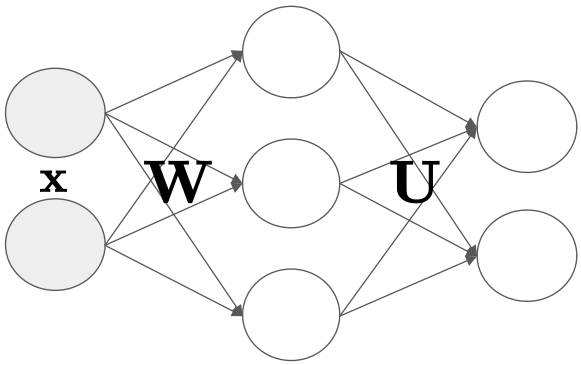
Transformers

LOT School 2024

Grzegorz Chrupała

Beyond multilayer perceptrons



Limitations?
Alternatives?

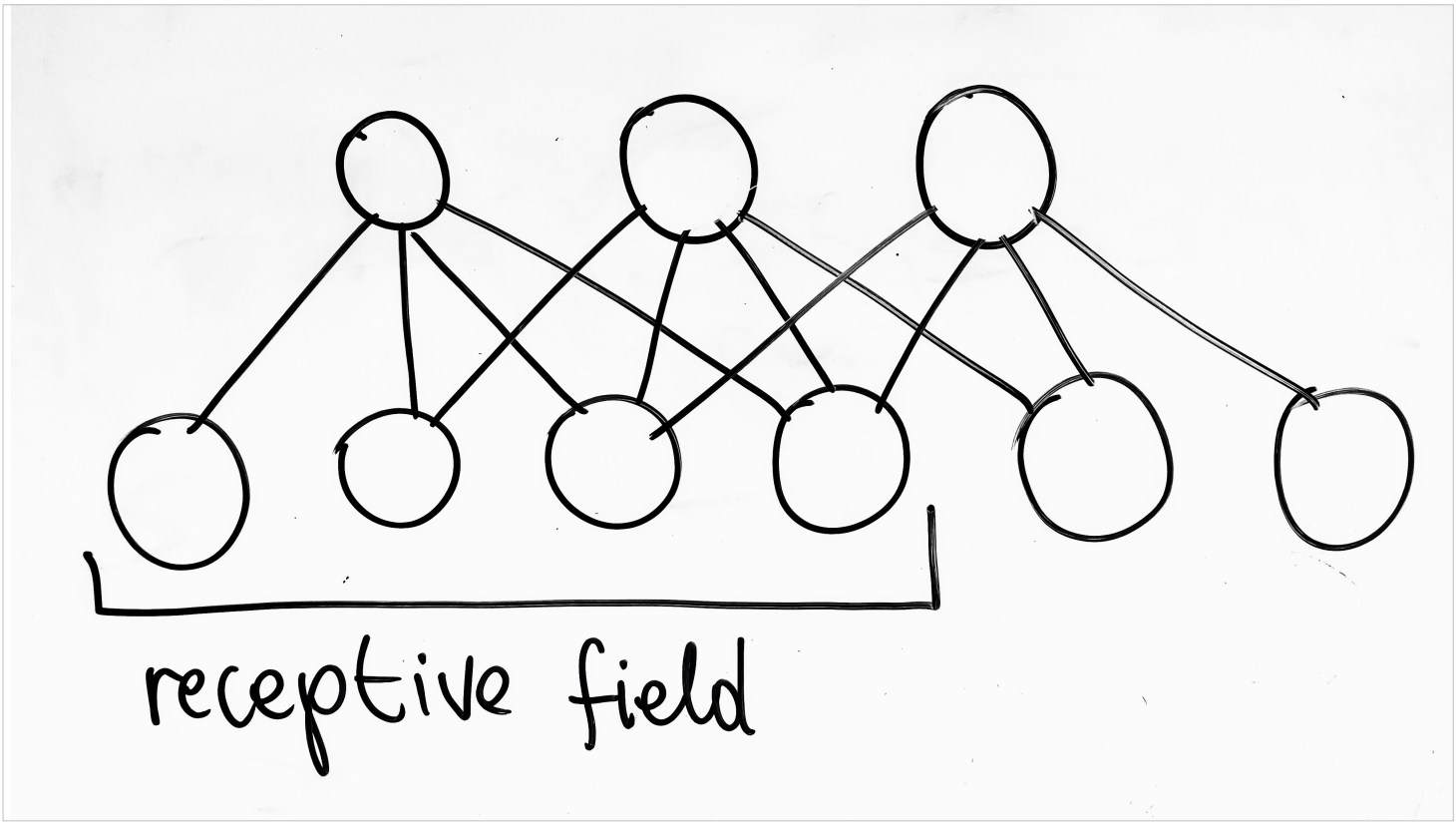
Limitations of MLPs:

- Fixed-size vectors as input and output. In many cases it's not convenient or effective to convert everything to vectors of a fixed dimension.
- Full conectivity between adjacent layers, no connectivity within layers. Not always the most suitable type of architecture.
- Three main alternatives:
  - Convolutional
  - Recurrent
  - Transformer

A related limitation is that for an MLP all dimensions of the input vector are completely independent.

If the input vector contains pixel values of a photograph, or of a spectrogram, then the same pattern but shifted in space or time, is treated by the MLP as a completely different pattern.

All three alternative architctures address this problem to various degrees and in various ways.



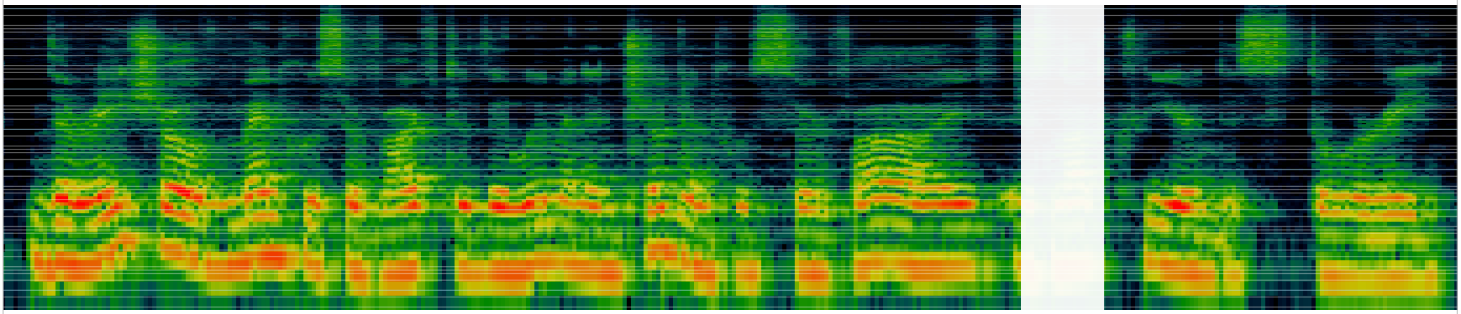
The main feature of a convolutional network is **constrained** connectivity.

Neurons in layer *n* are connected only to a subset of neurons in layer *n*-1, namely those that lie within a small patch: the receptive field. Additionally, the weights are **shared** between the units in layer *n*.

This is inspired by the connectivity in vertebrate retina. It is also designed to capture the **spatial correlation** present in images of the natural world.

Similar types of structure often also exists in the time dimension, as is the case with audio signals.

Combined with max pooling of activations, the connectivity of a CNN enable the detection of shifted patterns: **translation invariance**.



The ratio of men who survive to the women and children who survive is/are not clear in this story.

5

Language has a hierarchical syntactic structure. When this structure is linearized, it results in correlations across long distances.

If we want to reconstruct the blanked out part in this utterance, we need to keep track of the complete utterance and specifically the beginning of it, in order to account for grammatical number agreement between subject and verb.

A neural model with locally constrained connectivity, such as a convolutional architecture, is not a natural match for this task.

COGNITIVE SCIENCE **14,** 179–211 (1990)

Finding Structure in Time

JEFFREY L. ELMAN
University of California, San Diego

Time underlies many interesting human behaviors. Thus, the question of how to represent time in connectionist models is very important. One approach is to represent time implicitly by its effects on processing rather than explicitly (as in a spatial representation). The current report develops a proposal along these lines first described by Jordan (1986) which involves the use of recurrent links in order to provide networks with a dynamic memory. In this approach, hidden unit patterns are fed back to themselves; the internal representations which develop thus reflect task demands in the context of prior internal states. A set of simulations is reported which range from relatively simple problems (temporal version of XOR) to discovering syntactic/semantic features for words. The networks are able to learn interesting internal representations which incorporate task demands with memory demands; indeed, in this approach the notion of memory is inextricably bound up with task processing. These representations reveal a rich structure, which allows them to be highly context-dependent, while also expressing generalizations across classes of items. These representations suggest a method for representing lexical categories and the type/token distinction.

Machine Learning, 7, 195–225 (1991)

Distributed Representations, Simple Recurrent Networks, and Grammatical Structure

JEFFREY L. ELMAN (ELMAN@CRL.UCSD.EDU)
Departments of Cognitive Science and Linguistics, University of California, San Diego

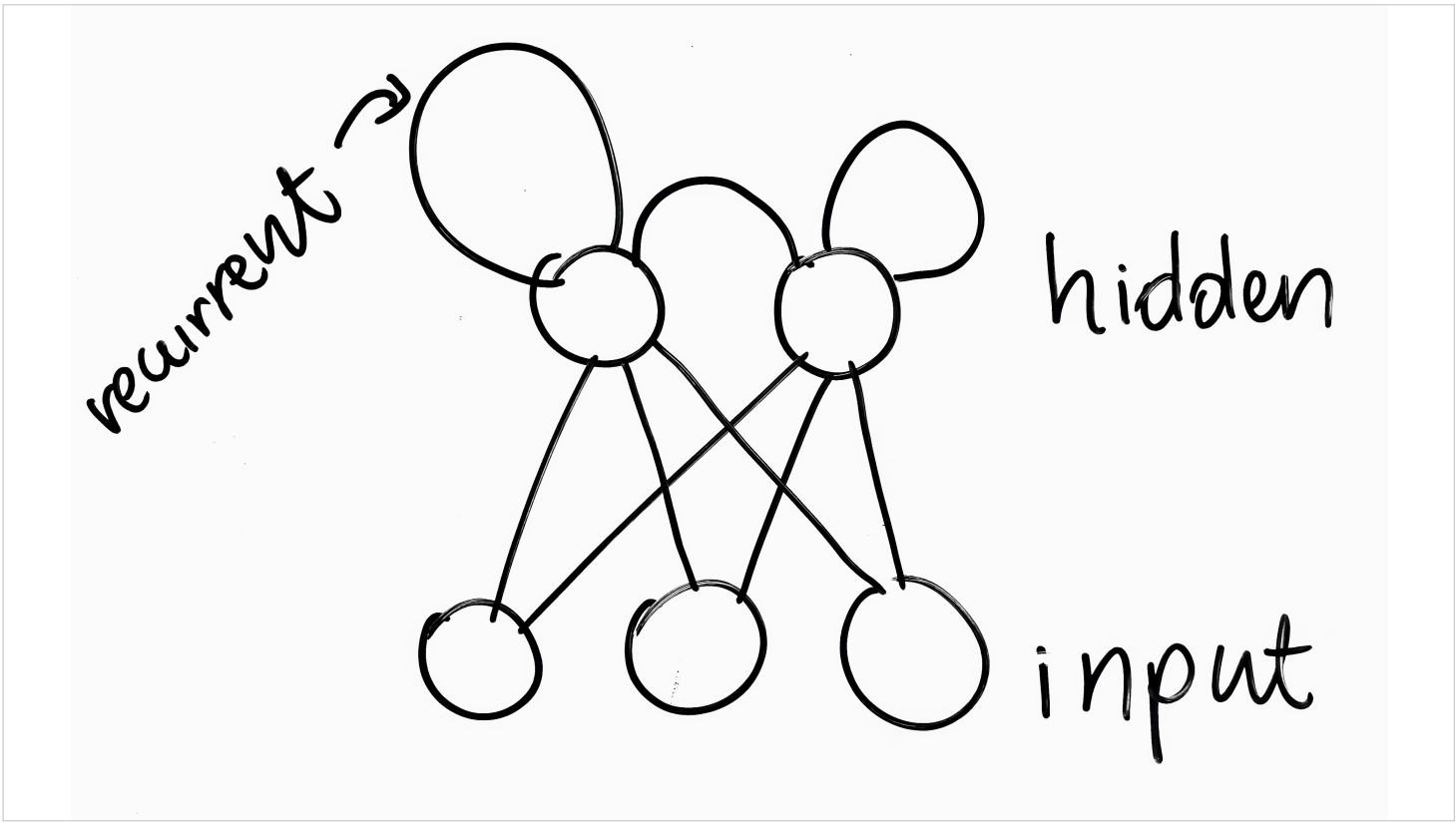
Abstract. In this paper three problems for a connectionist account of language are considered:

1. What is the nature of linguistic representations?
2. How can complex structural relationships such as constituent structure be represented?
3. How can the apparently open-ended nature of language be accommodated by a fixed-resource system?

Using a prediction task, a simple recurrent network (SRN) is trained on multiclausal sentences which contain multiply-embedded relative clauses. Principal component analysis of the hidden unit activation patterns reveals that the network solves the task by developing complex distributed representations which encode the relevant grammatical relations and hierarchical constituent structure. Differences between the SRN state representations and the more traditional pushdown store are discussed in the final section.

Jeffrey L. Elman (1948-2018) was interested in modeling the temporal / sequential dimension of cognitive processes, and specifically human language.

He came up with a simple recurrent network (SRN) architecture.



The core of an SRN is a network with an input layer and a hidden layer. In addition to the connections between the input and hidden layer there are also recurrent connections from the hidden layer at time t-1 to the hidden layer at time t.

The input to the input layer changes at each time step. The activation in the hidden layer evolves according to the dynamics encoded in the weights.

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t)$$

$\mathbf{h}_t$ – hidden state at time t

$\mathbf{x}_t$ – input at time t

W – hidden-to-hidden connections

U – input to hidden connections

σ – nonlinear activation function

This basic model can be augmented with additional components to generate output at the each times step, or at the end of the sequence.

How can we alter this network to enable it to make predictions at each time step?

We can add another weight matrix which maps the activations of the hidden layer to the output vectors.

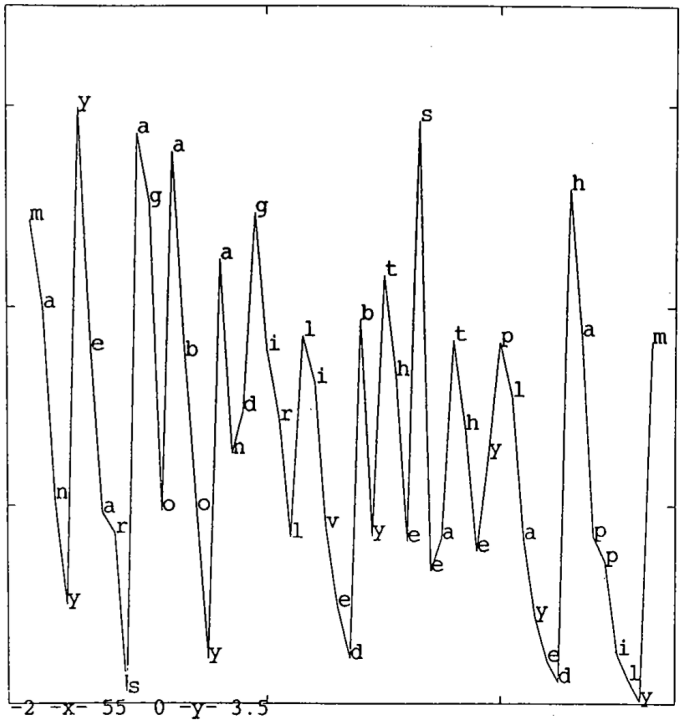


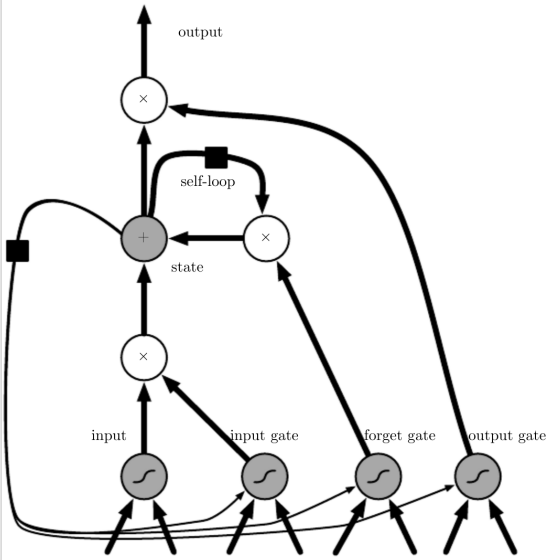
Figure 6. Graph of root mean squared error in letter-in-word precition task.

Elman trained an SRN by feeding it letters of a sentence in sequence and at each point getting it to predict the next letter.

The plot shows the error associated with these predictions for an example sentence.

The network shows a clear pattern of increased uncertainty for word initial letters. This indicates that the network implicitly learned something about words from this basic task of predicting letters.

The same core idea underlies currently popular Large Language Models.



LONG SHORT-TERM MEMORY

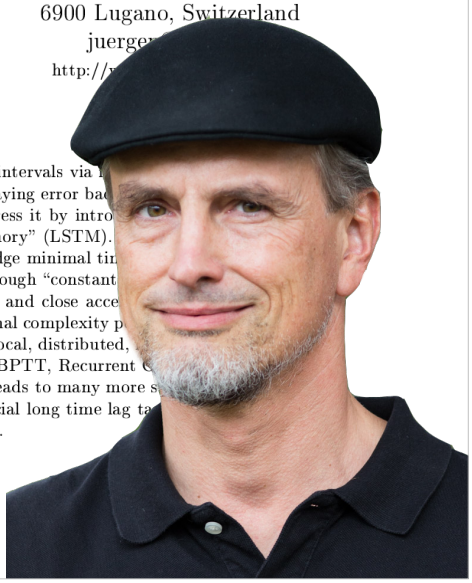
Neural Computation 9(8):1735–1780, 1997

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juerge
http://

Abstract

Learning to store information over extended time intervals via
takes a very long time, mostly due to insufficient, decaying error ba
Hochreiter's 1991 analysis of this problem, then address it by intro
gradient-based method called "Long Short-Term Memory" (LSTM).
where this does not do harm, LSTM can learn to bridge minimal ti
discrete time steps by enforcing *constant* error flow through "constant
special units. Multiplicative gate units learn to open and close acce
flow. LSTM is local in space and time; its computational complexity p
is $O(1)$. Our experiments with artificial data involve local, distributed,
pattern representations. In comparisons with RTRL, BPTT, Recurrent
Elman nets, and Neural Sequence Chunking, LSTM leads to many more s
learns much faster. LSTM also solves complex, artificial long time lag ta
been solved by previous recurrent network algorithms.



SRNs worked on toy problems as a proof of concept, but they turned out hard to train in practice, due to the issues with the erratic behavior of the gradient computation for long sequences.

The Long Short Term Memory architecture solved these problems and went on to be used in many successful applications such speech recognition and optical character recognition.

Recurrent networks have been largely superseded in practical applications by the transformer, but remain popular for cognitive modeling, because they process temporal signal sequentially and not in parallel.

SRNs, LSTMs and other recurrent networks process inputs sequentially.

What are the advantages of this architecture?

What are its drawbacks?

Sequential processing of temporal data is arguably more plausible as a cognitive model.

The processing time is linear as a function of the length of the sequence.

On the downside, all data in an RNN passes through the bottleneck of the hidden activation vector.

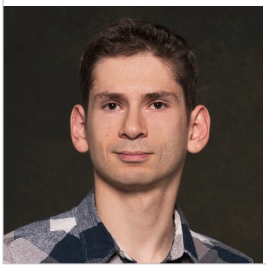
RNNs are also hard to scale to large datasets, as it makes it hard to make uses of parellel processing.

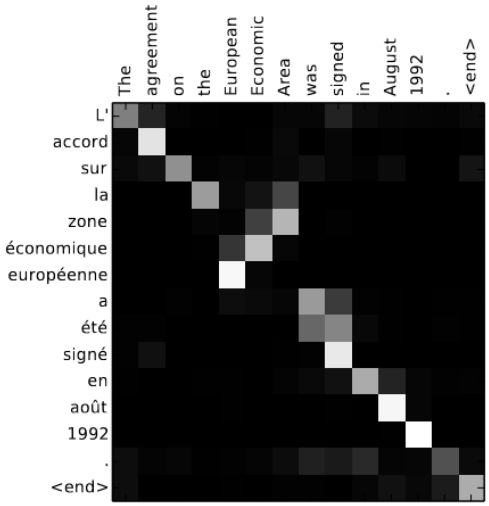
Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal





(a)

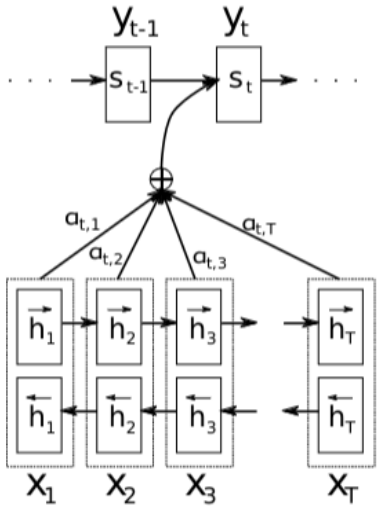


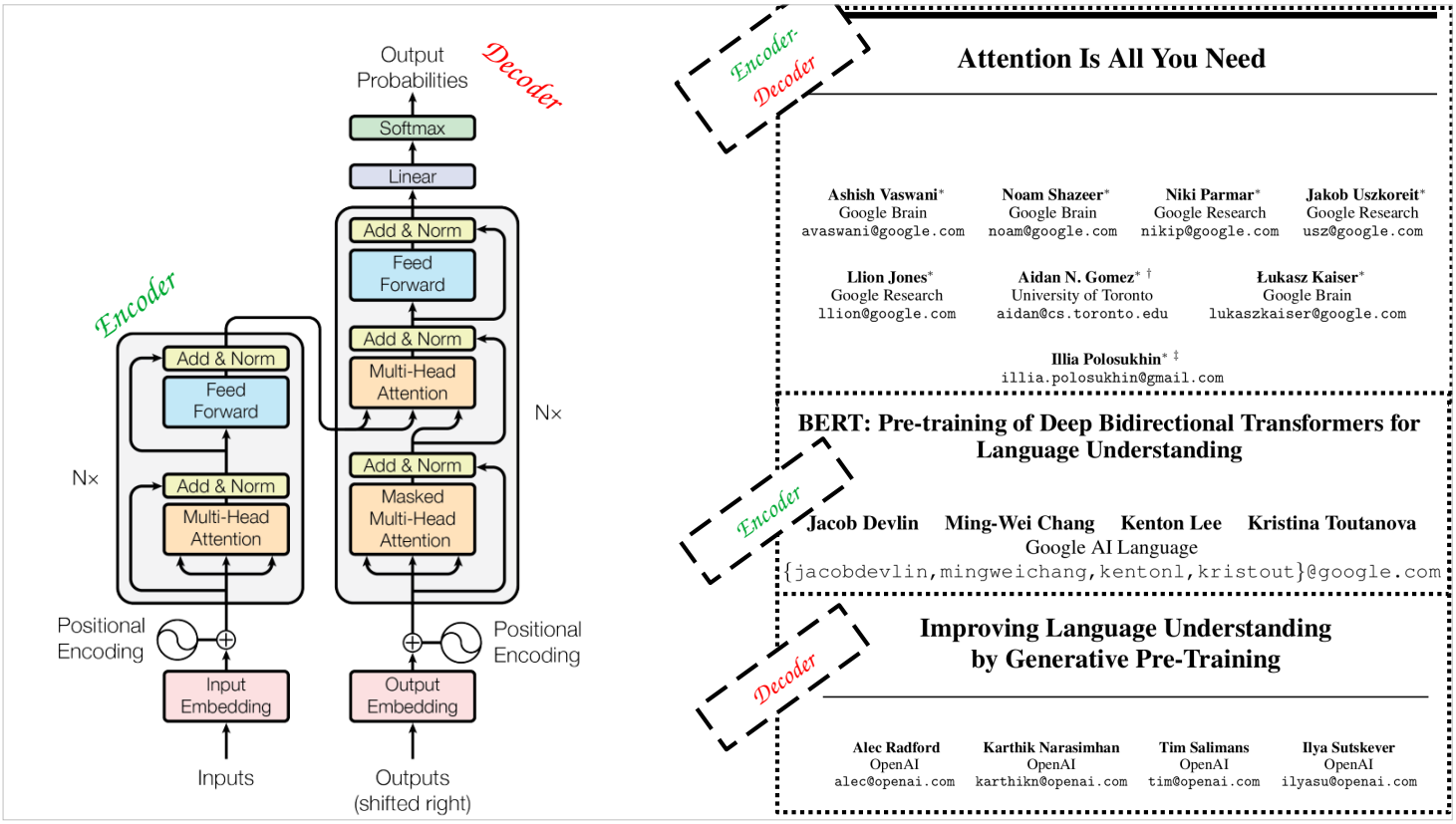
Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

One idea to address the bottleneck problem has been to use attention.

Attention is a mechanism to access all hidden activation at all timesteps and pool them via a weighted sum.

The weights of the pooling operation loosely correspond to the the degree to which the model *attends to* each specific timestep.

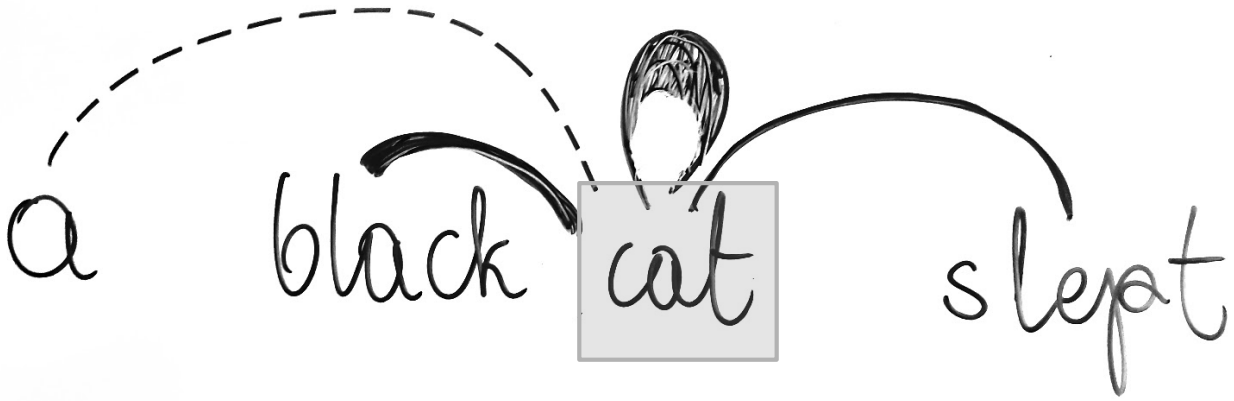
Eventually, the idea of attention led to the transformer architecture.



The Attention is All You Need paper introduced the transformer applied to the task of translation.

Here the transformer consists of two parts: the encoder module captures and representes the information in the source sentence, while the decoder generates the target sentence.

The core of each of these two modules is the self-attention mechanisms (see following slides). Additionally this transformer features cross-attention which maps from the source to the target words.



The key idea of self-attention is that it encodes pairwise relations between elements in a set of inputs.

Specifically, each element in the set is represented as a weighted sum of the representations of all the elements in the set, and this is repeated along a stack of layers.

Additionally, there are multiple such sets of weights (one set per attention head) and the weighted sums from each head are combined into a joint representation. Thus each attention head can learn to specialized in encoding different types of relations.

$$\mathbf{q}_i = \mathbf{W}_Q \mathbf{x}_i + \mathbf{b}_Q$$
$$\mathbf{k}_i = \mathbf{W}_K \mathbf{x}_i + \mathbf{b}_K$$
$$\mathbf{v}_i = \mathbf{W}_V \mathbf{x}_i + \mathbf{b}_V$$

$$\alpha_{ij} = \text{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{(d)}}\right)$$

$$z_i = \sum_{h=1}^{H} \sum_{j=1}^{N} \alpha_{ij}^h \mathbf{W}_O \mathbf{v}_j^h + \mathbf{b}_O + \mathbf{x}_i$$

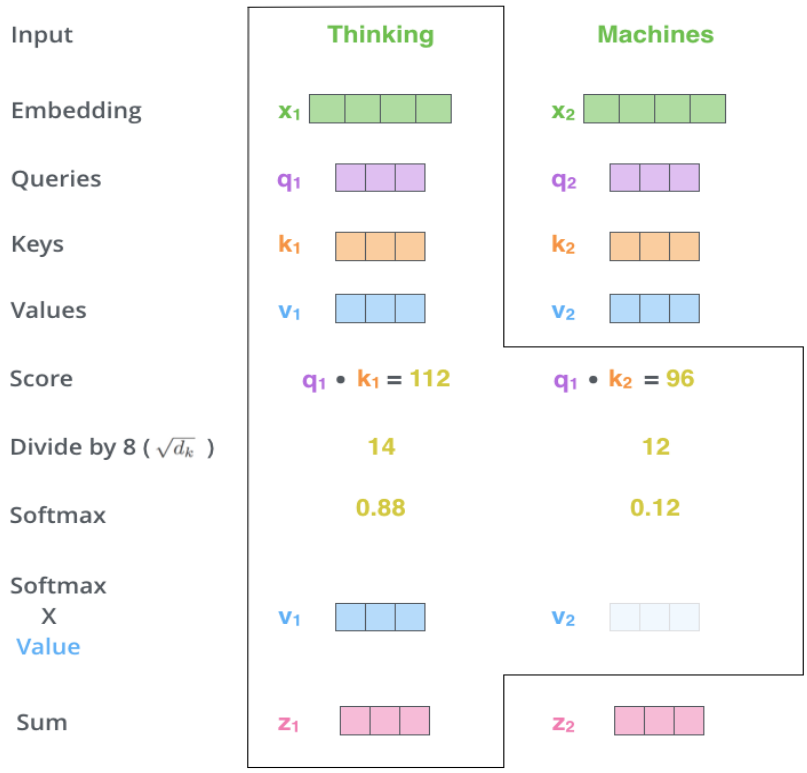
16

For one transformer layer, first the input items are linearly projected into three separate versions or views: query, key and value.

Then, for each item *i*, the incoming weight from each item *j* is computed, based on how much the query for *i* and the key for *j* match (according to scaled dot product). The head-specific representation of the ith item is given by the weighted sum of all the items in the set (inner sum).

Then, in order to get the representation z_i, the weighted sums from all the heads are added (outer sum).

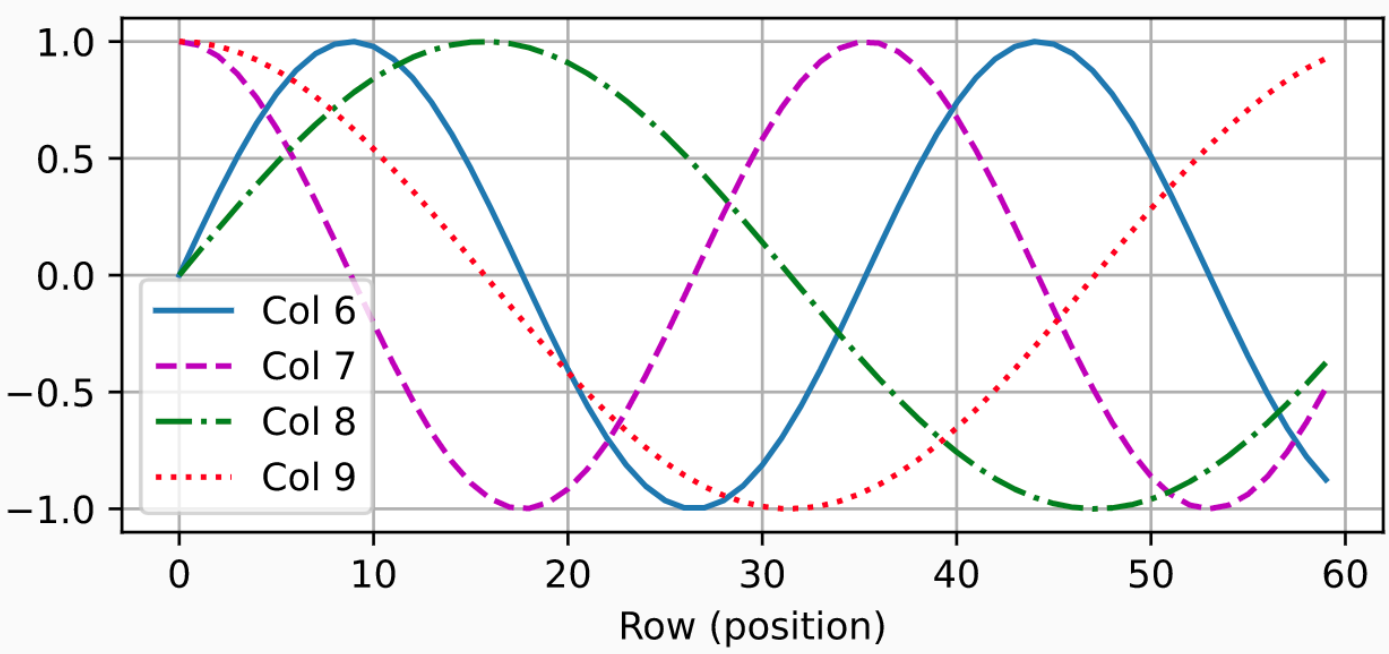
Finally, two linear transformations with a GELU activation function in between are applied to every z_i to produce an output representation.



Visualization from
https://jalammar.github.io/illustrated-transformer/

What about the ordering of elements?

How can it be captured?



The core transformer does not assume or represent any ordering of the input items: they are simply treated as an unordered set.

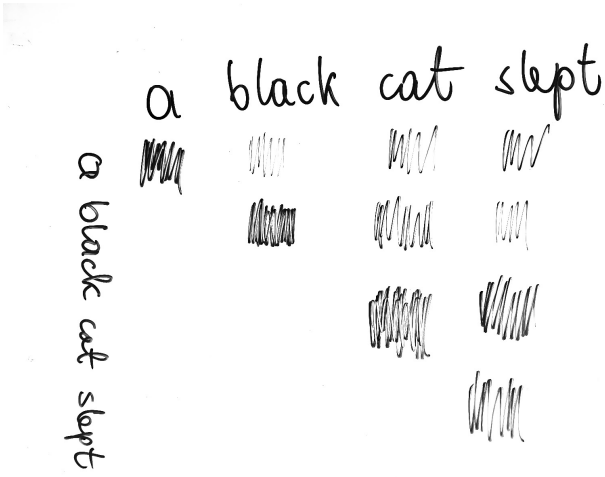
Ordering is added via a specialized encoding, known as positional encoding. There are several variants: the original are absolute positional encodings via a set of sinusoidal functions each with a different period.

The resulting vector encoding a position in the sequence is summed to the input vector.

Plot from https://paperswithcode.com/method/absolute-position-encodings

Full attention - encoder

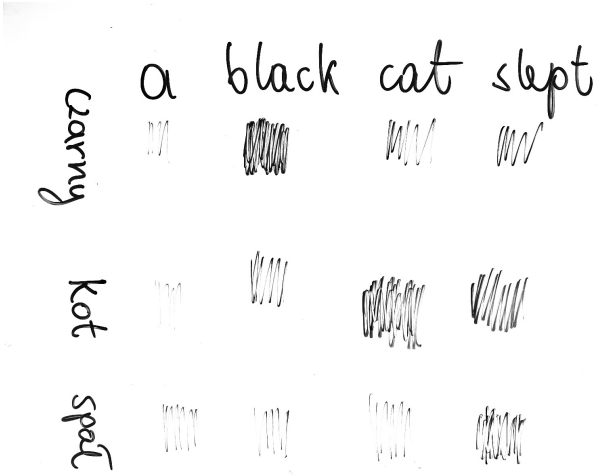
Masked attention - decoder



In the case of the decoder, attention is masked. This means that the representation of each item does not use the items coming after it in the sequence.

This allows for sequential, word-by-word generation of output.

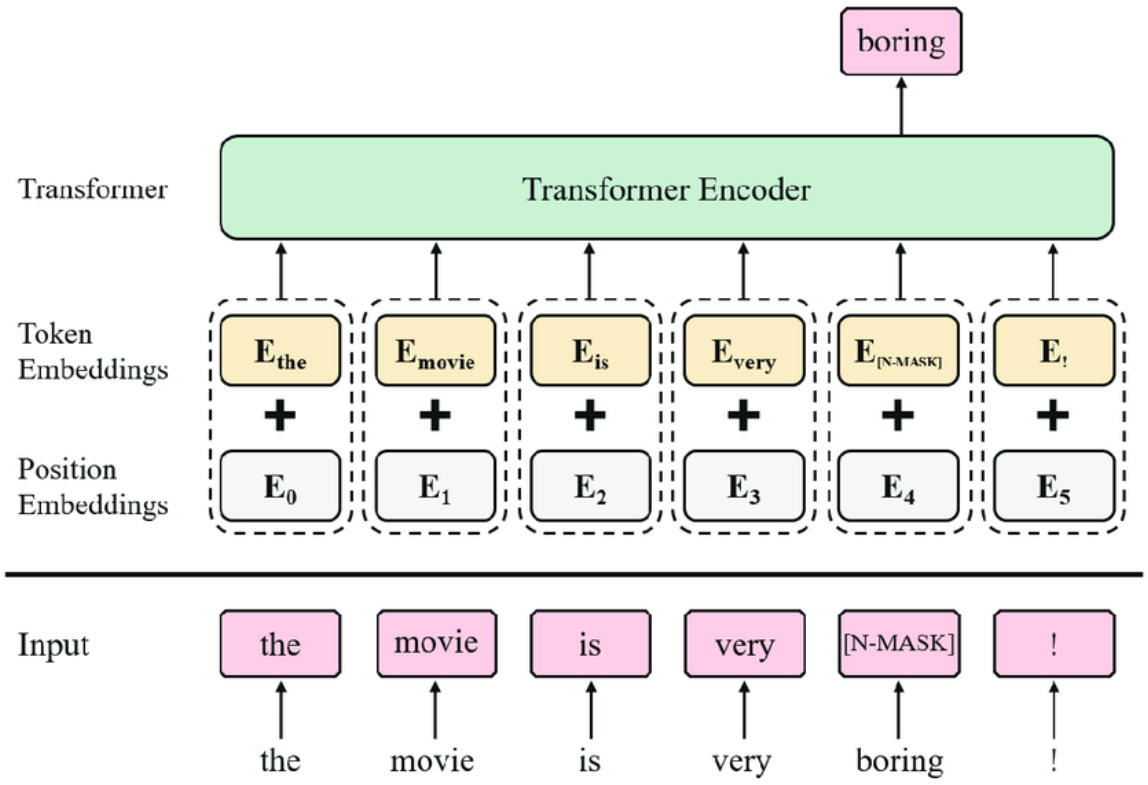
Full attention - from decoder to encoder



In the full encoder-decoder transformer there is also an attention mechanism operating from the decoder to the encoder.

In the canonical case of translation, this allows the transformer to selectively consider the relevant portions of the source sequence while generating the target sequence.


BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
Encoder
Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com



The core idea behind BERT is to train a transformer encoder to perform the cloze task, in order to learn generally useful sentence representations.

The advantage is that no labeled data is necessary: this is known as self-supervision.

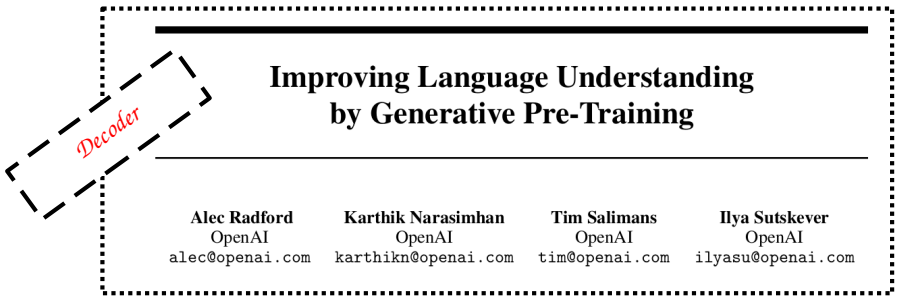
The original paper also uses an additional sentence pair classification task for extra supervision, but this is not crucial.

Visualization from https://ankur3107.github.io/blogs/masked-langauge-modeling/

How can BERT-style representations be used?

By using them as features in an external classifier.

By fine-tuning the BERT model on a down-stream task.



The core idea here is to have a generative model. Unlike with BERT, the point is not the representations but the actual ability to generate text.

How are generative language models different from BERT-style models in the way they are used?

With generative language models, the typical scenario is to represent each problem as a text completion task.

The language model may be additonally fine-tuned in a supervised way, or via reinforcement learning.