COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

# Neural sequence-to-sequence models for machine translation

**CMSC 470**

Marine Carpuat

# Machine Translation

- Translation system
  - Input: source sentence F
  - Output: target sentence E
  - Can be viewed as a function

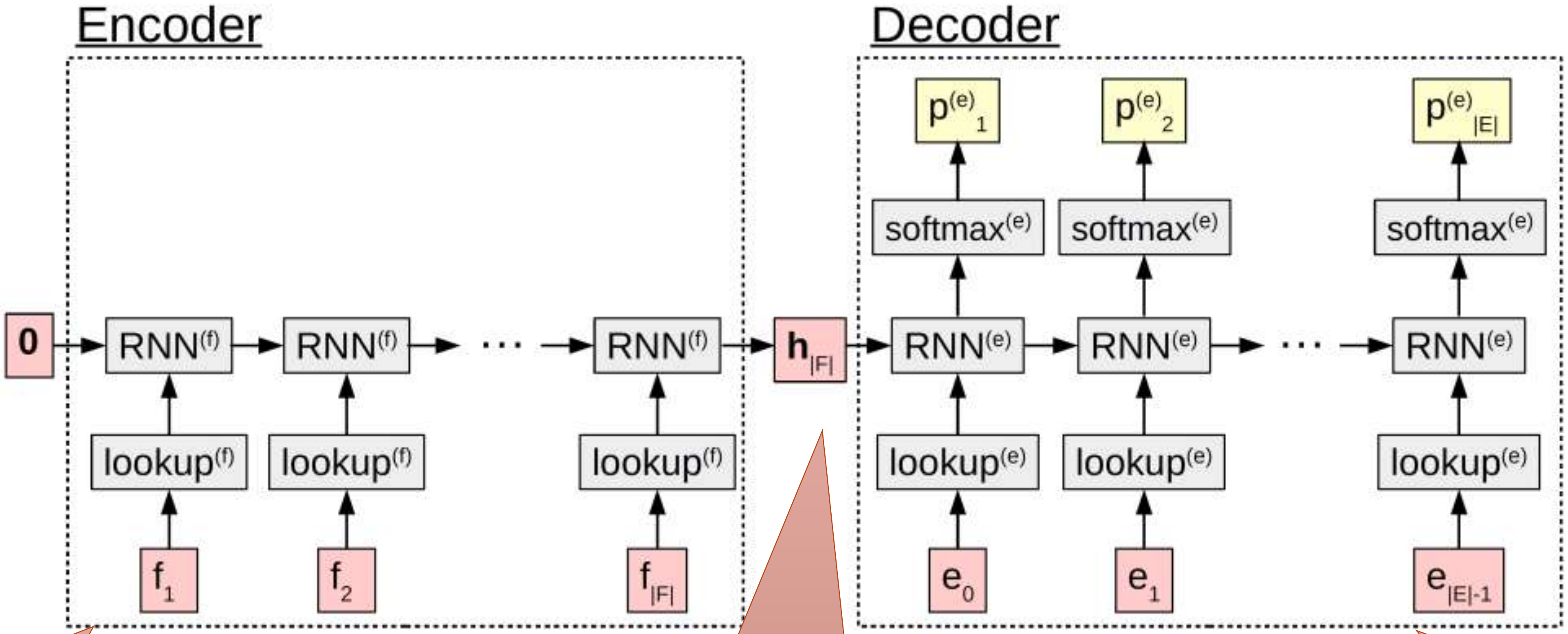
$$\hat{E} = \mathrm{mt}(F)$$

- Modern machine translation systems

$$\hat{E} = \underset{E}{\operatorname{argmax}} \; P(E \mid F; \theta)$$

- 3 problems
  - Modeling
    - how to define P(.)?
  - Training/Learning
    - how to estimate parameters from parallel corpora?
  - Search
    - How to solve argmax efficiently?

# P(E|F) as an encoder-decoder model




The Encoder models the input/source entence F=(f1,….f|F|)

The decoder hidden state is initialized with the last hidden state of the encoder

The Decoder models the output/target sentence E=(e1,….e|E|).

# Neural Machine Translation

- Neural language models review
- Sequence to sequence models for MT
  - Encoder-Decoder
  - Sampling and search (greedy vs beam search)
  - **How to train?**
  - **Model variants and practical tricks**
  - **Attention mechanism**

# Training

- Same as for RNN language modeling
  - Intuition: a good model assigns high probability to training examples
- Loss function
  - Negative log-likelihood of training data
    - Also called cross-entropy loss
  - Total loss for one example (sentence pair) = sum of loss at each time step (word)
- Backpropagation
  - Gradient of loss at time step t is propagated through the network all the way back to $1^{st}$ time step

# Aside: why don't we use BLEU as training loss?

N-gram overlap between machine translation output and reference translation

Compute precision for n-grams of size 1 to 4

Add brevity penalty (for too short translations)

$$\text{BLEU} = \min\left(1, \frac{\textit{output-length}}{\textit{reference-length}}\right) \left(\prod_{i=1}^{4} \textit{precision}_i\right)^{\frac{1}{4}}$$

Typically computed over the entire corpus, not single sentences

# Training in practice: online

**Algorithm 1** A fully online training algorithm

```
procedure ONLINE
    for several epochs of training do
        for each training example in the data do
            Calculate gradients of the loss
            Update the parameters according to this gradient
        end for
    end for
end procedure
```

# Training in practice: batch

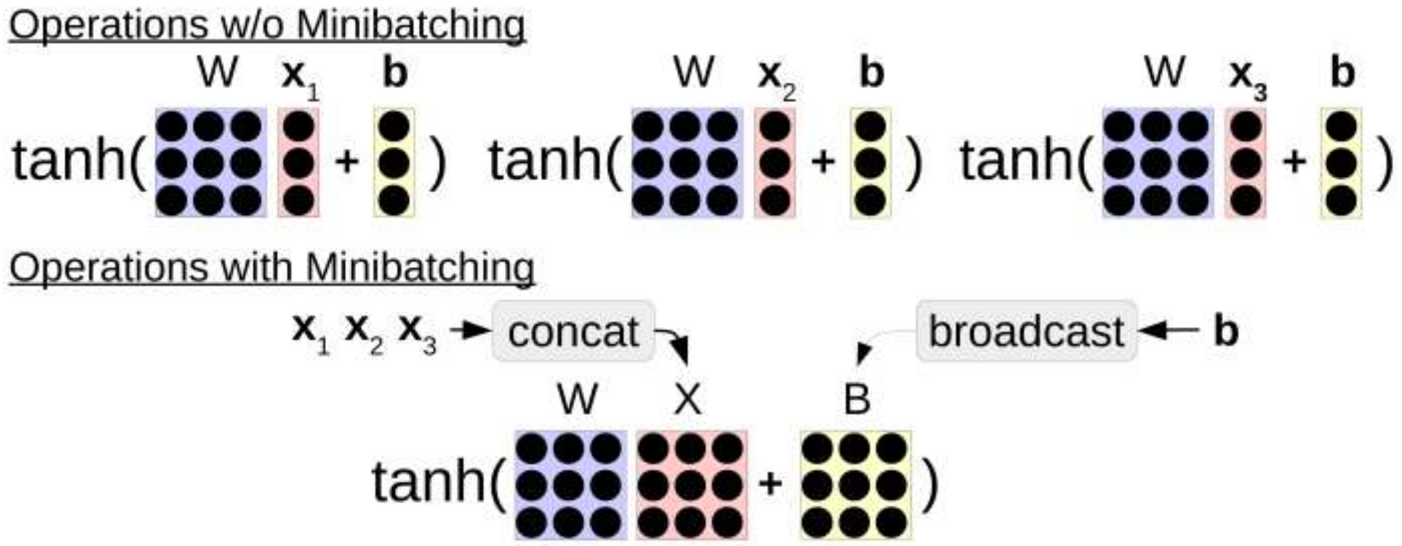
**Algorithm 2** A batch learning algorithm

```
procedure BATCH
    for several epochs of training do
        for each training example in the data do
            Calculate and accumulate gradients of the loss
        end for
        Update the parameters according to the accumulated gradient
    end for
end procedure
```

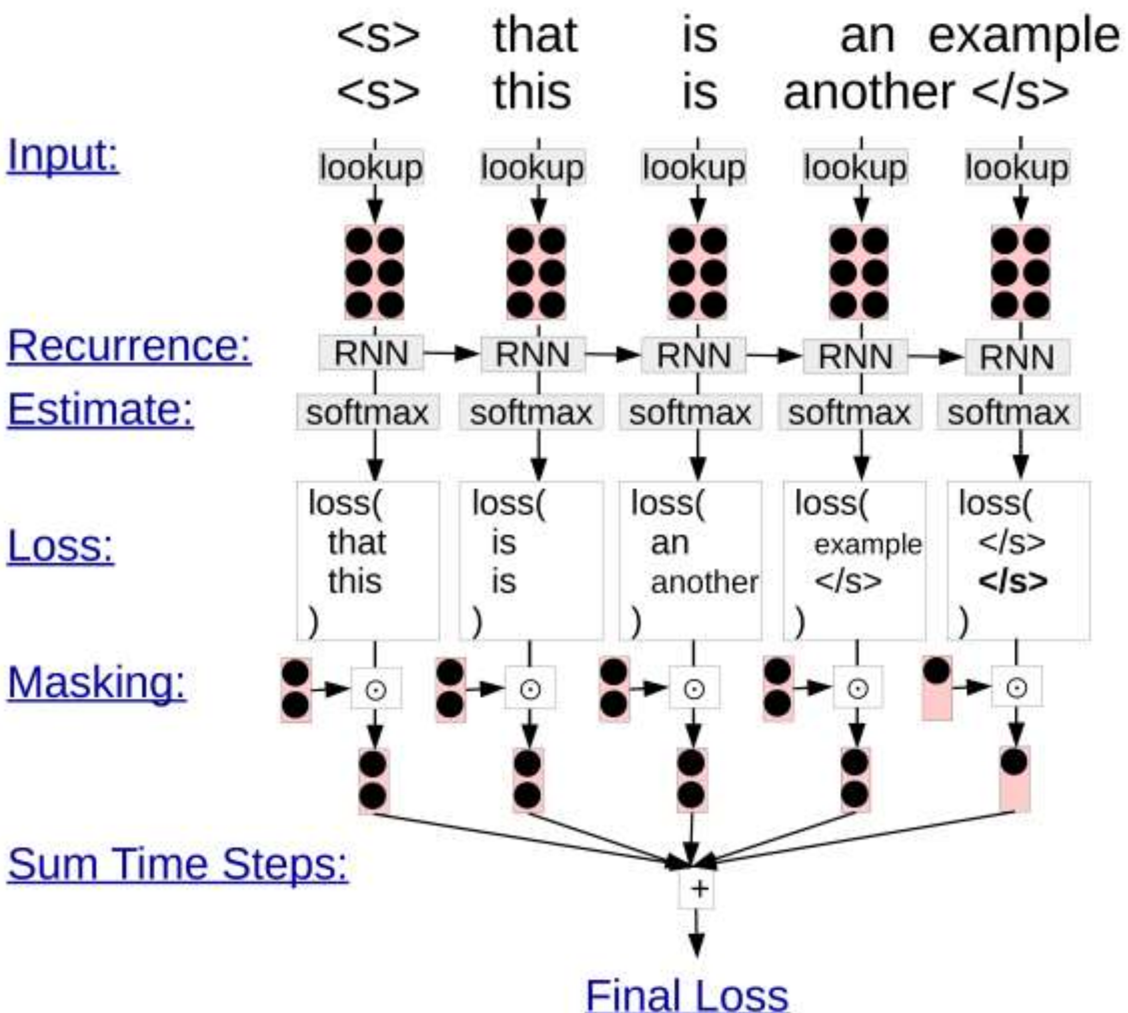
# Training in practice: minibatch

- Compromise between online and batch

- Computational advantages
  - Can leverage vector processing instructions in modern hardware
  - By processing multiple examples simultaneously



# Problem with minibatches: examples have varying length



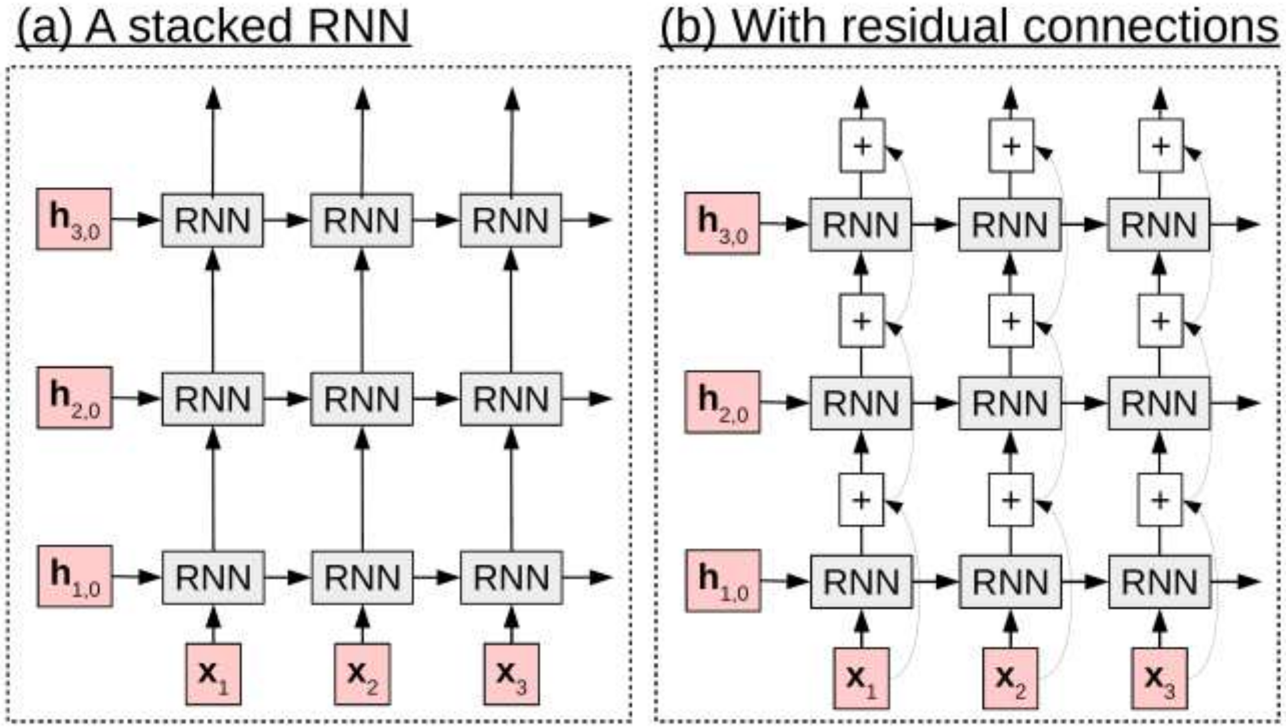

- 3 tricks (same as for language modeling)
  - Padding
    - Add </s> symbol to make all sentences same length
  - Masking
    - Multiply loss function calculated over padded symbols by zero
  - + sort sentences by length

# Neural Machine Translation

- Neural language models review
- Sequence to sequence models for MT
  - Encoder-Decoder
  - Sampling and search (greedy vs beam search)
  - **How to train?**
  - **Model variants and practical tricks**
  - **Attention mechanism**

# Other encoder structures: RNN variants

- LSTMs
  - Aim to address vanishing/exploding gradient issue
- Stacked RNNs




- …

# Other encoder structures: Bidirectional encoder

$$\overrightarrow{\boldsymbol{h}}^{(f)}_t = \begin{cases} \overrightarrow{\mathrm{RNN}}^{(f)}(\boldsymbol{m}^{(f)}_t, \overrightarrow{\boldsymbol{h}}^{(f)}_{t+-}) & t \geq 1, \\ \boldsymbol{0} & \text{otherwise.} \end{cases}$$

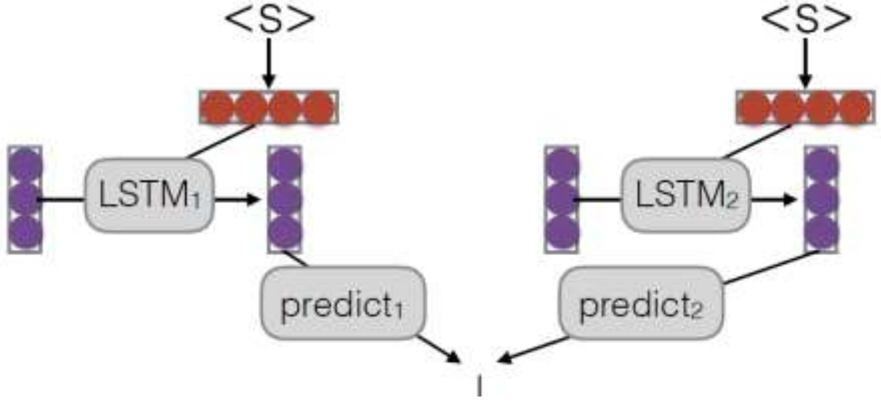
$$\overleftarrow{\boldsymbol{h}}^{(f)}_t = \begin{cases} \overleftarrow{\mathrm{RNN}}^{(f)}(\boldsymbol{m}^{(f)}_t, \overleftarrow{\boldsymbol{h}}^{(f)}_{t+1}) & t \leq |F|, \\ \boldsymbol{0} & \text{otherwise.} \end{cases}$$

Motivation:
- Help bootstrap learning
- By shortening length of dependencies

$$\boldsymbol{h}^{(e)}_0 = \tanh(W_{\overrightarrow{f}e}\overrightarrow{\boldsymbol{h}}_{|F|} + W_{\overleftarrow{f}e}\overleftarrow{\boldsymbol{h}}_1 + \boldsymbol{b}_e)$$

Motivation:
- Take 2 hidden vectors from source encoder
- Combine them into a vector of size required by decoder

# A few more tricks: ensembling




- Combine predictions from multiple models

- Methods
  - Linear or log-linear interpolation
  - Parameter averaging

# Tricks: addressing length bias

- Default models tend to generate short sentences
- Solutions:
  - Prior probability on sentence length

$$\hat{E} = \underset{E}{\operatorname{argmax}} \ \log P(|E| \mid |F|) + \log P(E \mid F).$$

  - Normalize by sentence length

$$\hat{E} = \underset{E}{\operatorname{argmax}} \ \log P(E \mid F)/|E|.$$

# Neural Machine Translation

- Neural language models review
- Sequence to sequence models for MT
  - Encoder-Decoder
  - Sampling and search (greedy vs beam search)
  - **How to train?**
  - **Model variants and practical tricks**
  - **Attention mechanism**