Convolutional Sequence to Sequence Learning

FAIR-Seq

Contents

Abstract

- We introduce an architecture based entirely on **convolutional neural networks**
- **Computations** over all elements can be fully **parallelized** during training to better exploit the GPU hardware and optimization is easier.
- Use of gated linear Units eases gradient propagation (Dauphin et al. 2016)
- Equip each decoder **layer** with a separation **attention** module

Introduction

- Convolutional layer
  - Fixed size contexts
  - But network can easily be made larger by **stacking** several layers on top of each other ➔ allows to precisely **control the maximum length of dependencies** to be modeled
  - Convolutional networks **do not depend on** the computations of the **previous time step** ➔ allow **parallelization** over every element in a sequence

Introduction

- Multi-layer convolutional neural networks
  - Create **hierarchical representations** over the input sequence
    - Nearby input elements interact at lower layers while distant elements interact at higher layers
  - Provides a **shorter path** to capture **long-range dependencies**
    - Obtain a feature representation capturing relationships within a window of n words
  - Inputs to a convolutional network are fed through a **constant number of kernels** and non-linearities, whereas RNN apply up to n operations and non-linearities

Introduction

- Summary of Intro
  - What we propose are…
    - Convolution Network Architecture
    - Gated linear units & Residual connections
    - Added Attention layer in every decoder layers
  - The combination of these choices enables us to tackle large scale problems

Introduction

- Recent studies related Convolution
  - Bradbury et al. (2016)
  - Kalchbrenner et al. (2016)
  - Meng et al. (2015)
  - Gehring et al. (2016)

RNN Seq2Seq

- RNN Seq2Seq
  - Encoder input sequence $\boldsymbol{x} = (x_1, x_2, \dots, x_m)$ returns state representation $\boldsymbol{z} = (z_1, z_2, \dots, z_m)$
  - Then Decoder generates output sequence $\boldsymbol{y} = (y_1, y_2, \dots, y_m)$ left to right, one element at a time
  - To generate y_{t+1}, the model generates like $P(y_{t+1} | y_1, y_2, \dots, y_t, \boldsymbol{z})$
- Attention Mechanism
  - Architectures compute c_i as a weighted sum of $(z_1, z_2, \dots, z_m)$ at each time step
  - The weights allow the network to focus on different parts of the input sequence as it generates the output sequences

A Convolution Architecture

- Position Embeddings
  - Position embeddings give our model **a sense of which portion of the sequence** in the input or output it is currently dealing with
    - Embed input elements $\boldsymbol{x} = (x_1, x_2, \dots, x_m)$ in distributional space as $\boldsymbol{w} = (w_1, w_2, \dots, w_m)$ where $w_j \in \mathbb{R}^f$
    - Equip our model with a sense of order by embedding **the absolute position of input** elements $\boldsymbol{p} = (p_1, p_2, \dots, p_m)$ where $p_j \in \mathbb{R}^f$
    - Combine to obtain input element representations $\boldsymbol{e} = (w_1 + p_1, \dots, w_m + p_m)$
    - Proceed similarly for output elements that were already generated by the decode network to yield output element representations

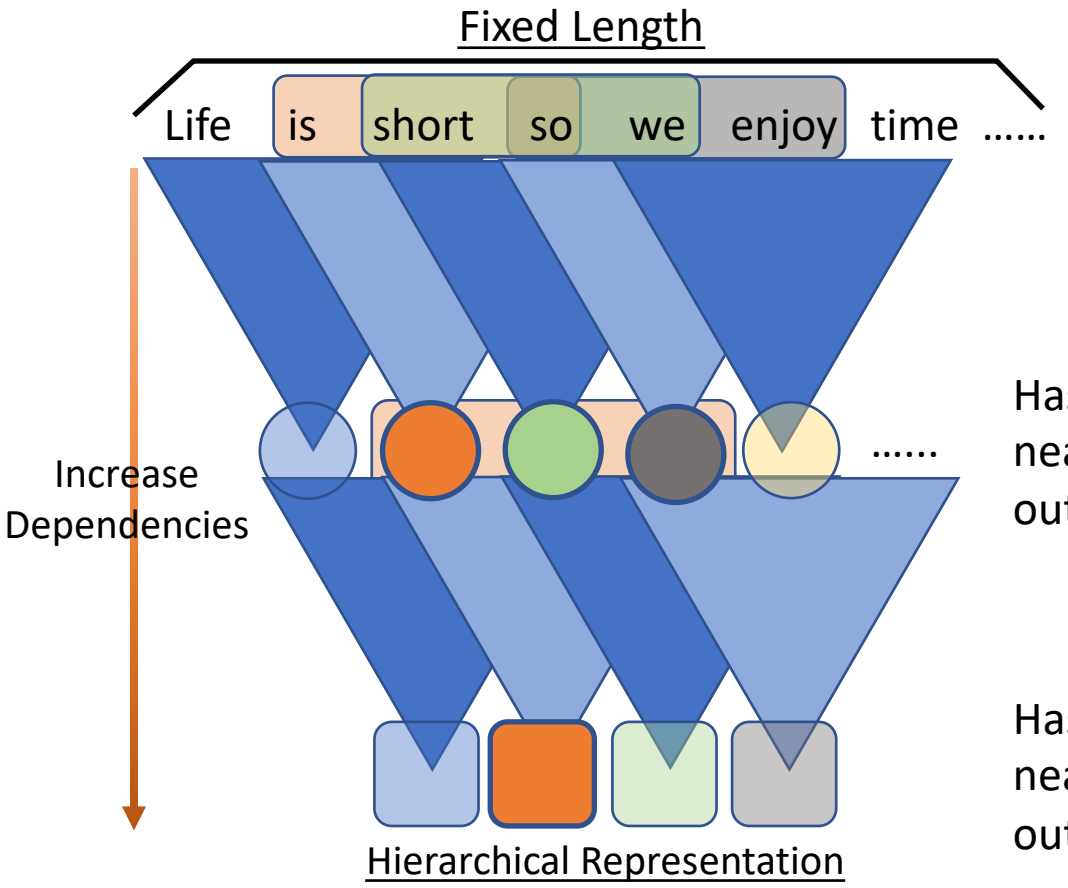
      $\boldsymbol{g} = (g_1, g_2, \dots, g_m)$

A Convolution Architecture

- Convolutional Block Structure
  - Both encoder and decoder networks share a simple block structure that computes intermediate states based on a **fixed number of input elements**
  - Each block contains a one dimensional convolution followed by a non-linearity
  - **Stacking several blocks** on top of each other **increases the number of input elements represented in a state**

A Convolution Architecture

- Convolutional Block Structure



- Stacking 2 blocks with $k = 3$ (k is kernel size)
- Each outputs depends on 5 token element inputs
- How many Conv blocks have to stack for 25 token inputs to represent all inputs in one output element?

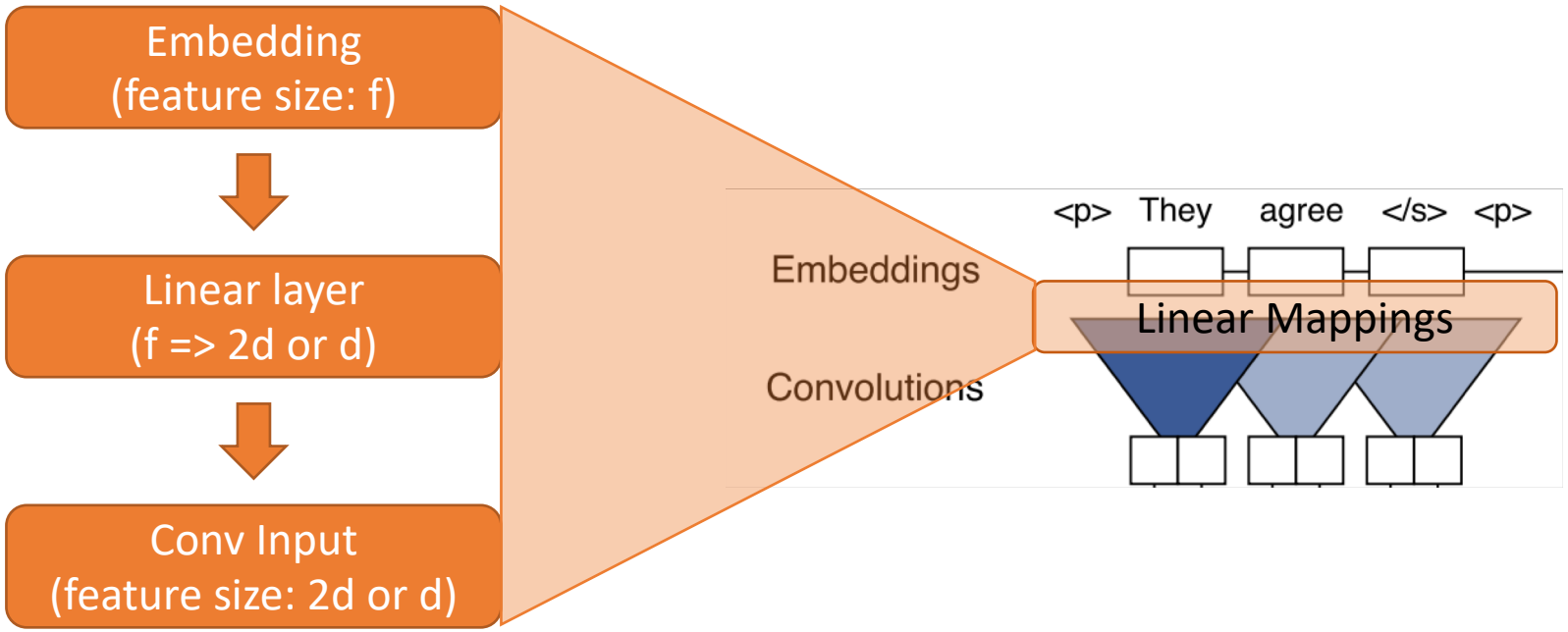
$$1 + (2 \times n) = 25 \ \therefore n = 6$$

A Convolution Architecture

- Convolutional Block Structure
  - For encoder networks we ensure that the output of the convolutional layers **matches the input length by padding** the input at each layer
  - However, for decoder networks we have to take care that **no future information is available** to the decoder (Oord et al. 2016a)

A Convolution Architecture

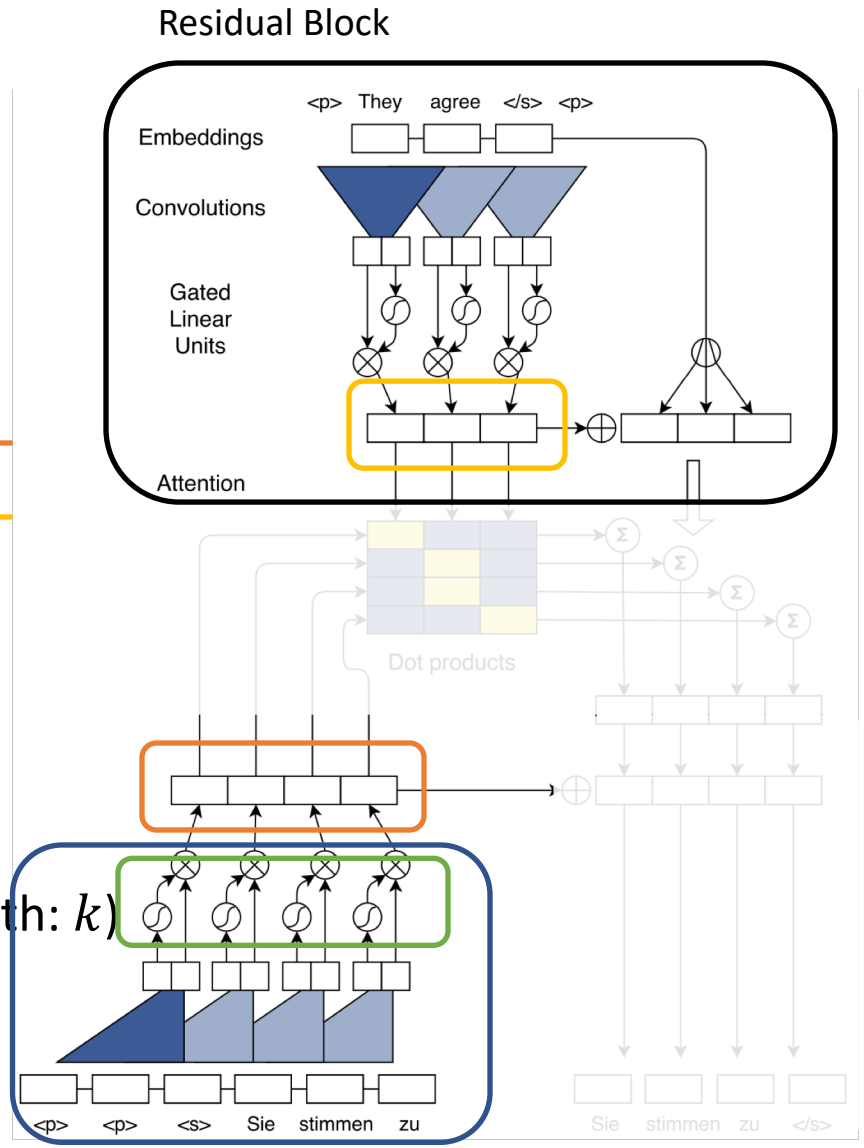
- Convolutional Block Structure
  - We also add linear mappings to project between the embedding size f and the convolution outputs that are of size $2d$



A Convolution Architecture

- Convolutional Block Structure
  - Input Source
    - l-th block of decoder output ➔ $\boldsymbol{h}^l = (h_1^l, h_2^l, \dots, h_m^l)$
    - l-th block of encoder output ➔ $\boldsymbol{z}^l = (z_1^l, z_2^l, \dots, z_m^l)$
  - Convolutional Block
    - $v([A\ B]) = A \otimes \sigma(B)$ (σ is sigmoid)
      - ➔ $[A\ B] \in \mathbb{R}^{2d}, v([A\ B]) \in \mathbb{R}^d$
    - $h_i^l = v\left(W^l \left[h_{i-\frac{k}{2}}^{l-1}, \dots, h_{i+\frac{k}{2}}^{l-1}\right] + b_w^l\right) + h_i^{l-1}$ (Kernel width: k)
      - ➔ $W \in \mathbb{R}^{2d \times kd}, b_w \in \mathbb{R}^{2d}$

Residual Connection



A Convolution Architecture

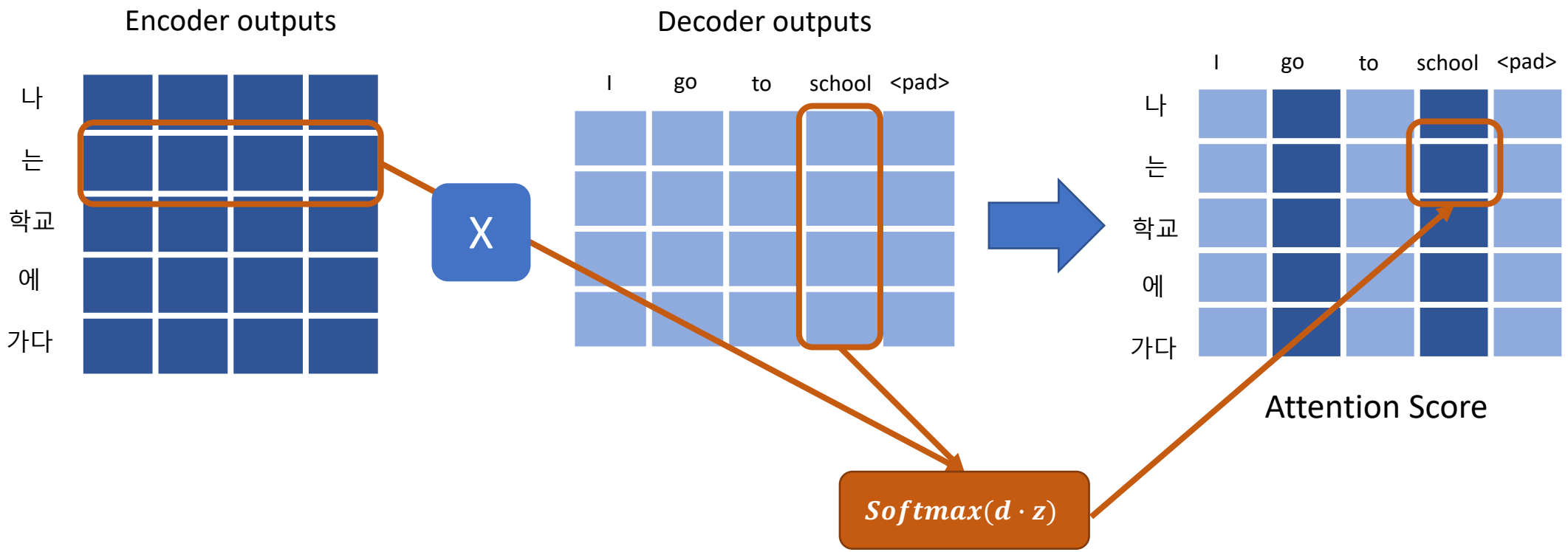
- Convolutional Block Structure
  - Finally, we compute a distribution over the T possible next target elements y_{i+1} by transforming the top decoder output h_i^L via a linear layer with weights W_o and b_o
  - $P(y_{t+1} | y_1, y_2, \ldots, y_t, \mathbf{x}) = \text{softmax}(W_o h_i^L + b_o) \in \mathbb{R}^T$

A Convolution Architecture

- Multi-step Attention
  - To compute the attention, we combine the current decoder state h_i^l with an embedding of the previous target element g_i
    - $d_i^l = W_d^l h_i^l + b_d^l + g_i$
  - For decoder layer l the attention α_{ij}^l of state i and source element j is computed as a dot-product between the decoder state summary d_i^l and each output z_j^u of the last encoder block u
    - $\alpha_{ij}^l = \frac{\exp(d_i^l \cdot z_j^u)}{\sum_{t=1}^m \exp(d_t^l \cdot z_t^u)}$

A Convolution Architecture

- Multi-step Attention

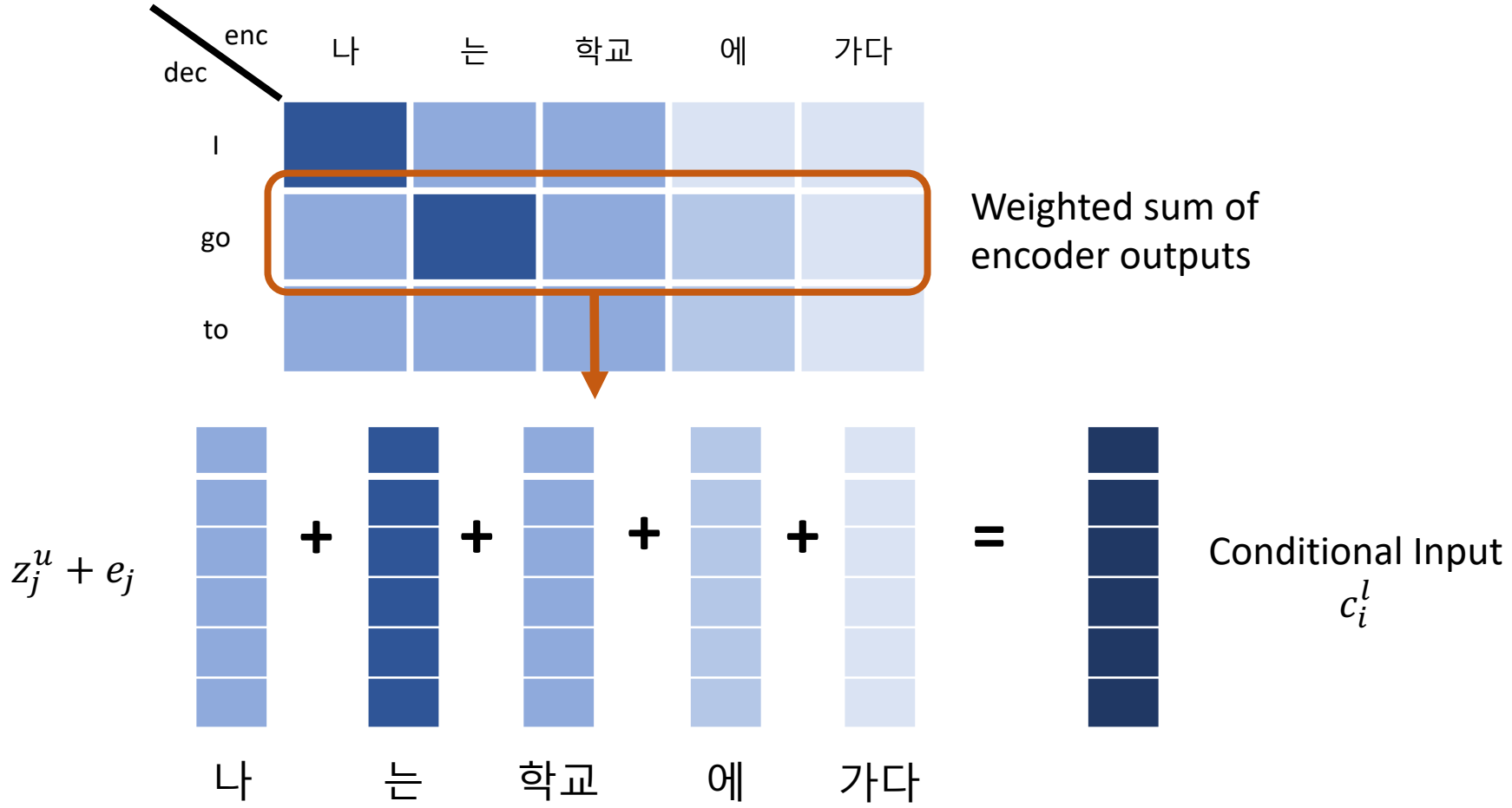


A Convolution Architecture

- Multi-step Attention
  - The conditional input c_i^l to the current decoder layer is **a weighted sum of the encoder outputs** as well as the input element embeddings e_j
    - $c_i^l = \sum_{j=1}^{m} \alpha_{ij}^l (z_j^u + e_j)$
  - It resembles key-value memory networks where the **keys are the $\boldsymbol{\alpha_{ij}^l}$** and the **values are $\boldsymbol{z_j^u + e_j}$**
  - Encoder outputs z_j^u represent potentially large input contexts and e_j provides point information about a specific input element that is useful when making a prediction e_j

A Convolution Architecture

- Multi-step Attention

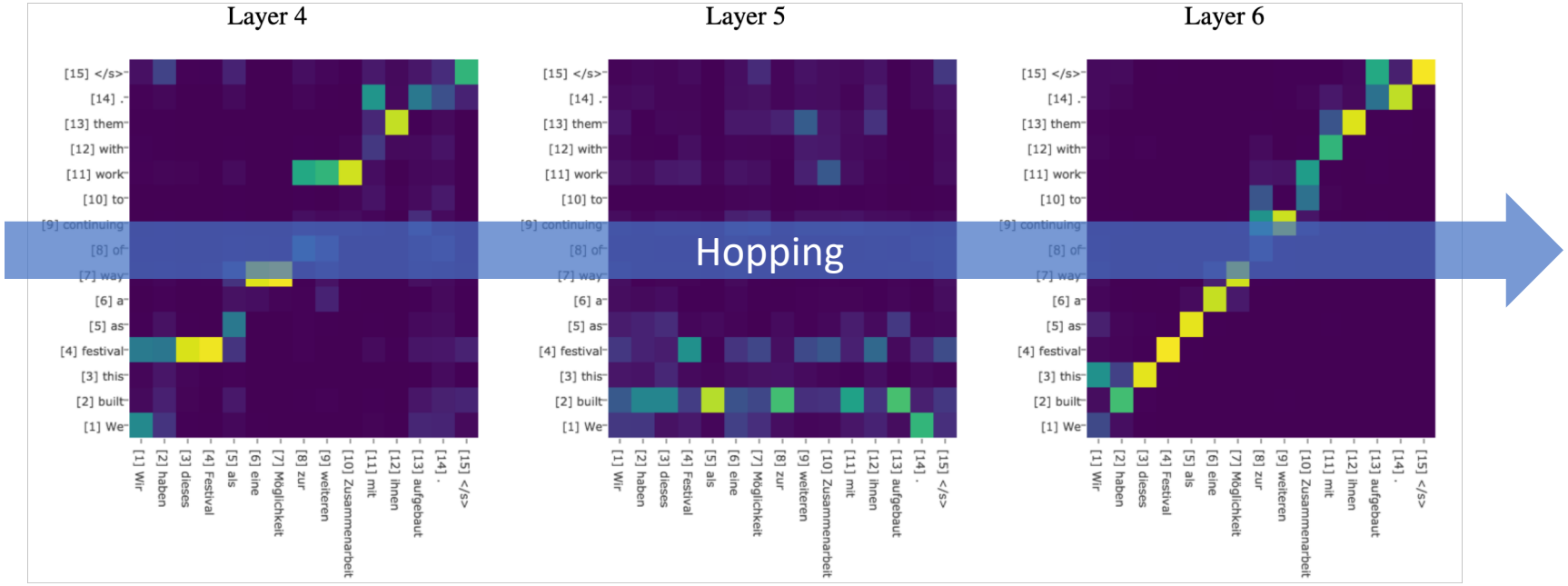


A Convolution Architecture

- Multi-step Attention
  - This can be seen as attention with multiple '**hops**' compared to single step attention
    - The attention of the first layer determines a useful source context which is then fed to the second layer that takes this information into account when computing attention.
    - The decoder also has immediate access to the attention history of the k-1 previous time steps

A Convolution Architecture

- Multi-step Attention
  - Our attention mechanism considers **which words we previously attended** to and performs multiple attention ‘hops’ per time step



A Convolution Architecture

- Normalization Strategy (work well in this practice)
  - We scale the output of residual blocks as well as the attention **to preserve the variance of activations**
    - We multiply the sum of the input and output of a residual block by $\sqrt{0.5}$ to halve the variance of the sum
  - The conditional input c_i^l generated by the attention is a weighted sum of m vectors
    - We counteract a change in variance through scaling by $m\sqrt{1/m}$; we multiply by m to scale up the inputs to their original size, assuming the attention scores are uniformly distributed

A Convolution Architecture

- Normalization Strategy (work well in this practice)
  - We scale the gradients for the encoder layers by the number of attention mechanisms we use; we exclude source word embeddings
    - ← Since the encoder received too much gradient otherwise

A Convolution Architecture

- Initialization
  - The motivation for our initialization is **maintaining the variance of activations** throughout the forward and backward passes
  - All embeddings are initialized from a normal distribution with mean 0 and standard deviation 0.1
  - For layers whose output is not directly fed to a gated linear unit, we initialize weights from $\mathcal{N}\left(0, \sqrt{\frac{1}{n_l}}\right)$ where n_l is the number of input connections to each neuron.

A Convolution Architecture

- Initialization
  - We initialize the GLU weights so that the input to the GLU activations have 4 times the variance of the layer input.

    ← If the GLU inputs are distributed with mean 0 and have sufficiently small variance, then we can approximate the output variance with ¼ of the input variance
  - This is achieved by drawing their initial values from $\mathcal{N}\left(0, \sqrt{\frac{4}{n_l}}\right)$

A Convolution Architecture

- Initialization
  - We apply dropout to the input of some layers so that inputs are retained with a probability of p
  - The application of dropout will then cause the variance to be scaled by $1/p$
  - We aim to restore the incoming variance by initializing the respective layers with larger weights
  - Specifically, we use $\mathcal{N}\left(0, \sqrt{\frac{4p}{n_l}}\right)$ for layers whose output is subject to a GLU and $\mathcal{N}\left(0, \sqrt{\frac{p}{n_l}}\right)$

Experimental Setup

- Datasets
  - Translation
    - WMT’ 16 English-Romanian
    - WMT’ 14 English-German
    - WMT’ 14 English-French
  - Abstractive summarization
    - Train – Gigaword corpus
    - Test – DUC-2004,

Experimental Setup

- Model Parameters and Optimization
  - Convolution hidden units: 512
  - # of Mini-batches: 64
  - Dropout
  - Nesterov’s accelerated gradient method
    - Momentum value: 0.09
    - Renormalize gradients if their norm exceeds 0.1
    - Learning rate: 0.25

Results

	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

Results

Attn Layers	PPL	BLEU
1,2,3,4,5	6.65	21.63
1,2,3,4	6.70	21.54
1,2,3	6.95	21.36
1,2	6.92	21.47
1,3,5	6.97	21.10
1	7.15	21.26
2	7.09	21.30
3	7.11	21.19
4	7.19	21.31
5	7.66	20.24

Kernel width	Encoder layers		
	5	9	13
3	20.61	21.17	21.63
5	20.80	21.02	21.42
7	20.81	21.30	21.09

Kernel width	Decoder layers		
	3	5	7
3	21.10	21.71	21.62
5	21.09	21.63	21.24
7	21.40	21.31	21.33

Conclusion

- We introduce the first fully convolutional model for sequence to sequence learning
- Our model relies on gating and performs multiple attention steps
- This model may benefit from learning hierarchical representations

Q&A