Deep Learning for NLP: Representation learning ad Attention based architectures (Transformers)

M. Vazirgiannis

https://bit.ly/2rwmvQU

LIX, Ecole Polytechnique

*January **2025***

OUTLINE

- **Representation Learning for Text**
  - SVD - Word2Vec
  - Document Representations
- CNNs for text classification
- Attention - Transformer Architecture

Language model

- Goal: determine $P(s = w_1 \dots w_k)$ in some domain of interest

$$P(s) = \prod_{i=1}^{k} P\left(w_i \mid w_1 \dots w_{i-1}\right)$$

e.g., $P\left(w_1 w_2 w_3\right) = P\left(w_1\right) P\left(w_2 \mid w_1\right) P\left(w_3 \mid w_1 w_2\right)$

- Traditional n-gram language model assumption:
  "probability of a word depends only on **context** of $n-1$ previous words"

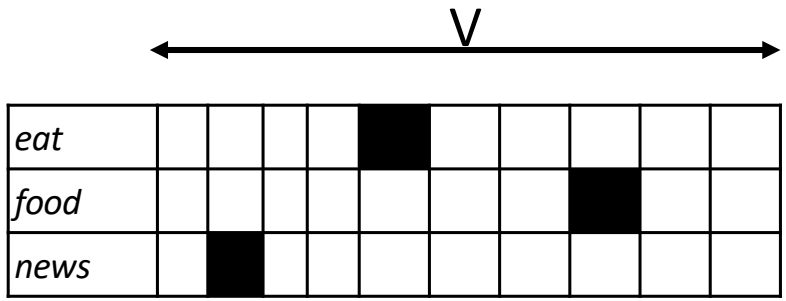
$$\Rightarrow \widehat{P}(s) = \prod_{i=1}^{k} P\left(w_i \mid w_{i-n+1} \dots w_{i-1}\right)$$

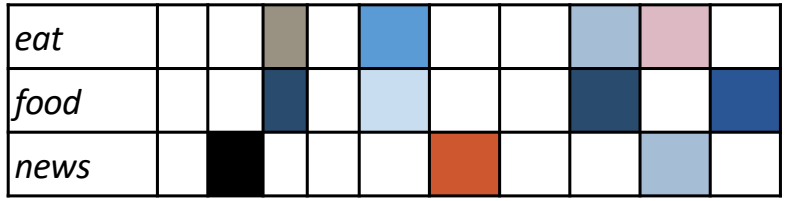
- i.e. "Paris is the capital of France located in Ile de ...."

- Typical ML-smoothing learning process (e.g., Katz 1987):
  1. compute $\widehat{P}\left(w_i \mid w_{i-n+1} \dots w_{i-1}\right) = \frac{\#w_{i-n+1} \dots w_{i-1} w_i}{\#w_{i-n+1} \dots w_{i-1}}$ on training corpus
  2. smooth to avoid zero probabilities

Representing Words

- **One-hot vector**
  - high dimensionality
  - sparse vectors
  - dimensions=|V| (10^6<|V|)
  - unable to capture semantic similarity between words
- **Distributional vector**
  - words that occur in similar contexts, tend to have similar meanings
  - each word vector contains the frequencies of all its neighbors
  - dimensions=|V|
  - computational complexity for ML algorithms



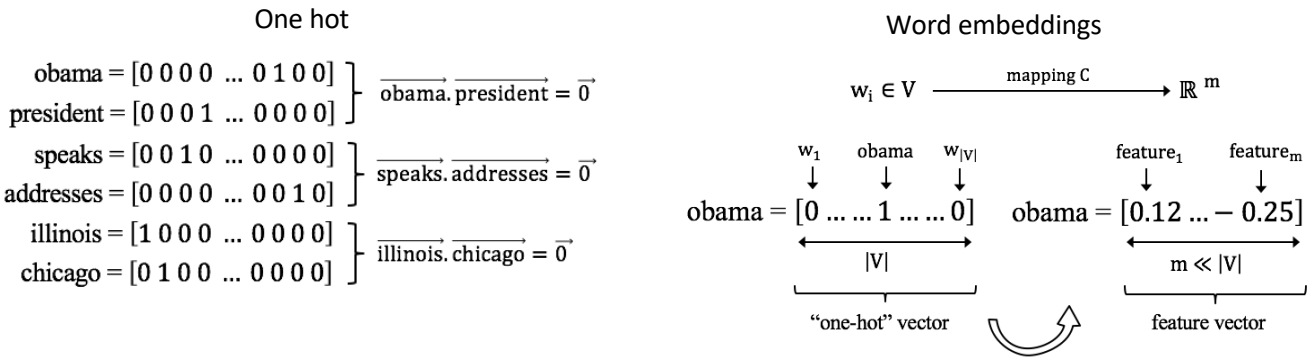


Representing Words

- **Word embeddings**
  - store the same contextual information in a low-dimensional vector
  - **densification** (sparse to dense)
  - **compression**
    - dimensionality reduction
    - dimensions=m 100<m<500
  - able to capture semantic similarity between words
  - learned vectors (unsupervised)
  - Learning methods
    - **SVD**
    - **word2vec**
    - **GloVe**

Text Similarity

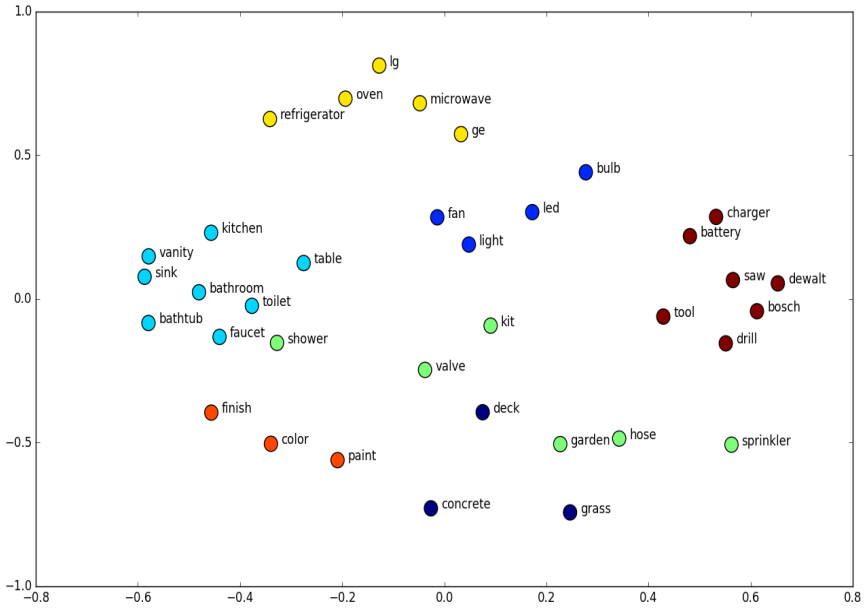
- We should assign similar probabilities (discover similarity) to *Obama speaks to the media in Illinois* and the *President addresses the press in Chicago*
- This does not happen because of the “one-hot” vector space representation



Representation Learning for Text

- **a word is defined by "the company it keeps" (Firth, 1957)**

- Word embeddings are a class of algorithms where each word is represented as real-valued vector.
- The learning process of these vectors is either joint with a neural network model on some task or is an unsupervised process.
- Similar words in meaning have similar representation.



SVD word embeddings

- Dimensionality reduction on co-occurrence matrix
- Create a $|V|x|V|$ word co-occurrence matrix *X*
- Apply SVD $X = USV^T$
- Take first *k* columns of *U*
- Use the k-dimensional vectors as representations for each word
- Able to capture semantic and syntactic similarity

LSI – an example

LSI application on a term – document matrix

C1: Human machine Interface for Lab ABC computer application

C2: A survey of user opinion of computer system response time

C3: The EPS user interface management system

C4: System and human system engineering testing of EPS

C5: Relation of user-perceived response time to error measurements

M1: The generation of random, binary unordered trees

M2: The intersection graph of path in trees

M3: Graph minors IV: Widths of trees and well-quasi-ordering

M4: Graph minors: A survey

- The dataset consists of 2 classes, 1st: "human – computer interaction" (c1-c5) 2nd: related to graph (m1-m4). After feature extraction the titles are represented as follows.

Indexing by Latent Semantic Analysis (1990) Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Richard Harshman, Journal of the American Society of Information Science

LSI – an example

$A=ULV^T$

A=

1	0	0	1	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0
0	1	1	2	0	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	0	0	0	0	0	1
0	0	0	0	0	1	1	1	0
0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1	1

LSI – an example

$A = ULV^T$

$U =$

0.22	-0.11	0.29	-0.41	-0.11	-0.34	0.52	-0.06	-0.41	0	0	0
0.20	-0.07	0.14	-0.55	0.28	0.50	-0.07	-0.01	-0.11	0	0	0
0.24	0.04	-0.16	-0.59	-0.11	-0.25	-0.30	0.06	0.49	0	0	0
0.40	0.06	-0.34	0.10	0.33	0.38	0.00	0.00	0.01	0	0	0
0.64	-0.17	0.36	0.33	-0.16	-0.21	-0.17	0.03	0.27	0	0	0
0.27	0.11	-0.43	0.07	0.08	-0.17	0.28	-0.02	-0.05	0	0	0
0.27	0.11	-0.43	0.07	0.08	-0.17	0.28	-0.02	-0.05	0	0	0
0.30	-0.14	0.33	0.19	0.11	0.27	0.03	-0.02	-0.17	0	0	0
0.21	0.27	-0.18	-0.03	-0.54	0.08	-0.47	-0.04	-0.58	0	0	0
0.01	0.49	0.23	0.03	0.59	-0.39	-0.29	0.25	-0.23	0	0	0
0.04	0.62	0.22	0.00	-0.07	0.11	0.16	-0.68	0.23	0	0	0
0.03	0.45	0.14	-0.01	-0.30	0.28	0.34	0.68	0.18	0	0	0

LSI – an example

$A=ULV^T$

L=

3.34	0	0	0	0	0	0	0	0
0	2.54	0	0	0	0	0	0	0
0	0	2.35	0	0	0	0	0	0
0	0	0	1.64	0	0	0	0	0
0	0	0	0	1.50	0	0	0	0
0	0	0	0	0	1.31	0	0	0
0	0	0	0	0	0	0.85	0	0
0	0	0	0	0	0	0	0.56	0
0	0	0	0	0	0	0	0	0.36
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

LSI – an example

$A=ULV^T$

V=

0.20	-0.06	0.11	-0.95	0.05	-0.08	0.18	-0.01	-0.06
0.61	0.17	-0.50	-0.03	-0.21	-0.26	-0.43	0.05	0.24
0.46	-0.13	0.21	0.04	0.38	0.72	-0.24	0.01	0.02
0.54	-0.23	0.57	0.27	-0.21	-0.37	0.26	-0.02	-0.08
0.28	0.11	-0.51	0.15	0.33	0.03	0.67	-0.06	-0.26
0.00	0.19	0.10	0.02	0.39	-0.30	-0.34	0.45	-0.62
0.01	0.44	0.19	0.02	0.35	-0.21	-0.15	-0.76	0.02
0.02	0.62	0.25	0.01	0.15	0.00	0.25	0.45	0.52
0.08	0.53	0.08	-0.03	-0.60	0.36	0.04	-0.07	-0.45

LSI – an example

Choosing the 2 largest singular values we have

$U_k =$

0.22	-0.11
0.20	-0.07
0.24	0.04
0.40	0.06
0.64	-0.17
0.27	0.11
0.27	0.11
0.30	-0.14
0.21	0.27
0.01	0.49
0.04	0.62
0.03	0.45

$L_k =$

3.34	0
0	2.54

$V_k^T =$

0.20	0.61	0.46	0.54	0.28	0.00	0.02	0.02	0.08
-0.06	0.17	-0.13	-0.23	0.11	0.19	0.44	0.62	0.53

LSI reconstruction (2 singular values)

$A_k =$

	C1	C2	C3	C4	C5	M1	M2	M3	M4
human	0.16	0.40	0.38	0.47	0.18	-0.05	-0.12	-0.16	-0.09
Interface	0.14	0.37	0.33	0.40	0.16	-0.03	-0.07	-0.10	-0.04
Computer	0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
User	0.26	0.84	0.61	0.70	0.39	0.03	0.08	0.12	0.19
System	0.45	1.23	1.05	1.27	0.56	-0.07	-0.15	-0.21	-0.05
Response	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
Time	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
EPS	0.22	0.55	0.51	0.63	0.24	-0.07	-0.14	-0.20	-0.11
Survey	0.10	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
Trees	-0.06	0.23	-0.14	-0.27	0.14	0.24	0.55	0.77	0.66
Graph	-0.06	0.34	-0.15	-0.30	0.20	0.31	0.69	0.98	0.85
Minors	-0.04	0.25	-0.10	-0.21	0.15	0.22	0.50	0.71	0.62

LSI Example

- Query: "human computer interaction" retrieves documents: c_1,c_2, c_4 but *not* c_3 and c_5.
- If we submit the same query (based on the transformation shown before) to the transformed matrix we retrieve (using cosine similarity) all c_1-c_5 even if c_3 and c_5 have no common keyword to the query.
- According to the transformation for the queries we have:

Query transformation

	query
human	1
Interface	0
computer	1
User	0
System	0
Response	0
Time	0
EPS	0
Survey	0
Trees	0
Graph	0
Minors	0

$q=$

1
0
1
0
0
0
0
0
0
0
0
0

Query transformation

$q^T =$

1	0	1	0	0	0	0	0	0	0	0	0

$U_k =$

0.22	-0.11
0.20	-0.07
0.24	0.04
0.40	0.06
0.64	-0.17
0.27	0.11
0.27	0.11
0.30	-0.14
0.21	0.27
0.01	0.49
0.04	0.62
0.03	0.45

$L_k =$

0.3	0
0	0.39

$q_n = q^T U_k L_k =$

0.138	-0.0273

Query transformation

Map docs to the 2 dim space $V_k L_k =$

0.20	-0.06
0.61	0.17
0.46	-0.13
0.54	-0.23
0.28	0.11
0.00	0.19
0.01	0.44
0.02	0.62
0.08	0.53

3.34	0
0	2.54

=

0.67	-0.15
2.04	0.43
1.54	-0.33
1.80	-0.58
0.94	0.28
0.00	0.48
0.03	1.12
0.07	1.57
0.27	1.35

$q_n L_k =$

0.138	-0.0273

3.34	0
0	2.54

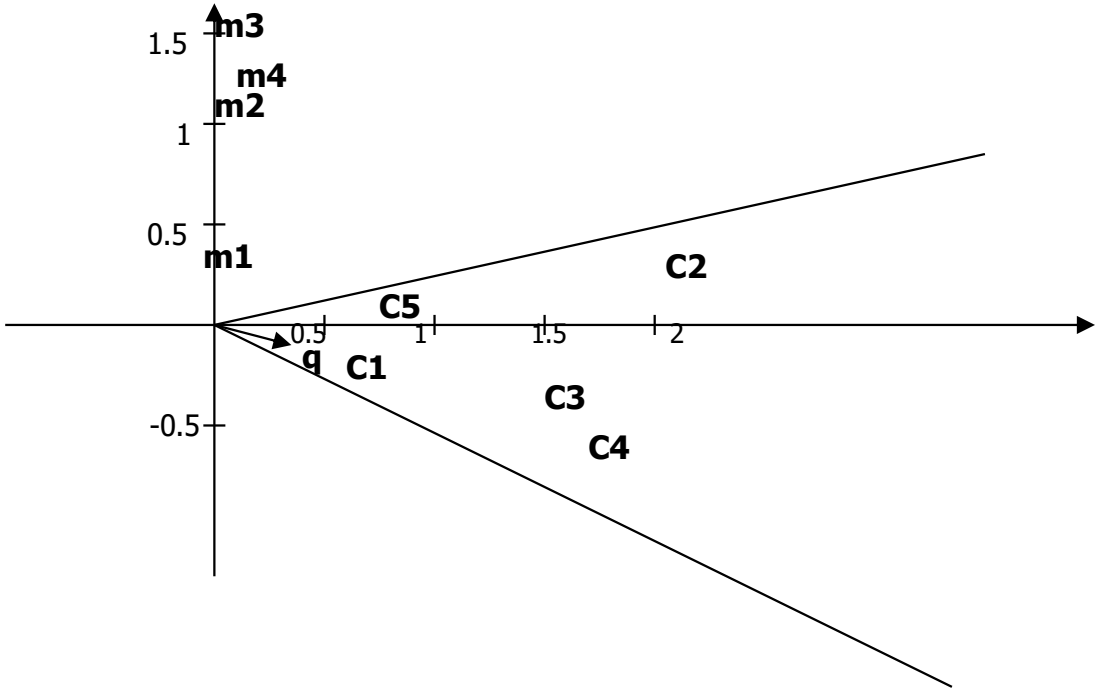
=

0.46	-0.069

Query transformation

- The cosine similarity matrix of query vector to the documents is:

	query
C1	0.99
C2	0.94
C3	0.99
C4	0.99
C5	0.90
M1	-0.14
M2	-0.13
M3	-0.11
M4	0.05



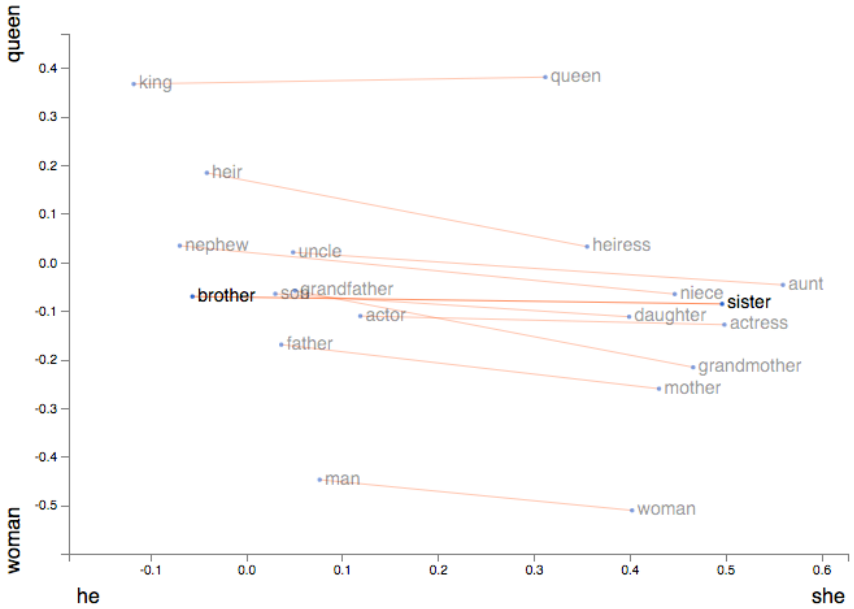
SVD problems

- The dimensions of the matrix change when dictionary changes
- The whole decomposition must be re-calculated when we add a word
- Sensitive to the imbalance of word frequency
- Very high dimensional matrix
- Not suitable for large corpora or dictionaries
- Quadratic cost to perform SVD
- Solution: Directly calculate a low-dimensional representation

Word analogy

- Words with similar meaning end up close to each other
- Words sharing similar contexts may be analogous
  - Synonyms
  - Antonyms
  - Names
  - Colors
  - Places
  - Interchangeable words

- Vector arithmetics to work with analogies
- i.e. **king - man + woman = queen**



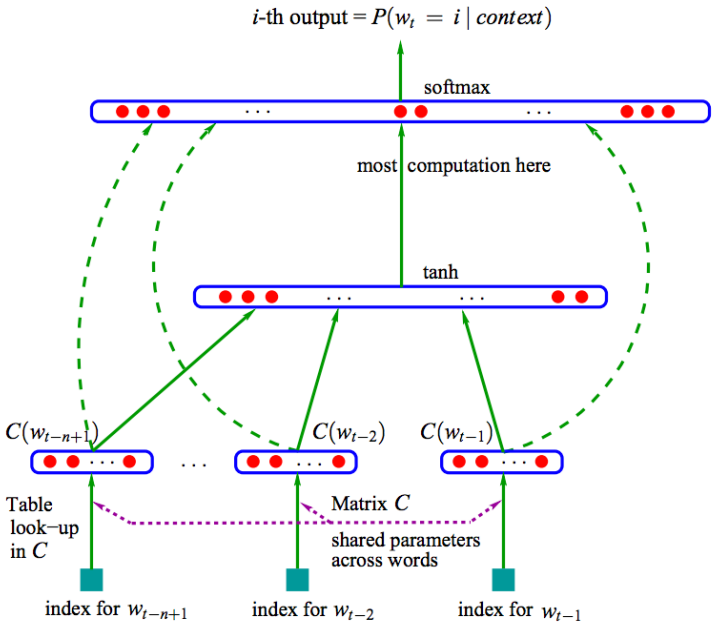
https://lamyiowce.github.io/word2viz/

Learning Word Vectors

- Corpus containing a sequence of T training words
- Objective: $f(w_t, \dots, w_{t-n+1}) = \hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$
- Decomposed in two parts:

$$w_i \in V \xrightarrow{\text{mapping C}} \mathbb{R}^m$$

  - Mapping **C** (1-hotv => lower dimensions)
  - Mapping any **g** s.t. (estimate prob t+1| t previous)
    $f(w_{t-1}, \cdots, w_{t-n+1}) = g(C(w_{t-1}), \cdots, C(w_{t-n+1}))$
- C(i) is the i-th word feature vector (Word embedding)
- Objective function: $J = \frac{1}{T} \sum f(w_t, \dots, w_{t-n+1})$

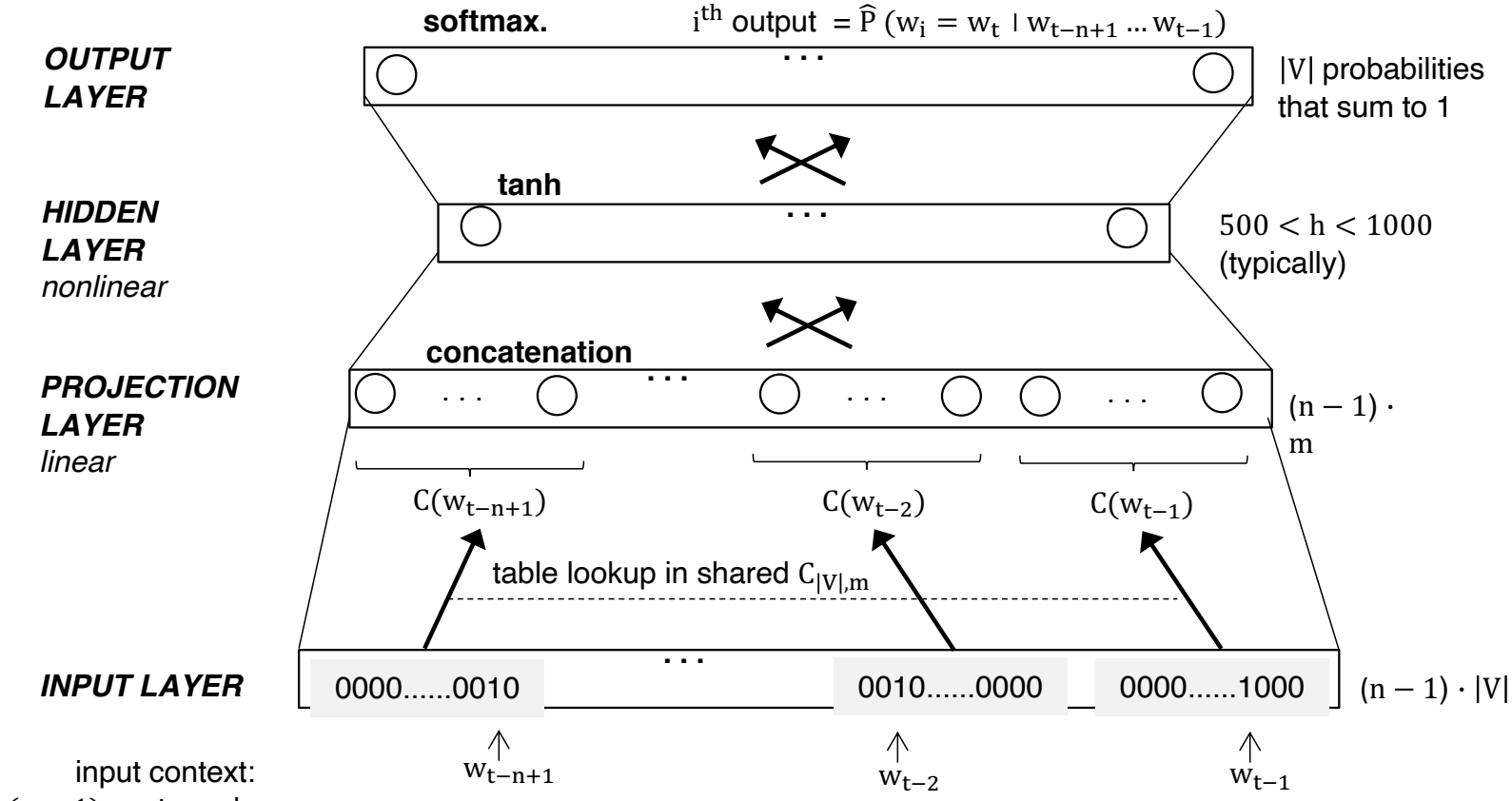


Bengio, Yoshua, et al. "A neural probabilistic language model." *The Journal of Machine Learning Research* 3 (2003): 1137-1155.

Neural Net Language Model

For each training sequence: input = (context, target) pair: $(w_{t-n+1} \ldots w_{t-1}, w_t)$

objective: minimize $E = -\log \hat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$



Objective function

- $E = -\log \widehat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$
- a probability between 0 and 1.
- On this support, the log is negative => –log term positive.
- makes sense to try to minimize it. ~
  - Probability of word given the context be as high as possible (1 for a perfect prediction).
    - case the error is equal to 0 (global minimum).

p	log(p)	-log(p)
0,7	-0,15490196	0,15490196
0,2	-0,698970004	0,698970004

NNLM facts

- tested on Brown (1.2M words, $|V| \cong 16K$) and AP News (14M words, $|V| \cong$ 150K reduced to 18K) corpuses
- Brown: $h = 100, n = 5, m = 30$
- AP News: $h = 60, n = 6, m = 100$, **3 week** training using **40 cores**
- 24% and 8% relative improvement (resp.) over traditional smoothed n-gram LMs
- in terms of test *set perplexity*: geometric average

$$1/\widehat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$$

- Due to **complexity**, NNLM can't be applied to large data sets → poor performance on rare words
- Bengio et al. (2003) initially thought their main contribution was a more accurate LM. They let the interpretation and use of the word vectors as **future work**

- On the opposite, Mikolov et al. (2013) focus on the **word vectors**

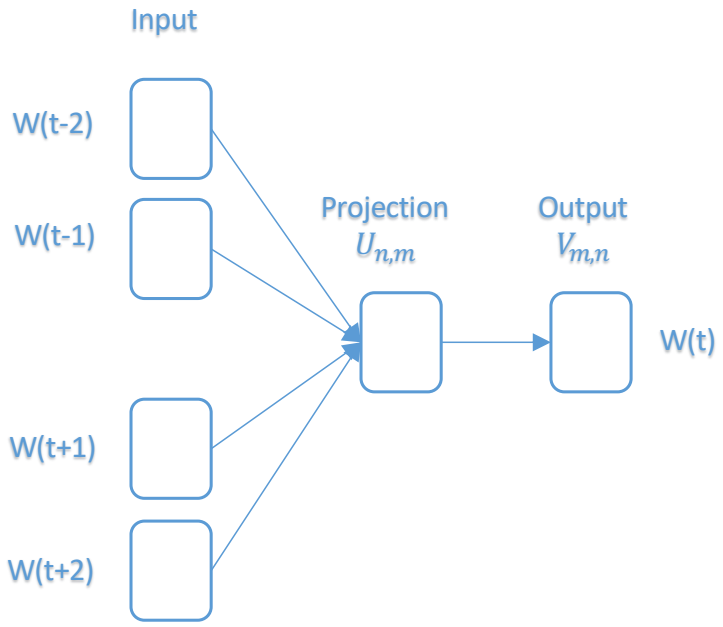
Word2Vec

- Mikolov et al. in 2013
- Word2vec key idea: achieve better performance not by using a more complex model (i.e., with more layers), but by allowing a **simpler (shallower) model** to be trained on **much larger amounts of data**
- no hidden layer (leads to 1000X speedup)
- projection layer is shared (not just the weight matrix) - C
- context: words from both history & future:
- Two algorithms for learning words vectors:
  - **CBOW**: from context predict target
  - **Skip-gram**: from target predict context

W2V: Continuous Bag Of Words – CBOW

- An unsupervised technique to learn word embeddings.
- CBOW learns the embeddings by predicting the target word (the center one) based on the context words (surrounding words).

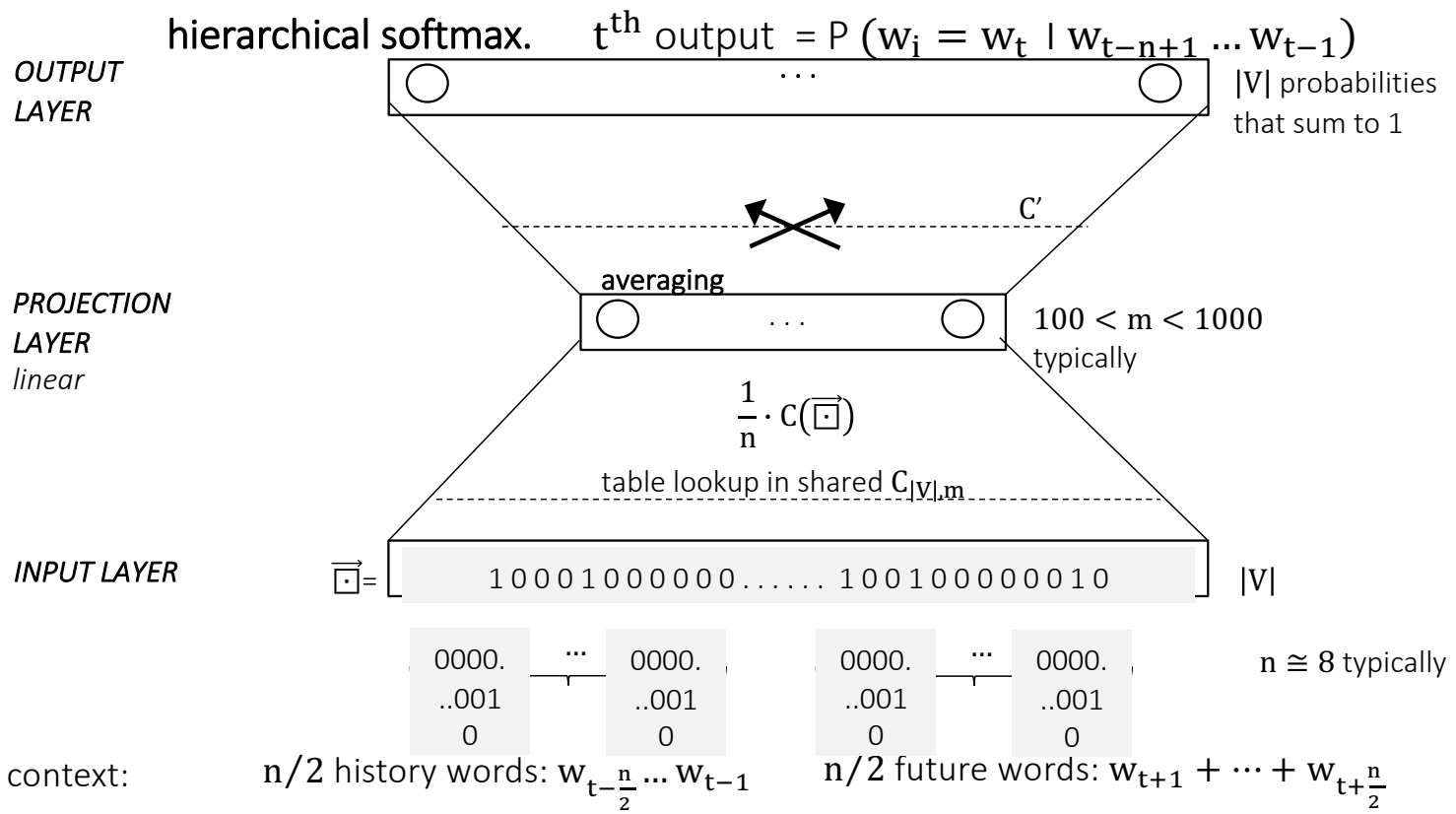
- i.e. “Paris is the capital of France located in Ile de France”



Continuous Bag-of-Words (CBOW)

For each training sequence: input = (context, target) pair: $(w_{t-\frac{n}{2}} \dots w_{t-1} w_{t+1} \dots w_{t+\frac{n}{2}}, w_t)$

objective: minimize $-\log \hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$



W2V: Continuous Bag Of Words – CBOW : Forward

- Each word **W(t)** is represented by one-hot vector of size **n** (vocabulary size).
- The **w** context words are averaged and forwarded to the projection layer to produce an embedded vector **z** of size **m:**

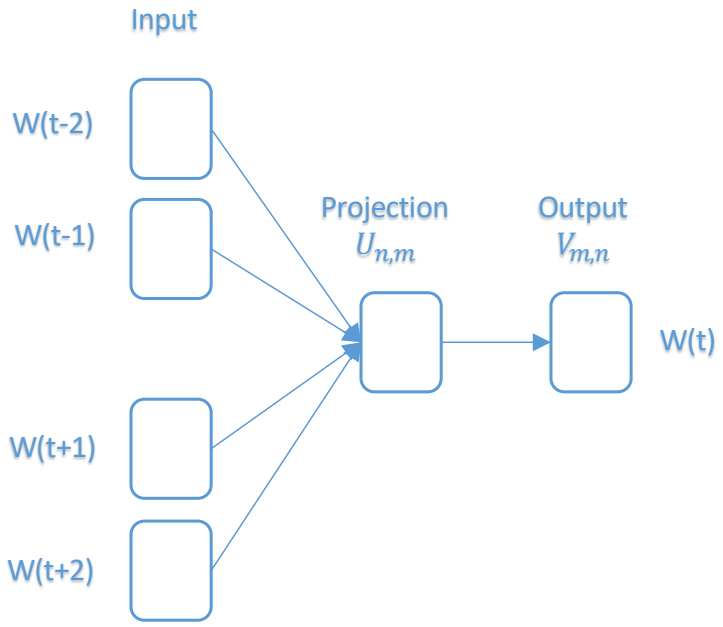
$$z_{1,m} = \frac{1}{w} \sum_{W \epsilon context} W_{1,n} U_{n,m}$$

- The vector **z** is forwarded to an output projection layer that produce the out vector **y** of size **n**.

$$y_{1,n} = z_{1,m} V_{m,n}$$

- Finally, a soft-max activation function is applied to the output to find a vector of probabilities for each word:

$$\sigma(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

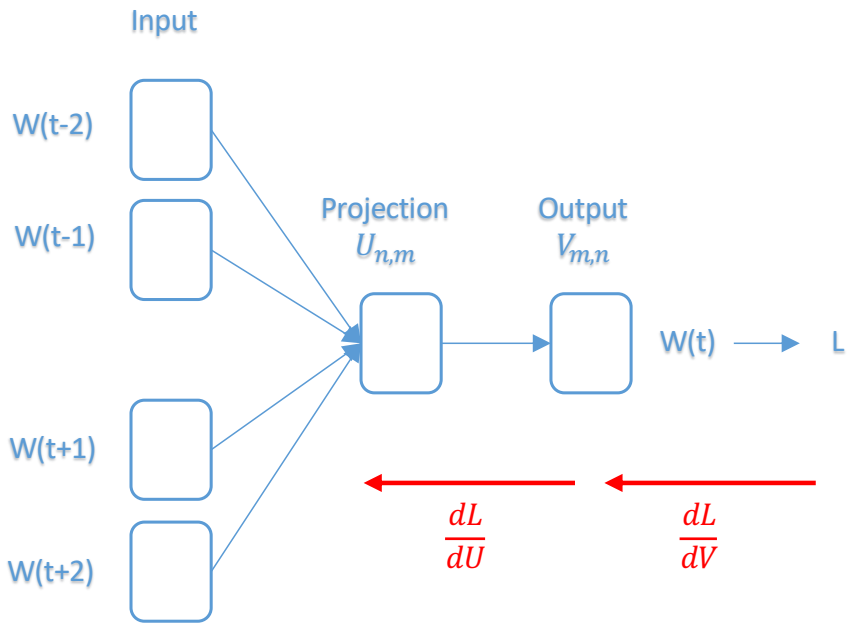


W2V: Continuous Bag Of Words – CBOW : Backward

- To update the weights, first we compute the log loss function (cross-entropy):

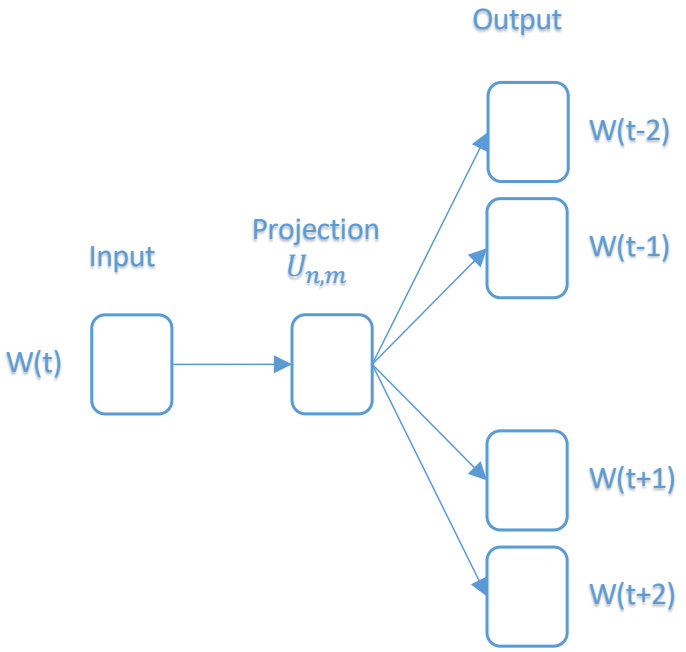
$$L = -\frac{1}{n}\sum_{i=1}^{n} W(t)_i \log(\sigma(y_i))$$

- The weights (matrices U and V) are now updated using gradient descent with learning rate α.
- $V = V - \alpha\frac{dL}{dV}$
- $U = U - \alpha\frac{dL}{dU}$
- Finally, after multiple passes through the corpus, **U** is the final **Word Embeddings Matrix** where each row represent a vector of size **m** for a specific word.



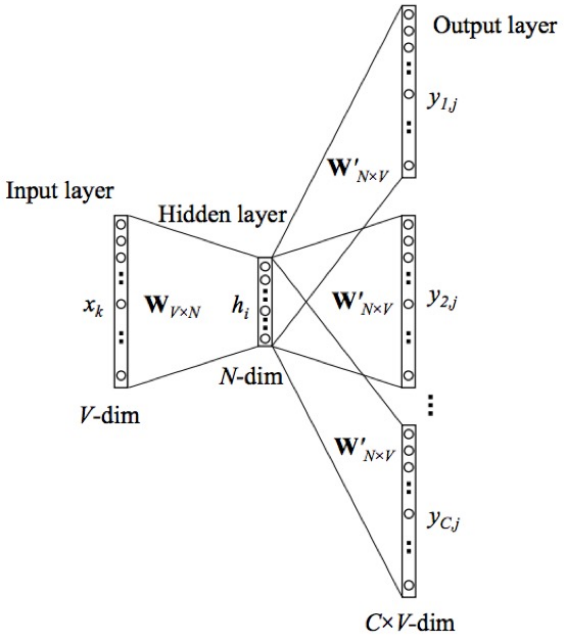
W2V: Skip-Gram

- An unsupervised technique to learn word embeddings.
- Skip-gram learn the embeddings by predicting the context of the word.
- The used loss function is cross-entropy as in CBOW.



W2V: Skip-gram

- skip-gram uses the context's center word to predict the surrounding words
- i.e. "Paris is the capital of France located in Ile de France"
- instead of computing the probability of the target word w_t given its previous words, we calculate the probability of the surrounding word w_{t+j} given w_t
- $p(w_{t+j}|w_t) = \frac{\exp(\boldsymbol{v}_{\boldsymbol{w_t}}^{\boldsymbol{T}} \boldsymbol{v}'_{\boldsymbol{w_{t+j}}})}{\sum_{w \in V} \exp(\boldsymbol{v}_{\boldsymbol{w_t}}^{\boldsymbol{T}} \boldsymbol{v}'_{\boldsymbol{w_{t+j}}})}$
- v^T_{wt} is a column of W_{VxN} and $v'_{w_{t+j}}$ is a column of W'_{NxV}
- Objective function $J = \frac{1}{T} \sum_{t=1}^{T} \sum_{-n \le j \le n} \log p(w_{t+j}|w_t)$



Efficient Estimation of Word Representations in Vector Space- Mikolov et al. 2013

Word2vec facts

- Complexity is $\mathbf{n} * \mathbf{m} + \mathbf{m} * \mathbf{log}|\mathbf{V}|$ (Mikolov et al. 2013a)
- **n**:size of context window (~10) **nxm**: dimensions of the projection layer, **|V|** size of the vocabulary
- On Google news 6B words training corpus, with $|\mathbf{V}| \sim 10^6$:
  - CBOW with $\mathrm{m} = 1000$ took **2 days** to train on **140 cores**
  - Skip-gram with $\mathrm{m} = 1000$ took **2.5 days** on **125 cores**
  - NNLM (Bengio et al. 2003) took **14 days** on **180 cores**, for $\mathrm{m} = 100$ only!

word2vec training speed ≅ 100K-5M words/s

- Quality of the word vectors:
  - ↗ significantly with **amount of training data** and **dimension of the word vectors** (m)
  - measured in terms of accuracy on 20K semantic and syntactic association tasks.

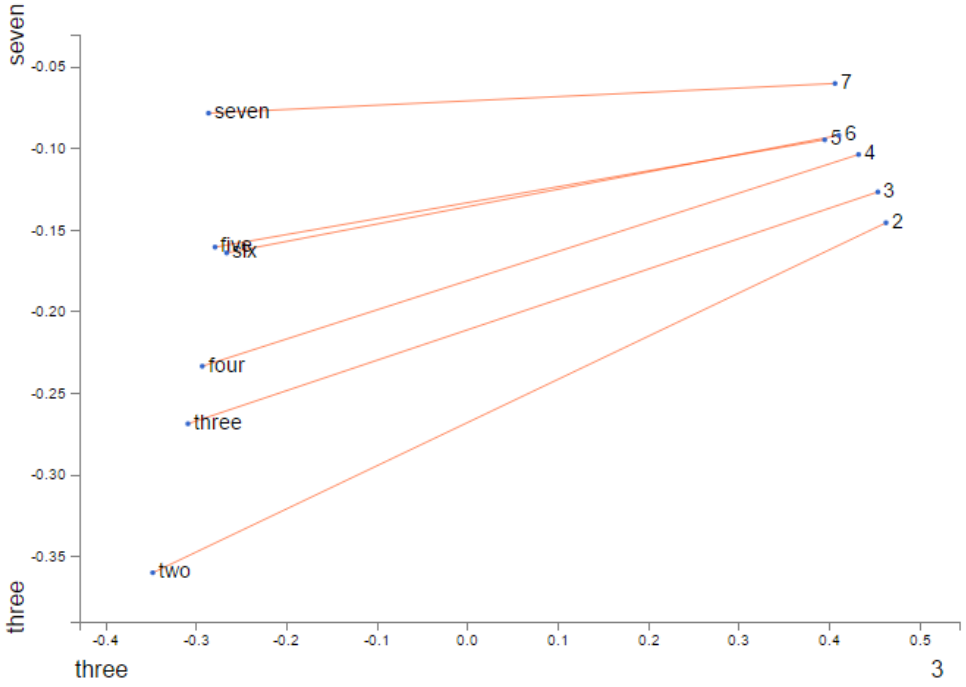
    e.g., words in **bold** have to be returned:

Capital-Country	Past tense	Superlative	Male-Female	Opposite
Athens: **Greece**	walking: **walked**	easy: **easiest**	brother: **sister**	ethical: **unethical**

- Best NNLM: 12.3% overall accuracy. Word2vec (with Skip-gram): 53.3%
- References: http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd ---- https://code.google.com/p/word2vec/

Applications

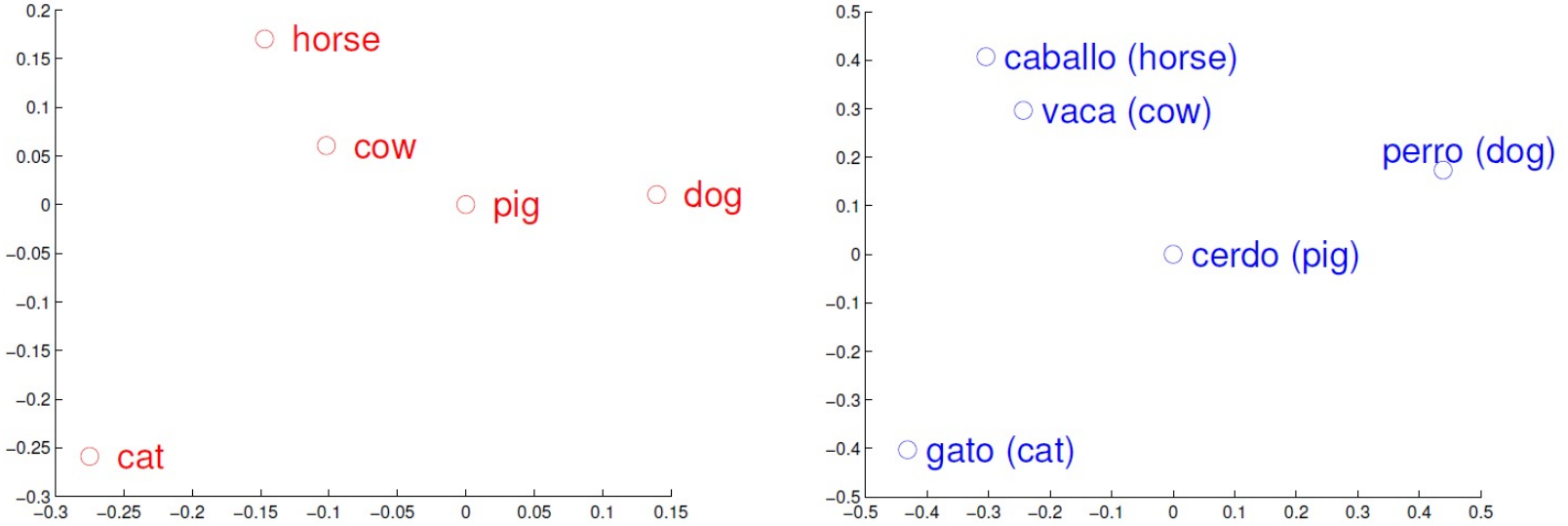
- Word analogies
- Find similar words
  - Semantic similarity
  - Syntactic similarity
- POS tagging
- Similar analogies for different languages
- Document classification



https://lamyiowce.github.io/word2viz/

Applications

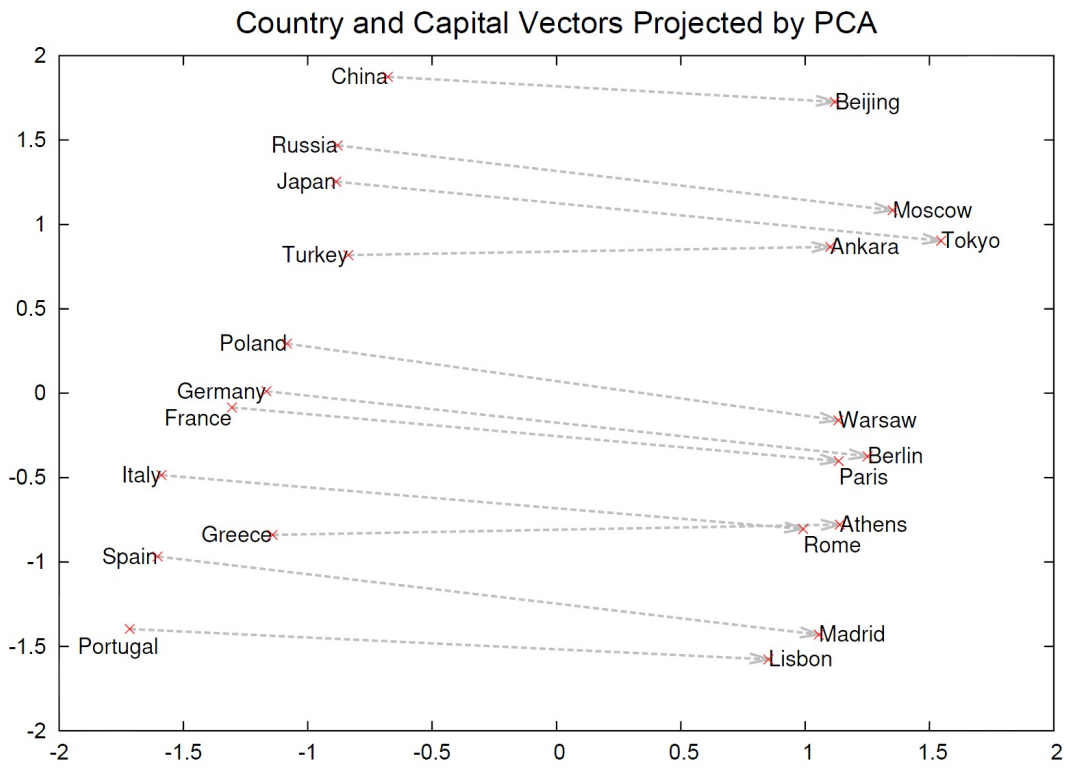
- High quality word vectors boost performance of all NLP tasks, including document classification, machine translation, information retrieval…
- Example for English to Spanish machine translation:



About 90% reported accuracy (Mikolov et al. 2013c)

Mikolov, T., Le, Q. V., & Sutskever, I. (2013). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

Remarkable properties of word vectors

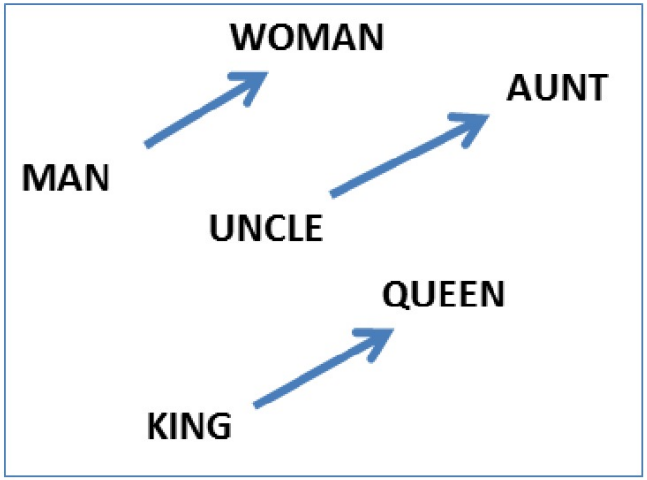


regularities between words are encoded in the difference vectors

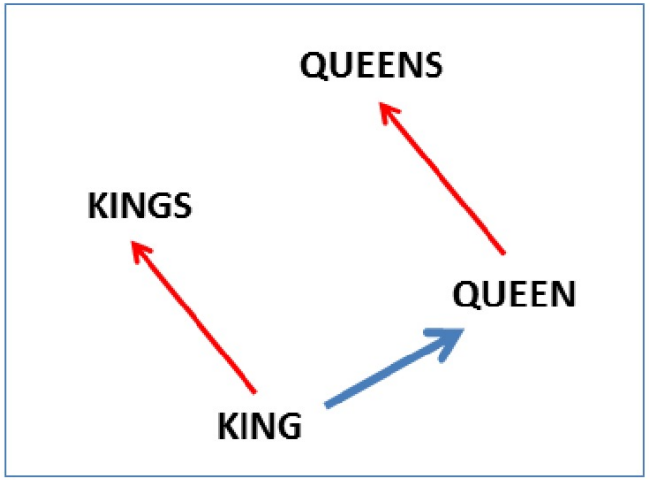
e.g., there is a constant **country-capital** difference vector

Mikolov et al. (2013b)
Distributed representations of words and phrases and their compositionality

Remarkable properties of word vectors



constant **male-female** difference vector



constant **singular-plural** difference vector

- Vector operations are supported and make intuitive sense:

$$w_{king} - w_{man} + w_{woman} \cong w_{queen}$$

$$w_{einstein} - w_{scientist} + w_{painter} \cong w_{picasso}$$

$$w_{paris} - w_{france} + w_{italy} \cong w_{rome}$$

$$w_{his} - w_{he} + w_{she} \cong w_{her}$$

$$w_{windows} - w_{microsoft} + w_{google} \cong w_{android}$$

$$w_{cu} - w_{copper} + w_{gold} \cong w_{au}$$

- Online demo (scroll down to end of tutorial)

http://rare-technologies.com/word2vec-tutorial/

OUTLINE

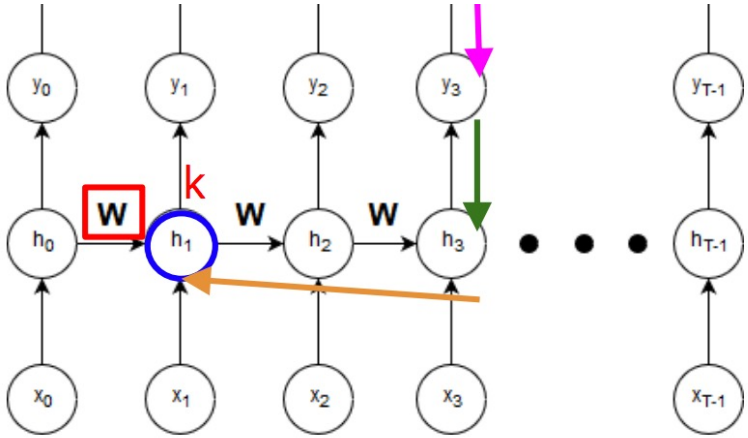
- Representation Learning for Text
  - SVD - Word2Vec
  - Document Representations
- **Attention - Transformer Architecture**

Recurrent Neural Networks for Sequential tasks

- CNNs are good at image classification
  - Input: an image
  - Output: probability of the class
    - $p(Red\ Panda|\ image) = 0,9$
    - $p(Cat\ |\ image) = 0,1$

- Sequence learning: study of machine learning algorithms designed for sequential data (time series, language…)
  - Translate: "*machine learning is a challenging topic*" to French: "*L'apprentissage automatique est un sujet difficile*"
  - Predict the next word: "*John visited Paris, the capital of ….* "
  - Input and output strongly correlated within the sequence.

Learning in RNNs – backpropagation in time



$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

OUTLINE

- **Representation Learning for Text**
  - SVD - Word2Vec
  - Document Representations
- CNNs for text classification
- **Attention - Transformer Architecture**

Attention is ubiquitous in DL today

Image captioning

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

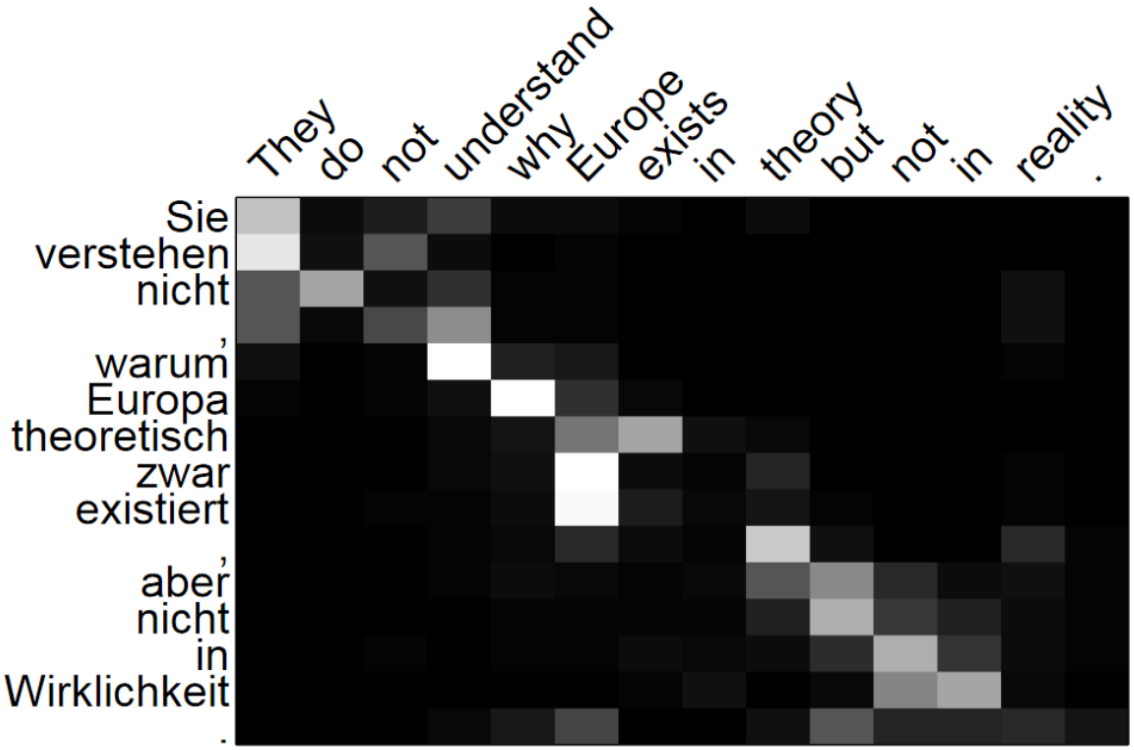
A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

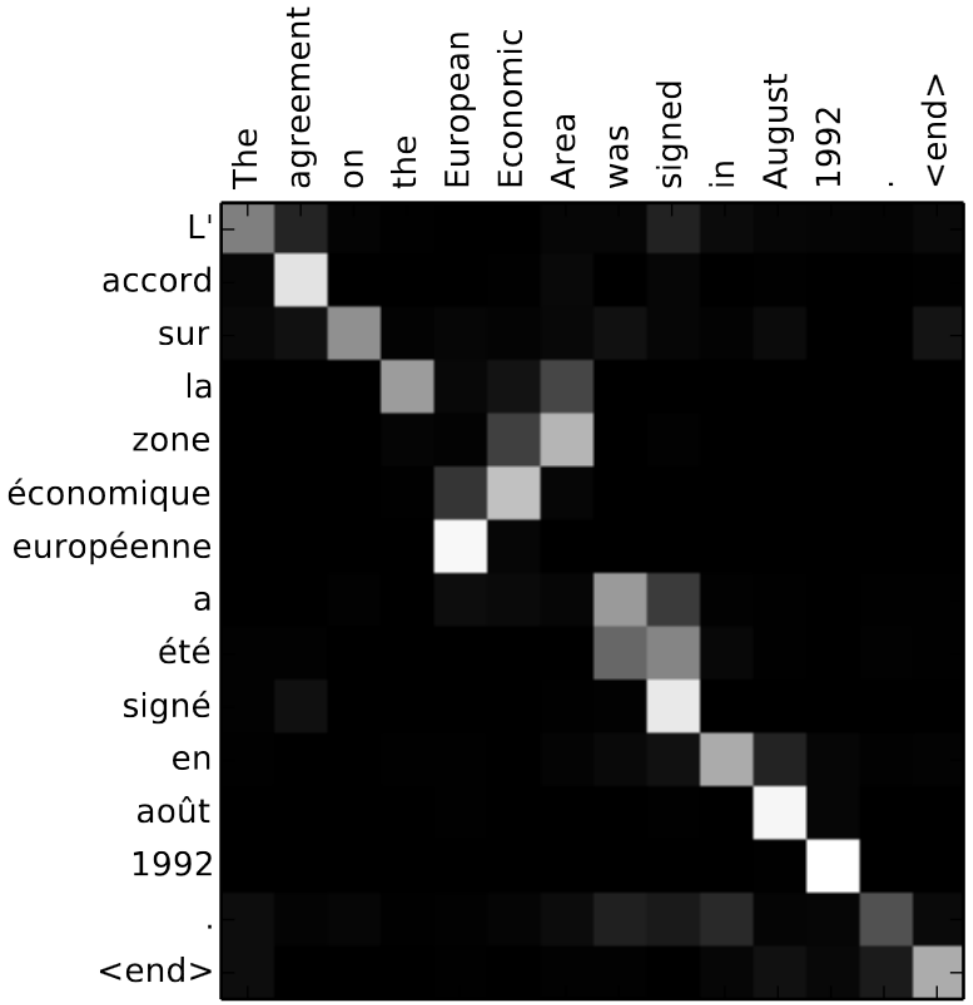
Show, attend and tell: Neural image caption generation with visual attention (Xu et al. 2015)

Attention is ubiquitous in DL today

Neural Machine Translation



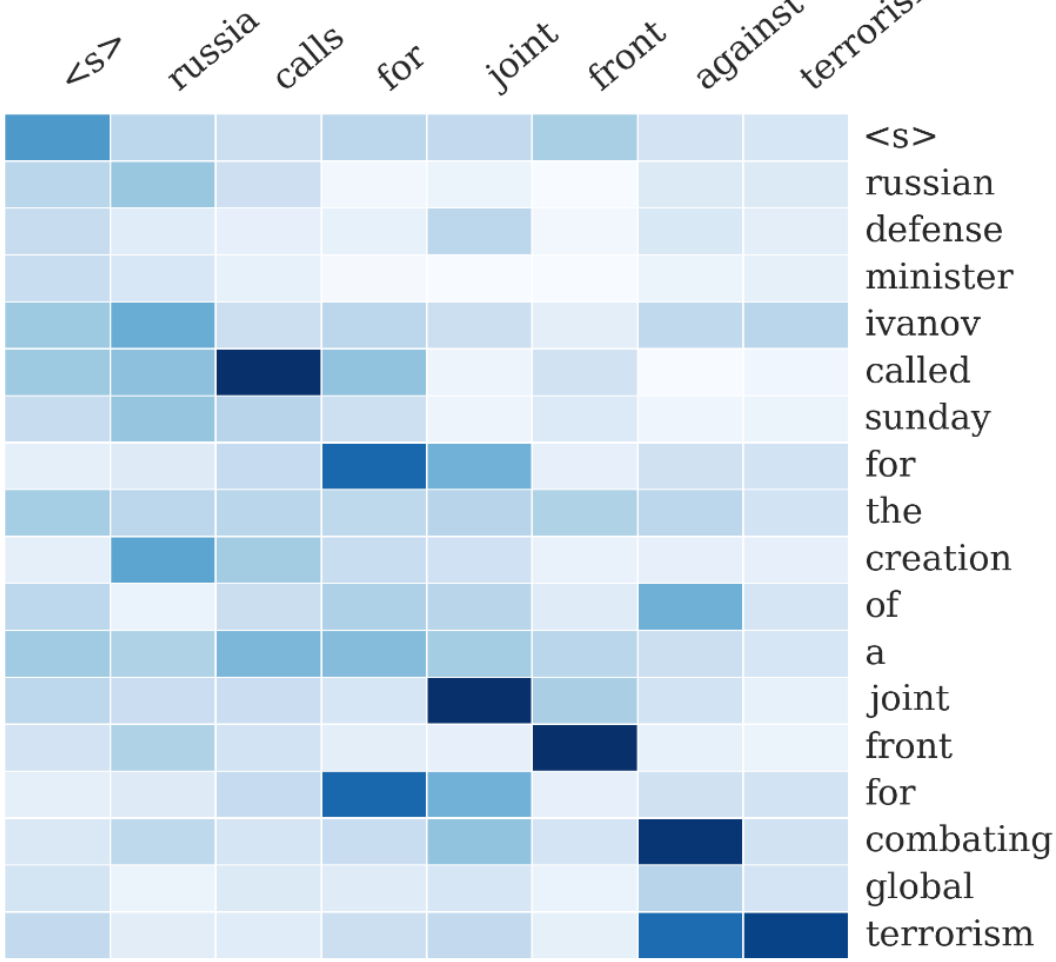
Effective approaches to attention-based neural machine translation (Luong et al. 2015)



Neural machine translation by jointly learning to align and translate (Bahdanau et al. 2014)

Attention is ubiquitous in DL today

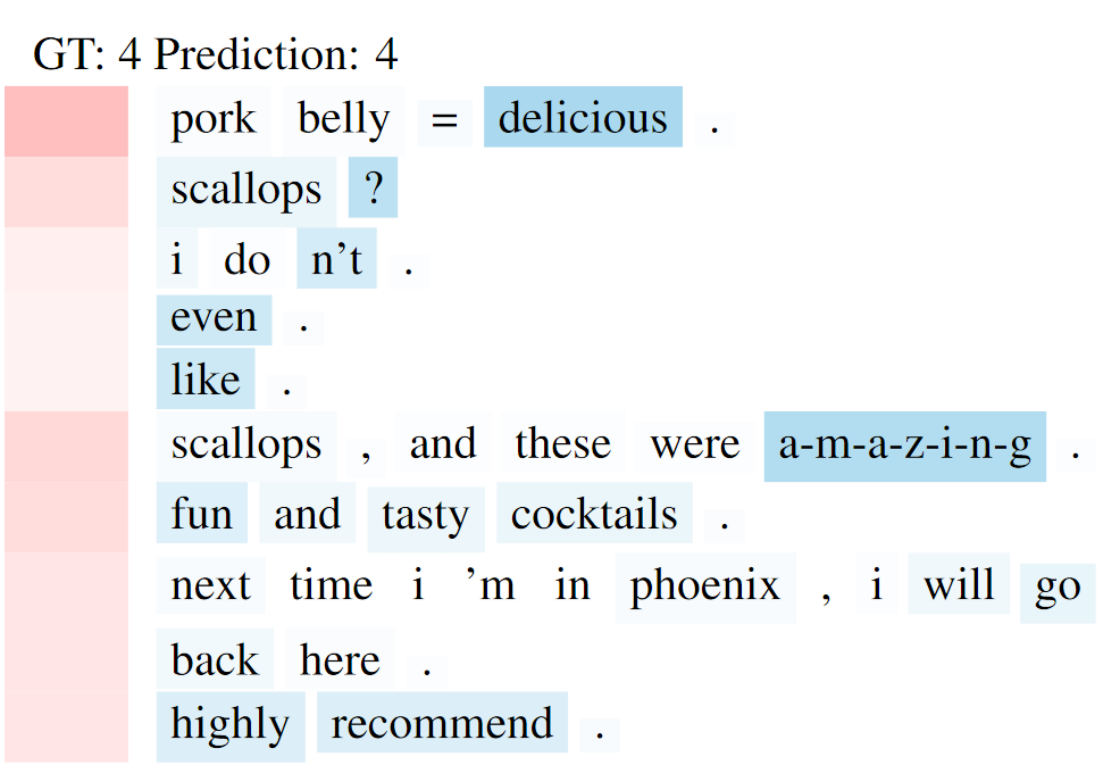
Abstractive summarization

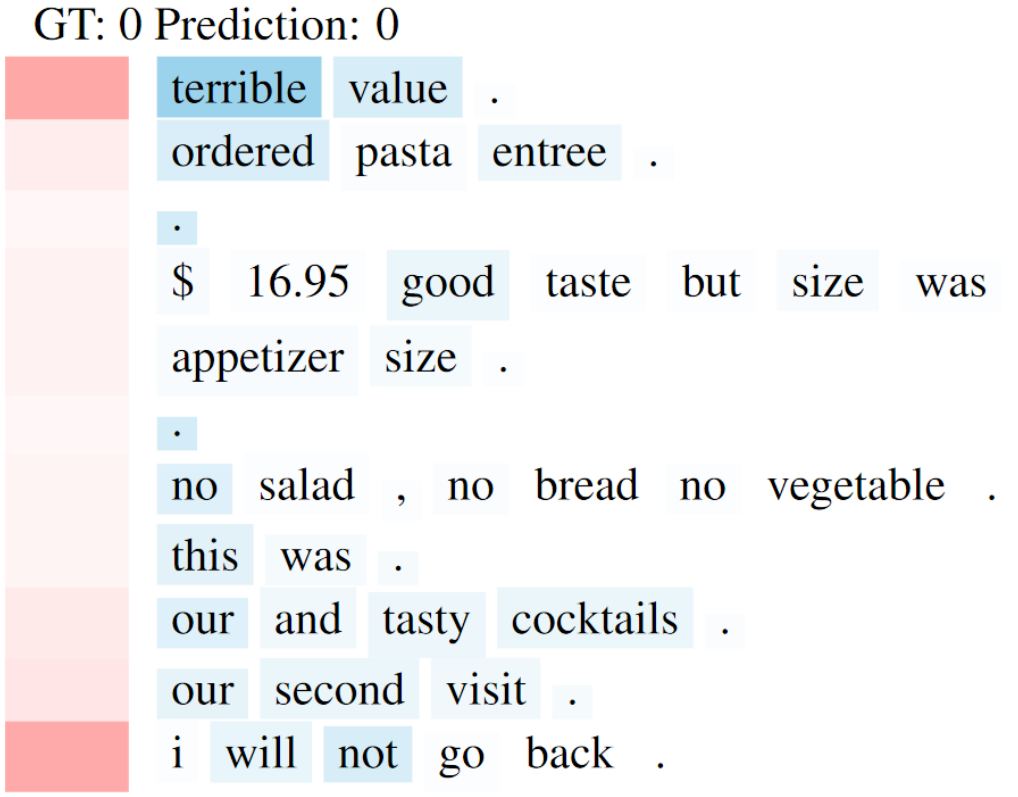


A neural attention model for abstractive sentence summarization (Rush et al. 2015)

Attention is ubiquitous in DL today

Sentiment analysis





Hierarchical attention networks for document classification (Yang et al. 2016)

Encoder-Decoder Architectures

General idea:

encoder **decoder**

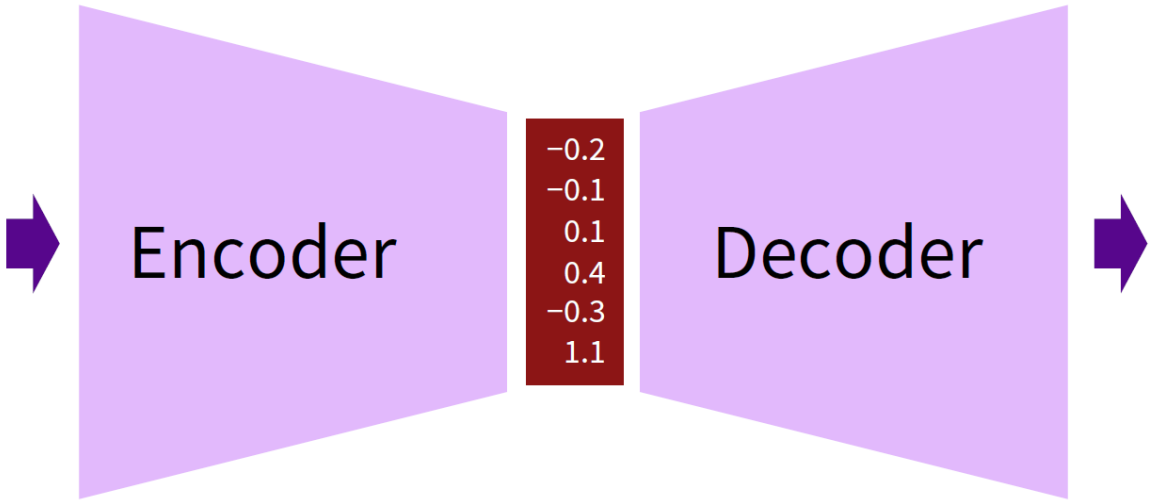
Input -> embedding -> Output

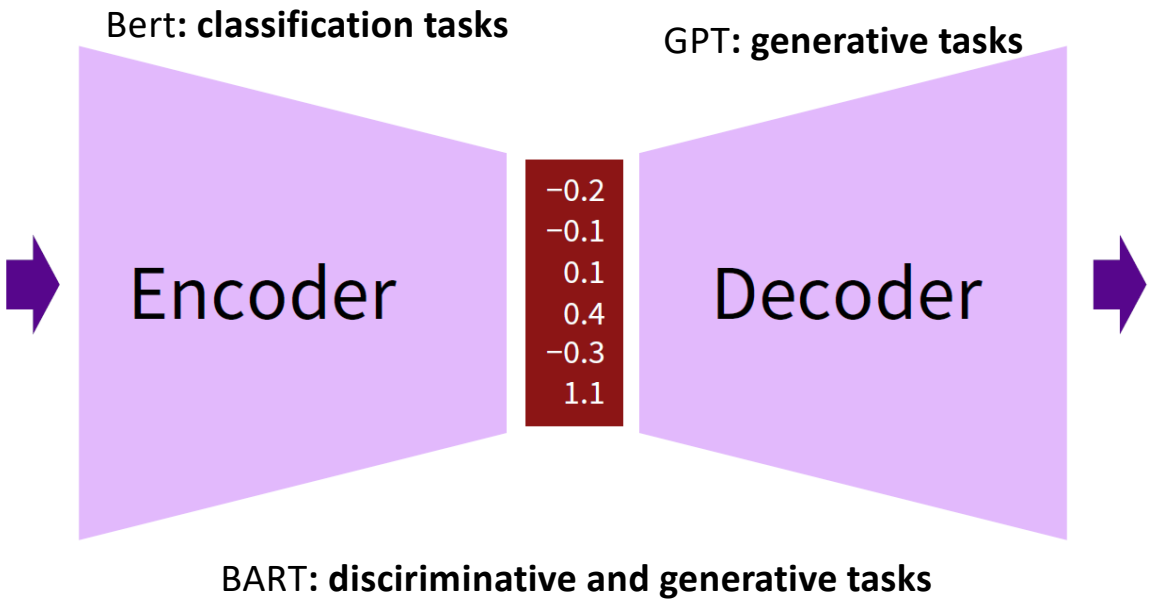
- image
- document
- source language
- …

- caption
- summary
- target language
- …

Known as *sequence-to-sequence* (seq2seq) when input and output are sequences (e.g., NLP applications)

The full architecture is differentiable => end-to-end training




Bert: classification tasks
GPT: generative tasks
Encoder
−0.2
−0.1
0.1
0.4
−0.3
1.1
Decoder
BART: disciriminative and generative tasks

What is attention?

Objective

- Traditional models (e.g., encoder) having to embed the input into a single fixed-length vector (lossy).
- Alternatively information can be kept and stored into multiple vectors.
- information can be retrieved later on (e.g., by the decoder). (Bahdanau et al. 2014)

Quick history:

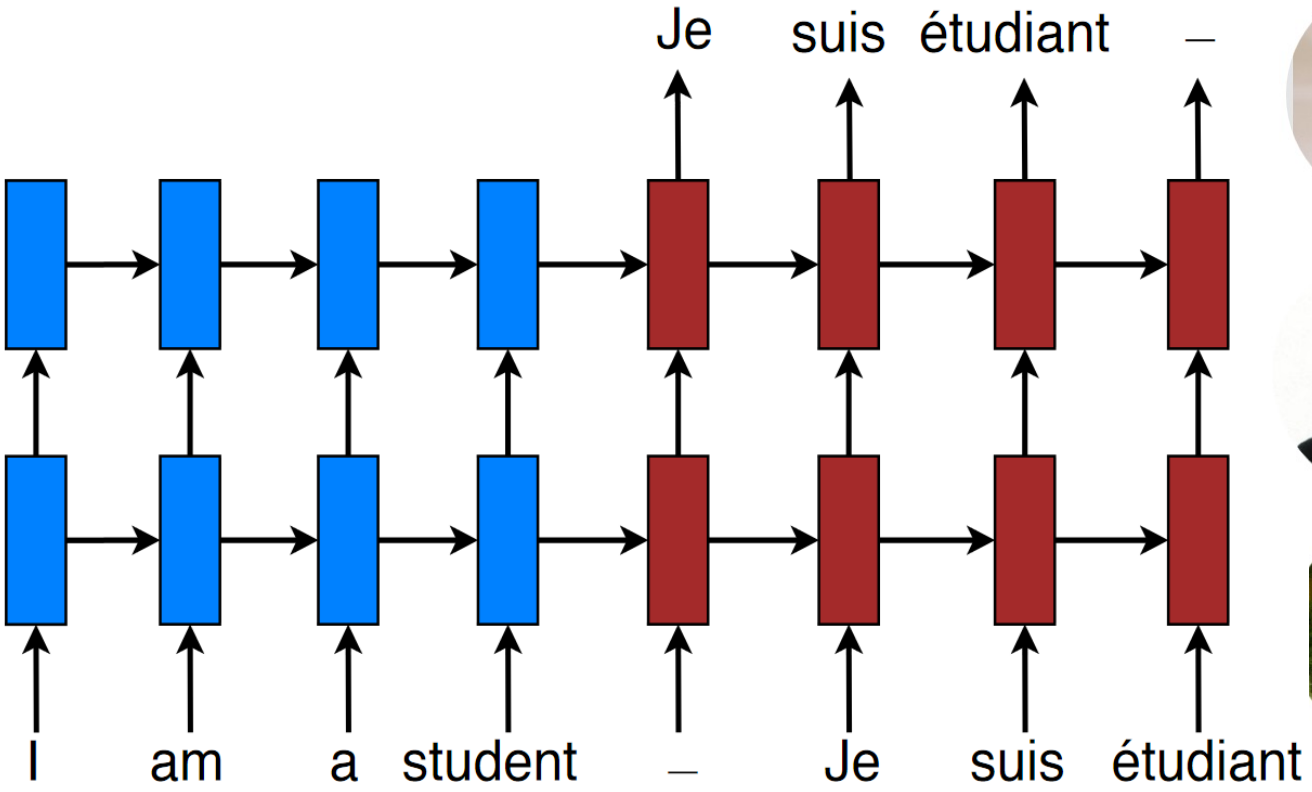
- developed in the context of **encoder-decoder** architectures for neural machine translation (Bahdanau et al. 2014)
- rapidly applied to naturally related tasks like image captioning (Xu et al. 2015) and summarization (*Luong et al. 2015*)
- also proposed for encoders only, e.g. for text classification (Yang et al. 2015) and representation learning (Conneau et al. 2017).

  Known as *self or inner attention* in such cases.

Encoder-Decoder for Neural Machine Translation

Encoder **Decoder**

Target *sentence (generated)*



Source *sentence (input)*

Minh-Thang Luong

Research Scientist at Google
Verified email at google.com - Homepage
Deep Learning Natural Language Processing

Hieu Pham

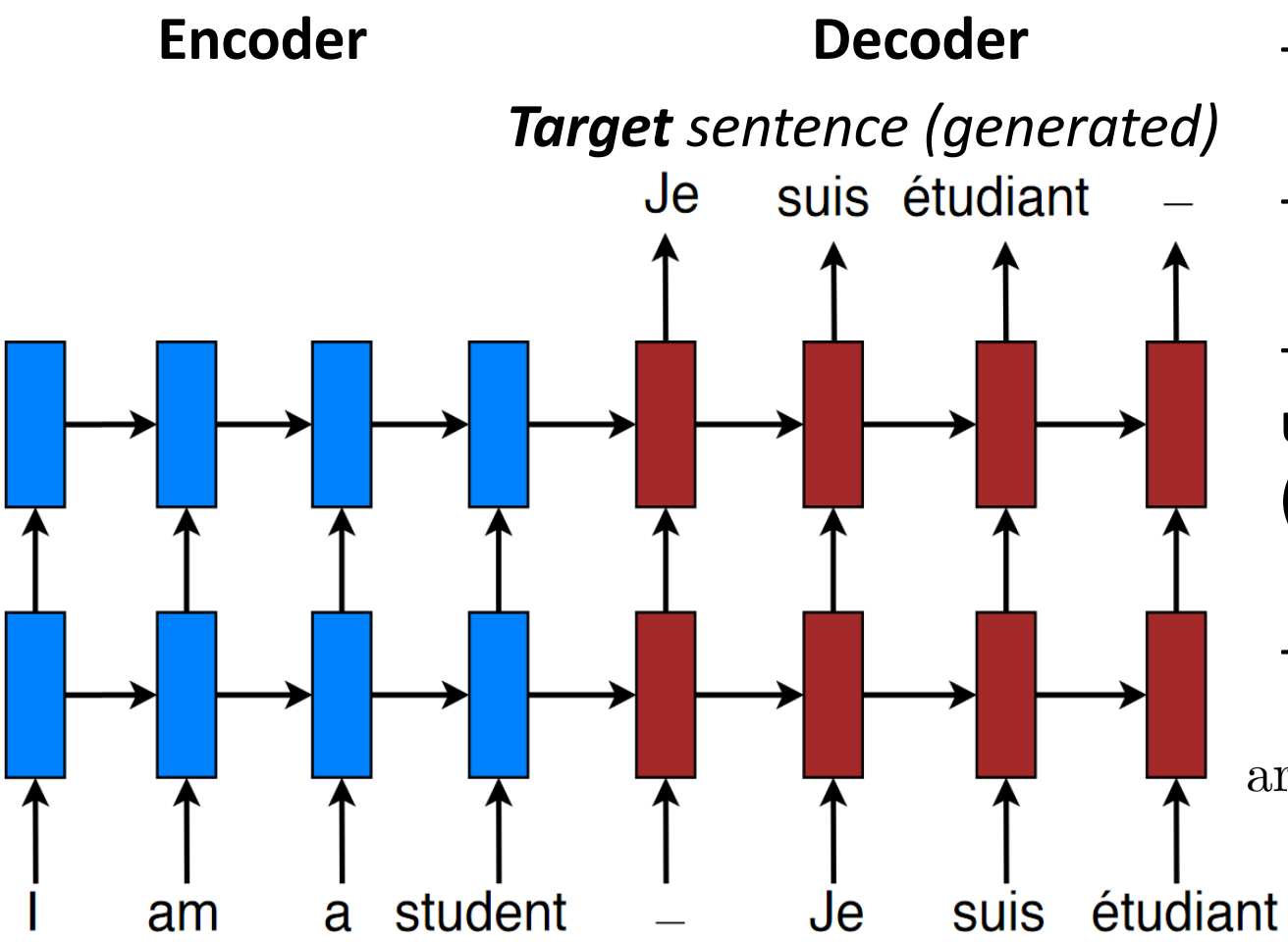
Carnegie Mellon University, Google Brain
Verified email at google.com
Machine Learning

Christopher D Manning

Professor of Computer Science and Linguistics
Verified email at stanford.edu - Homepage
Natural Language Processing

Effective approaches to Attention-Based Neural Machine Translation (2015)

Encoder-Decoder for Neural Machine Translation



- source: $\left(x_1, \ldots, x_{T_x}\right)$

- target: $\left(y_1, \ldots, y_{T_y}\right)$

- Both encoder & decoder are unidirectional deep RNNs (a.k.a. *stacking* RNNs)

- Training objective:

$$\text{argmax}_\theta \left\{ \sum_{(x,y)\in \text{corpus}} \log\, p(y|x;\theta) \right\}$$

Luong et al. (2015)

Encoder-Decoder for Neural Machine Translation

Encoder: usually: CNN, stacking RNN* with LSTM or GRU units…
* unidirectional (Luong et al. 2015) or
bidirectional (Bahdanau et al. 2014).

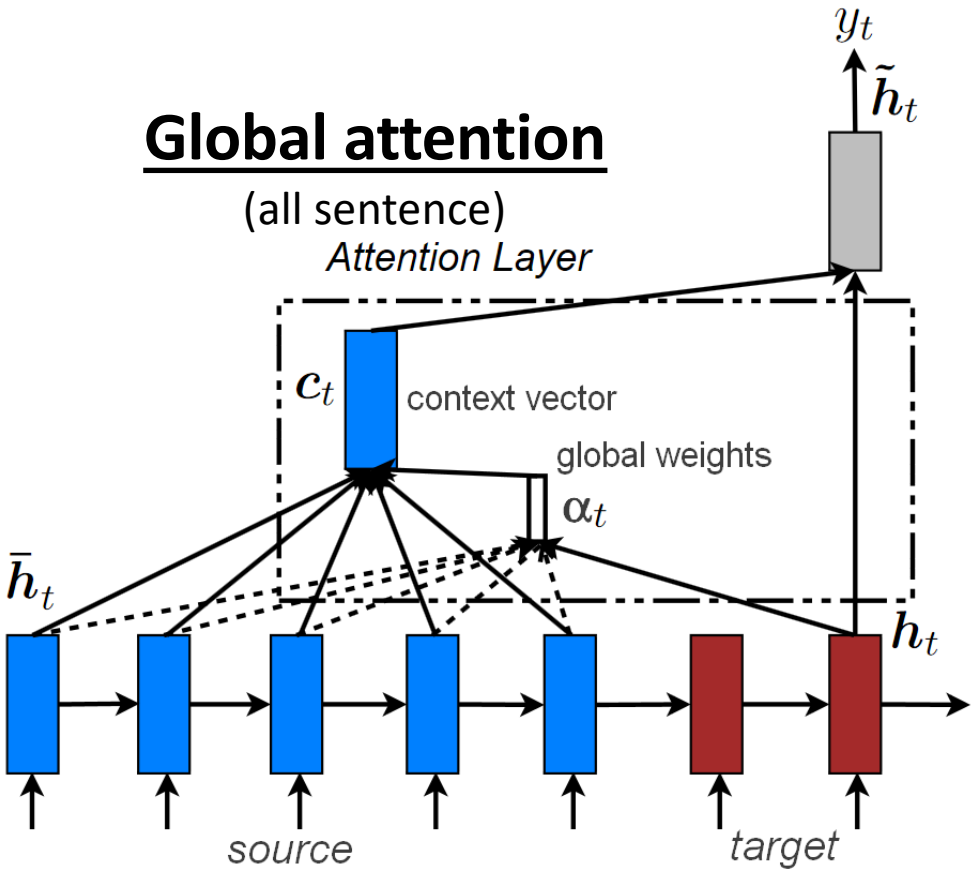
Decoder: unidirectional *RNN* (well suited to text generation), best if deep.

Generates the target sentence $\left(y_1, \cdots, y_{T_y}\right)$ one word at a time:

$$P\left[y_t | \{y_1, ..., y_{t-1}\}, c_t\right] = \text{softmax}\left(W_s \tilde{h}_t\right)$$

Luong et al. (2015)

Encoder-**Decoder** for Neural Machine Translation



$$P\left[y_t|\{y_1,...,y_{t-1}\},c_t\right] = \text{softmax}\left(W_s\tilde{h}_t\right)$$

attentional hidden state

$$\tilde{h}_t = \tanh\left(W_c\left[c_t;h_t\right]\right)$$

context vector

decoder hidden state

$$c_t = \sum_{i=1}^{T_x} \alpha_{t,i}\bar{h}_i$$

i^{th} encoder hidden state

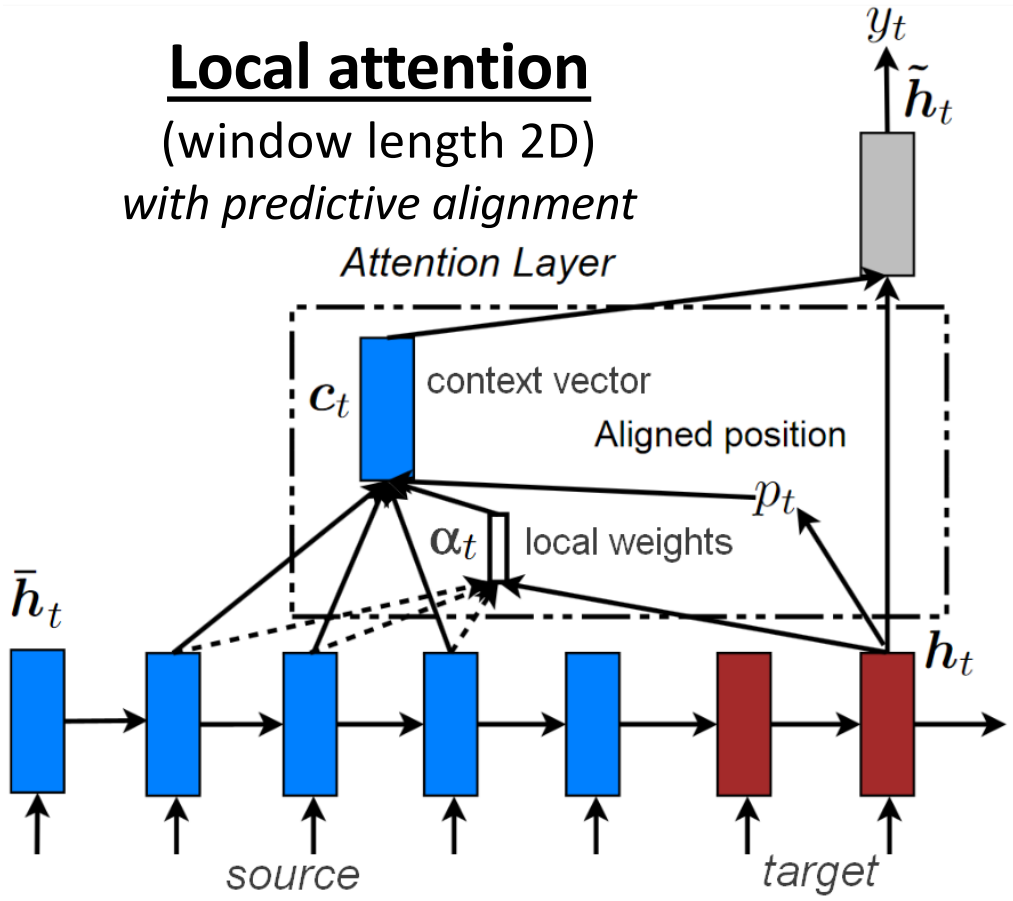
alignment vector

$$\alpha_{t,i} = \frac{\exp\left(\text{score}\left(h_t,\bar{h}_i\right)\right)}{\sum_{i'=1}^{T_x}\exp\left(\text{score}\left(h_t,\bar{h}_{i'}\right)\right)}$$

$$\text{score}\left(h_t,\bar{h}_i\right) = h_t^\top\bar{h}_i$$

Luong et al. (2015)

Encoder-**Decoder** for Neural Machine Translation



$$P\left[y_t|\{y_1,...,y_{t-1}\},c_t\right] = \text{softmax}\left(W_s\tilde{h}_t\right)$$

attentional hidden state

$$\tilde{h}_t = \tanh\left(W_c\left[c_t; h_t\right]\right)$$

decoder hidden state

$$p_t = T_x \cdot \sigma\left(v_p^\top \tanh\left(W_p h_t\right)\right)$$

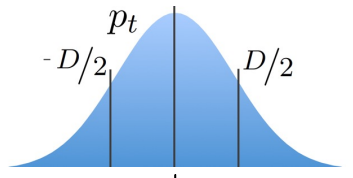
context vector

$$c_t = \sum_{i=p_t-D}^{p_t+D} \alpha_{t,i}\bar{h}_i$$

i^{th} encoder hidden state

alignment vector

$$\alpha_{t,i} = \frac{\exp\left(\text{score}(h_t,\bar{h}_i)\right)}{\sum_{i'=p_t-D}^{p_t+D}\exp\left(\text{score}(h_t,\bar{h}_{i'})\right)}\exp\left(-\frac{(i-p_t)^2}{2(D/2)^2}\right)$$



$$\text{score}(h_t,\bar{h}_i) = h_t^\top W_\alpha \bar{h}_i$$

Luong et al. (2015)

Encoder-Decoder for Neural Machine Translation

Facts:

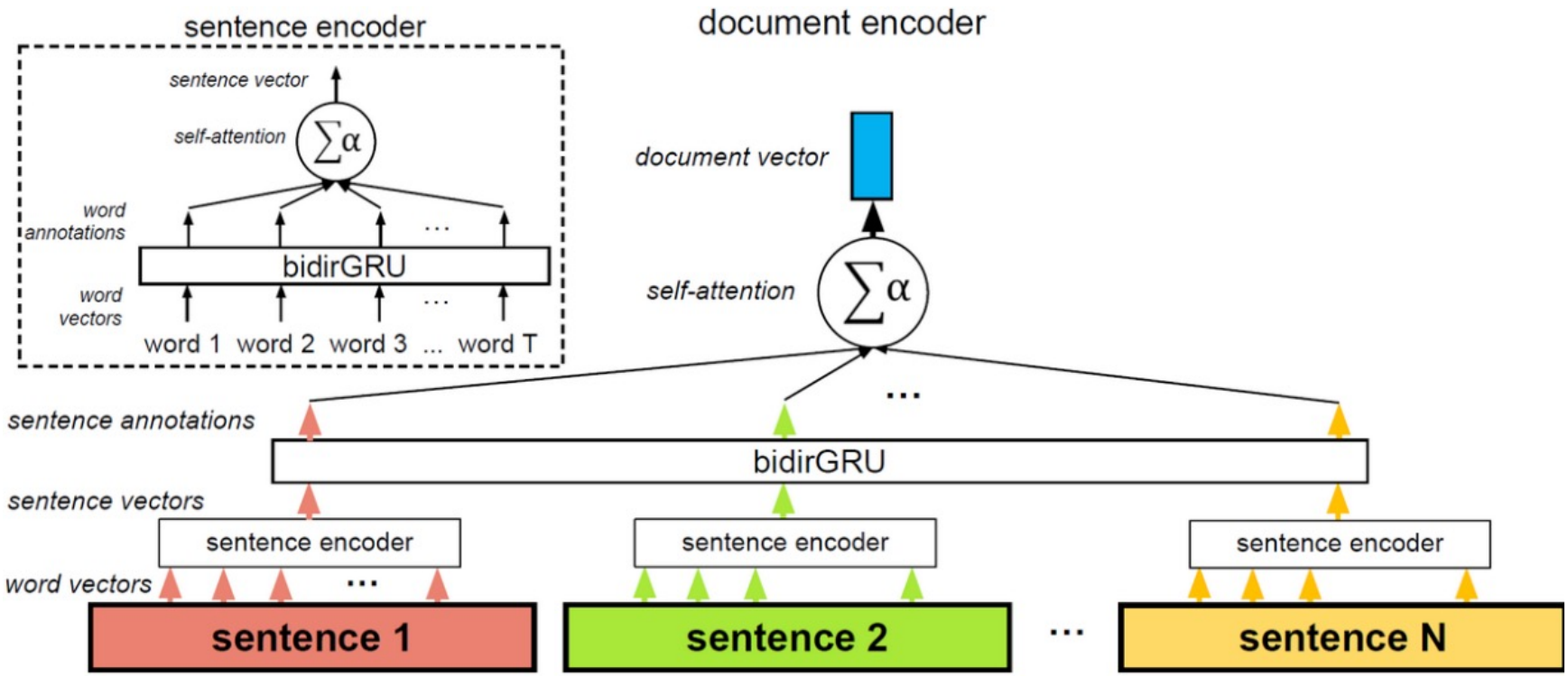
- Tested on the English <-> German task (WMT'14 dataset)
- 4.5M sentence pairs
- Encoder and decoder RNNs feature 4 layers of stacking and 1000-dimensional hidden states
- Window of size *D=10* for local attention
- Trained for 12 epochs (total of 7-10 days on a single GPU, at 1K target words/s)
- New state-of-the-art performance

Lessons learned:

- Local attention with predictive alignment gives better results than global attention
- Dot product ($\text{score}(h_t, \bar{h}_i) = h_t^\top \bar{h}_i$) works well for global attention
- The general formulation ($\text{score}(h_t, \bar{h}_i) = h_t^\top W_\alpha \bar{h}_i$) is better for local attention

Luong et al. (2015)

Self-attention for RNN encoders



Self-attention for RNN encoders

- Input is a sentence $(x_1, \ldots, x_T)$
- We're only interested in getting an embedding s of the sentence for some downstream task (e.g., classification)

encoder hidden state

context vector

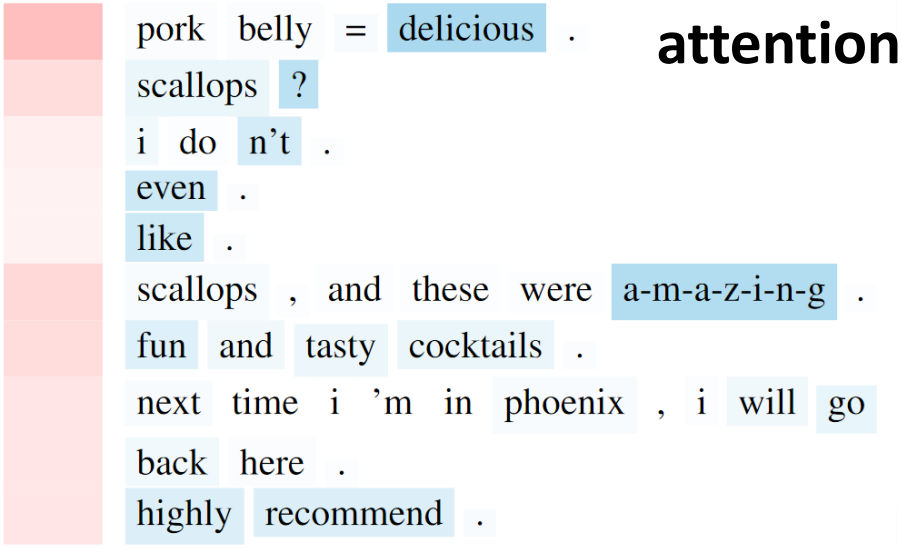
$$u_t = \tanh(W h_t)$$

$$\alpha_t = \frac{\exp(\text{score}(u_t, u))}{\sum_{t'=1}^{T} \exp(\text{score}(u_{t'}, u))}$$

$$s = \sum_{t=1}^{T} \alpha_t h_t$$

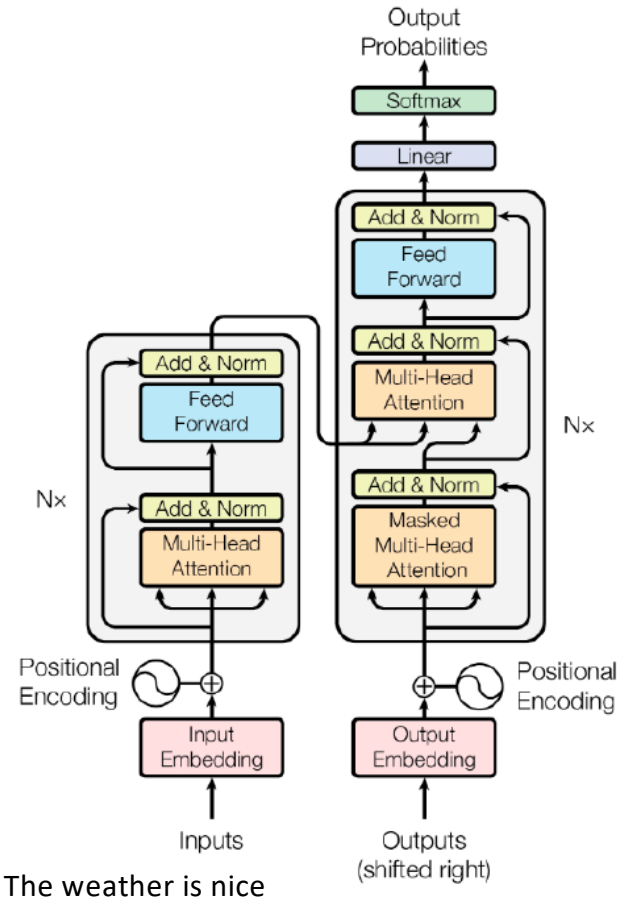
$$\text{Where score}(u_t, u) = u_t^\top u$$

The same process can be repeated over the sentence vectors s -> **hierarchical attention**



Transformer [1]

- A model that follows the encoder-decoder structure, where the input tokens are mapped to a sequence of continuous representations, and these representations are consumed by the decoder which generates a sequence of outputs.
- Does not use recurrent neural networks or convolutional neural networks. Instead, it only uses dense and attention layers.



[1]Attention is all you needA Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, AN Gomez, Advances in neural information processing systems 30, 2019. 93K citations as of 10/2023

Attention

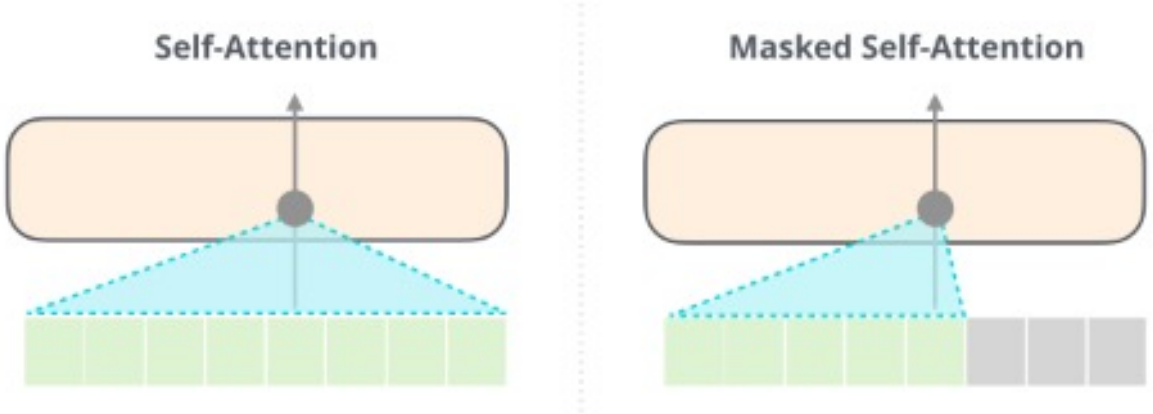


Figure: Self-attention of Encoder vs. Decoder.[1]

- Encoder-Decoder Attention (source-target alignment)

1) https://jalammar.github.io/illustrated-gpt2/

Transformer Architecture

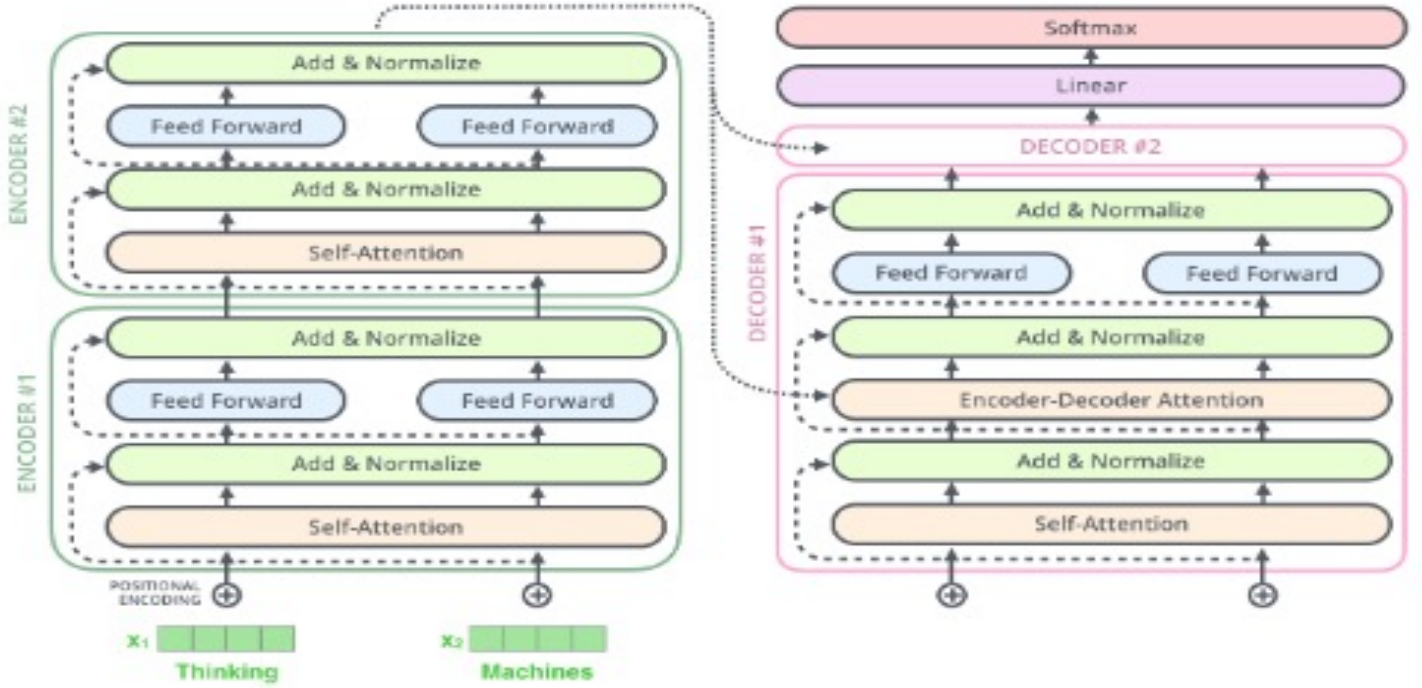


Figure: Transformer architecture.[1,2]

1) https://jalammar.github.io/illustrated-transformer/
2) http://nlp.seas.harvard.edu/annotated-transformer/

Postitional encoding

$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}})$$
$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}})$$

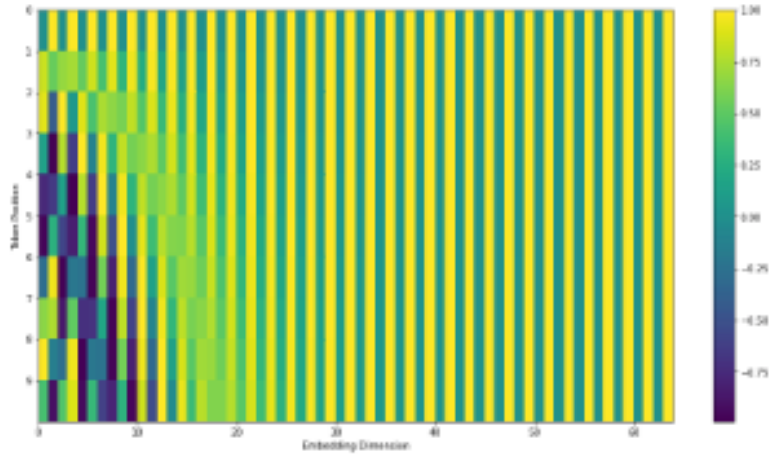


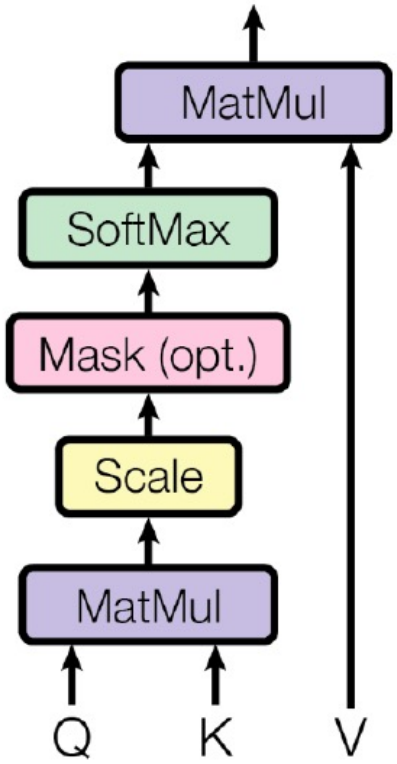
Figure: Sinusoid positional encoding.[1]

• RNN • Parallel computing • Learned positional embeddings

1) https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-

Transformer – Scaled dot product attention

- A query Q_i representing the token at position i attend to a sequence of tokens (represented by K): $Q_i K^T$.
- The result scaled by $\sqrt{d_k}$ (dimension of Q_i) is passed to a softmax function to compute a score for each of the tokens: $softmax(\frac{Q_i K^T}{\sqrt{d_k}})$
- A weighted sum of the Value vectors is calculated as the new representation of the token at position i: $softmax(\frac{Q_i K^T}{\sqrt{d_k}})V$
- Queries are stacked in one matrix Q, the output of the attention layer becomes: $softmax(\frac{QK^T}{\sqrt{d_k}})V$



Query - Key - Value

computations of vector representation of tokens “thinking” and “machines” IN CONTEXT via attention.

- W^Q, W^K, W^V are trainable matrices
- q,k,v vectors corresponding to the tokens in the context.
- For “thinking”: q1*k1 is the self similarity for this word which is normalized and softmaxed. The result multiplied by v_1.

- z1 = sum (q1, k1, v1) that represent “thinking” in the context of “thinking”).
- Similar process happens among “Thinining” and “Machines” resulting in the z2 vector that represents after all computations the vector that corresponds to “thinking” in the context of “machines”
- The W^* matrices represent attention weights among the tokens in the context.. ?

Zij representation of word i to the head j – then we MERGE and give them to the next layer.

- K, W^iV are intialised randomly and are trainable – i stand for the head
- X1 : [“Thinking” embedding: positional embedding of “Thinking”]
- q1 = X1 * W^Q, k1 = X1 * W^K, v1 = X1 * W^V
- qi*kj : attention among tokens i and j. Then normalized by softmax
- z1 = sum(v1j): representation of token 1 in context, j: words in context
- zi ‘s concatenated and passed to the next attention layers…

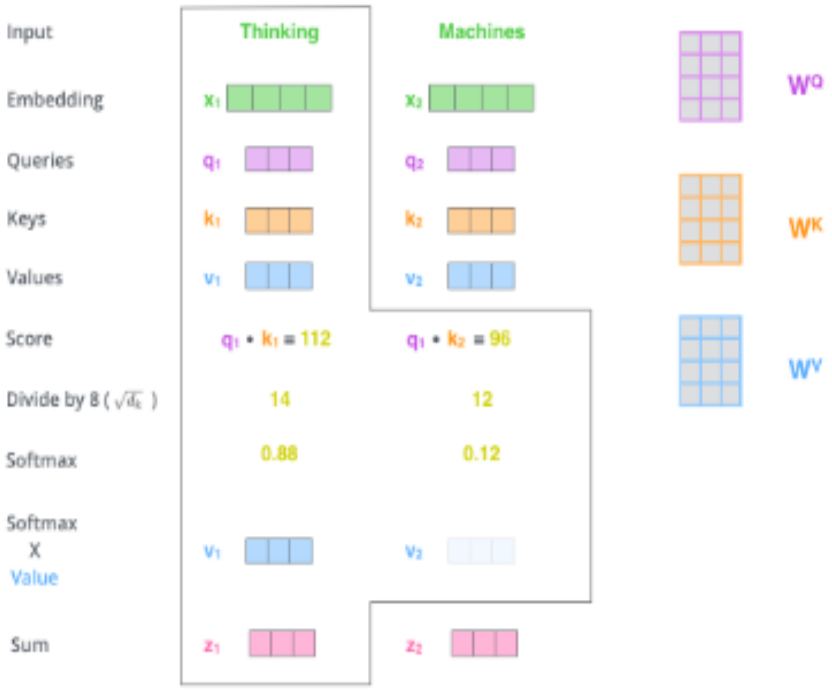


Figure: Self-attention calculation.

- Multi-Head Attention: $\{W_0^Q, W_0^K, W_0^V\}, \{W_1^Q, W_1^K, W_1^V\}, \ldots$

What is Attention?

- Attention allows a Transformer model to determine which words in a sentence are most relevant to each other.
- "The lion ate the antelope."

Example Sentence

- "The lion ate the antelope."
- compute attention for the word "ate".
- understand how the attention mechanism works.

Intuition Behind Q, K, and V

STEP1. Embedding the Words

- Each word represented by a vector (embedding):
  - The: [0.1, 0.2, 0.3]
  - lion: [0.4, 0.5, 0.6]
  - ate: [0.7, 0.8, 0.9] *(focus word)*
  - the: [1.0, 1.1, 1.2]
  - antelope: [1.3, 1.4, 1.5]

STEP2: Compute Queries, Keys, and Values:

- embeddings transformed into Q, K, and V using learnable weight matrices:
  - Q = Embedding × W_Q
  - K = Embedding × W_K
  - V = Embedding × W_V

- For simplicity, assume the transformations result in:
  - Q for "ate": [1.0, 2.0, 3.0]
  - K for each word:
    - The: [0.5, 1.0, 1.5]
    - lion: [1.0, 2.0, 3.0]
    - ate: [1.5, 3.0, 4.5]
    - the: [2.0, 4.0, 6.0]
    - antelope: [2.5, 5.0, 7.5]
  - V for each word:
    - The: [0.2, 0.4, 0.6]
    - lion: [0.4, 0.8, 1.2]
    - ate: [0.6, 1.2, 1.8]
    - the: [0.8, 1.6, 2.4]
    - antelope: [1.0, 2.0, 3.0]

Step 1: Assigning Q, K, and V

- **STEP 3. Compute Attention Scores:**
- - attention score between the **Query** ("ate") and each **Key**:
- - Score("ate", "The") = Q("ate") · K("The") = (1.0×0.5) + (2.0×1.0) + (3.0×1.5) = 7.0
- - Score("ate", "lion") = Q("ate") · K("lion") = (1.0×1.0) + (2.0×2.0) + (3.0×3.0) = 14.0
- - Score("ate", "ate") = Q("ate") · K("ate") = (1.0×1.5) + (2.0×3.0) + (3.0×4.5) = 21.0
- - Score("ate", "the") = Q("ate") · K("the") = (1.0×2.0) + (2.0×4.0) + (3.0×6.0) = 28.0
- - Score("ate", "antelope") = Q("ate")·K("antelope")= (1.0×2.5) + (2.0×5.0) + (3.0×7.5)=35.0
-
- **STEP4: Softmax the Scores**
- - Apply softmax function to convert scores into probabilities (attention weights):
- - Softmax([7.0, 14.0, 21.0, 28.0, 35.0]) ≈ [0.000, 0.000, 0.000, 0.000, **1.000**]
- - model assigns almost all attention to "antelope" as it has the highest score.

STEP 5: Weighted Sum of Values

- Multiply the attention weights by the corresponding Values and sum them up:

  - Z1 (ate) ≈ 0.000×V("The") + 0.000×V("lion") + 0.000×V("ate") + 0.000×V("the") + 1.000×V("antelope")

  - Z1 (ate)≈ [1.0, 2.0, 3.0]

Interpretation

- attention mechanism determined "antelope" is the most relevant word for understanding "ate" in this context.

- makes sense because "ate" and "antelope" closely related in the sentence (the lion ate the antelope).

- output is a weighted combination of the Values: weights determined by attention scores.

Key Takeaways

- - Query (Q) : What we're focusing on (e.g., "ate").
- - Key (K): What each word offers (e.g., "The", "lion", "ate", "the", "antelope").
- - Value (V): The actual content of each word.
- - attention mechanism computes ***how much each word should contribute to the representation of the current word*** (Query) based on their relationships.
- - process repeated for every word in the input sequence, allowing transformer to capture complex dependencies and relationships between words
- - model learned to focus on "antelope" when processing "ate": aligns with the semantic meaning of the sentence.

Example of self attention

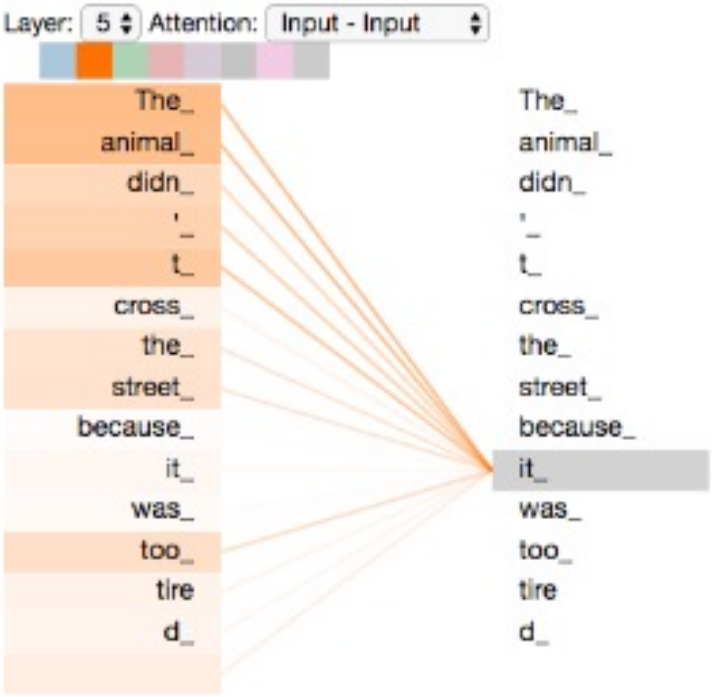


Figure: As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

Transformer - experiments

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost

Pretrained Language Models

- Pre-trained on unsupervised large corpora.
- Fine-tuned on supervised tasks, with an important gain in performance, especially in the case where the number of training examples is limited.

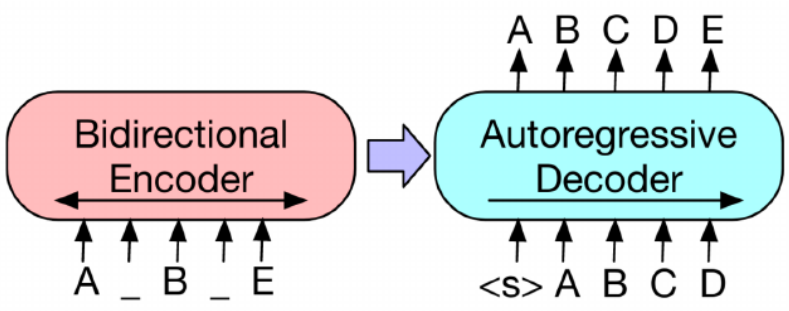
GPT

- Pre-trained auto-regressive Transformer decoder.
- Pre-training objective: maximize: $\sum_i \log P(u_i | u_{i-k}, ..., u_{i-1}; \theta)$

BERT

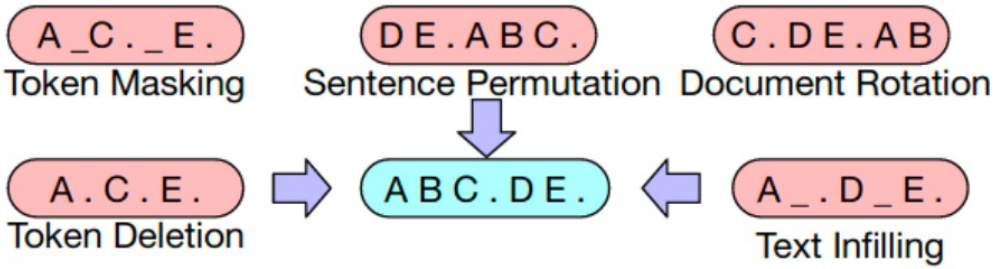
- Pre-trained stack of Transformer encoders.
- Two unsupervised pre-training tasks:
  - Masked language model.
  - Next sentence prediction.

Bart Overview



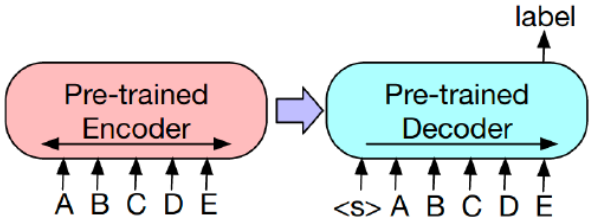
- A denoising autoencoder, that uses a bidirectional encoder and a left-to-right autoregressive decoder.
- Uses the standard Transformer architecture, with 12 encoder layers and 12 decoder layers.
- Pre-trained on 160GB of english text.
- Given its architecture, Bart can be used in both discriminative and generative tasks.

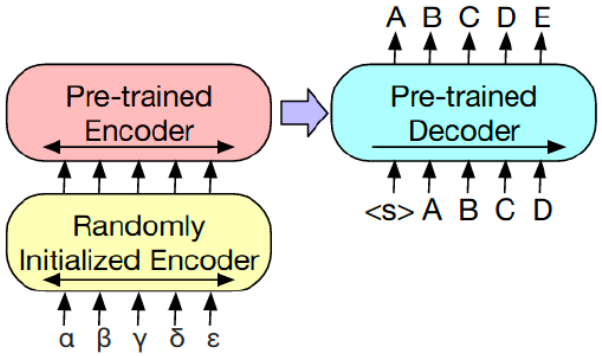
Bart – Pre-training Task



- The Pre-training task is to denoise a corrupted text. Several noising functions are tested:
  - Token Masking: random tokens are replaced with a [MASK] token.
  - Token Deletion: random tokens are deleted.
  - Text Infilling: Random text spans are sampled with span length drawn from Poisson distribution ($\lambda = 3$) and replaced with [MASK] token.
  - Sentence Permutation: Sentences are shuffled.
  - Sentence Rotation: sentences are rotated.
- Ablation experiments show that a combination of text infilling and sentence permutation perform consistently good on supervised tasks.

Bart – Fine tuning





- Bart can be fine-tuned on different types of tasks:
  - **Sequence Generation:** Given the decoder is autoregressive, Bart can be directly used to generate sequences in the same language as the input sequence.
  - **Sequence classification:** The final hidden state of the final decoder token is used. (Figure to the left)
  - **Tokens classification:** The top hidden state of the each decoder token is used.
  - **Machine translation:** The encoder embedding layer is replaced with an encoder that encodes the source language. (Figure to the right)

Bart – Evaluation

	SQuAD 1.1 EM/F1	SQuAD 2.0 EM/F1	MNLI m/mm	SST Acc	QQP Acc	QNLI Acc	STS-B Acc	RTE Acc	MRPC Acc	CoLA Mcc
BERT	84.1/90.9	79.0/81.8	86.6/-	93.2	91.3	92.3	90.0	70.4	88.0	60.6
UniLM	-/-	80.5/83.4	87.0/85.9	94.5	-	92.7	-	70.9	-	61.1
XLNet	**89.0**/94.5	86.1/88.8	89.8/-	95.6	91.8	93.9	91.8	83.8	89.2	63.6
RoBERTa	88.9/**94.6**	**86.5/89.4**	**90.2/90.2**	96.4	92.2	94.7	**92.4**	86.6	**90.9**	**68.0**
BART	88.8/**94.6**	86.1/89.2	89.9/90.1	**96.6**	**92.5**	**94.9**	91.2	**87.0**	90.4	62.8

	CNN/DailyMail			XSum		
	R1	R2	RL	R1	R2	RL
Lead-3	40.42	17.62	36.67	16.30	1.60	11.95
PTGEN (See et al., 2017)	36.44	15.66	33.42	29.70	9.21	23.24
PTGEN+COV (See et al., 2017)	39.53	17.28	36.38	28.10	8.02	21.72
UniLM	43.33	20.21	40.51	-	-	-
BERTSUMABS (Liu & Lapata, 2019)	41.72	19.39	38.76	38.76	16.33	31.15
BERTSUMEXTABS (Liu & Lapata, 2019)	42.13	19.60	39.18	38.81	16.50	31.27
BART	**44.16**	**21.28**	**40.90**	**45.14**	**22.27**	**37.25**