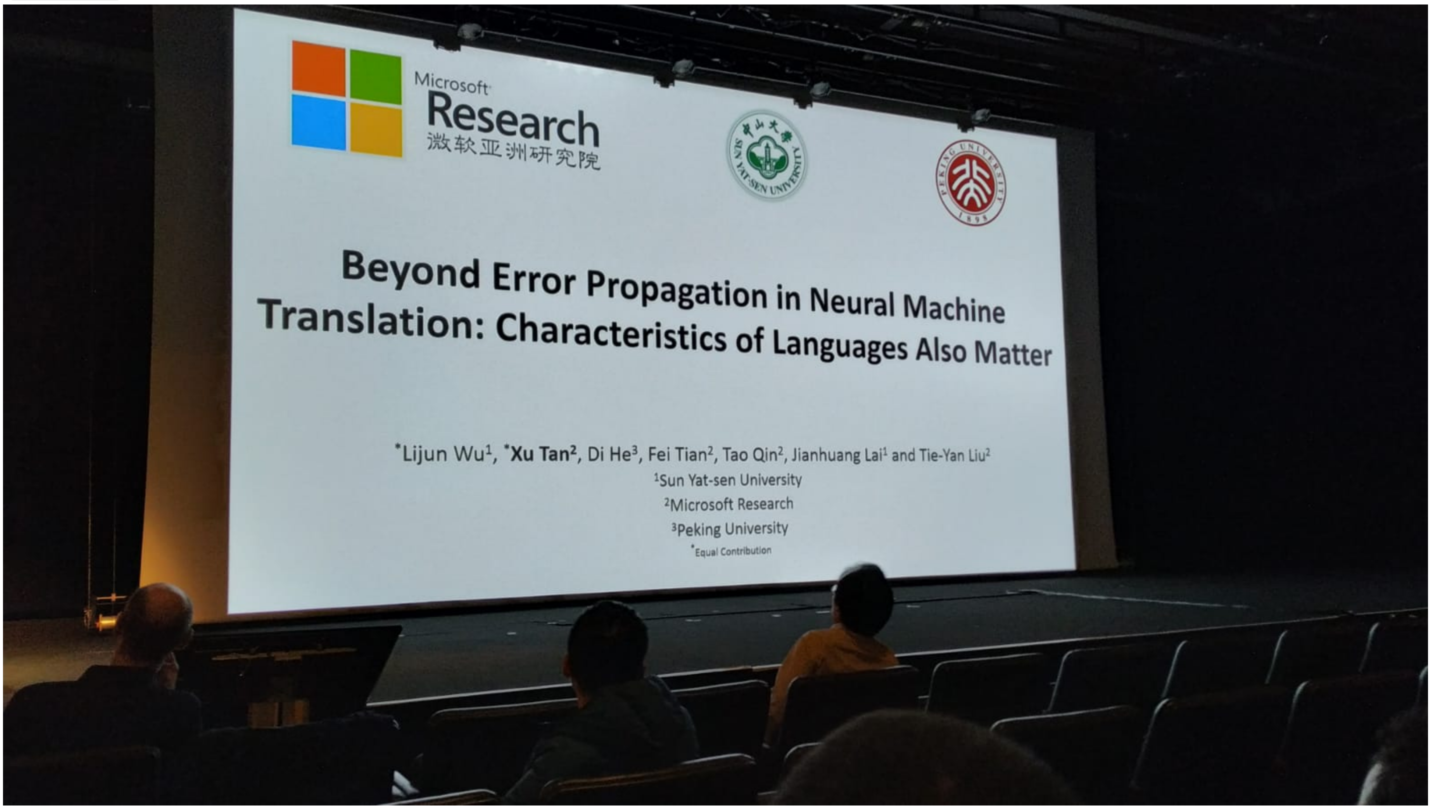
2019/2/1

`08:49` First day of the conference

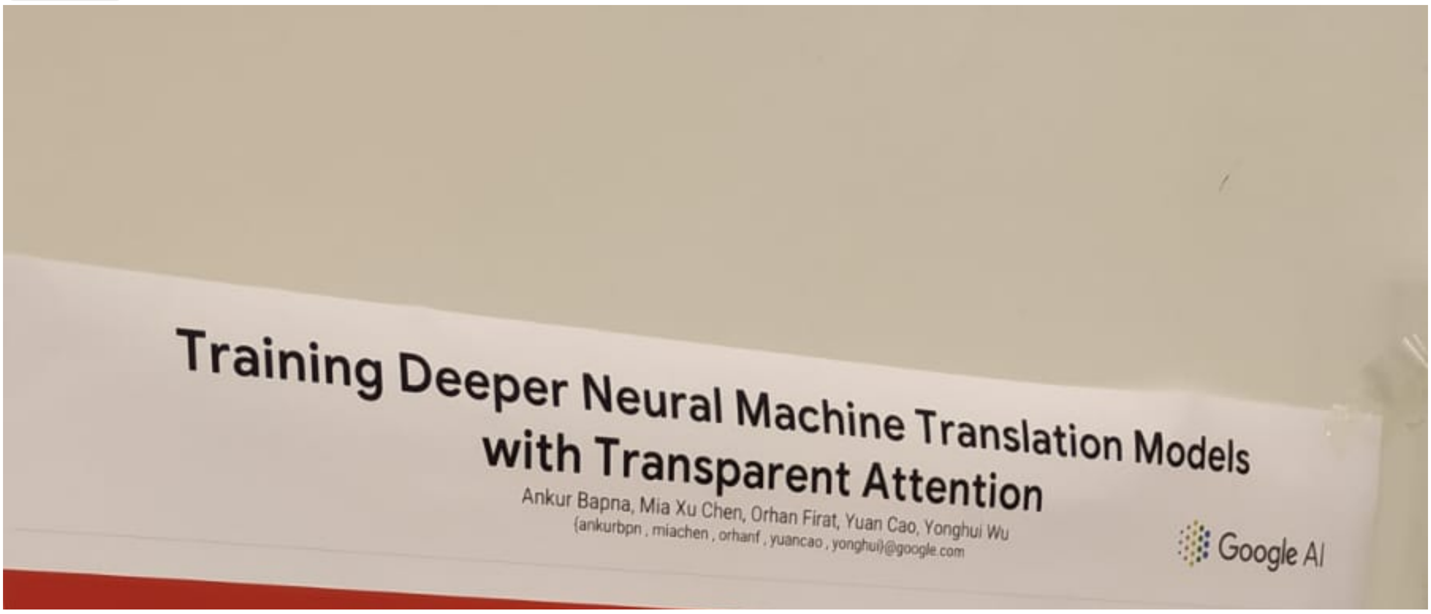
`09:10`



`09:17` During this talk, I had the idea to blah blah...

`11:00` Poster session

`11:04`



Highlights

- **Deep seq2seq encoders are hard to train**
  - Hindered gradient flow to lower layers
  - Models fail to converge
- We propose a simple modification to the attention mechanism
  - Attend weighted combination of encoder layers
  - Instead of just the final encoder layer output
- Enables us to train encoders with **upto 20 layers**
  - Significant gains over shallow (6 layer) baselines → on WMT En-De and Cs-En
  - Illustrate bottom-up training of encoder → lower layers converge first

Analyze Encoder Convergence

Train standard models with deeper encoders

- Transformer [1] → fail to optimize
- RNMT+ [2] → converge, but fail to improve quality

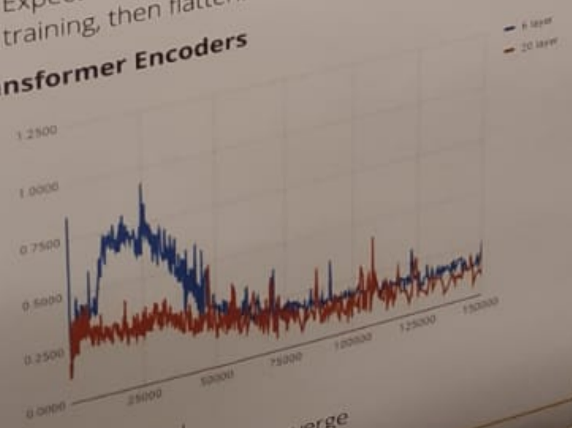
Why deeper models fail?

- Indicators of a healthy training [3]
  - → Lower layers converge quickly
  - → Topmost layers take longer
- Track gradient-norm ratio (1st and the final layer)

$$r_t = \left(\|\nabla_{h_1} L^{(t)}\| / \|\nabla_{h_N} L^{(t)}\|\right)$$

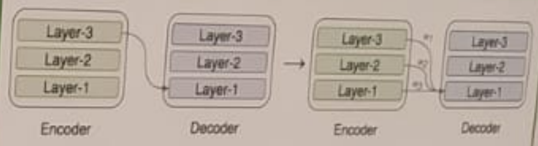
- Expect large grad-norm ratio at the early stages of the training, then flatten.

Transformer Encoders



- 20 layer model
  - → Lower layers fail to converge

One Simple Trick



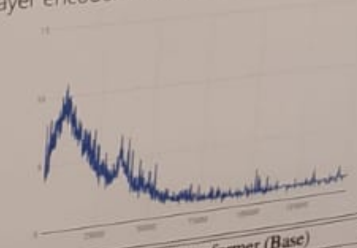
n: number of encoder layers

m: number of decoder layers

1. Define **m*(n+1)** matrix of weights, **W**
2. **W** will be learnt via gradient descent
3. Apply softmax to normalize **W**
4. Use normalized **W** to obtain **m** weighted sums of encoder layer outputs

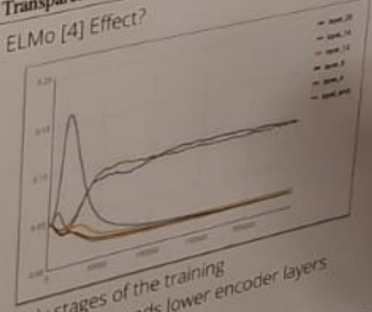
Results

Effect on Transformer encoder gradient flow (20 layer encoder with transparent attention)



En→De WMT 14	Transformer (Base)				(Big)
Encoder layers	6	12	16	20	6
Num. Parameters	94M	120M	137M	154M	375M
Baseline	27.26	*	*	*	27.94
Baseline - residuals	*	6.00	*	*	N/A
Transparent	27.52	27.79	**28.04**	27.96	N/A

The ELMo [4] Effect?



- Early stages of the training
  - → Decoder attends lower encoder layers
- Near convergence
  - → All weight on top few layers

[1] Vaswani et al. 2017 "Attention is All You Need"
[2] Chen et al. 2018 "The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation"
[3] Raghu et al. 2017 "SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability"
[4] Peters et al. 2018 "Deep contextualized word representations"

11:10

Pervasive Attention

2D Convolutional Neural Networks for Sequence-to-Sequence Prediction

Maha Elbayad†‡ Laurent Besacier‡ Jakob Verbeek†

†firstname.lastname@inria.fr

‡firstname.lastname@univ-grenoble-alpes.fr

Code available at: github.com/**elbayadm/attn2d**

Overview

n state-of-the-art encoder-decoder models, the source and target sequences are rocessed **separately**. The decoder, equipped with an **attention mechanism**, ocuses on different parts of the source at each decoding step. However, the ttention is limited to assigning weights to the **once and for all** computed ncoder states.

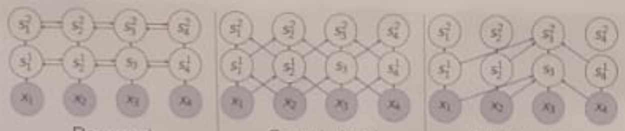
Contributions: build an architecture from the get go around attention by **ointly** encoding the source and target sequences and allowing for different source representations for every target position.

Encoder-Decoders

Encoder

Inputs: source sequence $\mathbf{x} = (x_1, x_2, \dots, x_{|\mathbf{x}|})$.

Depending on the chosen architecture, the encoder computes the source representations.



Decoder

Inputs: source codes $(s_1, \dots, s_{|\mathbf{x}|})$ and target sequence $\mathbf{y} = (y_1, y_2, \dots, y_{|\mathbf{y}|})$.

At every step t:

- Under the architecture, compute the hidden state h_t causally.
- Given the new state, the attention mechanism yields a context c_t.
- $h_t :=$ combine(h_t, c_t).

Pervasive attention: the input

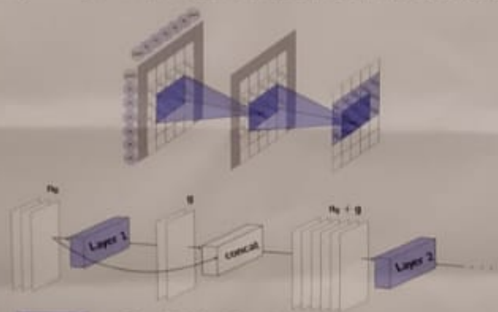
The initial 2D grid:

$\forall i, j : \{1, \dots |y|\} \times \{1, \dots |x|\}$

$$\begin{cases} u_i &= U_{y_i} \text{ (target embedding)} \\ v_j &= V_{x_j} \text{ (source embedding)} \\ h_{ij} &= concat(u_i, v_j) \end{cases}$$

Pervasive attention: the convolutional network

- **Causality:** with masked filters in the target direction.
- **Context:** grown with stacked convolutions.
- **Padding:** throughout the network to maintain source/target resolution.



Layer i = BN - ReLU - Conv(1) - BN - ReLU - Conv(k)

Pervasive attention: the aggregation

To aggregate activations across source positions e.g. with $H_3 = [h^L_{31}, \dots, h^L_{3|x|}] \in \mathbb{R}^{d \times |x|}$, we can use:

- Max/average pooling.
- Self-attention:
  $\rho = softmax(H_3^T W + b)$.
  $h_3 = H_3 \rho$.
- A combination of the above.

Experimental results

Benchmark: IWSLT'14 German↔English translation.

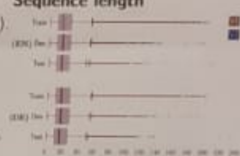
Pre-processing:

- Tokenization (Moses).
- Lower-casing.
- Length $\leq$ 175 words
- Lengths ratio $\leq$ 1.5.
- Train, Dev, Test : 160k, 7.2k, 6.7k

Sub-word segmentation

- BPE (Sennrich et al., 2016).
- 14k merge operations on EN+DE (V1) / on each separately (V2).
- V1(EN,DE) = 8.8k, 12k, V2(EN,DE) = 13.3k, 13.8k.

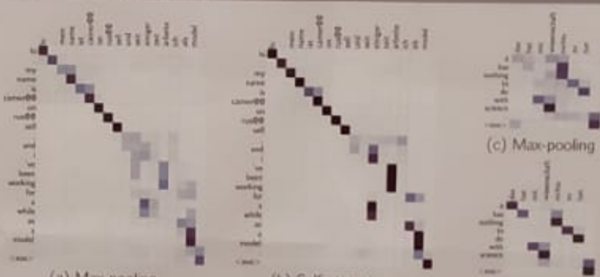
Sequence length



Comparison to the stat-of-the-art

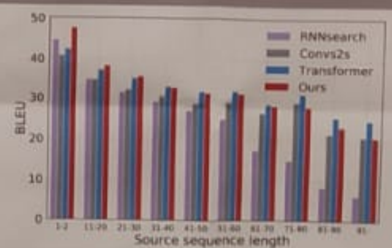
	De→En	#prms	En→De	#prms
•RNNsearch (Bahdanau et al., 2015)	29.98	13M	25.04	15M
Varational attention (Deng et al., 2018)	33.10	-	-	-
••ConvS2S (MLE) (Gehring et al., 2017)	31.59	21M	27.18	22M
ConvS2S (MLE+SLE) (Edunov et al., 2018)	32.84	-	-	-
••Transformer (Vaswani et al., 2017), V1	34.42	46M	**28.23**	48M
••Transformer (Vaswani et al., 2017), V2	**34.44**	52M	28.07	52M
Pervasive attention (ours), V1	33.86	**11M**	27.21	**11M**
Pervasive attention (ours), V2	34.05	22M	27.97	22M

Alignment visualization

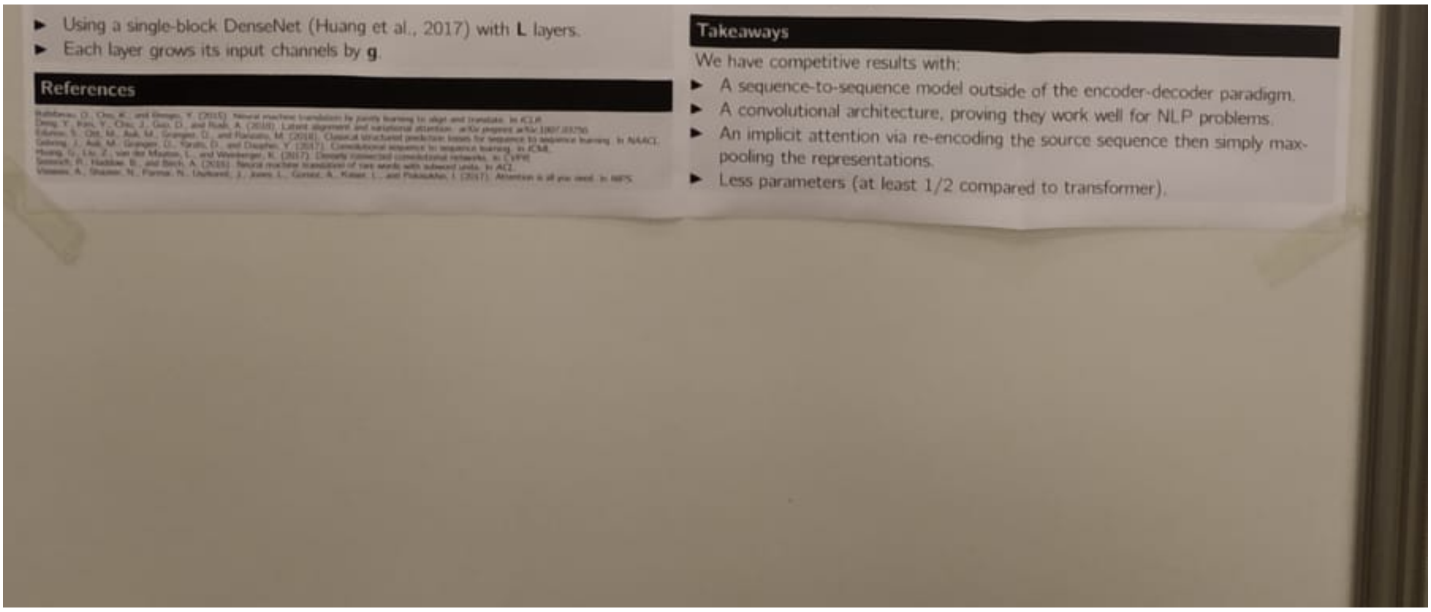


(a) Max-pooling (b) Self-attention (c) Max-pooling (d) Self-attention

BLEU per sequence length



Due to memory/compute limitations, $\mathcal{O}(|x|.|y|)$ instead of $\mathcal{O}(|y| + |x|)$, we truncate sequences longer than 80 tokens when training which affects the performance on long sequences.



2019/2/2

`09:00` Second day of the conference

`09:23`

`16:53` Fin