CS7150 Deep Learning

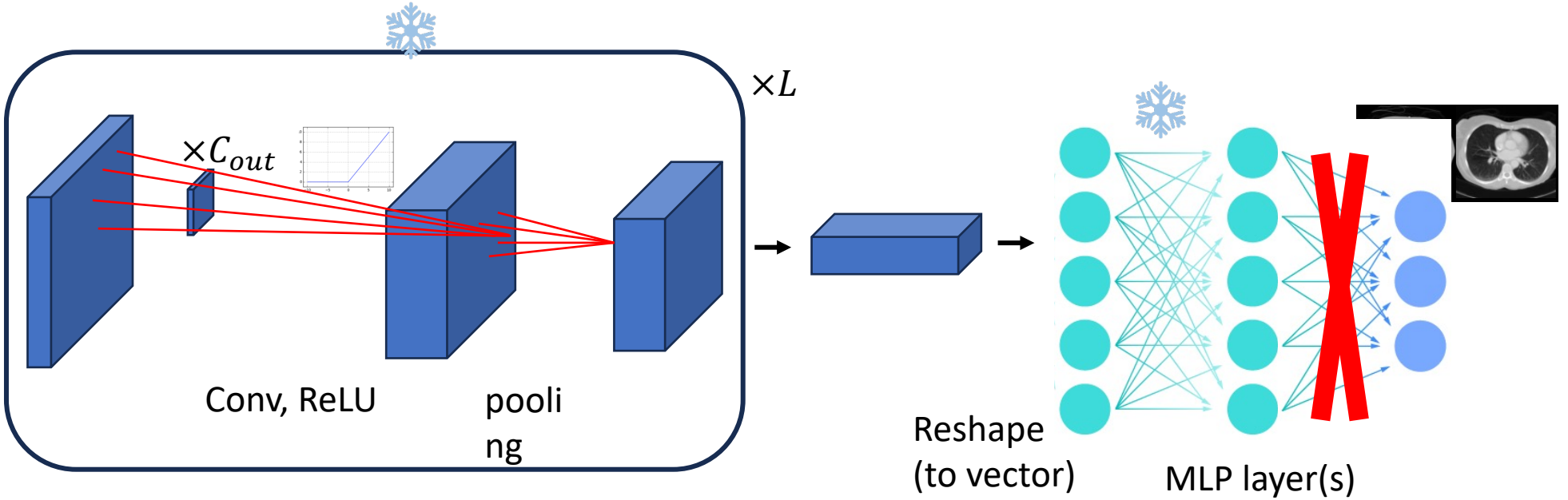
Jiaji Huang

https://jiaji-huang.github.io

02/03/2024

Recap of Last Lecture

- Convolution, correlation, pooling
- Convolutional Neural network
- Transfer learning
- Beyond Image Classification

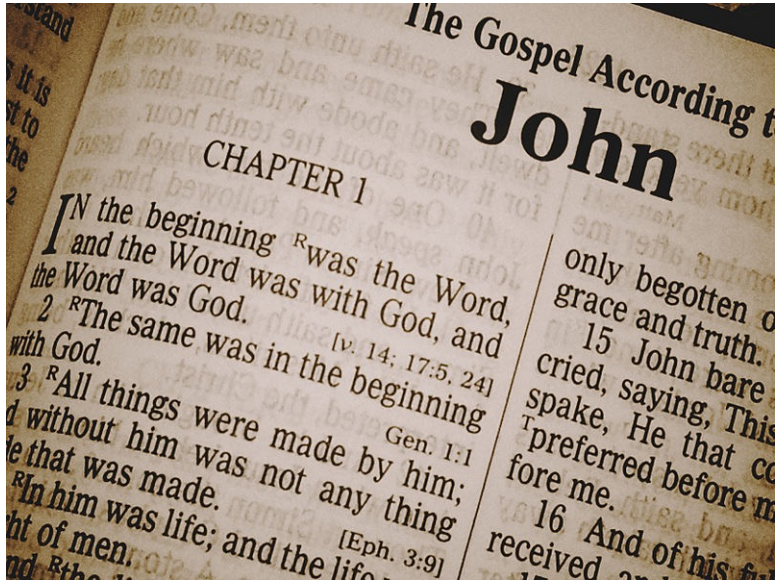


Agenda

- Learning on Sequences
- Word Embedding
- Recurrent Neural Networks
- Transformer

Modeling Sequences

- Natural Language



- Source Code

Illustrations from [wiki] and this [blogpost]

Modeling Sequences

- Speech Signal

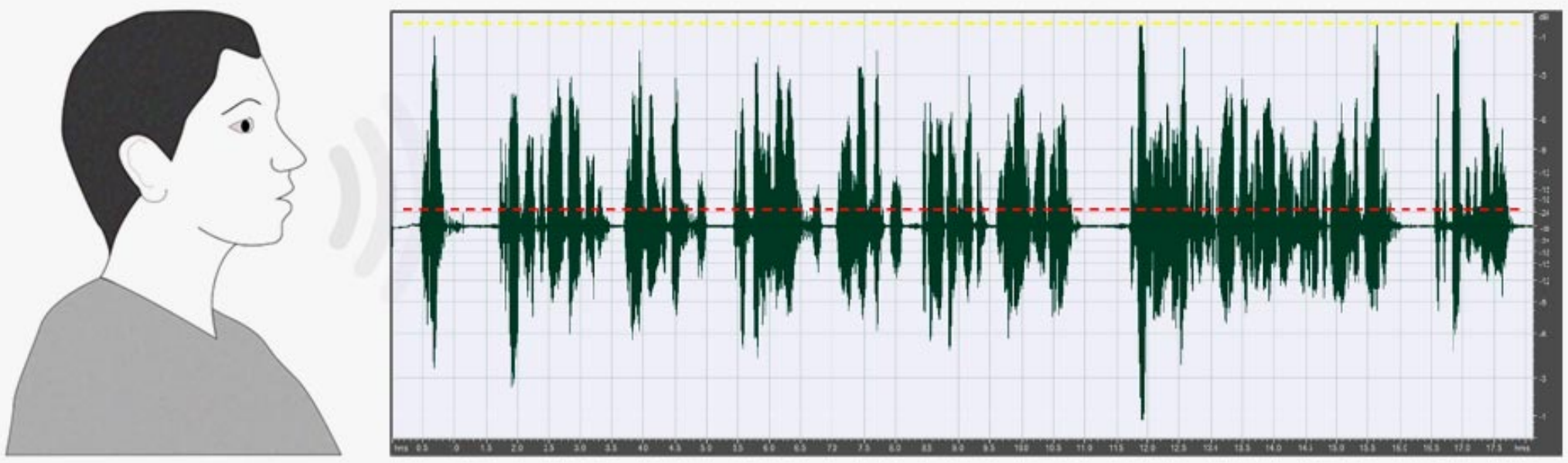


Illustration from this blogpost

Modeling Sequences

- DNA

Illustration from [Yang et. al 2020]

Shannon Game

- Guess the next letter (a-z and space) given previous (C. Shannon, 1950)

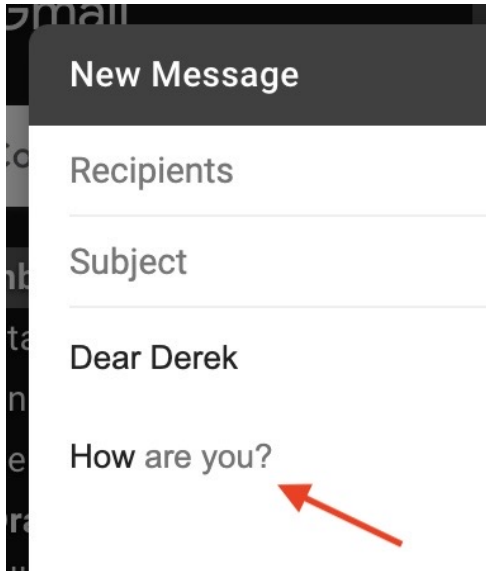
first line is the original text and the numbers in the second line indicate the guess at which the correct letter was obtained.

(1) T H E R E I S N O R E V E R S E O N A M O T O R C Y C LE A
(2) 1 1 1 5 1 1 2 1 1 2 1 1 1 5 1 1 7 1 1 1 2 1 3 2 1 2 2 7 1 1 1 1 4 1 1 1 1 3 1

(1) F R I E N D O F M I N E F O U N D T H I S O U T
(2) 8 6 1 3 1 1 1 1 1 1 1 1 1 1 6 2 1 1 1 1 1 2 1 1 1 1 1

(1) R A T H E R D R A M A T I C A L L Y T H E O T H E R D A Y
(2) 4 1 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1 1 1 1 1 6 1 1 1 1 1 1 1 1 1 1 1 1 1 (9)

- Modern example: auto word suggestion of email clients



Statistical Language Modeling

- Modeling $p(w_t|w_1^{t-1})$
- n-gram: assuming $p(w_t|w_1^{t-1}) = p(w_t|w_{t-(n-1)}^{t-1})$
- unigram $p(w) = \frac{\#w}{\sum_w \#w}$
  where $\#w$: occurrence of word type w in the corpus
- Bigram $p(w_t|w_{t-1}) = \frac{\#(w_{t-1},w_t)}{\#w_{t-1}}$, enumerate all types of (w_{t-1}, w_t)'s
- Trigram $p(w_t|w_{t-2}, w_{t-1}) = \frac{\#(w_{t-2},w_{t-1},w_t)}{\#(w_{t-2},w_{t-1})}$,
  enumerate all types of (w_{t-2}, w_{t-1}, w_t)'s

“Train” an n-gram Model

- $p\left(w_t \middle| w_{t-(n-1)}^{t-1}\right) = \frac{\#(w_{t-(n-1)},\dots w_{t-1},w_t)}{\#(w_{t-(n-1)},\dots w_{t-1})}$, enumerating all types of $\#(w_{t-(n-1)}, \dots w_{t-1}, w_t)$
- Gather a corpus
- Calculate unigram $p(w)$ for all word types w
- For $k = 2, \dots, n$:
  - Calculate k-grams for all types of k-tuples
- Store the probabilities as a look-up table
- Question: what if a n-tuple type has zero count

Back-off

- When $\#(w_{t-(n-1)}, \dots w_{t-1}, w_t) > \epsilon$
  - Discount the usual n-gram

$$p\left(w_t \middle| w_{t-(n-1)}^{t-1}\right) = d \cdot \frac{\#(w_{t-(n-1)}, \dots w_{t-1}, w_t)}{\#(w_{t-(n-1)}, \dots w_{t-1})}$$

- $p\left(w_t \middle| w_{t-(n-1)}^{t-1}\right)$ when $\#(w_{t-(n-1)}, \dots w_{t-1}, w_t) < \epsilon$
  - Back off to $(n-1)$ gram

$$p\left(w_t \middle| w_{t-(n-1)}^{t-1}\right) = \alpha \cdot p\left(w_t \middle| w_{t-(n-2)}^{t-1}\right)$$

- d by Good–Turing estimation, α such that $p\left(w_t \middle| w_{t-(n-1)}^{t-1}\right)$ adds to 1

Test an n-gram Model

- Gather a test corpus (word sequence $w_1, \dots, w_T$)
- Calculate

$$p(w_1, \dots, w_T) = \prod_{t=1}^{T} p(w_t | w_1^{t-1}) = \prod_{t=1}^{T} p\left(w_t | w_{\max(1,\, t-(n-1))}^{t-1}\right)$$

- Very tiny number, use entropy:

$$\frac{1}{T} \ln p(w_1, \dots, w_T) = \frac{1}{T} \sum_{t=1}^{T} \ln p\left(w_t | w_{\max(1,t-(n-1))}^{t-1}\right)$$

- (Test) Perplexity: $PPL \triangleq \exp\left(-\frac{1}{T}\sum_{t=1}^{T} \ln p\left(w_t | w_{\max(1,t-(n-1))}^{t-1}\right)\right)$

Entropy and PPL

- If we take base 2
  - The entropy $\frac{1}{T}\sum_{t=1}^{T}\log_2 p\left(w_t|w_{\max(1,t-(n-1))}^{t-1}\right)$ is measured in bits
  - PPL = 2^{entropy}
- $1 \leq$ PPL $\leq$ all possibilities (vocabulary size)
- Back to Shannon game
  - PPL ≤ 27
  - Therefore Bit per Character (bpc) ≤ 4.75
  - As n-gram order increases, we get lower PPL

Column	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	100
Upper	4.03	3.42	3.0	2.6	2.7	2.2	2.8	1.8	1.9	2.1	2.2	2.3	2.1	1.7	2.1	13
Lower	3.19	2.50	2.1	1.7	1.7	1.3	1.8	1.0	1.0	1.0	1.3	1.3	1.2	.9	1.2	.6

Table from (C. Shannon, 1950)

Packages

- Kenlm, srilm
- ARPA format

log_{10} unigram

log_{10} bigram

```
\data\
ngram 1=7
ngram 2=7

\1-grams:
-1.0000 <unk>   -0.2553
-98.9366 <s>     -0.3064
-1.0000 </s>     0.0000
-0.6990 wood     -0.2553
-0.6990 cindy   -0.2553
-0.6990 pittsburgh              -0.2553
-0.6990 jean     -0.1973

\2-grams:
-0.2553 <unk> wood
-0.2553 <s> <unk>
-0.2553 wood pittsburgh
-0.2553 cindy jean
-0.2553 pittsburgh cindy
-0.5563 jean </s>
-0.5563 jean wood

\end\
```

Backoff weights (log_{10})

Example from this blogpost

Agenda

- Learning on Sequences
- Word Embedding
- Recurrent Neural Networks
- Transformer

Motivating

- Many more tasks beyond predicting the next word/letter
- E.g., sentiment classification

Cyphis

[redacted] **One Star is Too Much for This Product**

Reviewed in the United States on September 7, 2012

I don't know if this is a scam or if mine was broken, but it doesn't work and I am still getting abducted by UFO's on a regular basis.

Amazon Customer

[redacted] **it works**

Reviewed in the United States on March 21, 2018

Verified Purchase

it works great

UFO detector sold on amazon

Motivating

- E.g., Named Entity Recognition (NER)

Apple Surprise Reveal: Biggest-Ever Shake-Up For Your iPhone Here In Weeks

?

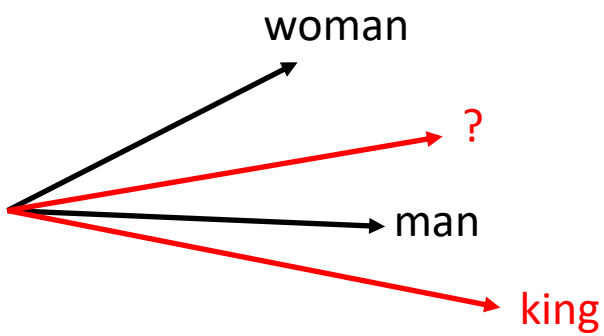
Credit: Forbes news

Word Representations

- Aka embeddings
- One-hot embedding: not very useful
- It’s better to encoding some semantics, e.g.,

Semantics: Word Analogy Task

- Allows us to use linear algebra
- Man to woman is like king to ?



man – woman = king - ?
$\Rightarrow$? = king – man + woman
$\Rightarrow$ Take ? as the nearest neighbor of RHS

Word2vec

Efficient Estimation of Word Representations in Vector Space

by T Mikolov · 2013 · Cited by 37327 — Download a PDF of the **paper** titled Efficient Estimation of Word Representations in Vector Space, by Tomas Mikolov and 3 other authors.

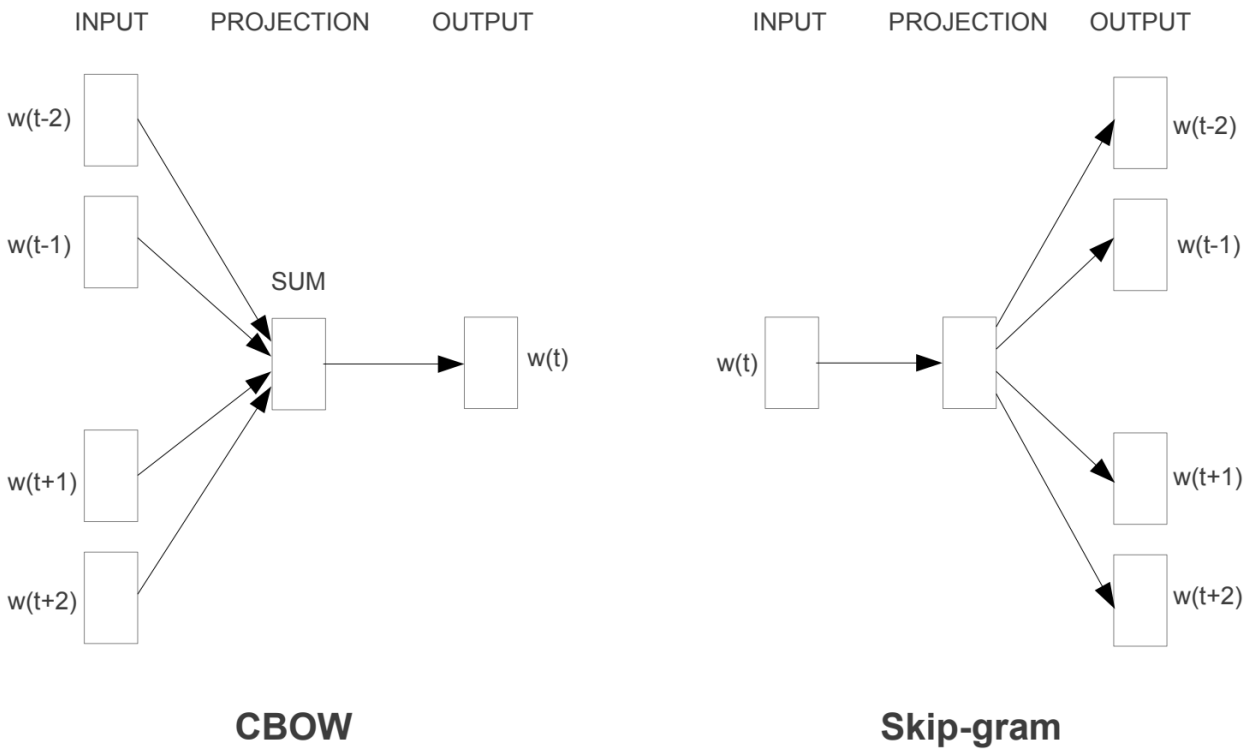


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Deep dive into skip-gram

How do we train it? Suppose a window size of 5

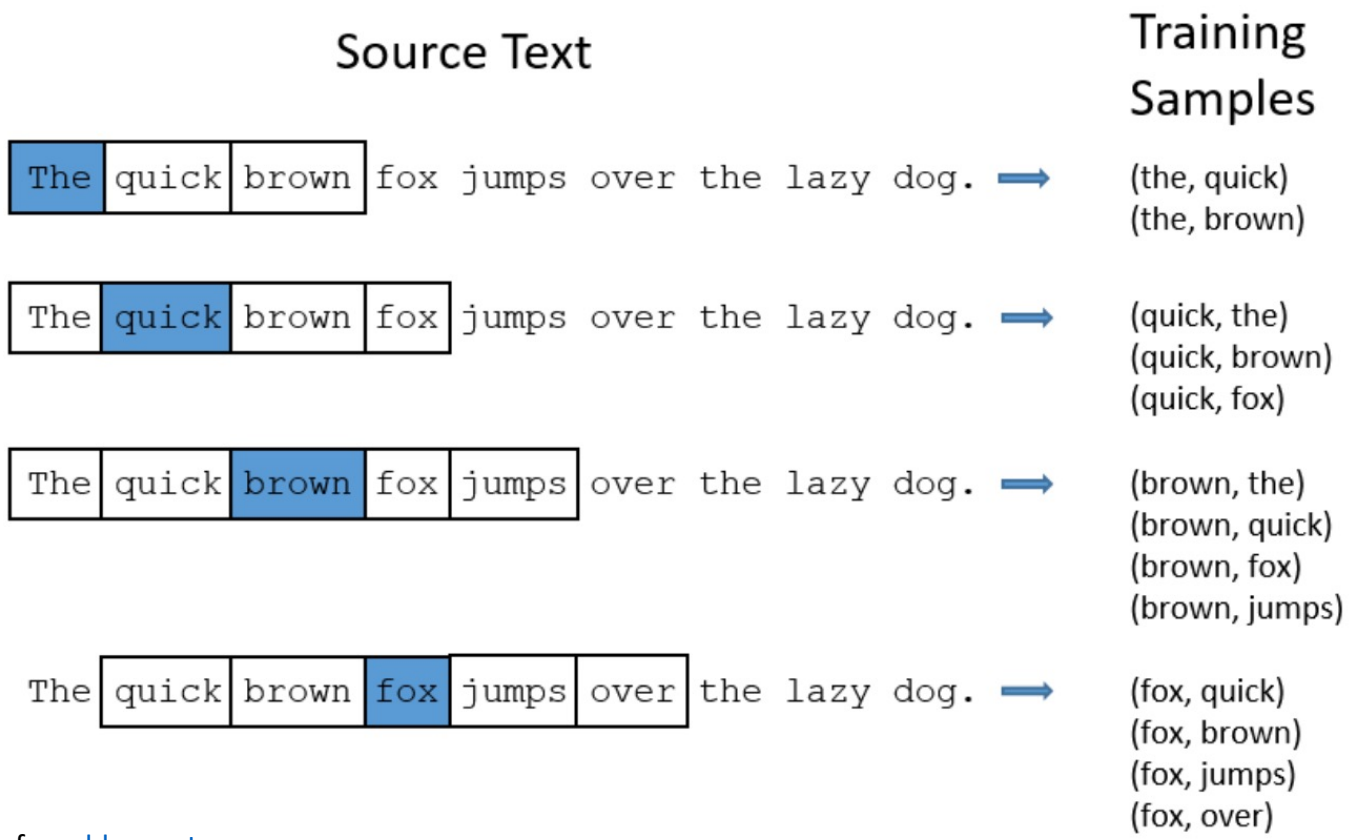
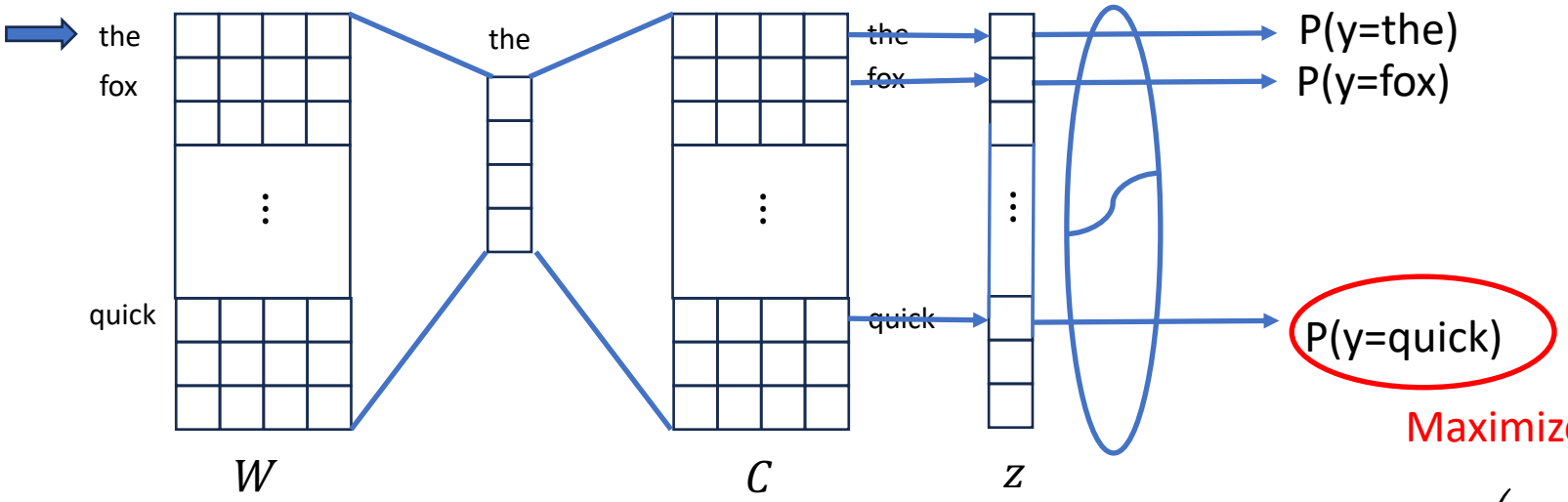


Figure taken from [blogpost]

Deep dive into skip-gram

- Viewed as 2-layer network
- e.g., receive a training sample (w="the", c="quick")



$$\text{softmax}\,(z_1,..,z_n) = \left(\frac{e^{z_1}}{\sum_{i=1}^{n} z_i}, \ldots, \frac{e^{z_n}}{\sum_{i=1}^{n} z_i}\right)$$

Deep dive into skip-gram

- The overall training objective is

$$\min_{\boldsymbol{W},\boldsymbol{C}} \sum_{w,c} -\log p_{\boldsymbol{W},\boldsymbol{C}}(c|w)$$

- Typically, we use $\boldsymbol{W}$ as word representation
- Trained by stochastic gradient descent (SGD)

Skip-Gram with Negative Sampling (SGNS)

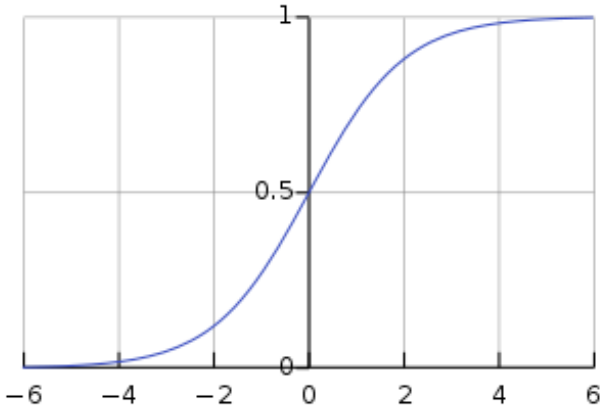
Neural Information Processing Systems
https://papers.nips.cc › paper › 5021-distributed-repre...

Distributed Representations of Words and Phrases ...

by T Mikolov · 2013 · Cited by 40525 — An inherent limitation of **word representations** is **their** indifference to **word** order and **their** inability to represent idiomatic **phrases**. For example, the…

- Vocab can be huge
- huge summation on softmax denominator!
- An efficient variant: contrast with negative samples
- P(x and y co-occur) = $\sigma\left(W_x^T C_y\right)$,

  where $\sigma(s) = \frac{1}{1+e^{-s}}$, sigmoid function



Skip-Gram with Negative Sampling (SGNS)

- Denote negative samples as $y_i,\ i = 1, \dots, k$ from $q(y)$
- Maximize:

$$\log \sigma\left(W_x^T C_y\right) + \sum_{i=1}^{k} \left[\log \sigma\left(-W_x^T C_{y_i}\right)\right]$$

- In practice, $q(y)$ is uni-gram probability

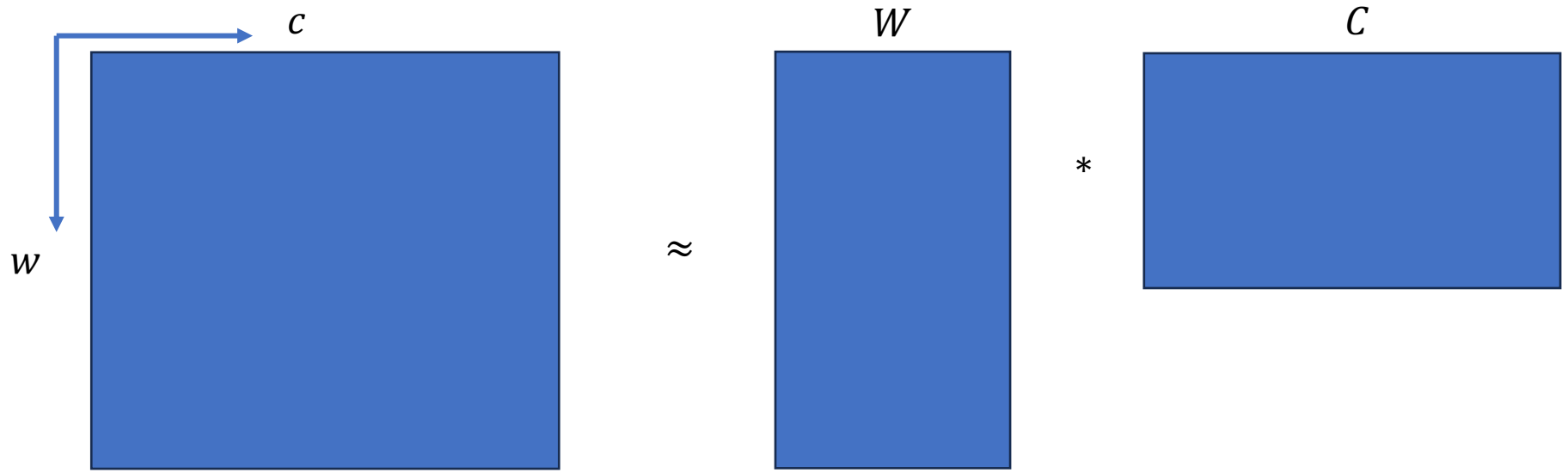
Relationship with Matrix Factorization

- (O. Levy et. al, 2014) shows that SGNS is implicitly factorizing

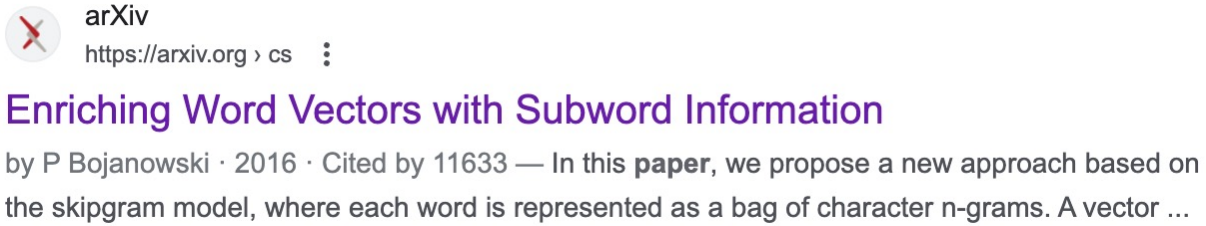
$$\text{PMI}(w, c) = \log \frac{\#(w,c)}{\#w \cdot \#c}$$

- Glove (Pennington et. al, 2014): Weighted Factorization of

a word-word co-occurrence matrix



FastText



- Embedding for a word not seen during training?
- Break word into char n-grams, e.g., n=3

  where → <wh whe her ere re>
- W[“where”] = W[“<wh”] + W[“whe”] + W[“her”] + W[“ere”] + W[“re>”]
- Training: plug above into SGNS
- Testing: for unseen word: sum its char n-gram embeddings

Model downloadable from here

Evaluation

- Extrinsic evaluation
  - Plug into an ML system and evaluate on a downstream task
  - depends on the ML system
  - But practically very indicative
  - Example: sentiment classification. “I like this movie” v.s. “This is a fair movie”
- Intrinsic evaluation
  - Intermediate, not directly related to downstream task
  - Example: Word analogy

Why so successful

Word2Vec

Published online by Cambridge University Press: **16 December 2016**

KENNETH WARD CHURCH

Show author details

Article Figures Metrics

Save PDF | Share | Cite | Rights & Permissions

Abstract

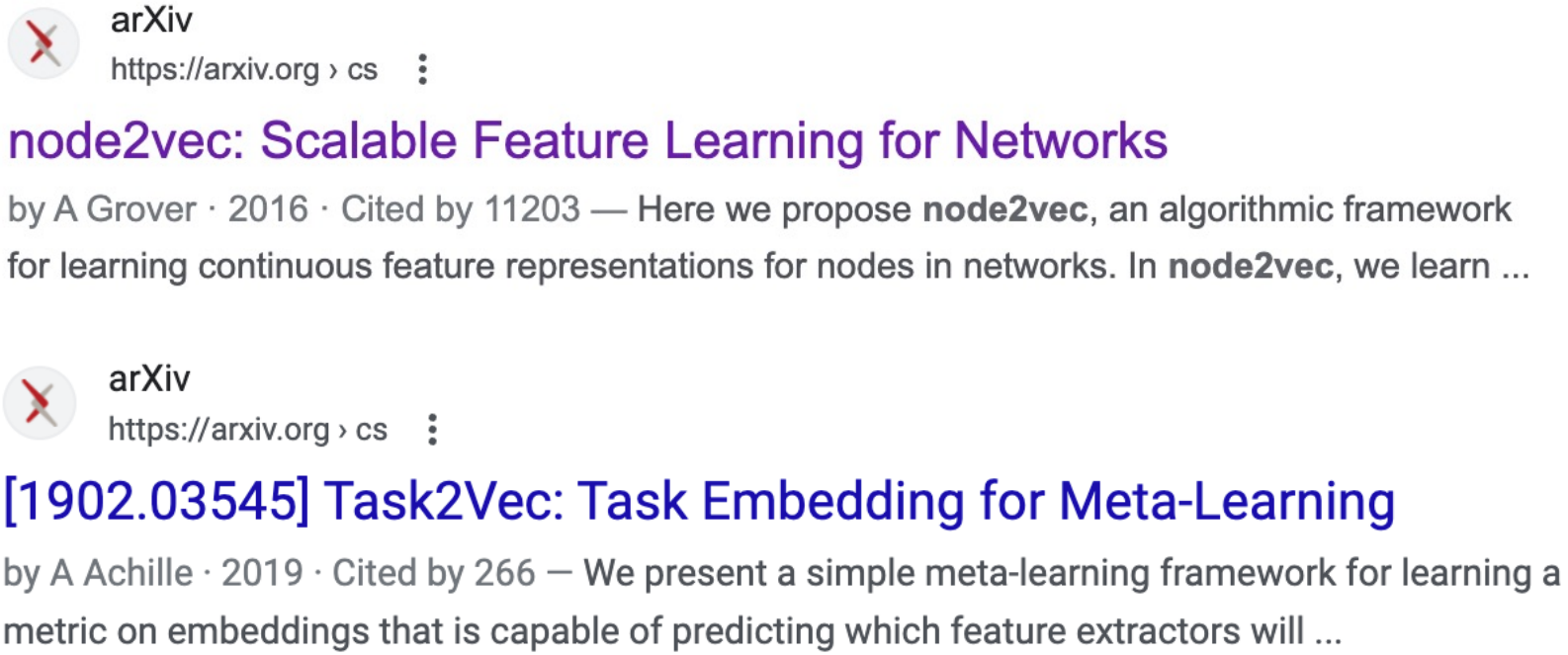
My last column ended with some comments about Kuhn and word2vec. Word2vec has racked up plenty of citations because it satisifies both of Kuhn's conditions for emerging trends: (1) a few initial (promising, if not convincing) successes that motivate early adopters (students) to do more, as well as (2) leaving plenty of room for early adopters to contribute and benefit by doing so. The fact that Google has so much to say on 'How does word2vec work' makes it clear that the definitive answer to that question has yet to be written. It also helps citation counts to distribute code and data to make it that much easier for the next generation to take advantage of the opportunities (and cite your work in the process).

- native c implementation
- handy python package: genism, fastText, …

- A few initial successes
- Plenty of room to improve
- Publish code and data

Generalize: *2vec

- Exploit pairwise associations between entities

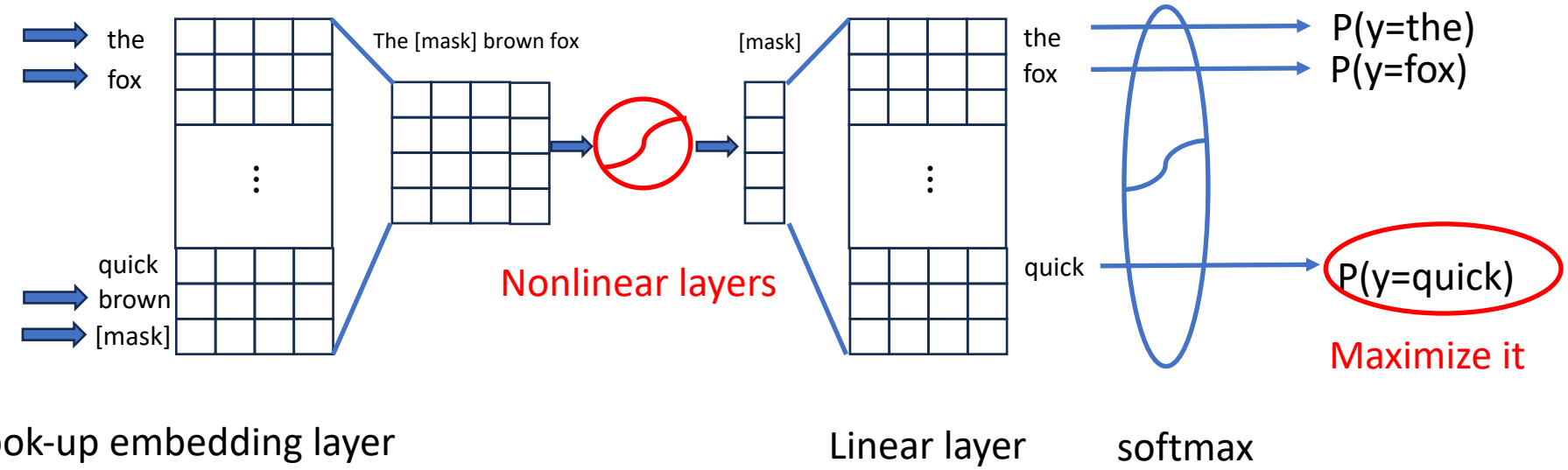


Agenda

- Learning on Sequences
- Word Embedding
- Recurrent Neural Networks
- Transformer

Motivation

- N-gram and Word2vec both look into a small context window
- Model the entire sequence? E.g.,



- Would MLP work?
- Would conv net work? (Y. Dauphin *et. al*, 2017)

Recurrent Neural Network

- Hidden state h_t
- Input x_t
- $h_t = g(Uh_{t-1} + Wx_t)$, e.g., $g(\cdot)$ as tanh
- $y_t = \text{softmax}(Vh_t)$

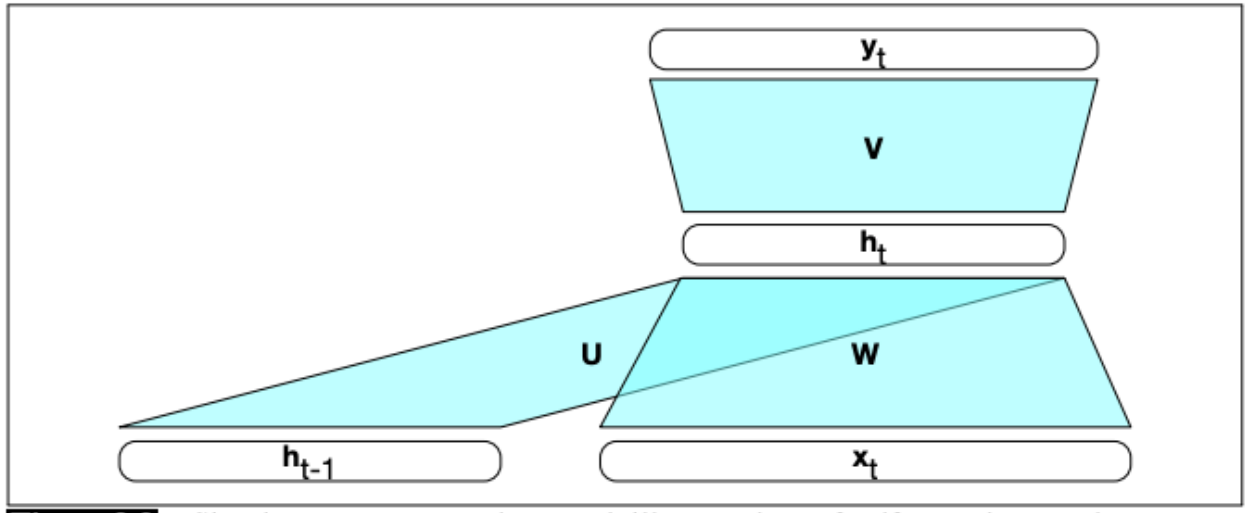


Figure 9.2 Simple recurrent neural network illustrated as a feedforward network.

Illustration from https://web.stanford.edu/~jurafsky/slp3/9.pdf

RNN

- “Unroll” along time axis

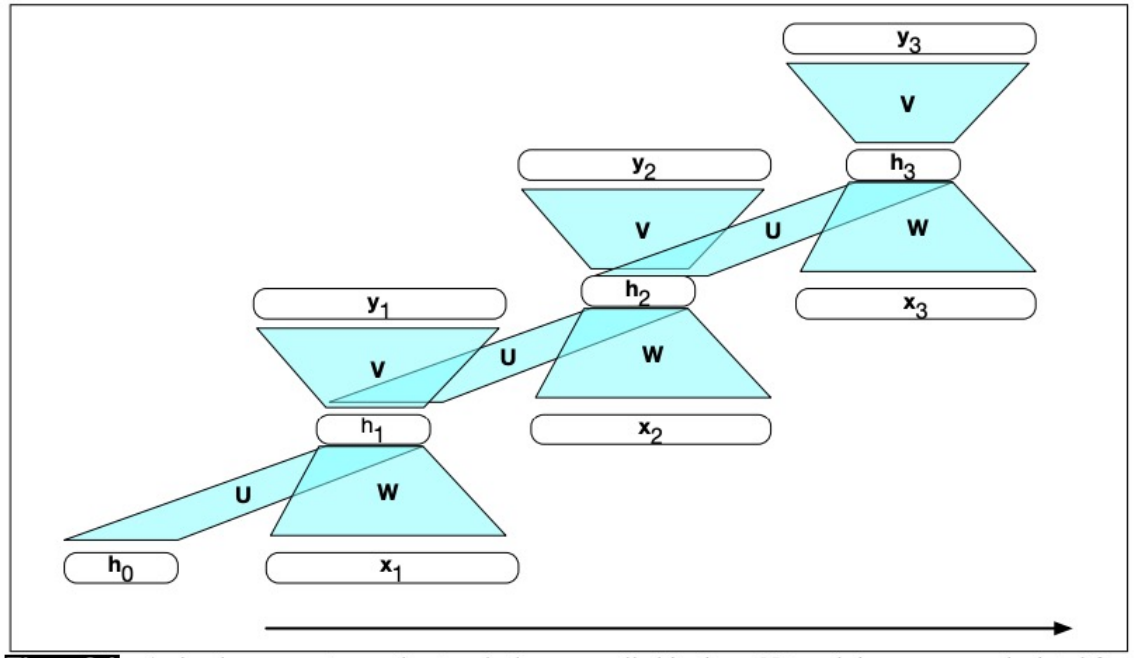


Figure 9.4 A simple recurrent neural network shown unrolled in time. Network layers are recalculated for each time step, while the weights **U**, **V** and **W** are shared across all time steps.

Illustration from https://web.stanford.edu/~jurafsky/slp3/9.pdf

Stacked RNN layers

$$h_t^1 = g(U^1 h_{t-1}^1 + W^1 x_t)$$

$$h_t^2 = g(U^2 h_{t-1}^2 + W^2 h_t^1)$$

$$h_t^3 = g(U^3 h_{t-1}^3 + W^3 h_t^2)$$

$$\vdots$$

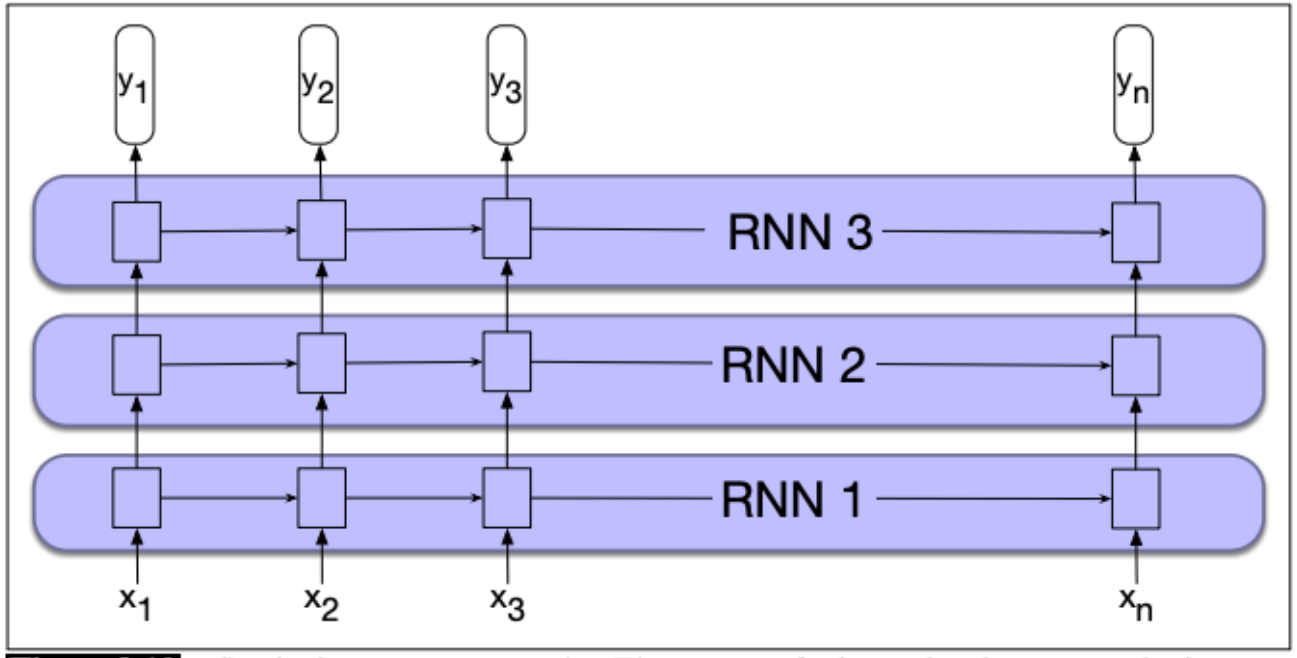


Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

Illustration from https://web.stanford.edu/~jurafsky/slp3/9.pdf

RNN for language Modeling

- Predict next word given previous ones

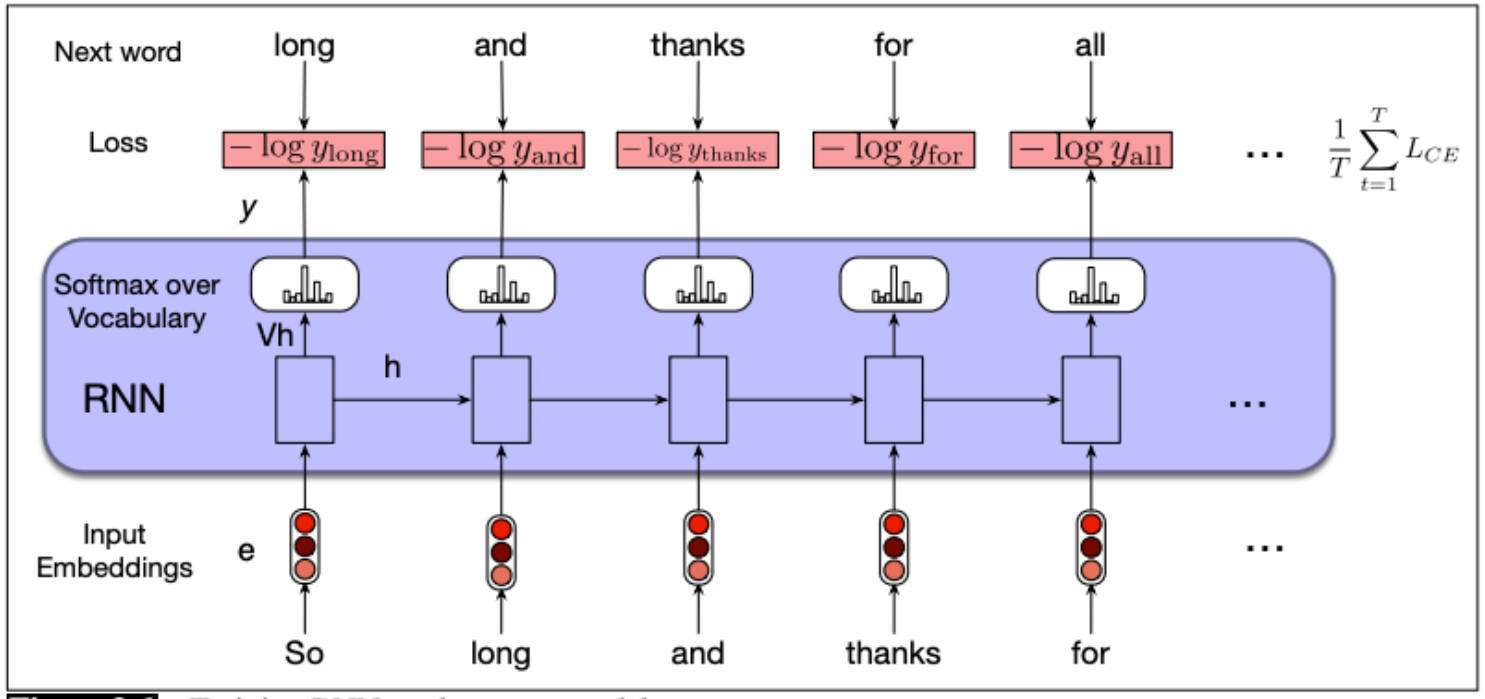


Figure 9.6 Training RNNs as language models.

Illustration from https://web.stanford.edu/~jurafsky/slp3/9.pdf

Bi-directional RNN

- Input $x_1, x_2, \ldots, x_T$
- $h_t^f = RNN_f(x_1, x_2, \ldots, x_t)$
- $h_t^b = RNN_b(x_T, x_{T-1}, \ldots x_t)$
- $h_t = [h_t^f, h_t^b]$

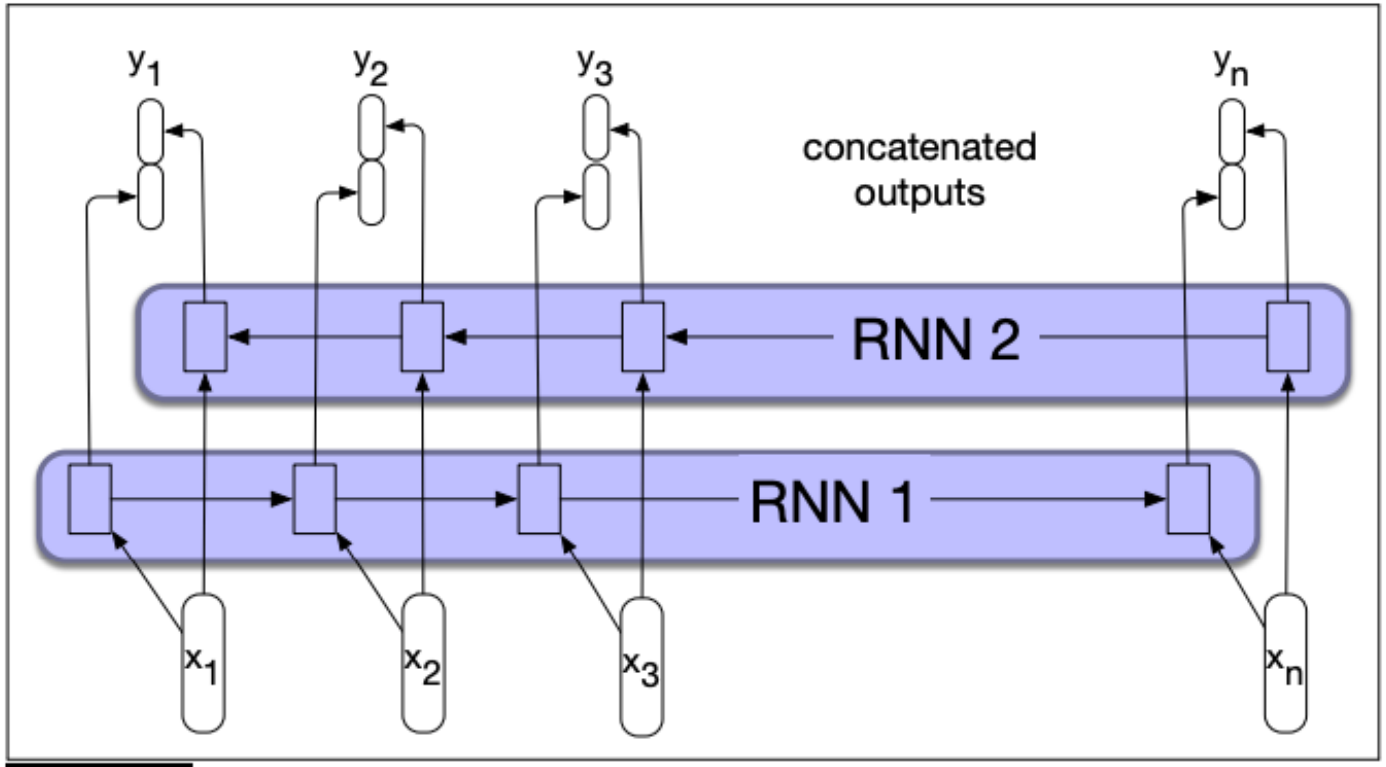


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

Illustration from https://web.stanford.edu/~jurafsky/slp3/9.pdf

Why bi-directional

- Some applications allow us to see the entire sequence, e.g.,
  - Text Classification
  - Speech Recognition
- Bi-directional often beats uni-directional:
  - e.g., a speech recognition example (lower WER is better)

	WSJ		HKUST		LibriSpeech			
Encoder Architecture	dev	test	dev	test	dev clean	dev other	test clean	test other
"Google"-BLSTM	7.9	4.7	29.9	28.9	4.7	14.1	4.9	15.2
"Google"-LSTM	9.9	6.5	35.5	33.8	6.3	18.2	6.5	19.4

ASR results from https://www.jonathanleroux.org/pdf/Moritz2019Interspeech09.pdf

LSTM, but why?

- RNNs can be hard to train
- Gradient explosion and vanishing problem

- In *modern words*, attention span is very short!

LSTM

- Introduces memory c_t
- Forget gate

$$f_t = \sigma(U_f h_{t-1} + W_f x_t)$$

- Add gate

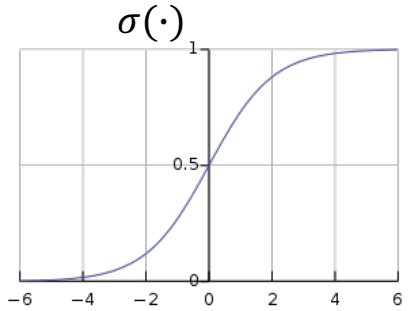
$$i_t = \sigma(U_i h_{t-1} + W_i x_t)$$
$$g_t = \tanh(U_g h_{t-1} + W_g x_t)$$

- Combined

$$c_t = g_t \odot i_t + c_{t-1} \odot f_t$$

- Output gate

$$o_t = \sigma(U_o h_{t-1} + W_o x_t)$$
$$h_t = o_t \odot \tanh(c_t)$$



sigmoid

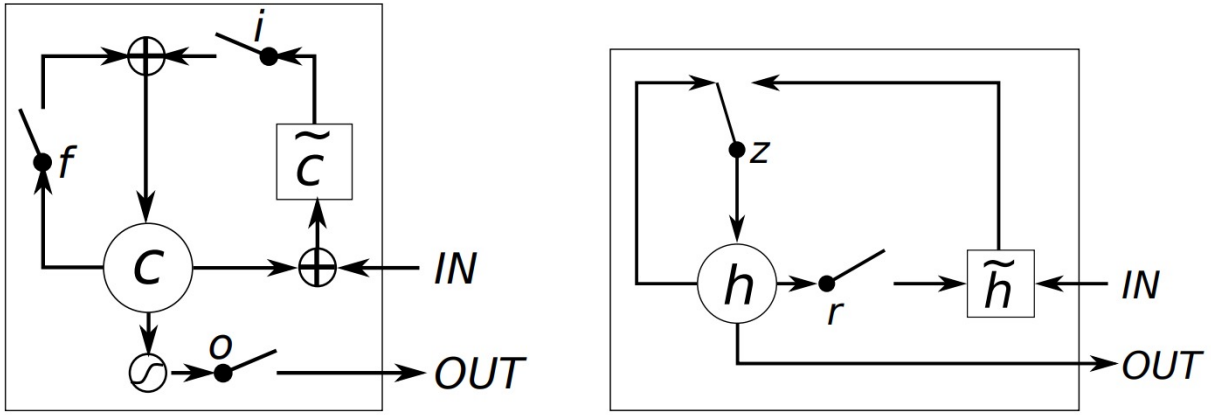
GRU

arXiv
https://arxiv.org › cs

Empirical Evaluation of Gated Recurrent Neural Networks ...

by J Chung · 2014 · Cited by 13908 — In this paper we compare different types of **recurrent** units in **recurrent neural networks** (RNNs). Especially, we focus on more sophisticated ...

- A Variant of LSTM, no memory cells
- fewer parameters



(a) Long Short-Term Memory (b) Gated Recurrent Unit

Figure 1: Illustration of (a) LSTM and (b) gated recurrent units. (a) i, f and o are the input, forget and output gates, respectively. c and $\tilde{c}$ denote the memory cell and the new memory cell content. (b) r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation.

Comparison

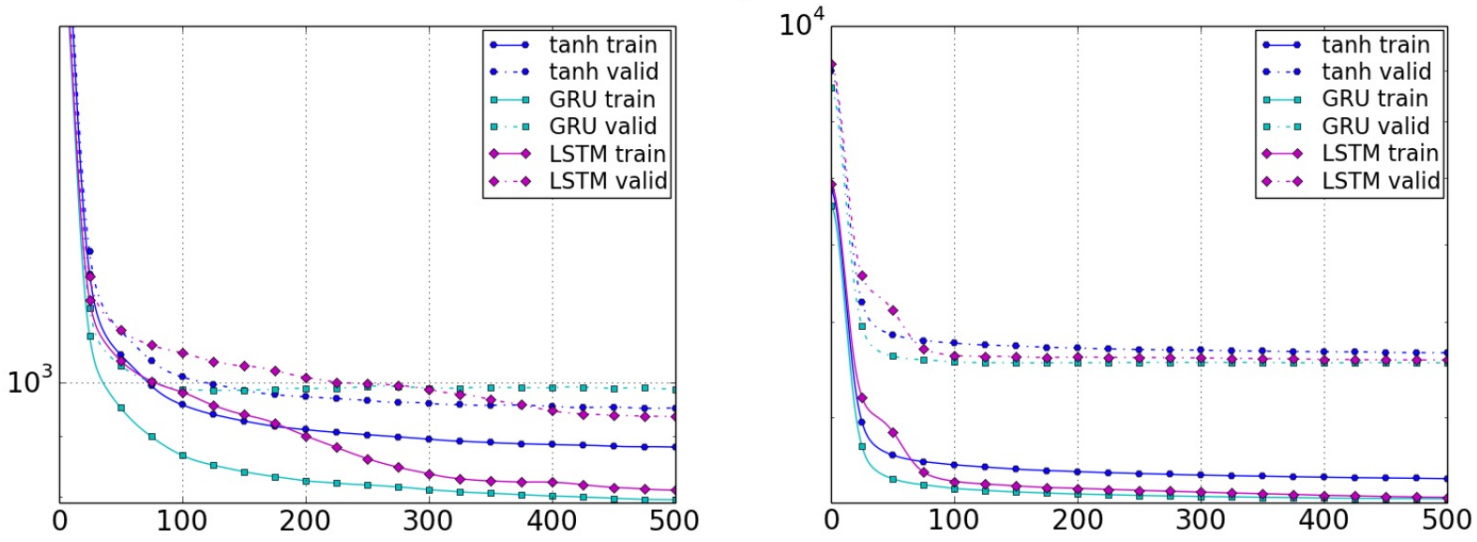
arXiv
https://arxiv.org › cs

Empirical Evaluation of Gated Recurrent Neural Networks ...

by J Chung · 2014 · Cited by 13908 — In this paper we compare different types of **recurrent** units in **recurrent neural networks** (RNNs). Especially, we focus on more sophisticated ...

These results clearly indicate the advantages of the gating units over the more traditional recurrent units. Convergence is often faster, and the final solutions tend to be better. However, our results are not conclusive in comparing the LSTM and the GRU, which suggests that the choice of the type of gated recurrent unit may depend heavily on the dataset and corresponding task.

Per epoch



Agenda

- Learning on Sequences
- Word Embedding
- Recurrent Neural Networks
- Transformer

Transformer Encoder

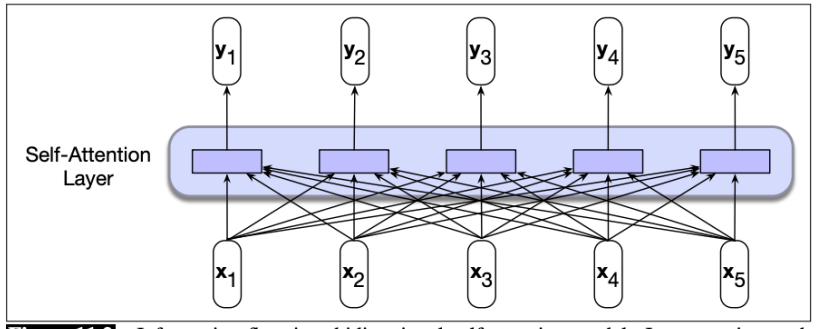
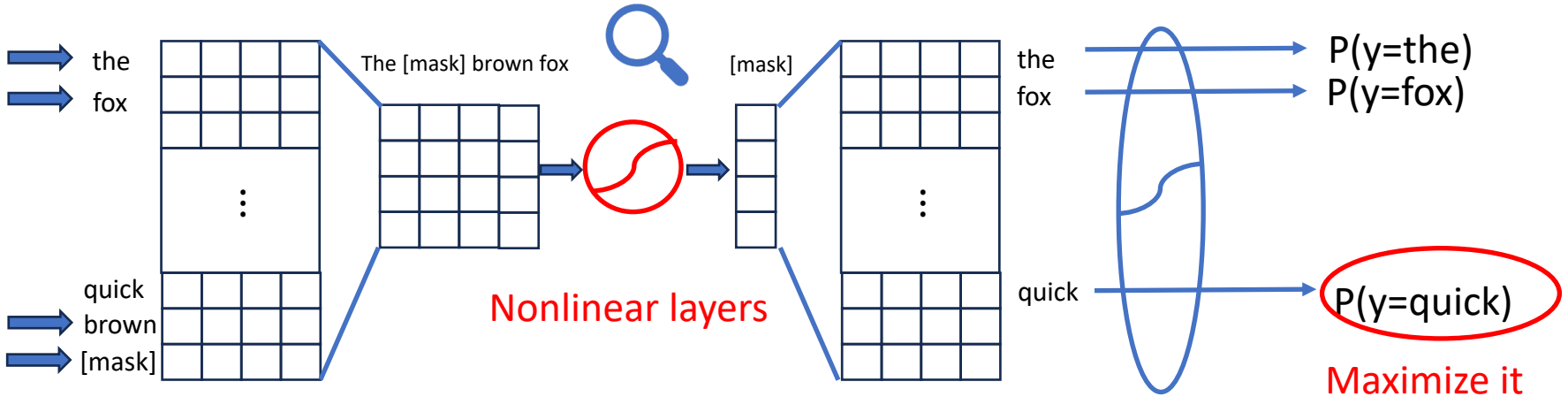


Figure 11.2 Information flow in a bidirectional self-attention model. In processing each element of the sequence, the model attends to all inputs, both before and after the current one.



Look-up embedding layer

Linear layer

softmax

Illustration from https://web.stanford.edu/~jurafsky/slp3/11.pdf

A Transformer Block

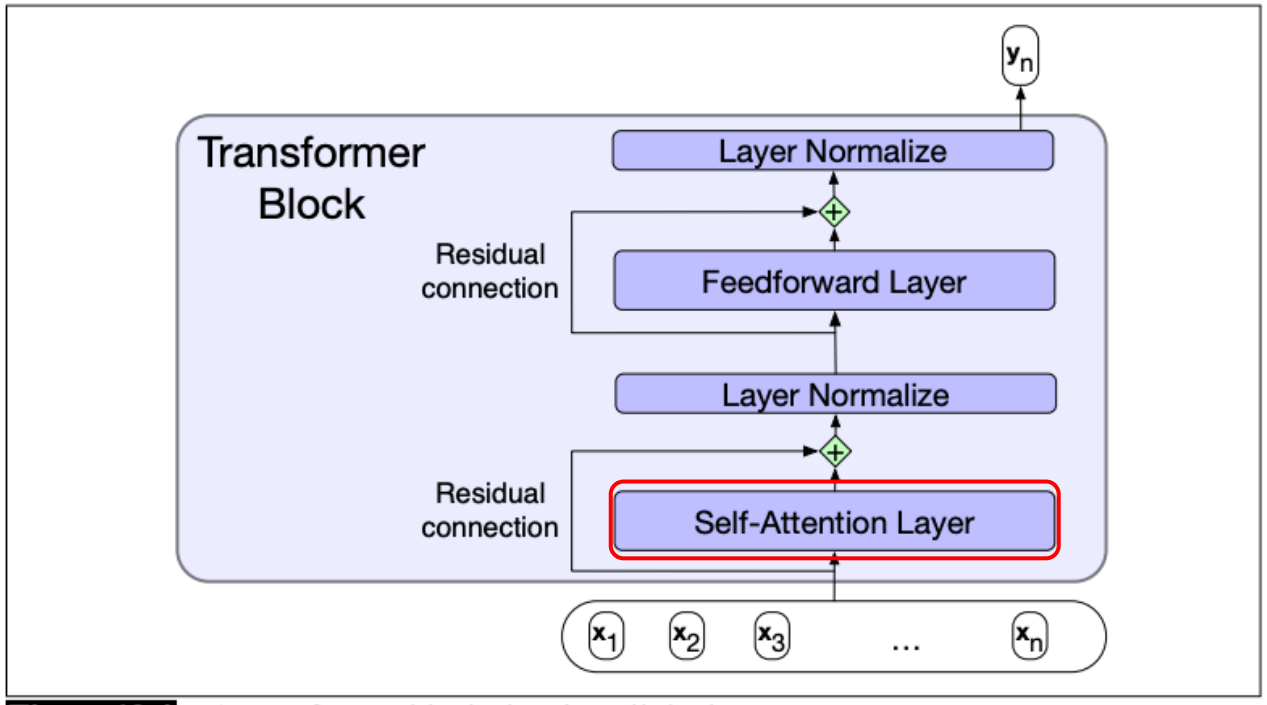


Figure 10.4 A transformer block showing all the layers.

Illustration from https://web.stanford.edu/~jurafsky/slp3/10.pdf

Math of Self-attention

- Map hidden states to keys, queries and values (d dimensional)

$$q_i = W^q x_i, k_i = W^k x_i, v_i = W^v x_i$$

- Raw score

$$s_{i,j} = q_i \cdot k_j / \sqrt{d}$$

- Attention weights

$$\alpha_{i,j} = \frac{e^{s_{i,j}}}{\sum_l e^{s_{i,l}}}$$

- Output Hidden states

$$y_i = \sum_j \alpha_{i,j} v_j$$

Multi-head Attention

- Input $x \in \mathbb{R}^d$
- Multiple sets of W^q, W^k and W^v
  - Each project to d/#heads
  - Typically,
    #heads=8, 12, 16, 24, …
- concatenate output from each head
- Project with W^o, usually same d

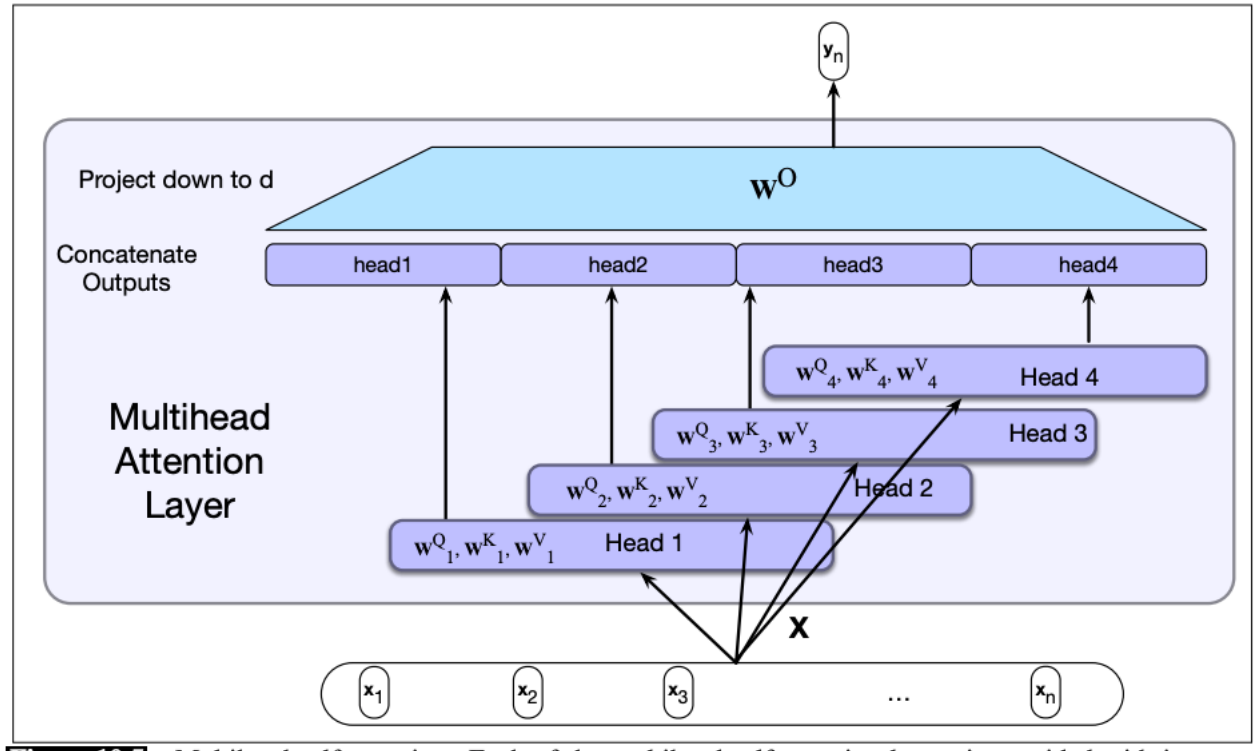


Figure 10.5 Multihead self-attention: Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices. The outputs from each of the layers are concatenated and then projected down to d, thus producing an output of the same size as the input so layers can be stacked.

Illustration from https://web.stanford.edu/~jurafsky/slp3/10.pdf

Residual Connection (refer to last lecture)

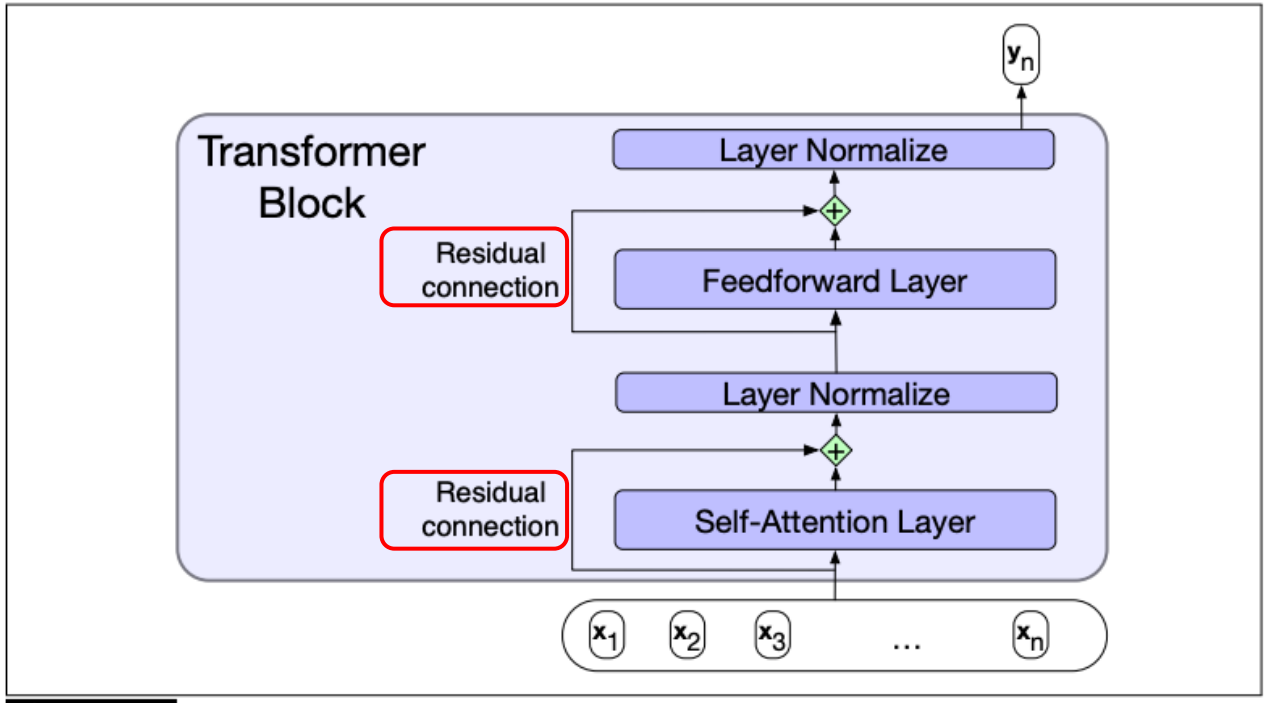


Figure 10.4 A transformer block showing all the layers.

$$x + FF(x)$$

$$x + selfAttn(x)$$

Illustration from https://web.stanford.edu/~jurafsky/slp3/10.pdf

Layer Normalization

arXiv
https://arxiv.org › stat

[1607.06450] Layer Normalization

by JL Ba · 2016 · Cited by 9138 — **Layer normalization** is very effective at stabilizing the hidden state dynamics in recurrent networks. Empirically, we show that layer ...

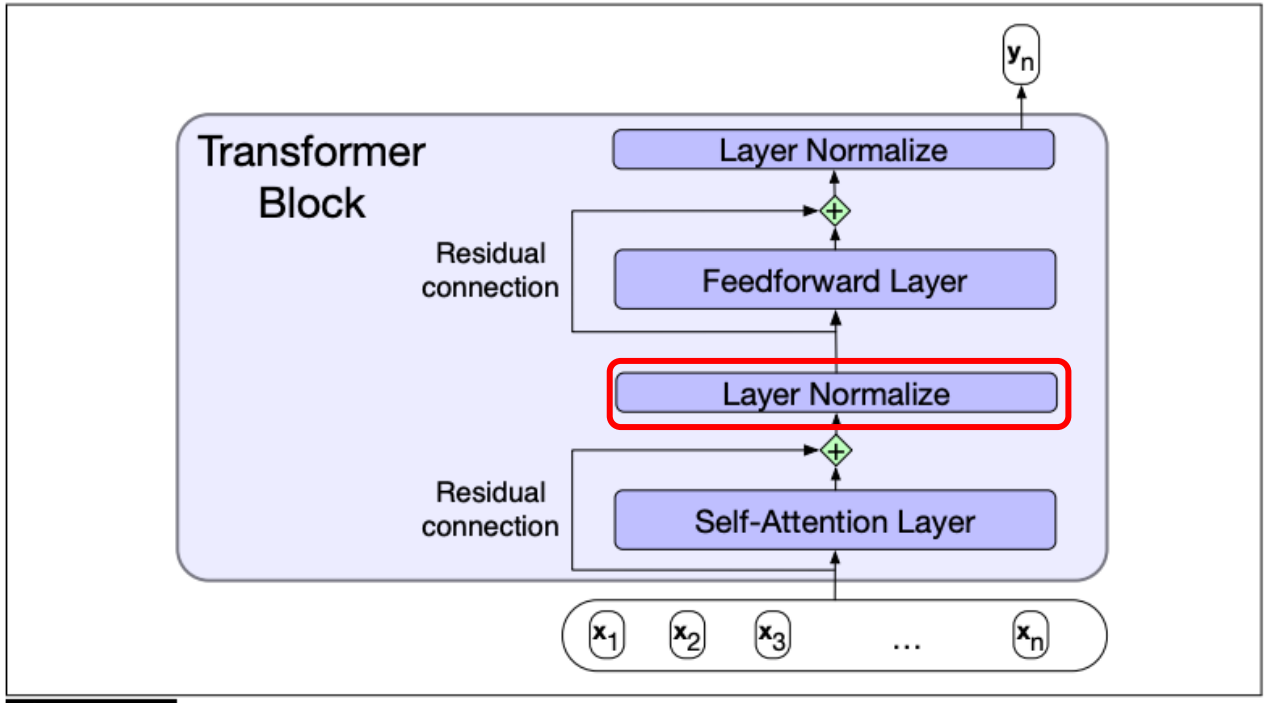


Figure 10.4 A transformer block showing all the layers.

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i-\mu)^2}$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x}-\mu)}{\sigma}$$

$$\text{LayerNorm} = \gamma\hat{\mathbf{x}} + \beta$$

Illustration from https://web.stanford.edu/~jurafsky/slp3/10.pdf

Feedforward Layer

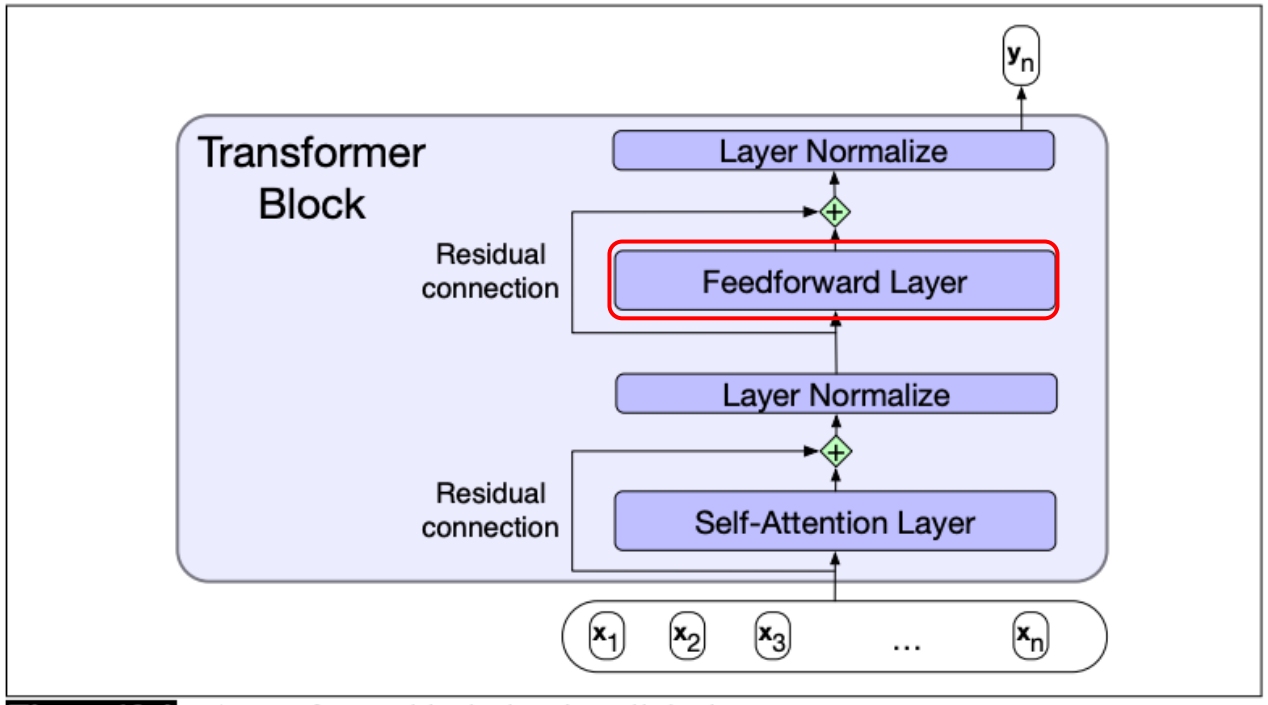


Figure 10.4 A transformer block showing all the layers.

$$y = W_2\sigma(W_1 x)$$

intermediate dimension: Usually 4 x input dimension

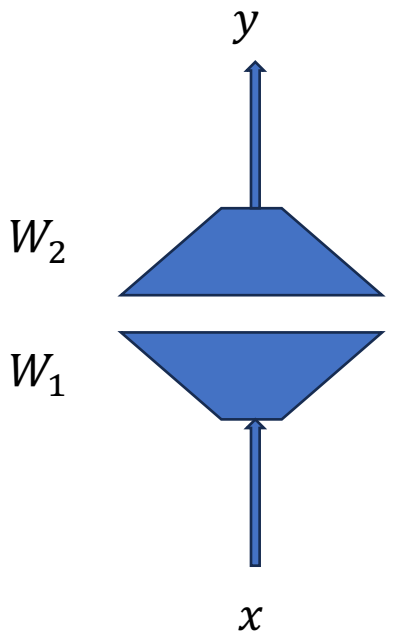


Illustration from https://web.stanford.edu/~jurafsky/slp3/10.pdf

(Causal) Attention Mask

- $\alpha_{i,j} = \frac{e^{s_{i,j}}}{\sum_l e^{s_{i,l}}}, y_i = \sum_j \alpha_{i,j} v_j$
- The range of j (or l) for causal language model: $j \leq i$

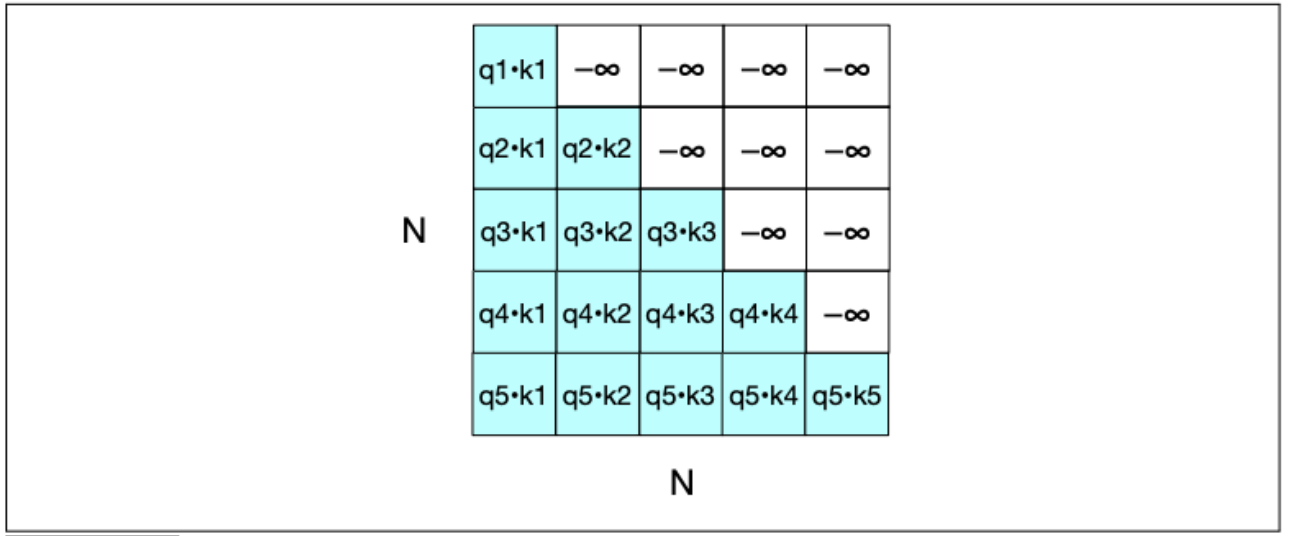


Figure 10.3 The $N \times N$ $\mathbf{QK}^\intercal$ matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero).

Illustration from https://web.stanford.edu/~jurafsky/slp3/10.pdf

Transformer Decoder

- With Attention mask, we can predict next word given previous ones

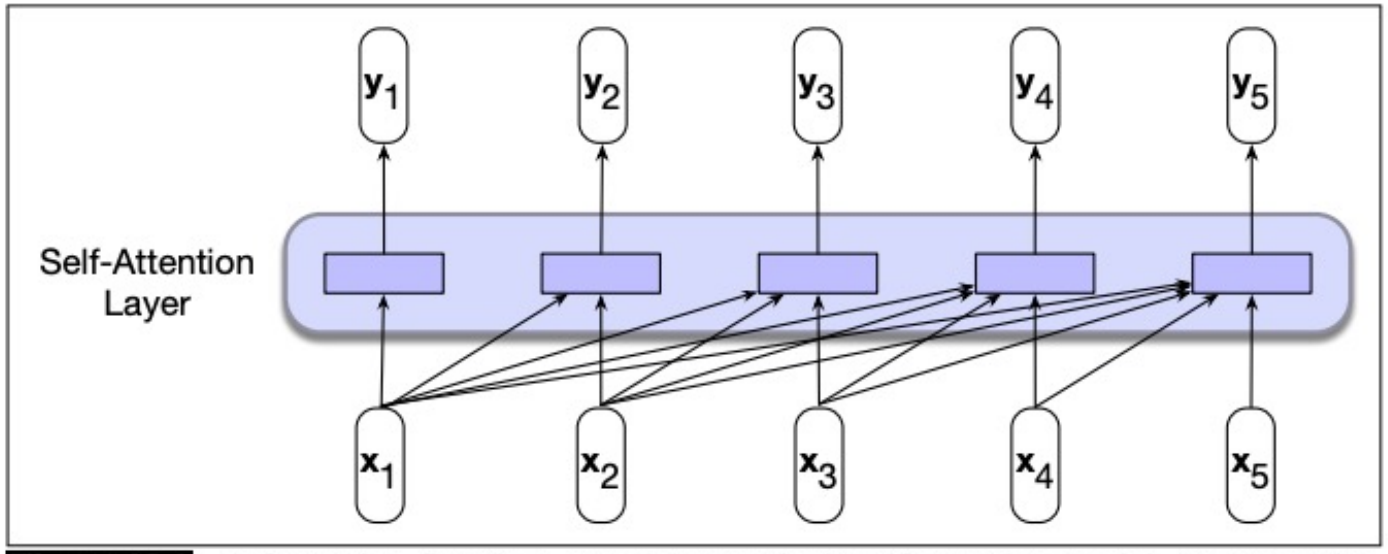


Figure 10.1 Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

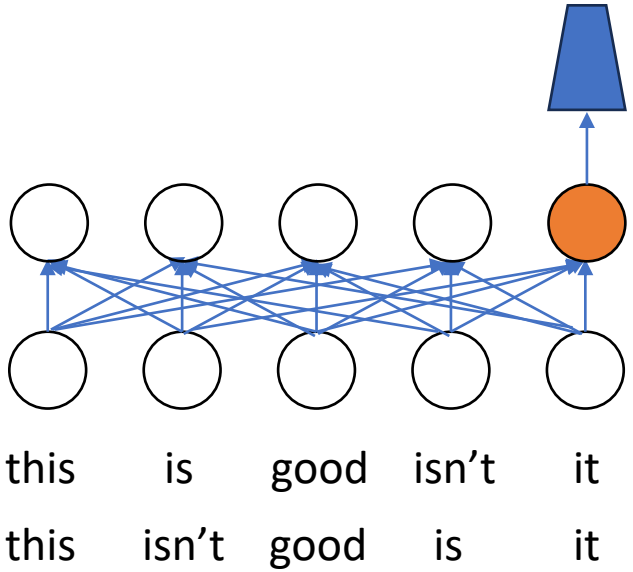
Did we miss anything?

- Consider sentiment classification

  “this is good, isn’t it?”

  vs

  “this isn’t good, is it?”
- Same hidden state for classification
- We’re agnostic to word orders!



Positional Encoding

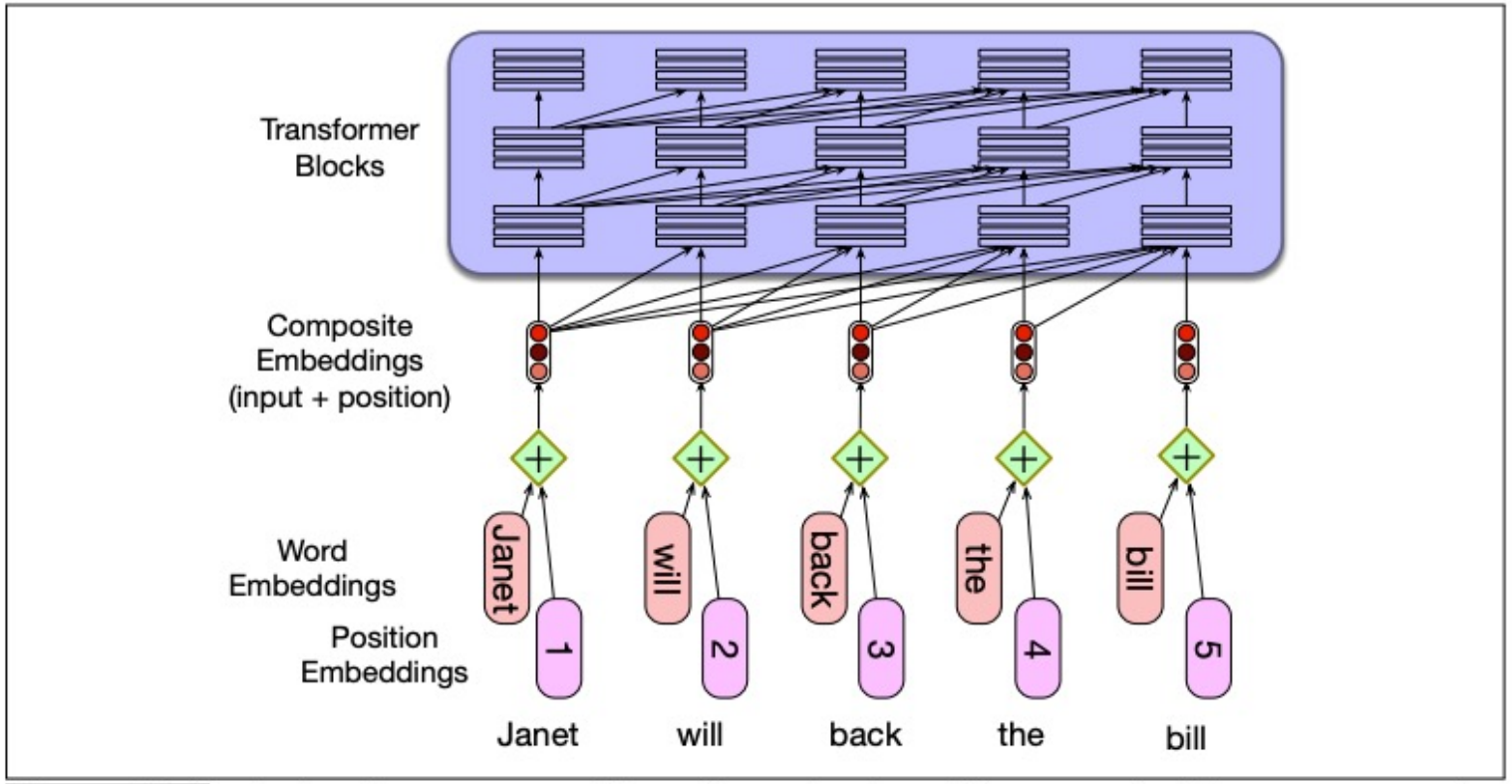


Figure 10.6 A simple way to model position: simply adding an embedding representation of the absolute position to the input word embedding to produce a new embedding of the same dimenionality.

Illustration from https://web.stanford.edu/~jurafsky/slp3/10.pdf

Absolute Position Encoding

Neural Information Processing Systems
https://papers.neurips.cc › paper › 7181-attentio...

Attention is All you Need

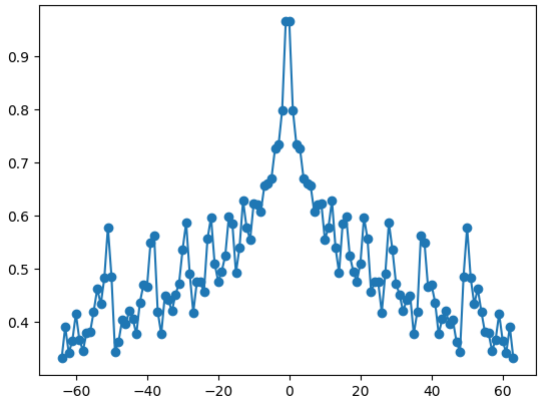
by A Vaswani · Cited by 92585 — **We** propose **a** new simple network architecture, the Transformer, based solely on **attention** mechanisms, dispensing with recurrence and…
11 pages

- Each position with a encoding vector
- E.g, sinusoidal

$$\vec{p}_i = \left[\sin \omega_1 i\,, \cos \omega_1\, i, \ldots, \sin \omega_{d/2} i, \cos \omega_{d/2}\, i\right]$$

- Why it helps

$$\vec{p}_i \cdot \vec{p}_j = \sum_{s=1}^{d/2} \cos \omega_s (i-j)$$

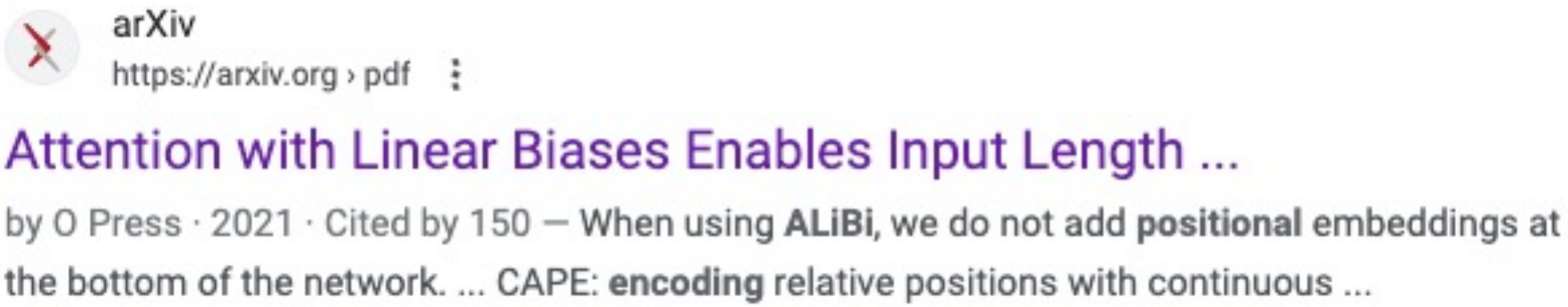


Imposes decaying for distant positions!

Relative Position Encoding

- Some function of $i - j$
- E.g., ALiBi's raw score

$$s_{i,j} = \frac{q_i \cdot k_j}{\sqrt{d}} - c|i - j|, c > 0$$



- Encourage to pay more attention to nearby tokens

RoPE: Another Relative Position Encoding

- Rotate q_i by $i \times \theta$, k_j by $j \times \theta$
- So if $|i - j|$ big, their inner product is small