Machine Translation

08: More on Neural Architectures

Rico Sennrich

18 April 2023

Department of Computational Linguistics
University of Zurich

Today's Lecture

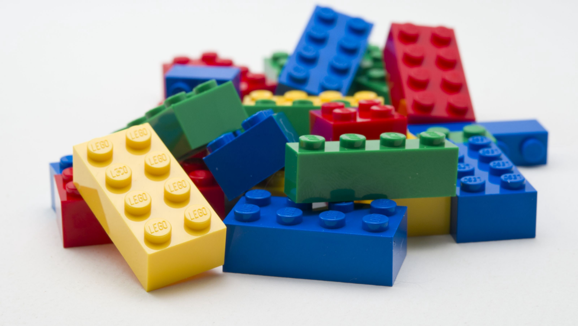
so far

- we discussed RNNs as encoder and decoder
- we discussed some architecture variants:
  - RNN vs. GRU vs. LSTM
  - attention mechanisms

today

- some important components of neural MT architectures:
  - dropout
  - layer normalization
  - deep networks
- non-recurrent architectures:
  - self-attentional networks

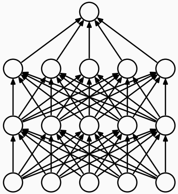
Why Architectures Matter



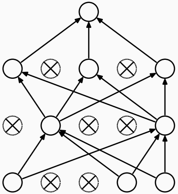
- architectures can bias learning (inductive bias):
  standard RNN more easily learns to consider recent symbols (recency bias)
- architectures interact with training algorithm:
  deep models are powerful, but face vanishing/exploding gradient problem
- architectures affect efficiency:
  - RNN introduces dependency between timesteps $\rightarrow$ no parallelisation
  - attention compares each source and target state: $O(m \times n)$ complexity

General Architecture Variants

Dropout



(a) Standard Neural Net

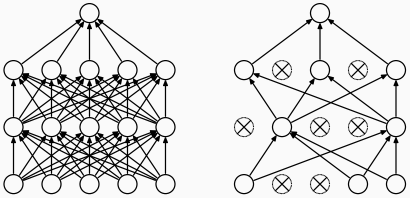


(b) After applying dropout.

- wacky idea: randomly set hidden states to 0 during training
- motivation: prevent ”co-adaptation” of hidden units
  $\rightarrow$ better generalization, less overfitting

[Srivastava et al., 2014]

Dropout



(a) Standard Neural Net (b) After applying dropout.

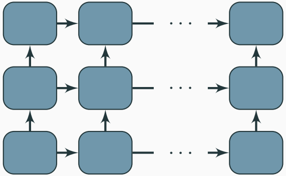
implementation:

- for training, multiply layer with "dropout mask"
- randomly sample new mask for each layer and training example
- hyperparameter p: probability that state is retained
  (some tools use p as probability that state is dropped)
- at test time, don't apply dropout,
  but re-scale layer with p to ensure expected output is the same
- (you can also re-scale by $\frac{1}{p}$ at training time instead)

Deep Networks

- increasing model depth often increases model performance
- example: stack RNN:

$$h_{i,1} = g(U_1 h_{i-1,1} + W_1 x_i)$$
$$h_{i,2} = g(U_2 h_{i-1,2} + W_2 h_{i,1})$$
$$h_{i,3} = g(U_3 h_{i-1,3} + W_3 h_{i,2})$$

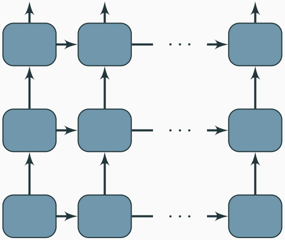


Deep Networks

often necessary to combat vanishing gradient:

residual connections between layers:

$$h_{i,1} = g(U_1 h_{i-1,1} + W_1 x_i)$$
$$h_{i,2} = g(U_2 h_{i-1,2} + W_2 h_{i,1}) + \mathbf{h_{i,1}}$$
$$h_{i,3} = g(U_3 h_{i-1,3} + W_3 h_{i,2}) + \mathbf{h_{i,2}}$$

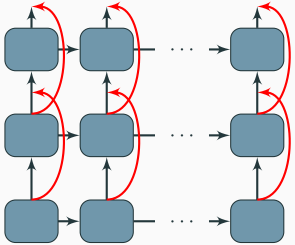


Deep Networks

often necessary to combat vanishing gradient:

residual connections between layers:

$$h_{i,1} = g(U_1 h_{i-1,1} + W_1 x_i)$$
$$h_{i,2} = g(U_2 h_{i-1,2} + W_2 h_{i,1}) + \mathbf{h_{i,1}}$$
$$h_{i,3} = g(U_3 h_{i-1,3} + W_3 h_{i,2}) + \mathbf{h_{i,2}}$$



Normalization

- family of related normalization methods:
  batch normalization, weight normalization, layer normalization
- empirically useful; still open debate why [Santurkar et al., 2018]
- common idea: normalize layer activations or network parameters

Layer Normalization

- re-center and re-scale each layer $\mathbf{a}$ (with H units)
- two bias parameters, $\mathbf{g}$ and $\mathbf{b}$, restore original representation power

$$\mu = \frac{1}{H}\sum_{i=1}^{H} a_i$$

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

$$\mathbf{h} = \left[\frac{\mathbf{g}}{\sigma} \odot (\mathbf{a} - \mu) + \mathbf{b}\right]$$

Layer Normalization and Deep Models: Results from UEDIN@WMT17

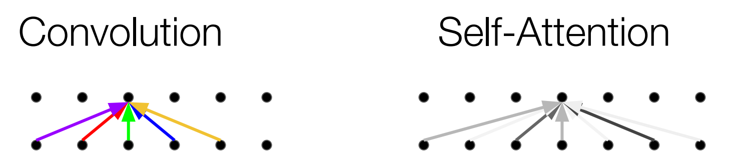
system	CS→EN 2017	DE→EN 2017	LV→EN 2017	RU→EN 2017	TR→EN 2017	ZH→EN 2017
baseline	27.5	32.0	16.4	31.3	19.7	21.7
+layer normalization	28.2	32.1	17.0	32.3	18.8	22.5
+deep model	28.9	33.5	16.6	32.7	20.6	22.9

- layer normalization and deep models generally improve quality
- layer normalization also speeds up convergence when training (fewer updates needed)
- dropout used for low-resource system (TR→EN)

NMT with Self-Attention

Attention Is All You Need [Vaswani et al., 2017]

- criticism of recurrent architecture:
  recurrent computations cannot be parallelized
- different ways to parallelize this operation:
  - use fixed-width convolutional filters over previous layer
  - calculate representation as weighted average of previous layer tokens
    weights computed by attention mechanism: **self-attention**
- there are different flavours of self-attention
  here: attend over previous layer of deep network



https://nlp.stanford.edu/seminar/details/lkaiser.pdf

Attention Is All You Need [Vaswani et al., 2017]

Transformer architecture

- stack of N blocks
  - self-attention, followed by feedforward layer
  - self-attention in decoder is *masked*
    $\rightarrow$ **causal** model
  - decoder also attends to encoder states
  - *Add & Norm*: residual connection and layer normalization
- Positional information is provided in input
  (attention is content-based, and model has no other way to learn about absolute/relative order of tokens)

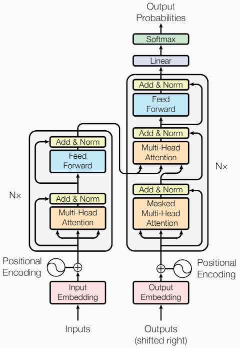


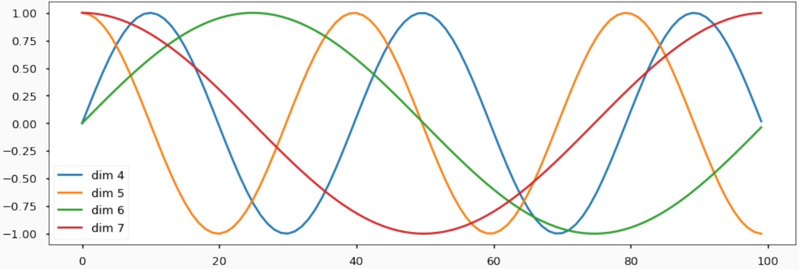
Figure 1: The Transformer - model architecture.

Positional Encoding

options:

- learned embedding for each position
- **sinusoid encoding: different periods for each element**

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



Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \underset{\text{seq}}{\text{softmax}}\left(\frac{Q \underset{\text{key}}{\odot} K}{\sqrt{d_k}}\right) \underset{\text{seq}}{\odot} V$$

- scaling by vector size ($\sqrt{d_k}$) empirically more stable
- each mechanism has:
  - $Q \in \mathbb{R}^{\text{seq} \times \text{key}}$: queries for which we want to compute attention
  - $K \in \mathbb{R}^{\text{seq} \times \text{key}}$: keys that flow into computation of attention weights
  - $V \in \mathbb{R}^{\text{seq} \times \text{val}}$: values that flow into computation of weighted average

Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \underset{\text{seq}}{\operatorname{softmax}}\left(\frac{Q \underset{\text{key}}{\odot} K}{\sqrt{d_k}}\right) \underset{\text{seq}}{\odot} V$$

- encoder-decoder attention:
  - Q: decoder states of previous (sub)layer
  - $K = V$: encoder states of **final** layer
- encoder self-attention:
  - Q: encoder states of previous layer
  - $K = V$: encoder states of previous layer
- decoder self-attention
  - Q: decoder states of previous layer
  - $K = V$: **masked** decoder states of previous layer

Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \underset{\text{seq}}{\operatorname{softmax}} \left(\frac{Q \underset{\text{key}}{\odot} K}{\sqrt{d_k}} \right) \underset{\text{seq}}{\odot} V$$

note:

- except for scaling, this is same mechanism as that by [Luong et al., 2015]: $h_t^\top$ is query, $\bar{h}_s$ is key and value

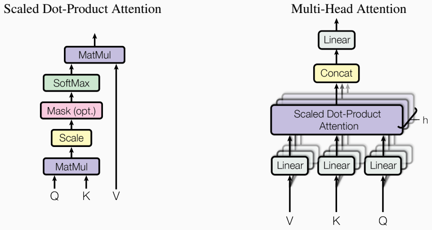
$$\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_{\boldsymbol{a}} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_{\boldsymbol{a}}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

- notation by [Vaswani et al., 2017] emphasizes:
  - attention can be computed for all queries in parallel
  - query, key and value can be different for different attention mechanisms

Multi-Head Attention

multi-head attention: use h parallel attention mechanisms (heads):

- each attention mechanism has a different input, obtained via linear projection of Q, K, and V
- outputs of each attention mechanism are concatenated
- to control size of model, linear projection produces small vectors ($\frac{d}{h}$)

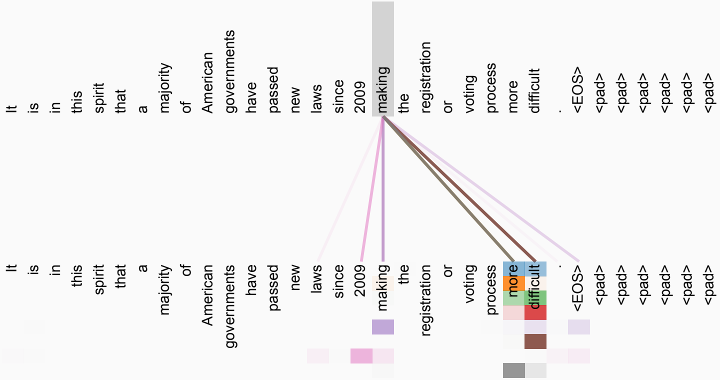


[Vaswani et al., 2017]

Multi-Head Attention

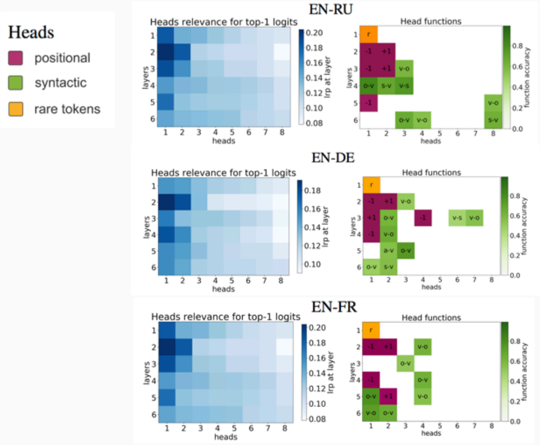
motivation for multi-head attention:

different heads can attend to different states



Analysis of Multi-Head Self-Attention

analysis by [Voita et al., 2019]:
important heads tend to be positional, syntactic, or focus on rare tokens.

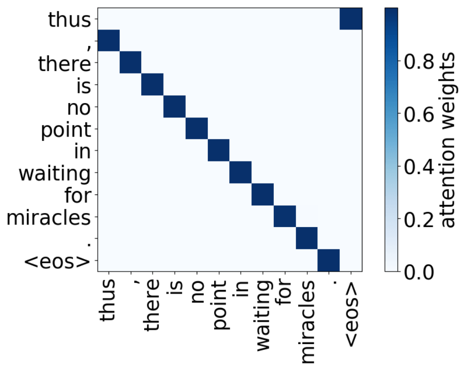


Analysis of Multi-Head Self-Attention

analysis by [Voita et al., 2019]:

important heads tend to be positional, syntactic, or focus on rare tokens.

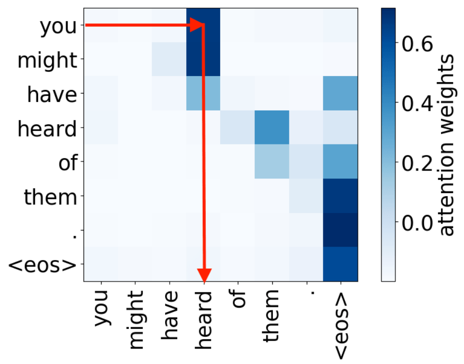
positional



Analysis of Multi-Head Self-Attention

analysis by [Voita et al., 2019]:
important heads tend to be positional, syntactic, or focus on rare tokens.

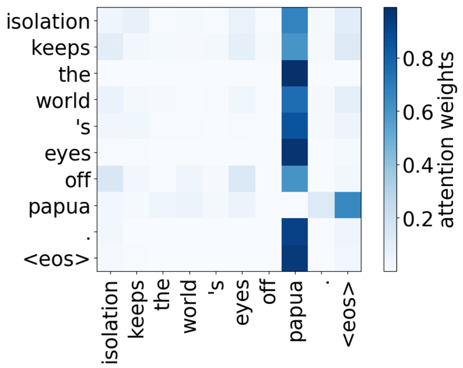
syntactic



Analysis of Multi-Head Self-Attention

analysis by [Voita et al., 2019]:
important heads tend to be positional, syntactic, or focus on rare tokens.

rare tokens



Feed-Forward Sublayer

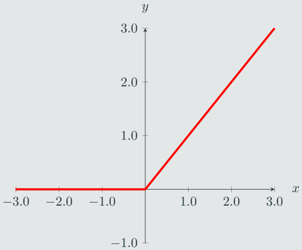
actually two feed-forward neural networs, with ReLU in between:

$$\mathsf{FFN}(x) = \max(0, x \underset{d_{\mathsf{model}}}{\odot} W_1 + b_1) \underset{d_{\mathsf{ff}}}{\odot} W_2 + b_2$$

(with $W_1 \in \mathbb{R}^{d_{\mathsf{model}} \times d_{\mathsf{ff}}}; W_2 \in \mathbb{R}^{d_{\mathsf{ff}} \times d_{\mathsf{model}}}; b_1 \in \mathbb{R}^{d_{\mathsf{ff}}}; b_2 \in \mathbb{R}^{d_{\mathsf{model}}}$)
typically, $d_{\mathsf{ff}} > d_{\mathsf{model}}$ (2048 and 512 for base Transformer, respectively)

refresher: rectified linear unit (ReLU)

ReLU: $y = max(0, x)$



Review: Global Architecture

pseudo-code:

```
h_enc = E(X) + PE(X)

for i in num_layers:
    h_enc = LN(MHAtt(h_enc, h_enc, h_enc) + h_enc)
    h_enc = LN(FF(h_enc) + h_enc)

h_dec = E(Y) + PE(Y)

for i in num_layers:
    h_dec = LN(MHatt(mask(h_dec), mask(h_dec), h_dec) + h_dec)
    h_dec = LN(MHatt(h_enc, h_enc, h_dec) + h_dec)
    h_dec = LN(FF(h_dec) + h_dec)

p_y = softmax(FF_lin(h_dec))
```

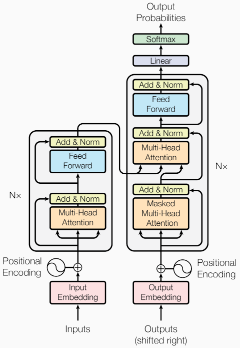


Figure 1: The Transformer - model architecture.

Variants

- original Transformer: **post-norm**

  `h_enc = LN(FF(h_enc) + h_enc)`

- popular variant: **pre-norm**

  `h_enc = FF(LN(h_enc)) + h_enc`

- layer normalisation in post-norm may cause gradient to vanish over many layers
- pre-norm has "pure' residual connection

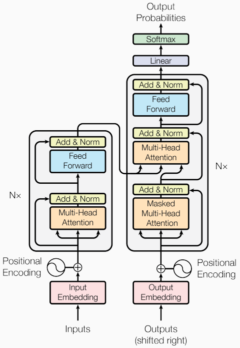


Figure 1: The Transformer - model architecture.

Typical Hyperparameters

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Comparison Between RNN and Transformer

empirical comparison difficult

- some components could be mix-and-matched
  - choice of attention mechanism
  - choice of positional encoding
  - hyperparameters and training tricks
- different test sets and/or evaluation scripts

Comparison Between RNN and Transformer

tokenized EN-DE; newstest2014

architecture	BLEU
RNN [Wu et al., 2016]	24.6
self-attention [Vaswani et al., 2017]	28.4
RNN [Chen et al., 2018]	28.5

Summary

- important "building blocks":
  - dropout
  - residual connections and deep networks
  - layer normalization
  - multihead attention and self-attention
  - positional encodings

  → you will encounter them in various arrangements
- Transformer is currently most popular architecture in MT and beyond
- search for better architectures is ongoing, but hard work

Further Reading

- Main Reading:
  Koehn (2020), chapter 11
- Recommended Reading:
  Vaswani et al. (2017). Attention Is All You Need.
  https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf
- Further Reading:
  - The Annotated Transformer:
    https://nlp.seas.harvard.edu/2018/04/03/attention.html
  - consider original literature cited on relevant slides

Chen, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., Jones, L., Schuster, M., Shazeer, N., Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Chen, Z., Wu, Y., and Hughes, M. (2018).
The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation.
In
Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1:
pages 76–86, Melbourne, Australia.

Luong, T., Pham, H., and Manning, C. D. (2015).
Effective Approaches to Attention-based Neural Machine Translation.
In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing,
pages 1412–1421, Lisbon, Portugal.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018).
How does batch normalization help optimization?
In Advances in Neural Information Processing Systems 31, pages 2488–2498. Curran Associates, Inc.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).
Dropout: A Simple Way to Prevent Neural Networks from Overfitting.
Journal of Machine Learning Research, 15:1929–1958.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017).
Attention is All you Need.
In Advances in Neural Information Processing Systems 30, pages 5998–6008.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. (2019).
Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned.
In Proceedings of the 57th Conference of the Association for Computational Linguistics, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016).
Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.
ArXiv e-prints.