291K
Deep Learning for Machine Translation
Recurrent Neural Network
Sequence to sequence learning

Lei Li
UCSB
10/11/2021

Outline

- Language Modeling
- Recurrent Neural Network
- Long-short term memory network (LSTM)
- Gated Recurrent Unit (GRU)
- Attention
- Encoder-decoder framework
- LSTM Seq2seq

Language Modeling

- Given a sentence y, estimate the probability
  - $P(y) = \prod_t P(y_{t+1} | y_1 \ldots y_t)$
  - $P(y_{t+1} | y_1 \ldots y_t) = f_\theta(y_1, \ldots, y_t)$

The	cat	sits	on	a	__
y_1	y_2	y_3	y_4	y_5	y_6

	$p(y_6 \| y_1, \ldots, y_5)$
mat	0.15
rug	0.13
chair	0.08
hat	0.05
dog	0.01

Predict Next Token Probability

There are many methods to predict the next token:

- N-gram: assuming $p\left(x_t \middle| x_1, \ldots,\ x_{t-1}\right) = p\left(x_t \middle| x_{t-k}, \ldots,\ x_{t-1}\right)$, and estimate it directly
- Context MLP: use DNN to estimate $p\left(x_t \middle| x_{t-k}, \ldots,\ x_{t-1}\right)$
- CNN-LM (previous lecture)
- RNN-LM, LSTM, GRU
- GPT

CNN Language Model (recap)

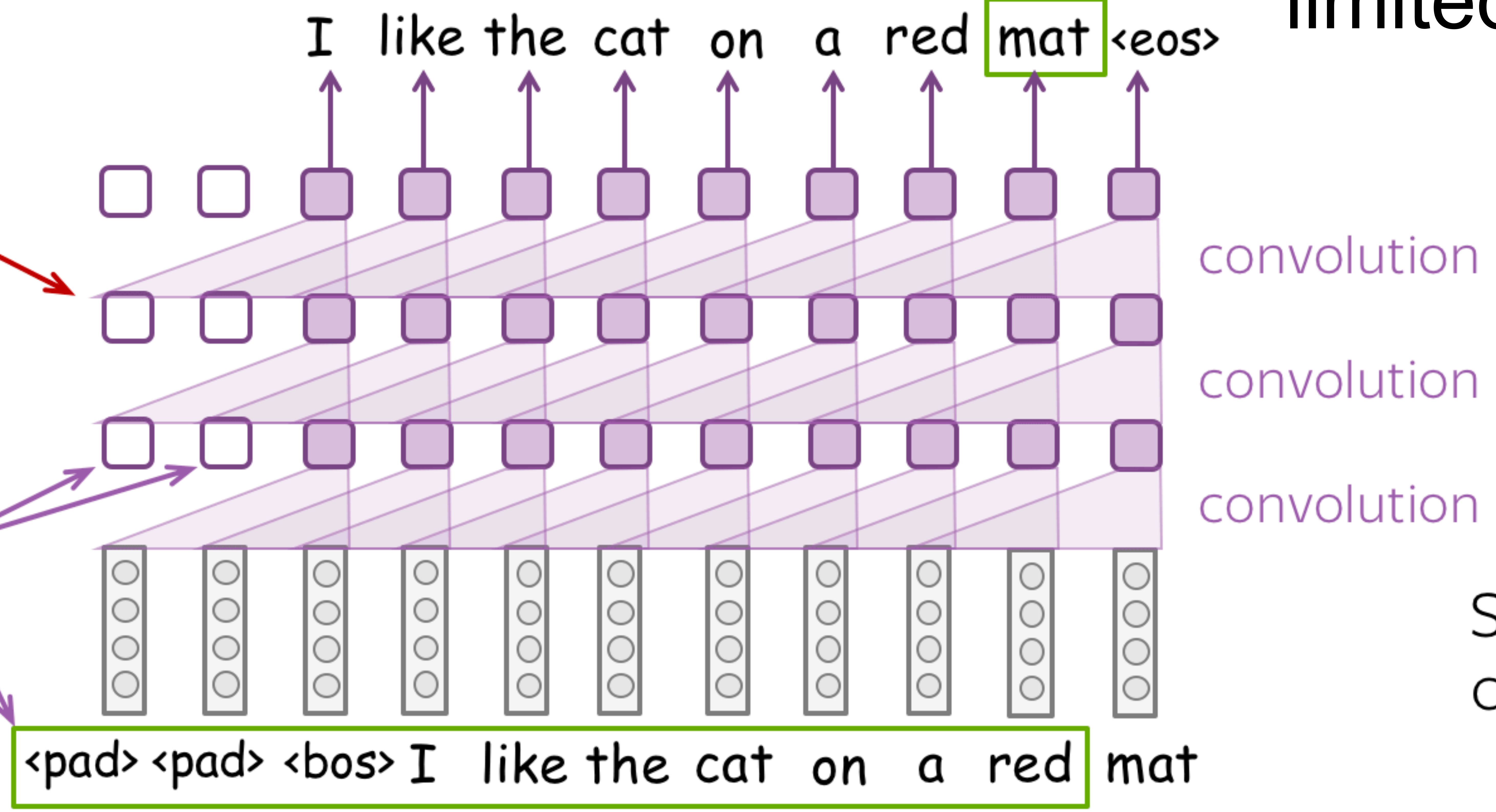
$$P(y_{t+1} \mid y_1, \ldots, y_t) \approx \mathrm{CNN}_\theta(y_{t-k}, \ldots, y_t)$$

predict the next token

But, limited context

No pooling between convolutions: do not want to lose positional information

Padding to shift tokens: we need to prevent information flow from future tokens



condition on the previous tokens

Stack several convolutions

https://lena-voita.github.io/nlp_course/models/convolutional.html

Limitation of CNN-LM

- CNN-LM only has a fixed-length receptive field
  - probability of next token only dependent on a fixed-size context
- But sentences are of variable length
- How to handle sentences with variable length?
- Idea:
  - adding memory to network
  - adaptive updating memory

Recurrent Memory

- Introduce memory representation
- RNN-LM: use RNN to estimate

$$p\left(x_t \middle| x_1, \ldots, x_{t-1}\right) = \text{softmax}(W \cdot h_t)$$

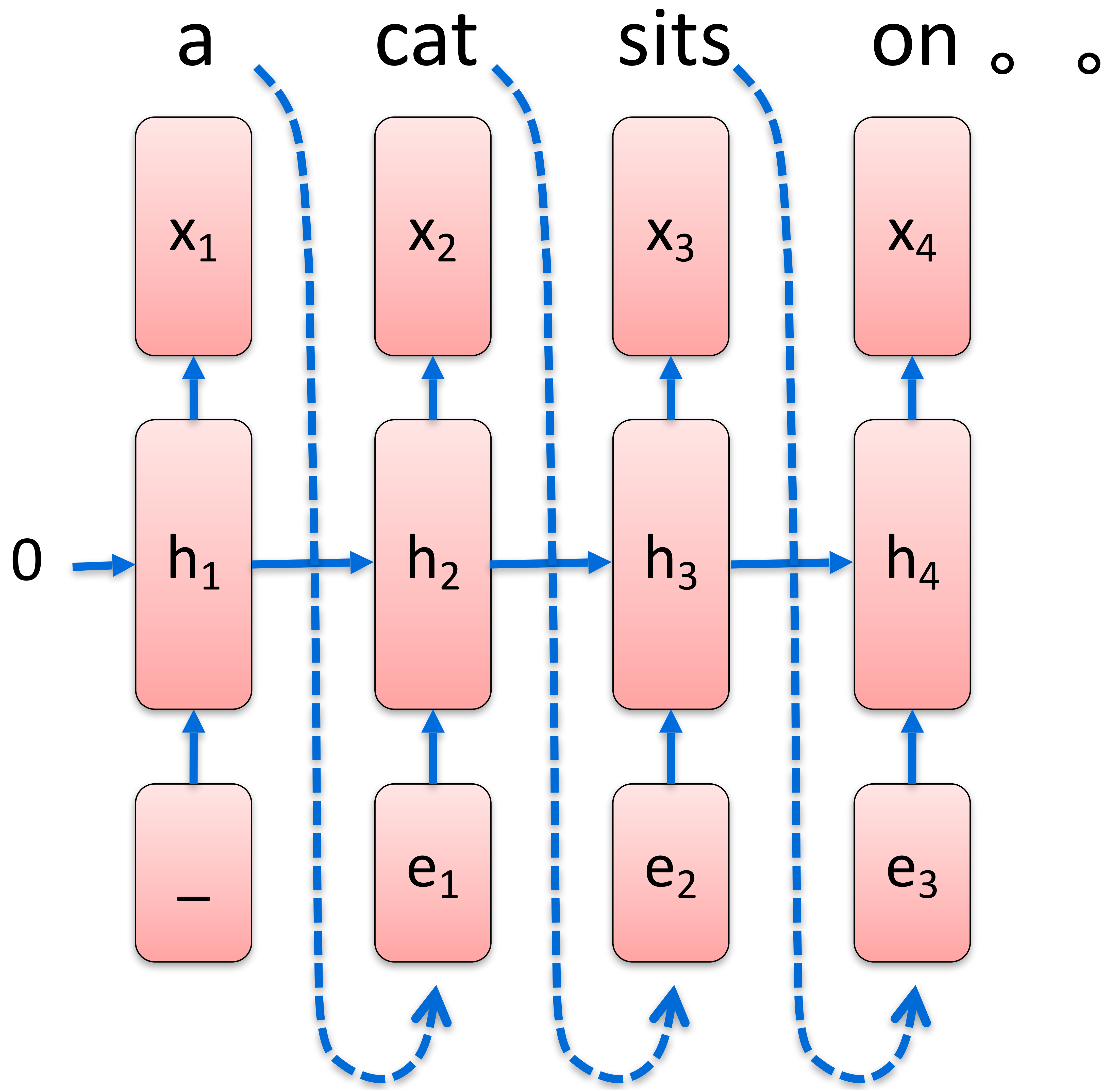
$$h_t = RNN(h_{t-1}, Emb(x_{t-1}))$$

- RNN cell can be
  - Simple feedforward neural network
  - Long-short term memory
  - Gated recurrent units

Recurrent Neural Network

$$p\left(x_t \middle| x_1, \ldots, x_{t-1}\right) = \mathrm{softmax}\left(U \cdot h_t\right)$$

$$h_t = \sigma\left(W \cdot \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b\right)$$



Elman, Finding Structure in Time. Cog. Sci. 1990.

Mikolov et al, Recurrent neural network based language model. Interspeech 2010.

Training RNN-LM

- Risk:
  - Loss: cross-entropy for every next-token given prefix context
  - CE(x_t+1, f(x_1, …, x_t))
- SGD
  - Calculate gradient: Back-propogation through time (BPTT)
  - ∇E_t

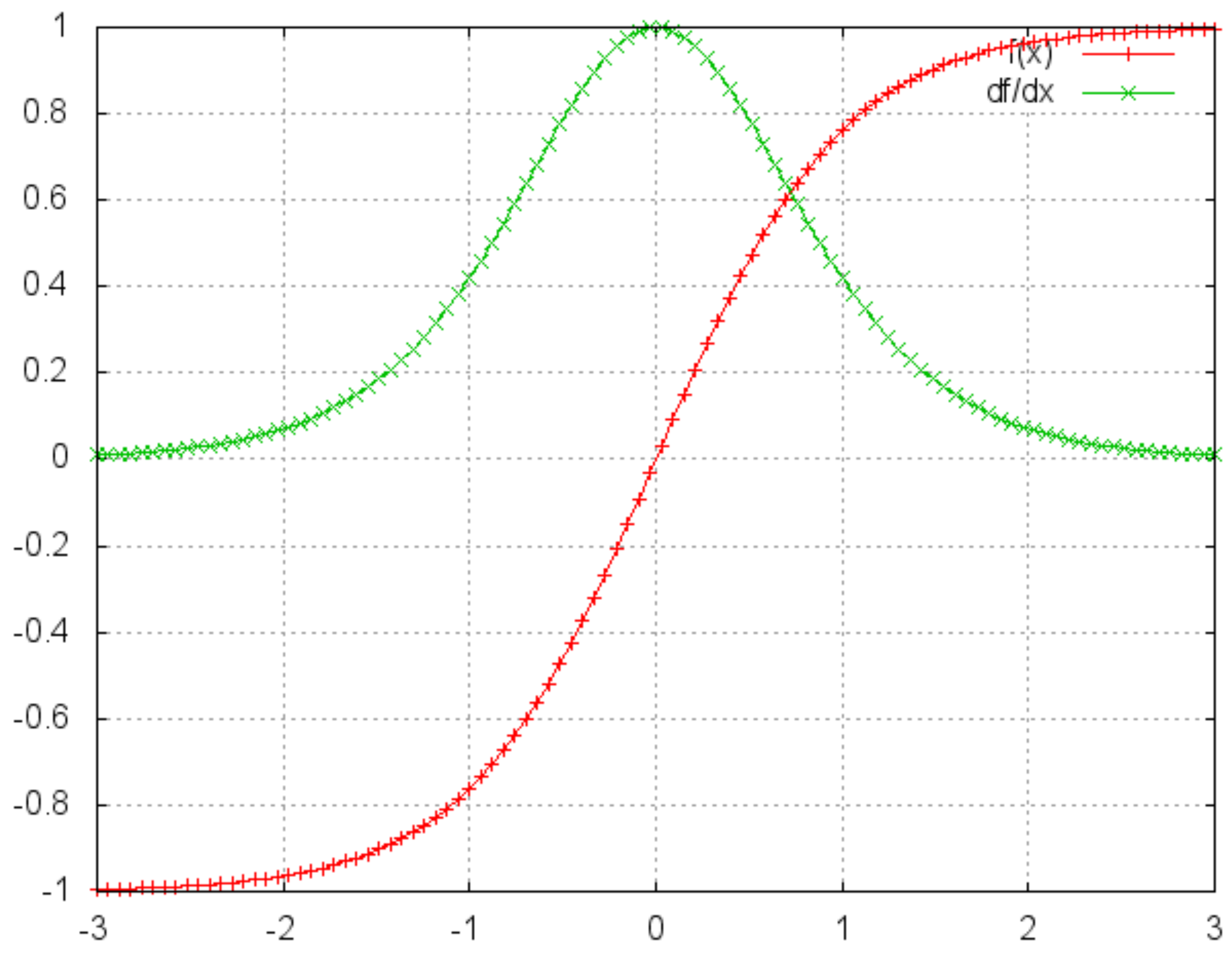
Exercise: Gradient for RNN

Back-propagation for RNN (python)

```
def bptt(self, x, y):
    T = len(y)
    # Perform forward propagation
    o, s = self.forward_propagation(x)
    # We accumulate the gradients in these variables
    dLdU = np.zeros(self.U.shape)
    dLdV = np.zeros(self.V.shape)
    dLdW = np.zeros(self.W.shape)
    delta_o = o
    delta_o[np.arange(len(y)), y] -= 1.
    # For each output backwards...
    for t in np.arange(T)[::-1]:
        dLdV += np.outer(delta_o[t], s[t].T)
        # Initial delta calculation: dL/dz
        delta_t = self.V.T.dot(delta_o[t]) * (1 - (s[t] ** 2))
        # Backpropagation through time (for at most self.bptt_truncate steps)
        for bptt_step in np.arange(max(0, t-self.bptt_truncate), t+1)[::-1]:
            # Add to gradients at each previous step
            dLdW += np.outer(delta_t, s[bptt_step-1])
            dLdU[:,x[bptt_step]] += delta_t
            # Update delta for next step dL/dz at t-1
            delta_t = self.W.T.dot(delta_t) * (1 - s[bptt_step-1] ** 2)
    return [dLdU, dLdV, dLdW]
```

Computational Issue: Gradient Vanishing

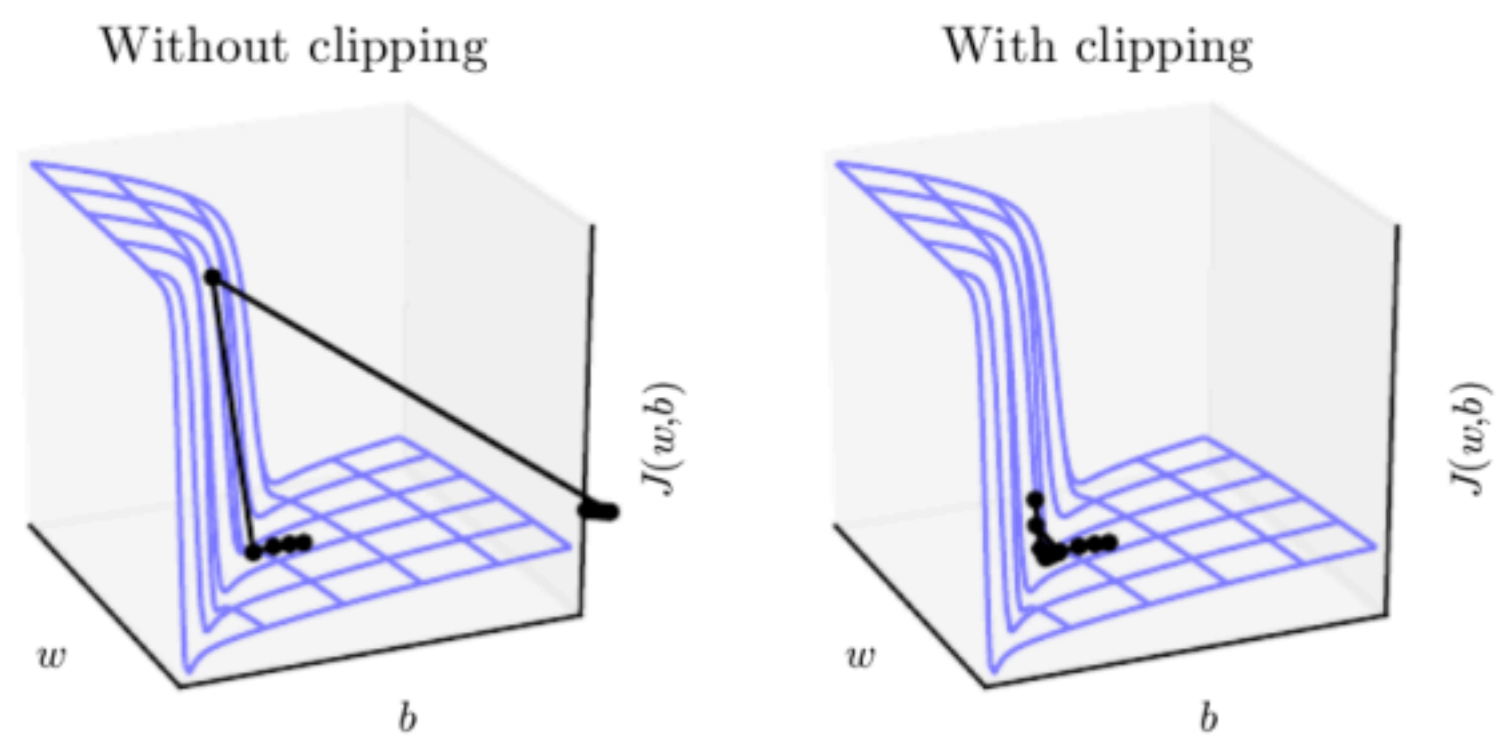
- tanh has derivative close to zero at both ends



Pascanu et al. On the difficulty of training recurrent neural networks. ICML 2013

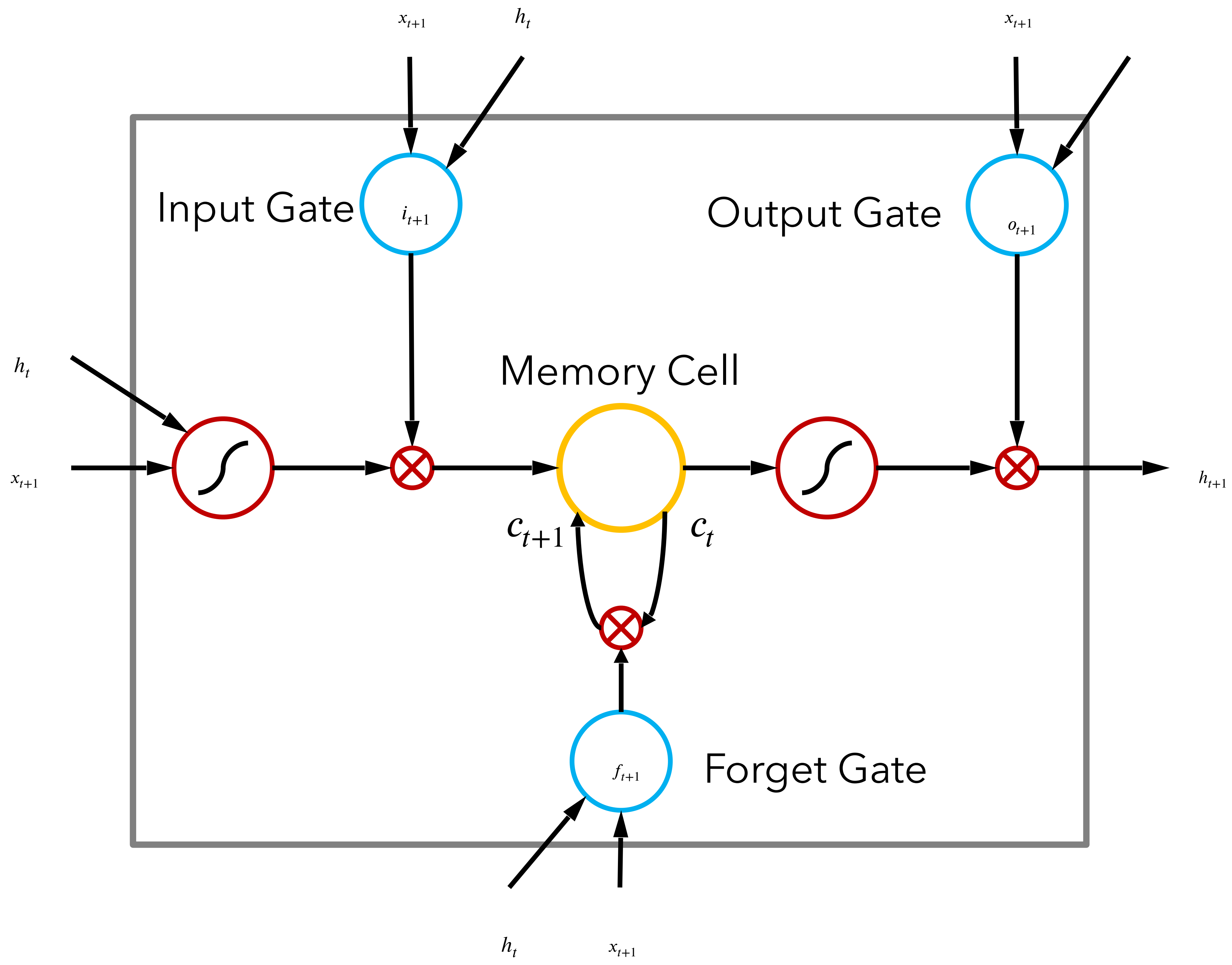
Gradient Exploding

- Use gradient clipping
- Two options: clip by absolute value or rescale norm
- if $|g| > \eta, \hat{g} \leftarrow \eta$
- if $|g| > \eta, \hat{g} \leftarrow \dfrac{\eta}{|g|} g$



Long-Short Term Memory (LSTM)

- Replace cell with more advanced one
- Adaptively memorize short and long term information



$$i_{t+1} = \sigma(M_{ix}x_{t+1} + M_{ih}h_t + b_i)$$
$$f_{t+1} = \sigma(M_{fx}x_{t+1} + M_{fh}h_t + b_f)$$
$$o_{t+1} = \sigma(M_{ox}x_{t+1} + M_{oh}h_t + b_o)$$

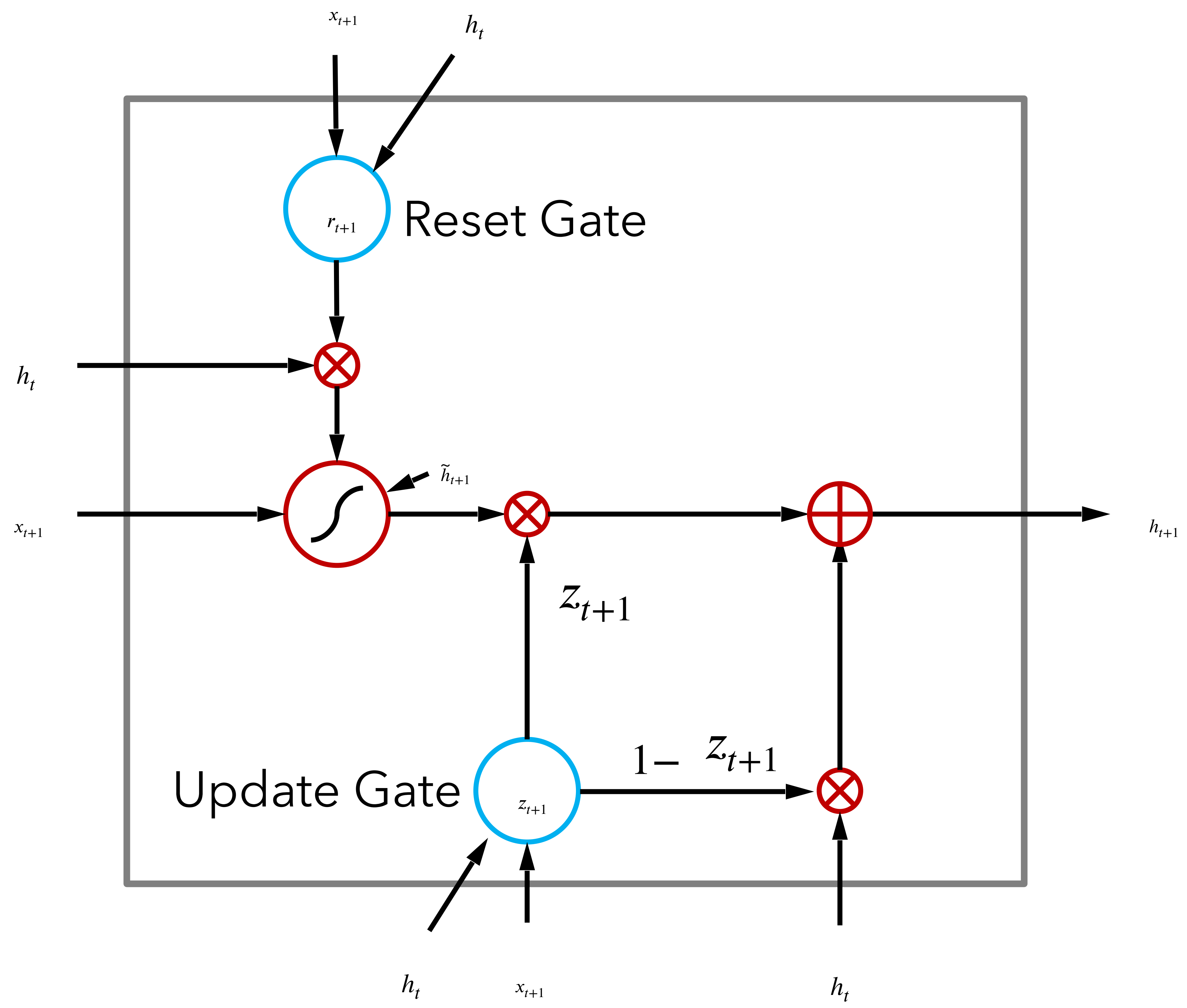
$$a_{t+1} = \tanh(M_{cx}x_{t+1} + M_{ch}h_t + b_a)$$

$$c_{t+1} = f_{t+1} \otimes c_t + i_{t+1} \otimes a_{t+1}$$
$$h_{t+1} = o_{t+1} \otimes \tanh(c_{t+1})$$

Hochreiter & Schmidhuber. Long Short-Term Memory, 1997
Gers et al. Learning to Forget: Continual Prediction with LSTM. 2000

Gated Recurrent Unit (GRU)

- Adaptively memorize short and long term information
- like LSTM, but fewer parameters



Input: x_t

Memory: h_t

$$r_{t+1} = \sigma(M_{rx}x_{t+1} + M_{rh}h_t + b_r)$$
$$z_{t+1} = \sigma(M_{zx}x_{t+1} + M_{zh}h_t + b_z)$$

$$\tilde{h}_{t+1} = \tanh(M_{hx}x_{t+1} + M_{hh}(r_{t+1} \otimes h_t) + b_h)$$

$$h_{t+1} = z_{t+1} \otimes \tilde{h}_{t+1} + (\mathbf{1} - \boldsymbol{z}_{t+1}) \otimes \boldsymbol{h}_t$$

Cho et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. 2014

Sequence Labelling using LSTM (Pytorch)

```
class LSTMTagger(nn.Module):

    def __init__(self, embedding_dim, hidden_dim, vocab_size, tagset_size):
        super(LSTMTagger, self).__init__()
        self.hidden_dim = hidden_dim

        self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)

        # The LSTM takes word embeddings as inputs, and outputs hidden states
        # with dimensionality hidden_dim.
        self.lstm = nn.LSTM(embedding_dim, hidden_dim)

        # The linear layer that maps from hidden state space to tag space
        self.hidden2tag = nn.Linear(hidden_dim, tagset_size)

    def forward(self, sentence):
        embeds = self.word_embeddings(sentence)
        lstm_out, _ = self.lstm(embeds.view(len(sentence), 1, -1))
        tag_space = self.hidden2tag(lstm_out.view(len(sentence), -1))
        tag_scores = F.log_softmax(tag_space, dim=1)
        return tag_scores
```

Training in Pytorch

```
model = LSTMTagger(EMBEDDING_DIM, HIDDEN_DIM, len(word_to_ix), len(tag_to_ix))
loss_function = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)

# See what the scores are before training
# Note that element i,j of the output is the score for tag j for word i.
# Here we don't need to train, so the code is wrapped in torch.no_grad()
with torch.no_grad():
    inputs = prepare_sequence(training_data[0][0], word_to_ix)
    tag_scores = model(inputs)
    print(tag_scores)

for epoch in range(300):  # again, normally you would NOT do 300 epochs, it is toy data
    for sentence, tags in training_data:
        # Step 1. Remember that Pytorch accumulates gradients.
        # We need to clear them out before each instance
        model.zero_grad()

        # Step 2. Get our inputs ready for the network, that is, turn them into
        # Tensors of word indices.
        sentence_in = prepare_sequence(sentence, word_to_ix)
        targets = prepare_sequence(tags, tag_to_ix)

        # Step 3. Run our forward pass.
        tag_scores = model(sentence_in)

        # Step 4. Compute the loss, gradients, and update the parameters by
        #  calling optimizer.step()
        loss = loss_function(tag_scores, targets)
        loss.backward()
        optimizer.step()
```

Testing in Pytorch

```
# See what the scores are after training
with torch.no_grad():
    inputs = prepare_sequence(training_data[0][0], word_to_ix)
    tag_scores = model(inputs)
```

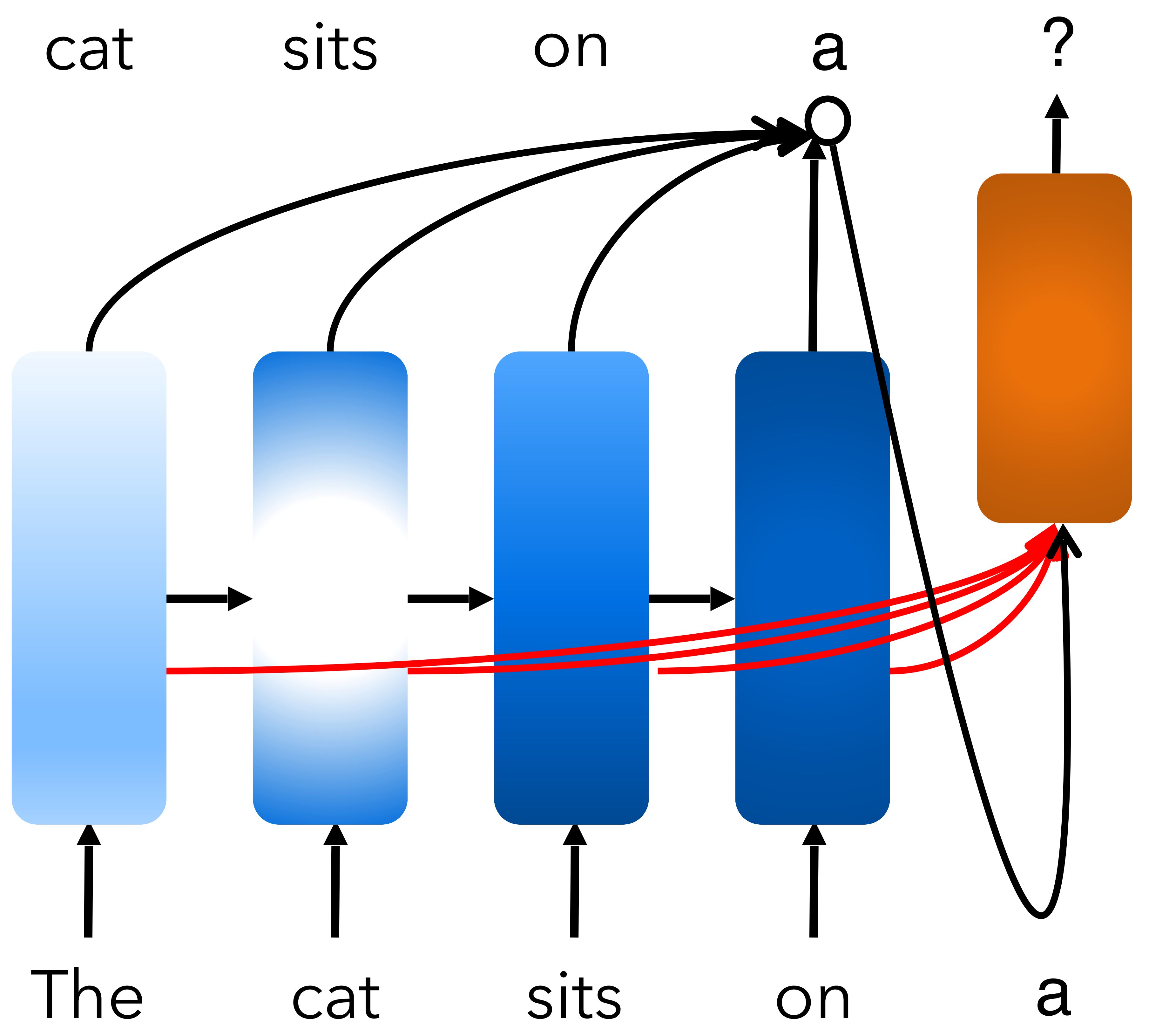
Expressive power of RNN-LM

Perplexity: $PPL = P(x_1, \ldots, x_N)^{-\frac{1}{N}} = \exp(-\frac{1}{N}\sum_{n=1}^{N} \log P(x_n \,|\, x_1 \ldots x_{n-1}))$

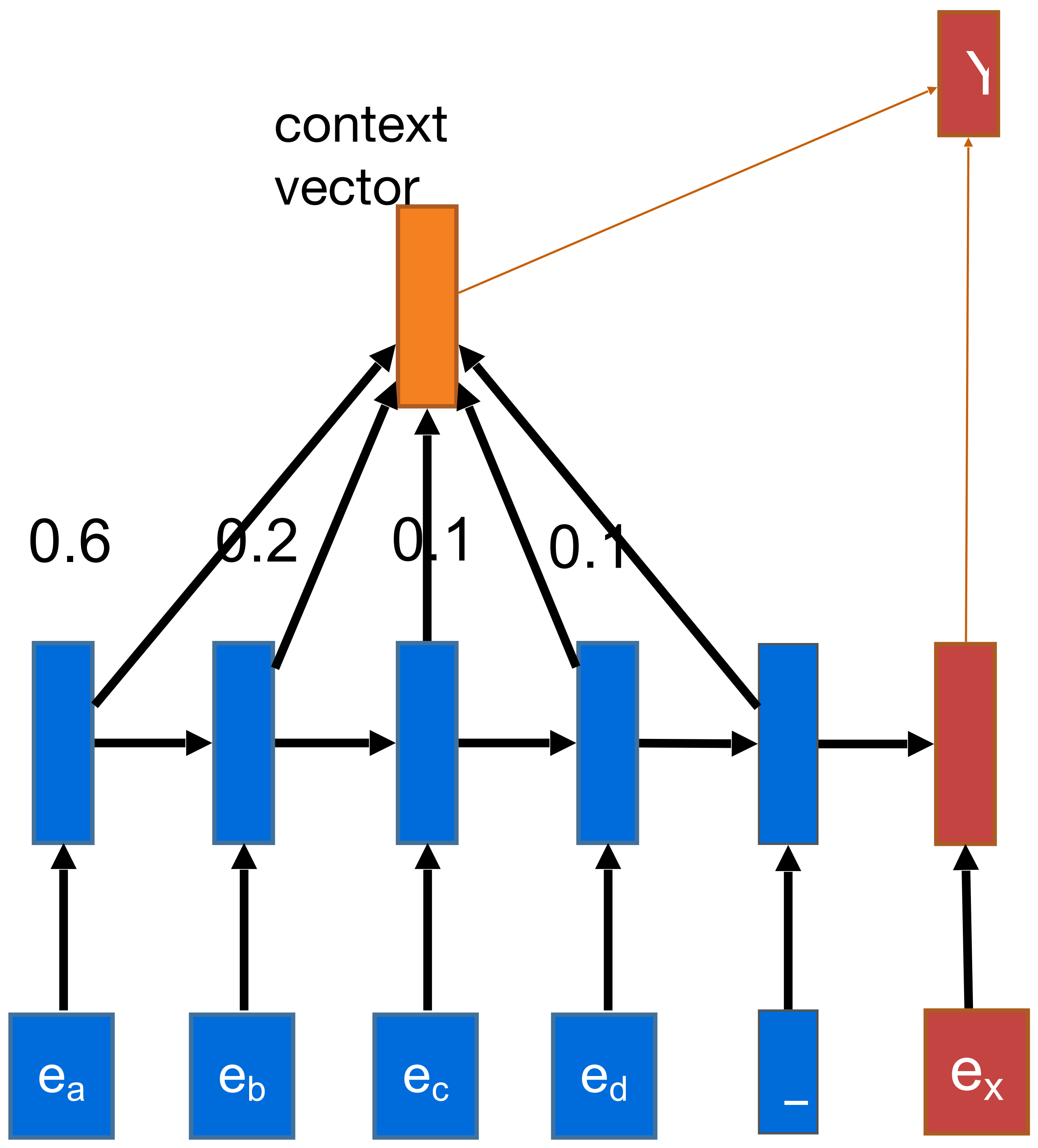
Model	Test Perplexity	Number of Params [billions]
Sigmoid-RNN-2048 (Ji et al., 2015a)	68.3	4.1
Interpolated KN 5-gram, 1.1B n-grams (Chelba et al., 2013)	67.6	1.76
Sparse Non-Negative Matrix LM (Shazeer et al., 2015)	52.9	33
RNN-1024 + MaxEnt 9-gram features (Chelba et al., 2013)	51.3	20
LSTM-512-512	54.1	0.82
LSTM-1024-512	48.2	0.82
LSTM-2048-512	43.7	0.83
LSTM-8192-2048 (No Dropout)	37.9	3.3
LSTM-8192-2048 (50% Dropout)	32.2	3.3
2-Layer LSTM-8192-1024 (BIG LSTM)	30.6	1.8
BIG LSTM+CNN Inputs	**30.0**	**1.04**
BIG LSTM+CNN Inputs + CNN Softmax	39.8	**0.29**
BIG LSTM+CNN Inputs + CNN Softmax + 128-dim correction	35.8	**0.39**
BIG LSTM+CNN Inputs + Char LSTM predictions	47.9	**0.23**

Jozefowicz et al. Exploring the limits of language modelling, 2016

Attention



Generation by Attention



A context vector c will be predicted before, which represents the related source context for current predicted word.

$$\alpha_{nj} = \mathrm{Softmax}(D(s_n, h_{1\ldots n-1})) = \frac{\exp(D(s_n, h_j))}{\sum_k \exp(D(s_n, h_k))}$$

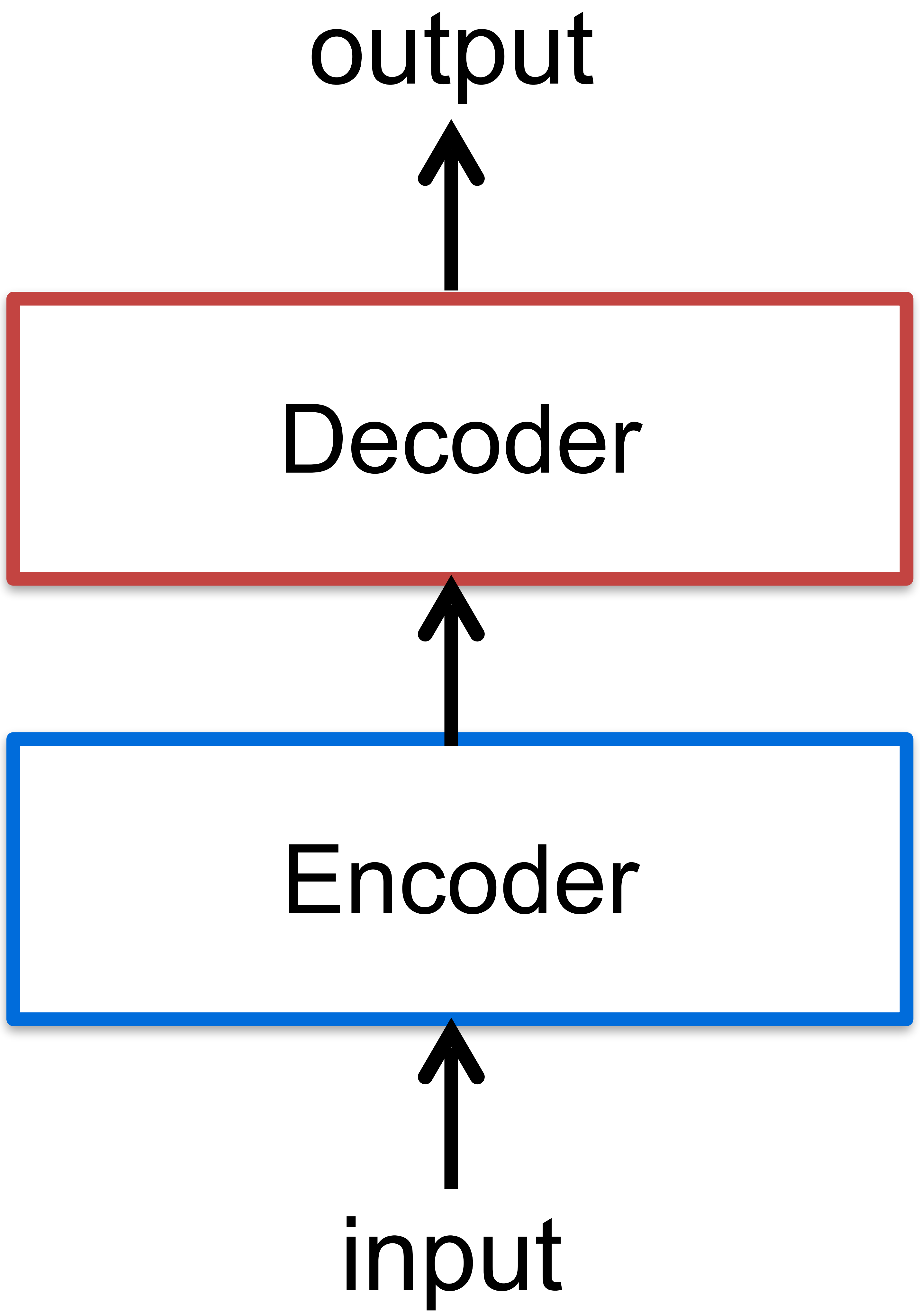
$$c_n = \sum_j \alpha_{nj} h_j$$

The probability of word y_i is computed as:

$$p(y_i) \propto \exp(Wh_i) \Rightarrow p(y_i) \propto \exp(Wh_i + Vc_i)$$

Mnih et al. Recurrent Models of Visual Attention. 2014.

Encoder-decoder framework



A generic formulation

ImageCaption

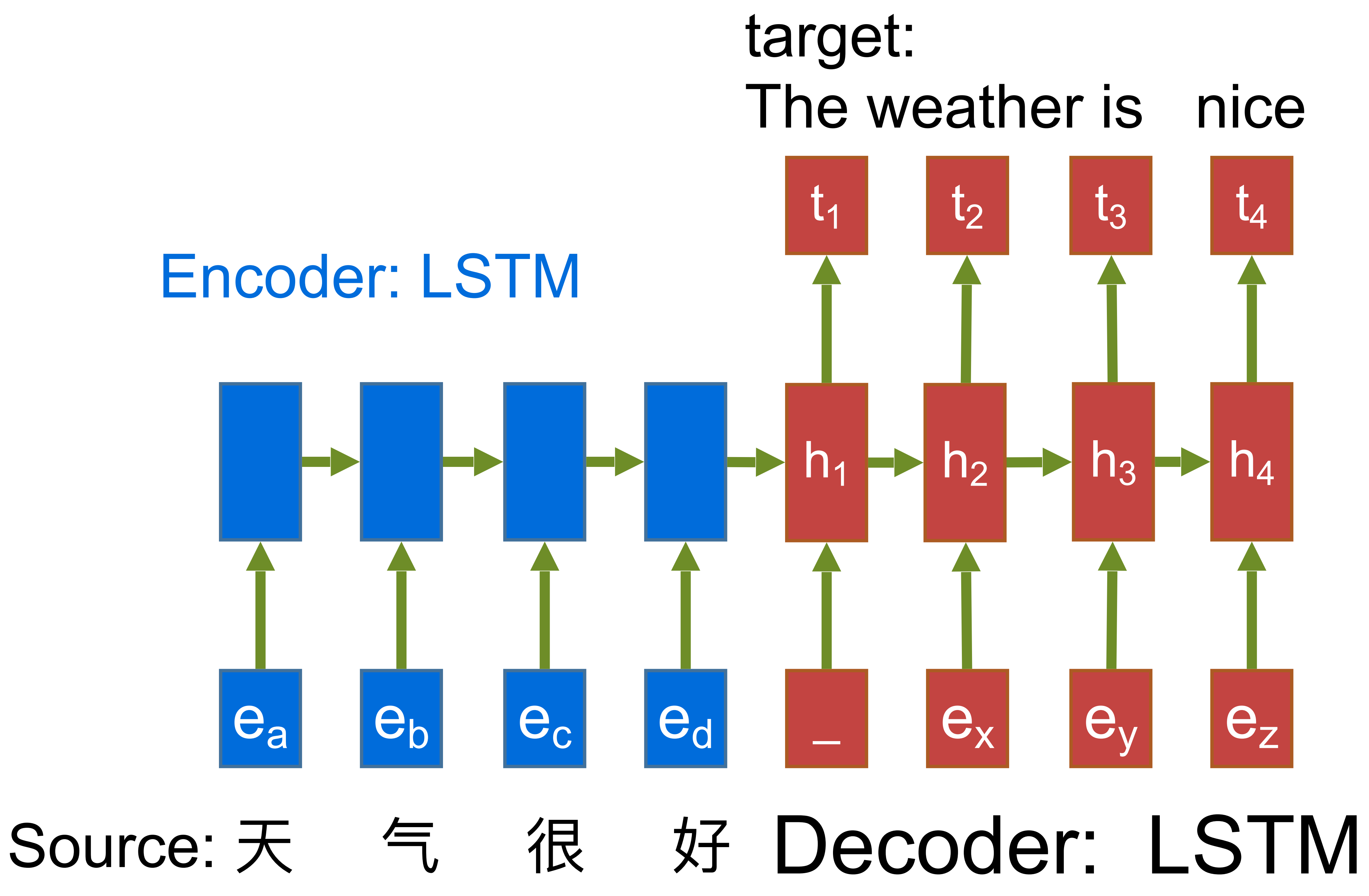
Text-to-Image Generation

ASR (speech-to-text)

MT (text-to-text)

Sequence To Sequence (Seq2seq)

- Machine translation as directly learning a function mapping from source sequence to target sequence



$$P(Y|X) = \prod P(y_t | y_{<t}, x)$$

Training loss: Cross-Entropy

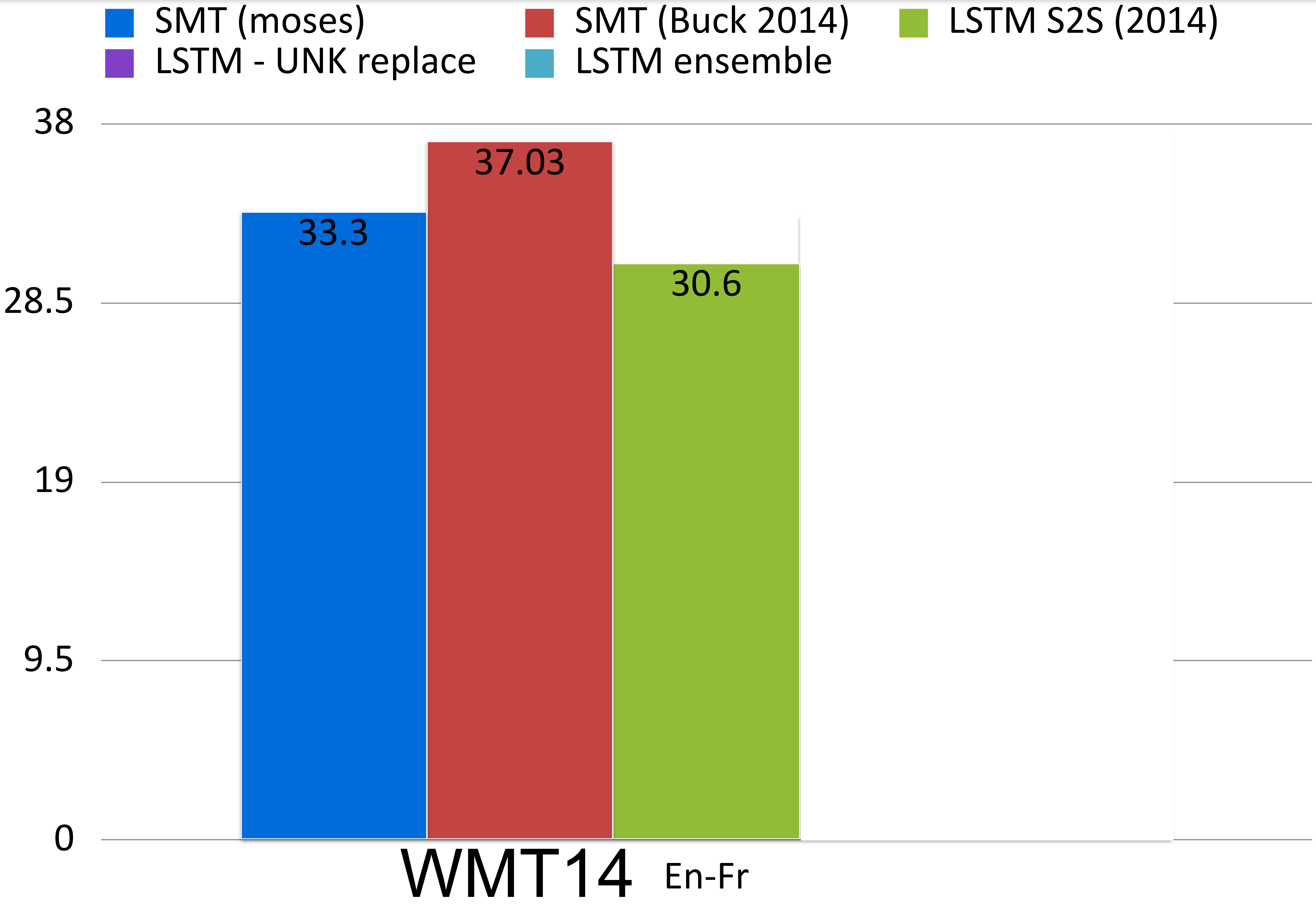
$$l = -\sum_n \sum_t \log f_\theta(x_n, y_{n,1}, \ldots, y_{n,t-1})$$

Teacher-forcing during training.

(pretend to know groundtruth for prefix)

Sutskever et al. Sequence to Sequence Learning with Neural Networks. 2014

Performance (2014)

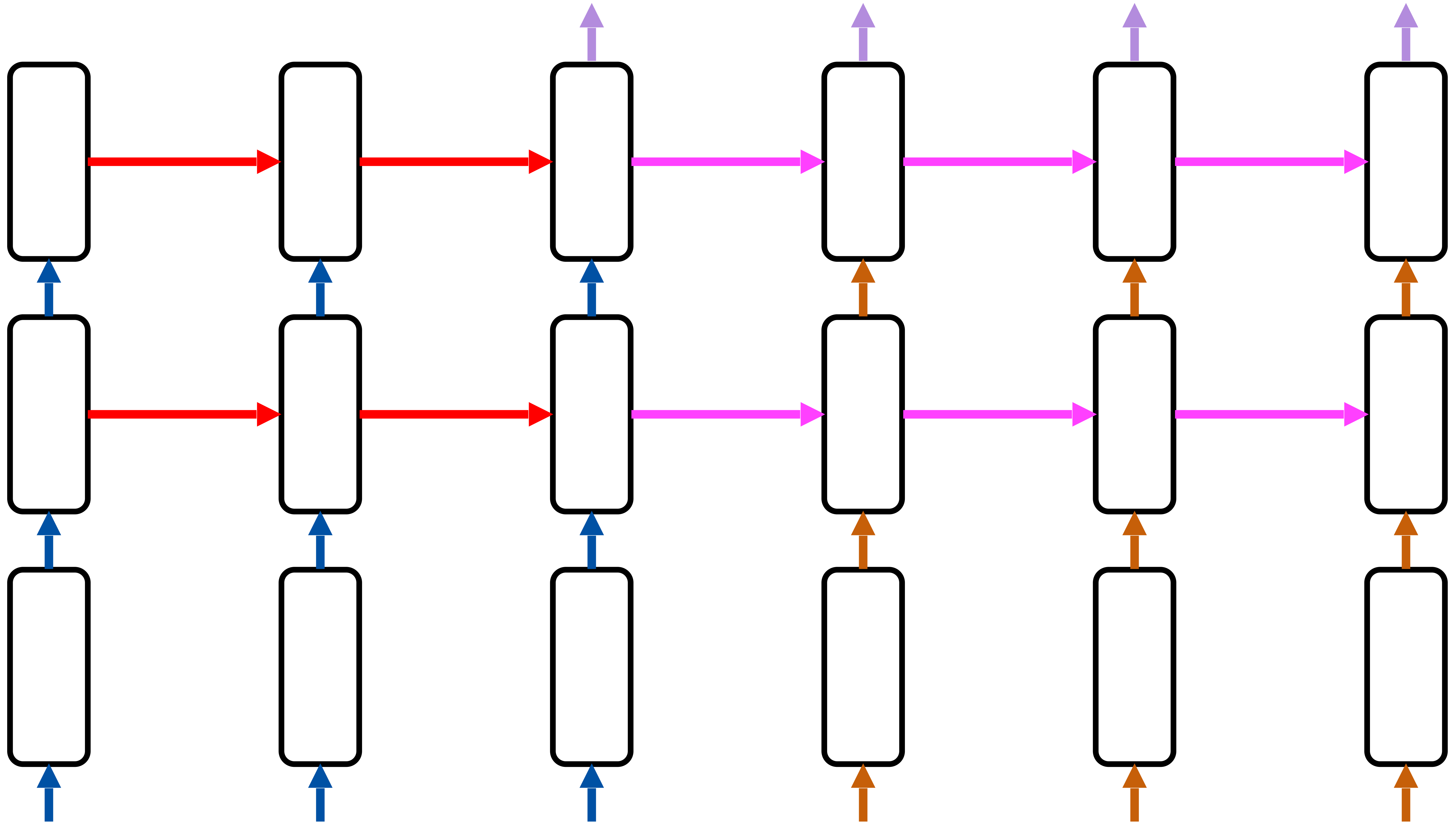


Sutskever et al. Sequence to Sequence Learning with Neural Networks. 2014

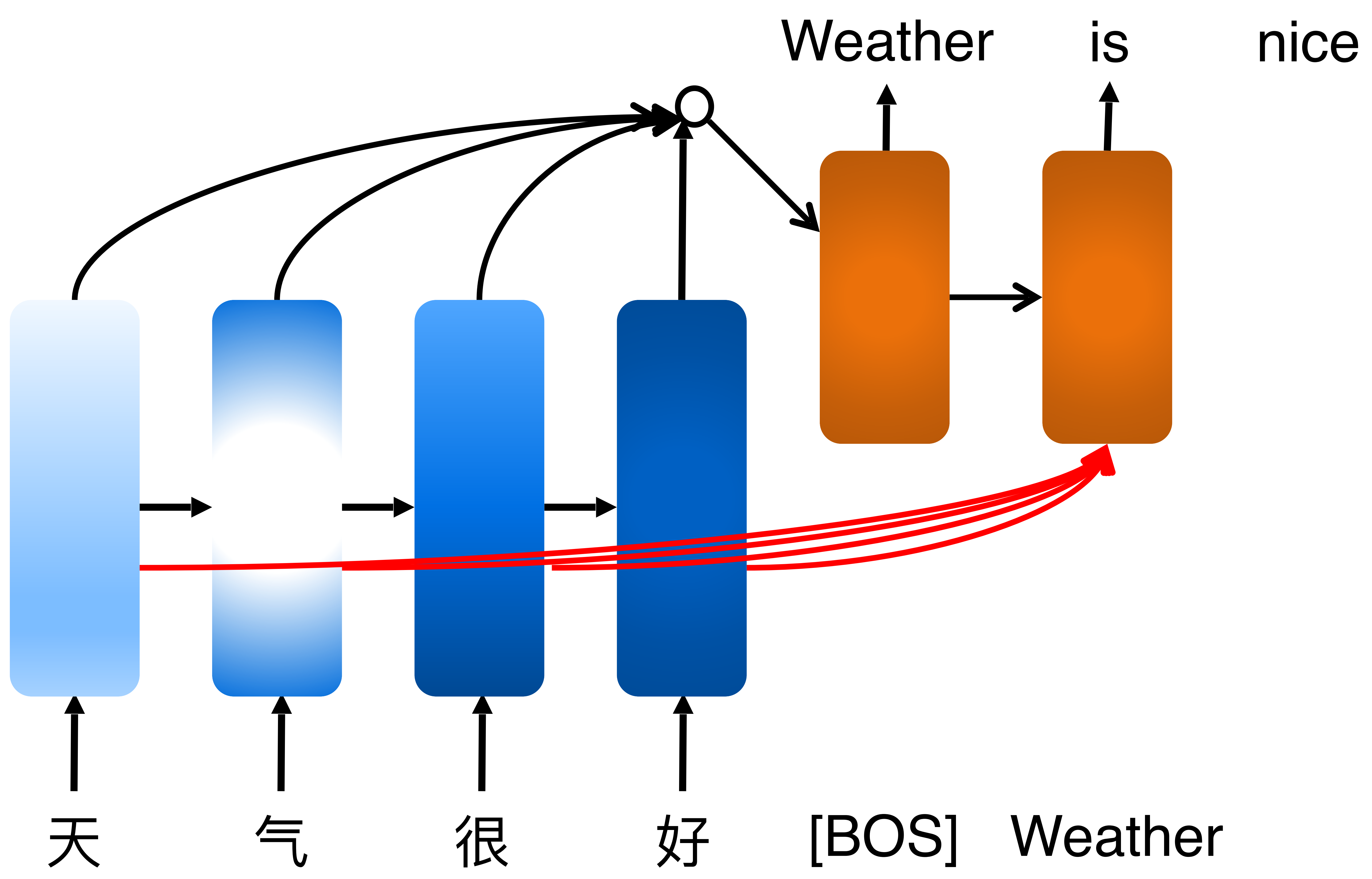
Durrani et al. Edinburgh's Phrase-based Machine Translation Systems for WMT-14. 2014

Stacked LSTM for seq-2-seq

- More layers of LSTM

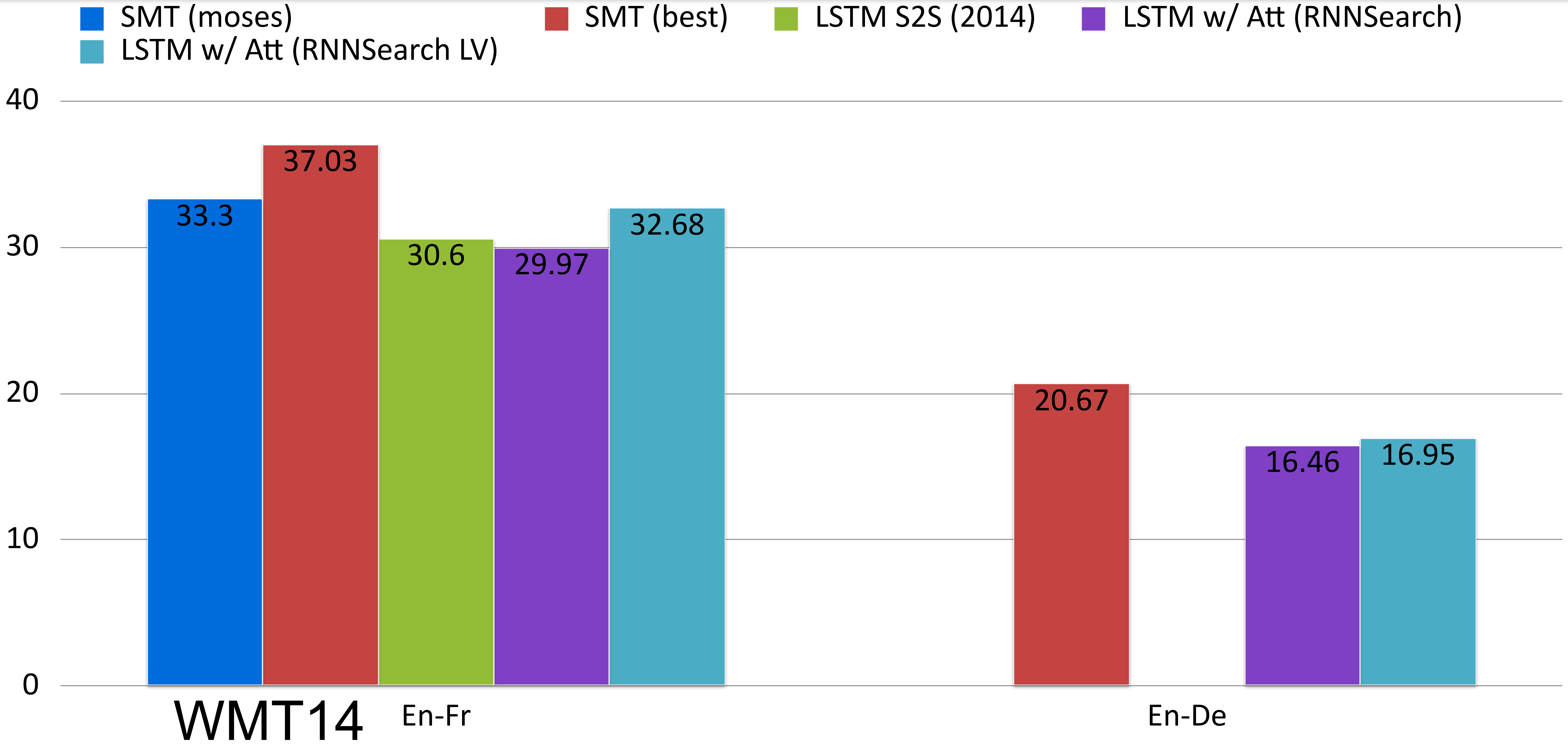


LSTM Seq2seq with Attention



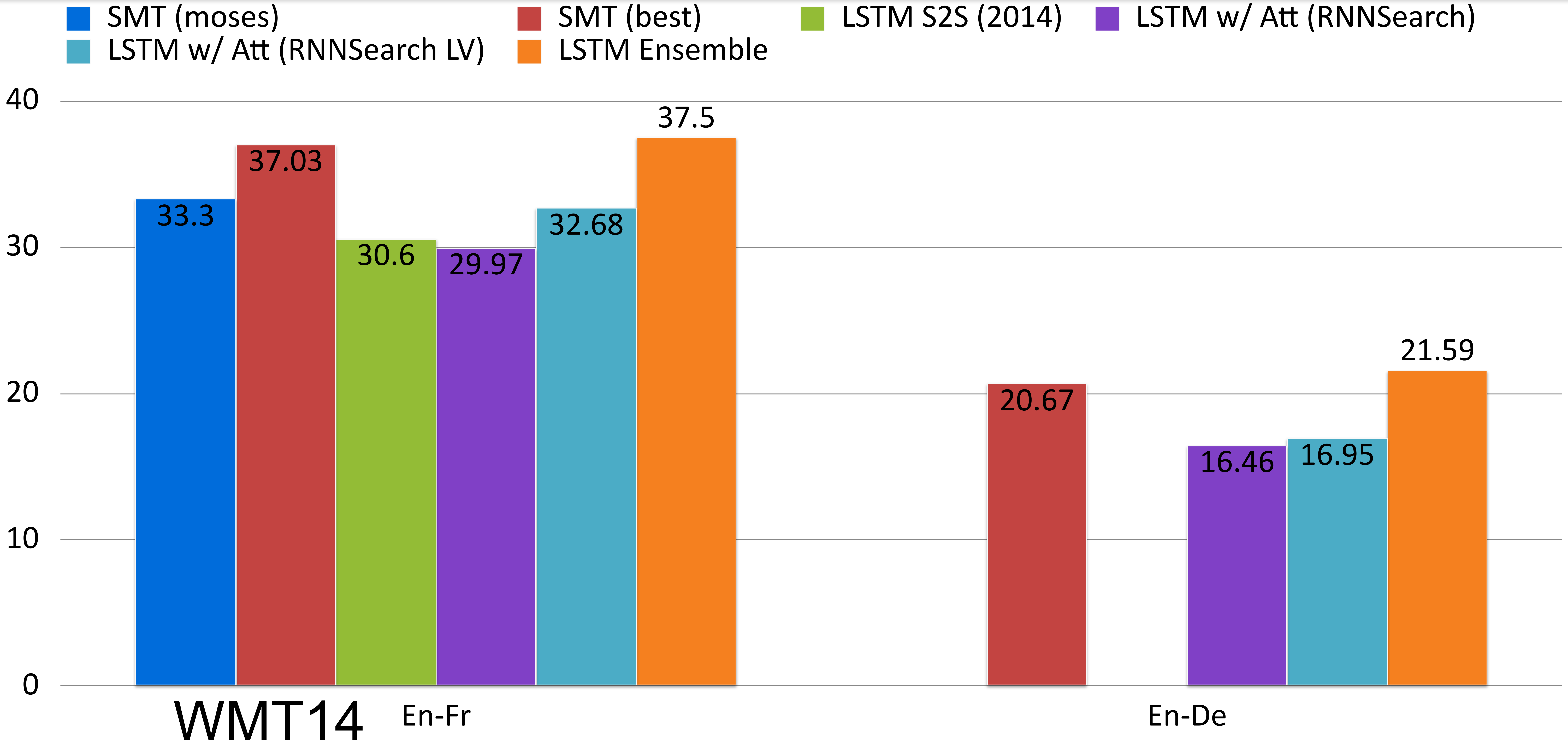
Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate. 2015

LSTM Seq2Seq w/ Attention



Jean et al. On Using Very Large Target Vocabulary for Neural Machine Translation. 2015

Performance with Model Ensemble



Luong et al. Effective Approaches to Attention-based Neural Machine Translation. 2015

Reading

- Gers et al. Learning to Forget: Continual Prediction with LSTM. 2000
- Sutskever et al. Sequence to Sequence Learning with Neural Networks. 2014
- Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate. 2015
- Luong et al. Effective Approaches to Attention-based Neural Machine Translation. 2015