.

Representing Text - 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]

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

One-hot encoding

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

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

$$\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}$$

- 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]

$$\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}$$

- 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]

$$\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}$$

- 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

$$
\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ \dots \\ 1 \\ 0 \\ 1 \\ 0
\end{pmatrix}
$$

- 1 if word is present in the sentence

One-hot vector

$$
\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ \vdots \\ 1 \\ 0 \\ 1 \\ 0
\end{pmatrix}
$$

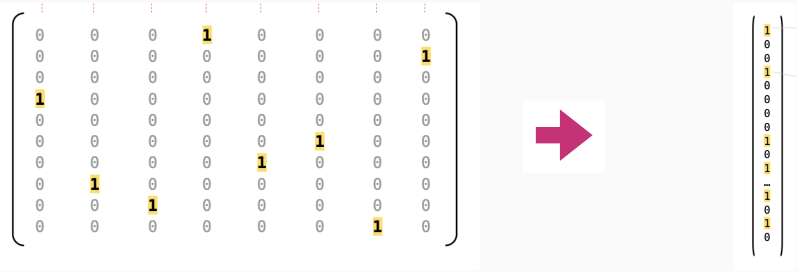
- 1 if word is present in the sentence
- Size does not depend on length of the sentence

One-hot vector

$$
\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ \dots \\ 1 \\ 0 \\ 1 \\ 0
\end{pmatrix}
$$

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

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
- (Note: Bag of Words representation sufficient to classify newspaper articles per topic accurately)

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

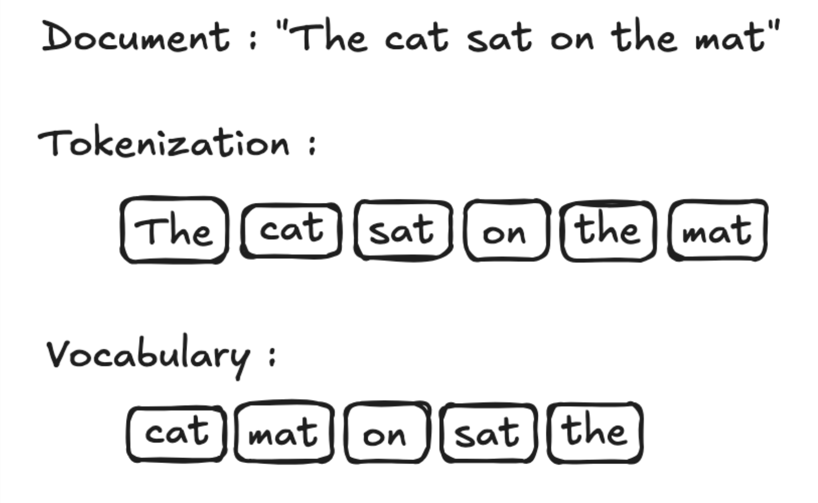
Embedding layer

- 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

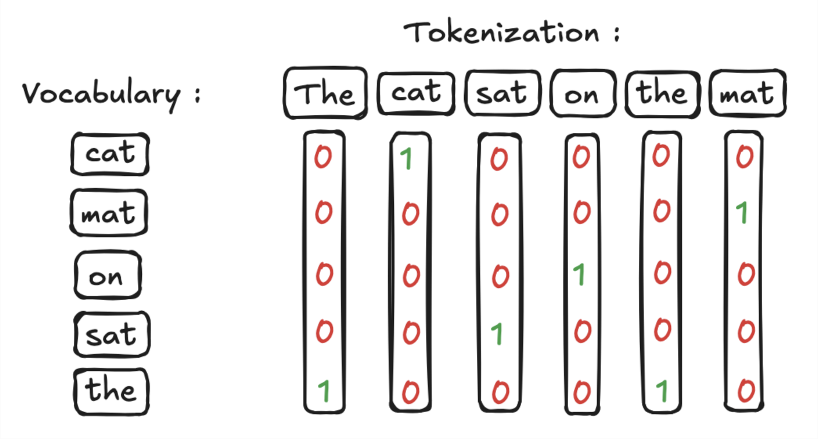
- Text of 300 words
- Embedding size: 200
- Memory: 60 000 parameters to store the text!

Word embedding: illustrative example



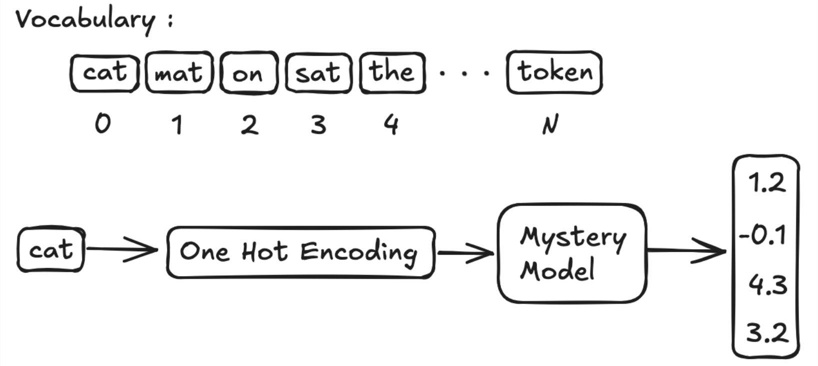
Word embedding: illustrative example

One hot encoding

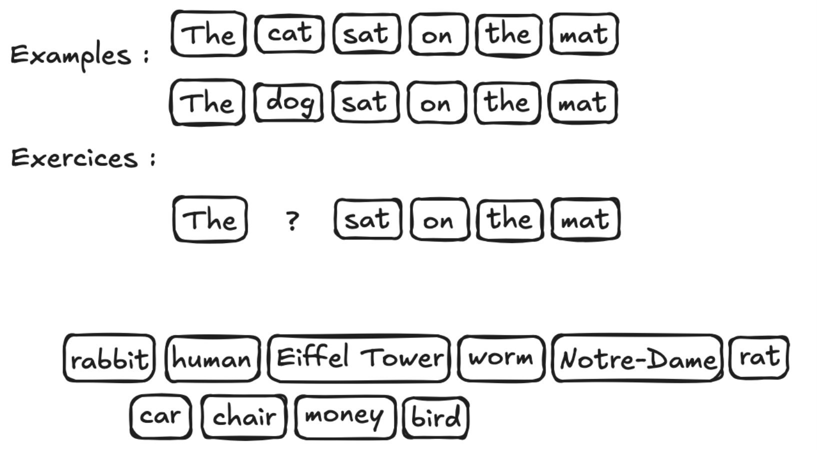


Word embedding: illustrative example

Goal: obtain a (small) dense vector for each word

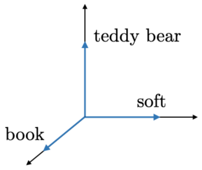
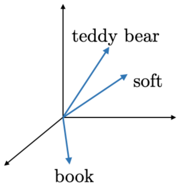


Goal: obtain vectors with semantic meaning. How ?



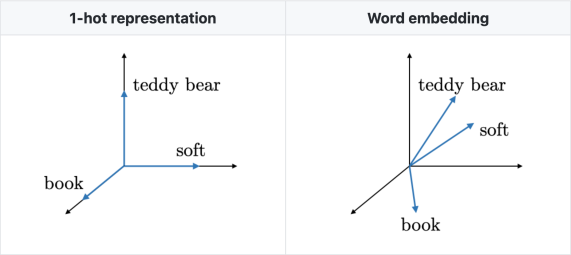
Word embedding

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

1-hot representation	Word embedding
	

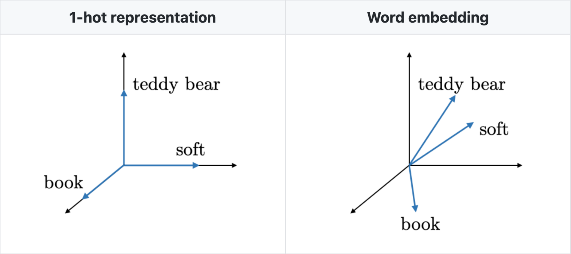
Word embedding

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



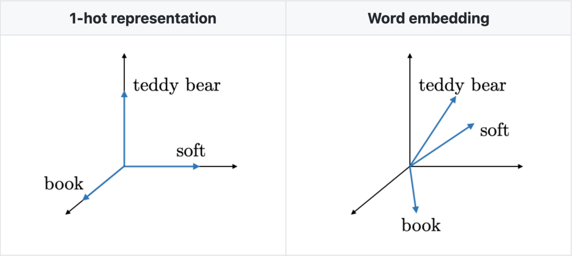
Word embedding

- 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:



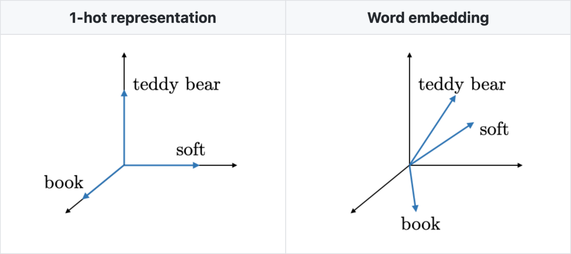
Word embedding

- 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)

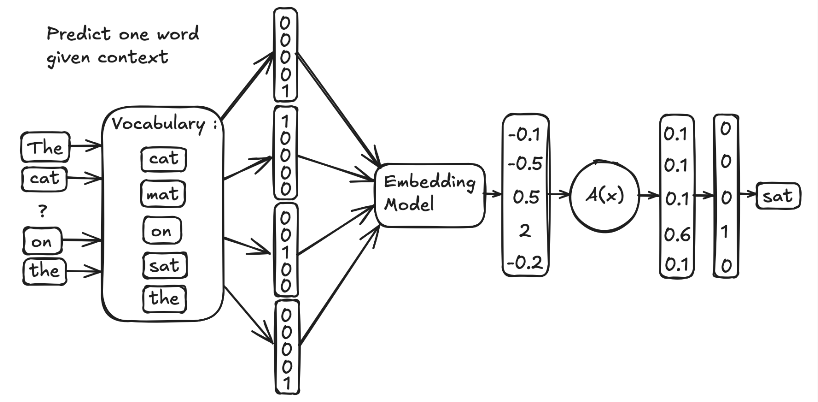


Word embedding

- 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)

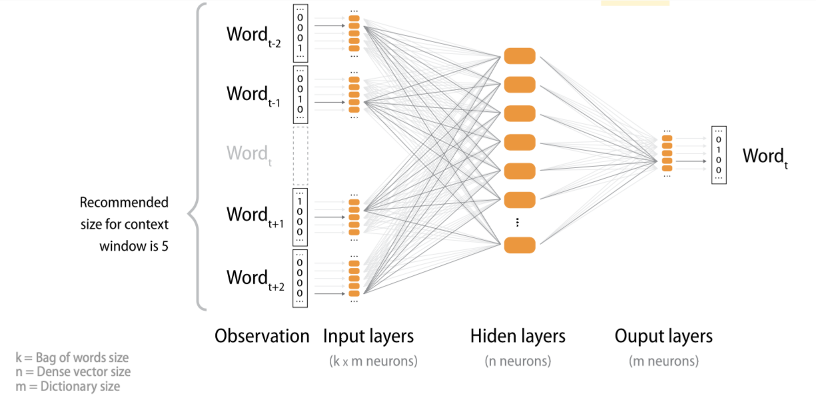


CBOW (with illustrative example)



- Objective: find a word from its context

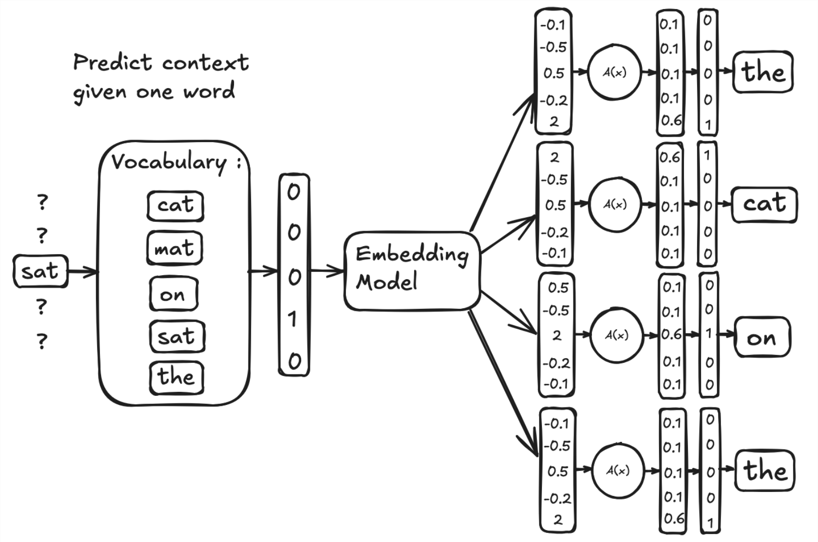
CBOW (architecture)



source: fidle-cnrs

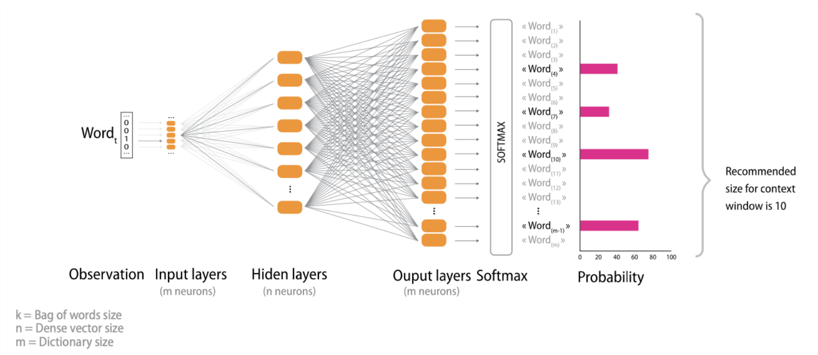
- Objective: find a word from its context

Skip-Gram (with illustrative example)



- Objective: find the context from a word

Skip-Gram (architecture)

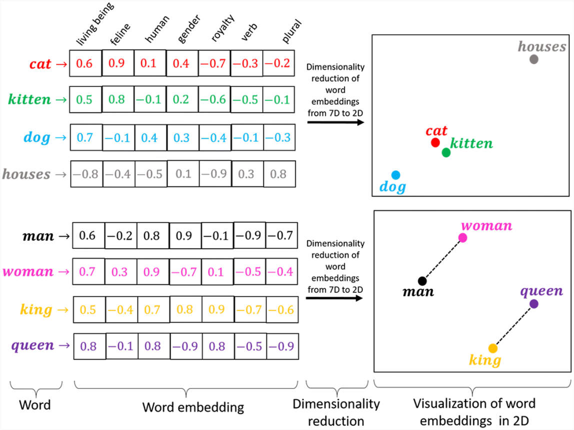


source: fidle-cnrs

- Objective: find the context from a word

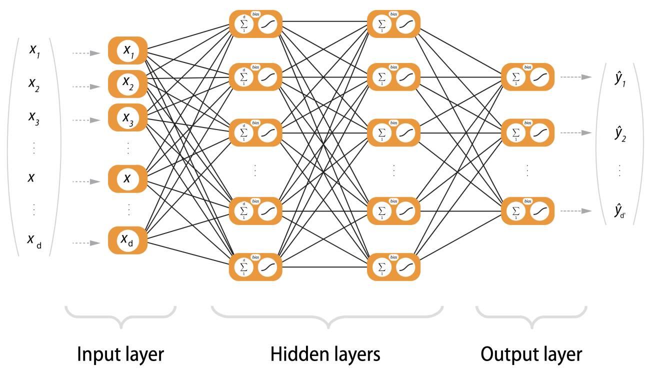
Resulting trained (unsupervised) word embeddings

from https://medium.com/@hari4om/



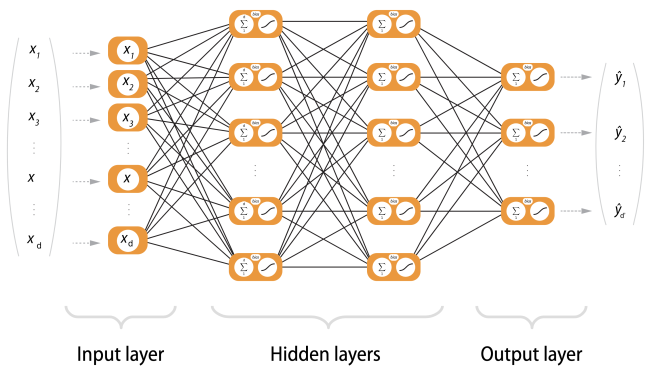
Sommaire

Recall: neural networks



source: fidle-cnrs

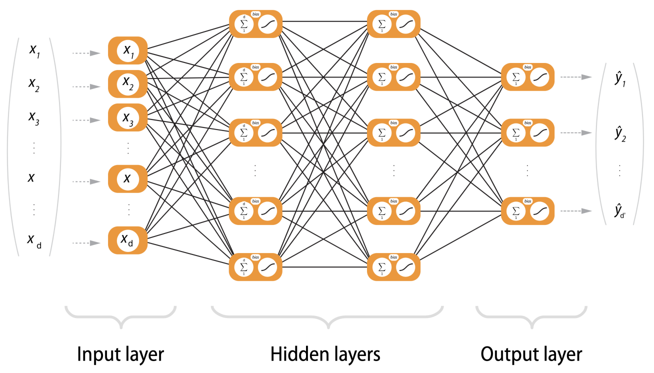
Recall: neural networks



source: fidle-cnrs

- Cascade of linear and nonlinear functions

Recall: neural networks



source: fidle-cnrs

- Cascade of linear and nonlinear functions
- Formally

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

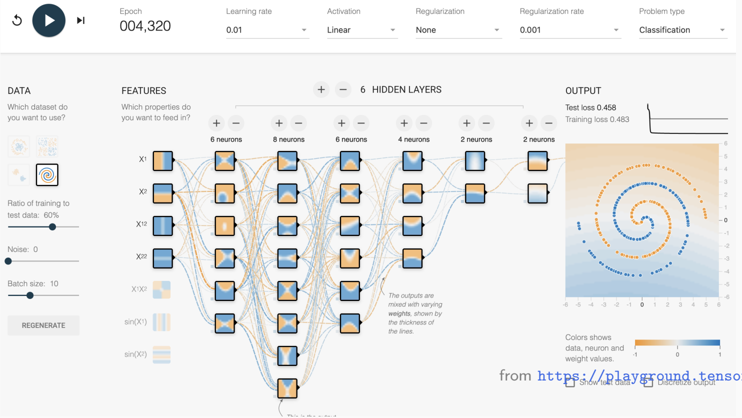
Recall: neural networks - Questions to check you're awake

- Why add non-linearities in the neural network? What happens without them?

Recall: neural networks - Questions to check you're awake

- Why add non-linearities in the neural network? What happens without them?

→ Without non-linearities, no matter how many layers, we have a linear model (one layer equiv.)



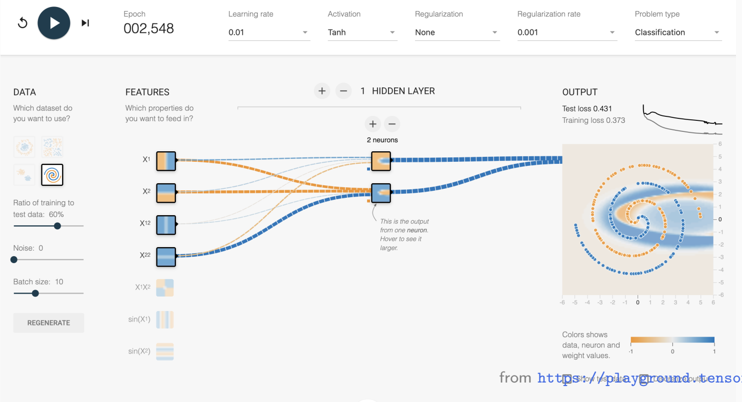
from https://playground.tensorflow.org

Recall: neural networks - Questions to check you're awake

- Why add layers?

- Why add layers?

→ Increases the capacity of the model (enables AND/OR/XOR like operations, etc.)

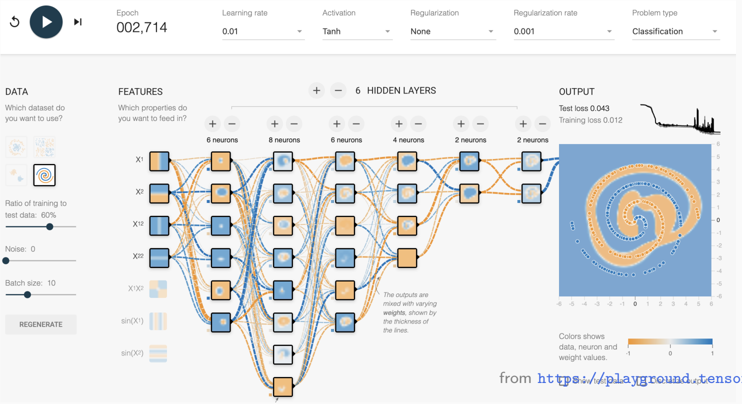


from https://playground.tensorflow.org

Recall: neural networks - Questions to check you're awake

- Why add layers?

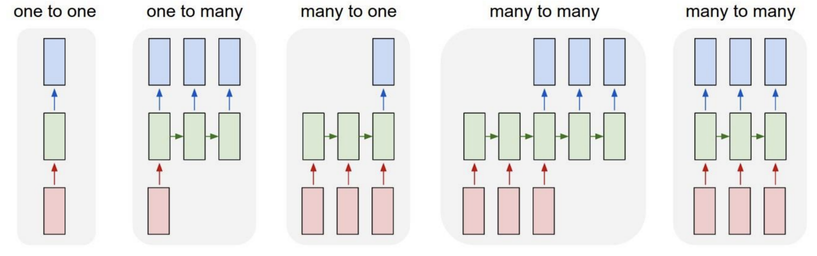
→ Increases the capacity of the model (enables AND/OR/XOR like operations, etc.)



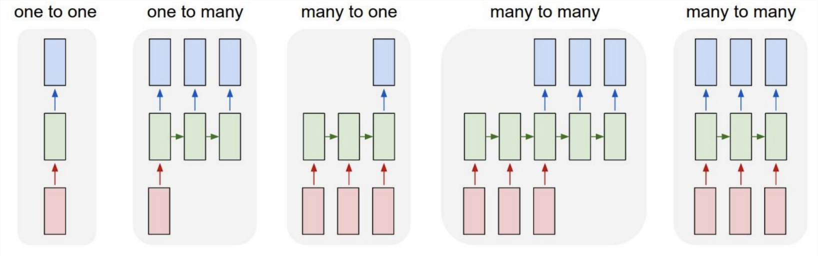
from https://playground.tensorflow.org

Recurrent neural networks

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

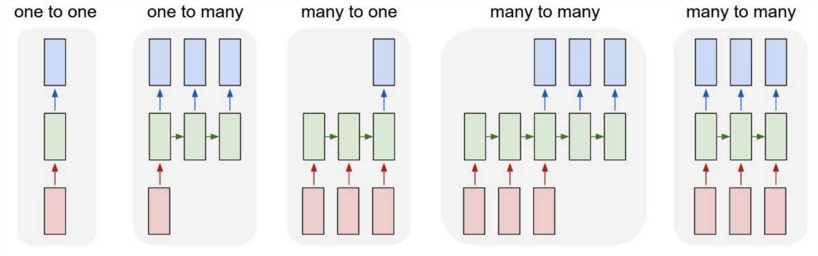


Different uses of recurrent neural networks



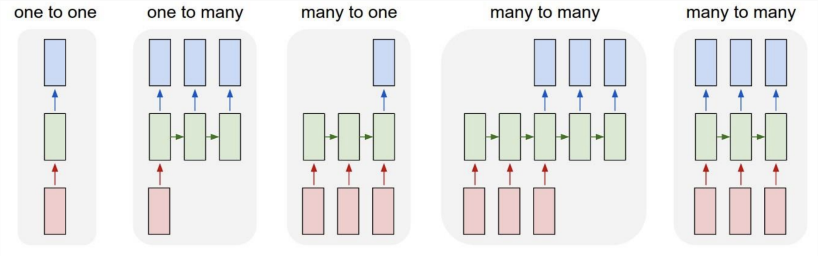
- Image classification (one-to-one)

Different uses of recurrent neural networks



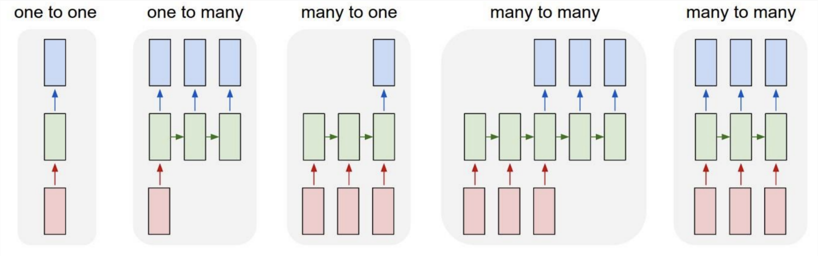
- Image classification (one-to-one)
- Image Captioning (one-to-many): image/sequence of words

Different uses of recurrent neural networks



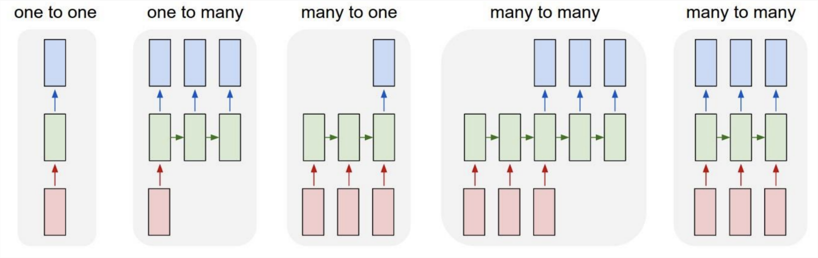
- Image classification (one-to-one)
- Image Captioning (one-to-many): image/sequence of words
- Sentiment classification (many-to-one): sequence of words/sentiment

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: $\mathbb{P}\left(\text{next word} \middle| \text{current word and past}\right)$

Language generating NN: training

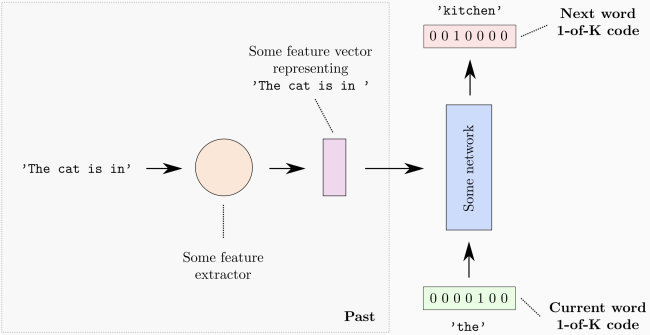
How to learn "`The cat is in the kitchen drinking milk.`"?

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

Language generating NN: training

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

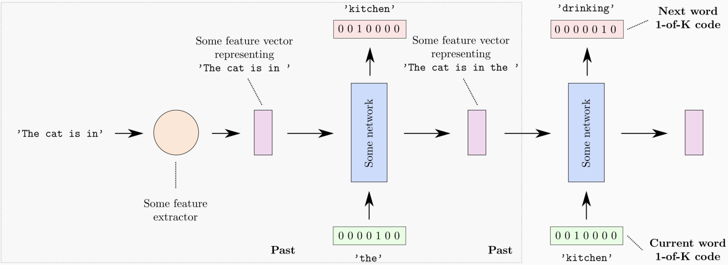
- Learn: $\mathbb{P}\,(\text{next word}|\text{current word and past})$
- Represent the past as a feature vector



Language generating NN: training

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

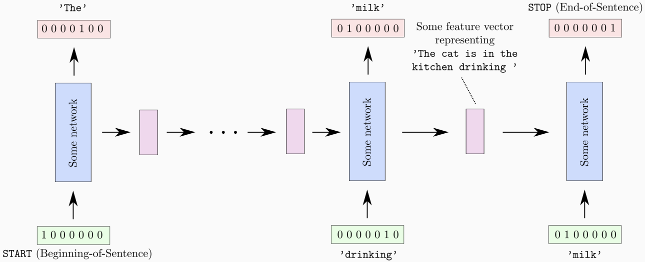
- Learn: $\mathbb{P}\left(\text{next word} \middle| \text{current word and past}\right)$
- Represent the past as a feature vector



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

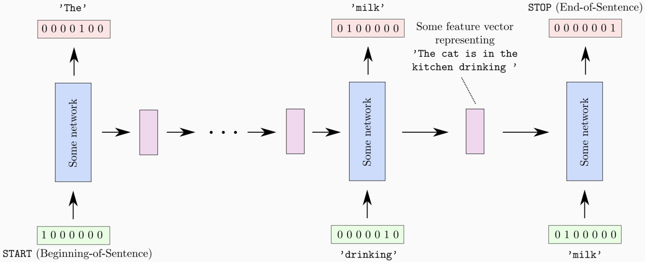
Language generating NN: training

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



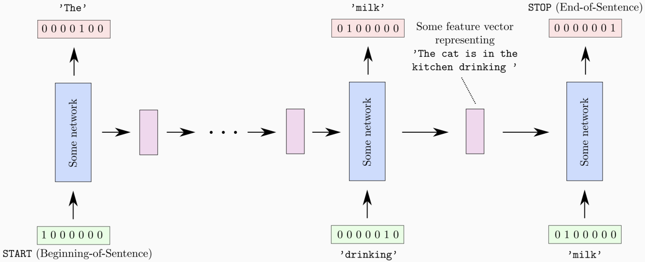
Language generating NN: training

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



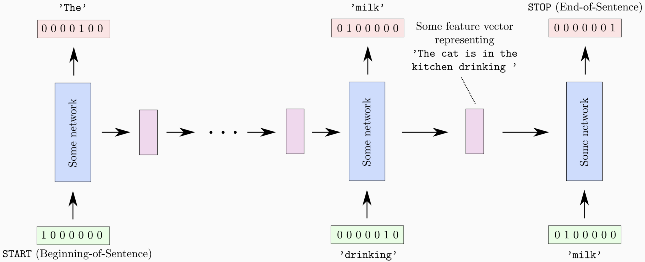
Language generating NN: training

- 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



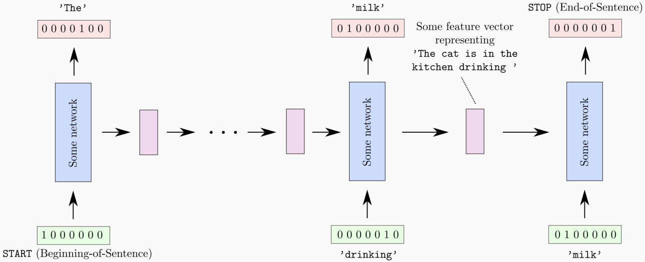
Language generating NN: training

- 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

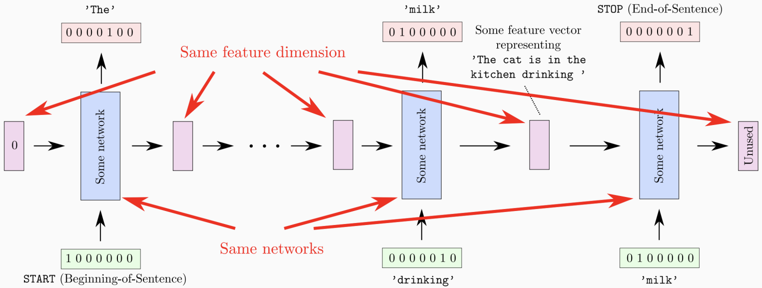


▷ *How should the network architecture and size of intermediate features evolve with the location in the sequence?*

Language generating NN: training

from Charles Deledalle's lectures

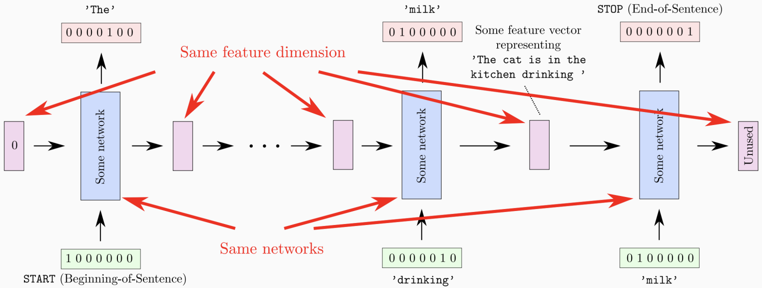
- Use the same networks and the same feature dimension



Language generating NN: training

from Charles Deledalle's lectures

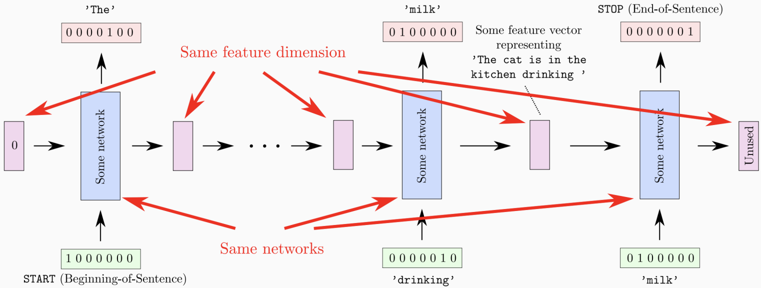
- Use the same networks and the same feature dimension
- The past is always embedded in a fix-sized feature



Language generating NN: training

from Charles Deledalle's lectures

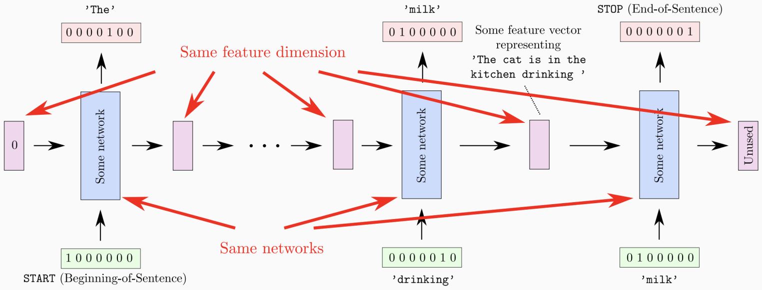
- Use the same networks and the same feature dimension
- The past is always embedded in a fix-sized feature
- Set the first feature as a zero tensor



Language generating NN: training

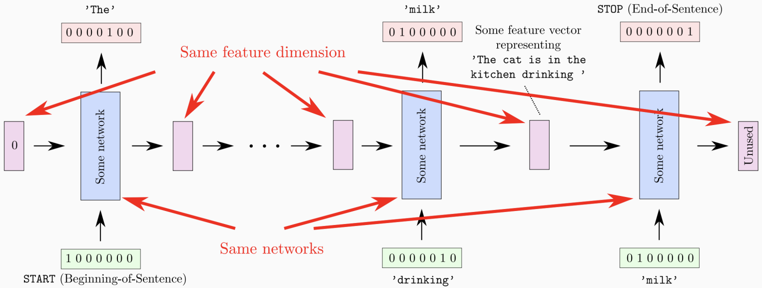
from Charles Deledalle's lectures

- 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 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

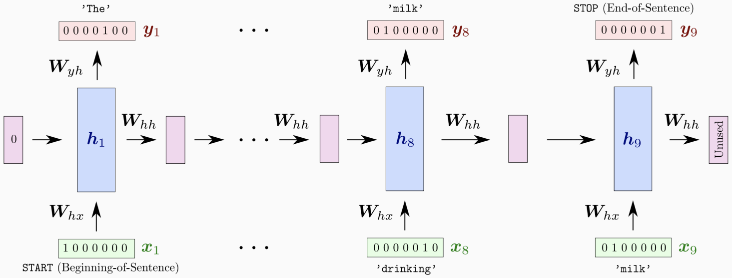


▷ Allows you to learn from arbitrarily long sequences

▷ Sharing the architecture ⇒ fewer parameters ⇒ training requires less data and the final prediction can be expected to be more accurate

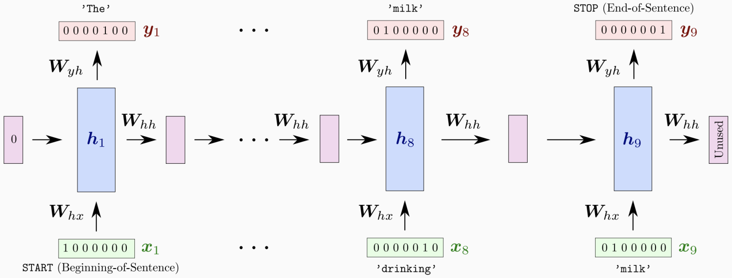
A simple shallow RNN for sentence generation

- This is an unfolded representation of an RNN



A simple shallow RNN for sentence generation

- This is an unfolded representation of an RNN



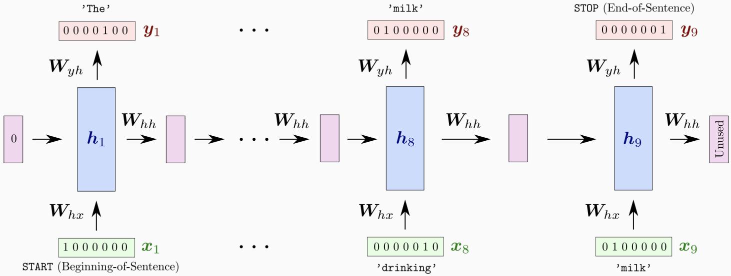
Vanilla RNN

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

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

A simple shallow RNN for sentence generation

- This is an unfolded representation of an RNN

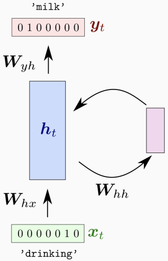


Vanilla RNN

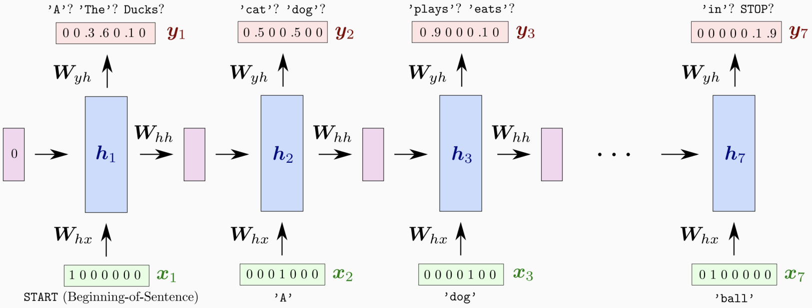
$$h_t = g\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$$

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

- Folded representation: RNN $\approx$ ANN with loops

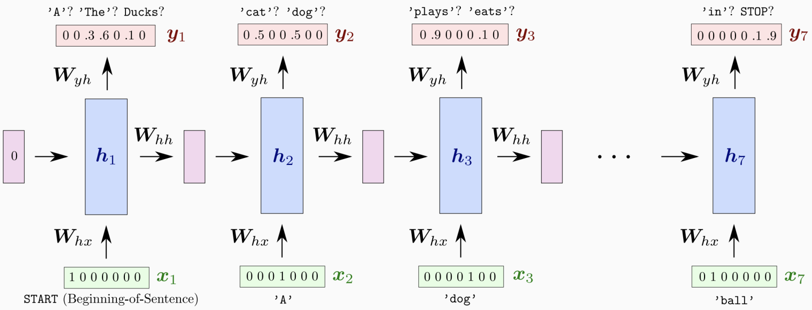


Generate a sentence in practice



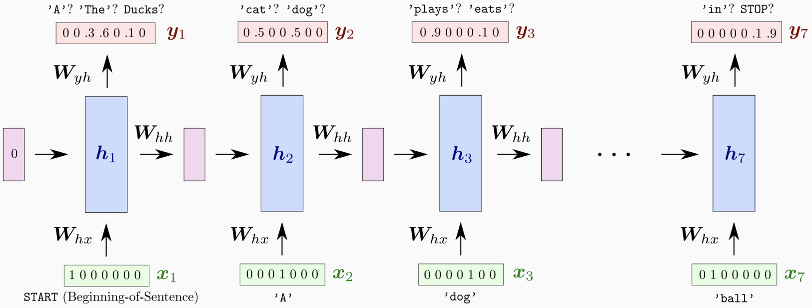
- Provide START, get all the probabilities
  $\mathbb{P}(\text{next word} | \text{current word} = \texttt{START})$

Generate a sentence in practice



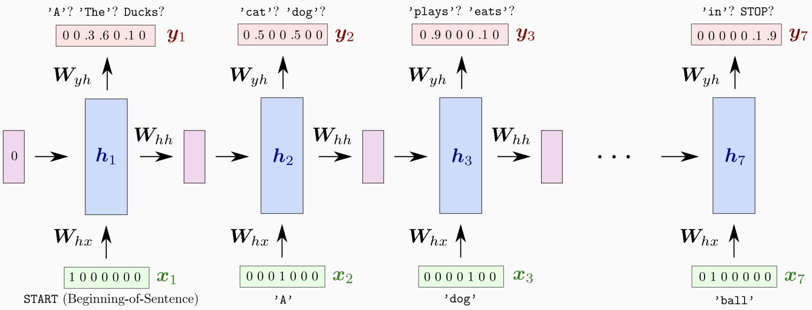
- Provide START, get all the probabilities
  $\mathbb{P}\left(\text{next word}|\text{current word} = \texttt{START}\right)$
- Select one of these words according to their probabilities, let say 'A',

Generate a sentence in practice



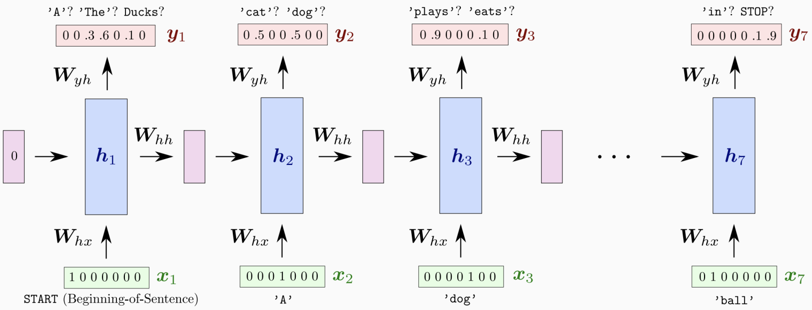
- Provide `START`, get all the probabilities $\mathbb{P}\left(\text{next word}|\text{current word} = \texttt{START}\right)$
- Select one of these words according to their probabilities, let say '`A`',
- Provide '`A`' and the past, and get $\mathbb{P}\left(\text{next word}|\text{current word} = \texttt{A}\right)$

Generate a sentence in practice



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

Generate a sentence in practice



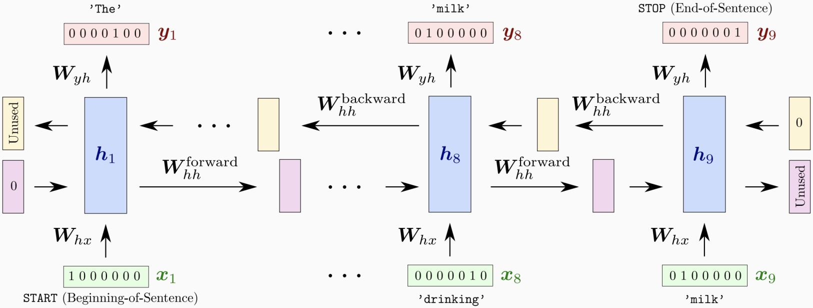
- Provide `START`, get all the probabilities $\mathbb{P}(\text{next word}|\text{current word} = \texttt{START})$
- Select one of these words according to their probabilities, let say '`A`',
- Provide '`A`' and the past, and get $\mathbb{P}(\text{next word}|\text{current word} = \texttt{A})$
- Repeat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

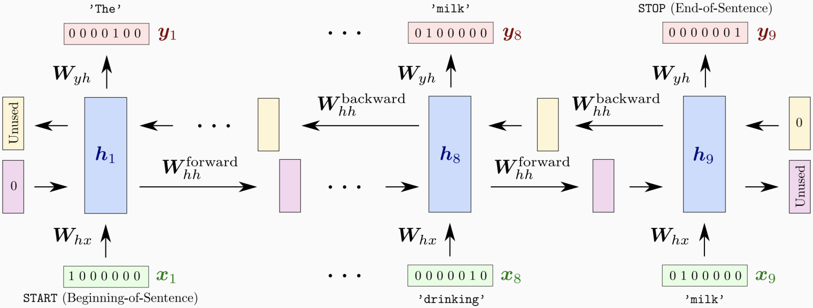
Bidirectional RNN

- 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



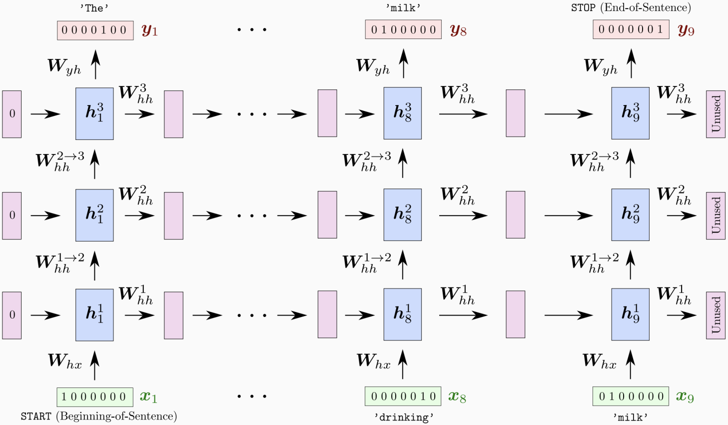
Bidirectional RNN

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

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

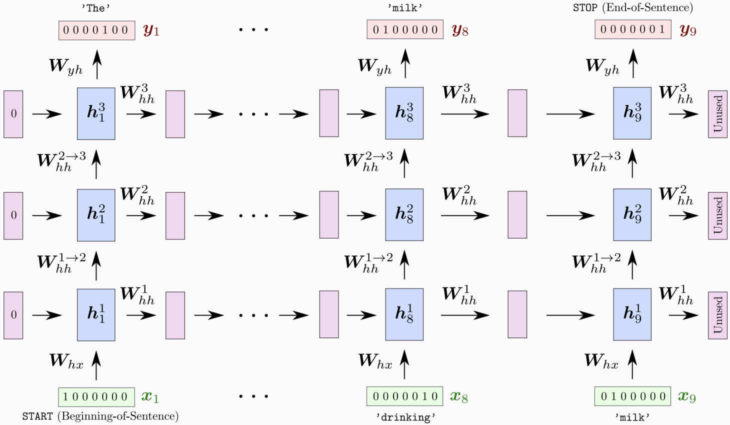
Deep RNN

- Multiple layers per time step (a feature hierarchy)



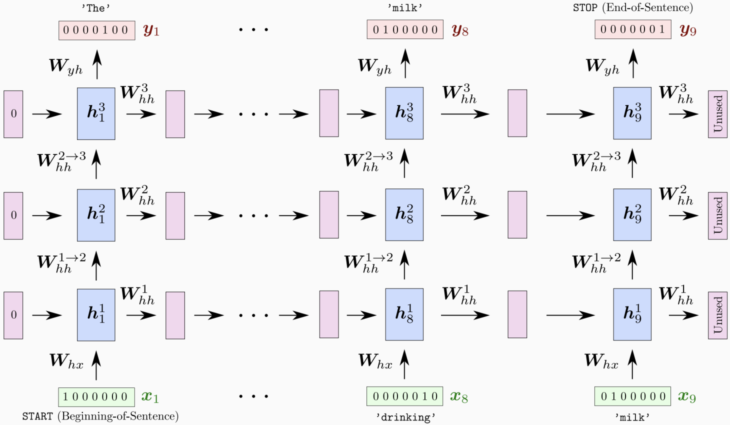
Deep RNN

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

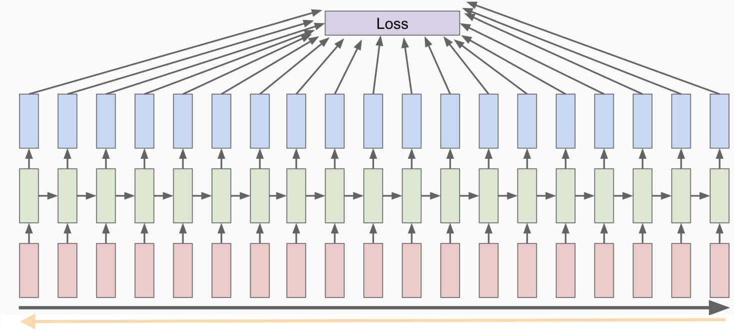


Deep RNN

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

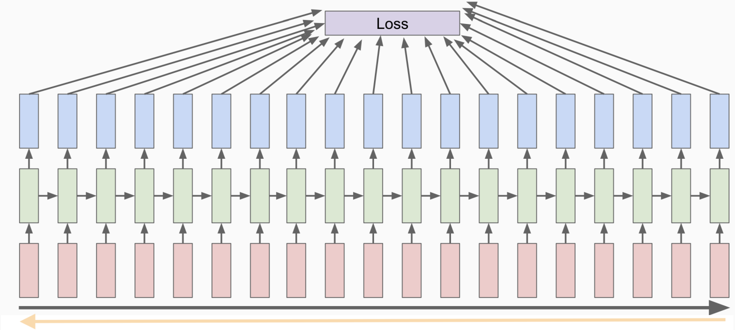


Learning phase for RNN

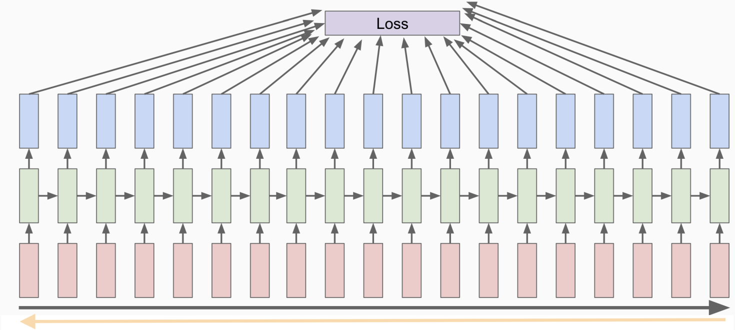


- Similar to standard backprop for training a traditional NN

Learning phase for RNN

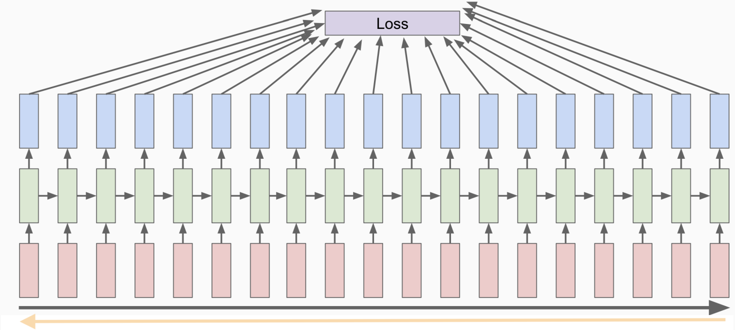


- Similar to standard backprop for training a traditional NN
- Take into account that parameters are shared by all steps in the network



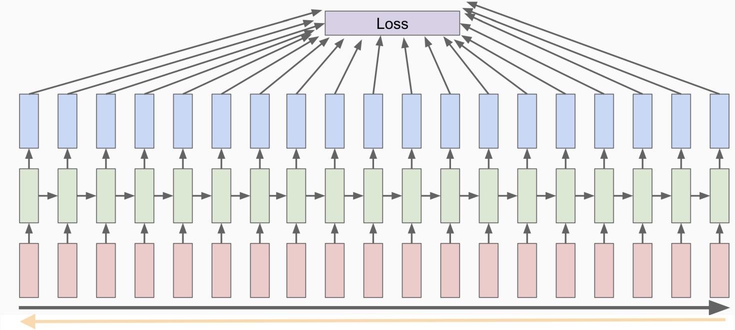
- Similar to standard backprop for training a traditional NN
- Take into account that parameters are shared by all steps in the network
- Forward through the entire sequence to compute the loss

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

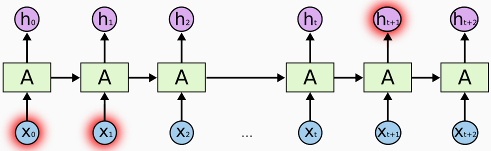
Learning phase for RNN



- 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
- Exercise: derive the gradients for the loss (cross-entropy for classification task), w.r.t. W_{hh}.

Language generating RNN: limitations

- Vanilla RNN have difficulties learning long-term dependencies

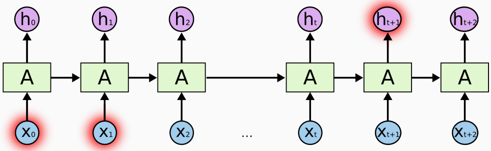


`I grew up in France ... I speak fluent ???`

(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 ???`

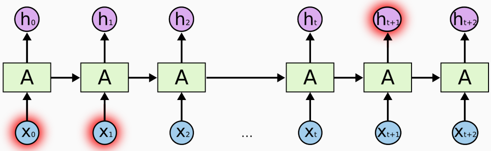
(we need the context of France from further back)

- Vanishing/exploding gradient problem

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

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)

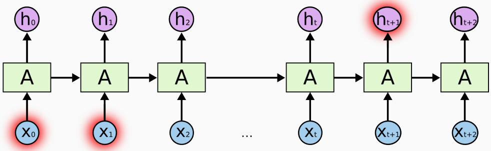
- Vanishing/exploding gradient problem

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

- 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

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

- 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

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

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

Long-Short Term Memory

$$
\begin{aligned}
g_t &= g\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right) && \text{(input modulation gate)}\\
c_t &= g_t && \text{(place memory in a cell unit c)}\\
h_t &= c_t && \\
y_t &= \mathsf{softmax}\left(W_{yh}h_t + b_y\right) && \text{(use } h_t \text{ for prediction)}
\end{aligned}
$$

Long-Short Term Memory

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

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

$$h_t = c_t$$

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

Long-Short Term Memory

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

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

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

$$h_t = c_t$$

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

Long-Short Term Memory

$$
\begin{aligned}
i_t &= \text{sigm}\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right) && \text{(input gate)} \\
f_t &= \text{sigm}\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f\right) && \text{(forget gate)} \\
g_t &= g\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right) && \text{(input modulation gate)} \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t && \text{(but can forget some of its memories)} \\
h_t &= c_t \\
y_t &= \text{softmax}\left(W_{yh}h_t + b_y\right)
\end{aligned}
$$

Long-Short Term Memory

$$o_t = \mathrm{sigm}\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right) \qquad \text{(output gate)}$$

$$i_t = \mathrm{sigm}\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right) \qquad \text{(input gate)}$$

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

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

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

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

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

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

Sommaire

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.

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

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.

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

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

A preliminary version of self-attention

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

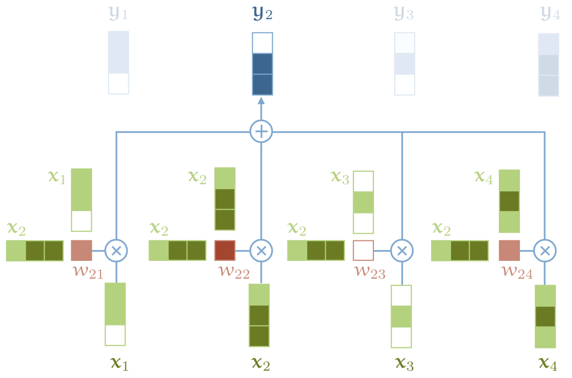
- Each output is a weighted average of all inputs where the weights w_{ij} are row-normalized such that they sum to 1
- The weights are directly derived from the inputs, e.g.

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

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

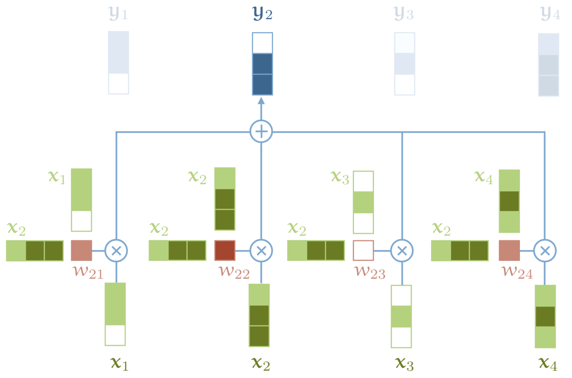
A preliminary version of self-attention

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



A preliminary version of self-attention

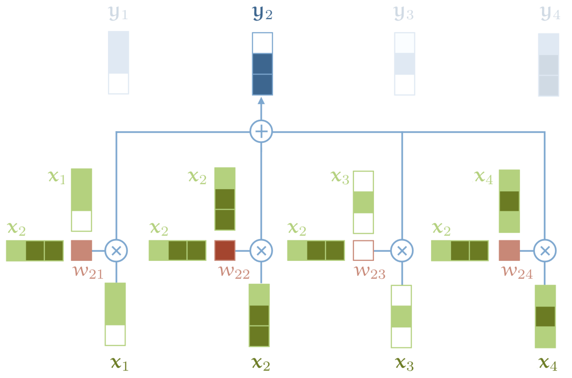
from http://peterbloem.nl/blog/transformers



- A few other ingredients are needed for a complete transformer

A preliminary version of self-attention

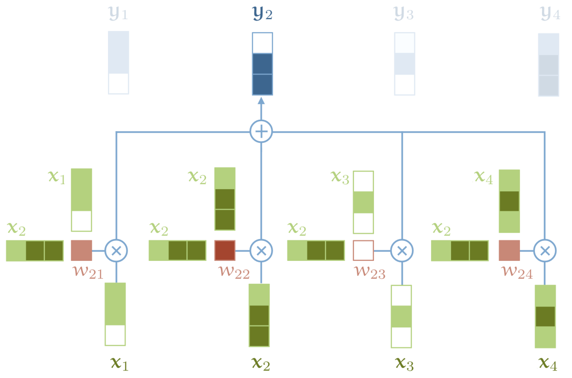
from http://peterbloem.nl/blog/transformers



- 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

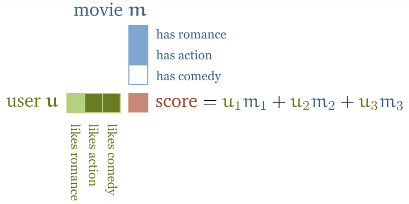
What's the point?

- 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

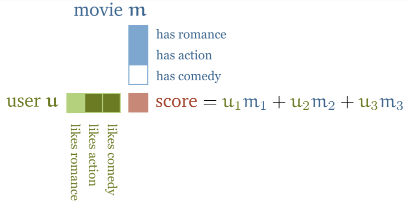
1. create manual features for movies and for users



A preliminary version of self-attention

Another example: movie recommendation

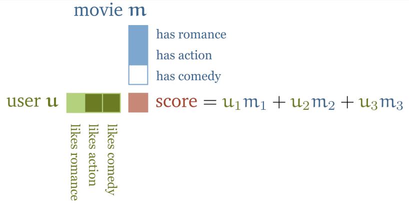
1. create manual features for movies and for users
   - how much romance there is in the movie, and how much action,



A preliminary version of self-attention

Another example: movie recommendation

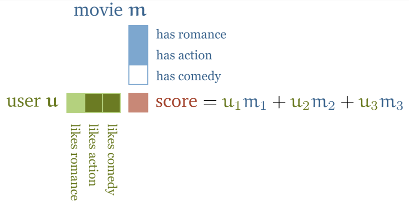
1. create manual features for movies and for users
   - how much romance there is in the movie, and how much action,
   - how much they enjoy romantic movies and how much they enjoy action-based movies



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

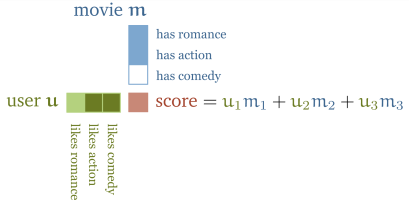


2. The dot product between the two feature vectors gives a score for how well the attributes of the movie match what the user enjoys

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

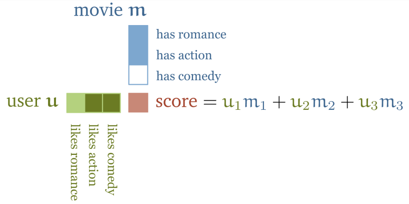


2. The dot product between the two feature vectors gives a score for how well the attributes of the movie match what the user enjoys
3. Replace user feature u by movies that she liked

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 ≈ 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 $x_1, \ldots x_N$
- 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 $x_1, \ldots x_N$
- 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 $v_{\texttt{The}}$ 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 $x_1, \ldots x_N$
- 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

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)
  - ▷ 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`"

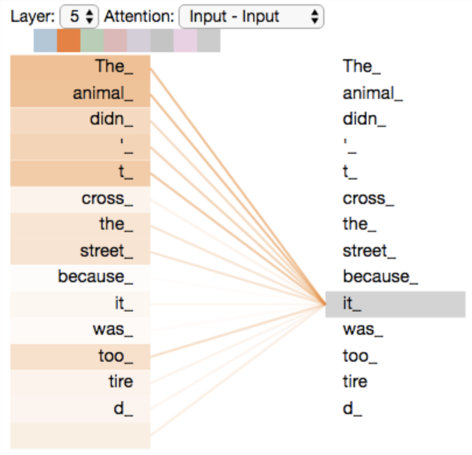
- '`The`': not very relevant to the interpretation of the other words in the sentence

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

- Helpful to interpret who sleeps

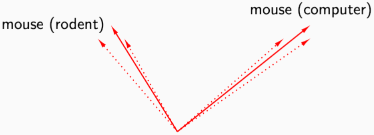
▷ Desire 2: for nouns like '`dog`' and verbs like '`sleeps`', learn an embedding $v_{\texttt{dog}}$ and $v_{\texttt{sleeps}}$ that have a high, positive dot product

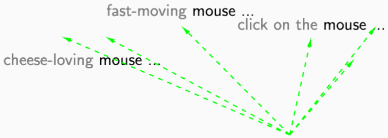
Learning the embedding: attention weights



- 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`"

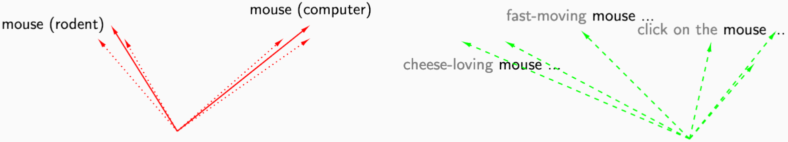
from https://ai.stanford.edu/blog/contextual/





Contextual embeddings

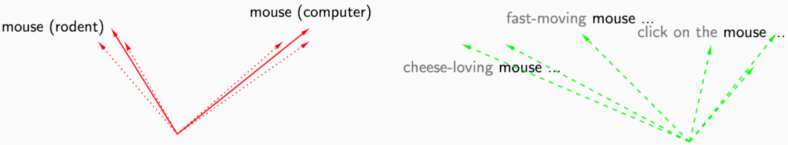
from https://ai.stanford.edu/blog/contextual/



- Word2Vec: word embeddings with semantic meaning learnt through unsupervised learning (predict word from context)

Contextual embeddings

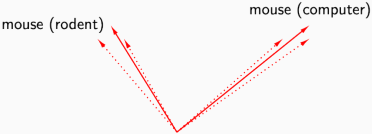
from https://ai.stanford.edu/blog/contextual/

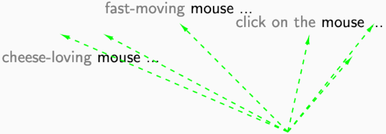


- Word2Vec: word embeddings with semantic meaning learnt through unsupervised learning (predict word from context)
- Yields one fixed learnt embedding for each word

Contextual embeddings

from https://ai.stanford.edu/blog/contextual/

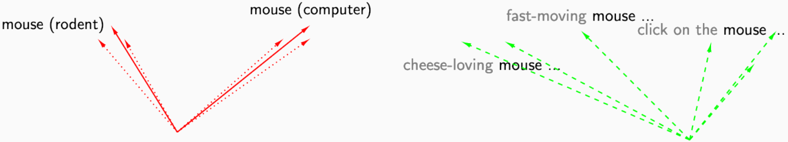




- Word2Vec: word embeddings with semantic meaning learnt through unsupervised learning (predict word from context)
- Yields one fixed learnt embedding for each word
- But meaning depends on context

Contextual embeddings

from https://ai.stanford.edu/blog/contextual/



- Word2Vec: word embeddings with semantic meaning learnt through unsupervised learning (predict word from context)
- Yields one fixed learnt embedding for each word
- But meaning depends on context
  - ▷ *Self-attention transforms input individual word embeddings (x_i) into contextual word embeddings (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

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

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

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

- **(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

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

- **(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

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

- **(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:

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

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:

$$
\begin{aligned}
q_i &= x_i && \text{(Query)} \\
k_i &= x_i && \text{(Key)} \\
v_i &= x_i && \text{(Value)}
\end{aligned}
$$

and then write out the output as:

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

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

$$
\begin{aligned}
q_i &= W_q x_i && \text{(Query)} \\
k_i &= W_k x_i && \text{(Key)} \\
v_i &= W_v x_i && \text{(Value)}
\end{aligned}
$$

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

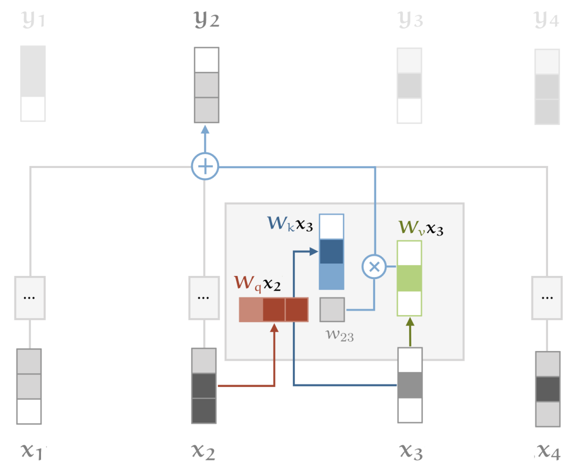


Figure 1: Illustration of the self-attention with key, query and value transformations

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 $r = 1, 2, \ldots$

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

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

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

Multi-head self attention layer

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

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

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

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

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

Multi-head self attention layer

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

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

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

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

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

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

A multi-head attention

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

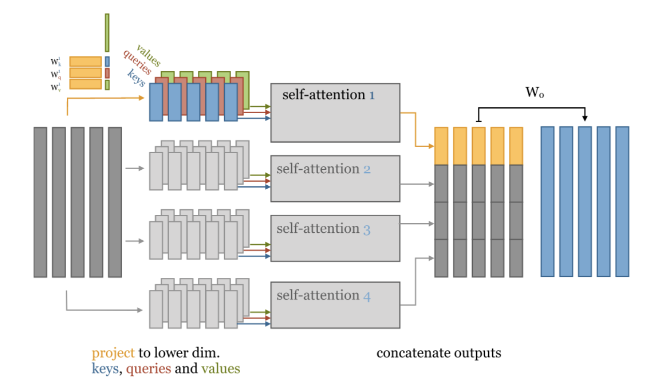


Figure 2: Illustration of multi-head self-attention with 4 heads. To get our keys, queries and values, we project the input down to vector sequences of smaller dimension.

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)

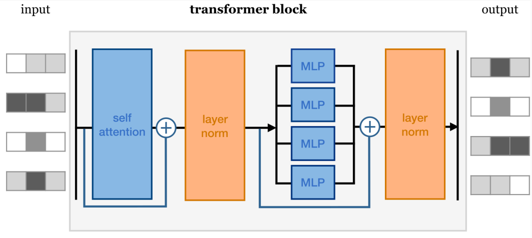
  If we give a query key and match it to a database of available keys, then the data structure returns the corresponding matched value
- Similar here
  - matching done by scalar products
  - softmax ensures a soft-matching
  - keys are matched to queries in some extent

- "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?

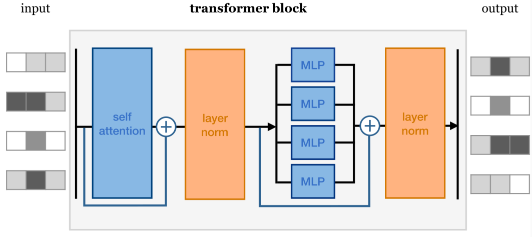
- This is an architecture



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

Transformer?

- This is an architecture

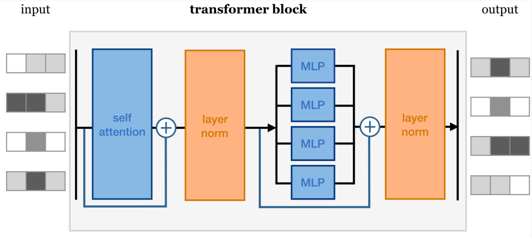


from http://peterbloem.nl/blog/transformers

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

Transformer?

- This is an architecture

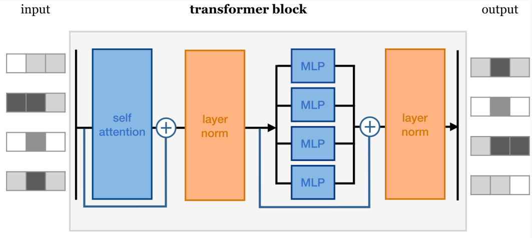


from http://peterbloem.nl/blog/transformers

- 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

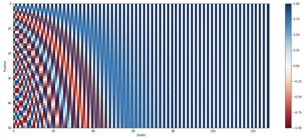
- Unlike sequence models (such as RNNs or LSTMs), self-attention layers are permutation-equivariant
- Meaning that

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

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

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

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



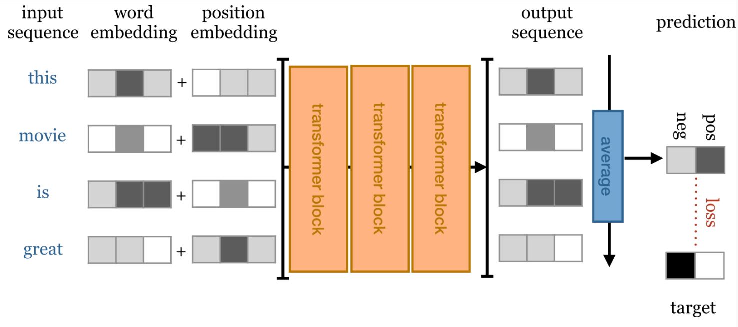
The 128-dimensional positional encoding for a sentence with a maximum length of 50. Each row represents the encoding vector.

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

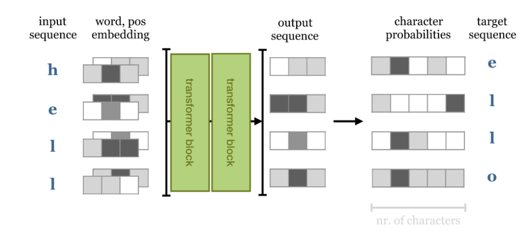
- Goal: build a sequence classifier for sentiment analysis
- IMDb sentiment classification dataset
  - (input) movie reviews (sequences of words)
  - (output) classification labels: positive or negative



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

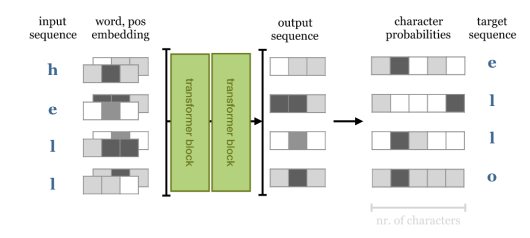
Text generation transformer

- Goal: predict the next character in a sequence



Text generation transformer

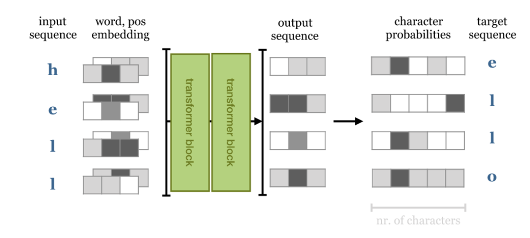
- Goal: predict the next character in a sequence



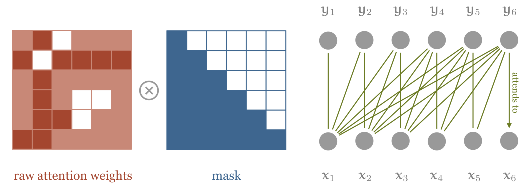
- With a transformer, the output depends on the entire input sequence: vacuously easy task!

Text generation transformer

- Goal: predict the next character in a sequence

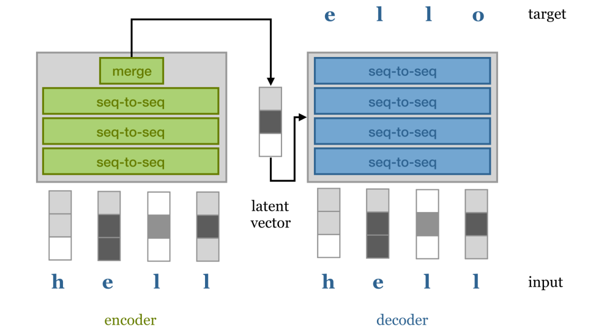


- With a transformer, the output depends on the entire input sequence: vacuously easy task!
- Solution: apply a mask to ensure that it cannot look forward into the sequence



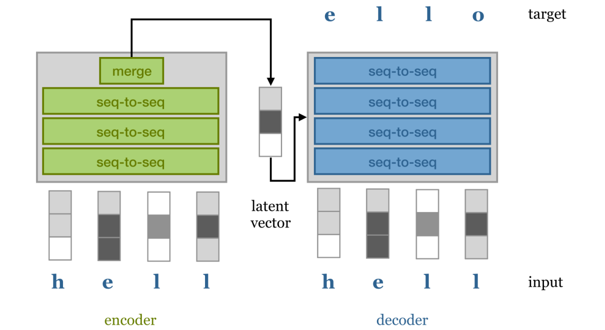
The original transformer

- Vaswani et al. (2017)



The original transformer

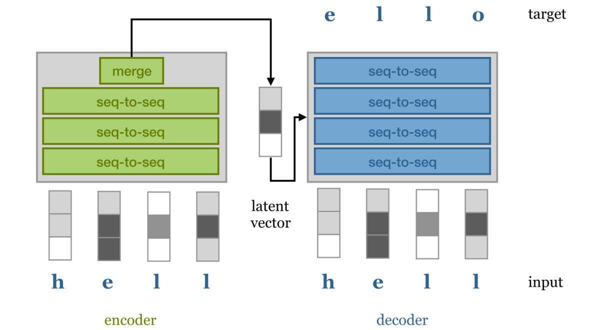
- Vaswani et al. (2017)



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

The original transformer

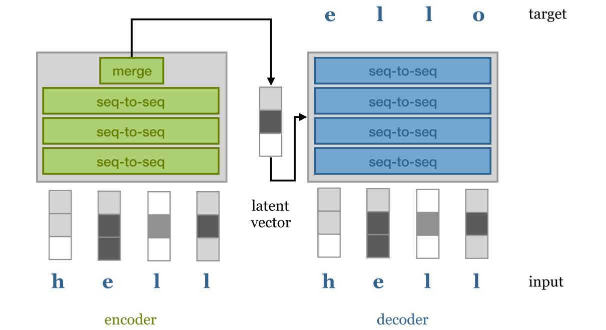
- Vaswani et al. (2017)



- 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

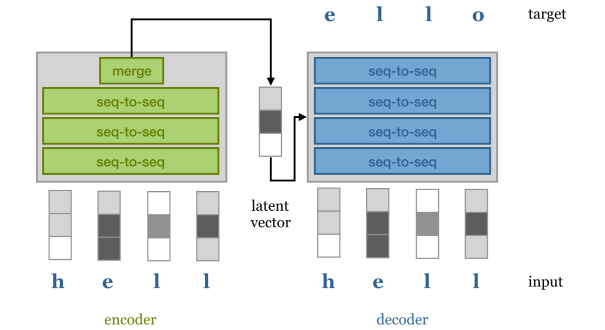
- Vaswani et al. (2017)



- 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 encoder 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 encoder 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 encoder blocks
  - Pre-trained on a large general-domain corpus (English books and Wikipedia)
  - Pre-training possible through masking or next-sequence prediction

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 encoder blocks
  - Pre-trained on a large general-domain corpus (English books and Wikipedia)
  - Pre-training possible through masking or next-sequence prediction
- GPT, Mistral, LLaMa, *etc.*: text generation (prediction of the next word with a stack of decoder only transformers)

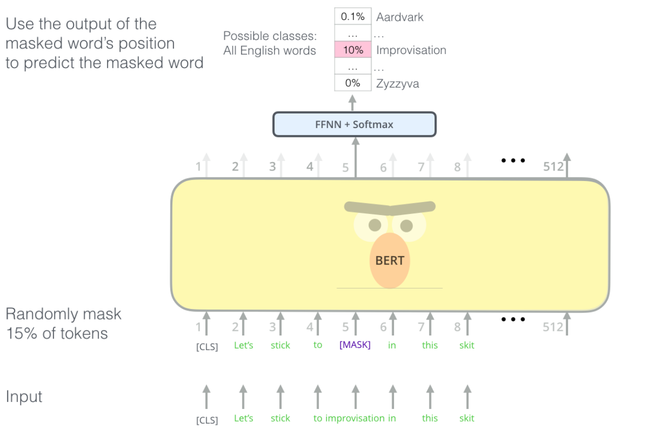
Modern transformers

- 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 encoder blocks
  - Pre-trained on a large general-domain corpus (English books and Wikipedia)
  - Pre-training possible through masking or next-sequence prediction
- GPT, Mistral, LLaMa, *etc.*: text generation (prediction of the next word with a stack of decoder only transformers)
- Recent models (e.g., T5, BART): utilize both encoder and decoder for tasks like translation and summarization

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 encoder blocks
  - Pre-trained on a large general-domain corpus (English books and Wikipedia)
  - Pre-training possible through masking or next-sequence prediction
- GPT, Mistral, LLaMa, *etc.*: text generation (prediction of the next word with a stack of decoder only transformers)
- Recent models (e.g., T5, BART): utilize both encoder and decoder for tasks like translation and summarization
- Large Language Models (LLMs): trained on vast corpora, capable of performing a wide range of tasks

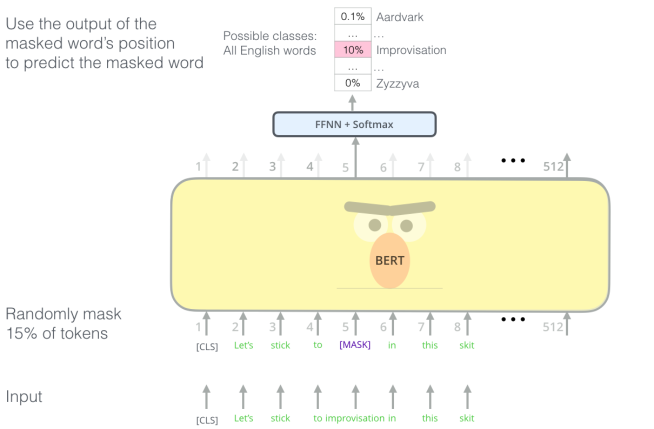
BERT pretraining: masked language model (MLM)



from https://jalammar.github.io/illustrated-bert/

- BERT: stack of encoder-only transformers

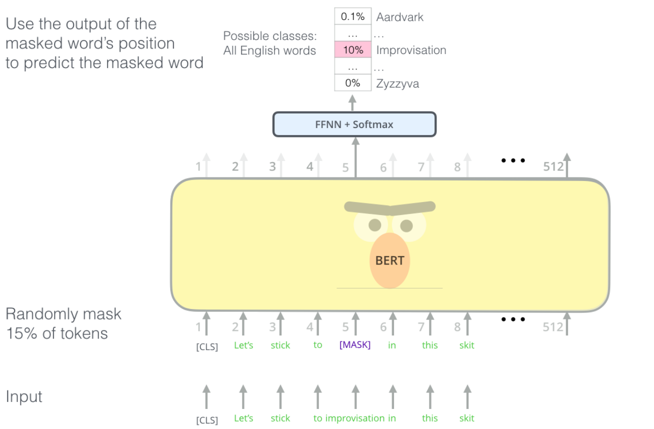
BERT pretraining: masked language model (MLM)



from https://jalammar.github.io/illustrated-bert/

- BERT: stack of encoder-only transformers
- Pre-training the LM: predict randomly masked tokens

BERT pretraining: masked language model (MLM)



from https://jalammar.github.io/illustrated-bert/

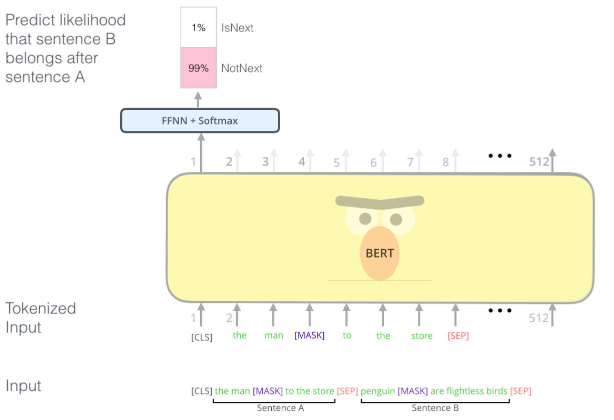
- BERT: stack of encoder-only transformers
- Pre-training the LM: predict randomly masked tokens
- Not a generative model: it is not a next word prediction task (not autoregressive), so the full sequence can be processed simultaneously

ICD-10 classification of patient records

```
JJ/MM/AAAA  Consultation en cours de traitement  Docteur DOCTEUR
VCE

Fait le JJ/MM/AAAA par le Docteur DOCTEUR

Rappel : T1 N2a du cavum (UCNT) traité par évidement en Mois
AAAA + x cures de chimiothérapie + radiothérapi Mois AAAA :
carcinome neuroendocrine à petites cellules avec métastases
hépatiques et osseuses. NSE :443.
VEPESIDE-HOLOXAN.

Mois AAAA : progression hépatique : CARBOPLATINE - VP16

Patient revu avant 3 ème cycle de chimiothérapie par
CARBOPLATINE -VP 16.

Intercure :
Pas d'asthénie.
Pas de toxicité digestive.
Pas de fièvre.
```

- What is the diagnosis of this document in the International Classification of Diseases (ICD-10)?

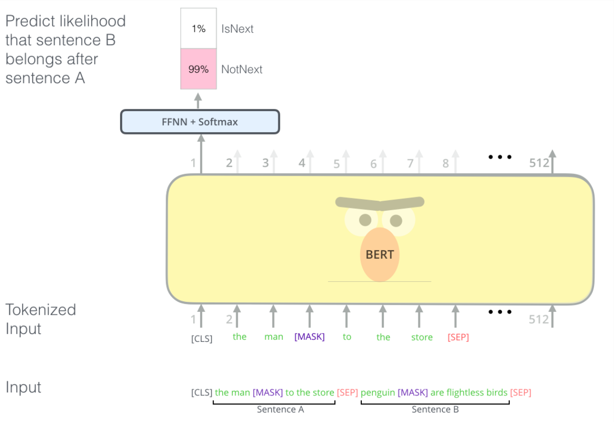
BERT for classification



from https://jalammar.github.io/illustrated-bert/

- Special [CLS] token for classification, gets signal from full sequence through attention mechanism

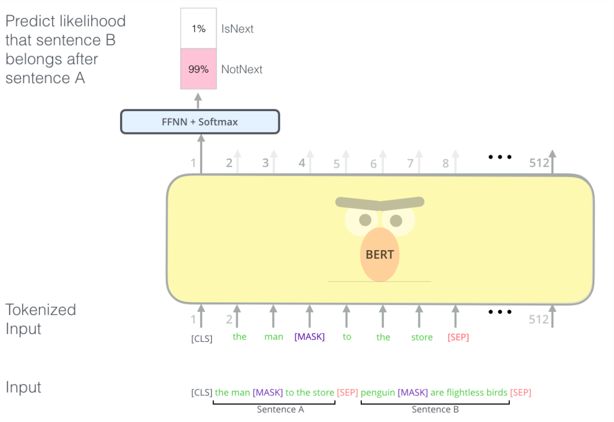
BERT for classification



from https://jalammar.github.io/illustrated-bert/

- Special [CLS] token for classification, gets signal from full sequence through attention mechanism
- Then goes through FFNN + softmax to determine score of each class

BERT for classification

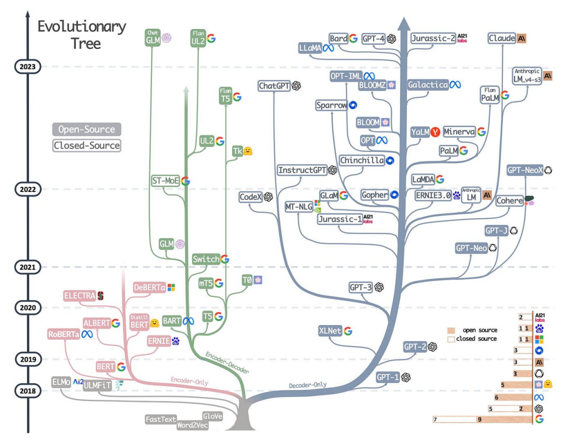


from https://jalammar.github.io/illustrated-bert/

- Special [CLS] token for classification, gets signal from full sequence through attention mechanism
- Then goes through FFNN + softmax to determine score of each class
- Model must be trained on the task

Sommaire

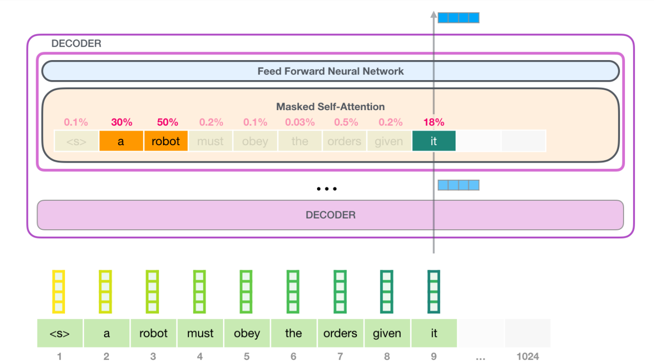
Large Language Models (LLM) Jungle



from https://arxiv.org/pdf/2304.13712.pdf

- Plenty of LLMs, especially since the release of open-source foundation models (LLaMa, etc.)
- Most recent models are decoder-only (like ChatGPT): text generation only

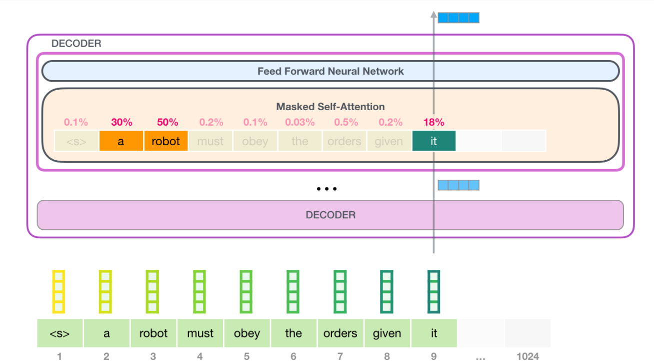
Decoder-only LLMs (text generation)



from https://jalammar.github.io/illustrated-gpt2/

- Goal: predict the next token in a sequence (trained on text data with a causal mask to ensure that it cannot look forward into the sequence)

Decoder-only LLMs (text generation)



from https://jalammar.github.io/illustrated-gpt2/

- Goal: predict the next token in a sequence (trained on text data with a causal mask to ensure that it cannot look forward into the sequence)
- Also include instruction datasets for specific virtual assistant tasks

Training Large Language Models

- **Causal Language Modeling (CLM)**: training the model to predict the next token given previous tokens (autoregressive)

Training Large Language Models

- **Causal Language Modeling (CLM)**: training the model to predict the next token given previous tokens (autoregressive)
- Uses vast amounts of unlabeled text data

Training Large Language Models

- **Causal Language Modeling (CLM)**: training the model to predict the next token given previous tokens (autoregressive)
- Uses vast amounts of unlabeled text data
- **Instruction Tuning / Supervised Fine-Tuning**: fine-tunes the model on datasets containing instructions and appropriate responses

Training Large Language Models

- **Causal Language Modeling (CLM)**: training the model to predict the next token given previous tokens (autoregressive)
- Uses vast amounts of unlabeled text data
- **Instruction Tuning / Supervised Fine-Tuning**: fine-tunes the model on datasets containing instructions and appropriate responses
- Helps the model better follow human instructions and improve usability

Reinforcement Learning from Human Feedback (RLHF)

- **Goal**: align the model's outputs with human preferences

Reinforcement Learning from Human Feedback (RLHF)

- **Goal**: align the model's outputs with human preferences
- **Process**:
  1. Collect data: human feedback on model outputs (e.g., rank responses)

Reinforcement Learning from Human Feedback (RLHF)

- **Goal**: align the model's outputs with human preferences
- **Process**:
  1. Collect data: human feedback on model outputs (e.g., rank responses)
  2. Train a **reward model** to predict human preferences

Reinforcement Learning from Human Feedback (RLHF)

- **Goal**: align the model's outputs with human preferences
- **Process**:
    1. Collect data: human feedback on model outputs (e.g., rank responses)
    2. Train a **reward model** to predict human preferences
    3. Fine-tune the language model using reinforcement learning to maximize the reward model's score

Reinforcement Learning from Human Feedback (RLHF)

- **Goal**: align the model's outputs with human preferences
- **Process**:
  1. Collect data: human feedback on model outputs (e.g., rank responses)
  2. Train a **reward model** to predict human preferences
  3. Fine-tune the language model using reinforcement learning to maximize the reward model's score
- **Benefit**: reduces harmful, biased, or unhelpful outputs

RLHF Example: Generating Code

- **Scenario**: Generate code snippets that perform a desired function

RLHF Example: Generating Code

- **Scenario**: Generate code snippets that perform a desired function
- **Preference Generation**:
  - Generate multiple code samples

RLHF Example: Generating Code

- **Scenario**: Generate code snippets that perform a desired function
- **Preference Generation**:
  - Generate multiple code samples
  - Automatically test if the code compiles or passes unit tests

RLHF Example: Generating Code

- **Scenario**: Generate code snippets that perform a desired function
- **Preference Generation**:
  - Generate multiple code samples
  - Automatically test if the code compiles or passes unit tests
  - Use results to rank or score the code samples

RLHF Example: Generating Code

- **Scenario**: Generate code snippets that perform a desired function
- **Preference Generation**:
  - Generate multiple code samples
  - Automatically test if the code compiles or passes unit tests
  - Use results to rank or score the code samples
- **Automated Feedback**: Enables large-scale data collection without human annotation

RLHF Example: Generating Code

- **Scenario**: Generate code snippets that perform a desired function
- **Preference Generation**:
  - Generate multiple code samples
  - Automatically test if the code compiles or passes unit tests
  - Use results to rank or score the code samples
- **Automated Feedback**: Enables large-scale data collection without human annotation
- **Training**: Use this feedback to train the reward model and fine-tune the LLM

How Generative LLMs Work: A Real Example

- LLMs predict the next token based on all previous tokens
- Each token is generated one after another in an autoregressive manner

My prompt to an LLM

I have to prepare slides for a 3 hour NLP course at Mines ParisTech during the Large Scale Machine Learning week.
Can you write a beamer (latex) presentation for me? It should cover most essential topics in NLP.

How Generative LLMs Work: A Real Example

- LLMs predict the next token based on all previous tokens
- Each token is generated one after another in an autoregressive manner

My prompt to an LLM

I have to prepare slides for a 3 hour NLP course at Mines ParisTech during the Large Scale Machine Learning week.
Can you write a beamer (latex) presentation for me? It should cover most essential topics in NLP.

LLM's response generation process

1. Model processes the entire prompt
2. Generates first token: "#"
3. Given prompt + "#", generates next token: "␣" (space)
4. Given prompt + "# ", generates: "LaTeX"
5. Continues token by token: "# LaTeX Beamer Presentation..."

Beginning of the LLM response

 # LaTeX Beamer Presentation for NLP Course
I'll create a comprehensive beamer presentation structure for your 3-

How Large are Large Language Models?

- **BERT (base / large)**: 110M / 340M parameters
- **GPT-2**: 1.5B parameters
- **GPT-3**: 175B parameters
- **Megatron-Turing NLG**: 530B parameters
- **PaLM**: 540B parameters
- **GPT-4.5**: Exact parameter count not publicly disclosed; speculated to be 12T (!)

How Large are Large Language Models?

- **BERT (base / large)**: 110M / 340M parameters
- **GPT-2**: 1.5B parameters
- **GPT-3**: 175B parameters
- **Megatron-Turing NLG**: 530B parameters
- **PaLM**: 540B parameters
- **GPT-4.5**: Exact parameter count not publicly disclosed; speculated to be 12T (!)
- Larger models generally perform better but are more resource-intensive. Also seems to be a plateau (+ limited amount of data)

Zero-shot and Few-shot Learning with LLMs

- **Zero-shot Learning**: Using a model for tasks it wasn't explicitly trained on

Zero-shot and Few-shot Learning with LLMs

- **Zero-shot Learning**: Using a model for tasks it wasn't explicitly trained on
- **Prompt Engineering**: Crafting prompts with examples to elicit desired responses

Zero-shot and Few-shot Learning with LLMs

- **Zero-shot Learning**: Using a model for tasks it wasn't explicitly trained on
- **Prompt Engineering**: Crafting prompts with examples to elicit desired responses
- **In-Context Learning**: Providing a few examples in the prompt to guide the model
  - The model learns from the context provided in the prompt without updating its parameters
  - Enables adaptation to new tasks with minimal examples

Zero-shot and Few-shot Learning with LLMs

- **Zero-shot Learning**: Using a model for tasks it wasn't explicitly trained on
- **Prompt Engineering**: Crafting prompts with examples to elicit desired responses
- **In-Context Learning**: Providing a few examples in the prompt to guide the model
  - The model learns from the context provided in the prompt without updating its parameters
  - Enables adaptation to new tasks with minimal examples
- **Few-shot Learning**: Fine-tune the model on a small number of examples

Chain-of-Thought Prompting

- **Idea**: Encourage the model to generate intermediate reasoning steps before giving the final answer

Chain-of-Thought Prompting

- **Idea**: Encourage the model to generate intermediate reasoning steps before giving the final answer
- **Benefits**:
  - Improves performance on complex reasoning tasks
  - Makes the model's reasoning process transparent

Chain-of-Thought Prompting

- **Idea**: Encourage the model to generate intermediate reasoning steps before giving the final answer
- **Benefits**:
  - Improves performance on complex reasoning tasks
  - Makes the model's reasoning process transparent
- **Implementation**:
  - Include phrases like "Let's think step by step" in the prompt
  - Provide examples where the reasoning process is explicit
- New models (OpenAI o1, DeepSeekv2) use CoT "under the hood"

Chain-of-Thought Example

Question:

If there are 3 cars and each car has 4 wheels, how many wheels are there in total?

Chain-of-Thought Example

Question:

If there are 3 cars and each car has 4 wheels, how many wheels are there in total?

Standard LLM Response:

There are 12 wheels in total.

Chain-of-Thought Example

Question:

If there are 3 cars and each car has 4 wheels, how many wheels are there in total?

Standard LLM Response:

There are 12 wheels in total.

Chain-of-Thought LLM Response:

Each car has 4 wheels. There are 3 cars, so the total number of wheels is $3 \times 4 = 12$*. Therefore, there are 12 wheels in total.*

Hot Topics in NLP and LLMs (Updated)

- **Chain-of-Thought Reasoning**: Enhancing LLM reasoning capabilities through step-by-step explanations
- **In-Context Learning**: Models learning from context without parameter updates
- **Retrieval-Augmented Generation (RAG)**: Combining LLMs with external knowledge bases
- **Multimodal Models**: Combining text with images, audio, video (e.g., GPT-4, CLIP)
- **Alternatives to Transformers**: Exploring new architectures (e.g., Perceiver, Transformer-XL, Titans, Latent Space Models, Diffusion models)
- **Long Context models**: Handling longer inputs efficiently (e.g., Longformer, BigBird)
- **Efficient Fine-Tuning Methods**: Parameter-efficient tuning (LoRA, adapters)

Wrapping Up

- Two main neural network architectures for NLP: recurrent networks and transformers (based on attention)

Wrapping Up

- Two main neural network architectures for NLP: recurrent networks and transformers (based on attention)
- Ideas behind attention are surprisingly simple!

Wrapping Up

- Two main neural network architectures for NLP: recurrent networks and transformers (based on attention)
- Ideas behind attention are surprisingly simple!
- Backpropagation is used in all of them during the learning phase

Wrapping Up

- Two main neural network architectures for NLP: recurrent networks and transformers (based on attention)
- Ideas behind attention are surprisingly simple!
- Backpropagation is used in all of them during the learning phase
- LLMs (based on transformer architecture) have revolutionized NLP

Wrapping Up

- Two main neural network architectures for NLP: recurrent networks and transformers (based on attention)
- Ideas behind attention are surprisingly simple!
- Backpropagation is used in all of them during the learning phase
- LLMs (based on transformer architecture) have revolutionized NLP
- Advanced prompting techniques like chain-of-thought improve model capabilities

Wrapping Up

- Two main neural network architectures for NLP: recurrent networks and transformers (based on attention)
- Ideas behind attention are surprisingly simple!
- Backpropagation is used in all of them during the learning phase
- LLMs (based on transformer architecture) have revolutionized NLP
- Advanced prompting techniques like chain-of-thought improve model capabilities
- Full fine-tuning is often unnecessary; parameter-efficient methods suffice for many tasks

Wrapping Up

- Two main neural network architectures for NLP: recurrent networks and transformers (based on attention)
- Ideas behind attention are surprisingly simple!
- Backpropagation is used in all of them during the learning phase
- LLMs (based on transformer architecture) have revolutionized NLP
- Advanced prompting techniques like chain-of-thought improve model capabilities
- Full fine-tuning is often unnecessary; parameter-efficient methods suffice for many tasks
- Advanced topics like sparse attention, quantization, and adapters help manage large models and long sequences (see Appendix)

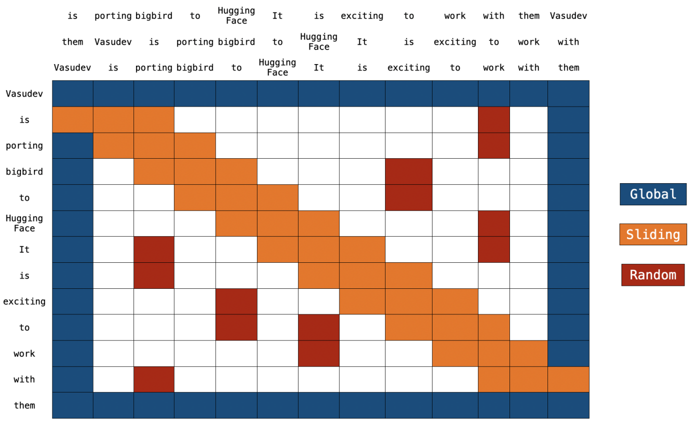
References i

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.

Sparse attention

- Attention computation is quadratic in sequence length
- Sparse attention: each token attends only to a few others



Downsizing LLMs: quantization

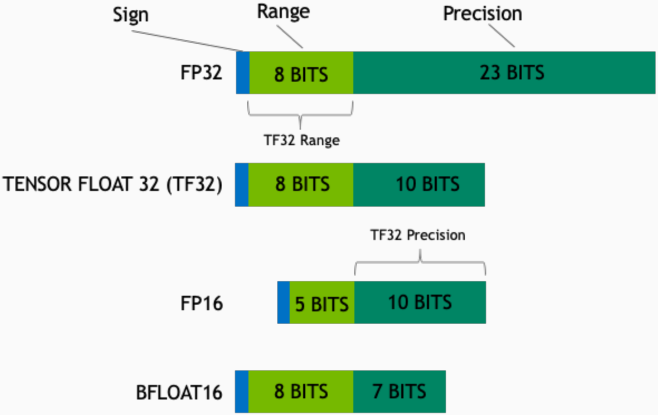
- Default parameter size on computers: 32 bits (float32)

Downsizing LLMs: quantization

- Default parameter size on computers: 32 bits (float32)
- For large values (far from 0), we do not need much precision for decimals

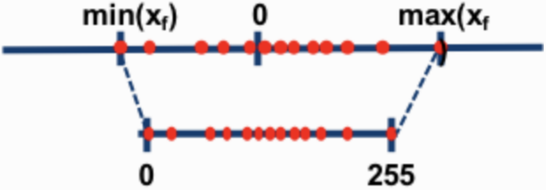
Downsizing LLMs: quantization

- Default parameter size on computers: 32 bits (float32)
- For large values (far from 0), we do not need much precision for decimals
- $\rightarrow$ Idea: use fewer bits for each parameter (smaller range of values, or fewer digits after decimal)



Downsizing LLMs: quantization

- Basic conversion to smaller floating point precision



from https://huggingface.co/blog/hf-bitsandbytes-integration

Fine-tuning LLMs

- LLMs are too large to be fully fine-tuned (and actually not even worth it)

Fine-tuning LLMs

- LLMs are too large to be fully fine-tuned (and actually not even worth it)
- Parameter-Efficient Fine-Tuning: fine-tune a subset of parameters

Fine-tuning LLMs

- LLMs are too large to be fully fine-tuned (and actually not even worth it)
- Parameter-Efficient Fine-Tuning: fine-tune a subset of parameters
- Adapters: add a trained "small" module to the frozen transformer. Can also incorporate non-textual data into a text model.

Fine-tuning LLMs

- LLMs are too large to be fully fine-tuned (and actually not even worth it)
- Parameter-Efficient Fine-Tuning: fine-tune a subset of parameters
- Adapters: add a trained "small" module to the frozen transformer. Can also incorporate non-textual data into a text model.
- LoRA-like methods: approximate fine-tuned matrices with low-rank matrices

Fine-tuning LLMs

- LLMs are too large to be fully fine-tuned (and actually not even worth it)
- Parameter-Efficient Fine-Tuning: fine-tune a subset of parameters
- Adapters: add a trained "small" module to the frozen transformer. Can also incorporate non-textual data into a text model.
- LoRA-like methods: approximate fine-tuned matrices with low-rank matrices
- Need "instruction datasets" to fine-tune models. These can be generated by another LLM (self-instruct).

Fine-tuning LLMs: LoRA

- Hypothesis: after fine-tuning, $W_{\text{finetuned}} - W_0$ is low-rank for matrices in self-attention layers (W_Q, W_K, W_V)

Fine-tuning LLMs: LoRA

- Hypothesis: after fine-tuning, $W_{\text{finetuned}} - W_0$ is low-rank for matrices in self-attention layers (W_Q, W_K, W_V)
- Directly train low-rank "delta" matrices

- Hypothesis: after fine-tuning, $W_{\text{finetuned}} - W_0$ is low-rank for matrices in self-attention layers (W_Q, W_K, W_V)
- Directly train low-rank "delta" matrices
- $h = W_0 x + \Delta W x = W_0 x + BAx$