Sequence Learning

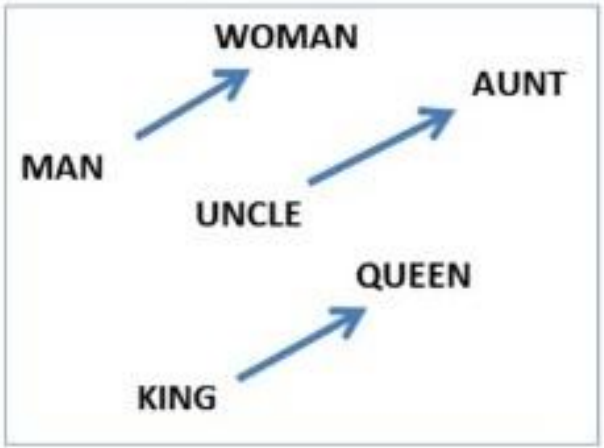
Grigory Sapunov  CTO / Intento

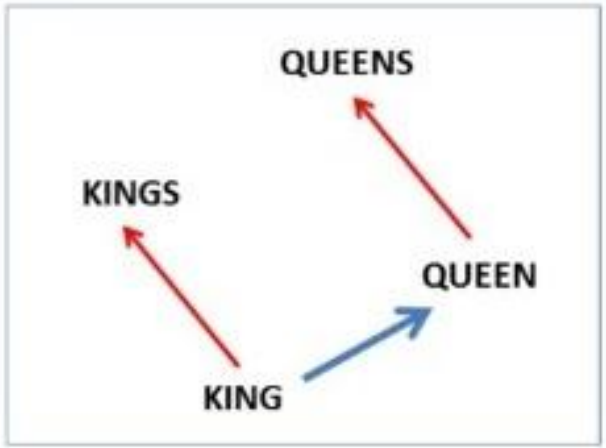
HSE / #12 / 04.12.2018

Multimodal Learning

Example: Semantic Spaces (word2vec, GloVe)

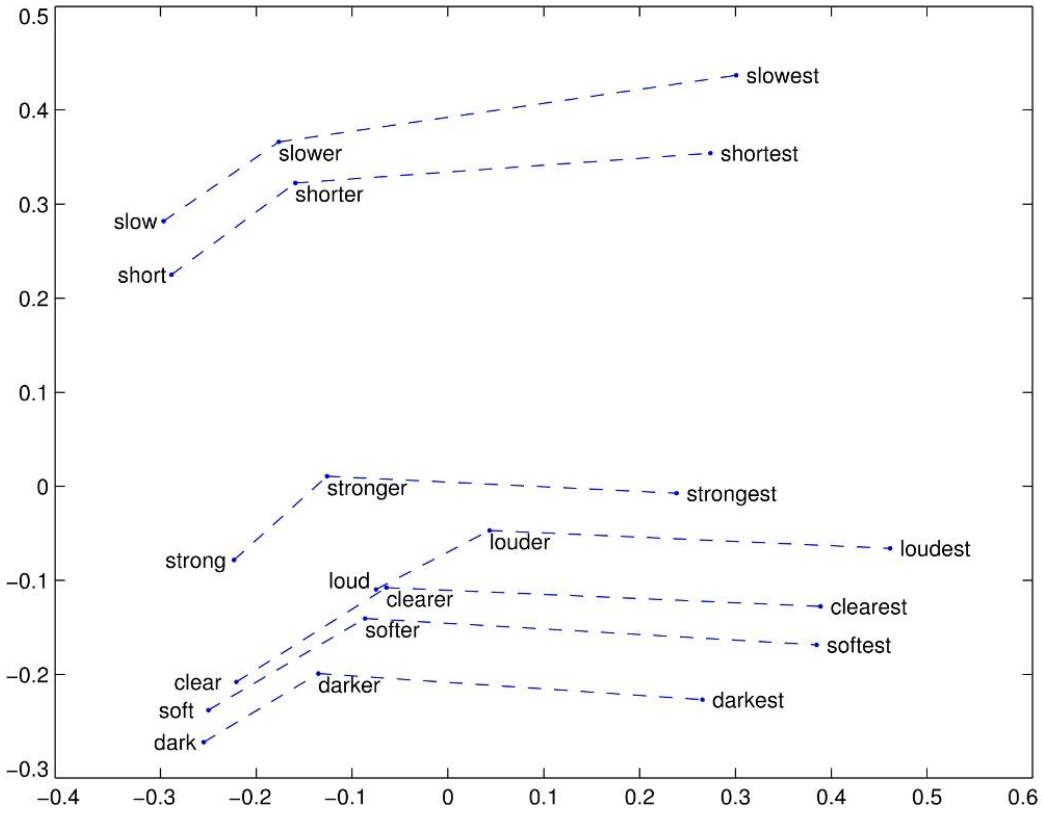
vec(“man”) – vec(“king”) + vec(“woman”) = vec(“queen”)





https://code.google.com/archive/p/word2vec/

Example: Semantic Spaces (word2vec, GloVe)



http://nlp.stanford.edu/projects/glove/

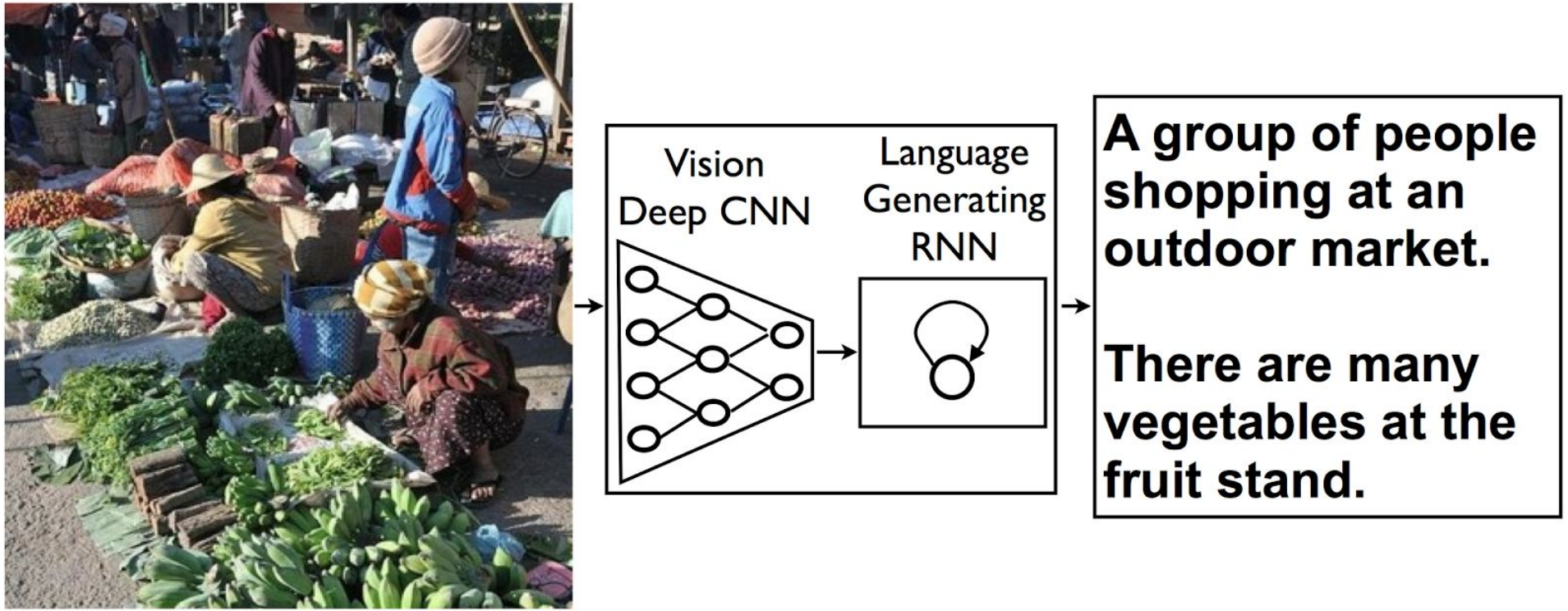
Encoding semantics

Using word2vec instead of word indexes allows you to better deal with the word meanings (e.g. no need to enumerate all synonyms because their vectors are already close to each other).

But the naive way to work with word2vec vectors still gives you a “bag of words” model, where phrases “The man killed the tiger” and “The tiger killed the man” are equal.

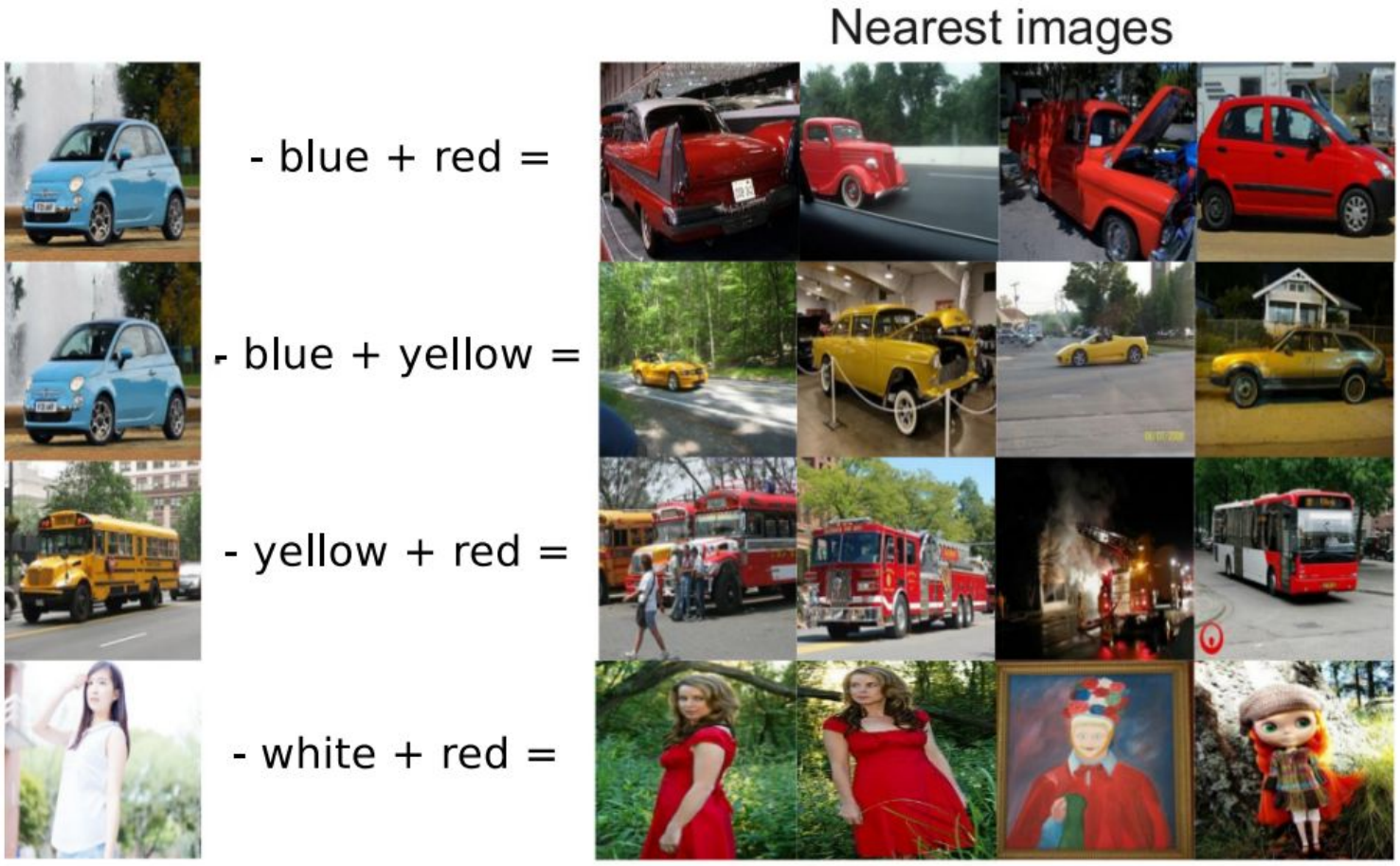
Need models which pay attention to the word ordering: paragraph2vec, sentence embeddings (using RNN/LSTM), even World2Vec (LeCunn @CVPR2015).

Image caption generation

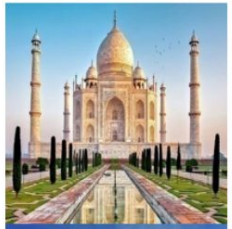


http://arxiv.org/abs/1411.4555 “Show and Tell: A Neural Image Caption Generator”

Example: More multi-modal learning



Nearest Images



- day + night =

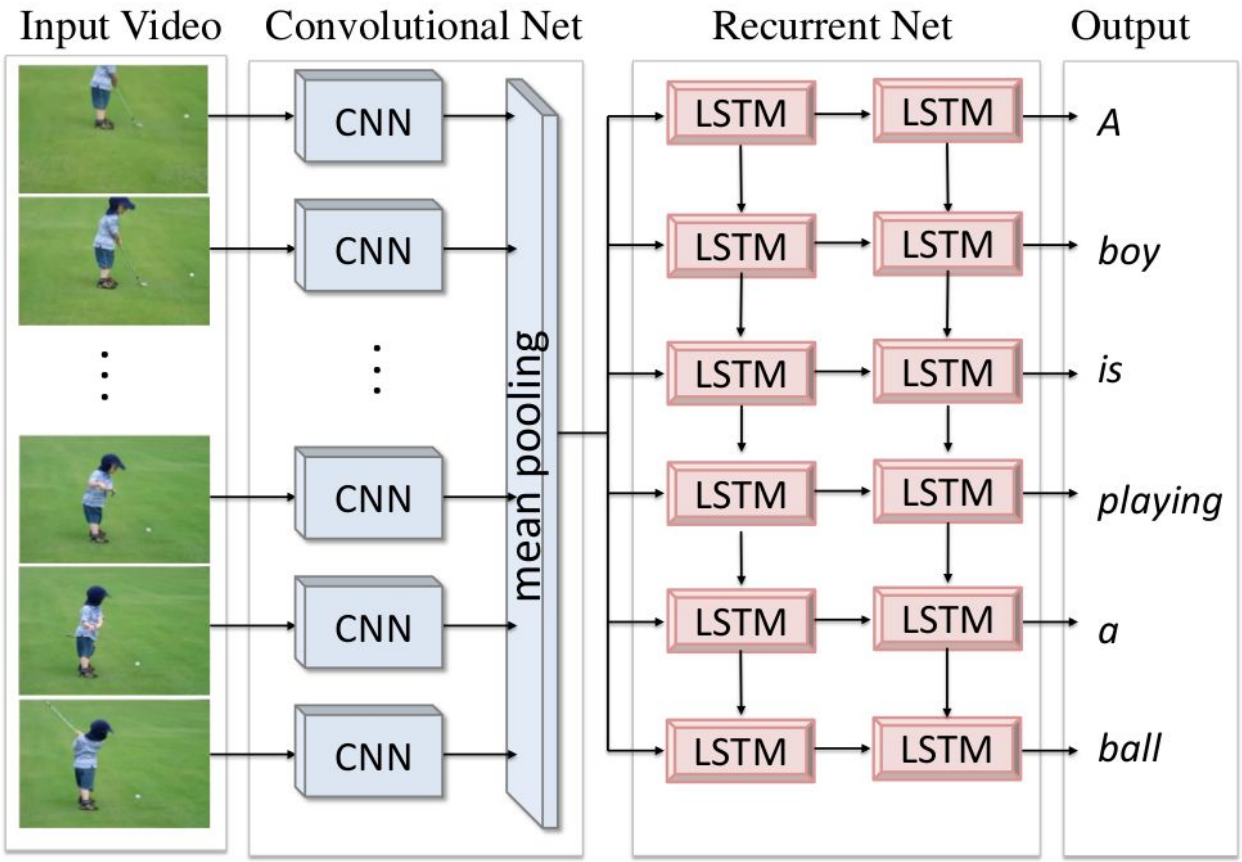
- flying + sailing =

- bowl + box =

- box + bowl =

(Kiros, Salakhutdinov, Zemel, TACL 2015)

Video description



https://www.cs.utexas.edu/~vsub/

Image description

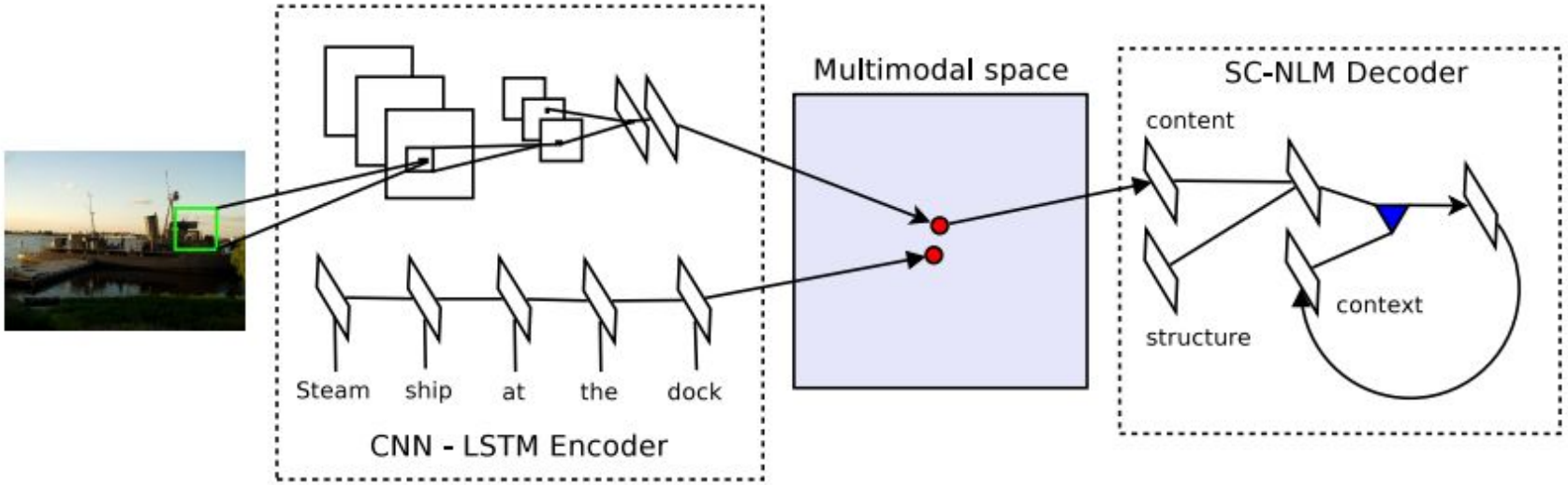


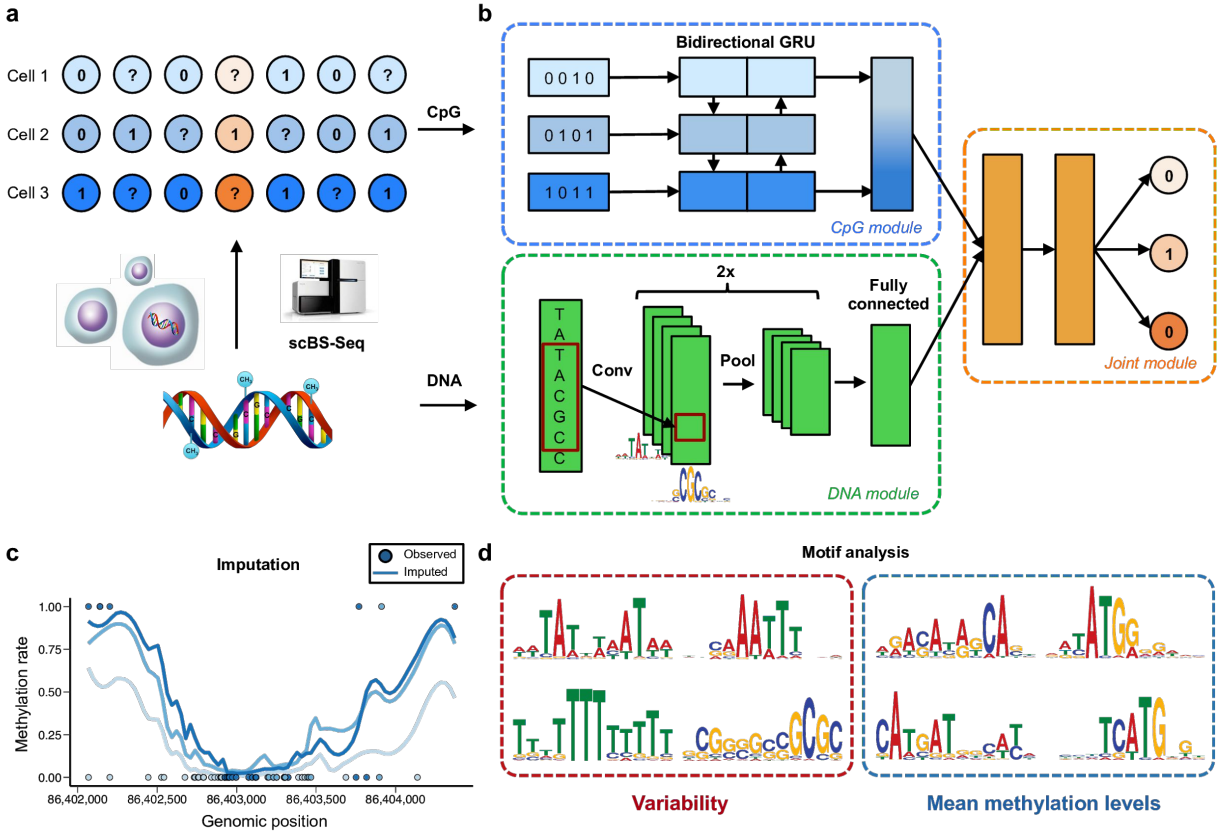
Figure 2: **Encoder:** A deep convolutional network (CNN) and long short-term memory recurrent network (LSTM) for learning a joint image-sentence embedding. **Decoder:** A new neural language model that combines structure and content vectors for generating words one at a time in sequence.

http://arxiv.org/abs/1411.2539 Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models

Example: Image generation by text

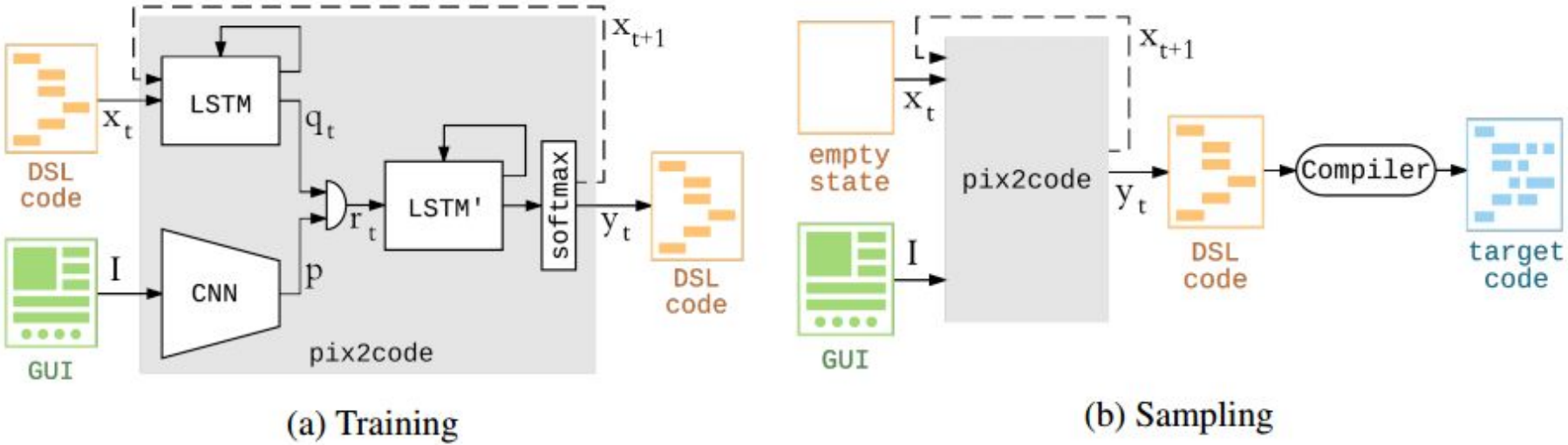
StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, https://arxiv.org/abs/1612.03242

Mixing different bioinformatics signals



Accurate prediction of single-cell DNA methylation states using deep learning
http://biorxiv.org/content/early/2017/02/01/055715

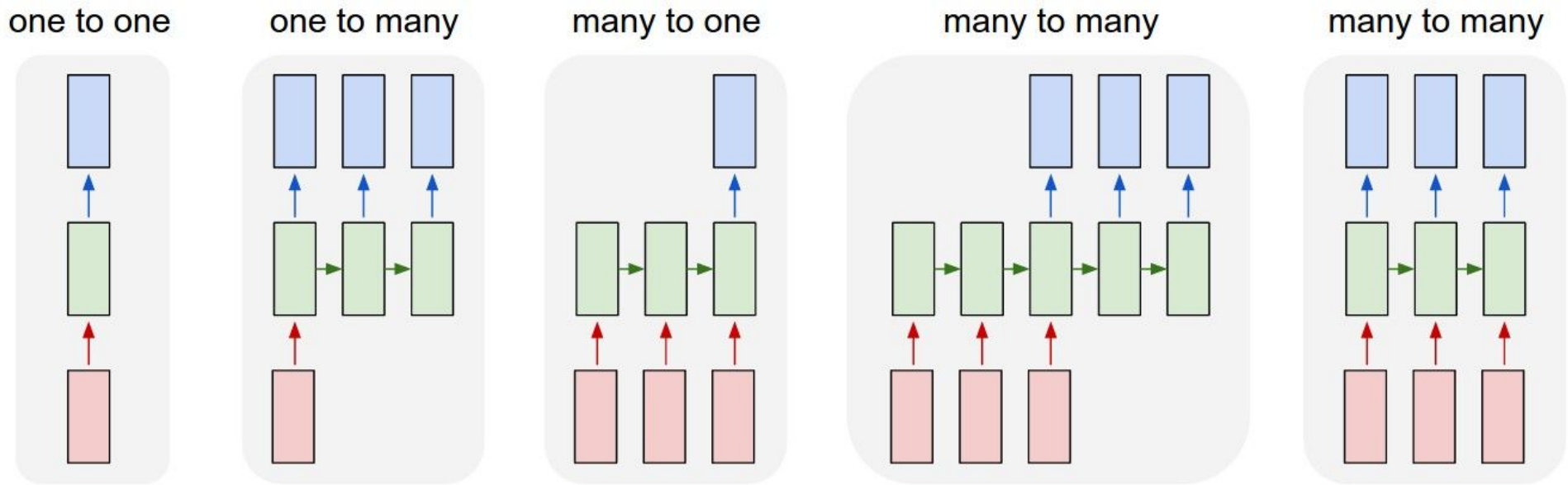
Example: Code generation by image



pix2code: Generating Code from a Graphical User Interface Screenshot,
https://arxiv.org/abs/1705.07962

Sequence Learning / seq2seq

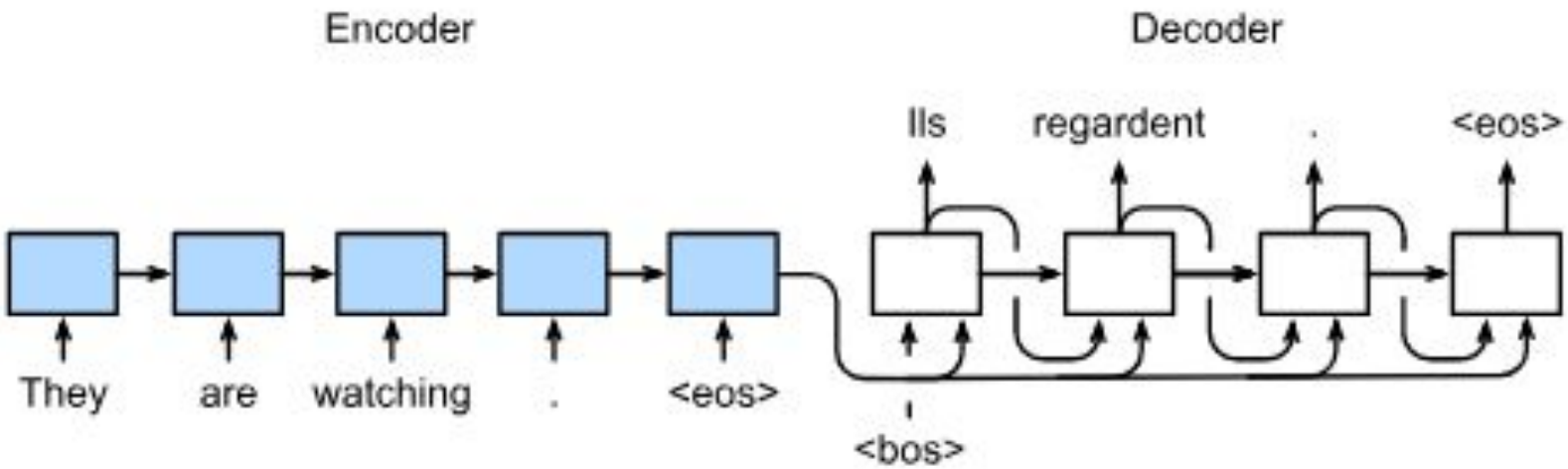
Sequence to Sequence Learning (seq2seq)



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Encoder-Decoder

Encoder-Decoder architecture



http://en.diveintodeeplearning.org/chapter_natural-language-processing/seq2seq.html

Encoder-Decoder architecture

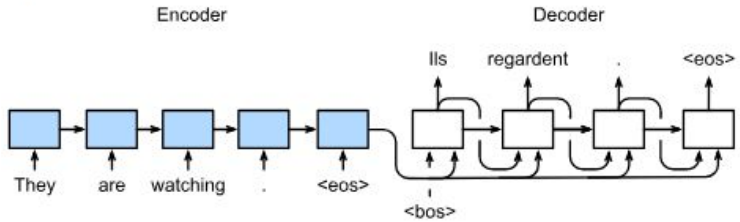
The encoder transforms the hidden state of each time step into **context variables** through custom function ***q***:

$$\boldsymbol{c} = q(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T)$$

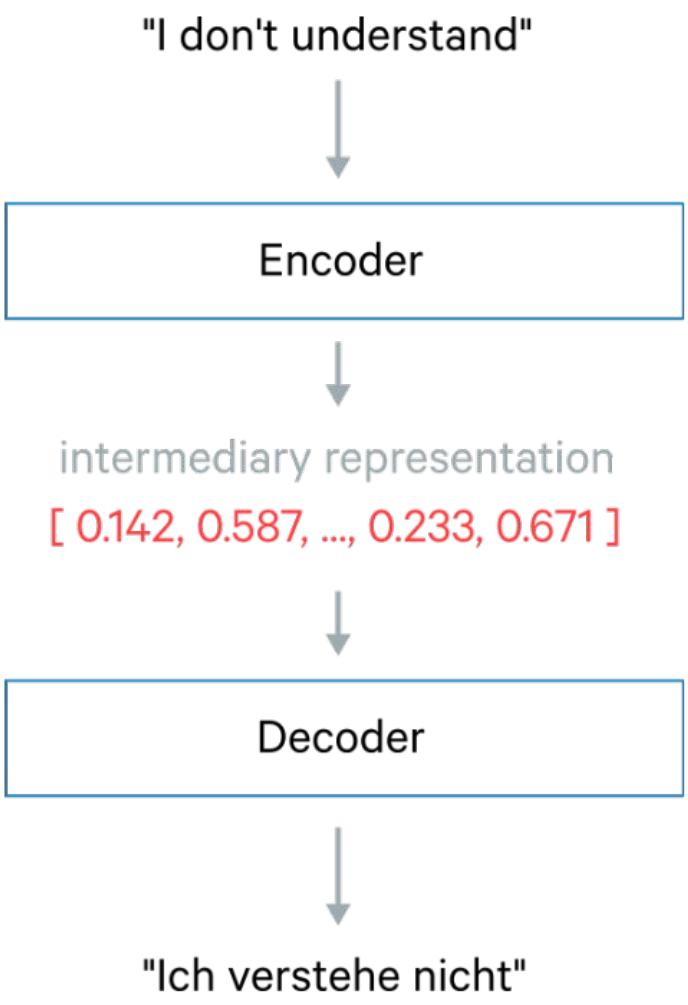
For example, when we select $q(\boldsymbol{h}_1,\ldots,\boldsymbol{h}_T)=\boldsymbol{h}_T$, the context variable is the hidden state of input sequence $\boldsymbol{h}_T$ for the final time step.

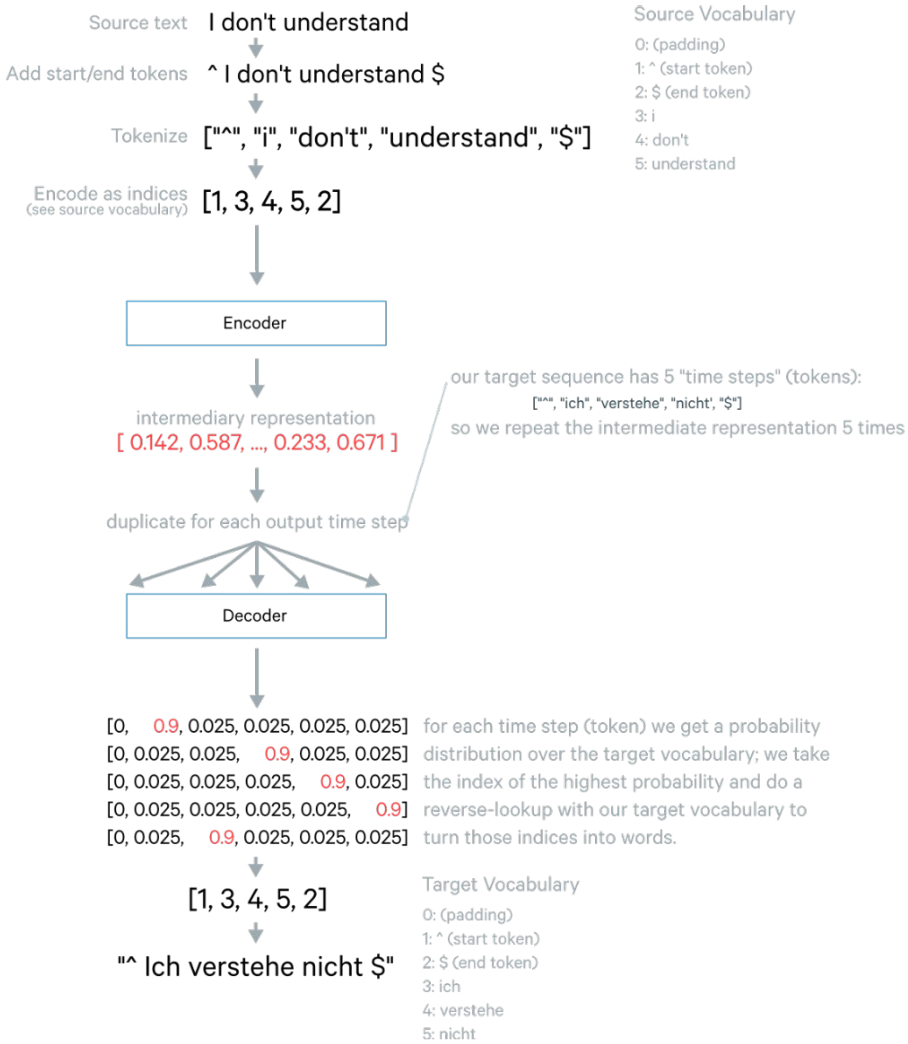
The transformation of the decoder's hidden layer:

$$\boldsymbol{s}_{t'} = g(y_{t'-1}, \boldsymbol{c}, \boldsymbol{s}_{t'-1})$$



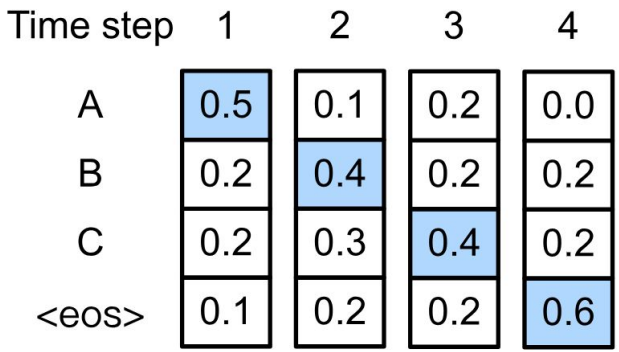
http://en.diveintodeeplearning.org/chapter_natural-language-processing/seq2seq.html


"I don't understand"
Encoder
intermediary representation
[0.142, 0.587, ..., 0.233, 0.671]
Decoder
"Ich verstehe nicht"

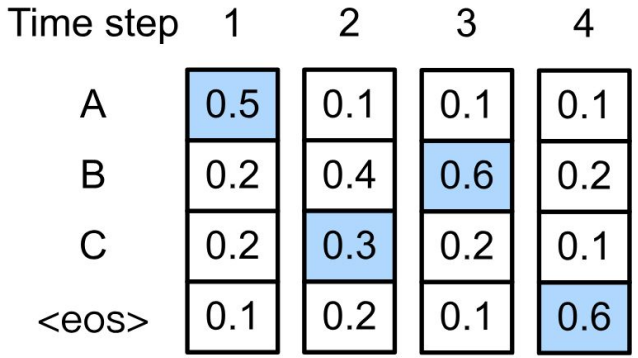

Source text
I don't understand
Add start/end tokens
^ I don't understand $
Tokenize
["^", "i", "don't", "understand", "$"]
Encode as indices
(see source vocabulary)
[1, 3, 4, 5, 2]
Source Vocabulary
0: (padding)
1: ^ (start token)
2: $ (end token)
3: i
4: don't
5: understand
Encoder
intermediary representation
[0.142, 0.587, ..., 0.233, 0.671]
duplicate for each output time step
our target sequence has 5 "time steps" (tokens):
["^", "ich", "verstehe", "nicht', "$"]
so we repeat the intermediate representation 5 times
Decoder
[0, 0.9, 0.025, 0.025, 0.025, 0.025]
[0, 0.025, 0.025, 0.9, 0.025, 0.025]
[0, 0.025, 0.025, 0.025, 0.9, 0.025]
[0, 0.025, 0.025, 0.025, 0.025, 0.9]
[0, 0.025, 0.9, 0.025, 0.025, 0.025]
for each time step (token) we get a probability distribution over the target vocabulary; we take the index of the highest probability and do a reverse-lookup with our target vocabulary to turn those indices into words.
[1, 3, 4, 5, 2]
"^ Ich verstehe nicht $"
Target Vocabulary
0: (padding)
1: ^ (start token)
2: $ (end token)
3: ich
4: verstehe
5: nicht

Decoding: Greedy search

Greedy search is not optimal. Exhaustive Search is not possible.



P = 0.5×0.4×0.4×0.6 = 0.048

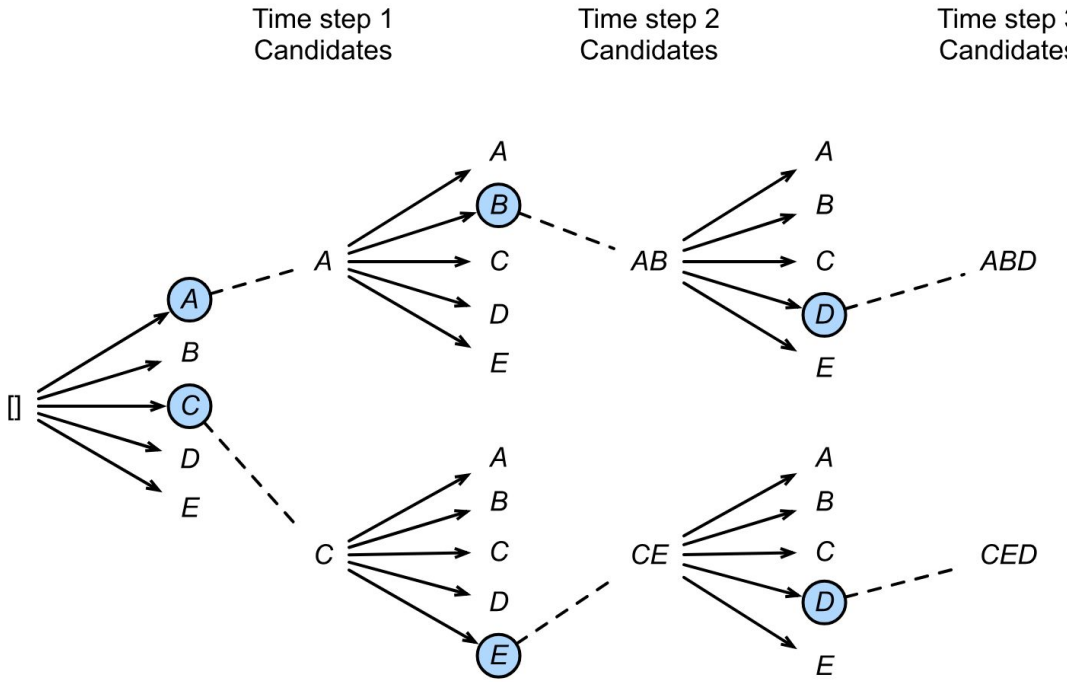


P = 0.5×0.3×0.6×0.6 = 0.054

http://en.diveintodeeplearning.org/chapter_natural-language-processing/beam-search.html

Decoding: Beam search

Beam search can give better solutions. Hyper-parameter: beam size.



http://en.diveintodeeplearning.org/chapter_natural-language-processing/beam-search.html

Neuraltalk2

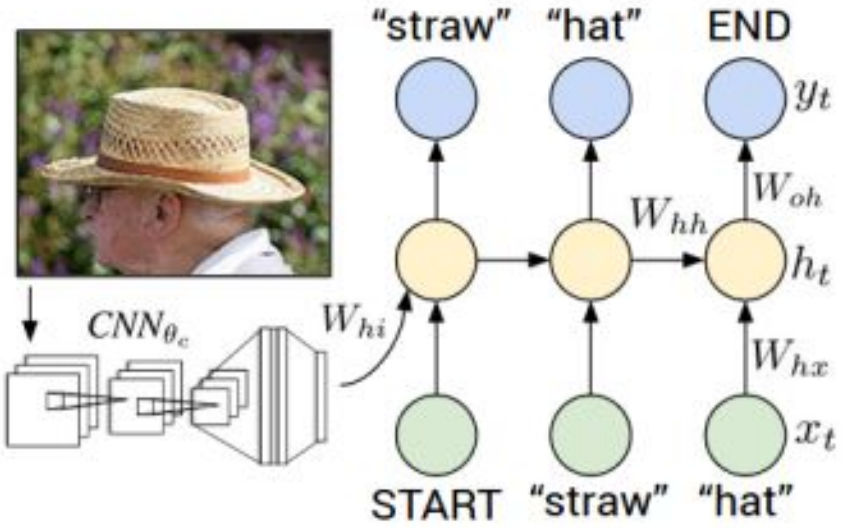


Figure 4. Diagram of our multimodal Recurrent Neural Network generative model. The RNN takes a word, the context from previous time steps and defines a distribution over the next word in the sentence. The RNN is conditioned on the image information at the first time step. START and END are special tokens.

http://cs.stanford.edu/people/karpathy/deepimagesent/

Deep Visual-Semantic Alignments for Generating Image Descriptions

Neuraltalk2

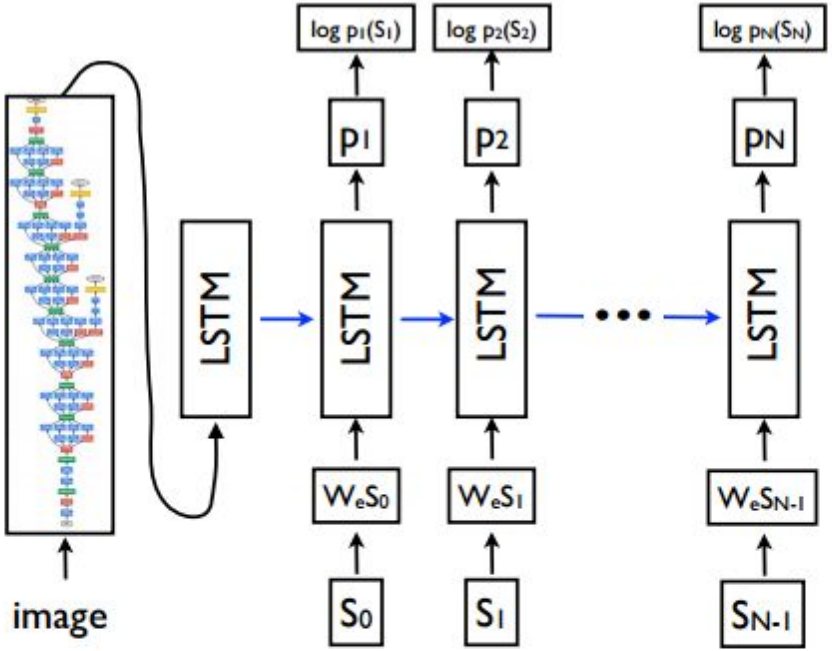


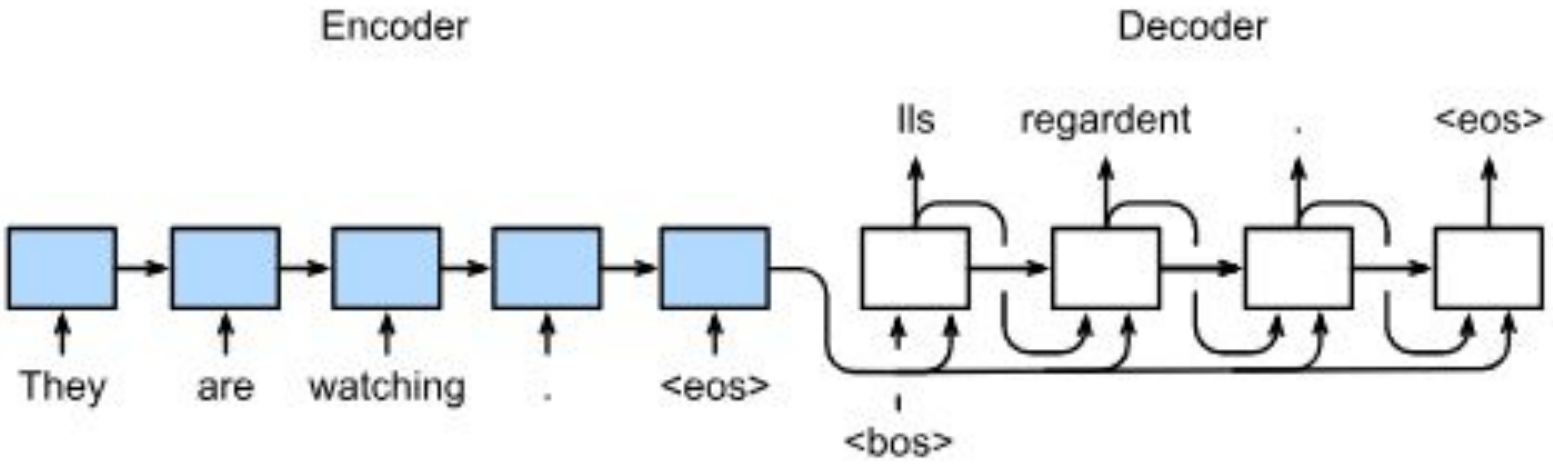
Figure 3. LSTM model combined with a CNN image embedder (as defined in [12]) and word embeddings. The unrolled connections between the LSTM memories are in blue and they correspond to the recurrent connections in Figure 2. All LSTMs share the same parameters.

https://arxiv.org/abs/1411.4555 Show and Tell: A Neural Image Caption Generator

Attention mechanisms

Encoder-Decoder shortcomings

Encoder-Decoder can be applied to N-to-M sequence, yet an Encoder reads and encodes a source sentence into a fixed-length vector. Is one hidden state really enough? A neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector.



Encoder-Decoder with Attention

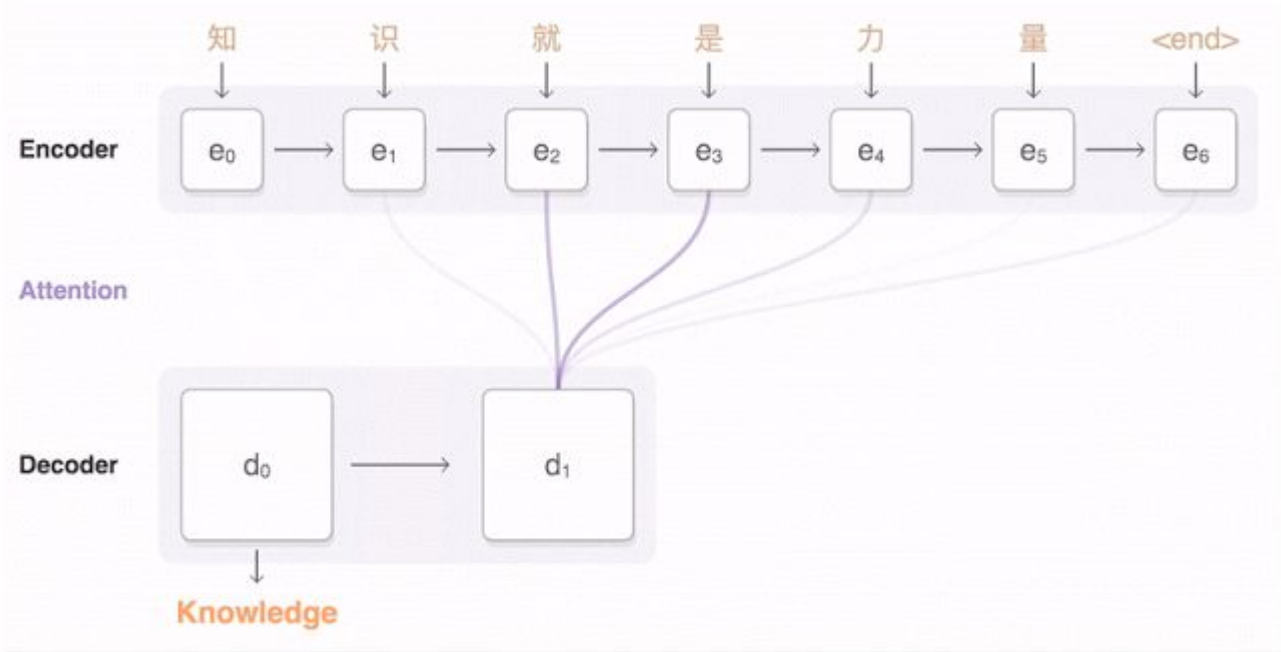
Attention Mechanism allows the decoder to attend to different parts of the source sentence at each step of the output generation.

Instead of encoding the input sequence into a single fixed context vector, we let the model learn how to generate a context vector for each output time step. That is we let the model learn what to attend based on the input sentence and what it has produced so far.

https://hackernoon.com/attention-mechanism-in-neural-network-30aaf5e39512

Encoder-Decoder with Attention

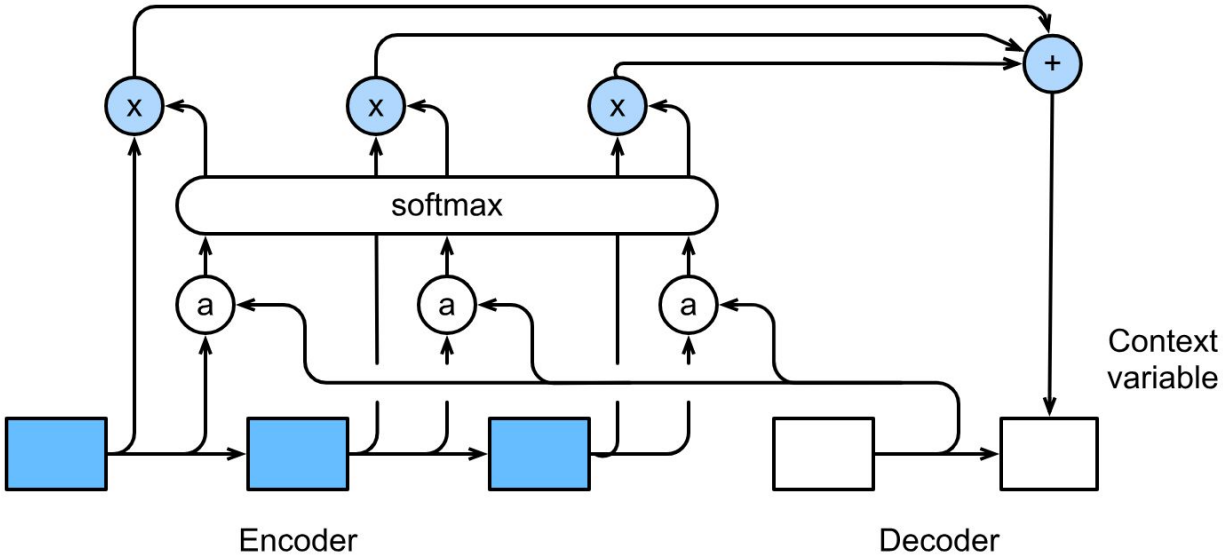
Attention Mechanism allows the decoder to attend to different parts of the source sentence at each step of the output generation.



https://research.googleblog.com/2016/09/a-neural-network-for-machine.html

Attention Mechanism

The attention mechanism obtains the context variable by weighting the hidden state of all time steps of the encoder.

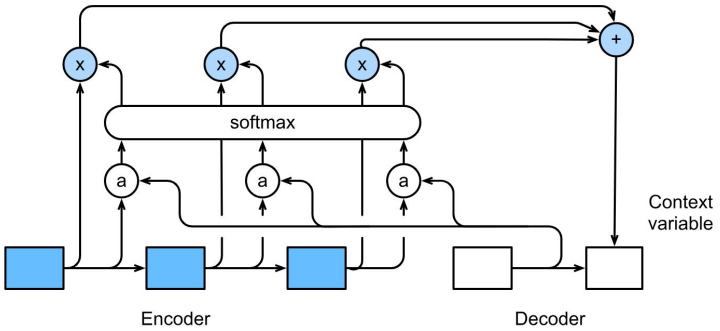


http://en.diveintodeeplearning.org/chapter_natural-language-processing/attention.html

Attention Mechanism

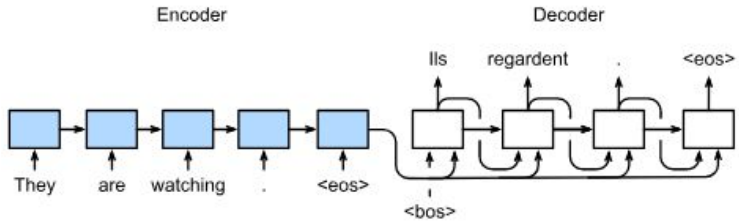
Now the hidden state of the decoder at time step ***t'*** can be rewritten as:

$$\boldsymbol{s}_{t'} = g(\boldsymbol{y}_{t'-1}, \boldsymbol{c}_{t'}, \boldsymbol{s}_{t'-1})$$



Compare to simple encoder-decoder:

$$\boldsymbol{s}_{t'} = g(y_{t'-1}, \boldsymbol{c}, \boldsymbol{s}_{t'-1})$$



http://en.diveintodeeplearning.org/chapter_natural-language-processing/attention.html

Attention Mechanism

The context variable of the decoder at time step ***t'*** is the weighted average on all the hidden states of the encoder ($\boldsymbol{h}_t$).

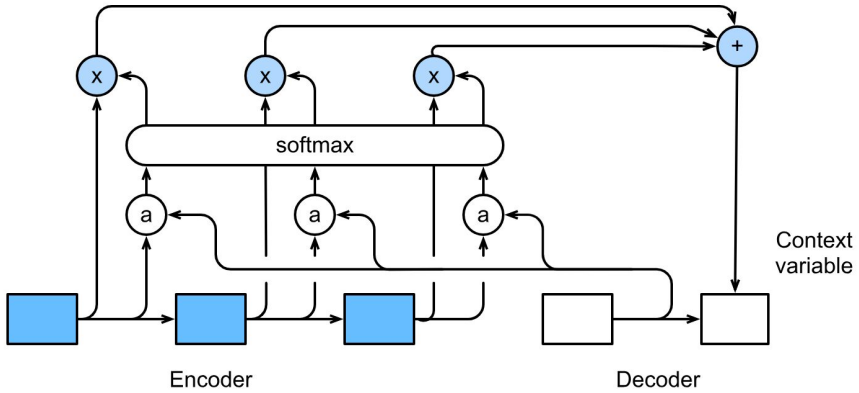
$\boldsymbol{s}_{t'-1}$: decoder's hidden state at that time step ***t'−1***

$$\boldsymbol{c}_{t'} = \sum_{t=1}^{T} \alpha_{t't} \boldsymbol{h}_t$$

$$\alpha_{t't} = \frac{\exp(e_{t't})}{\sum_{k=1}^{T} \exp(e_{t'k})}, \quad t = 1, \ldots, T$$

$$e_{t't} = a(\boldsymbol{s}_{t'-1}, \boldsymbol{h}_t)$$

$$a(\boldsymbol{s}, \boldsymbol{h}) = \boldsymbol{v}^\top \tanh(\boldsymbol{W}_s \boldsymbol{s} + \boldsymbol{W}_h \boldsymbol{h})$$



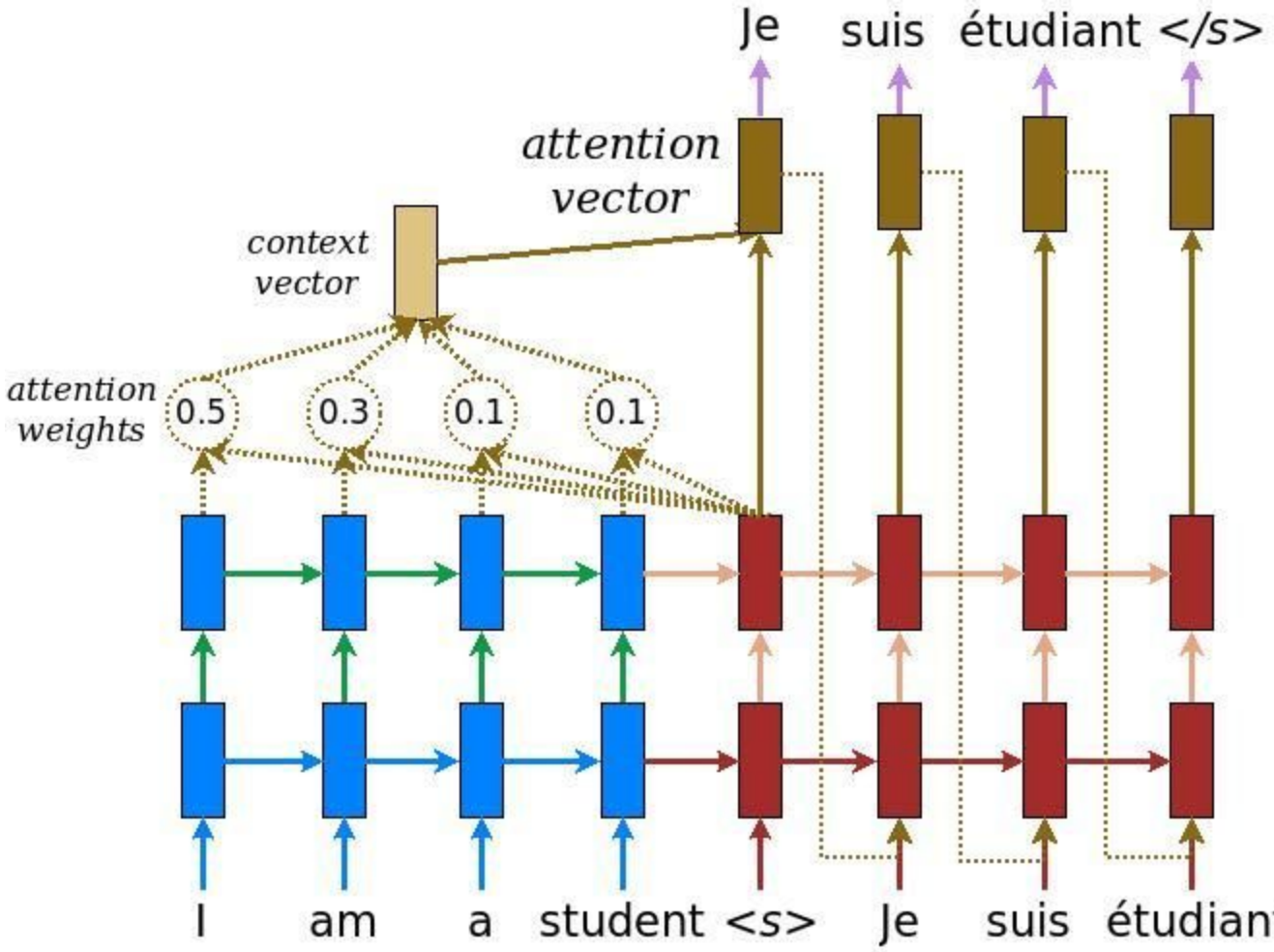
http://en.diveintodeeplearning.org/chapter_natural-language-processing/attention.html

Attention Mechanism

Attention is simply a vector, often the outputs of dense layer using softmax function.

It is just an interface, you could plug it anywhere you find it suitable.

Can use different functions: tahn, sigmoid, dot product, learned transformations



https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129

Attention Mechanism

q is the query (decoder state) and **k** is the key (encoder states)

- **Multi-layer Perceptron** (Bahdanau et al. https://arxiv.org/abs/1409.0473)

  $$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_2^\intercal \tanh(W_1[\boldsymbol{q}; \boldsymbol{k}])$$

  Sometimes called ‘concat’.
- **Bilinear** (Luong et al. https://arxiv.org/abs/1508.04025)

  $$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal W \boldsymbol{k}$$
- **Dot Product** (Luong)

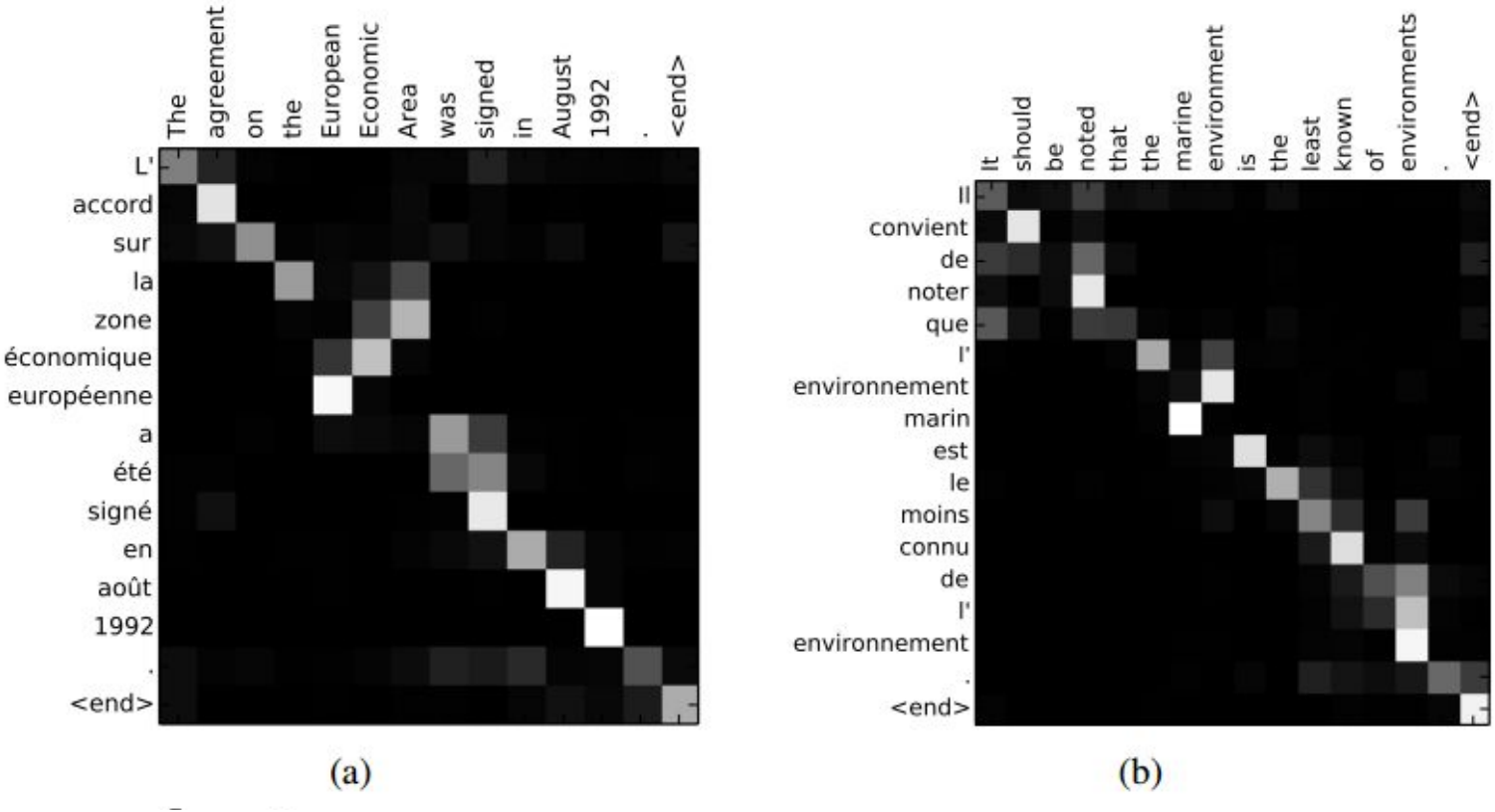
  $$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal \boldsymbol{k}$$
- **Scaled Dot Product** (Vaswani et al. https://arxiv.org/abs/1706.03762)

  $$a(\boldsymbol{q}, \boldsymbol{k}) = \frac{\boldsymbol{q}^\intercal \boldsymbol{k}}{\sqrt{|\boldsymbol{k}|}}$$

  Scale by size of the vector

http://phontron.com/class/nn4nlp2017/assets/slides/nn4nlp-09-attention.pdf

Visualizing RNN attention weights α_{ij} on MT



Neural Machine Translation by Jointly Learning to Align and Translate, https://arxiv.org/abs/1409.0473

Visualizing RNN attention heat maps on QA

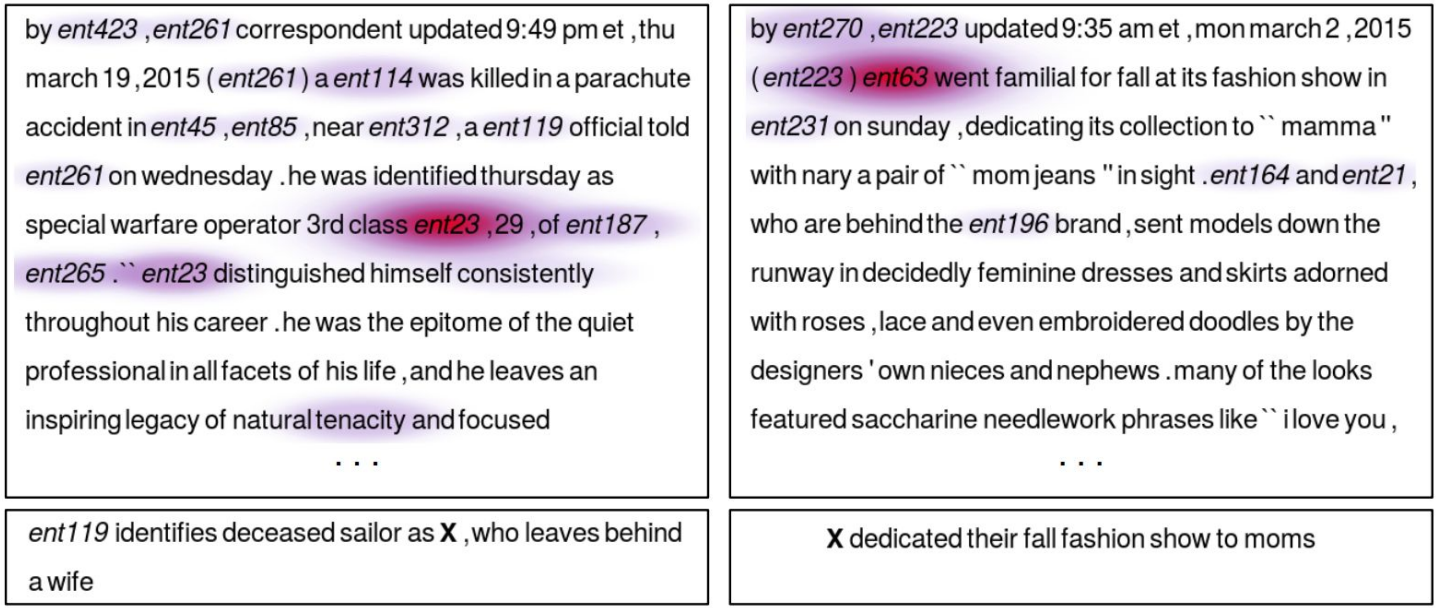
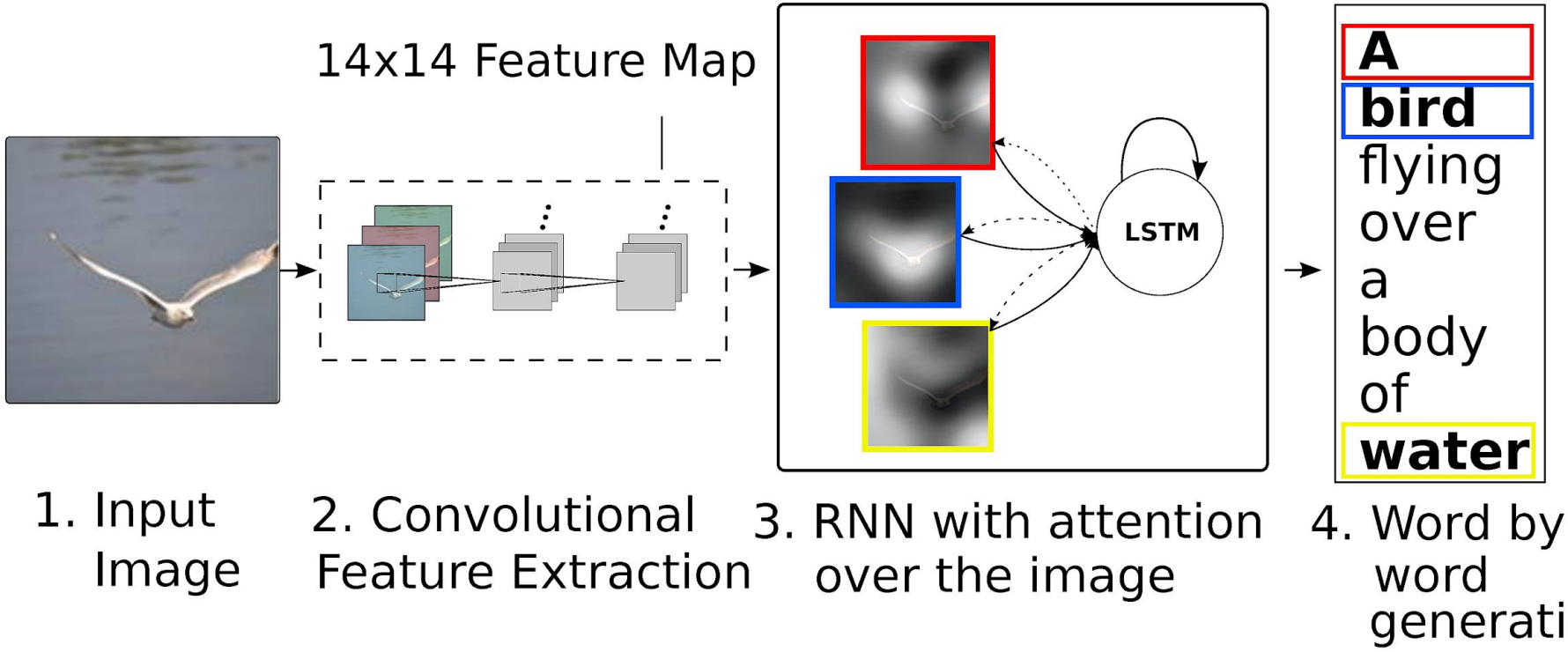


Figure 3: Attention heat maps from the Attentive Reader for two correctly answered validation set queries (the correct answers are *ent23* and *ent63*, respectively). Both examples require significant lexical generalisation and co-reference resolution in order to be answered correctly by a given model.

Teaching Machines to Read and Comprehend, https://arxiv.org/abs/1506.03340

CNN+RNN with Attention



http://kelvinxu.github.io/projects/capgen.html

CNN+RNN with Attention

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

A woman is throwing a frisbee in a park.

http://kelvinxu.github.io/projects/capgen.html

Hard vs. Soft attention

Soft means differentiable, Hard means non-differentiable.

The mechanism described previously is called **Soft attention** because it is a fully differentiable deterministic mechanism that can be plugged into an existing system, and the gradients are propagated through the attention mechanism at the same time they are propagated through the rest of the network.

Hard attention is a stochastic process: instead of using all the hidden states as an input for the decoding, the system samples a hidden state y_i with the probabilities s_i. In order to propagate a gradient through this process, we estimate the gradient by Monte Carlo sampling. Alternatively you can use Reinforcement Learning approaches.

https://blog.heuritech.com/2016/01/20/attention-mechanism/
https://jhui.github.io/2017/03/15/Soft-and-hard-attention/

Self-attention (Intra-Attention)

Each element in the sentence attends to other elements. It gives context sensitive encodings.

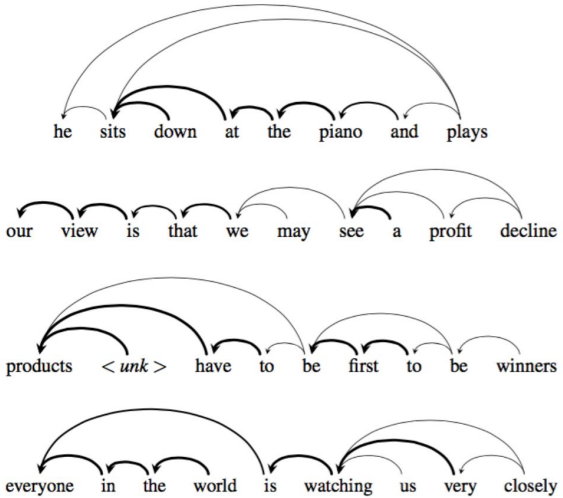


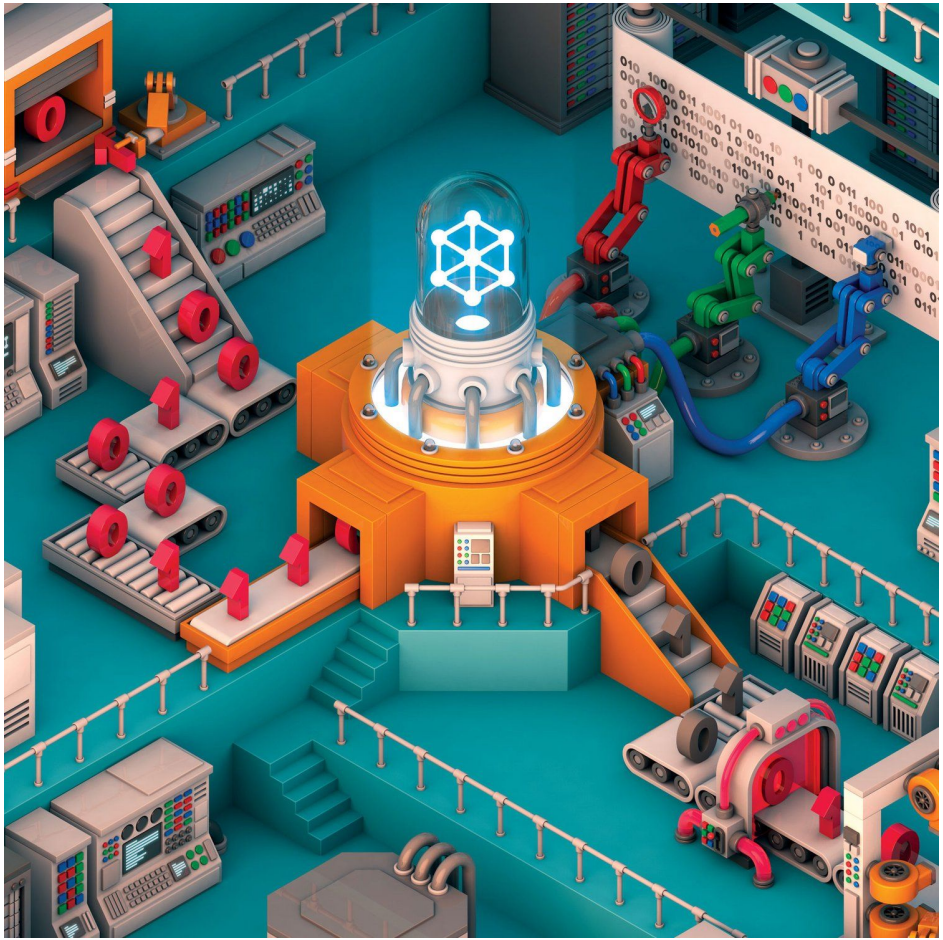
Figure 4: Examples of intra-attention (language modeling). Bold lines indicate higher attention scores. Arrows denote which word is being focused when attention is computed, but not the direction of the relation.

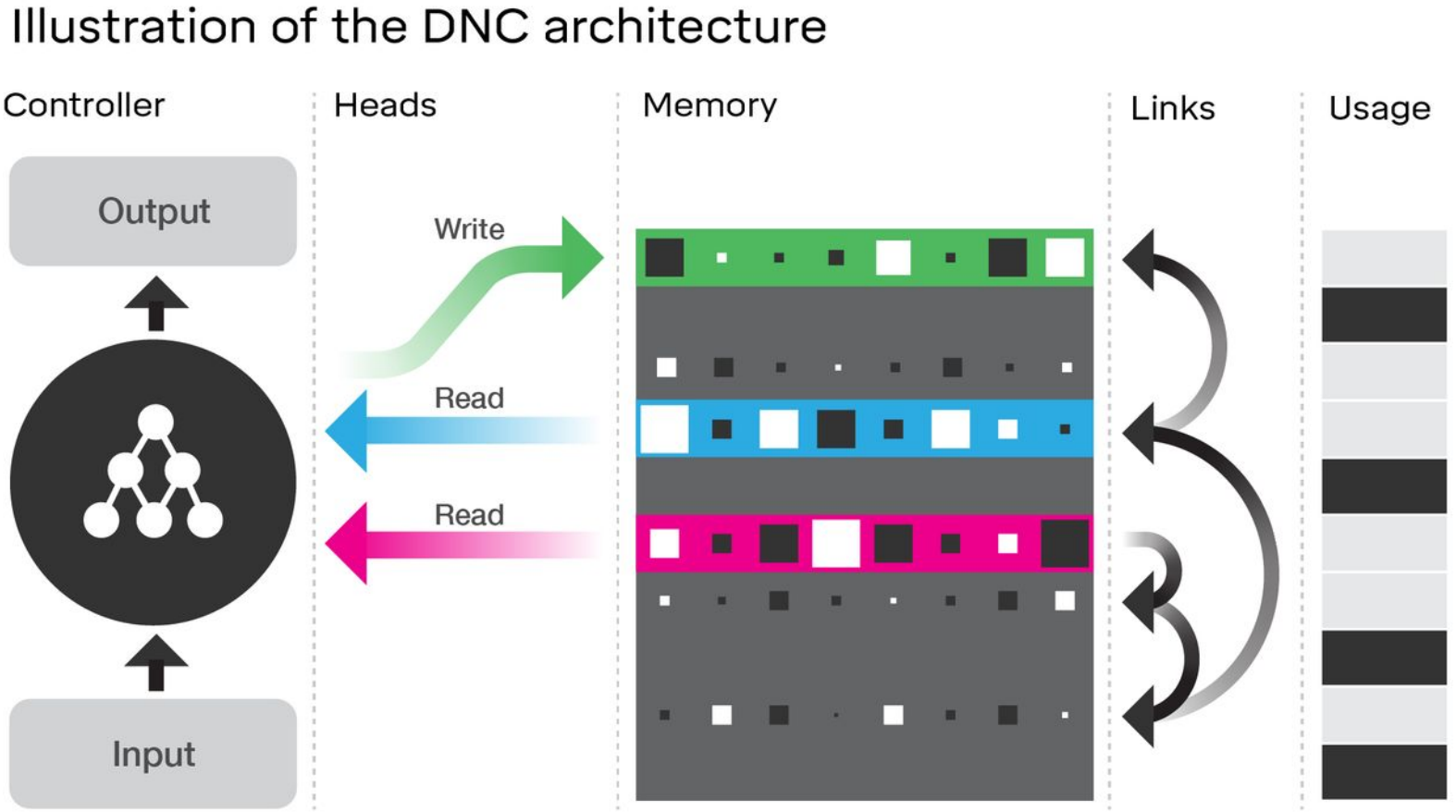
Long Short-Term Memory-Networks for Machine Reading, https://arxiv.org/abs/1601.06733

Augmented RNNs

More augmented RNNs

- Attentional Interfaces (Hard attention, Soft attention)
- Differentiable Memory (Neural Turing Machines, Differentiable neural computer, Hierarchical Attentive Memory, Memory Networks, ...)
- Adaptive Computation Time
- Differentiable Data Structures (structured memory: stack, list, queue, …)
- Differential Programming (Neural Programmer, Differentiable Functional Program Interpreters, …)
- ...





https://deepmind.com/blog/differentiable-neural-computers/

Resources

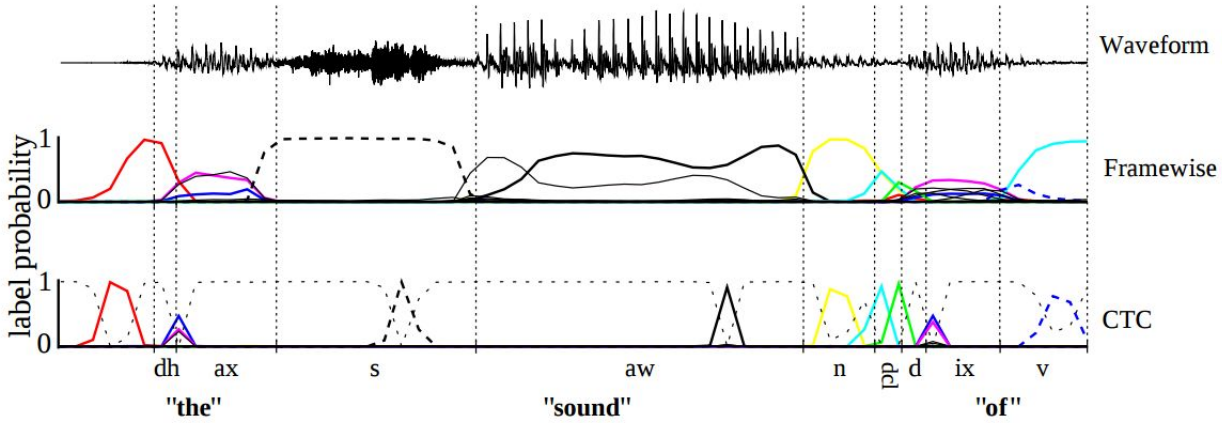
- Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention) https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
- Memory, attention, sequences https://towardsdatascience.com/memory-attention-sequences-37456d271992
- Attention? Attention! https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html
- Attention and Augmented Recurrent Neural Networks http://distill.pub/2016/augmented-rnns/
- https://mchromiak.github.io/articles/2017/Sep/01/Primer-NN/#attention-basis

Connectionist Temporal Classification (CTC)

CTC (Connectionist Temporal Classification)

There are many tasks where the exact label placement is not important, only the order is important. E.g. speech recognition, handwriting recognition, licence plate recognition.

Special CTC output layer (Graves, Fernández, Gomez, Schmidhuber, 2006) was created for such tasks of temporal classification, where alignment between input frames and output labels is not known and is not required.



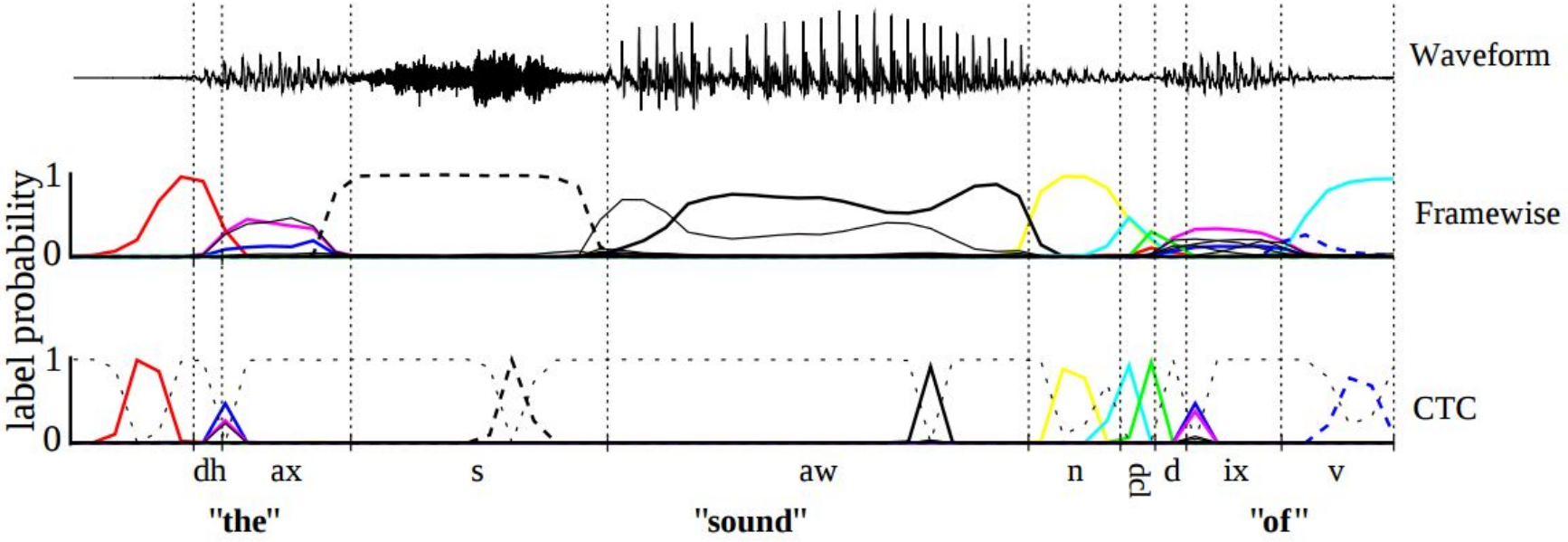
Another useful thing: CTC Output Layer

CTC (Connectionist Temporal Classification; Graves, Fernández, Gomez, Schmidhuber, 2006) was specifically designed for temporal classification tasks; that is, for sequence labelling problems where the alignment between the inputs and the target labels is unknown.

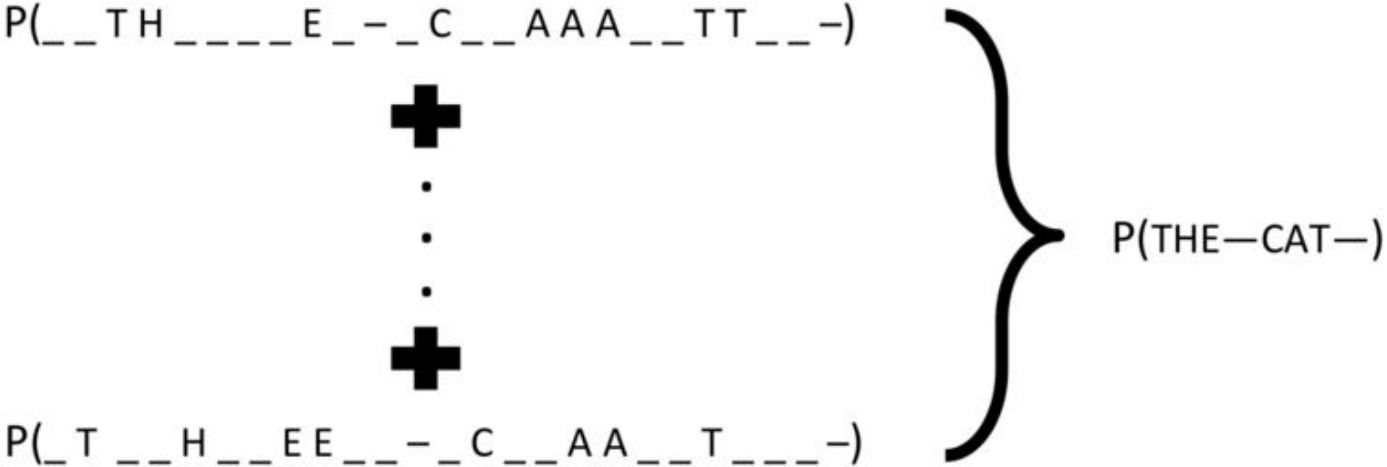
CTC models all aspects of the sequence with a single neural network, and does not require the network to be combined with a hidden Markov model. It also **does not require presegmented training data**, or external post-processing to extract the label sequence from the network outputs.

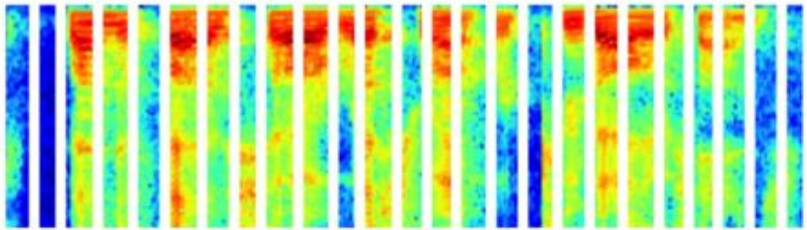
The CTC network **predicts only the sequence** of phonemes (typically as a series of spikes, separated by ‘blanks’, or null predictions), while the framewise network attempts to align them with the manual segmentation.

Example: CTC vs. Framewise classification



CTC (Connectionist Temporal Classification)





https://github.com/baidu-research/warp-ctc

Resources

- Supervised Sequence Labelling with Recurrent Neural Networks (Multidimensional RNN, CTC and sequence labeling) https://www.cs.toronto.edu/~graves/preprint.pdf
- CTC and Speech Recognition https://habrahabr.ru/company/google/blog/269747/
- Speech Recognition: You down with CTC? https://gab41.lab41.org/speech-recognition-you-down-with-ctc-8d3b558943f0

Non-RNN Sequence Learning

Encoder-Decoder: original architecture

Recurrent Encoder / Recurrent Decoder

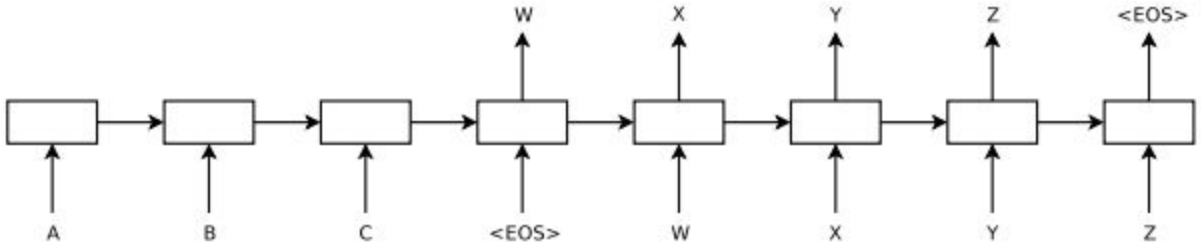


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Sequence to Sequence Learning with Neural Networks, https://arxiv.org/abs/1409.3215

Case: Machine Translation

Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

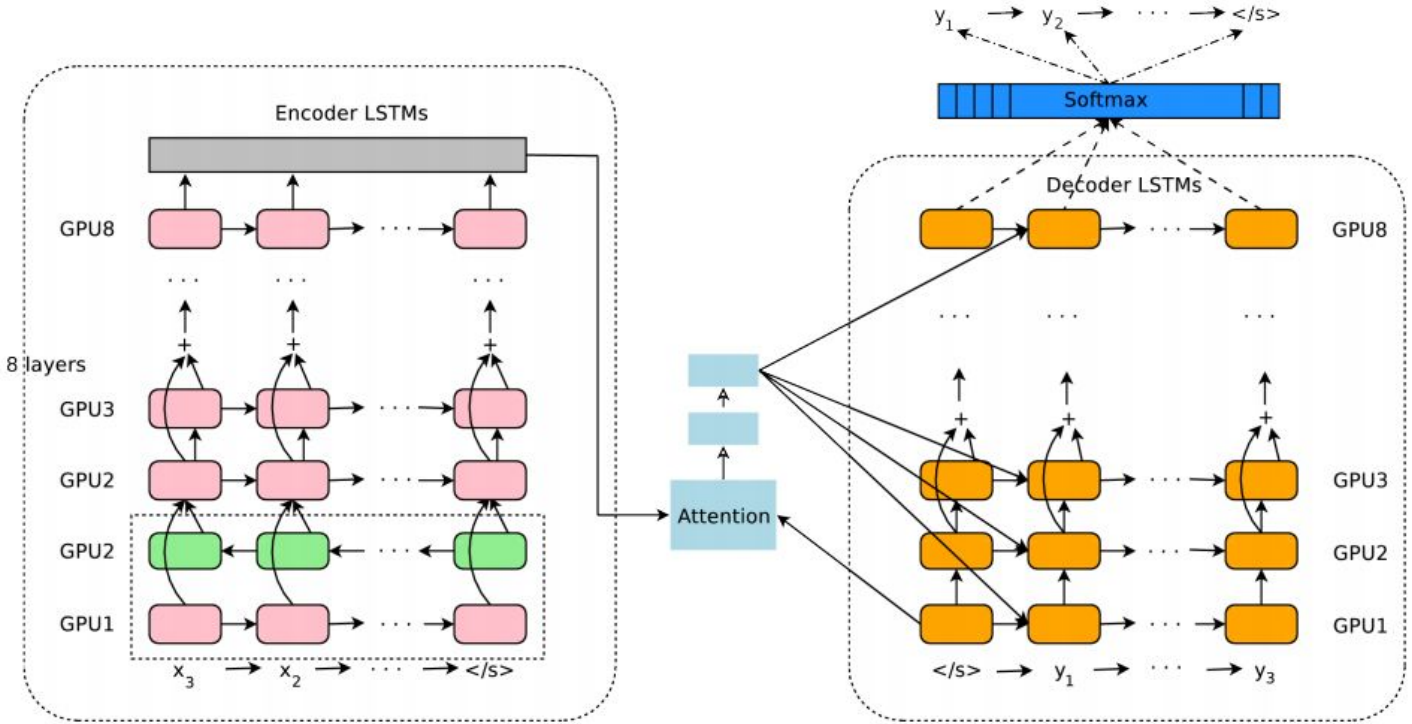
Table 1: The performance of the LSTM on WMT'14 English to French test set (ntst14). Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than of a single LSTM with a beam of size 12.

Method	test BLEU score (ntst14)
Baseline System [29]	33.30
Cho et al. [5]	34.54
State of the art [9]	**37.0**
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	**36.5**
Oracle Rescoring of the Baseline 1000-best lists	~45

Table 2: Methods that use neural networks together with an SMT system on the WMT'14 English to French test set (ntst14).

Sequence to Sequence Learning with Neural Networks, http://arxiv.org/abs/1409.3215

Encoder-Decoder: modern architecture



Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,
https://arxiv.org/abs/1609.08144
Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation
https://arxiv.org/abs/1611.04558

Encoder-Decoder: character-level models

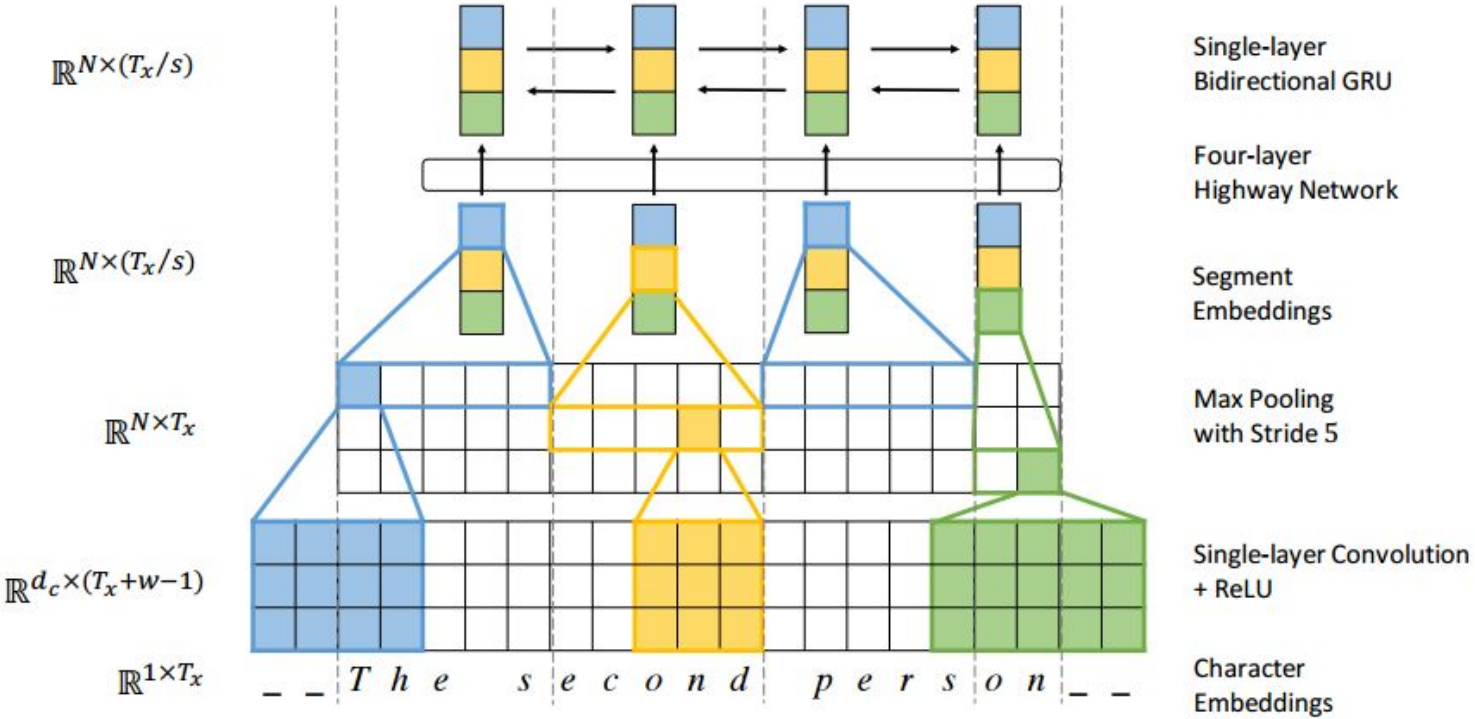


Figure 1: Encoder architecture schematics. Underscore denotes padding. A dotted vertical line delimits each segment.

Fully Character-Level Neural Machine Translation without Explicit Segmentation, https://arxiv.org/abs/1610.03017

The Problem:
RNNs are slow.

The solution #1: Optimizing RNNs

"Pruning RNNs reduces the size of the model and can also help achieve significant inference time speed-up using sparse matrix multiply. Benchmarks show that using our technique model size can be reduced by 90% and speed-up is around 2× to 7×."

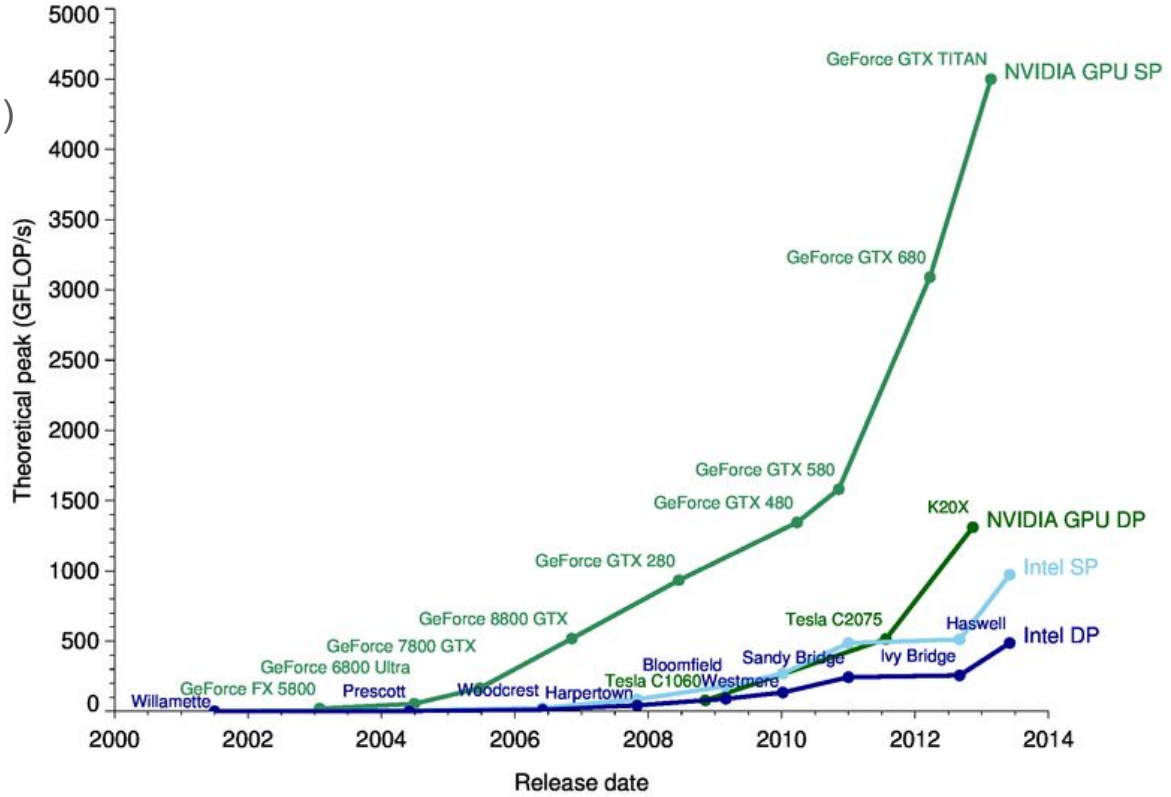
Table 3: GRU & bidirectional RNN model results

MODEL	# UNITS	CER	# PARAMS	RELATIVE PERF
RNN Dense Baseline	1760	10.67	67 million	0.0%
RNN Dense Small	704	14.50	11.6 million	-35.89%
RNN Dense Medium	2560	9.43	141 million	11.85%
RNN Sparse 1760	1760	12.88	8.3 million	-20.71%
RNN Sparse Medium	2560	**10.59**	11.1 million	0.75%
RNN Sparse Big	3072	**10.25**	16.7 million	3.95%
GRU Dense	2560	9.55	115 million	0.0%
GRU Sparse	2560	10.87	13 million	-13.82%
GRU Sparse Medium	3568	**9.76**	17.8 million	-2.20%

Exploring Sparsity in Recurrent Neural Networks, https://arxiv.org/abs/1704.05119

The solution #2: Better hardware

- Google TPU gen.2
  - 180 TFLOPS?
- NVIDIA DGX-1 (8*P100) ($129,000)
  - 170 TFLOPS (FP16)
  - 85 TFLOPS (FP32)
- NVIDIA Tesla V100
  - 15 TFLOPS (FP32)
  - 120 TFLOPS (Tensor Core)
- NVIDIA Tesla P100
  - 10.6 TFLOPS (FP32)
- NVIDIA GTX Titan X ($1000)
  - 11 TFLOPS (FP32)
- NVIDIA GTX 1080/1080 Ti ($700)
  - 8/11.3 TFLOPS (FP32)



The solution #2: Better hardware

Why this solution could be among the most interesting ones?

Current success of NNs (especially CNNs) is backed by a large amounts of data available _AND_ more powerful hardware (using the decades-old algorithms). We potentially could achieve the same performance in the past, but the learning process was just too slow (and we were too impatient).

The processor performance grows exponentially and in 5-10 years the available computing power can increase 1000x. There may appear computing units more suitable for RNN computations as well.

The situation could repeat. When the hardware will allow fast training of RNNs, we could achieve a new kind of results. Remember, RNNs are Turing complete. They are (potentially) much more powerful than feed-forward NNs.

Convolutional Sequence Learning

CNN encoder

Convolutional Encoder / Recurrent Decoder

Encoder	Words/s	BLEU
BiLSTM	139.7	22.4
Deep Conv. 6/3	187.9	23.1

(a) IWSLT'14 German-English generation speed on *tst2013* with beam size 10.

Encoder	Words/s	BLEU
2-layer BiLSTM	109.9	23.6
Deep Conv. 8/4	231.1	23.7
Deep Conv. 15/5	203.3	24.0

(b) WMT'15 English-German generation speed on *newstest2015* with beam size 5.

Table 3: Generation speed in source words per second on a single CPU core.

A Convolutional Encoder Model for Neural Machine Translation, https://arxiv.org/abs/1611.02344

CNN encoder

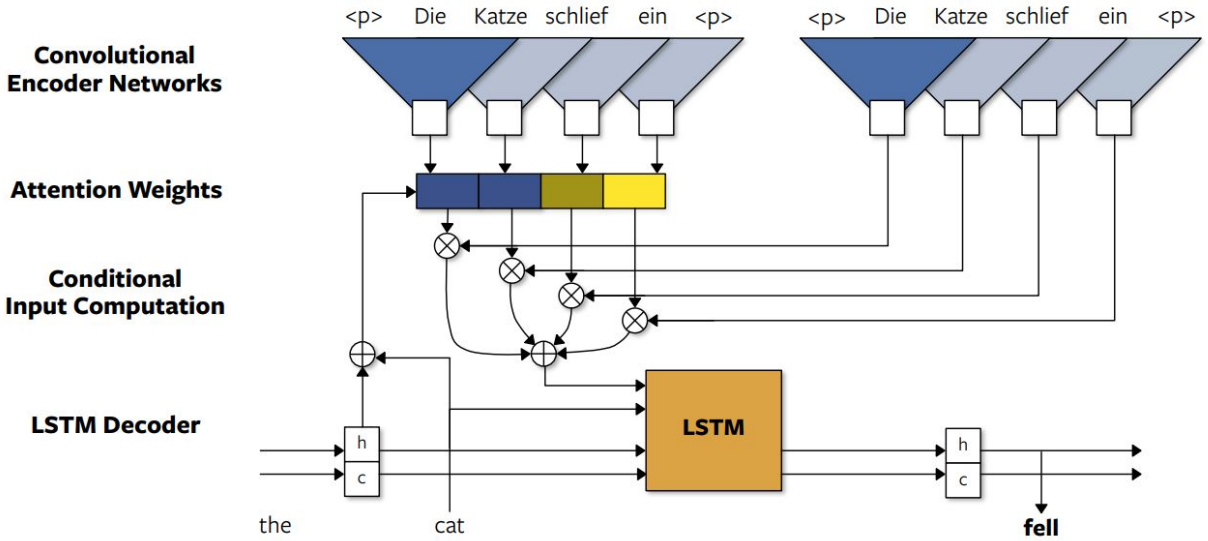


Figure 1: Neural machine translation model with single-layer convolutional encoder networks. CNN-a is on the left and CNN-c is at the right. Embedding layers are not shown.

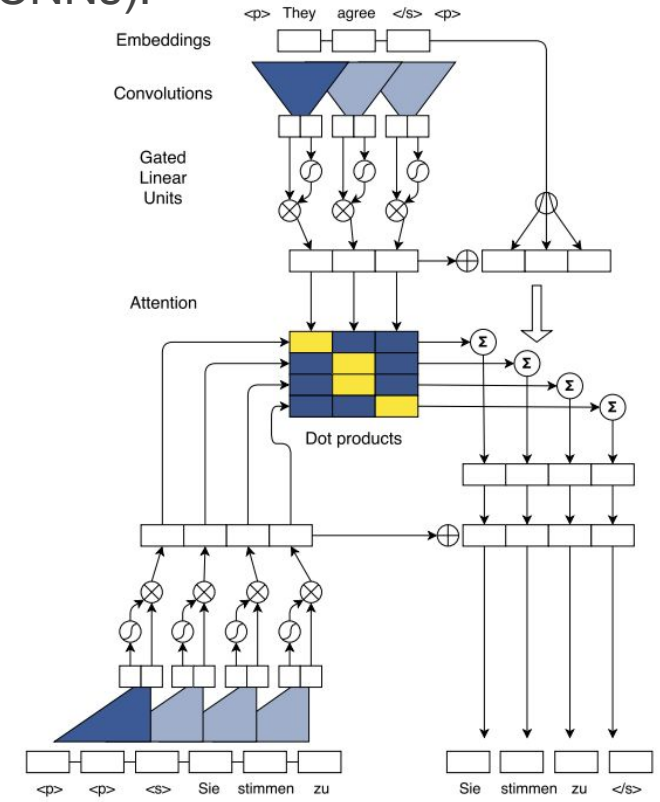
A Convolutional Encoder Model for Neural Machine Translation, https://arxiv.org/abs/1611.02344

CNN encoder + decoder

Actually no RNN here (Facebook AI Research loves CNNs).

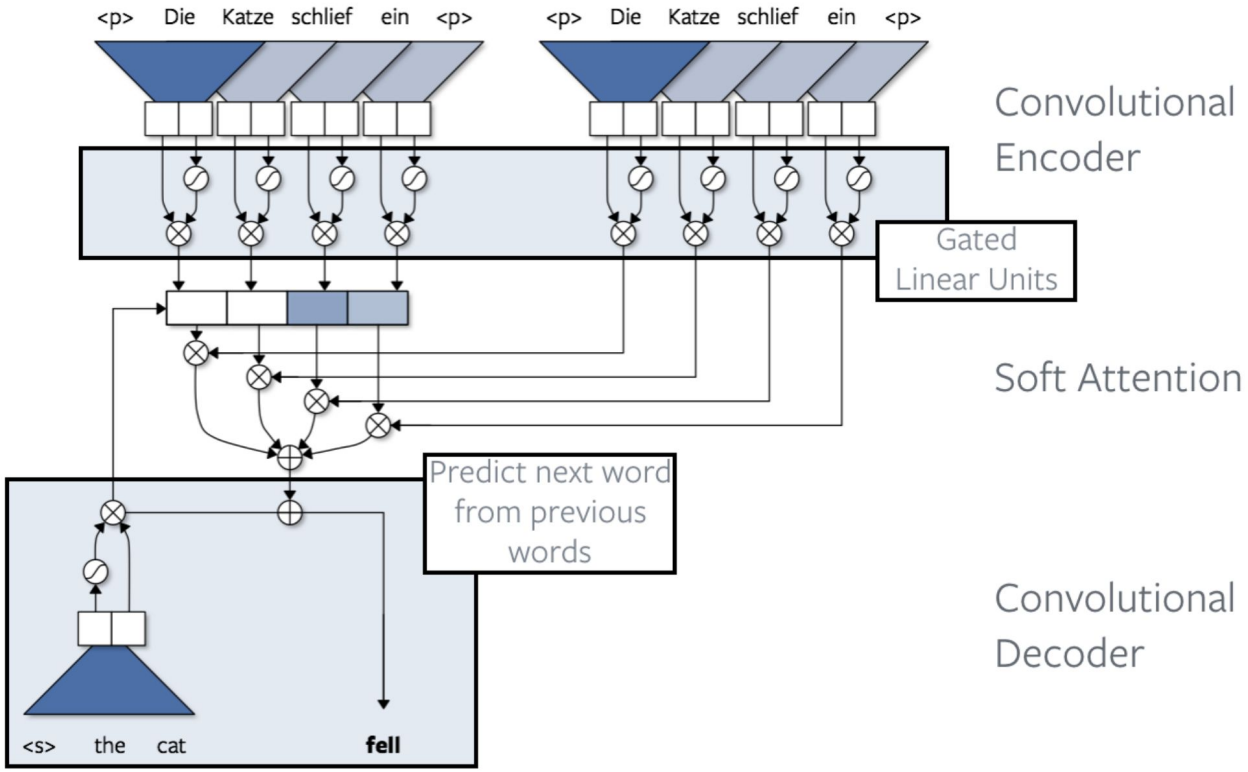
	BLEU	Time (s)
GNMT GPU (K80)	31.20	3,028
GNMT CPU 88 cores	31.20	1,322
GNMT TPU	31.21	384
ConvS2S GPU (K40) $b = 1$	33.45	327
ConvS2S GPU (M40) $b = 1$	33.45	221
ConvS2S GPU (GTX-1080ti) $b = 1$	33.45	142
ConvS2S CPU 48 cores $b = 1$	33.45	142
ConvS2S GPU (K40) $b = 5$	34.10	587
ConvS2S CPU 48 cores $b = 5$	34.10	482
ConvS2S GPU (M40) $b = 5$	34.10	406
ConvS2S GPU (GTX-1080ti) $b = 5$	34.10	256

Table 3. CPU and GPU generation speed in seconds on the development set of WMT'14 English-French. We show results for different beam sizes b. GNMT figures are taken from Wu et al. (2016). CPU speeds are not directly comparable because Wu et al. (2016) use a 88 core machine compared to our 48 core setup.



Convolutional Sequence to Sequence Learning, https://arxiv.org/abs/1705.03122

CNN encoder + decoder



https://aiukraine.com/wp-content/uploads/2017/10/2_1-Yarats.pdf

Resources

- https://github.com/facebookresearch/fairseq (old)
- https://github.com/pytorch/fairseq (new)

  - **Convolutional Neural Networks (CNN)**
    - Dauphin et al. (2017): Language Modeling with Gated Convolutional Networks
    - Gehring et al. (2017): Convolutional Sequence to Sequence Learning
    - Edunov et al. (2018): Classical Structured Prediction Losses for Sequence to Sequence Learning
    - ***New*** Fan et al. (2018): Hierarchical Neural Story Generation
  - **Long Short-Term Memory (LSTM) networks**
    - Luong et al. (2015): Effective Approaches to Attention-based Neural Machine Translation
    - Wiseman and Rush (2016): Sequence-to-Sequence Learning as Beam-Search Optimization
  - **Transformer (self-attention) networks**
    - Vaswani et al. (2017): Attention Is All You Need
    - ***New*** Ott et al. (2018): Scaling Neural Machine Translation
    - ***New*** Edunov et al. (2018): Understanding Back-Translation at Scale

Self-Attention Neural Networks (SAN): Transformer Architecture

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

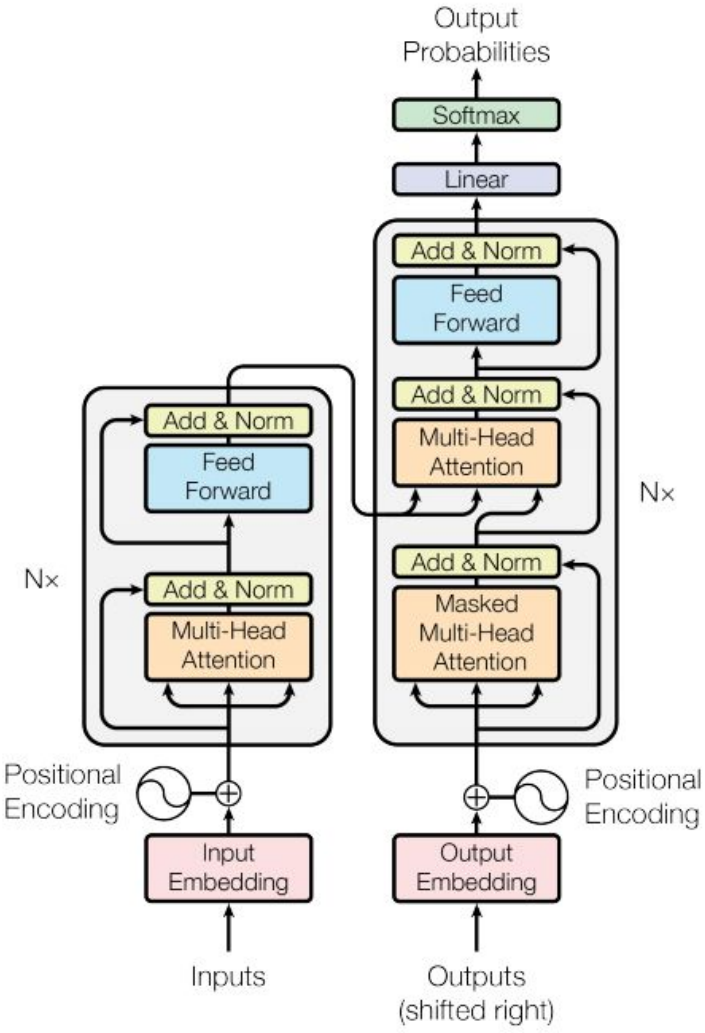
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Attention Is All You Need, https://arxiv.org/abs/1706.03762

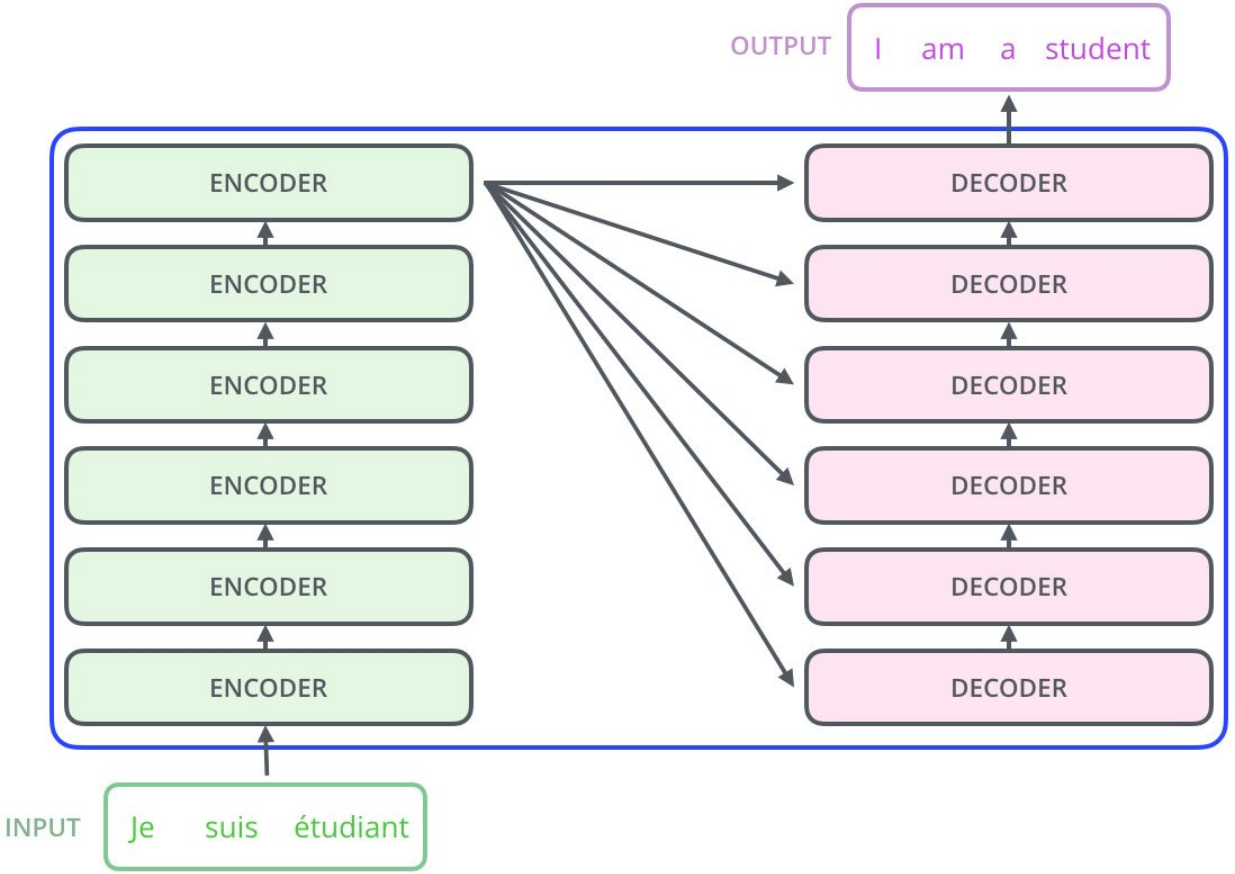
Transformer

A new simple network architecture,
the Transformer:

- Is a Encoder-Decoder architecture
- Based solely on attention mechanisms (no RNN/CNN)
- The major component in the transformer is the unit of multi-head self-attention mechanism.
- Fast: only matrix multiplications
- Strong results on standard WMT datasets



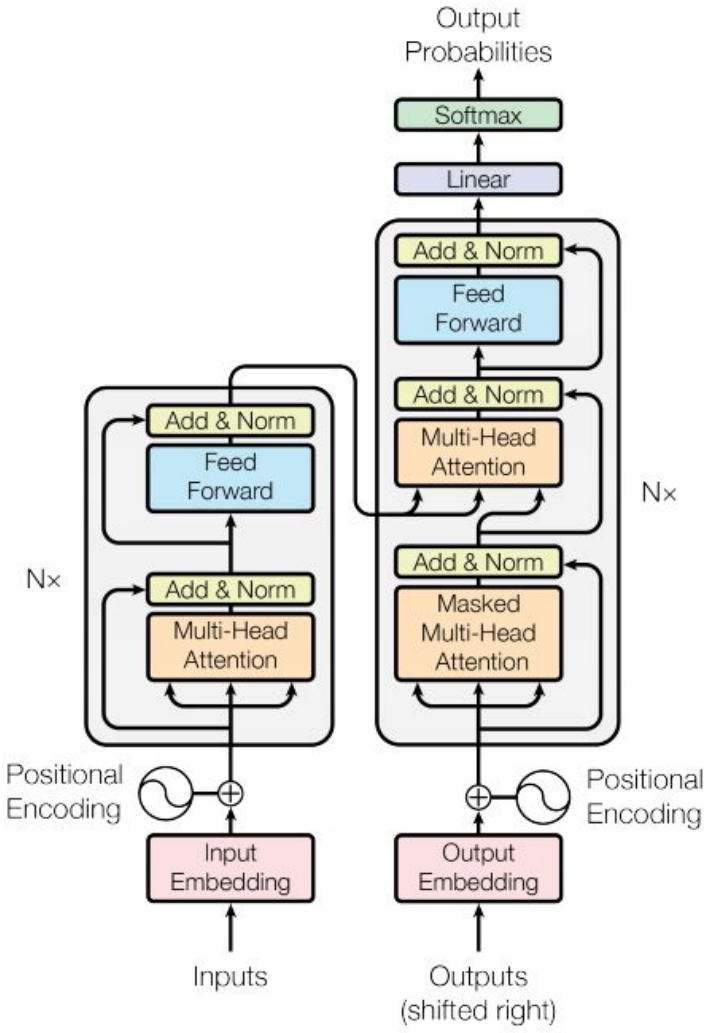
Working pipeline



http://jalammar.github.io/illustrated-transformer/

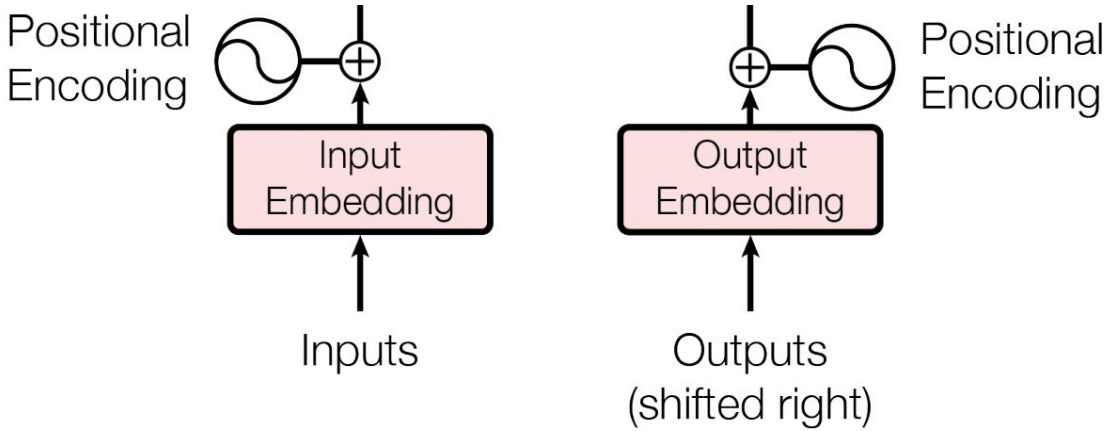
Inputs

The initial inputs to the encoder are the embeddings of the input sequence, and the initial inputs to the decoder are the embeddings of the outputs up to that point.

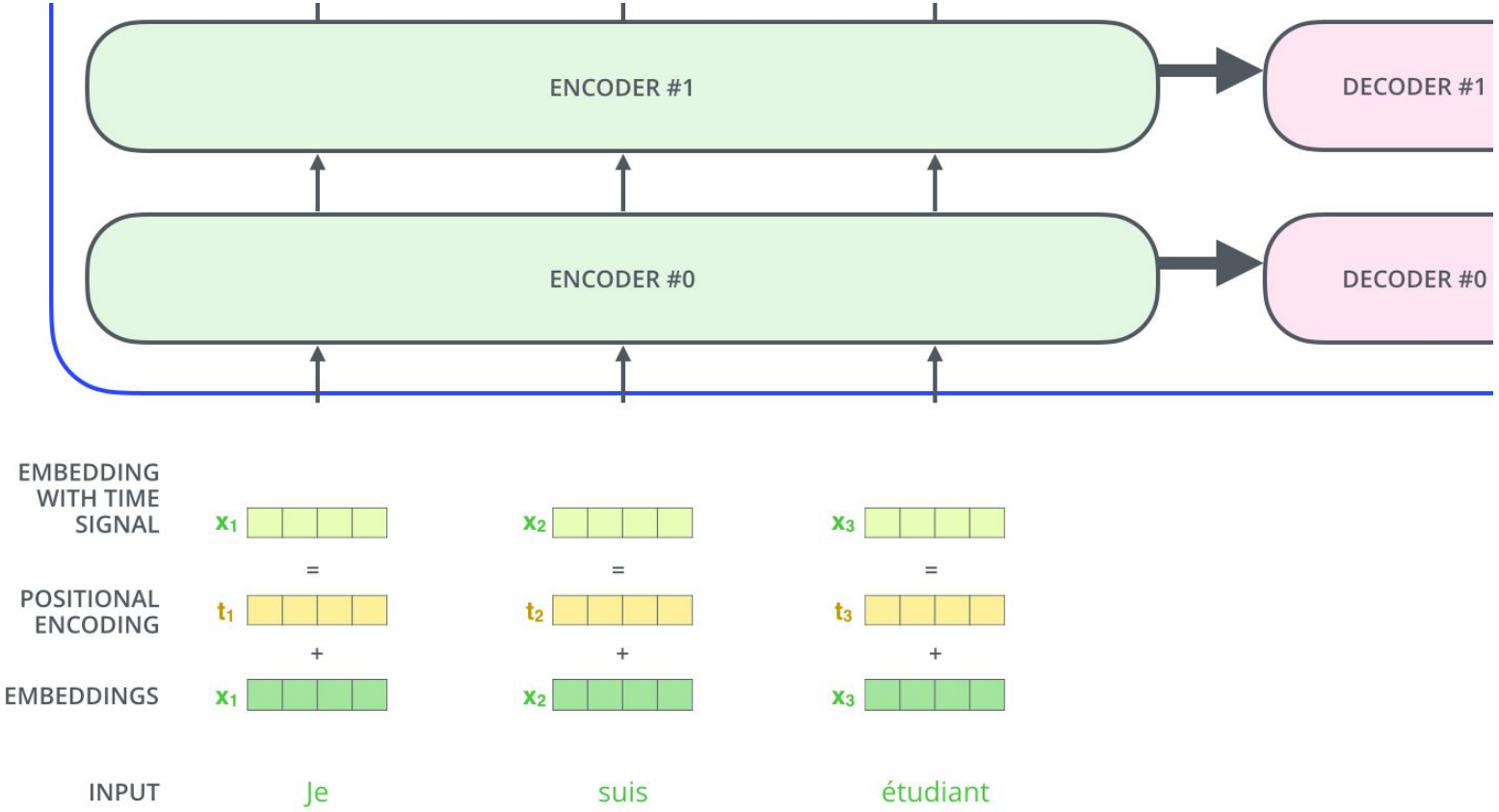


Embeddings

Similarly to other sequence transduction models, we use **learned embeddings** to convert the input tokens and output tokens to vectors of dimension d_{model}. We share the same weight matrix between the two embedding layers.



Embeddings



http://jalammar.github.io/illustrated-transformer/

Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add “positional encodings” to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed.

In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

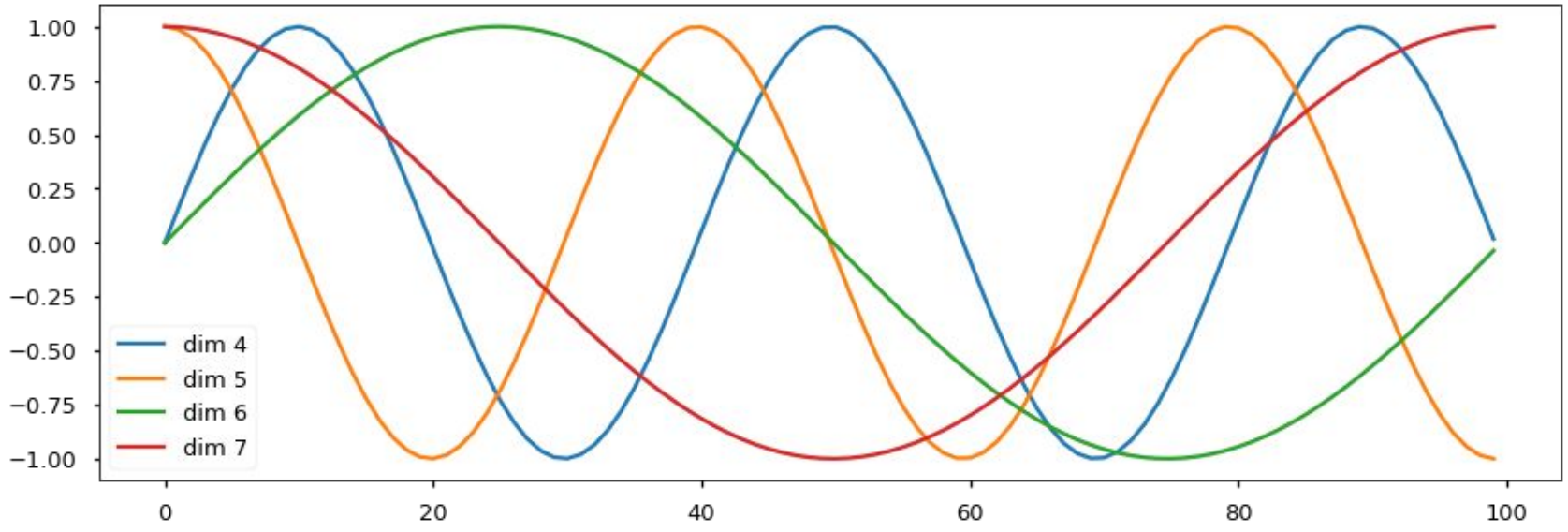
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where *pos* is the position and *i* is the dimension.

Positional Encoding

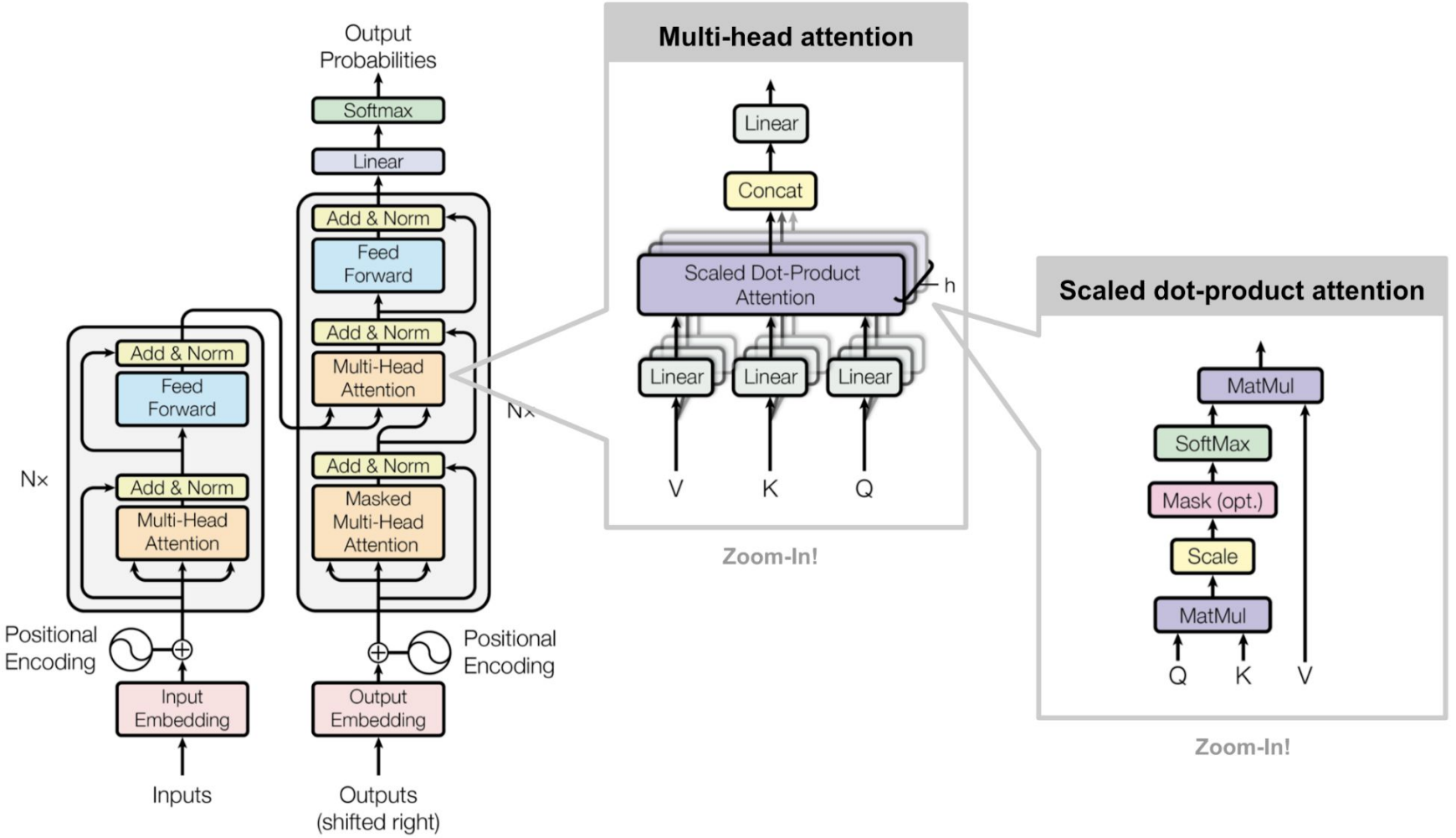
That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$.

Below the positional encoding will add in a sine wave based on position. The frequency and offset of the wave is different for each dimension.



Positional Encoding

1. **Sinusoidal Position Encoding** (Vaswani et al, 2017, https://arxiv.org/abs/1706.03762)
   Use sine/cosine waves as previously described.
2. **Learned Position Encoding** (Gehring et al, 2017, https://arxiv.org/abs/1705.03122)
   Embed the absolute position of input elements. Can't extrapolate to lengths it has never seen during training.
3. **Relative Position Representations** [currently work the best] (Shaw et al, 2018, https://arxiv.org/abs/1803.02155)
   Model the input as a labeled, directed, fully-connected graph. Learn edge representation.



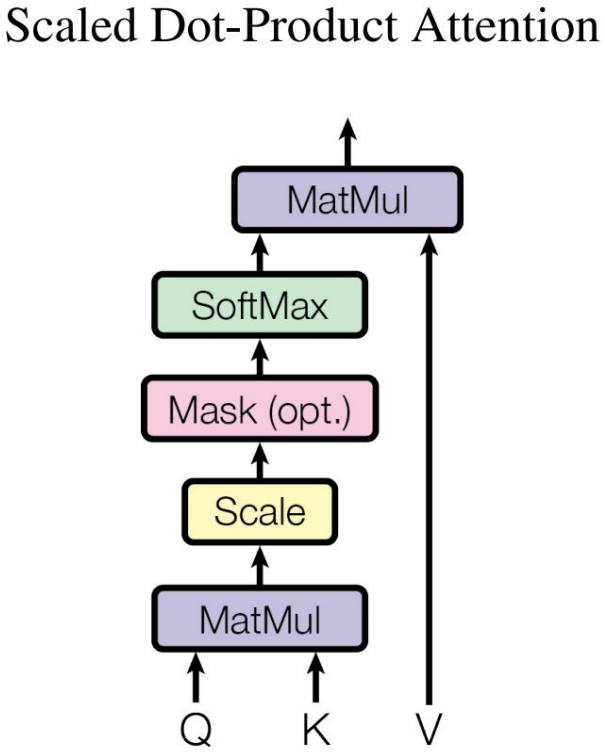
https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Scaled dot-product attention

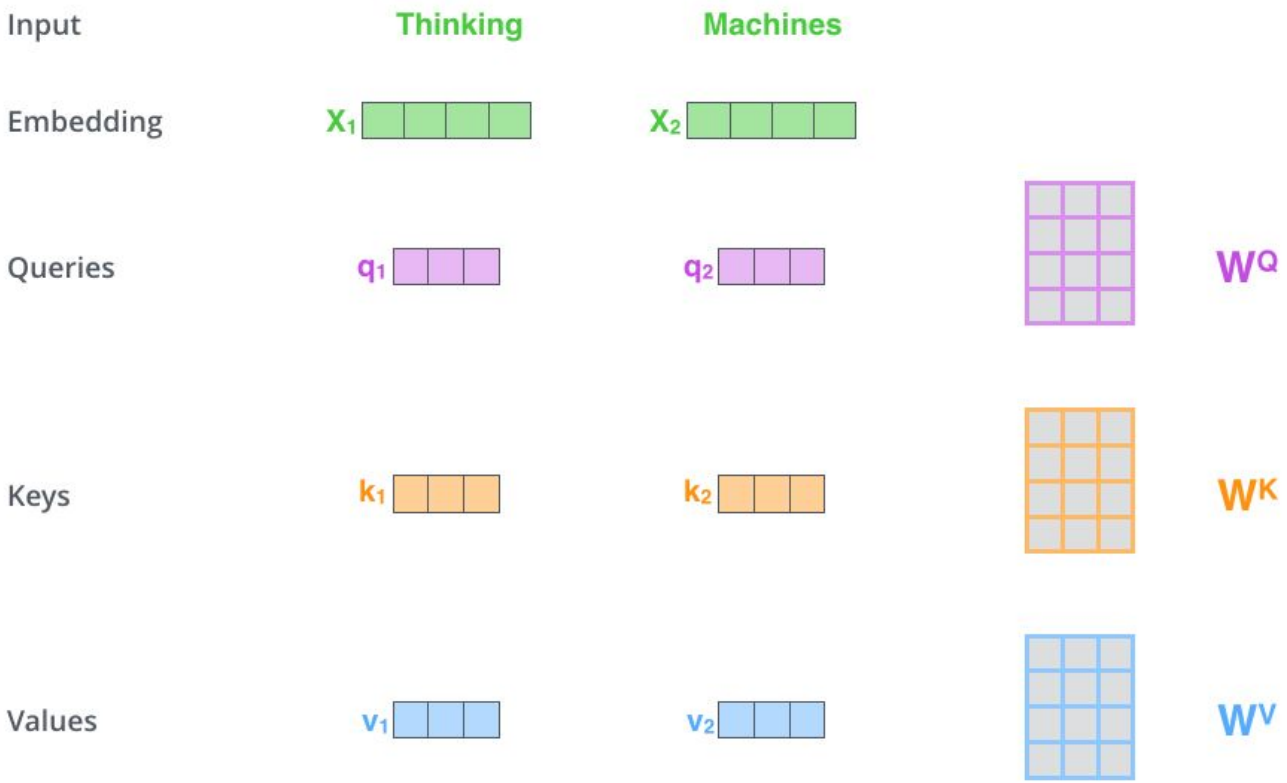
The transformer adopts the scaled dot-product attention: the output is a weighted sum of the values, where the weight assigned to each value is determined by the dot-product of the query with all the keys:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V}$$

The input consists of queries and keys of dimension d_k, and values of dimension d_v.



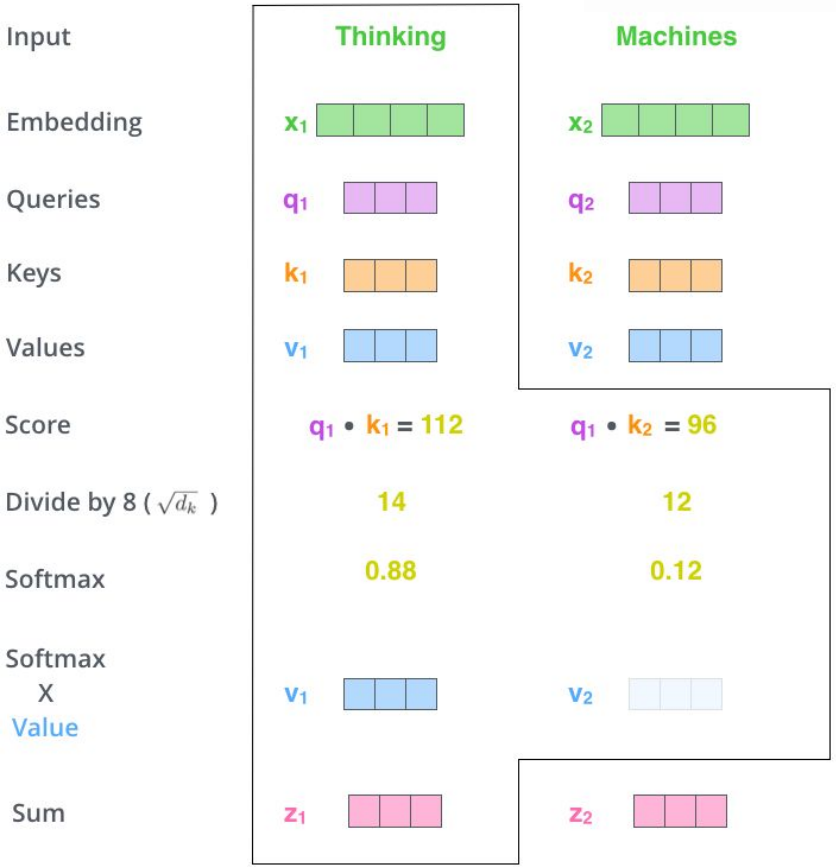
Scaled dot-product attention



http://jalammar.github.io/illustrated-transformer/

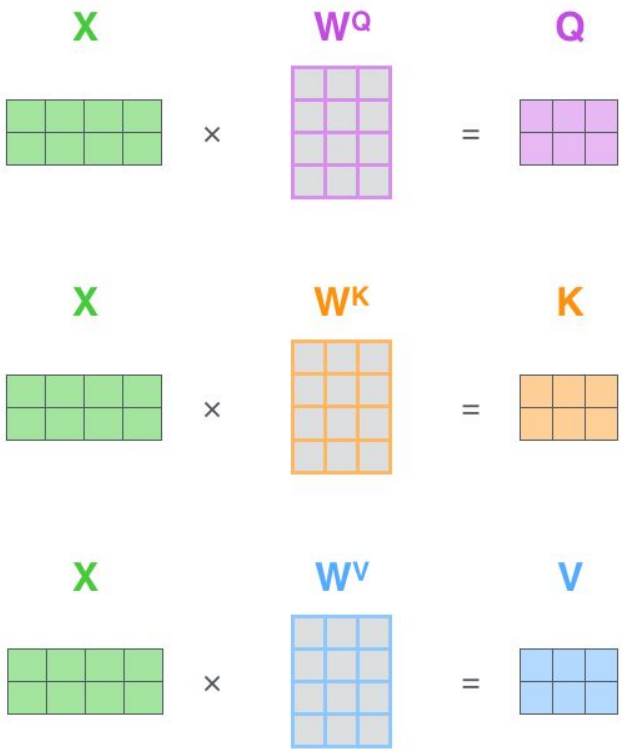
Scaled dot-product attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V}$$



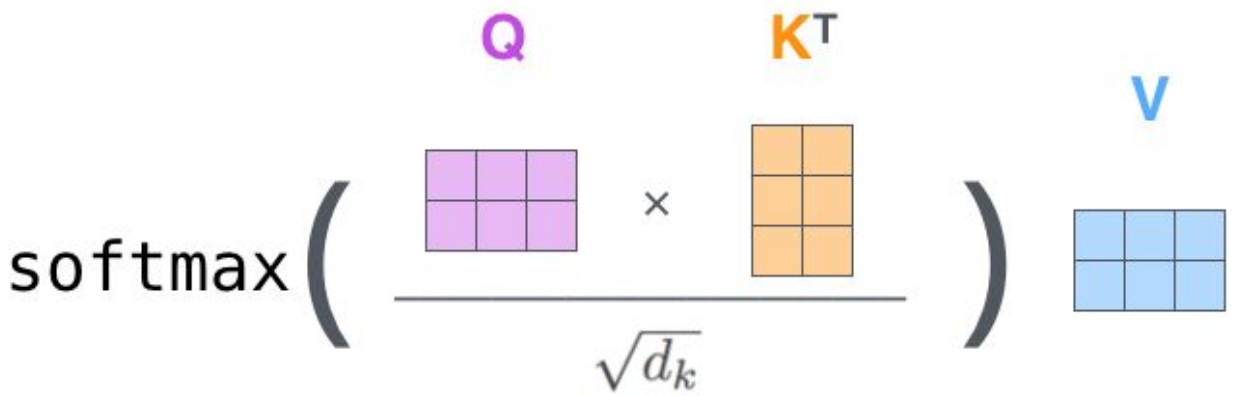
http://jalammar.github.io/illustrated-transformer/

Scaled dot-product attention: Matrix formulation



http://jalammar.github.io/illustrated-transformer/

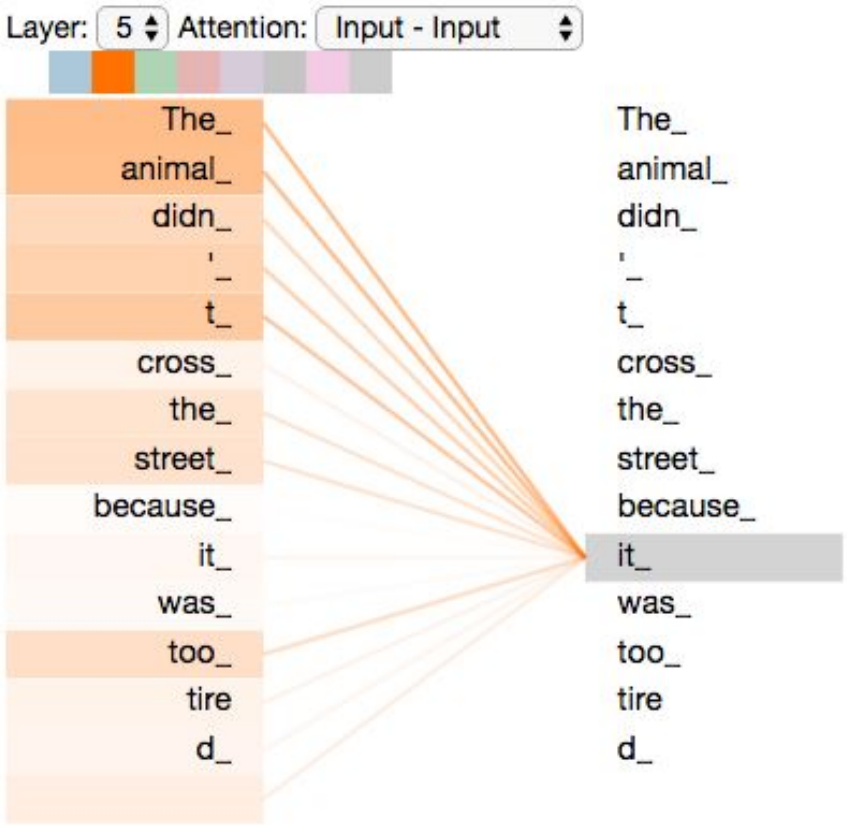
Scaled dot-product attention: Matrix formulation



$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V}$$

http://jalammar.github.io/illustrated-transformer/

Self-attention example

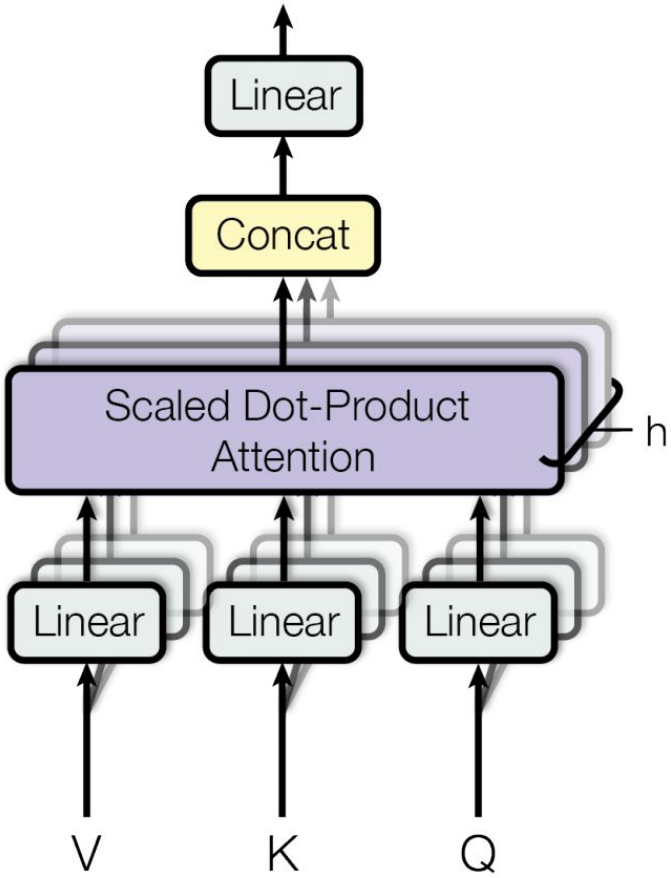


http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention mechanism

Rather than only computing the attention once, the multi-head mechanism runs through the scaled dot-product attention multiple times in parallel. The independent attention outputs are simply concatenated and linearly transformed into the expected dimensions.

According to the paper, “multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.”



Multi-head self-attention mechanism

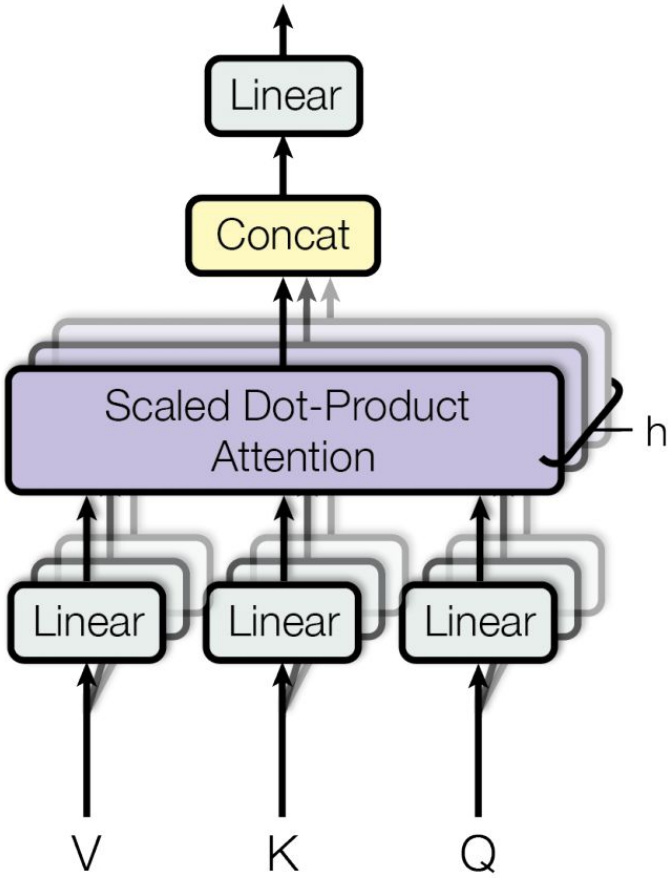
$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1; \ldots; \text{head}_h]\mathbf{W}^O$$
$$\text{where head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$$

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

$$W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

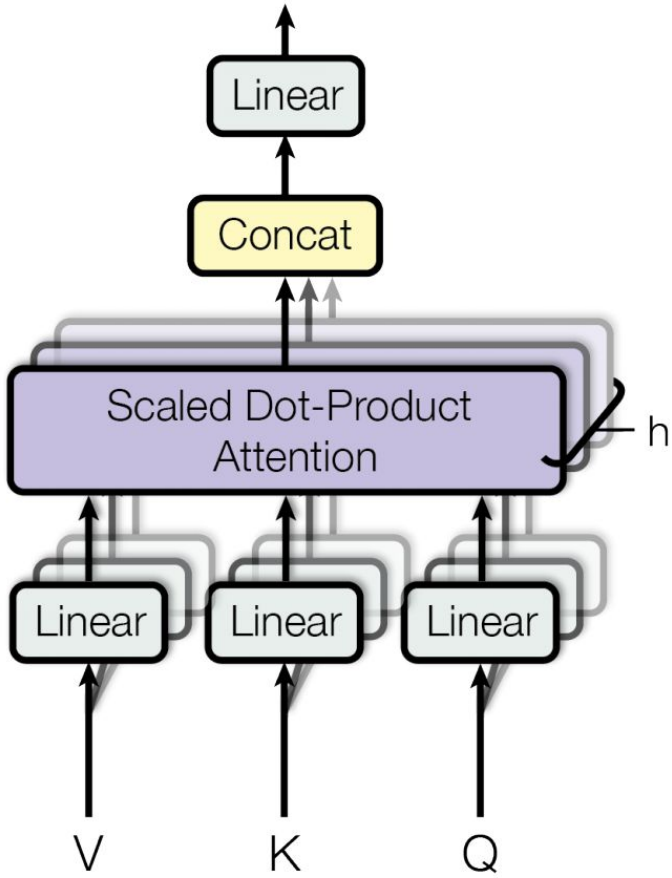
In this work we employ h=8 parallel attention layers, or heads. For each of these we use $d_k=d_v=d_{model}/h=64$.

Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

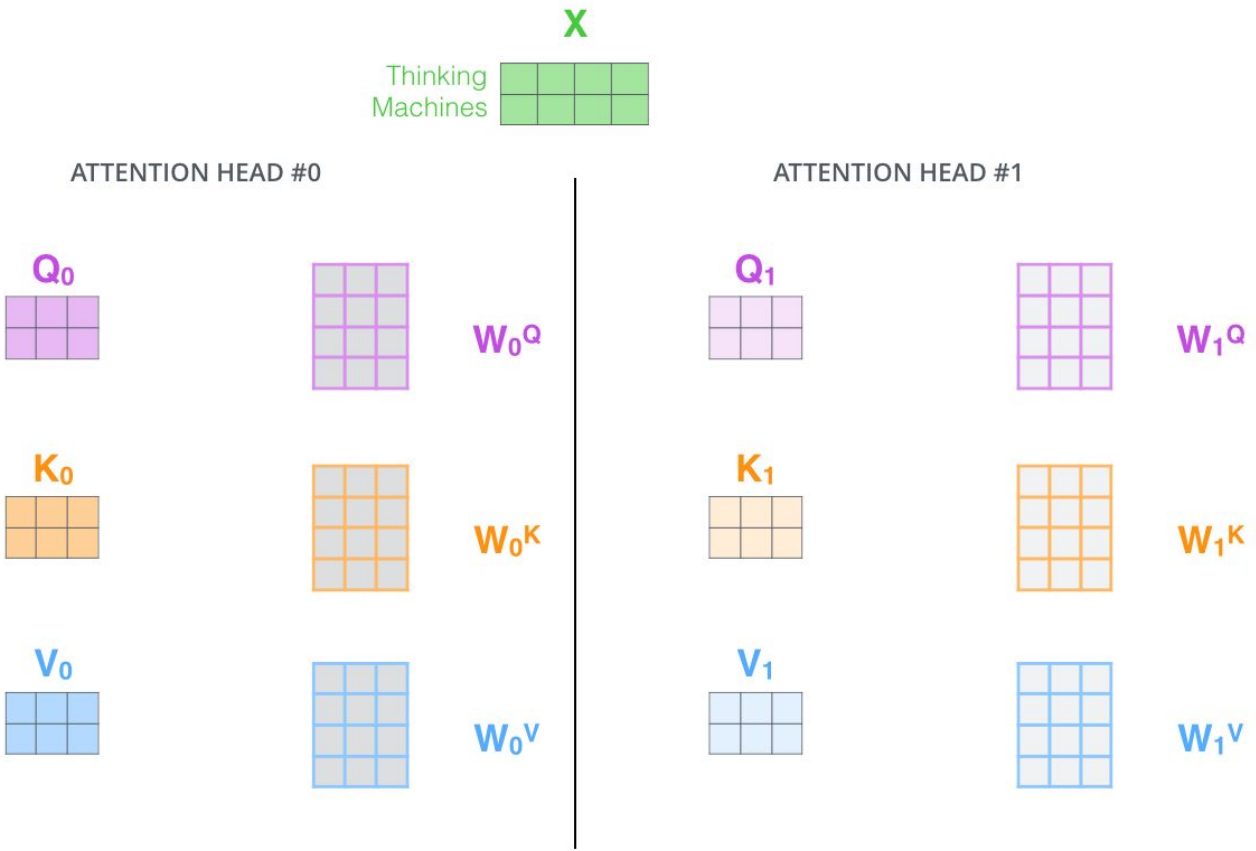


Multi-head self-attention mechanism

Essentially, the Multi-Head Attention is just several attention layers stacked together with different linear transformations of the same input.

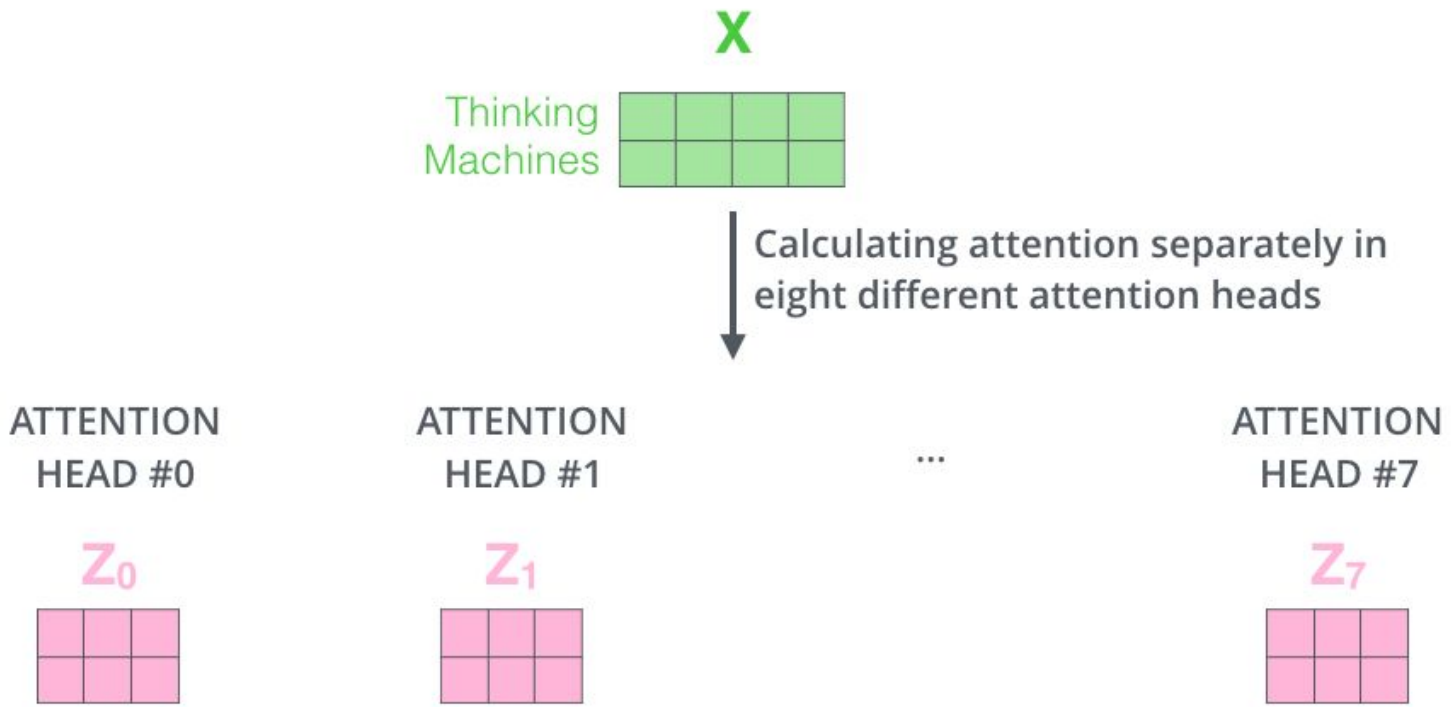


Multi-head self-attention mechanism



http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention mechanism



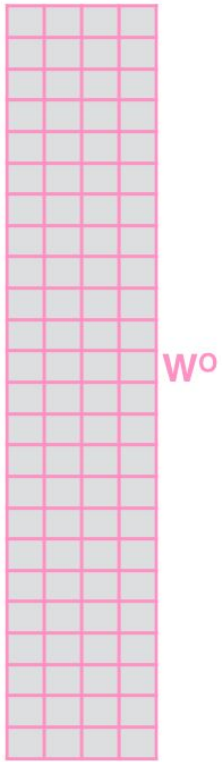
http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention mechanism

1) Concatenate all the attention heads

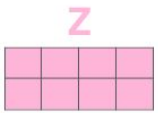
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



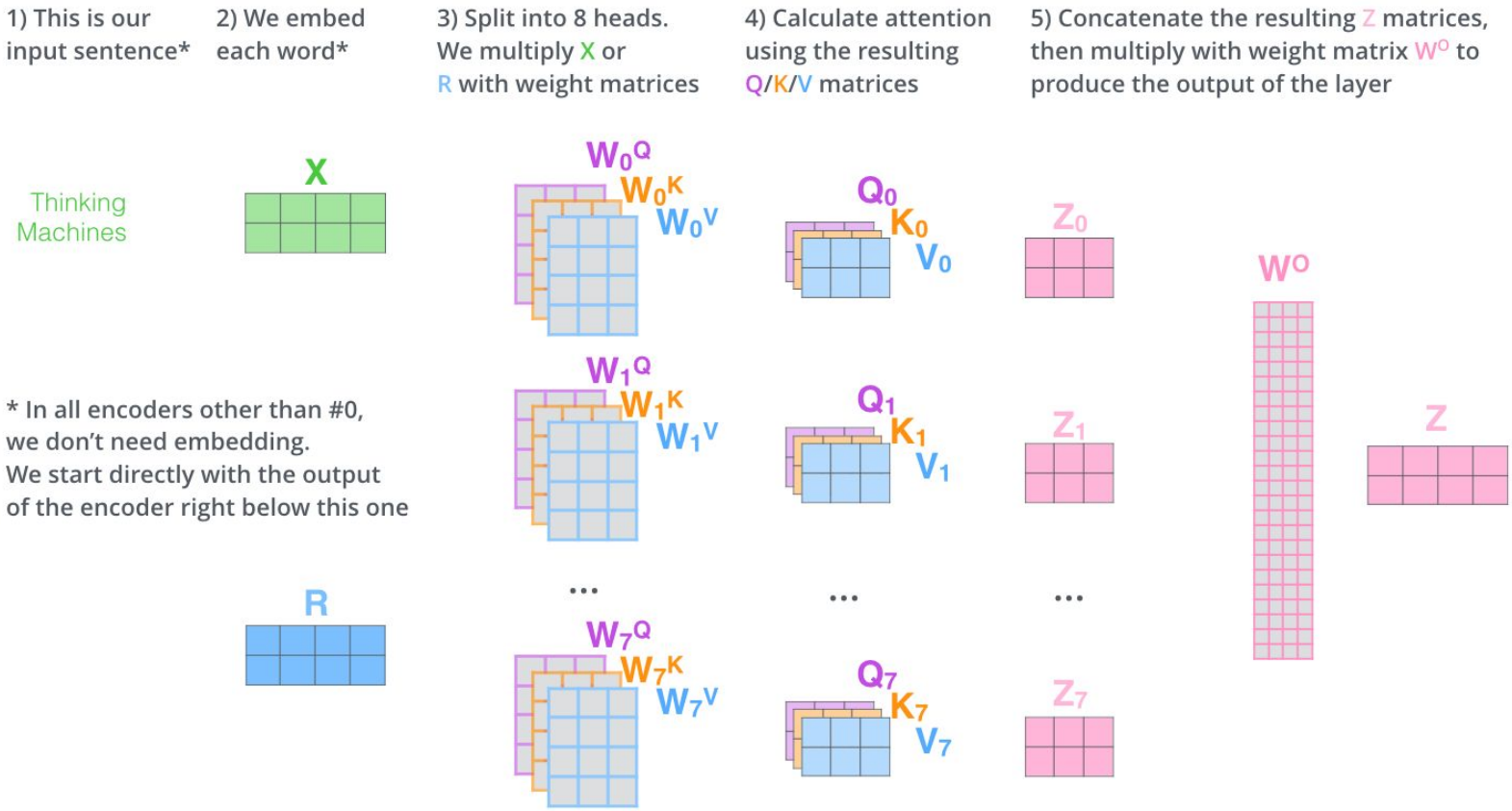
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



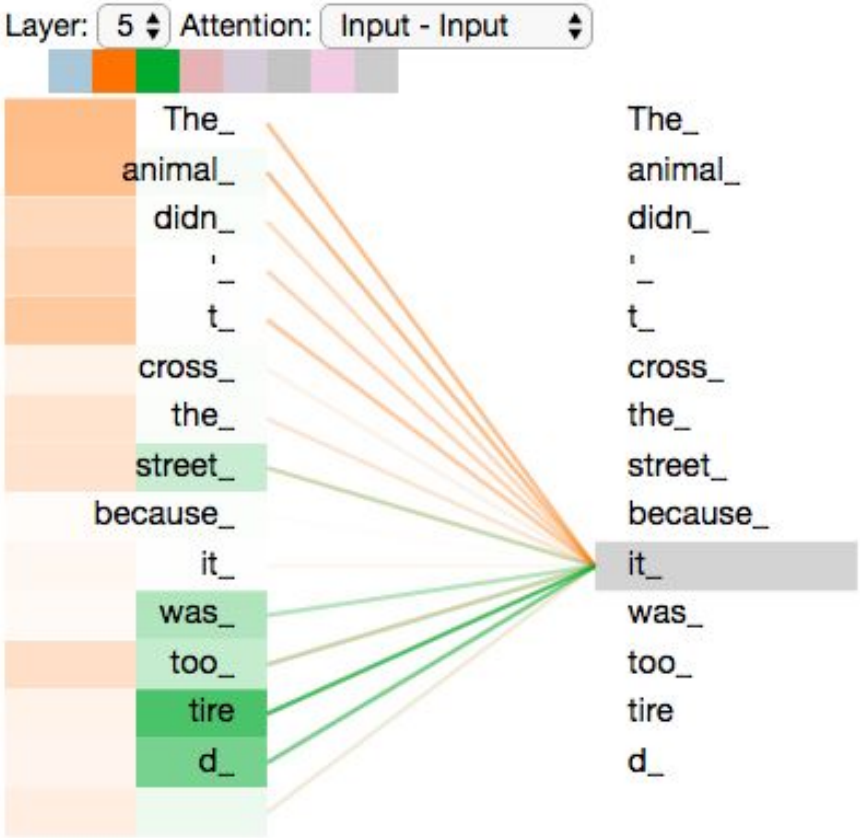
http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention mechanism



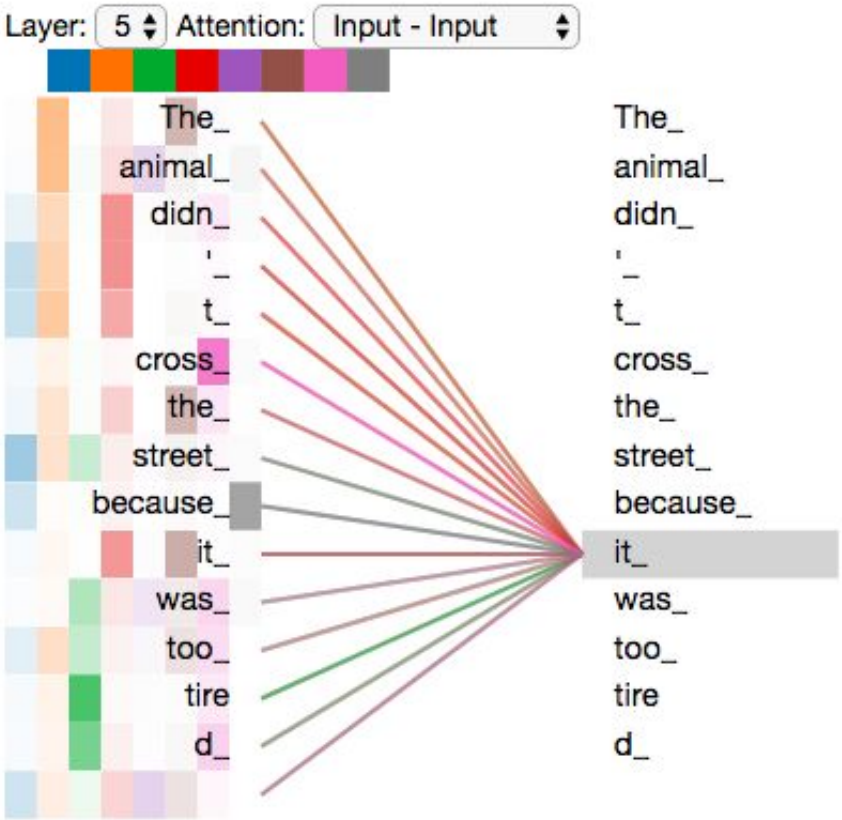
http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention example (2 heads shown)



http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention example (8 heads shown)



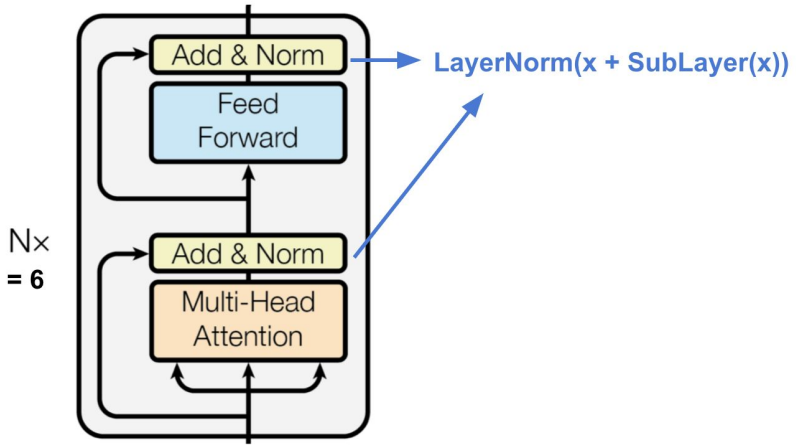
http://jalammar.github.io/illustrated-transformer/

Encoder

The encoder is composed of a stack of N=6 identical layers.

Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network.

To facilitate residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension d_{model}=512.



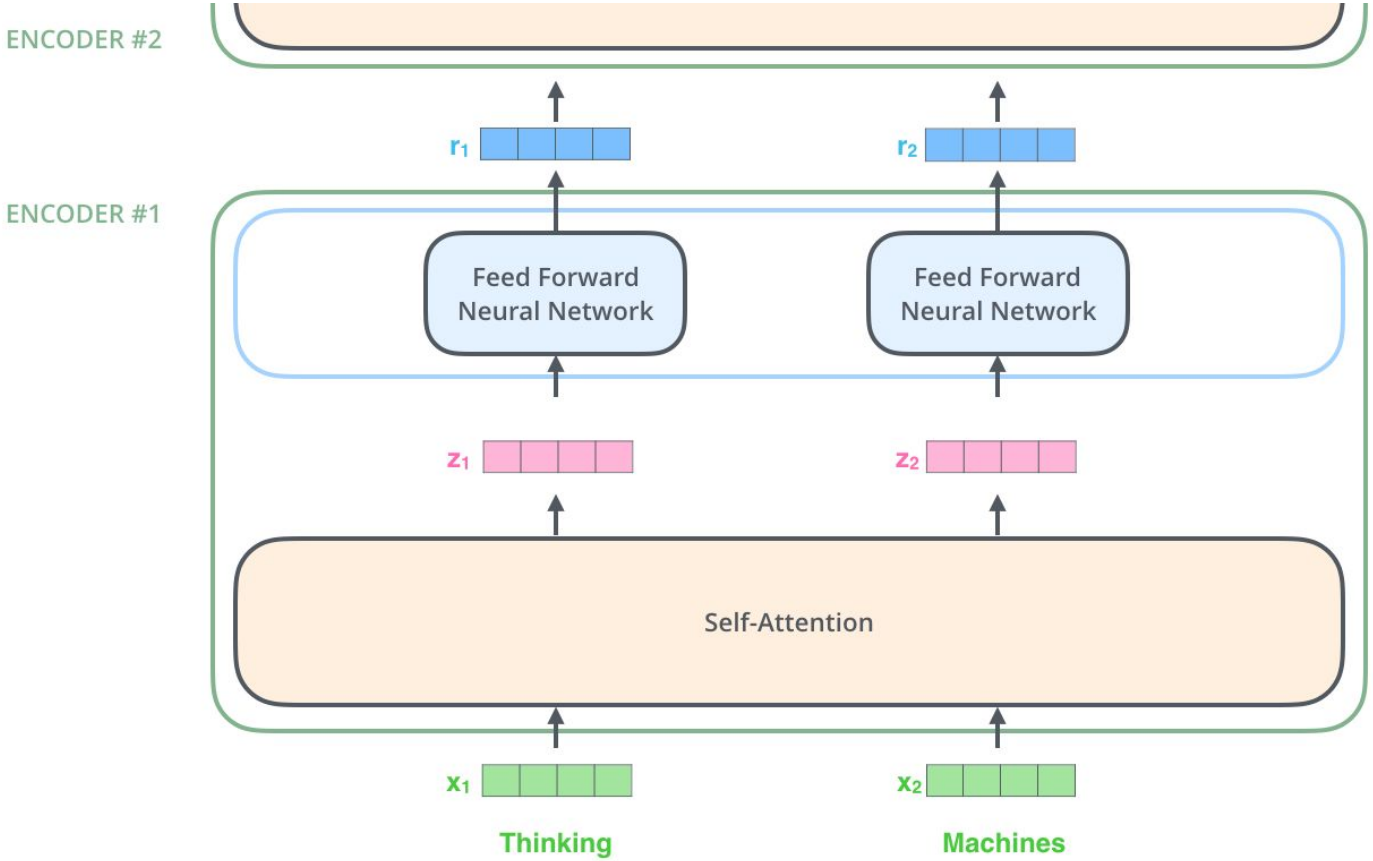
Position-wise Feed-Forward Networks

Each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

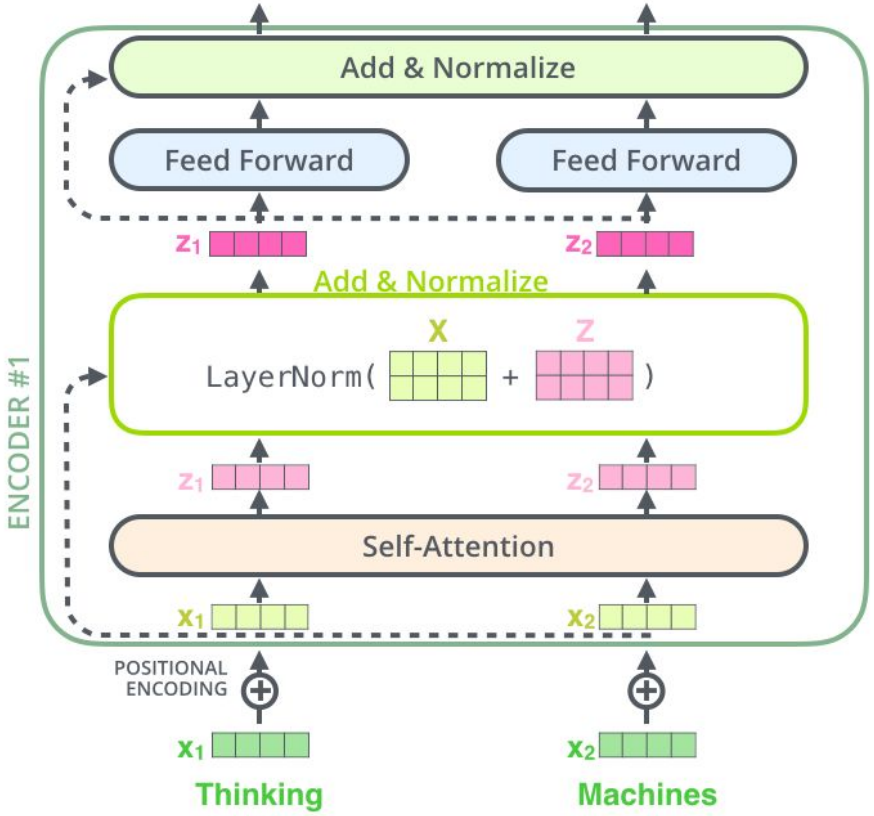
The dimensionality of input and output is d_{model}=512, and the inner-layer has dimensionality d_{ff}=2048.

Encoder



http://jalammar.github.io/illustrated-transformer/

Encoder



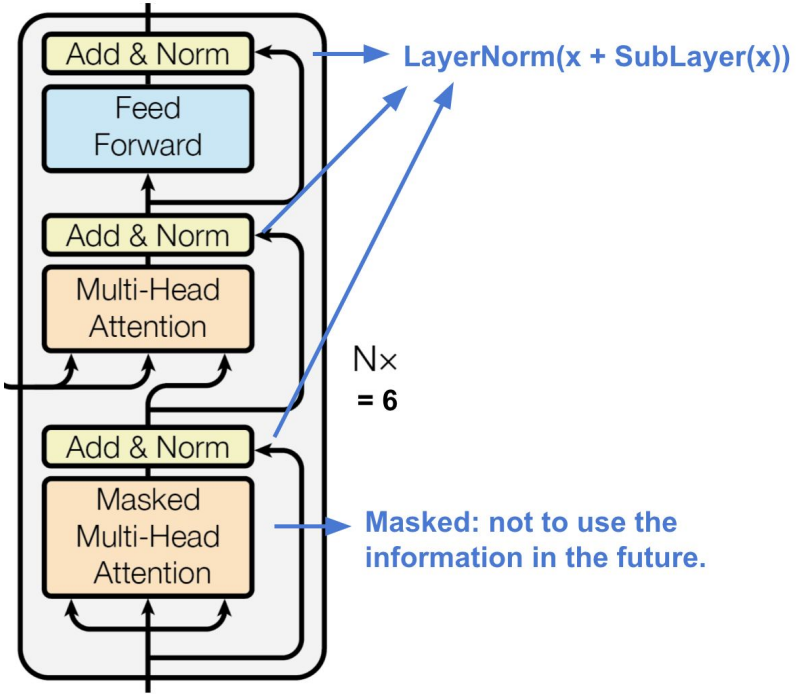
http://jalammar.github.io/illustrated-transformer/

Decoder

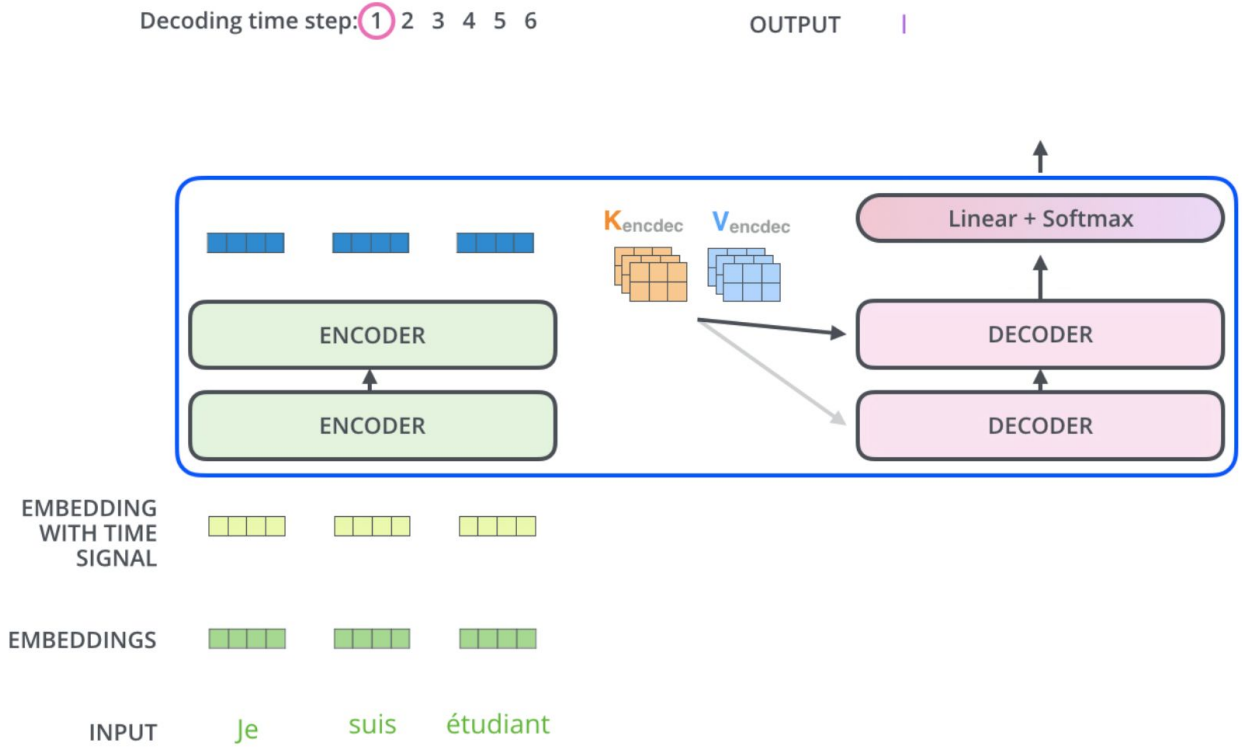
The decoder is also composed of a stack of N=6 identical layers.

In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack.

The Multi-Head Attention layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack.



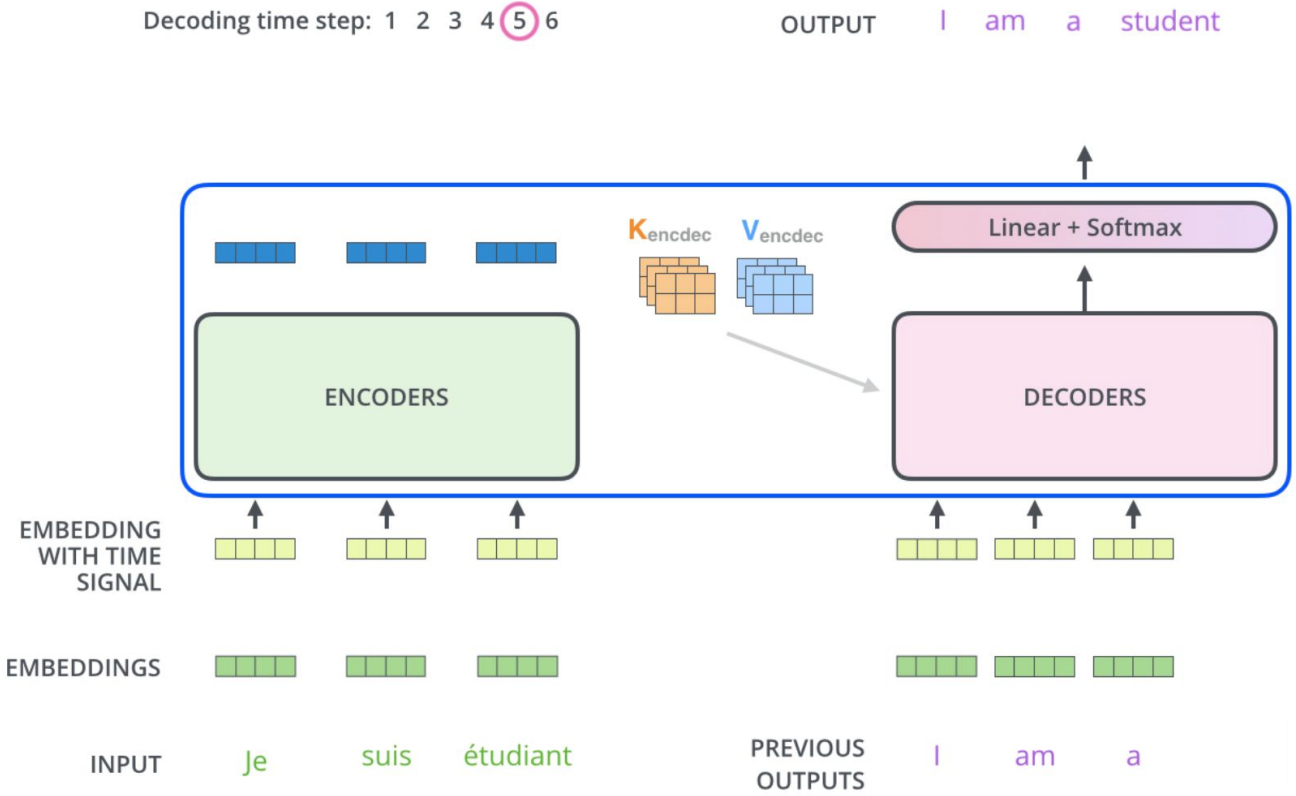
Decoder



After finishing the encoding phase, we begin the decoding phase. Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case).

http://jalammar.github.io/illustrated-transformer/

Decoder



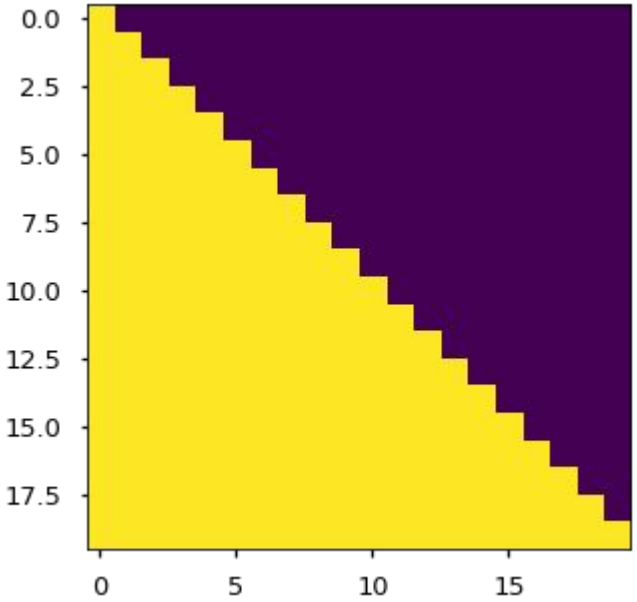
http://jalammar.github.io/illustrated-transformer/

Masking in Decoder

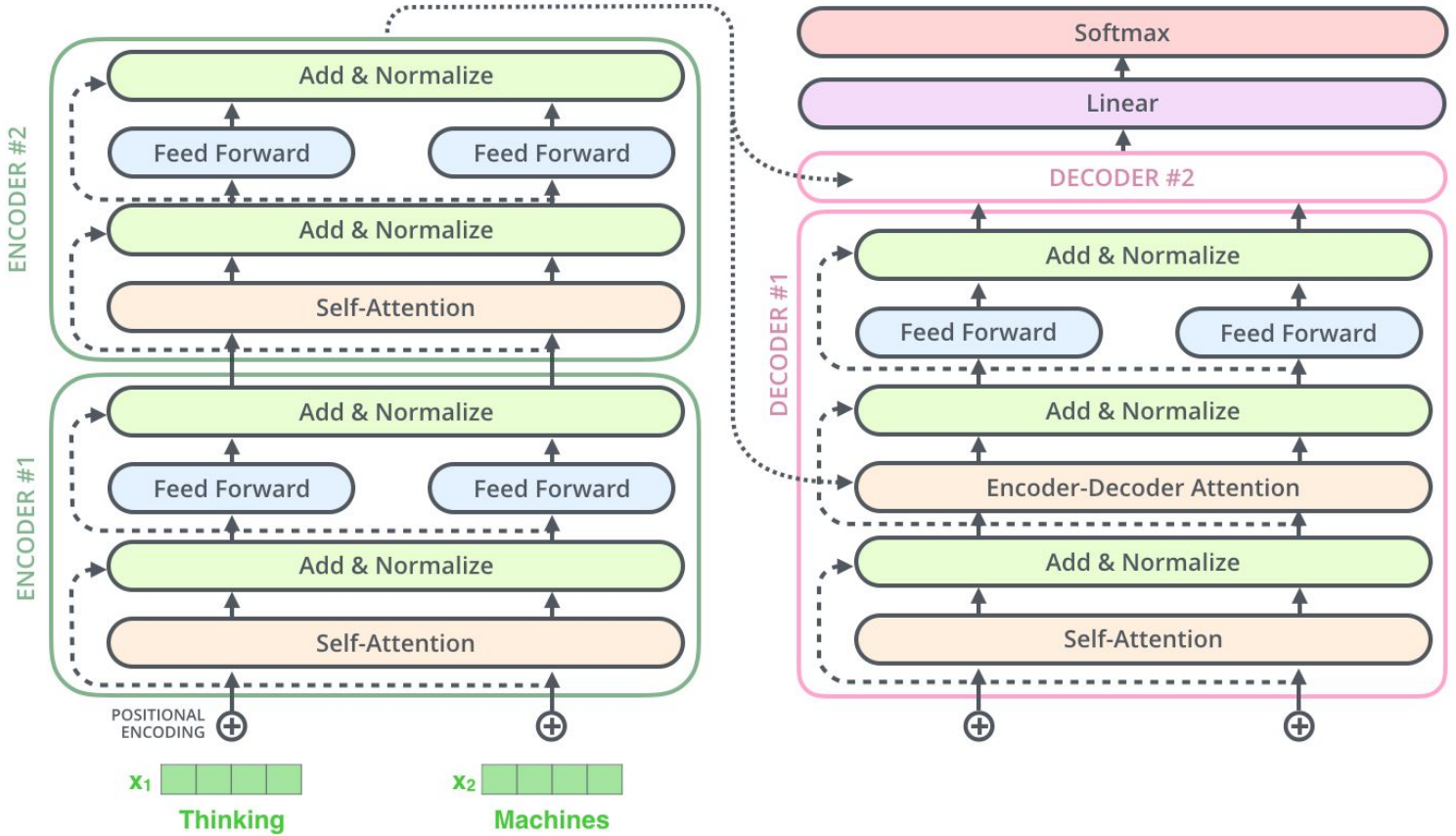
We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions.

In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

The attention mask shows the position each tgt word (row) is allowed to look at (column). Words are blocked for attending to future words during training.



Working pipeline



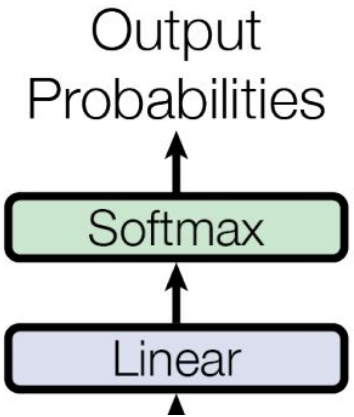
http://jalammar.github.io/illustrated-transformer/

Attention in the model

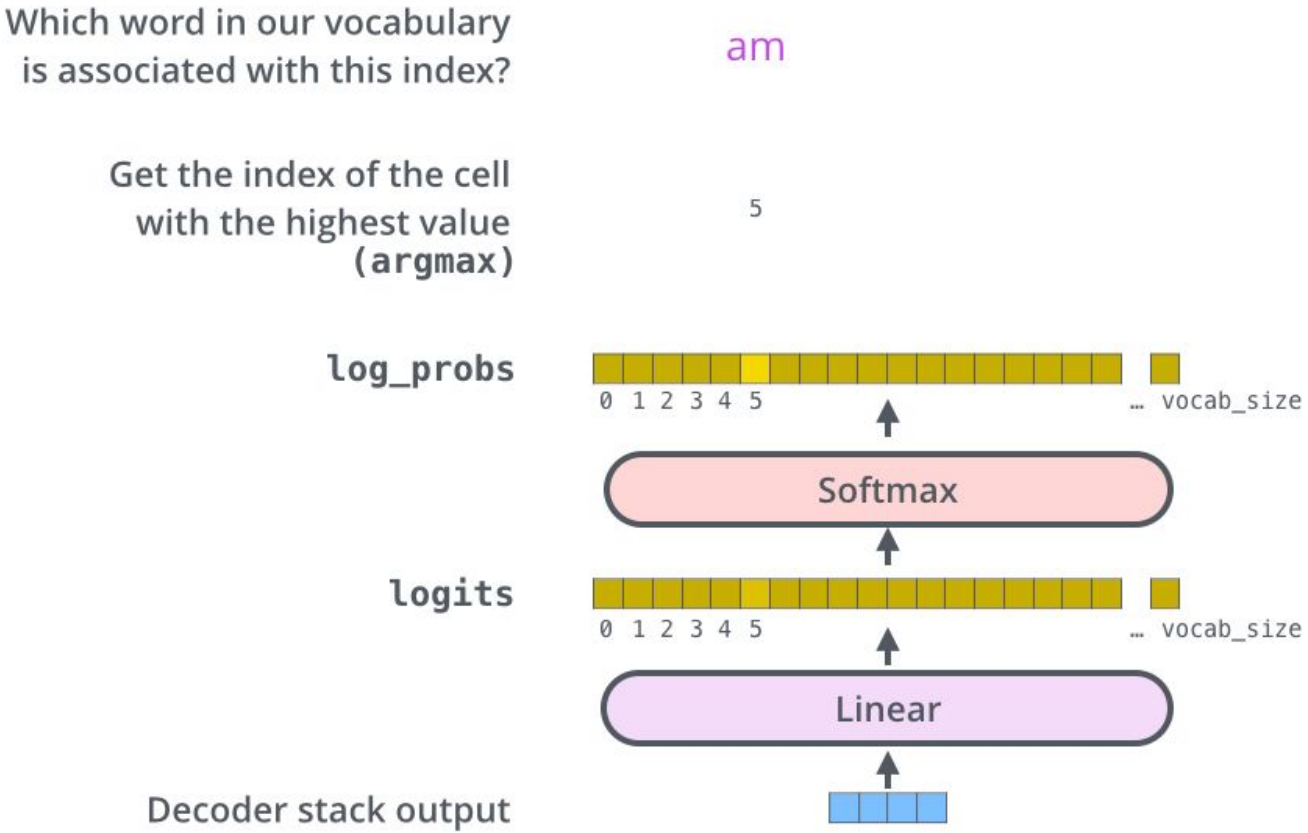
1. In “encoder-decoder attention” layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models.
2. The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
3. Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property (by masking out all values in the input of the softmax which correspond to illegal connections).

The Final Linear and Softmax Layer

We use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities.



The Final Linear and Softmax Layer



http://jalammar.github.io/illustrated-transformer/

Attention visualization

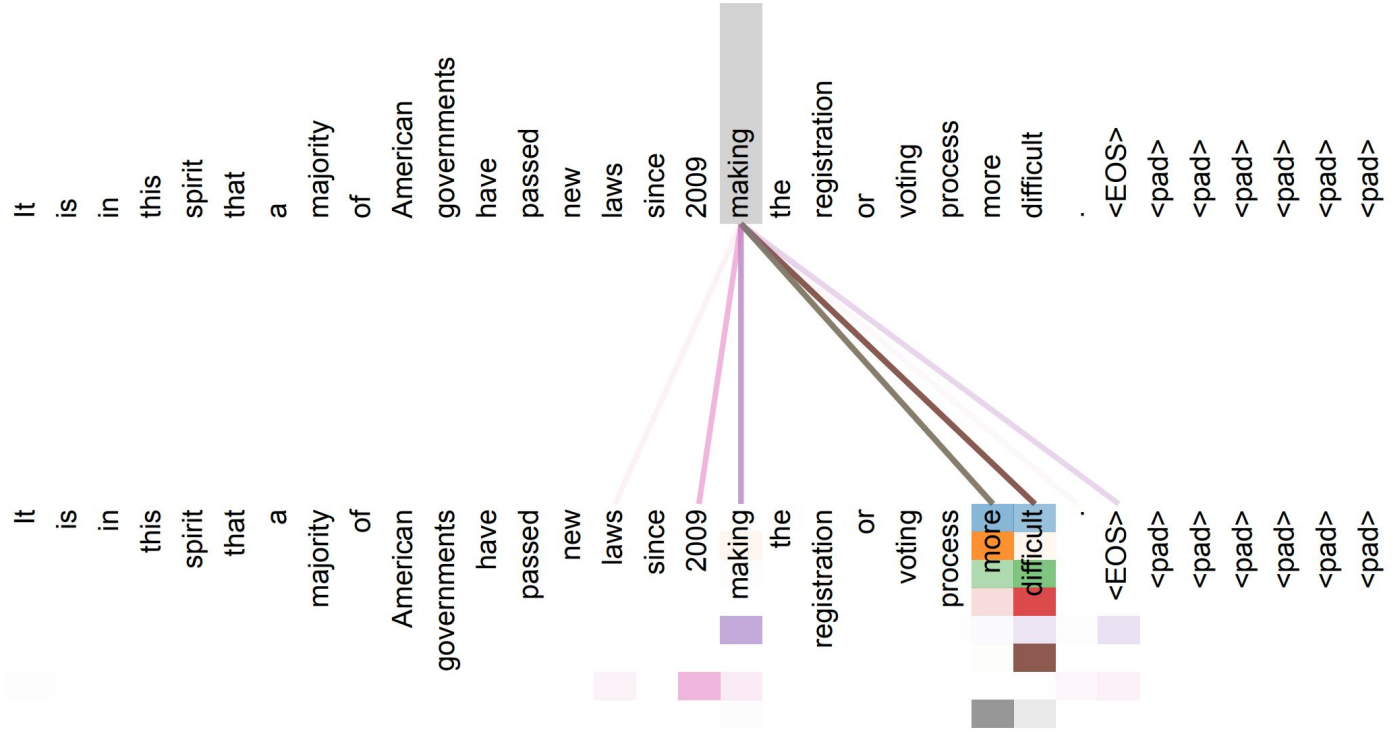


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.

Applying the Transformer to machine translation

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Different architectures for NMT

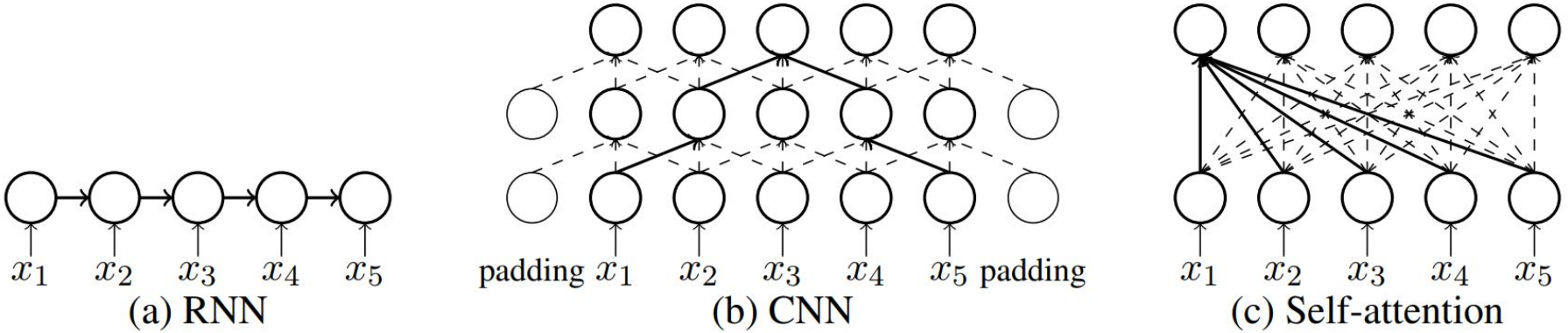


Figure 1: Architectures of different neural networks in NMT.

Why Self-Attention? A Targeted Evaluation of Neural Machine Translation Architectures,
https://arxiv.org/abs/1808.08946

Resources

- The Annotated Transformer
  http://nlp.seas.harvard.edu/2018/04/03/attention.html
- Attention? Attention!
  https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html
- The Illustrated Transformer
  http://jalammar.github.io/illustrated-transformer/
- Paper Dissected: “Attention is All You Need” Explained
  http://mlexplained.com/2017/12/29/attention-is-all-you-need-explained/
- The Transformer – Attention is all you need.
  https://mchromiak.github.io/articles/2017/Sep/12/Transformer-Attention-is-all-you-need/
- When Recurrent Models Don't Need to be Recurrent
  https://bair.berkeley.edu/blog/2018/08/06/recurrent/
- Self-Attention Mechanisms in Natural Language Processing,
  https://www.alibabacloud.com/blog/self-attention-mechanisms-in-natural-language-processing_593968

Code

- https://github.com/tensorflow/tensor2tensor
- Running the Transformer with Tensor2Tensor https://cloud.google.com/tpu/docs/tutorials/transformer
- https://ai.googleblog.com/2017/06/accelerating-deep-learning-research.html
- https://github.com/pytorch/fairseq

Transformer: The next steps

Image Transformer

In this work, we generalize a recently proposed model architecture based on self-attention, the Transformer, to a sequence modeling formulation of image generation with a tractable likelihood. By restricting the self-attention mechanism to attend to local neighborhoods we significantly increase the size of images the model can process in practice, despite maintaining significantly larger receptive fields per layer than typical convolutional neural networks.

Image Transformer, https://arxiv.org/abs/1802.05751

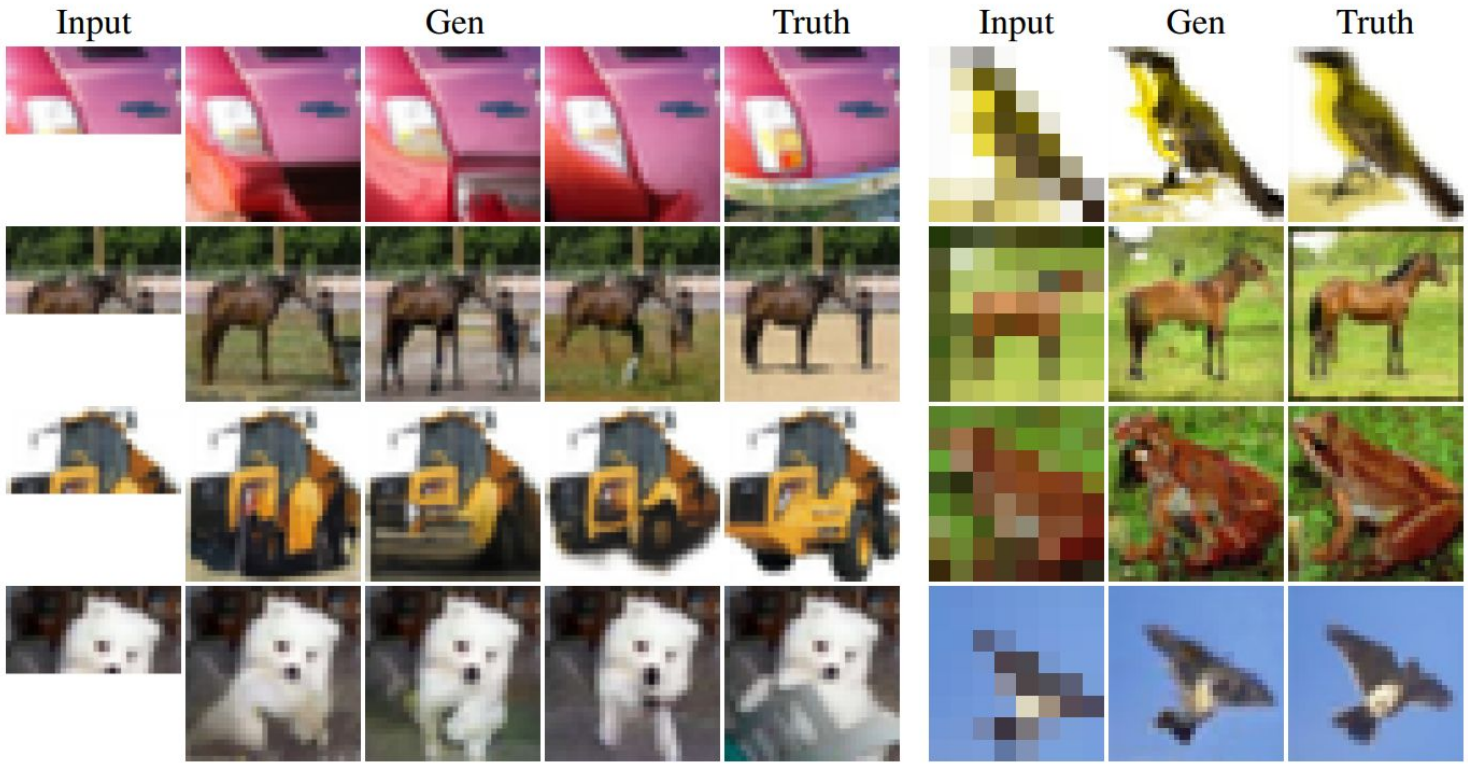


Table 2. On the left are image completions from our best conditional generation model, where we sample the second half. On the right are samples from our four-fold super-resolution model trained on CIFAR-10. Our images look realistic and plausible, show good diversity among the completion samples and observe the outputs carry surprising details for coarse inputs in super-resolution.

Image Transformer, https://arxiv.org/abs/1802.05751

BERT

Bidirectional Encoder Representations from Transformers, or BERT.

BERT is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT representations can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

BERT uses only the encoder part of the Transformer.

Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing,
https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html
Best NLP Model Ever? Google BERT Sets New Standards in 11 Language Tasks
https://medium.com/syncedreview/best-nlp-model-ever-google-bert-sets-new-standards-in-11-language-tasks-4a2a189bc155

BERT

Bidirectional Encoder Representations from Transformers, or BERT

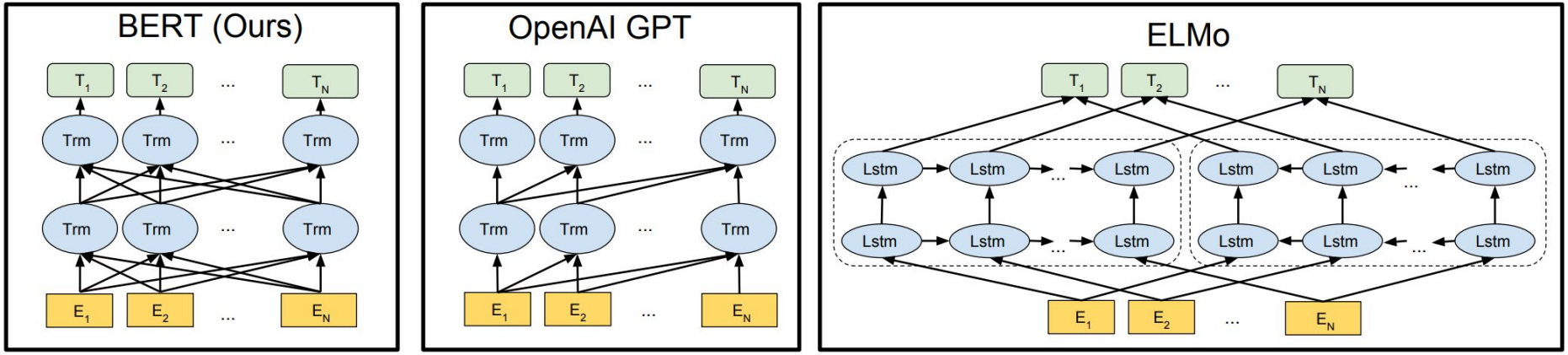


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,
https://arxiv.org/abs/1810.04805

BERT

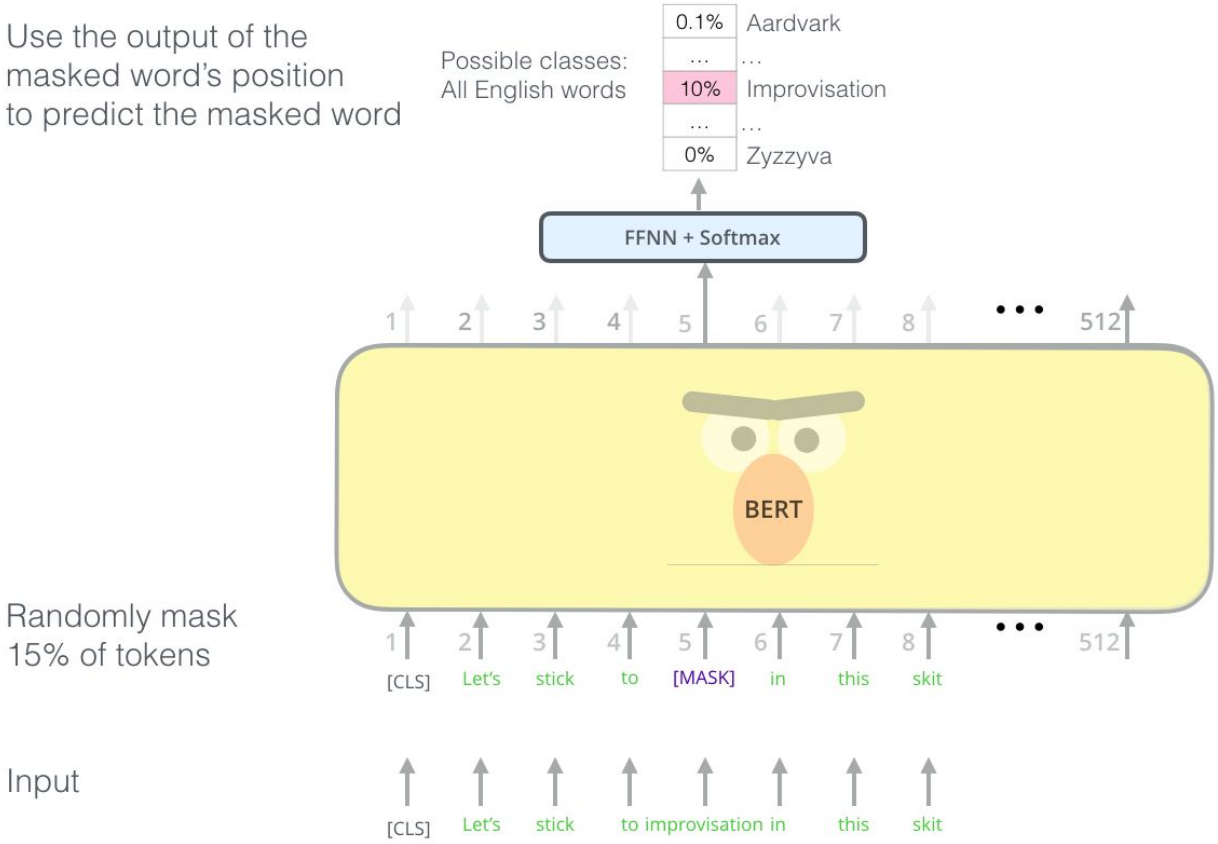
Pre-training tasks:

- **Masked Language Model**: predict random words from within the sequence, not the next word for a sequence of words.
- **Next Sentence Prediction**: give the model two sentences and ask it to predict if the second sentence follows the first in a corpus or not.

Input =

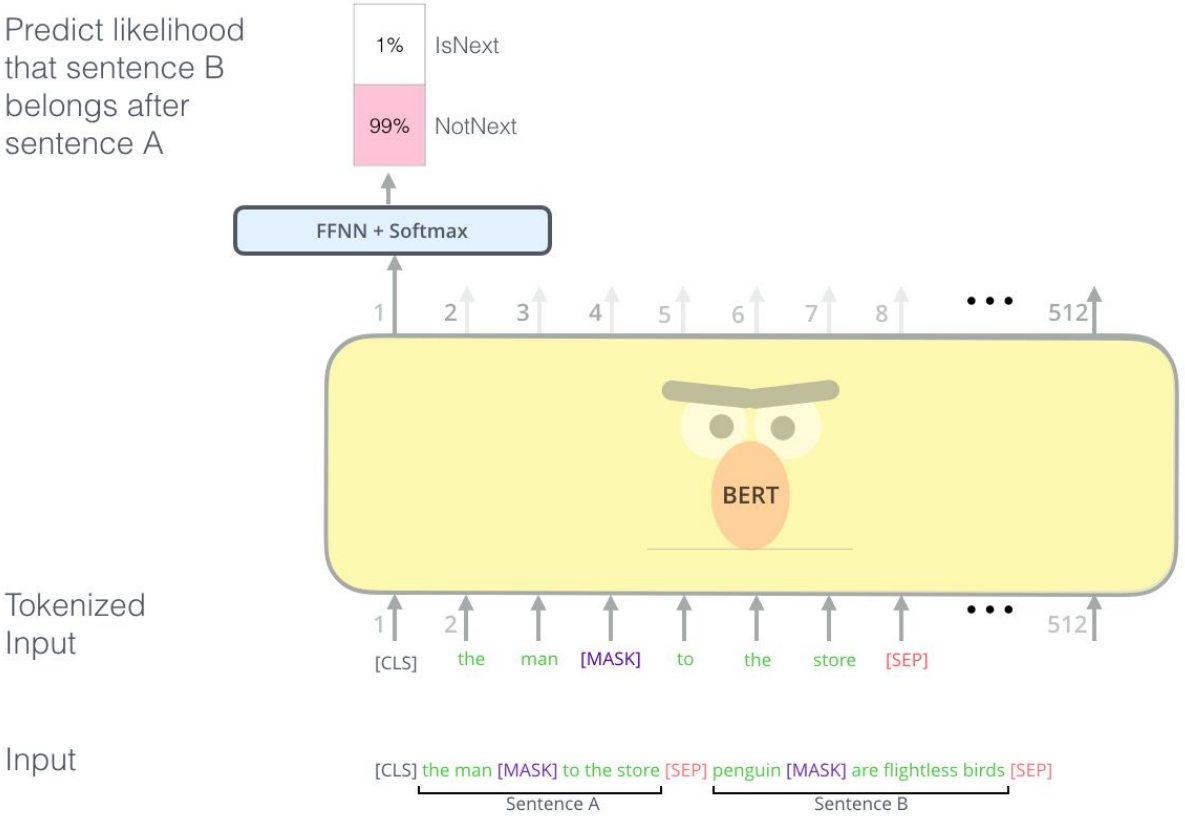
[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]

BERT: masked language model



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

BERT: next sentence prediction



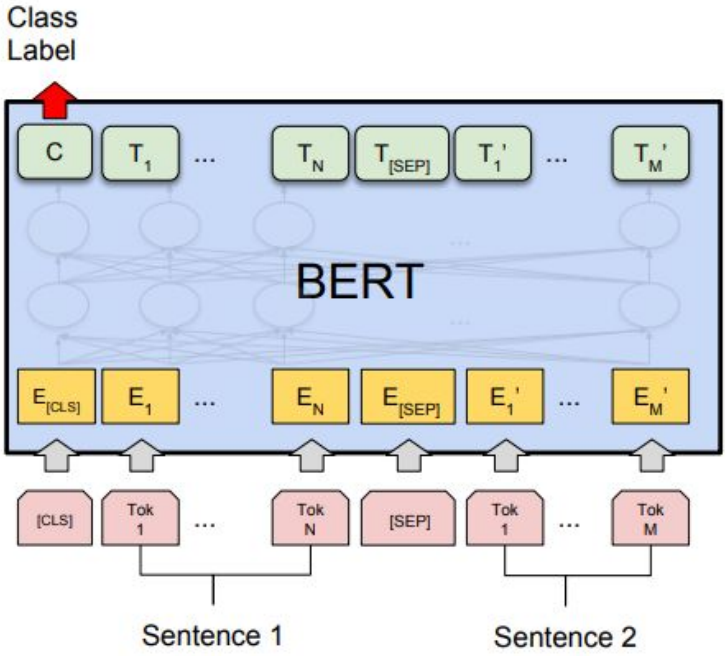
https://jalammar.github.io/illustrated-bert/

BERT

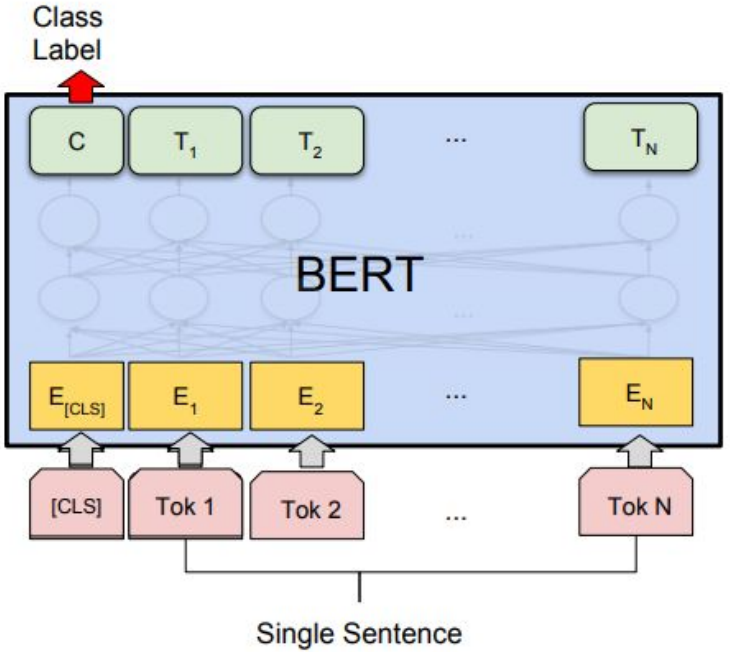
How to use:

- **Fine-tuning approach**: pre-train some model architecture on a LM objective before fine-tuning that same model for a supervised downstream task.
  - Our task specific models are formed by incorporating BERT with one additional output layer, so a minimal number of parameters need to be learned from scratch.
- **Feature-based approach**: learned representations are typically used as features in a downstream model.
  - Not all NLP tasks can be easily be represented by a Transformer encoder architecture, and therefore require a task-specific model architecture to be added.
  - There are major computational benefits to being able to pre-compute an expensive representation of the training data once and then run many experiments with less expensive models on top of this representation

BERT: using fine-tuning approach

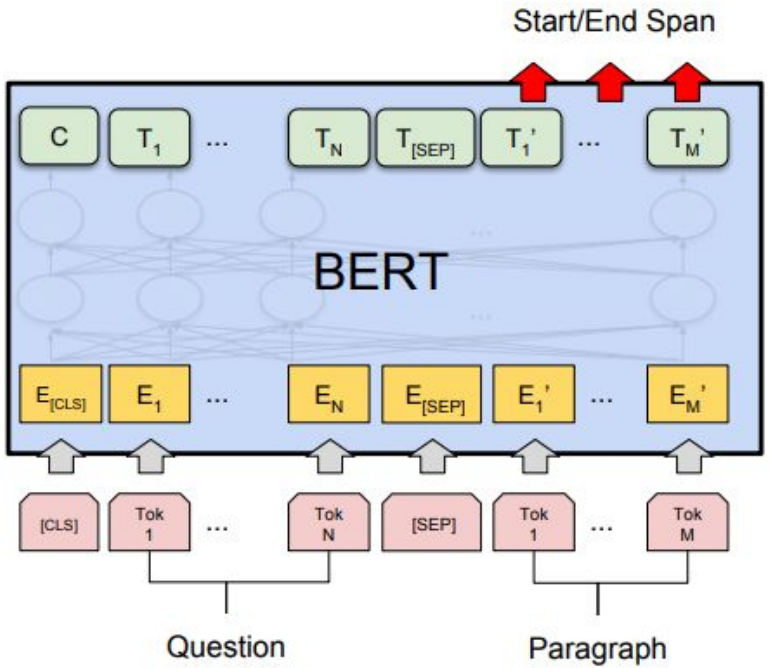


(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

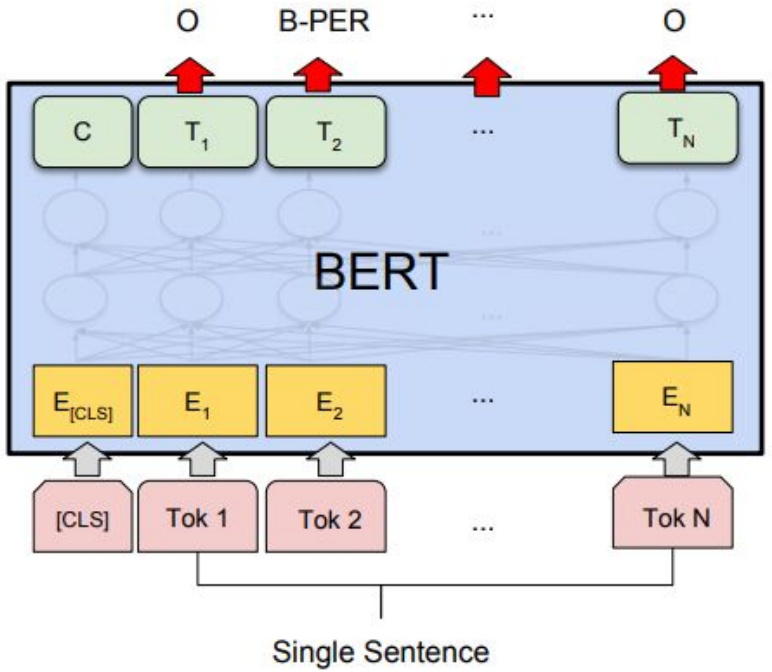


(b) Single Sentence Classification Tasks: SST-2, CoLA

BERT: using fine-tuning approach

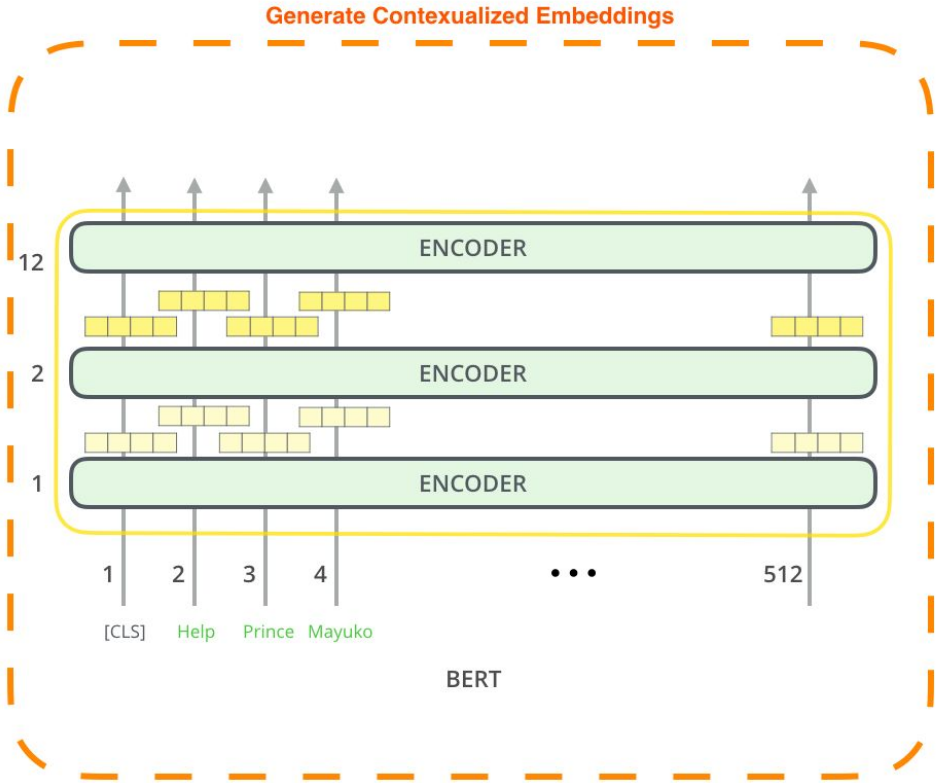


(c) Question Answering Tasks: SQuAD v1.1

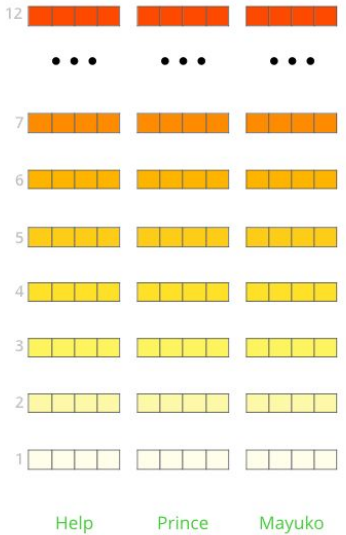


(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

BERT: feature extraction



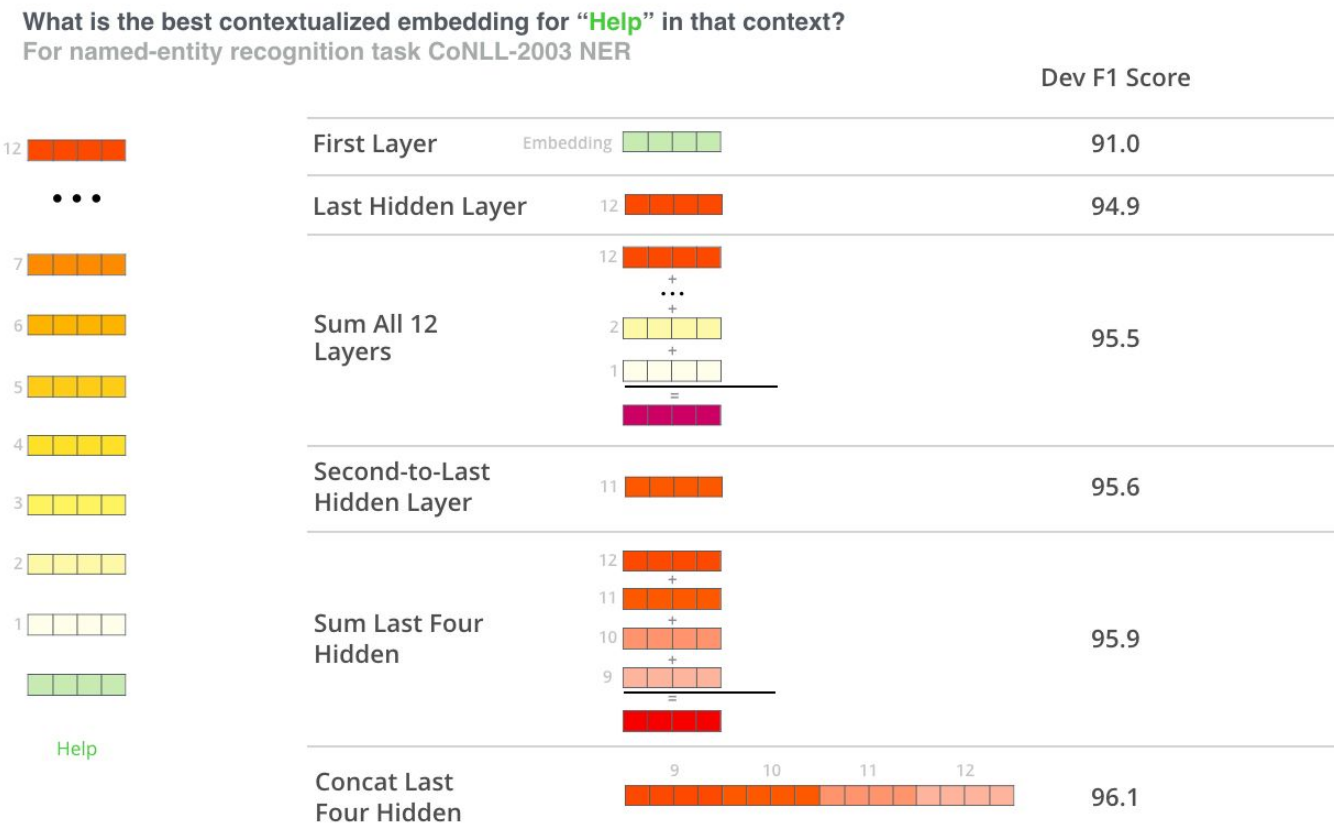
The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

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

BERT: feature extraction



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

Resources

- Dissecting BERT Part 1: Understanding the Transformer
  https://medium.com/@mromerocalvo/dissecting-bert-part1-6dcf5360b07f
- Understanding BERT Part 2: BERT Specifics
  https://medium.com/dissecting-bert/dissecting-bert-part2-335ff2ed9c73
- Dissecting BERT Appendix: The Decoder
  https://medium.com/dissecting-bert/dissecting-bert-appendix-the-decoder-3b86f66b0e5f
- The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)
  https://jalammar.github.io/illustrated-bert/

Code

- https://github.com/google-research/bert
- https://github.com/huggingface/pytorch-pretrained-BERT

Universal Transformer

In “Universal Transformers” we extend the standard Transformer to be computationally universal (**Turing complete**) using a novel, efficient flavor of parallel-in-time recurrence which yields stronger results across a wider range of tasks.

Crucially, where an RNN processes a sequence symbol-by-symbol (left to right), the Universal Transformer processes all symbols at the same time (like the Transformer), but then refines its interpretation of every symbol in parallel over a variable number of recurrent processing steps using self-attention.

This parallel-in-time recurrence mechanism is both faster than the serial recurrence used in RNNs, and also makes the Universal Transformer more powerful than the standard feedforward Transformer.

Moving Beyond Translation with the Universal Transformer
https://ai.googleblog.com/2018/08/moving-beyond-translation-with.html

Universal Transformer

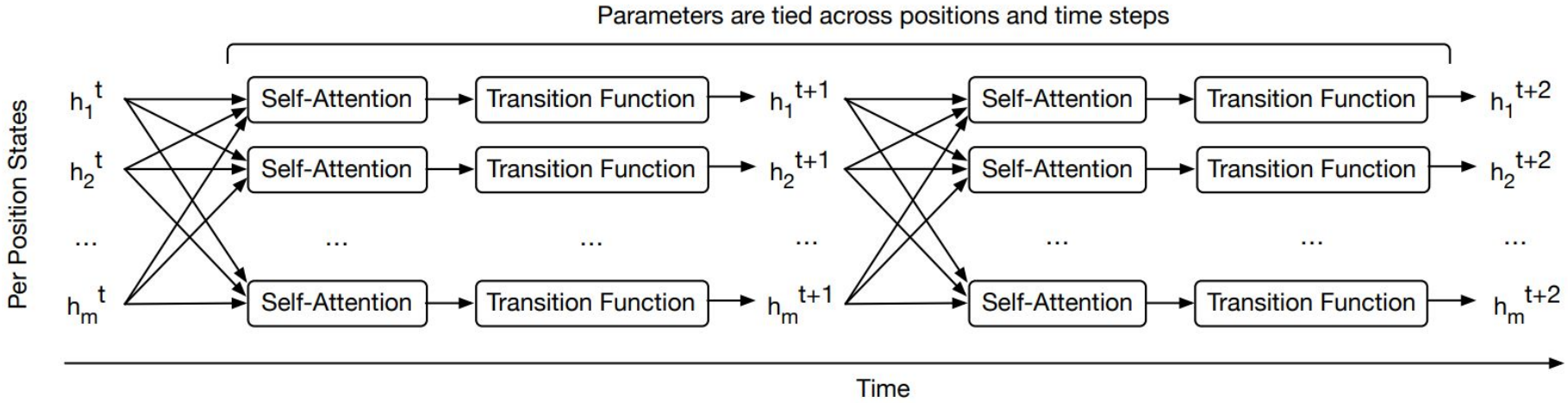


Figure 1: The Universal Transformer repeatedly refines a series of vector representations for each position of the sequence in parallel, by combining information from different positions using self-attention and applying a recurrent transition function. We show this process over two recurrent time-steps. Arrows denote dependencies between operations. Initially, h^0 is initialized with the embedding for each symbol in the sequence. h_i^t represents the representation for input symbol $1 \leq i \leq m$ at recurrent time-step t.

Universal Transformers, https://arxiv.org/abs/1807.03819

Universal Transformer

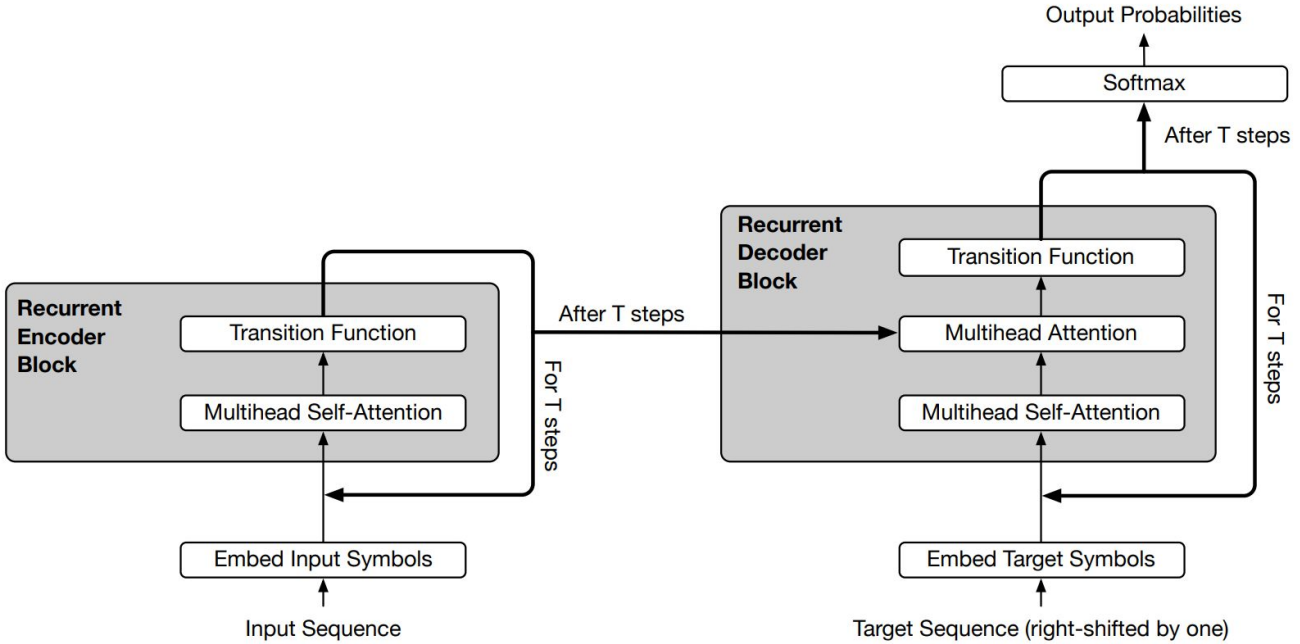


Figure 2: The recurrent blocks of the Universal Transformer encoder and decoder. This diagram omits position and time-step encodings as well as dropout, residual connections and layer normalization. A complete version can be found in the appendix. The Adaptive Universal Transformer dynamically determines the number of steps T for each position using ACT.

Universal Transformers, https://arxiv.org/abs/1807.03819

Thanks!

grigory.sapunov@ieee.org
gs@inten.to