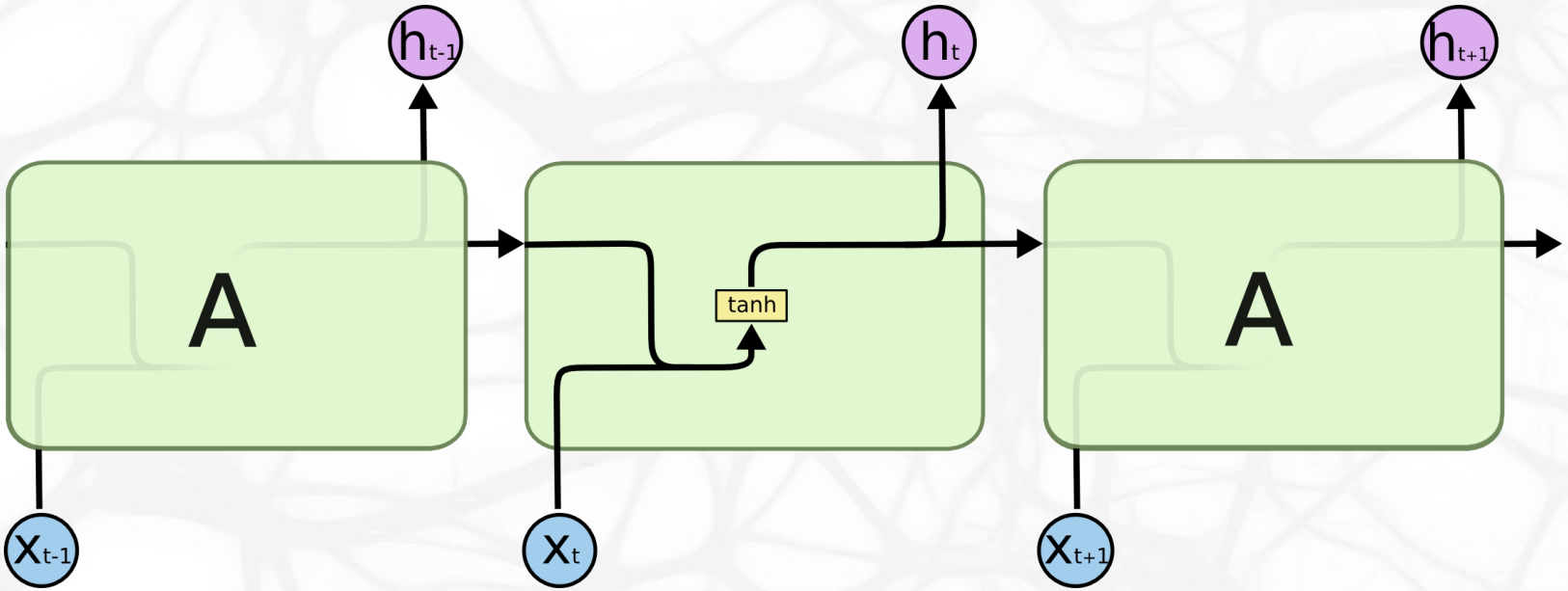
NPFL114, Lecture 09

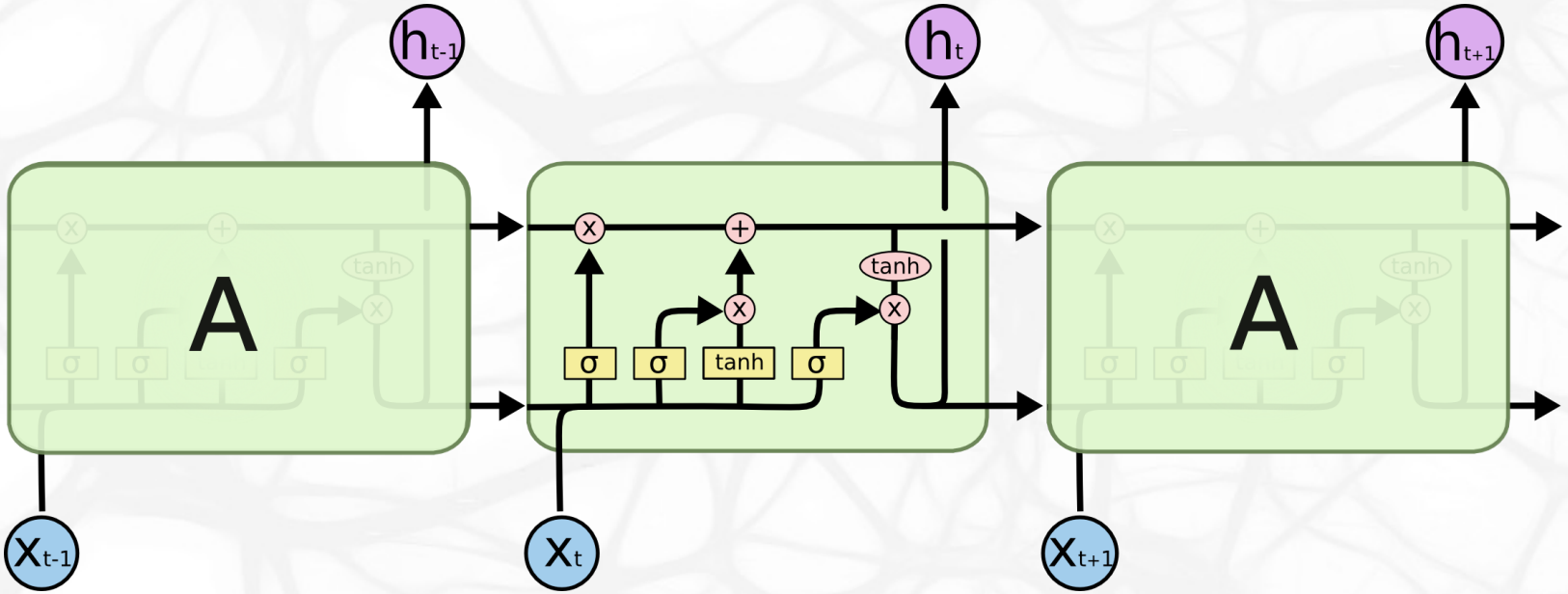
Recurrent Neural Networks III, Machine Translation

Milan Straka

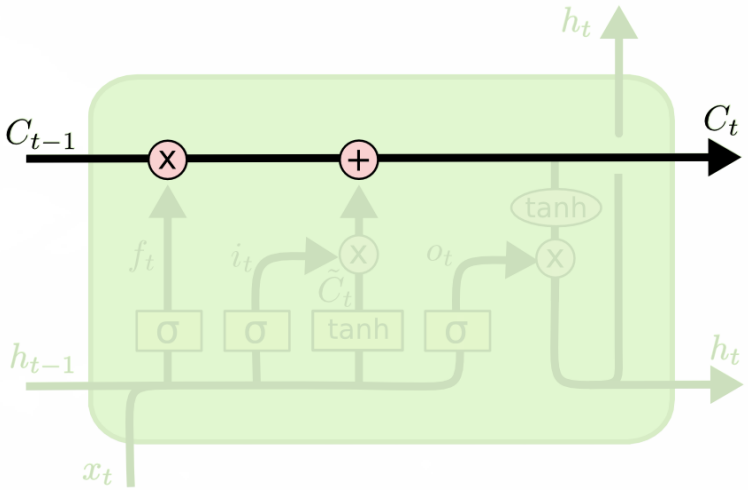
Basic RNN Cell



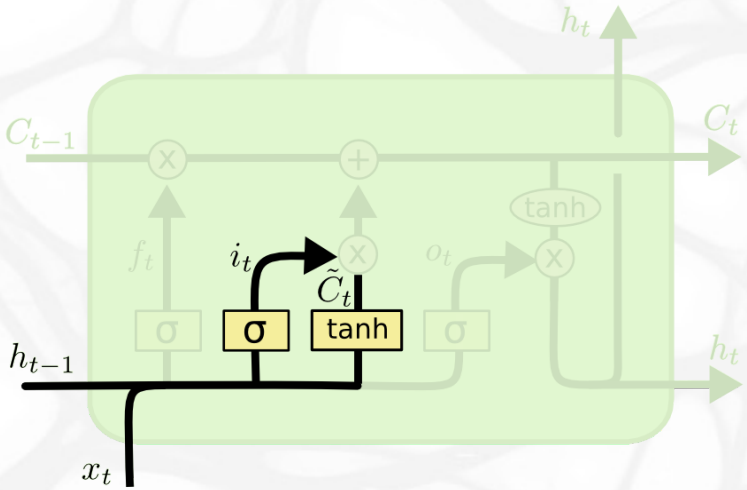
LSTM Cell



LSTM Cell



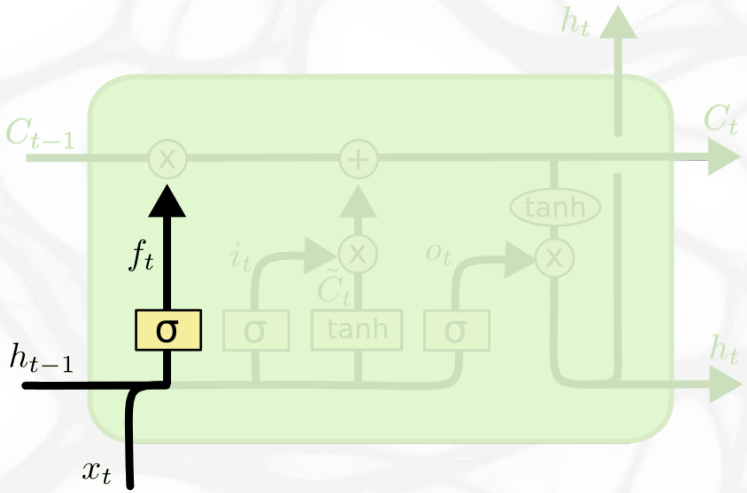
LSTM Cell



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

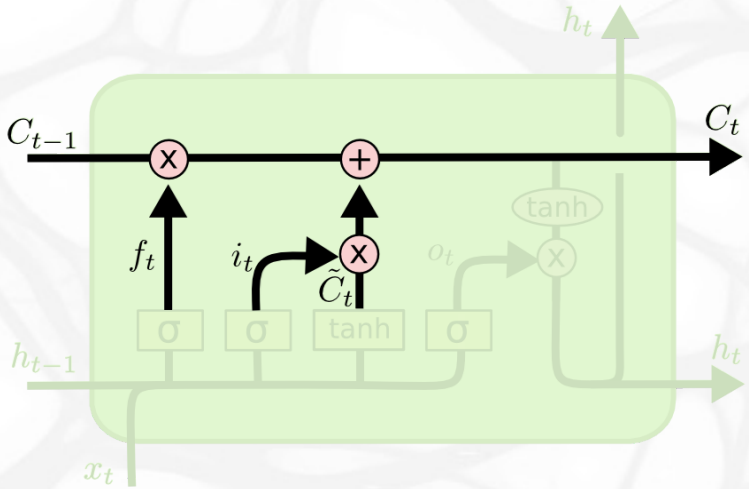
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM Cell



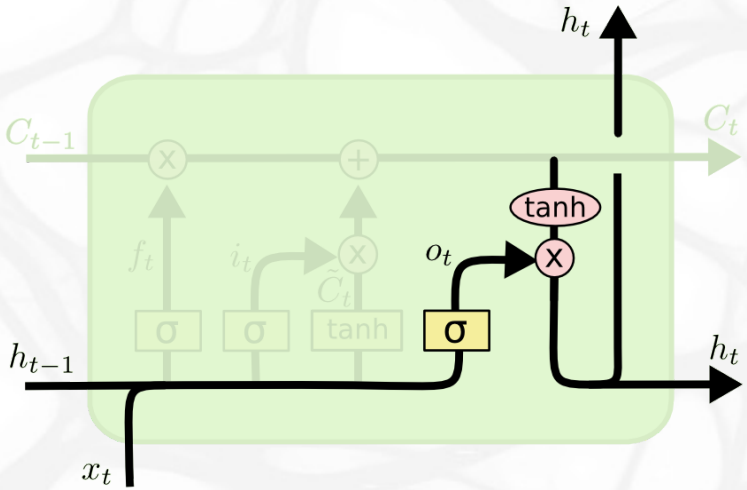
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM Cell



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

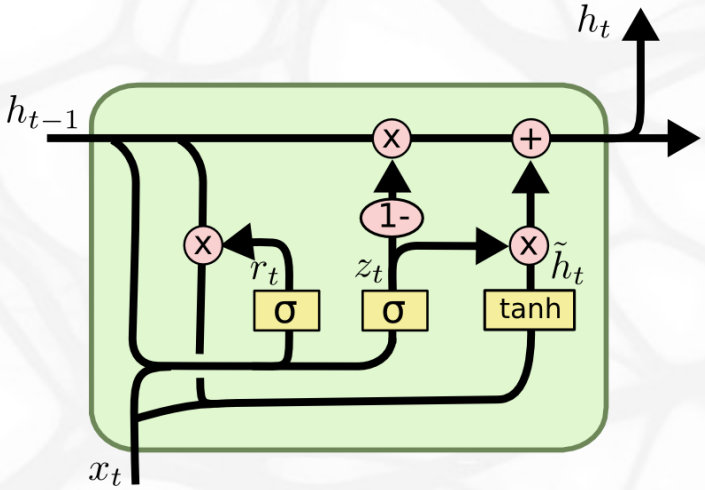
LSTM Cell



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

GRU Cell



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Highway Networks

Highway Networks

For input $\boldsymbol{x}$, fully connected layer computes

$$\boldsymbol{y} \leftarrow H(\boldsymbol{x}, \boldsymbol{W}_H).$$

Highway Networks

For input $\boldsymbol{x}$, fully connected layer computes

$$\boldsymbol{y} \leftarrow H(\boldsymbol{x}, \boldsymbol{W}_H).$$

Highway networks add residual connection with gating:

$$\boldsymbol{y} \leftarrow H(\boldsymbol{x}, \boldsymbol{W}_H) \cdot T(\boldsymbol{x}, \boldsymbol{W}_T) + \boldsymbol{x} \cdot (1 - T(\boldsymbol{x}, \boldsymbol{W}_T)).$$

Highway Networks

For input $\boldsymbol{x}$, fully connected layer computes

$$\boldsymbol{y} \leftarrow H(\boldsymbol{x}, \boldsymbol{W}_H).$$

Highway networks add residual connection with gating:

$$\boldsymbol{y} \leftarrow H(\boldsymbol{x}, \boldsymbol{W}_H) \cdot T(\boldsymbol{x}, \boldsymbol{W}_T) + \boldsymbol{x} \cdot (1 - T(\boldsymbol{x}, \boldsymbol{W}_T)).$$

Usually, the gating is defined as

$$T(\boldsymbol{x}, \boldsymbol{W}_T) \leftarrow \sigma(\boldsymbol{W}_T \boldsymbol{x} + \boldsymbol{b}_T).$$

Highway Networks

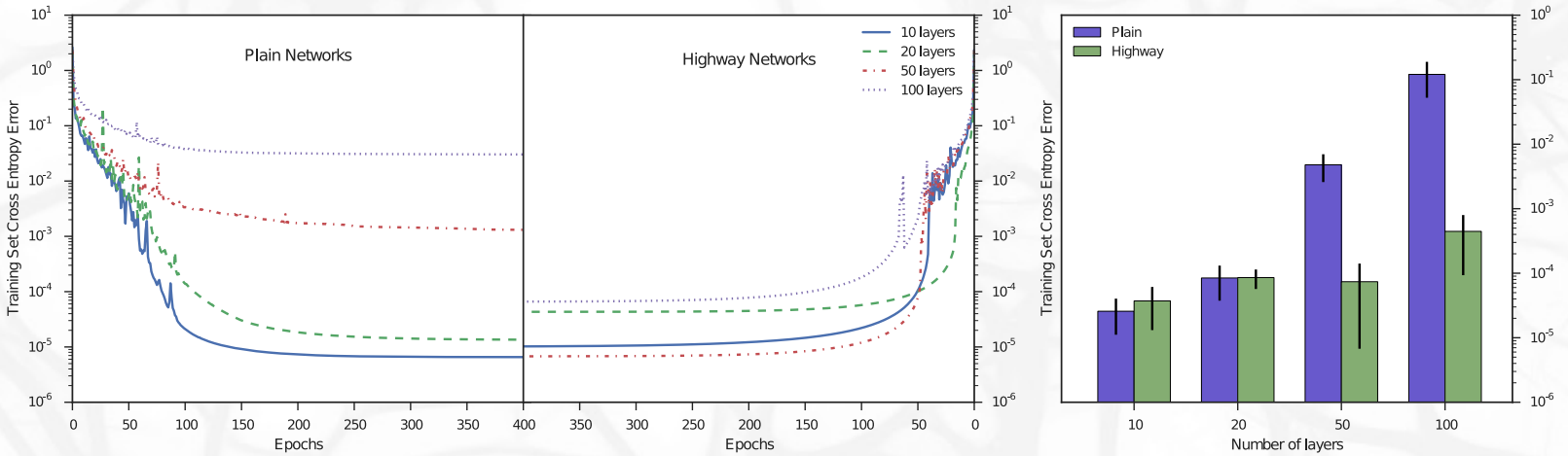
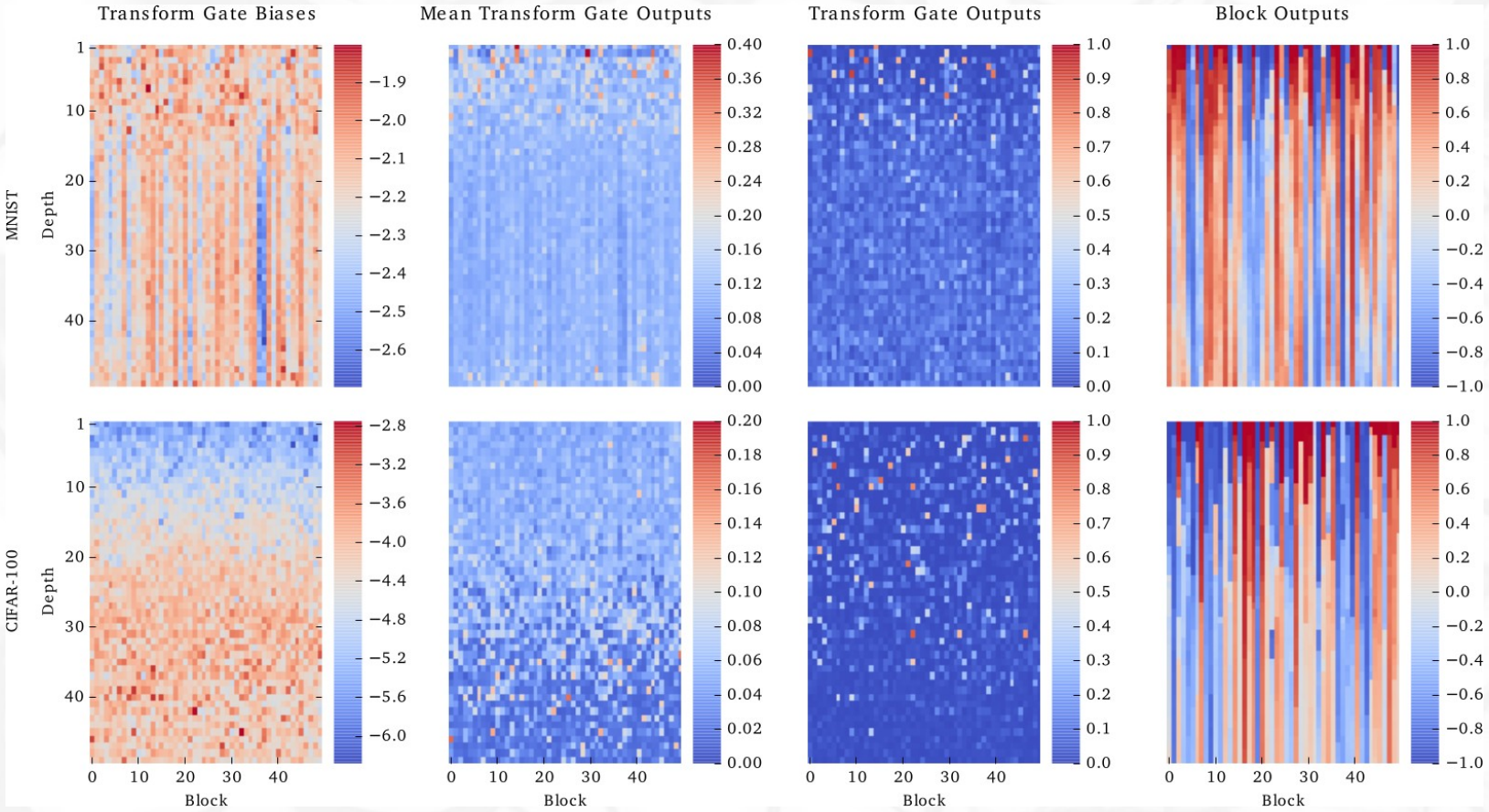


Figure 1: Comparison of optimization of plain networks and highway networks of various depths. *Left:* The training curves for the best hyperparameter settings obtained for each network depth. *Right:* Mean performance of top 10 (out of 100) hyperparameter settings. Plain n tworks become much harder to optimize with increasing depth, while highway networks with up to 100 layers can still be optimized well. Best viewed on screen (larger version included in Supplementary Material).

Highway Networks



Highway Networks

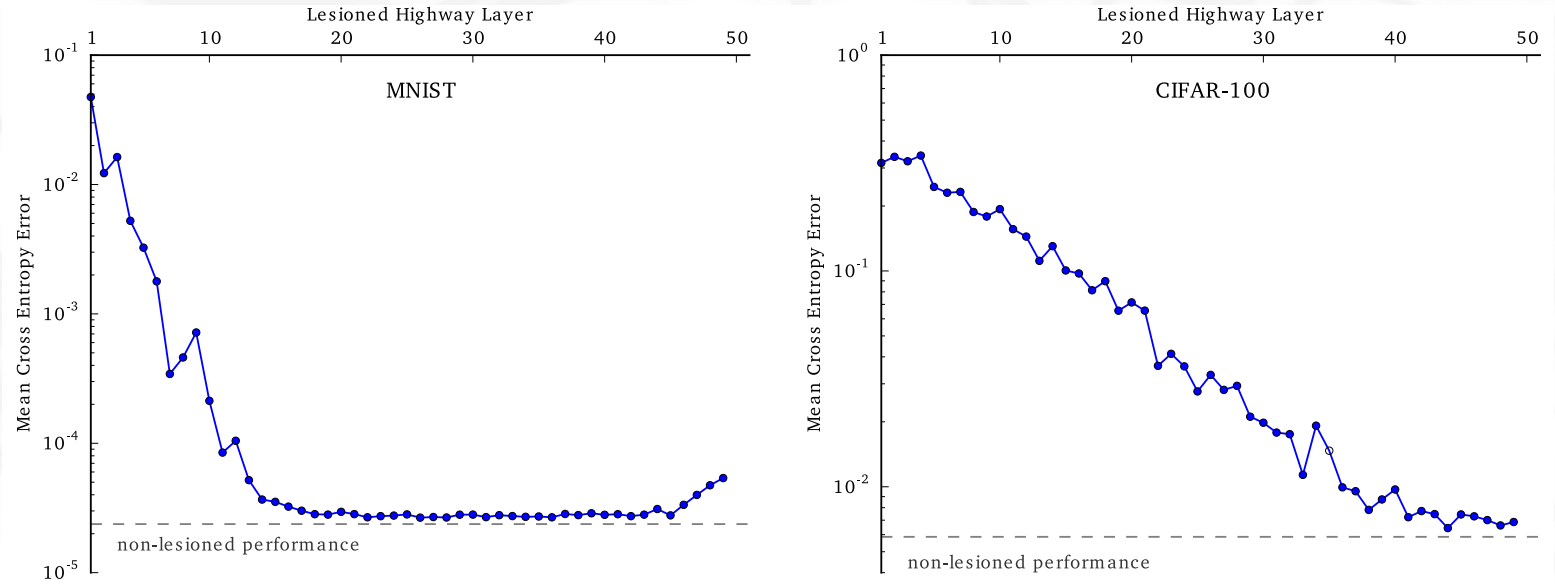
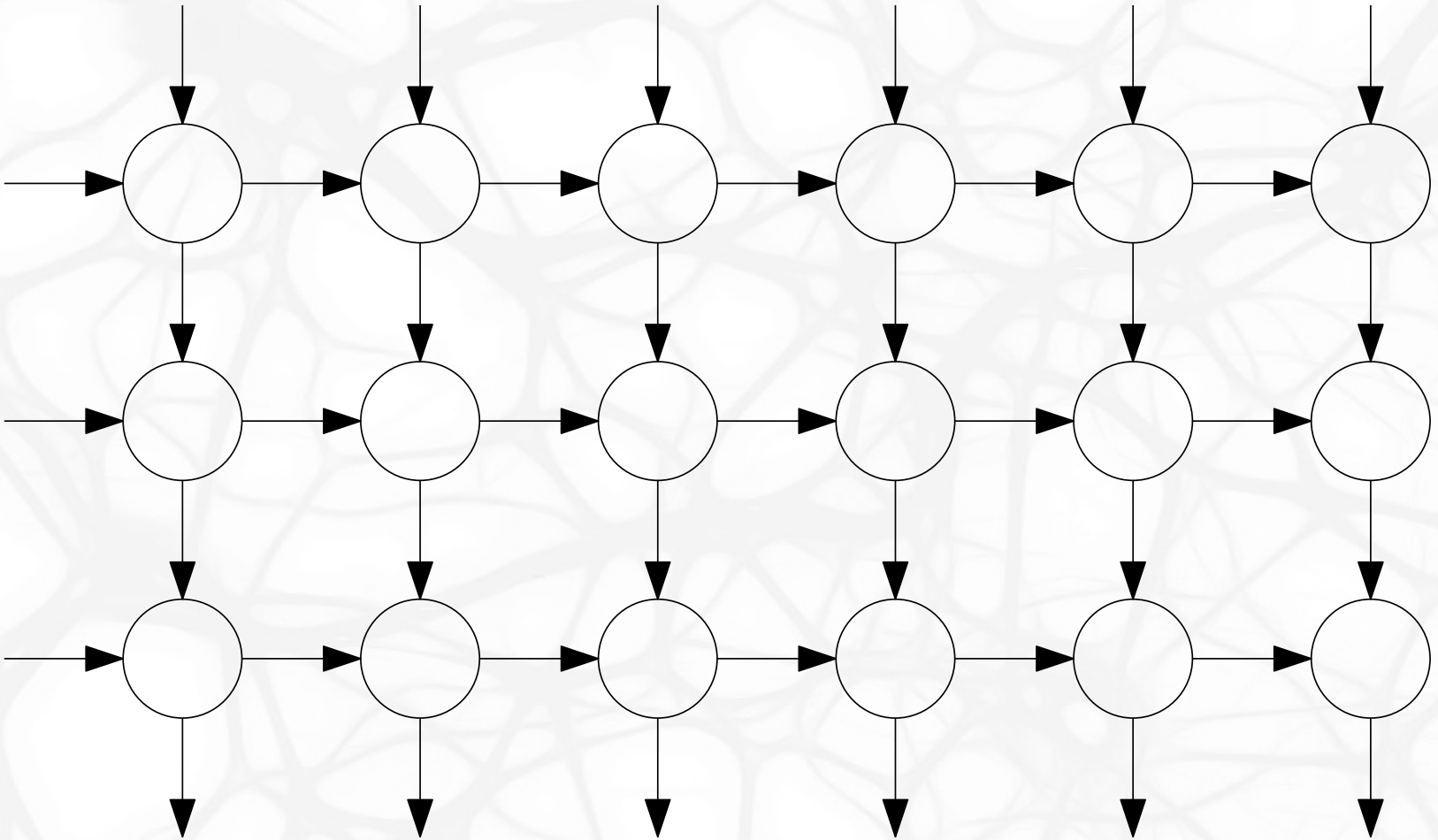
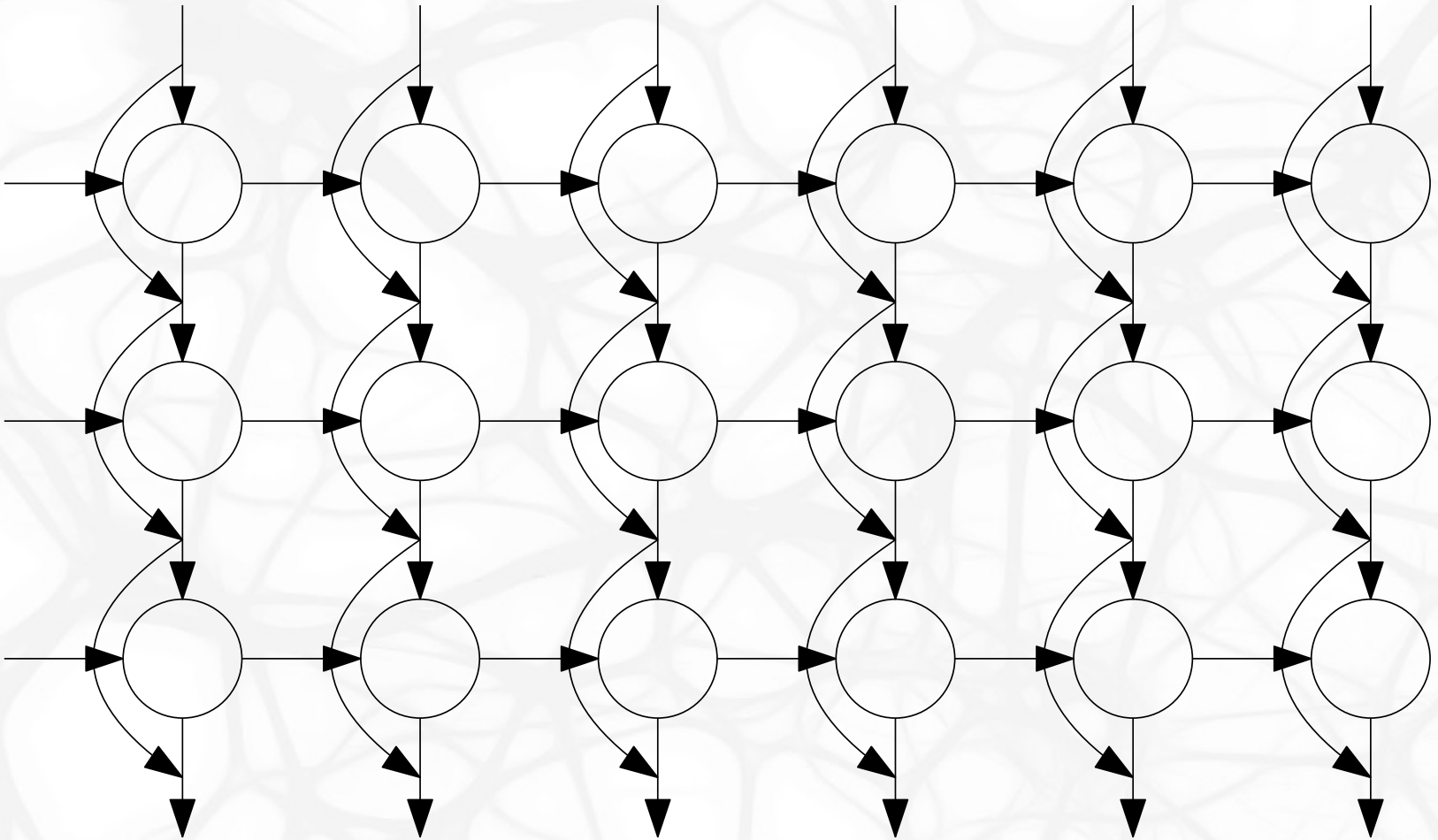


Figure 4: Lesioned training set performance (y-axis) of he best 50-layer highway networks on MNIST (left) and CIFAR-100 (right), as a function of the lesioned layer (x-axis). Evaluated on the full training set while forcefully closing all the transform gates of a single layer at a time. The non-lesioned performance is indicated as a dashed line at the bottom.

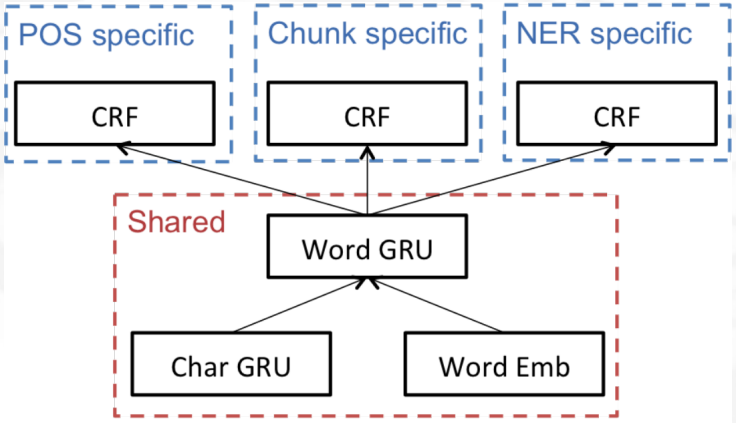
Multilayer RNNs



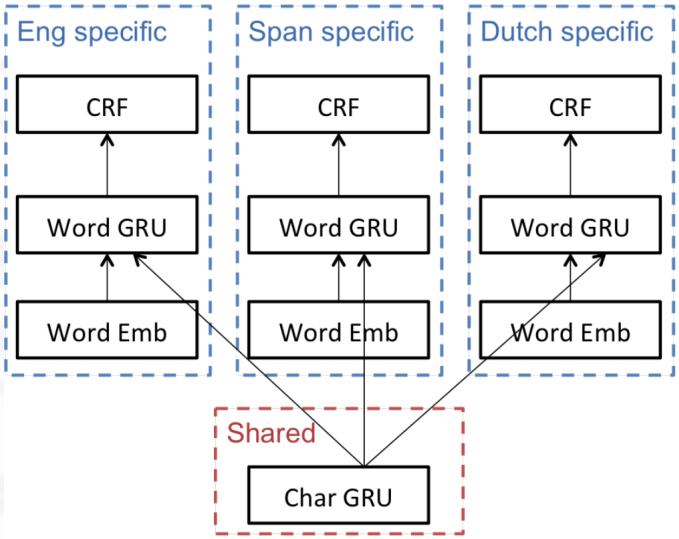
Multilayer RNNs



Multitask Learning

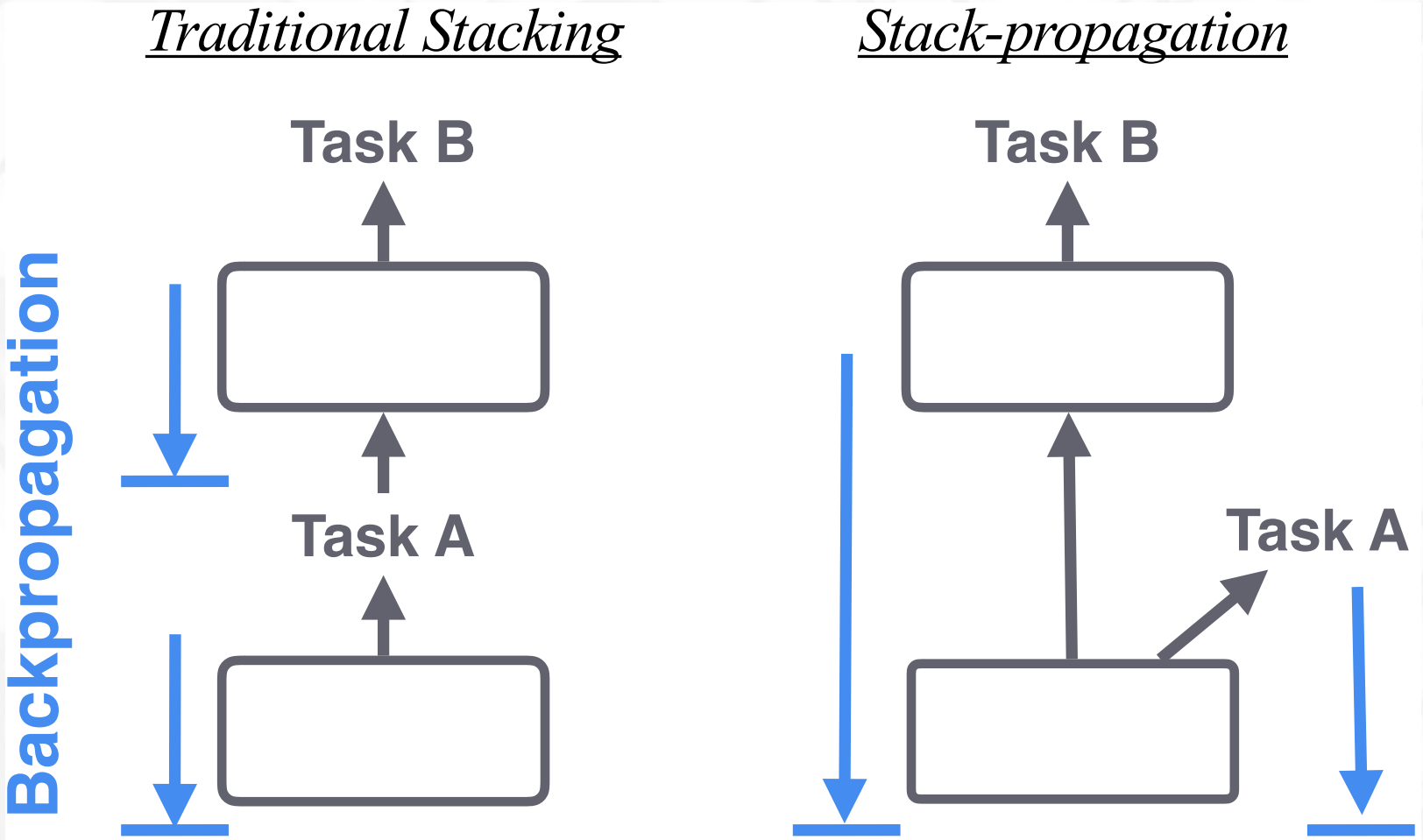


(a) Multi-Task Joint Training

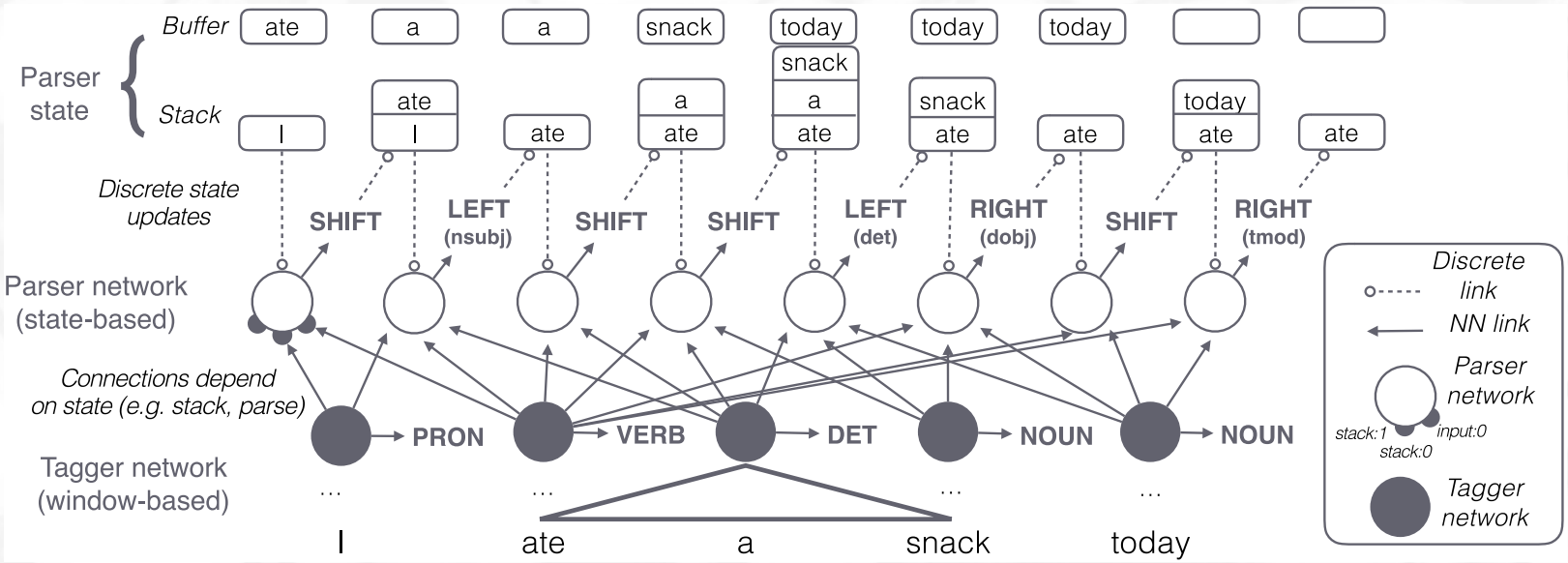


(b) Cross-Lingual Joint Training

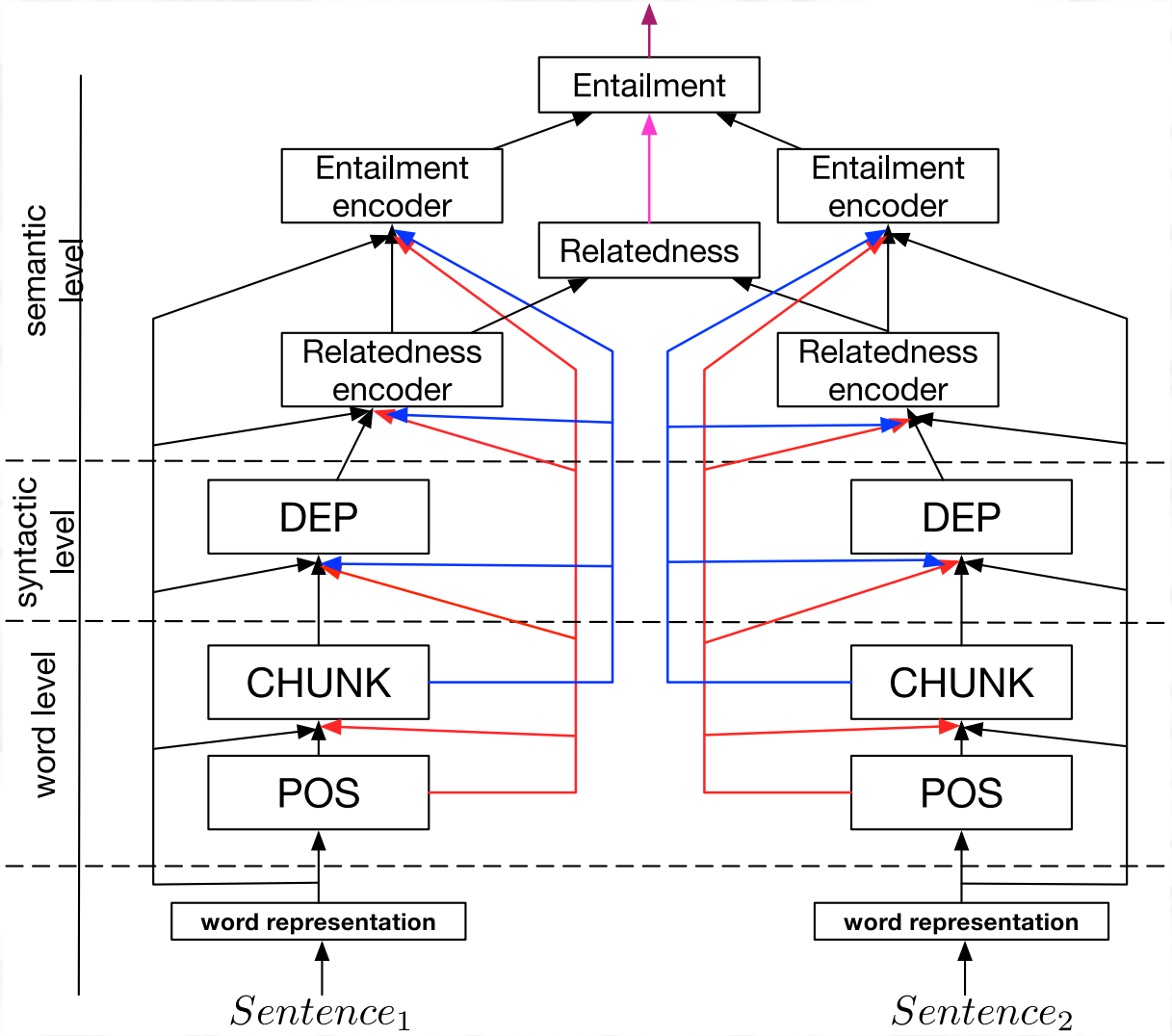
Multitask Learning



Multitask Learning



Multitask Learning



Regularizing RNNs

Dropout

- Using dropout on hidden states interferes with long-term dependencies.

Regularizing RNNs

Dropout

- Using dropout on hidden states interferes with long-term dependencies.
- However, using dropout on the inputs and outputs works well and is used frequently.

Regularizing RNNs

Dropout

- Using dropout on hidden states interferes with long-term dependencies.
- However, using dropout on the inputs and outputs works well and is used frequently.
  - In case residual connections are present, the output dropout needs to be applied before adding the residual connection.

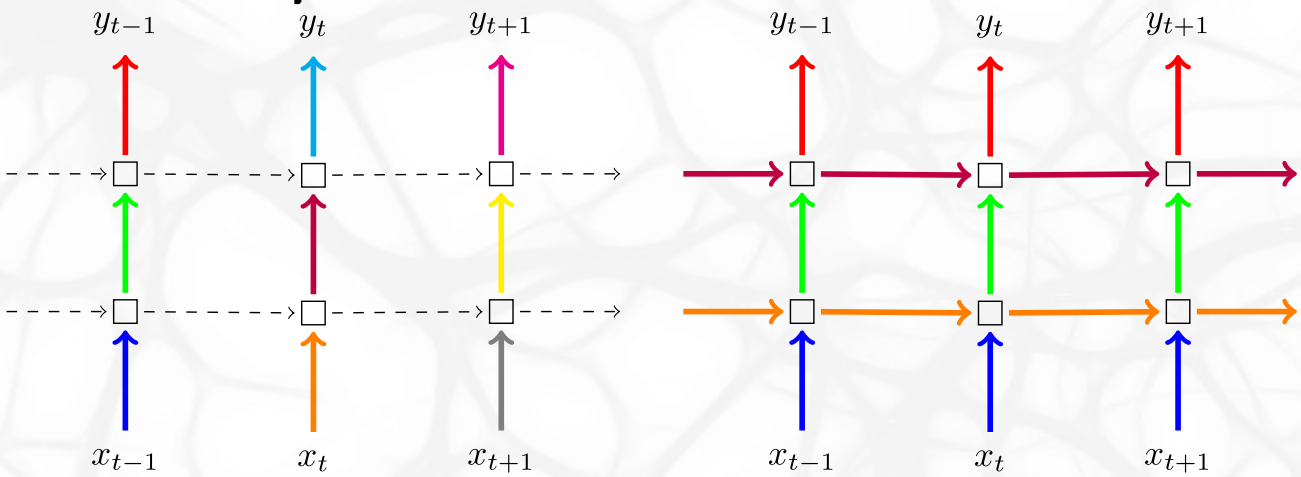
Regularizing RNNs

Dropout

- Using dropout on hidden states interferes with long-term dependencies.
- However, using dropout on the inputs and outputs works well and is used frequently.
  - In case residual connections are present, the output dropout needs to be applied before adding the residual connection.
- Several techniques were designed to allow using dropout on hidden states.
  - Variational Dropout
  - Recurrent Dropout
  - Zoneout

Regularizing RNNs

Variational Dropout

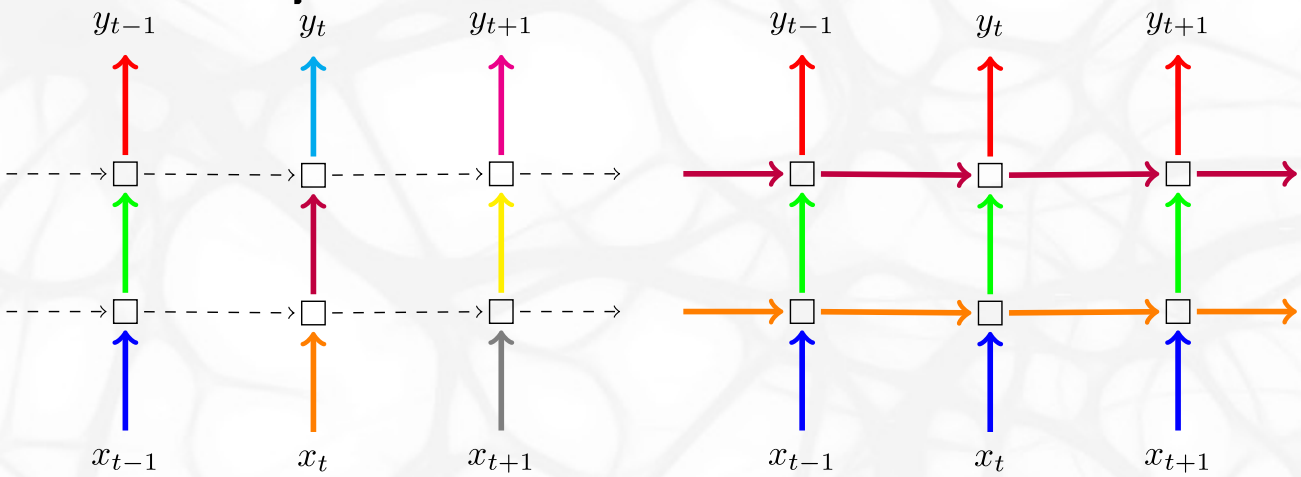


(a) Naive dropout RNN

(b) Variational RNN

Regularizing RNNs

Variational Dropout



(a) Naive dropout RNN

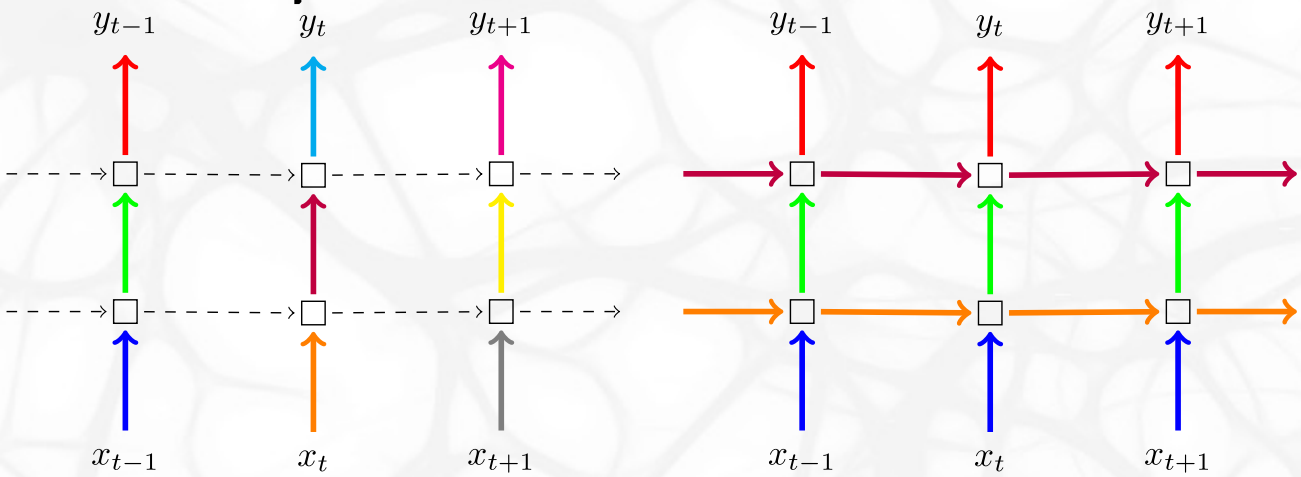
(b) Variational RNN

Recurrent Dropout

Dropout only candidate states (i.e., values added to the memory cell in LSTM and previous state in GRU).

Regularizing RNNs

Variational Dropout



(a) Naive dropout RNN

(b) Variational RNN

Recurrent Dropout

Dropout only candidate states (i.e., values added to the memory cell in LSTM and previous state in GRU).

Zoneout

Randomly preserve hidden activations instead of dropping them.

Regularizing RNNs

Batch Normalization

Very fragile and sensitive to proper initialization (there were papers with negative results until people managed to make it work).

Regularizing RNNs

Batch Normalization

Very fragile and sensitive to proper initialization (there were papers with negative results until people managed to make it work).

Layer Normalization

Much more stable than batch normalization.

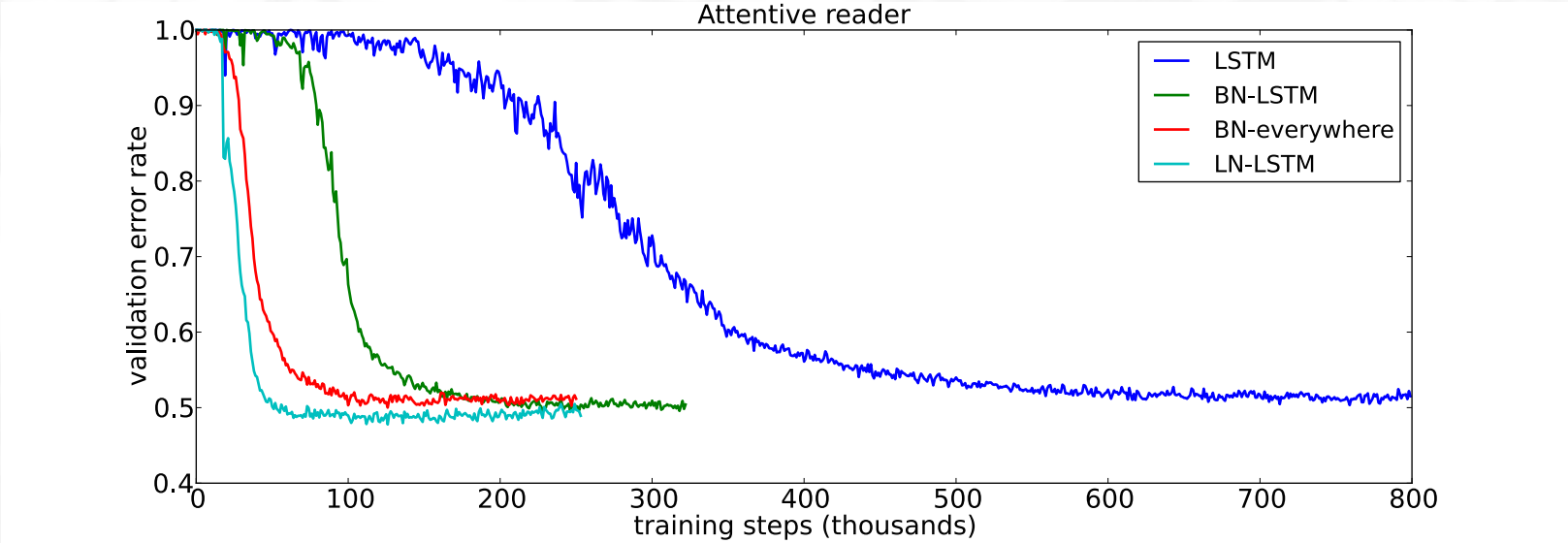


Figure 2: Validation curves for the attentive reader model. BN results are taken from [Cooijmans et al., 2016].

TF Layers and Variable Creation

When tf.layers.Dense or tf.nn.rnn_cell.{GRU,...}Cell is constructed, no variables are yet created (one of the reasons is that the size of the input is not known at that time).

TF Layers and Variable Creation

When `tf.layers.Dense` or `tf.nn.rnn_cell.{GRU,...}Cell` is constructed, no variables are yet created (one of the reasons is that the size of the input is not known at that time).

The variables are instantiated either automatically during first layer application, or during `build` call (which usually needs to be passed an input shape).

TF Layers and Variable Creation

When tf.layers.Dense or tf.nn.rnn_cell.{GRU,...}Cell is constructed, no variables are yet created (one of the reasons is that the size of the input is not known at that time).

The variables are instantiated either automatically during first layer application, or during build call (which usually needs to be passed an input shape).

If no variable scope is defined during variable instantiation, a unique one is generated; otherwise, it is respected. This causes problems during tf.nn.[bidirectional_]dynamic_rnn, which opens a fixed scope if None is passed. Therefore, the safest approach is to always provide a unique scope for tf.nn.[bidirectional_]dynamic_rnn.

Refreshing Word Embeddings

Recall that instead of one-hot encoding, distributed representation of words is preferred.

Refreshing Word Embeddings

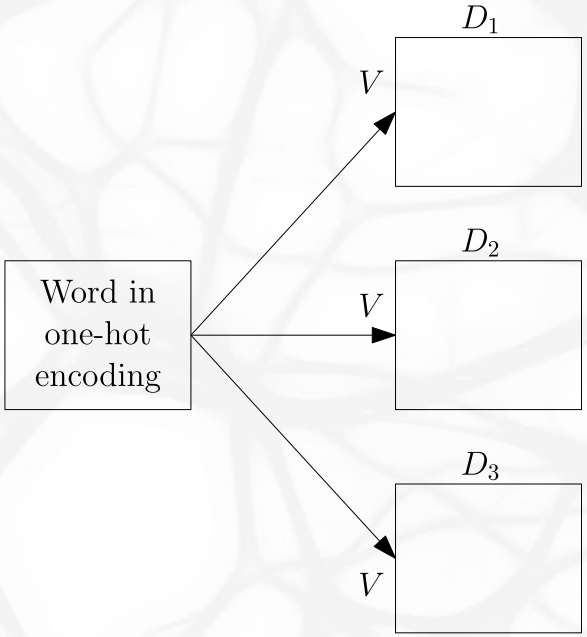
Recall that instead of one-hot encoding, distributed representation of words is preferred.

The word embedding layer is in fact just a fully connected layer on top of one-hot encoding. However, it is important that this layer is *shared* across the whole network.

Refreshing Word Embeddings

Recall that instead of one-hot encoding, distributed representation of words is preferred.

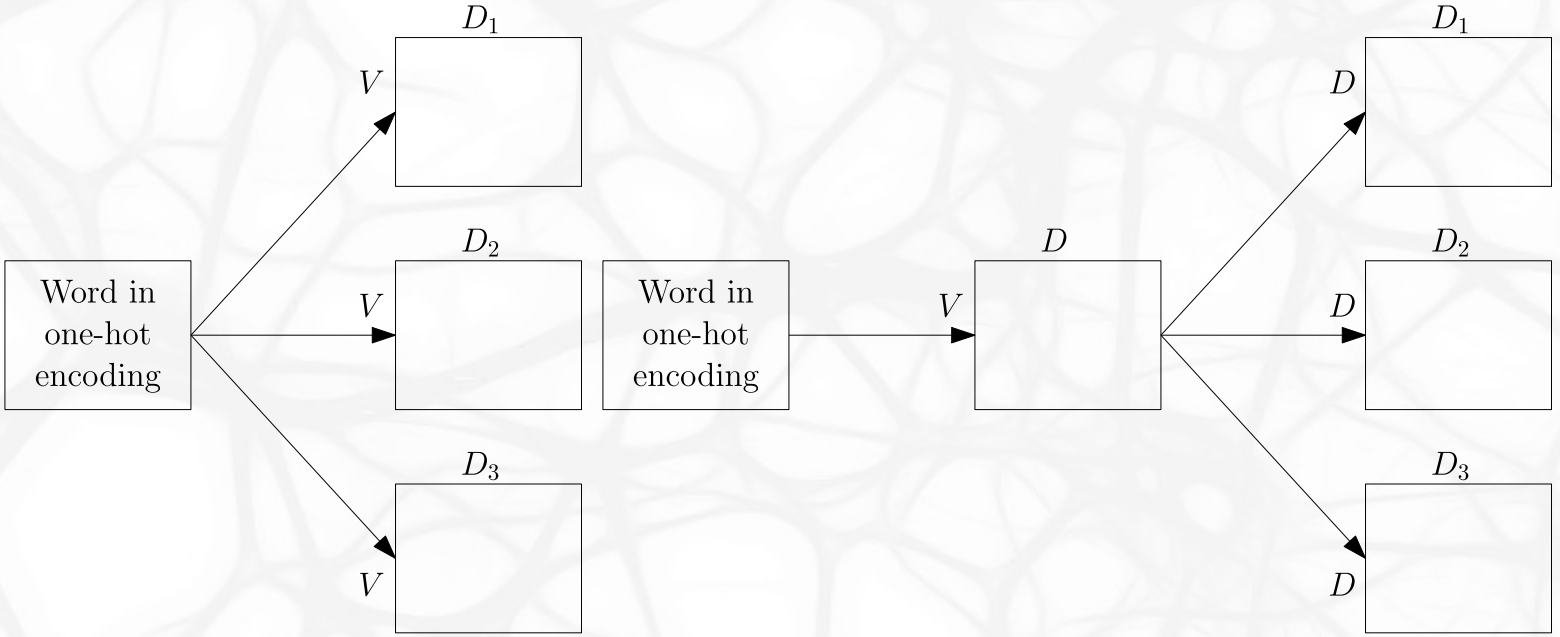
The word embedding layer is in fact just a fully connected layer on top of one-hot encoding. However, it is important that this layer is *shared* across the whole network.



Refreshing Word Embeddings

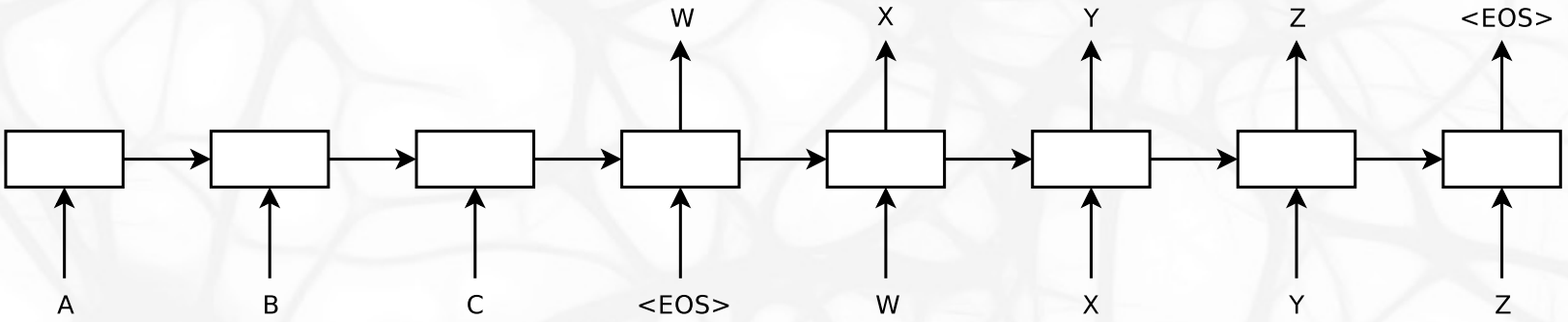
Recall that instead of one-hot encoding, distributed representation of words is preferred.

The word embedding layer is in fact just a fully connected layer on top of one-hot encoding. However, it is important that this layer is *shared* across the whole network.

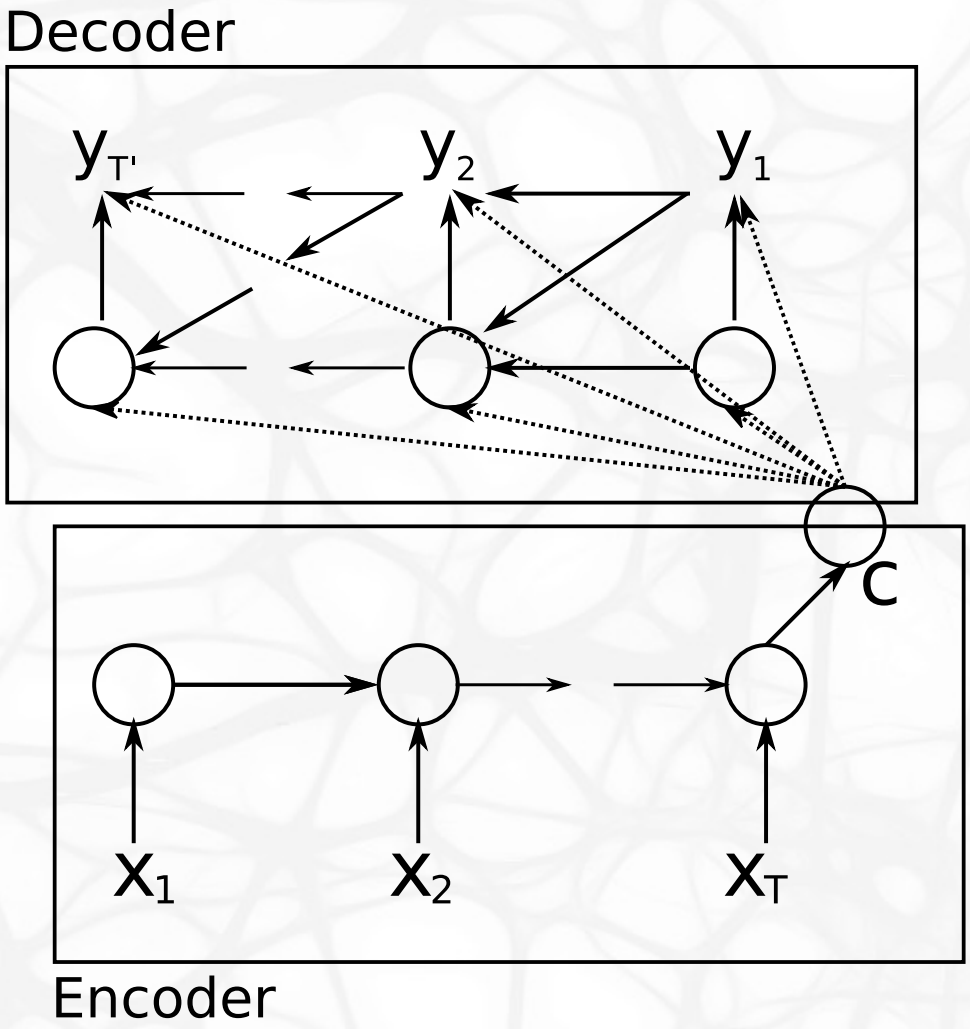


Sequence-to-Sequence Architecture

Sequence-to-Sequence Architecture



Sequence-to-Sequence Architecture



Sequence-to-Sequence Architecture

Training

The so-called *teacher forcing* is used during training – the gold outputs are used as inputs during training.

Sequence-to-Sequence Architecture

Training

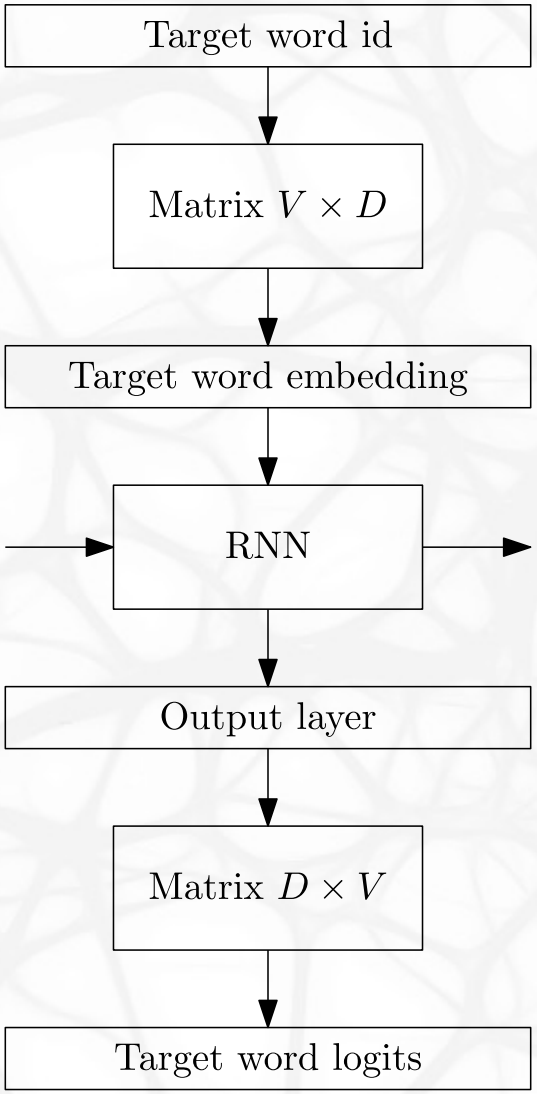
The so-called *teacher forcing* is used during training – the gold outputs are used as inputs during training.

Inference

During inference, the network processes its own predictions.

Usually, the generated logits are processed by an $\arg\max$, the chosen word embedded and used as next input.

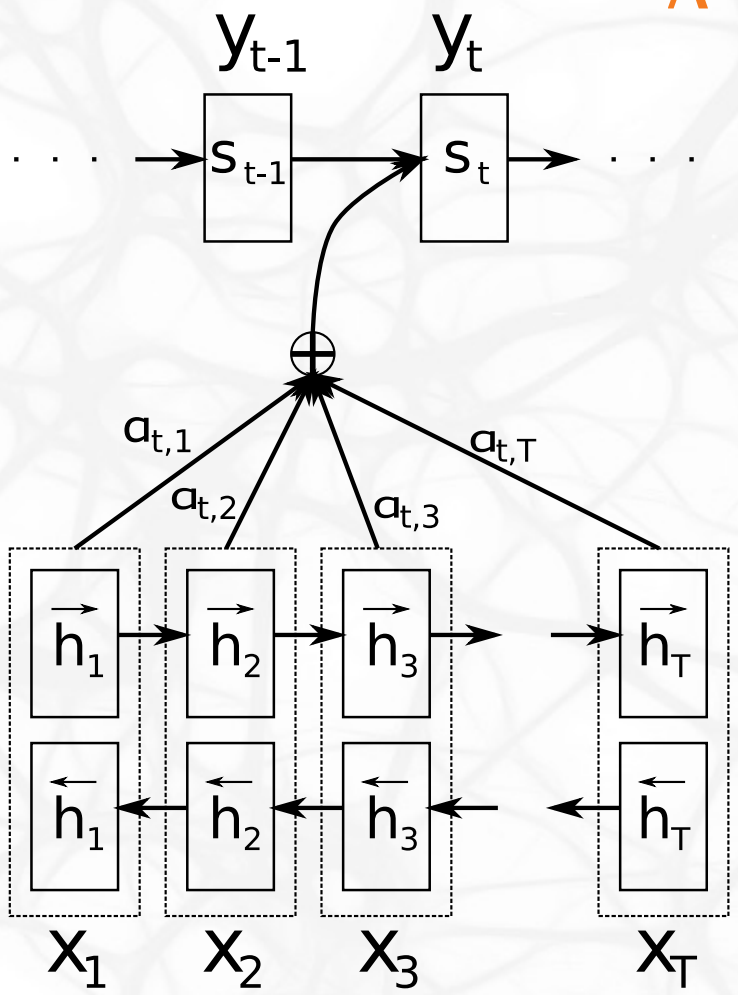
Tying Word Embeddings



Attention

As another input during decoding, we add *context vector* c_i:

$$\boldsymbol{s}_i = f(\boldsymbol{s}_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i).$$



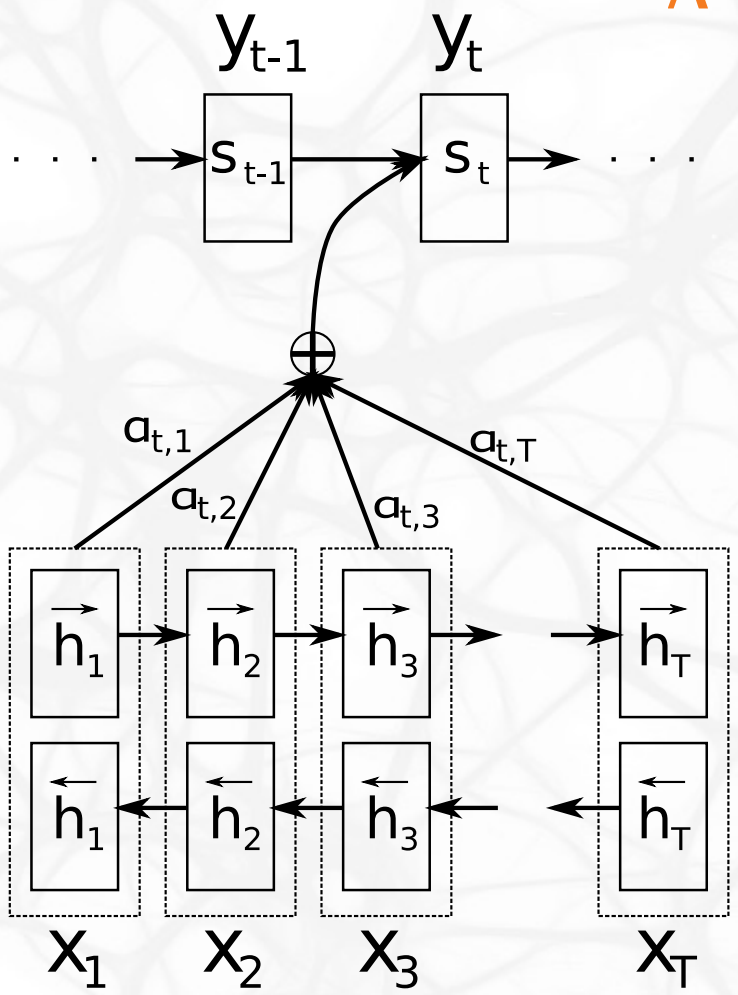
Attention

As another input during decoding, we add *context vector* c_i:

$$\boldsymbol{s}_i = f(\boldsymbol{s}_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i).$$

We compute the context vector as a weighted combination of source sentence encoded outputs:

$$\boldsymbol{c}_i = \sum_j \alpha_{ij} \boldsymbol{h}_j$$



Attention

As another input during decoding, we add *context vector* c_i:

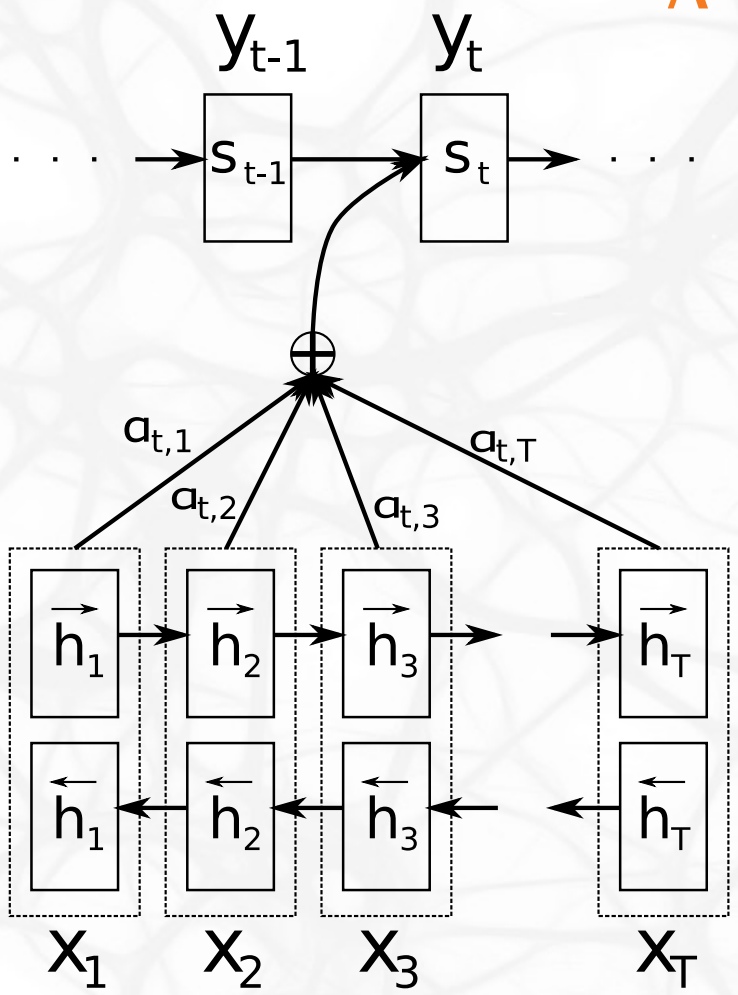
$$\boldsymbol{s}_i = f(\boldsymbol{s}_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i).$$

We compute the context vector as a weighted combination of source sentence encoded outputs:

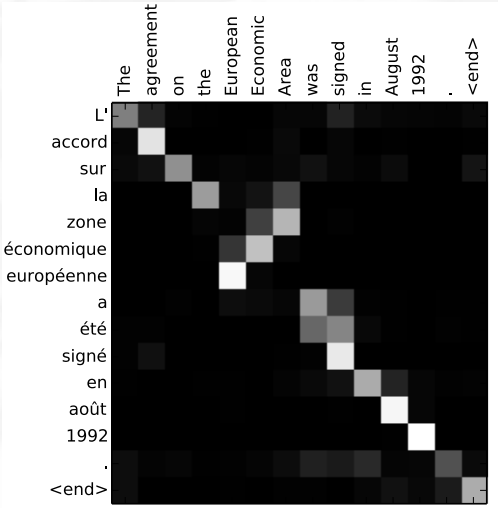
$$\boldsymbol{c}_i = \sum_j \alpha_{ij} \boldsymbol{h}_j$$

The weights α_{ij} are softmax of e_{ij} over j, $\boldsymbol{\alpha}_i = \mathrm{softmax}(\boldsymbol{e}_i)$, with e_{ij} being

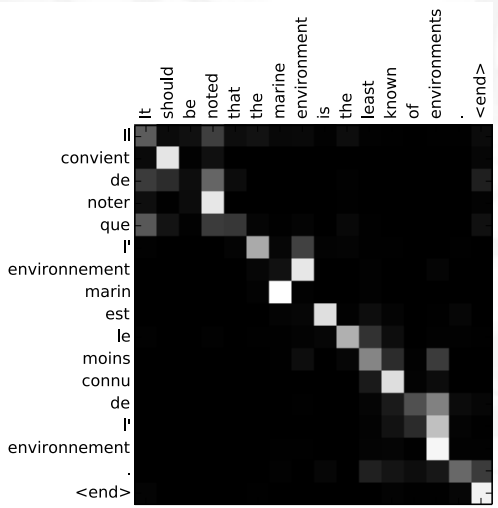
$$e_{ij} = \boldsymbol{v}^\top \tanh(\boldsymbol{V}\boldsymbol{h}_j + \boldsymbol{W}\boldsymbol{s}_{i-1} + \boldsymbol{b}).$$



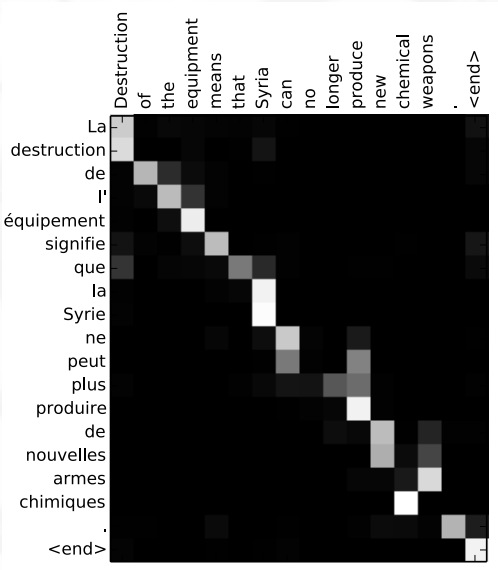
Attention



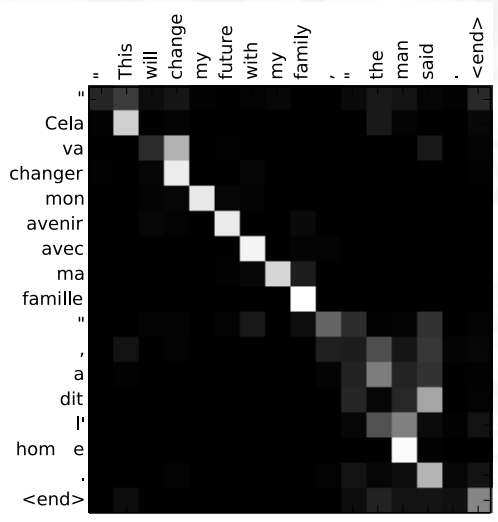
(a)



(b)



(c)



(d)

Subword Units

Translate *subword units* instead of words. The subword units can be generated in several ways, the most commonly used are

Subword Units

Translate *subword units* instead of words. The subword units can be generated in several ways, the most commonly used are

- BPE – Using the *byte pair encoding* algorithm. Start with characters plus a special end-of-word symbol $\cdot$. Then, merge the most occurring symbol pair A, B by a new symbol AB, with the symbol pair never crossing word boundary.

Subword Units

Translate *subword units* instead of words. The subword units can be generated in several ways, the most commonly used are

- BPE – Using the *byte pair encoding* algorithm. Start with characters plus a special end-of-word symbol $\cdot$. Then, merge the most occurring symbol pair A, B by a new symbol AB, with the symbol pair never crossing word boundary.

  Considering a dictionary with words *low, lowest, newer, wider*:

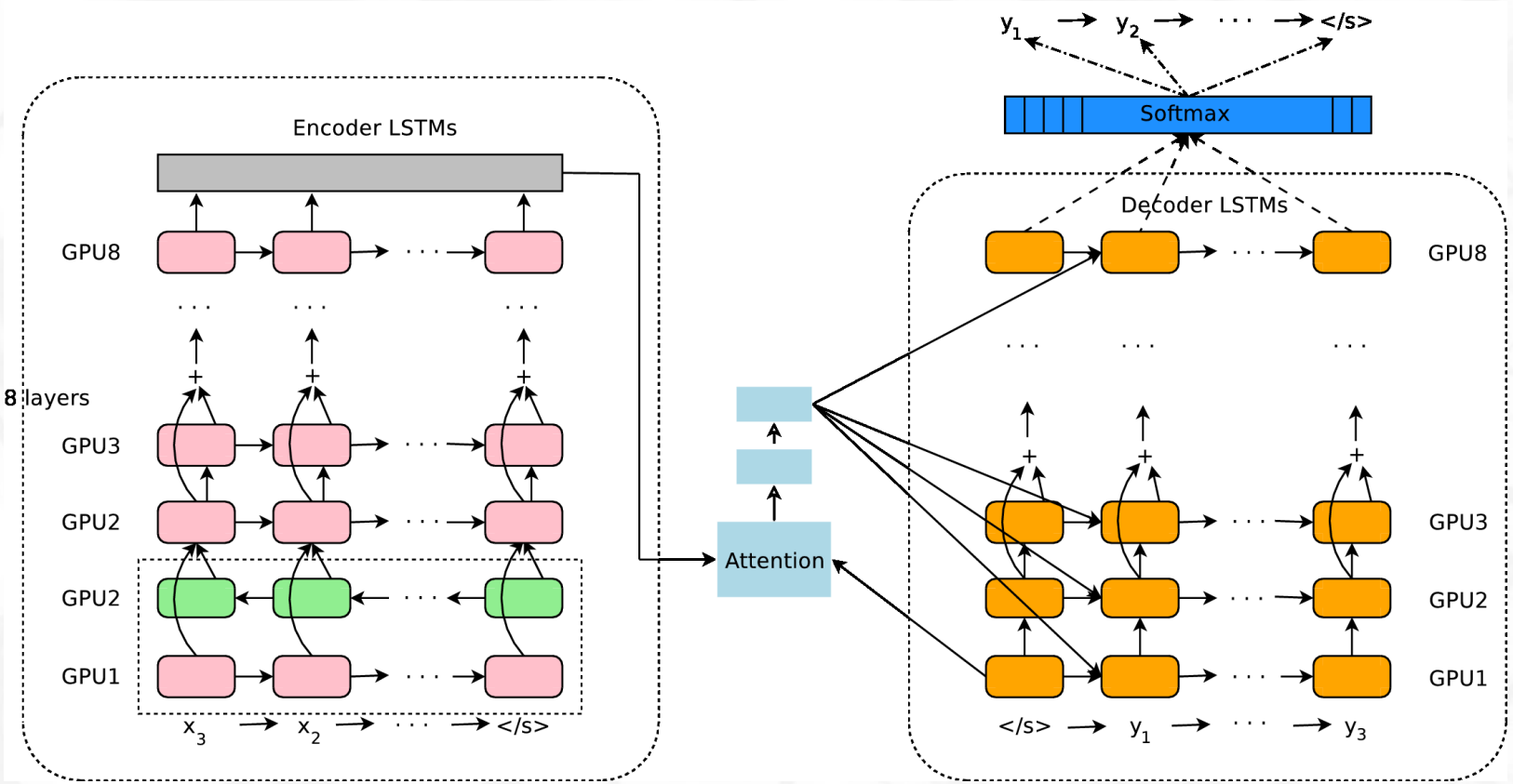
$$r \ \cdot \rightarrow r\cdot$$
$$l \ \ o \rightarrow lo$$
$$lo \ \ w \rightarrow low$$
$$e \ \ r\cdot \rightarrow er\cdot$$

Subword Units

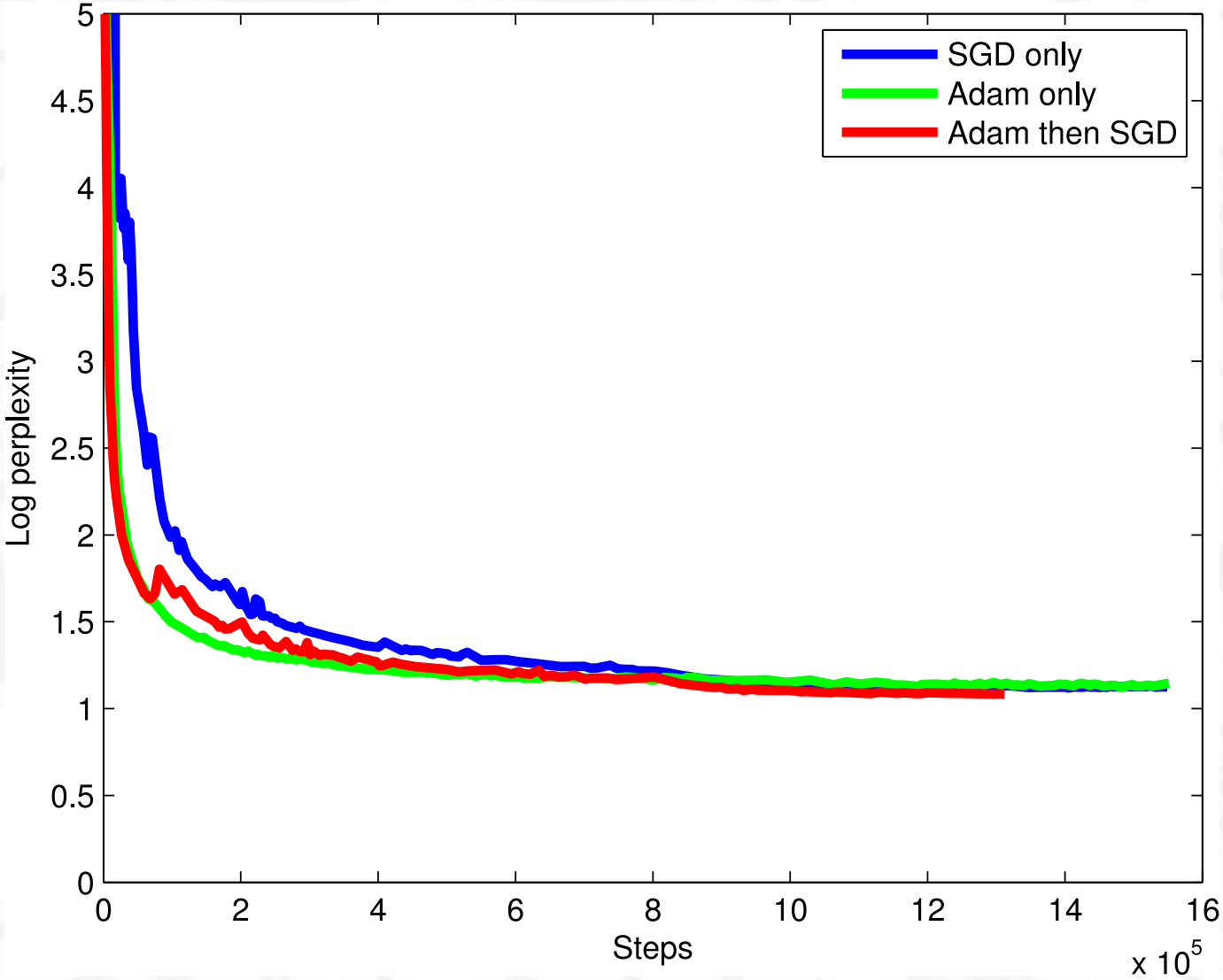
- Wordpieces – Similar heuristic of joining two neighboring symbols (maximizing likelihood under unigram language model).

Usually quite little subword units are used (32k-64k), often generated on the union of the two vocabularies (the so-called *joint BPE* or *shared wordpieces*).

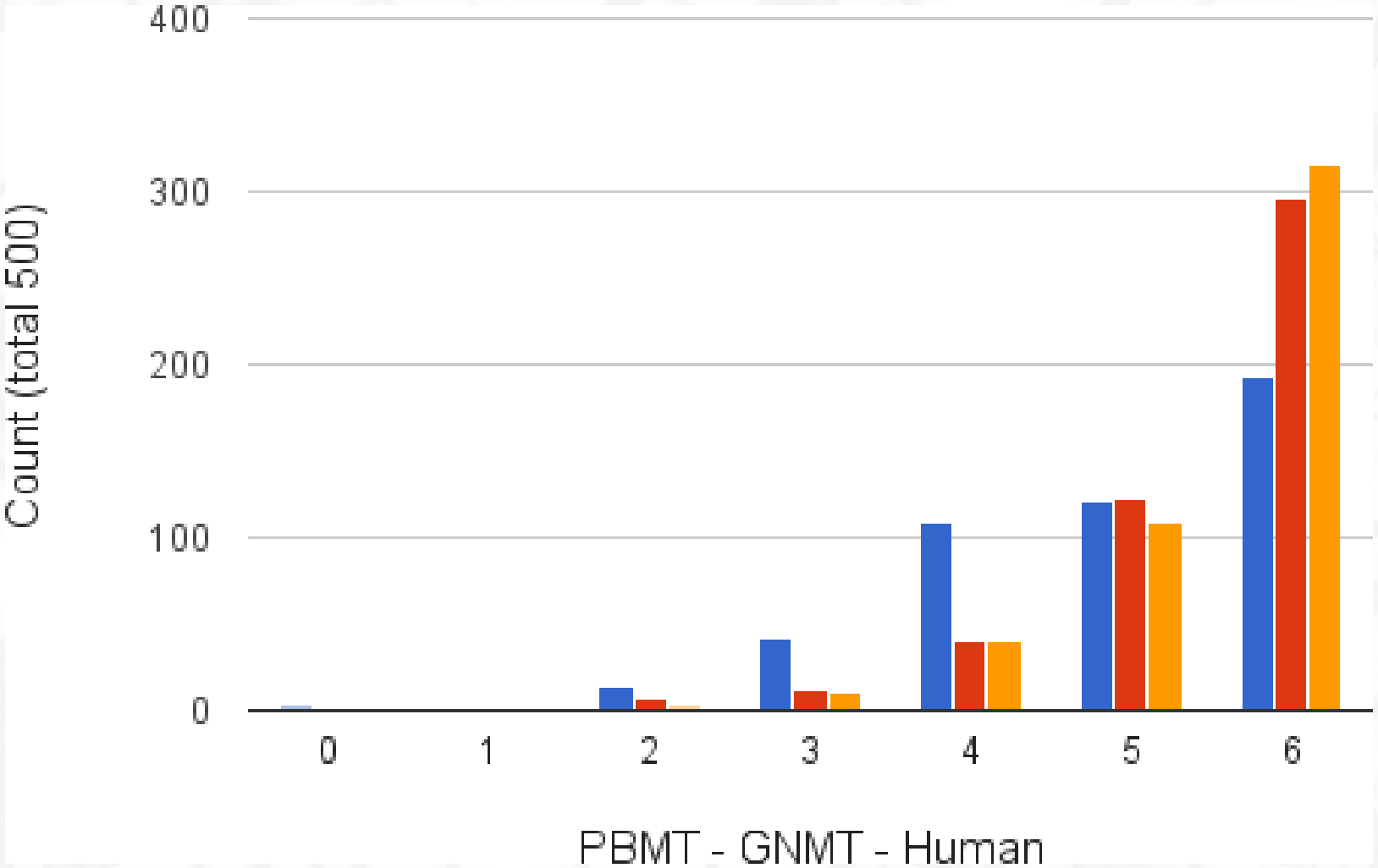
Google NMT


Encoder LSTMs
GPU8
8 layers
GPU3
GPU2
GPU2
GPU1
x3
x2
</s>
Attention
y1
y2
</s>
Softmax
Decoder LSTMs
GPU8
GPU3
GPU2
GPU1
</s>
y1
y3

Google NMT


SGD only
Adam only
Adam then SGD
Log perplexity
Steps
x 10^5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
2
4
6
8
10
12
14
16

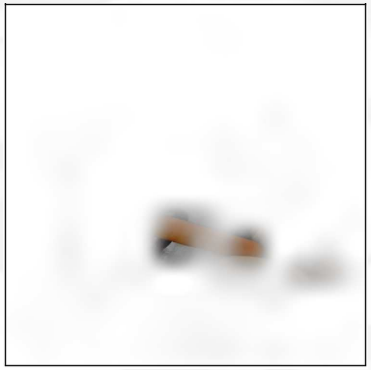
Google NMT



Beyond one Language Pair

Fig. 5. A selection of evaluation results, grouped by human rating.

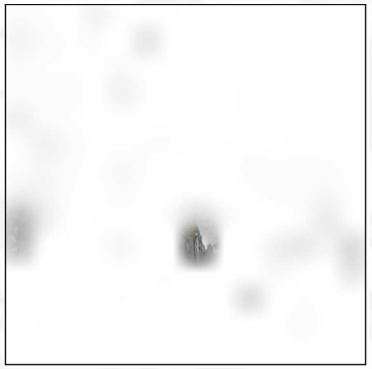
Beyond one Language Pair



What vegetable is the dog chewing on?
MCB: carrot
GT: carrot

What kind of dog is this?
MCB: husky
GT: husky

What kind of flooring does the room have?
MCB: carpet
GT: carpet



What color is the traffic light?
MCB: green
GT: green

Is this an urban area?
MCB: yes
GT: yes

Where are the buildings?
MCB: in background
GT: on left

Multilingual Translation

Many attempts at multilingual translation.

- Individual encoders and decoders, shared attention.
- Shared encoders and decoders.

Attention is All You Need

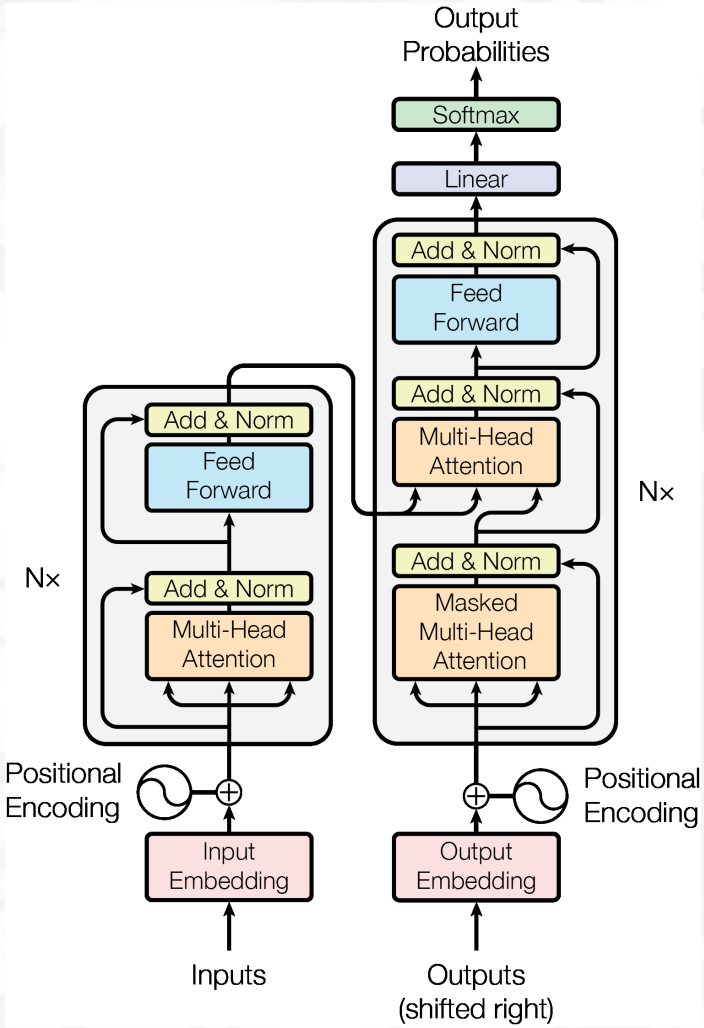


Figure 1: The Transformer - model architecture.

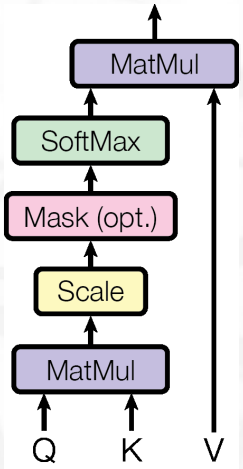
Attention is All You Need

The attention module is defined as:

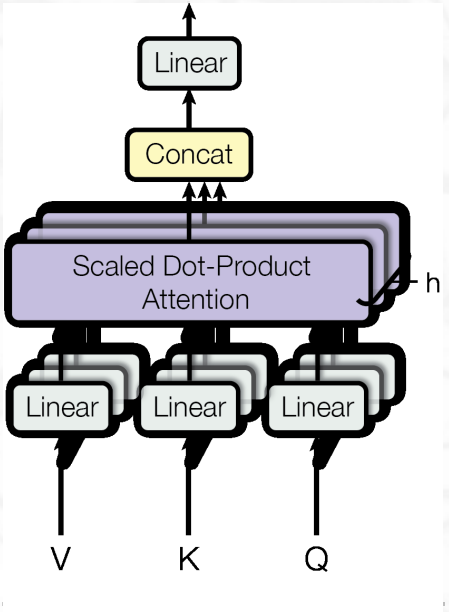
$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}.$$

Multihead attention is used in practice.

Scaled Dot-Product Attention



Multi-Head Attention



Positional Embeddings

We need to encode positional information (which was implicit in RNNs).

Positional Embeddings

We need to encode positional information (which was implicit in RNNs).

- Learned embeddings for every position.

Positional Embeddings

We need to encode positional information (which was implicit in RNNs).

- Learned embeddings for every position.
- Sinusoids of different frequencies:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(pos/10000^{2i/d}\right)$$
$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d}\right)$$

This choice of functions should allow the model to attend to relative positions, since for any fixed k, PE_{pos+k} is a linear function of PE_{pos}.

Why Attention

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Transformers Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	