Is Attention All You Need?

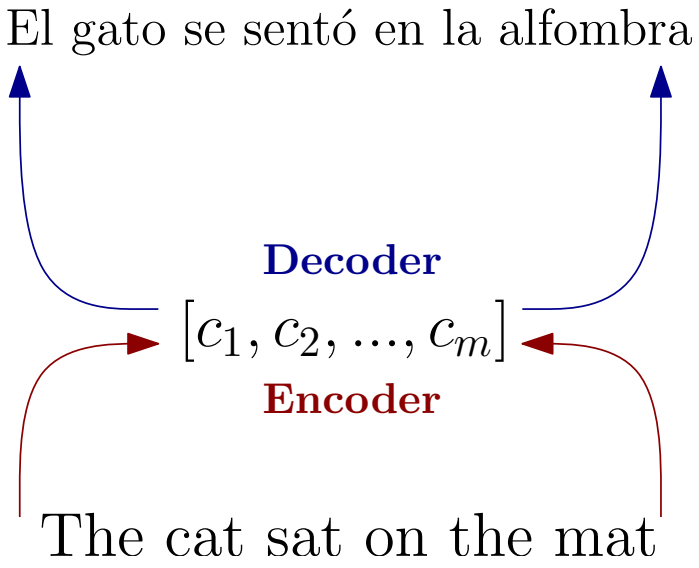
Álvaro Peris
lvapeab@prhlt.upv.es

Pattern Recognition and Human Language Technologies
Research Center
Universitat Politècnica de València

Presentation Outline

Introduction: Sequence-to-sequence Neural Machine Translation

- Distributed representation of words and sentences.
- Input and output sequences $\rightarrow$ ~~Recurrent Neural Networks~~.
- Encoder – Decoder approach:



Sequence-to-Sequence Neural Machine Translation

- Source sentence: $x_1^J = x_1, \ldots, x_J, x_j \in V_x$.
- Target sentence: $y_1^I = y_1, \ldots, y_I, y_i \in V_y$.

- Training objective: Maximum likelihood:

$$\hat{\theta} = \arg\max_{\theta} \sum_{n=1}^{N} \sum_{i=1}^{I_n} \log(p(y_i^{(n)} \mid y_1^{i-1(n)}, x^{(n)}; \theta))$$

  - Parallel corpus: $\mathcal{S} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$.
  - Model parameters: θ.

- Stochastic Gradient Descent.

Sequence-to-Sequence Machine Translation

1. Project source words to a continuous vector → *Source word embedding*.
2. Generate a compact representation of the source sentence → *Encoder NN*.
3. Obtain a continuous representation of the target sentence → *Decoder NN*.
4. Decode target representation to target words → *Target word embedding* .

Sequence-to-Sequence Machine Translation

- Word embedding: Linear projection from discrete words to a vector of size m.
- Input: One-hot coded source word:

$$x_j \in \mathbb{N}, 0 \leq x_j \leq |V|$$

V: Vocabulary.

- Linear projection: embedding matrix $\mathbf{W}_s$ $(|V| \times m)$:

$$x_j \mathbf{W}_s = \mathbf{x}_j \in \mathbb{R}^m$$

- Estimated with the rest of model parameters.
- Limited vocabulary size! $\rightarrow$ **Solved via sub-words (byte pair encoding)**.

Recurrent encoder

- Recurrent neural network (RNN):
  - Encoder: Bidirectional RNN.
  - Input sequence analyzed in both directions.
  - Sequence of annotations:

$$\mathbf{h}_1^J = \mathbf{h}_1, \ldots, \mathbf{h}_J; \mathbf{h}_j = [\mathbf{h}_j^{\mathrm{f}}; \mathbf{h}_j^{\mathrm{b}}]$$

    - Forward layer: $\mathbf{h}_j^{\mathrm{f}}$.
    - Backward layer: $\mathbf{h}_j^{\mathrm{b}}$.

Attention-based encoder–decoder

- Context vector ($\mathbf{c}_i$): Dynamic representation of the relevant part of the sentence at time i:

$$\mathbf{c}_i = \sum_{j=1}^{J} \alpha_{ij} \mathbf{h}_j$$

  - Weights provided by a soft alignment model.
  - Add (Bahdanau *et al.* , 2015):

$$e_{ij} = \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a \mathbf{h}_j)$$

  - Dot (Luong *et al.* , 2015):

$$e_{ij} = \mathbf{s}_i^\top \mathbf{h}_j$$

- And normalized with the softmax function:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{J} \exp(e_{ik})}$$

Recurrent decoder

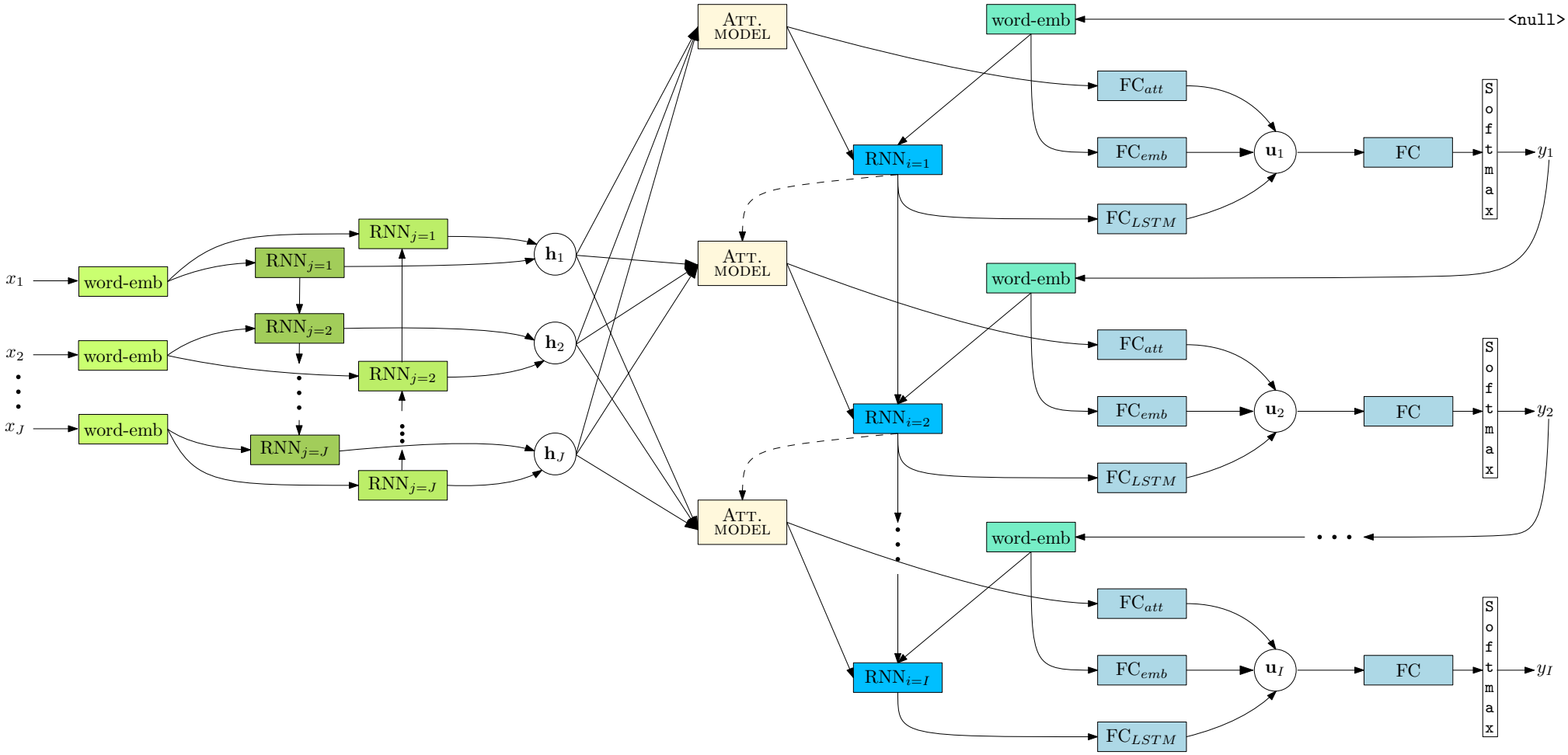
- Recurrent neural network:
  - Input:
    - Context vector $\mathbf{c}_i$.
    - Embedding of the previous word $\mathbf{E}y_{i-1}$.
    - (and the recurrent hidden state $\mathbf{s}_{i-1}$).
  - Output: current hidden state $\mathbf{s}_i$.

- Word probability:

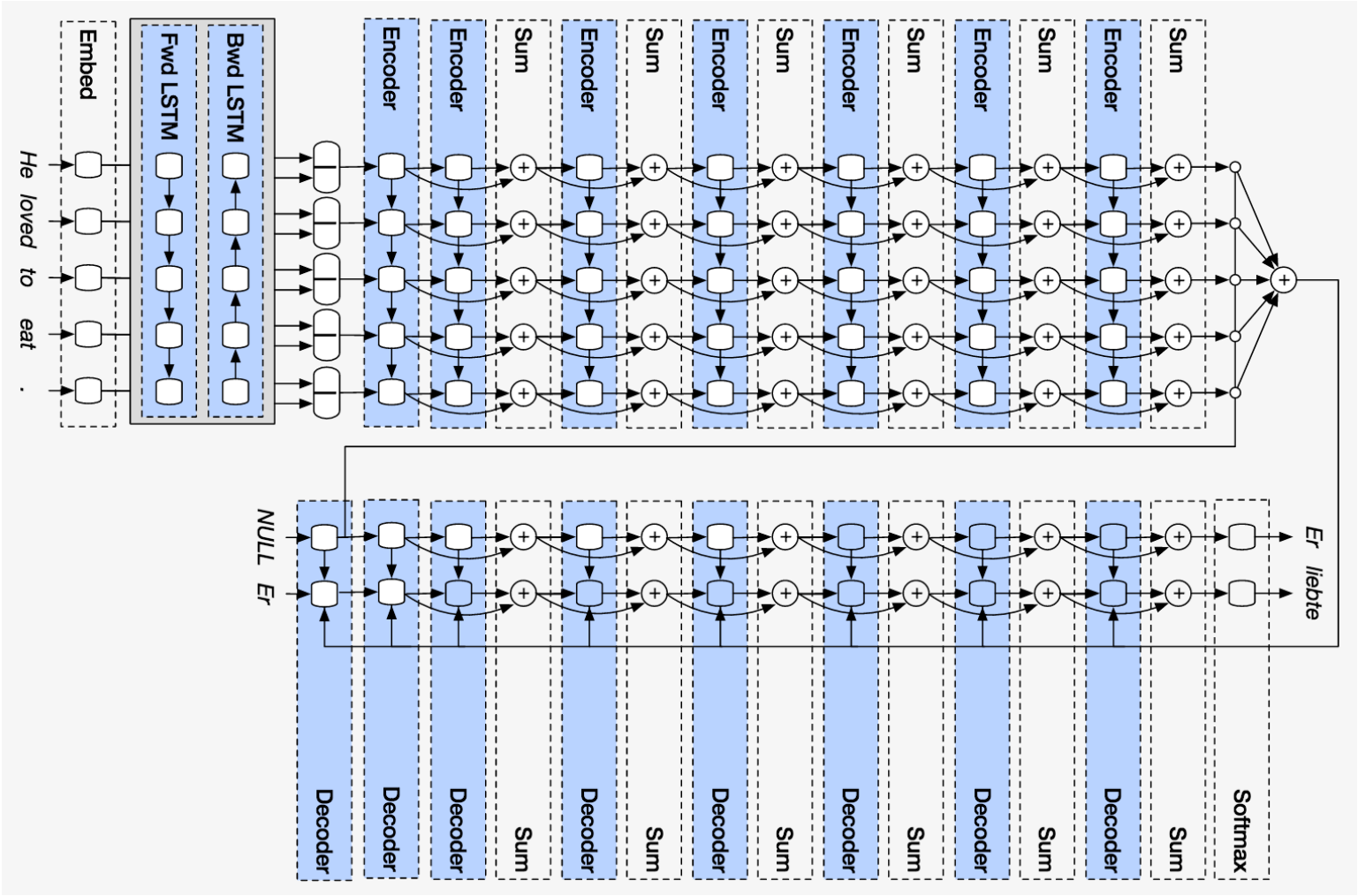
$$p(y_i|y_1^{i-1}, x_1^J) = \text{softmax}(\mathbf{V}g(\mathbf{E}y_{i-1}, \mathbf{s}_i, \mathbf{c}_i))$$

  - g: Deep output layer.
  - $\mathbf{V}$: Weight matrix.

Recurrent encoder-decoder with attention



Deep Recurrent encoder-decoder with attention: GNMT



Recurrent encoder-decoder with attention

- Problem: RNN imply sequential computations. Non-parallelization.
- Idea: Use attention exclusively $\rightarrow$ Transformer [1] model (Vaswani *et al.* , 2017):
  - Stack of N attention blocks in encoder and in decoder.
  - Each block contains several attention and feed-forward layers.
  - Residual connections + layer normalization.
  - All layers produce outputs of dimension d_{model}.
- What is attention?
  - Mapping of a query ($\mathbf{q} \in \mathbb{R}^{d_k}$) and a set of key-values ($\mathbf{k} \in \mathbb{R}^{d_k}$, $\mathbf{v} \in \mathbb{R}^{d_v}$) to an output.

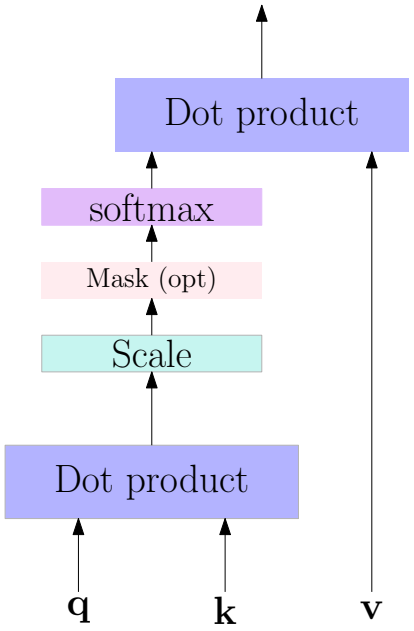
[1] In the following, slight changes in the nomenclature of attention models

Scaled dot-product attention

- Mapping of a query ($\mathbf{q} \in \mathbb{R}^{d_k}$) and a set of key-values ($\mathbf{k} \in \mathbb{R}^{d_k}$, $\mathbf{v} \in \mathbb{R}^{d_v}$) to an output.

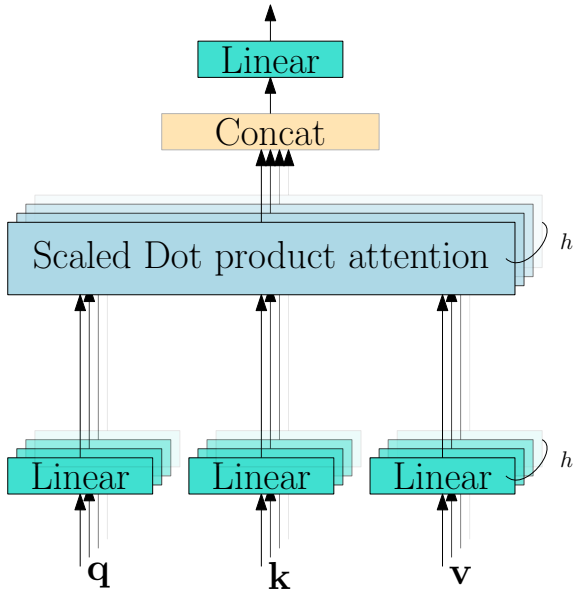
- Scaled dot-product attention:

$$\text{Attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \underbrace{\text{softmax}\left(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{d_k}}\right)}_{\alpha} \mathbf{v}$$



Multi-head attention

- Learn multiple representations.
- Multi-head attention:
  1. Linearly project $\mathbf{q}$, $\mathbf{k}$ and $\mathbf{v}$ h times (head) with different learned linear projections.
  2. Perform dot attention on the projections.
  3. Concatenate the attended vectors.



- Multi-head attention:

$$\text{MultiHead}(\mathbf{q},\mathbf{k},\mathbf{v}) = \text{Concat}\big(\text{head}_1,\ldots,\text{head}_h\big)\mathbf{W}^O$$

$$\text{head}_i = \text{Attention}(\mathbf{q}\mathbf{W}_i^Q, \mathbf{k}\mathbf{W}_i^K, \mathbf{v}\mathbf{W}_i^V)$$

Where to use attention

1. Encoder: self-attention.
   - Queries, keys and values come from the previous encoder layer.
   - The encoder can attend on all positions in the previous layer.
2. Decoder: self-attention.
   - Queries, keys and values come from the previous decoder layer.
   - Masked attention to prevent *looking* into the future.
3. Encoder–decoder architecture:
   - Queries come from the previous decoder layer.
   - Keys and values come from the output of the encoder.

Additional layers

- Position-wise feed-forward networks. Two linear transformations + ReLU:

$$\mathrm{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

  - This can be seen as two convolution with kernel size 1.

- Layer-normalization.
  - Normalize the outputs for diminishing internal covariate shift.

- Residual connections.
  - Improve gradient flow by adding the inputs to the output of a layer.

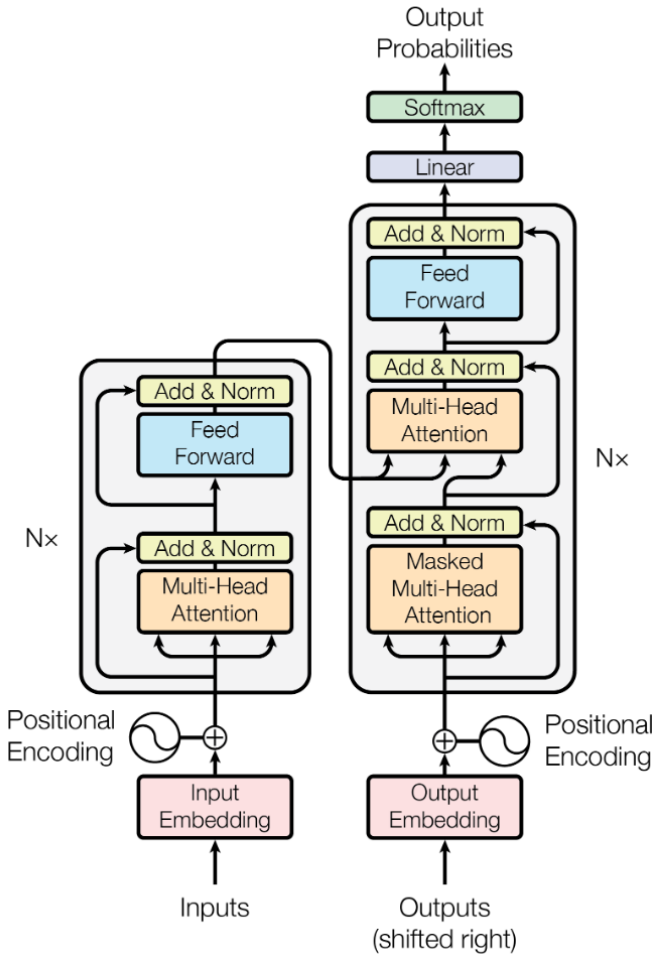
Additional layers

- Tied embeddings: Same input/output embedding weights.
  - Embeddings are scaled by a given factor ($\sqrt{d_{\text{model}}}$).
- Use the order of the sequences: positional encodings:
  - Add positional encodings (PE) to the input embeddings.
  - Let pos be the position of an embedding and i the dimension.

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

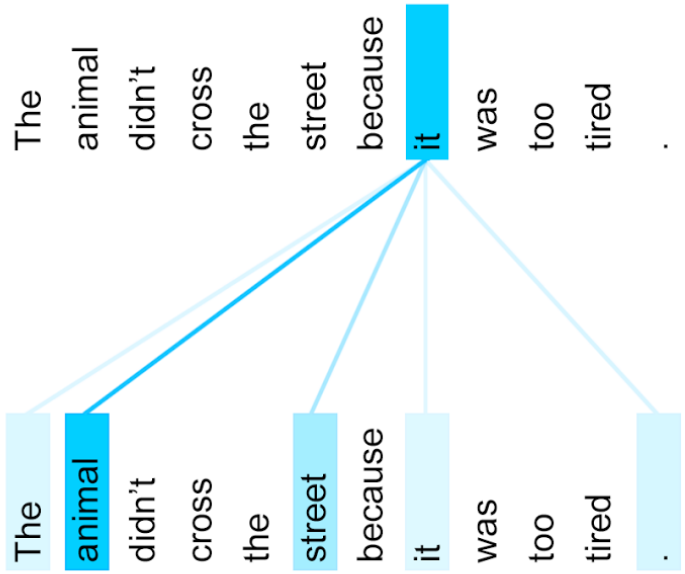
$$PE_{(\text{pos},2i+1)} = \cos(\text{pos}/10000^{2i/d_{\text{model}}})$$

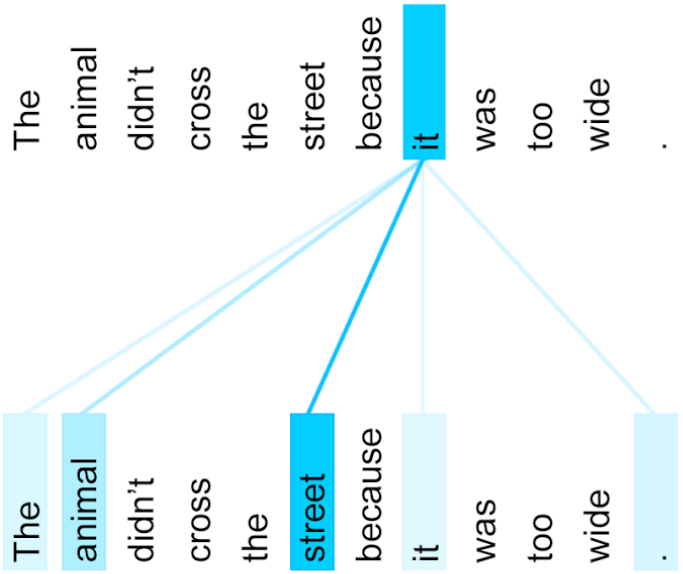
The general picture of the Transformer



Why does this work?

- Multiple levels of parallel representations.
- Self-attention directly models relationships between all words in a sentence.
  - More disambiguation, anaphora and coreference resolution capability.





Training and *haute cuisine*

- Two models (Vaswani *et al.* , 2017):
  - **Base**: $N: 6,\ d_{\text{model}} = 512,\ \ d_{\text{FF}} = 2048,\ h = 8,\ \ \ d_k = 64,\ d_v = 64,\ P_{\text{drop}} = 0.1.$
  - **Big**: $\ \ N: 6,\ d_{\text{model}} = 1024,\ d_{\text{FF}} = 4096,\ h = 16,\ d_k = 64,\ d_v = 64,\ P_{\text{drop}} = 0.3.$

- Regularization
  - Dropout to the output of each sub-layer (before being added to the input and normalized).
  - Dropout to the sums of embeddings and positional encodings.
  - Label smoothing (0.1).

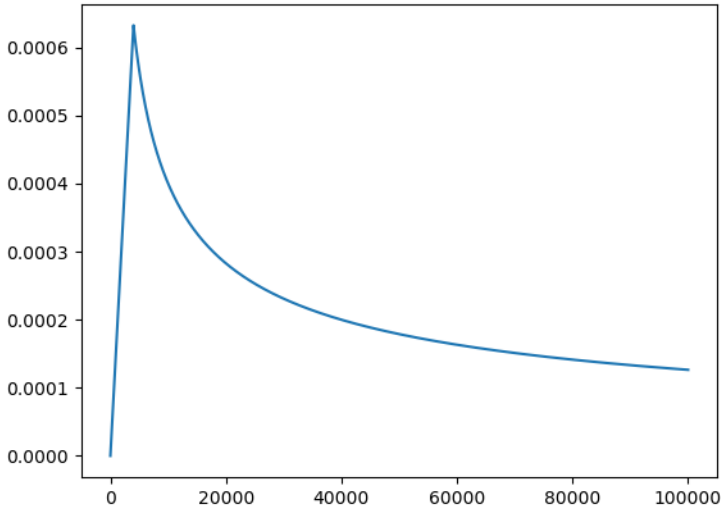
Training and *haute cuisine*

- Optimizer: Adam. $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-9}$

- Learning rate schedule: "Noam".

$$\rho = d_{\text{model}}^{-0.5} \cdot \min(\text{step\_num}^{-0.5}, \text{step\_num} \cdot \text{warmup\_steps}^{-1.5})$$

  - $\text{warmup\_steps} = 4000$.
  - $512^{-0.5} \approx 0.044$.
  - $1024^{-0.5} \approx 0.031$.



Training and *haute cuisine*

- Byte pair encoding or word-piece vocabulary.
- Model averaging with the last 5 checkpoints (computed each 10 minutes).
- Beam size: 4. Length penalty: 0.6.
- Trained on a single machine with 8 GPUs.
  - Base model: $100,000$ steps or 12 hours.
  - Big model: $300,000$ steps.

Experiments and results

- English – French WMT'14 (36M sentences).
- English – German WMT'14 (4.5M sentences).

Model	En → De	En → Fr
GNMT + RL	24.6	39.9
Convolutional Seq2Seq	26.4	41.3
Transformer base	27.3	38.1
Transformer big	**28.4**	**41.8**

The best of both worlds: RNMT+

- Combine strengths from RNNs and Transformer Chen *et al.* (2018).

RNNs

- Very expressive and powerful.
- No Markovian assumptions for sequential data.

RNNs

- Difficult to train.

Transformer:

- No sequentiality → Parallelization.
- Self attention prevents limited contexts.

Transformer:

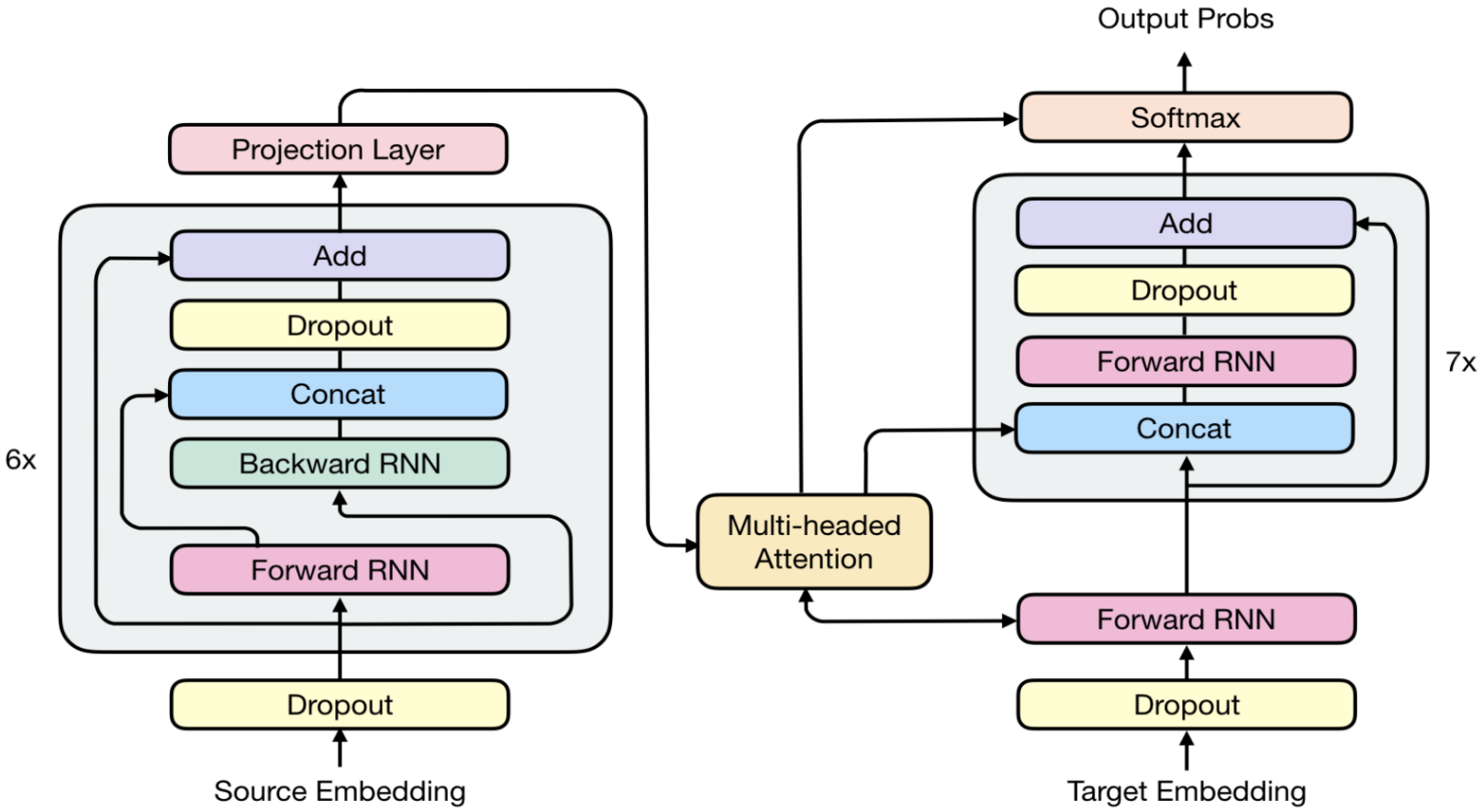
- It is not a sequential model.
- Requires from positional information.

The best of both worlds: RNMT+

- Deep (bidirectional) LSTMs.
- Residual connections, layer normalization and dropout.
- Multi-headed attention.

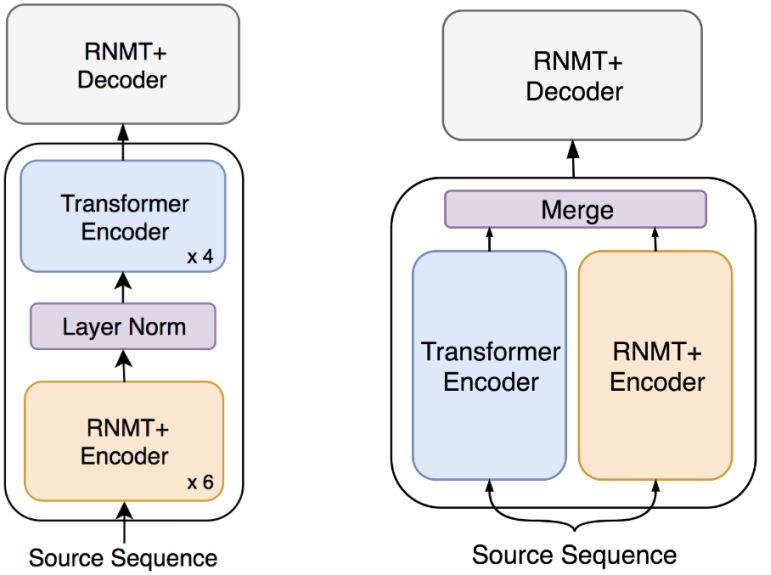
☹ No architectural parallelism → Data parallelism.

The general picture of RNMT+



Hybridation

- Mix Transformer RNMT+ encoder / decoders.
- Get richer representations by mixing encoders.



(a) Cascaded Encoder (b) Multi-Column Encoder

Training and *haute cuisine*

- Regularization
  - Dropout to the output of each LSTM.
  - Dropout to the embeddings and attention.
  - Label smoothing (0.1).
  - Weight decay with *small* corpora $(4.5M$ sentences).
- Optimizer: Adam. $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$
  - Learning rate schedule: Noam with plateau.
- Adaptive gradient clipping.

Experiments and results

- Same data as in previous section.
- Slight differences due to post-processing/model averaging stages.

Model	En → De	En → Fr
GNMT + RL	24.7	39.0
Convolutional Seq2Seq	25.0	39.5
Transformer base	27.3	39.4
Transformer big	27.9	40.7
RNMT+	**28.5**	**41.0**

Experiments and results: Combining encoders/decoders

Encoder	Decoder	En → Fr	En → De
Trans. big	Trans. big	40.7	27.9
RNMT+	RNMT+	41.0	28.5
Trans. big	RNMT+	41.1	–
RNMT+	Trans. big	39.9	–
Cascaded	RNMT+	**41.7**	28.6
MultiCol	RNMT+	**41.7**	**28.8**

Experiments and results: Model sizes

Model	Examples/s	# parameters
Convolutional Seq2Seq	80	263.4M
Transformer base	160	93.3M
Transformer big	50	375.4M
RNMT+	30	378.9M

Conclusions

- Transformer: faster models at the expense of expressive capacity.
  - Learning multiple levels of representations.
  - Self-attention is fundamental.
- RNMT+: Very expressive model.
  - Probably the most sound approach.
  - To explore the hybrid architectures seems to be the following steps to take.
- These models tackle better anaphora, coreference resolution, etc.
  - They seem mandatory for contextual NMT.
  - ...and probably also perform well on multimodal NMT, captioning, etc.

References

Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio.
Neural Machine Translation by Jointly Learning to Align and Translate.
In *ICLR*. 2015.

Minh-Thang Luong, Ilya Sutskever, Hieu Pham and Christopher D Manning.
Effective Approaches to Attention-based Neural Machine Translation.
In *Proceedings of EMNLP, pages 1412–1321*. 2015

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin
Attention Is All You Need.
In *Proceedings of NIPS. Volume 30. Pages 5998–6008*. 2017.

M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar, M. Schuster, Z. Chen, Y. Wu, M. Hughes
The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation.
In *arXiv:1804.09849*. 2018.