Neural Machine Translation

Dan Klein, John DeNero
UC Berkeley

Attention

Conditional Sequence Generation

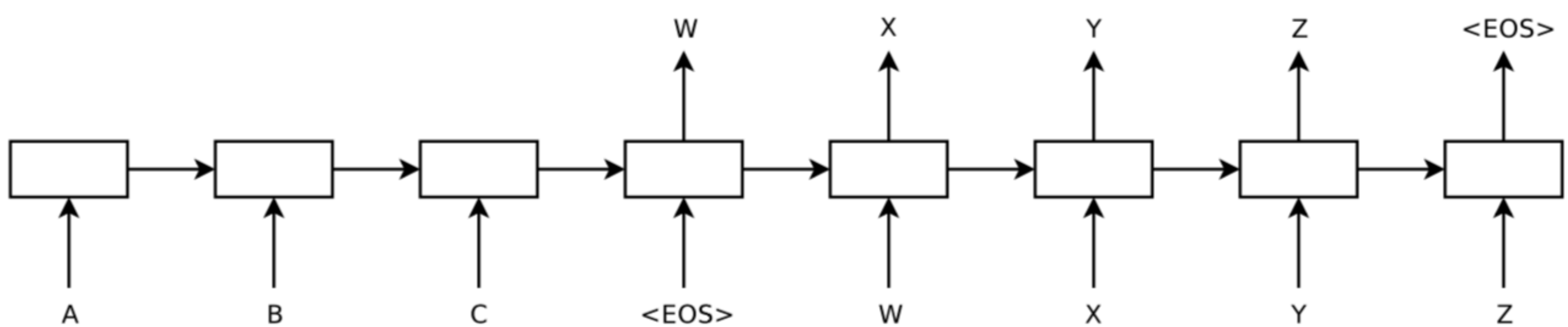
P(e|f) could just be estimated from a sequence model P(f, e)

<f> das Haus ist klein </f>	the house is small </e>

Run an RNN over the whole sequence, which first computes P(f), then computes P(e, f).

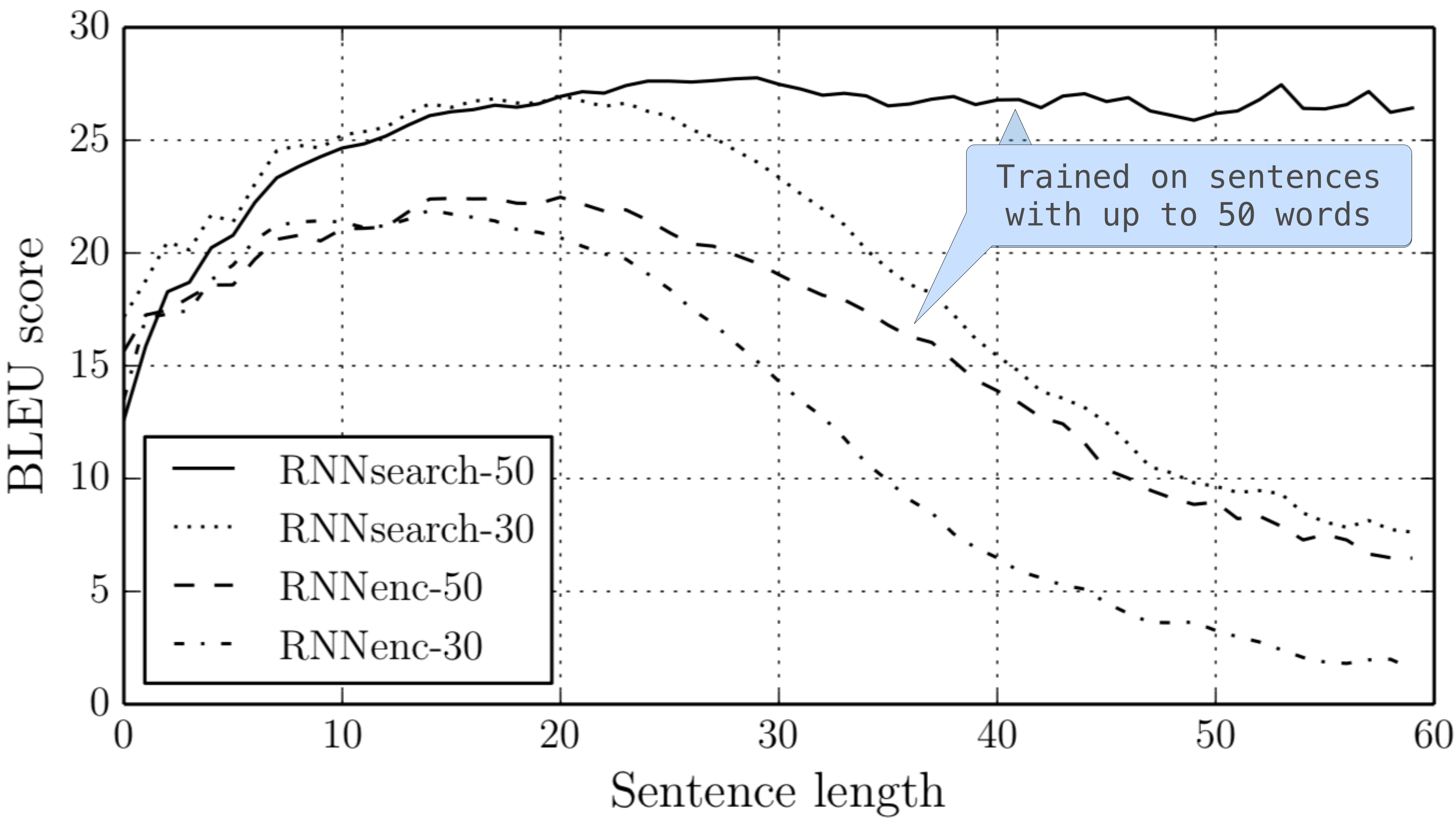
Encoder-Decoder: Use different parameters or architectures encoding f and predicting e.

"Sequence to sequence" learning (Sutskever et al., 2014)



(Sutskever et al., 2014) Sequence to sequence learning with neural networks.

Impact of Attention on Long Sequence Generation



(Badhanau et al., 2016) Neural Machine Translation by Jointly Learning to Align and Translate

Conditional Gated Recurrent Unit with Attention

$$\mathbf{s}_j = \mathrm{cGRU}_{\mathrm{att}}\left(\mathbf{s}_{j-1}, y_{j-1}, \mathrm{C}\right)$$

Architecture for the top research system in WMT16 and WMT17 (Univ. Edinburgh)

$$\mathbf{s}'_j = (1 - \mathbf{z}'_j) \odot \underline{\mathbf{s}}'_j + \mathbf{z}'_j \odot \mathbf{s}_{j-1}$$

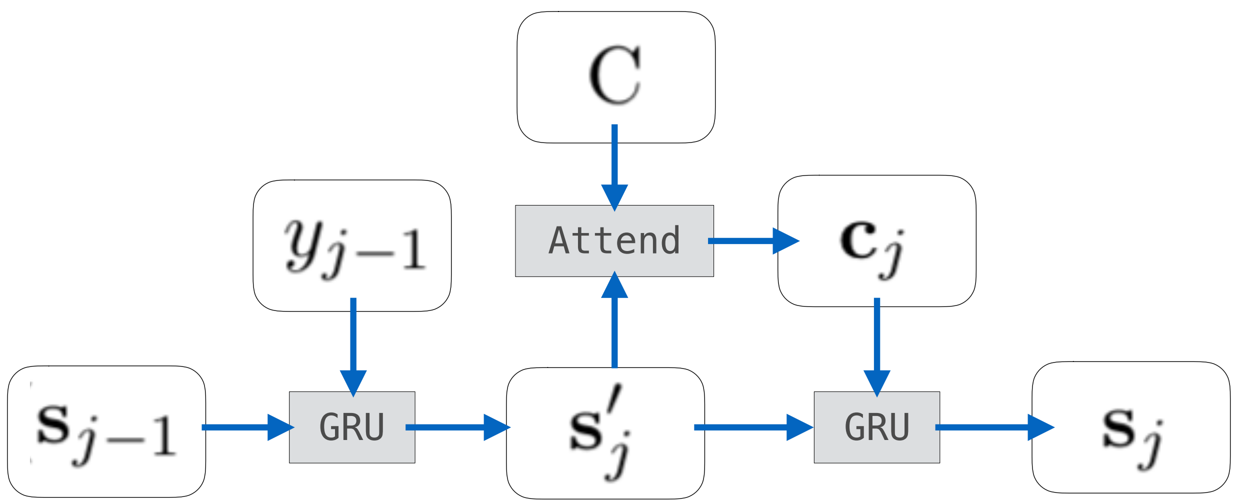
$$\underline{\mathbf{s}}'_j = \tanh\left(\mathbf{W}'\mathbf{E}[y_{j-1}] + \mathbf{r}'_j \odot (\mathbf{U}'\mathbf{s}_{j-1})\right),$$

$$\mathbf{r}'_j = \sigma\left(\mathbf{W}'_r\mathbf{E}[y_{j-1}] + \mathbf{U}'_r\mathbf{s}_{j-1}\right),$$

$$\mathbf{z}'_j = \sigma\left(\mathbf{W}'_z\mathbf{E}[y_{j-1}] + \mathbf{U}'_z\mathbf{s}_{j-1}\right),$$

Reset gate masks the previous state's projection within the nonlinear forward step

Update gate mixes the output of the forward step with the previous state



Conditional Gated Recurrent Unit with Attention

$$\mathbf{s}_j = \mathrm{cGRU}_{\mathrm{att}}\left(\mathbf{s}_{j-1}, y_{j-1}, \mathrm{C}\right)$$

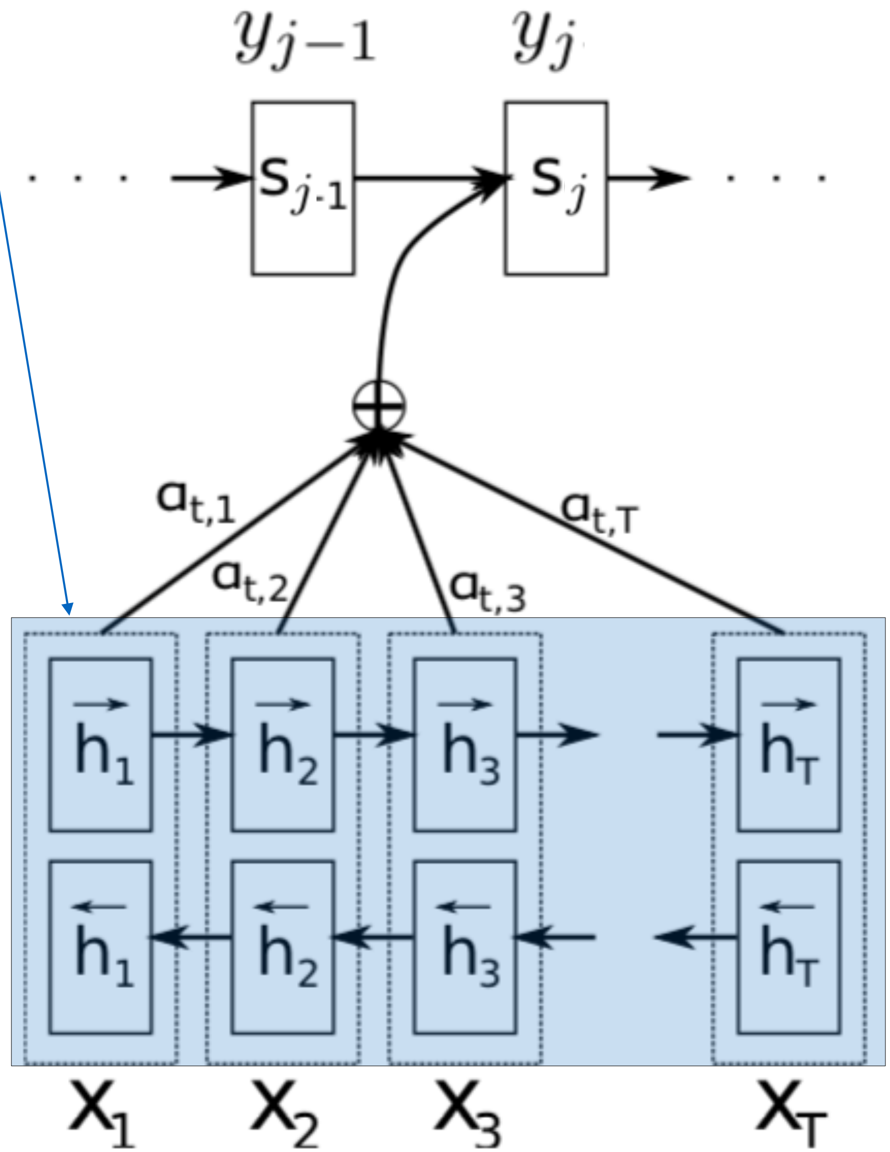
$$\mathbf{s}'_j = (1 - \mathbf{z}'_j) \odot \underbrace{\underline{\mathbf{s}}'_j}_{\mathbf{E}[y_{j-1}]} + \mathbf{z}'_j \odot \mathbf{s}_{j-1}$$

$$\mathbf{c}_j = \mathrm{ATT}\left(\mathrm{C}, \mathbf{s}'_j\right) = \sum_i^{T_x} \alpha_{ij} \mathbf{h}_i,$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{kj})},$$

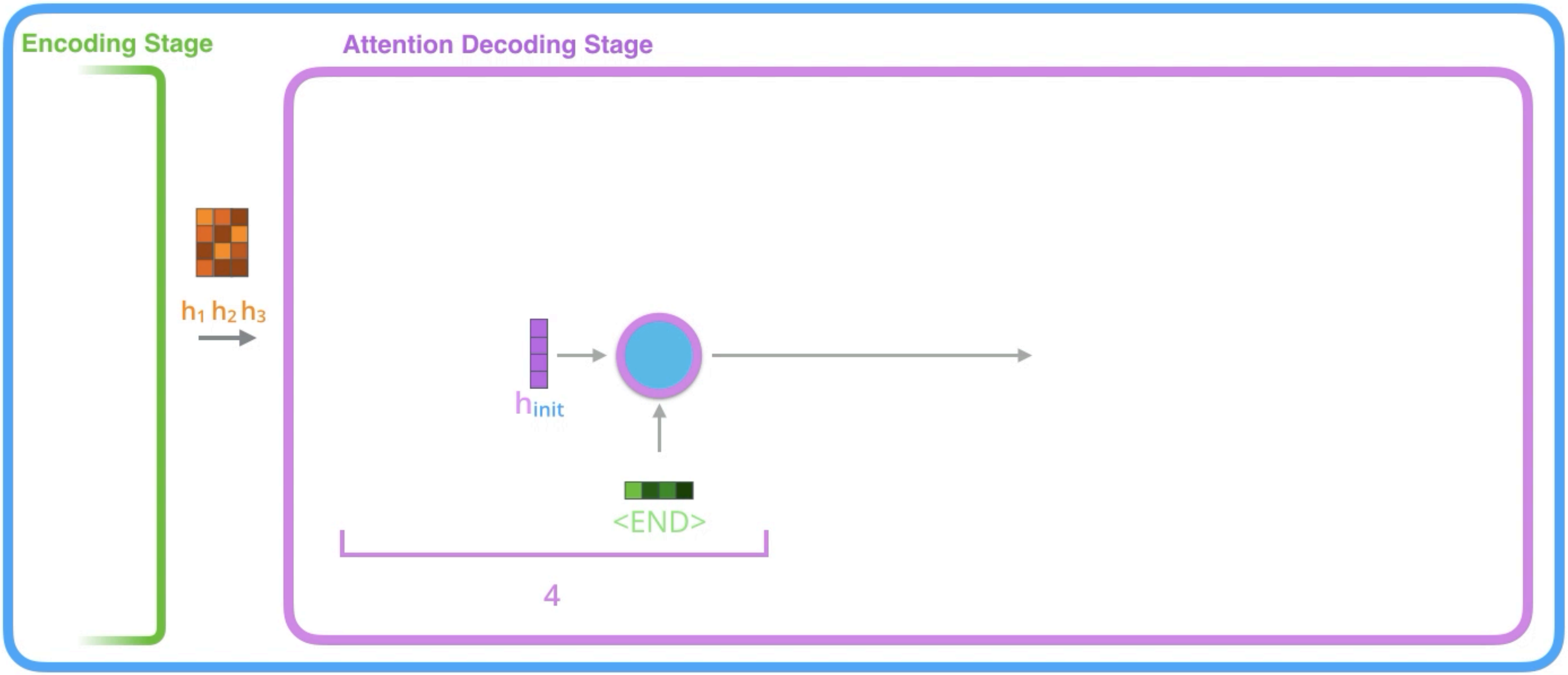
$$e_{ij} = \mathbf{v}_a^{\mathsf{T}} \tanh\left(\mathbf{U}_a \mathbf{s}'_j + \mathbf{W}_a \mathbf{h}_i\right)$$

$$\mathbf{s}_j = (1 - \mathbf{z}_j) \odot \underbrace{\underline{\mathbf{s}}_j}_{\mathbf{c}_j} + \mathbf{z}_j \odot \mathbf{s}'_j$$

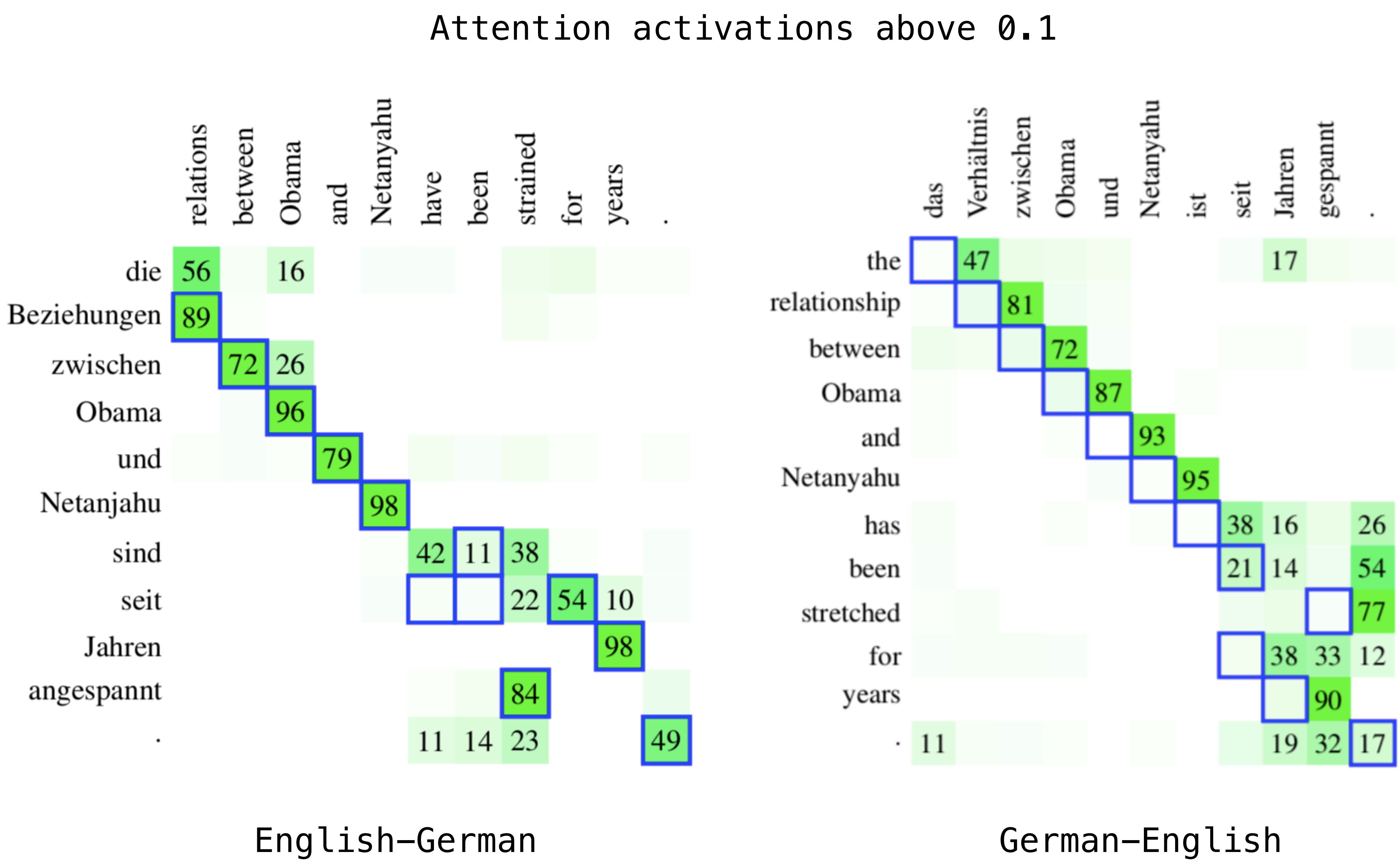


Neural Machine Translation

SEQUENCE TO SEQUENCE MODEL WITH ATTENTION



Attention Activations



(Koehn & Knowles 2017) Six Challenges for Neural Machine Translation

Transformer Architecture

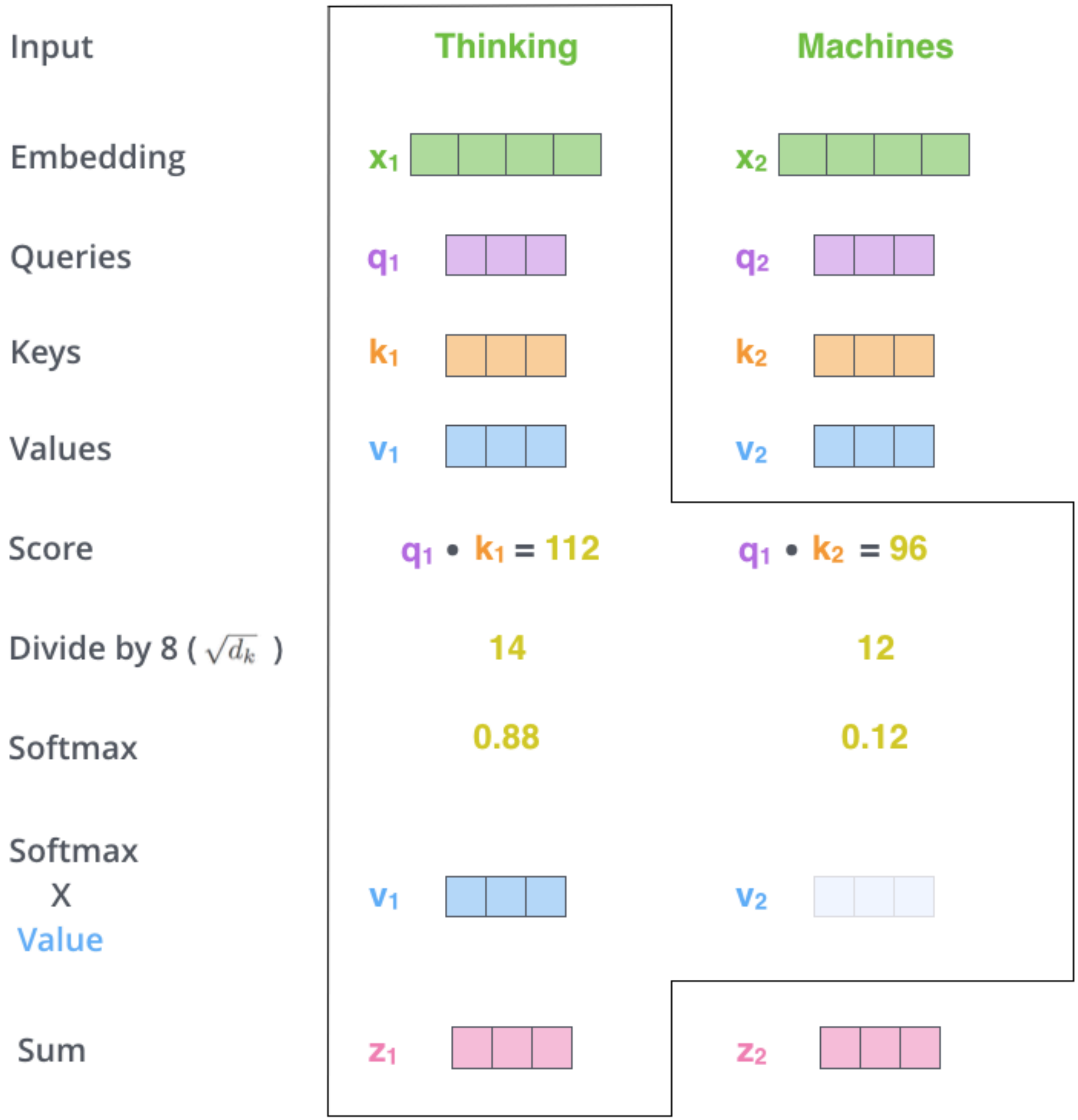
Transformer

In lieu of an RNN, use attention.

High throughput & expressivity: compute queries, keys and values as (different) linear transformations of the input.

Attention weights are queries • keys; outputs are sums of weighted values.

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



(Vaswani et al., 2017) Attention is All You Need
Figure: http://jalammar.github.io/illustrated-transformer/

Some Transformer Concerns

Problem: Bag-of-words representation of the input.
Remedy: Position embeddings are added to the word embeddings.

Problem: During generation, can't attend to future words.
Remedy: Masked training that zeroes attention to future words.

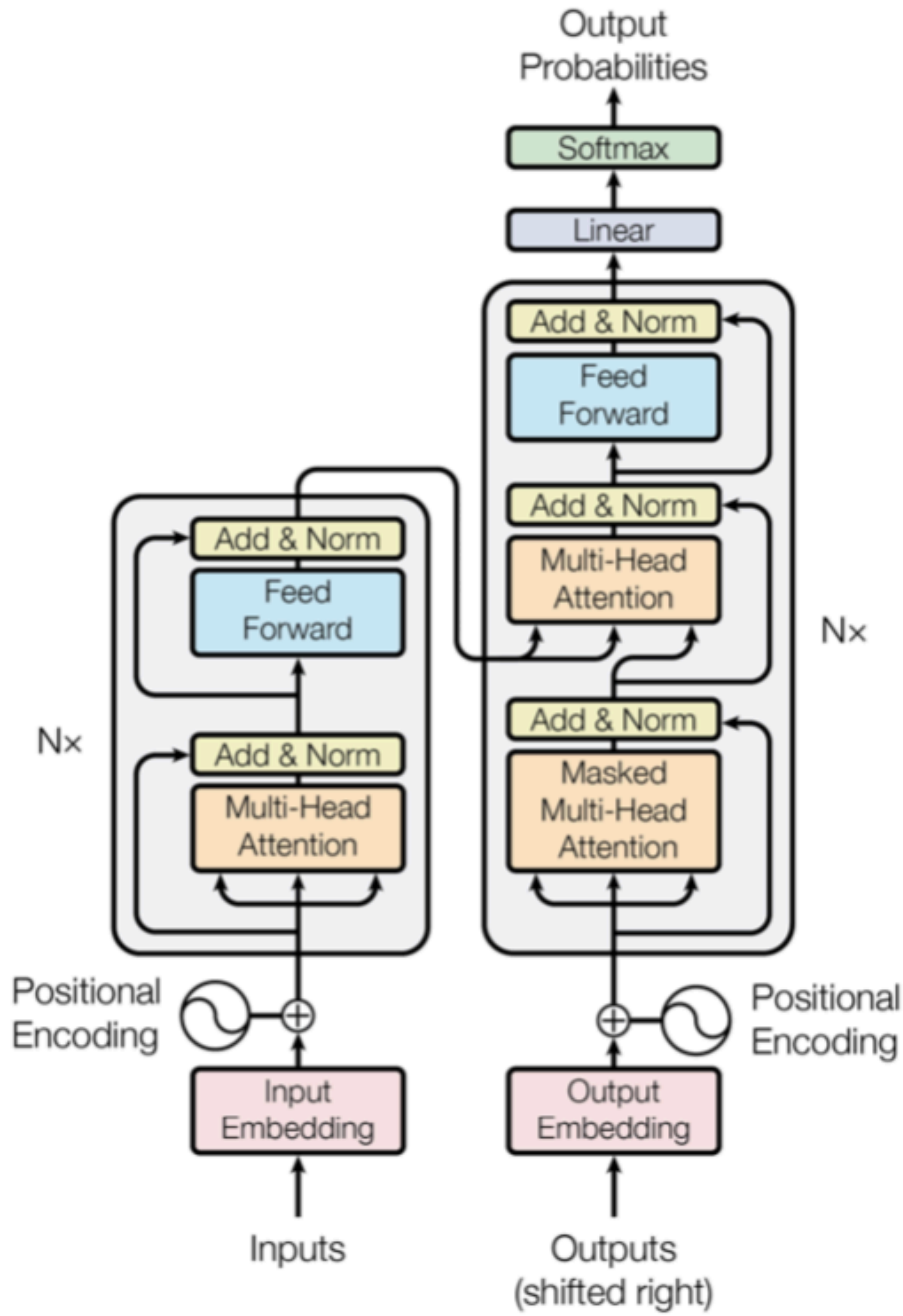
Problem: Deep networks needed to integrated lots of context.
Remedies: Residual connections and multi-head attention.

Problem: Optimization is hard.
Remedies: Large mini-batch sizes and layer normalization.

Transformer Architecture

- Layer normalization ("Add & Norm" cells) helps with RNN+attention architectures as well.
- Positional encodings can be learned or based on a formula that makes it easy to represent distance.

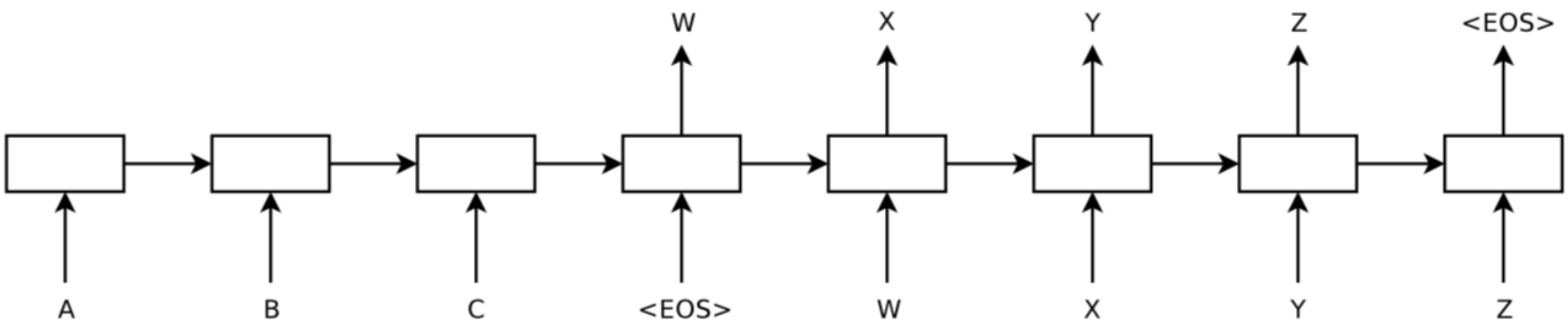
	EN-DE
ByteNet [18]	23.75
Deep-Att + PosUnk [39]	
GNMT + RL [38]	24.6
ConvS2S [9]	25.16
MoE [32]	26.03
Deep-Att + PosUnk Ensemble [39]	
GNMT + RL Ensemble [38]	26.30
ConvS2S Ensemble [9]	26.36
Transformer (base model)	27.3
Transformer (big)	**28.4**



Training and Inference

Training Loss Function

Teacher forcing: During training, only use the predictions of the model for the loss, not the input.



Label smoothing: Update toward a distribution in which
- 0.9 probability is assigned to the observed word, and
- 0.1 probability is divided uniformly among all other words.

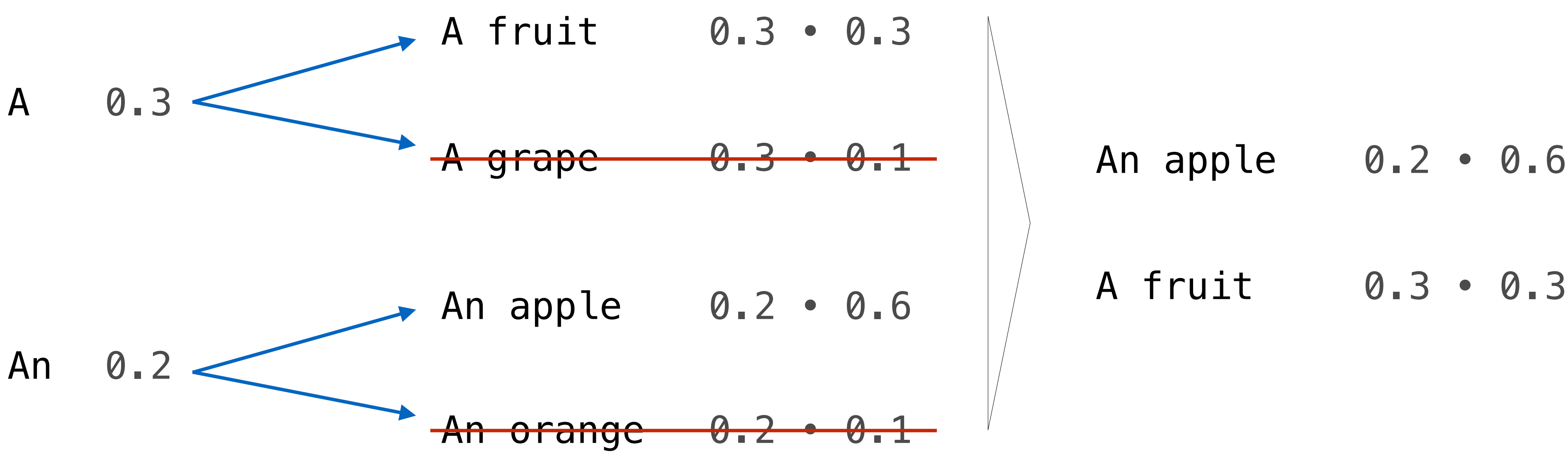
Sequence-level loss has been explored, but (so far) abandoned.

Search Strategies

For each target position, each word in the vocabulary is scored. (Alternatively, a restricted list of vocabulary items can be selected based on the source sentence, but quality can degrade.)

Greedy decoding: Extend a single hypothesis (partial translation) with the next word that has highest probability.

Beam search: Extend multiple hypotheses, then prune.



Training Data

Subwords

The sequence of symbols that are embedded should be common enough that an embedding can be estimated robustly for each, and all symbols have been observed during training.

Solution 1: Symbols are words with rare words replaced by UNK.

- Replacing UNK in the output is a new problem (like alignment).
- UNK in the input loses all information that might have been relevant from the rare input word (e.g., tense, length, POS).

Solution 2: Symbols are subwords.

- Byte-Pair Encoding is the most common approach.
- Other techniques that find common subwords work equally well (but are more complicated).
- Training on many sampled subword decompositions improves out-of-domain translations.

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

(Sennrich et al., 2016) Neural Machine Translation of Rare Words with Subword Units
(Kudo, 2018) Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates

BPE Example

system	sentence
source	health research institutes
reference	Gesundheitsforschungsinstitute
word-level (with back-off)	Forschungsinstitute
character bigrams	Fo|rs|ch|un|gs|in|st|it|ut|io|ne|n
BPE	Gesundheits|forsch|ungsin|stitute

Example from Rico Sennrich

Back Translations

Synthesize an en-de parallel corpus by using a de-en system to translate monolingual de sentences.

- Better generating systems don't seem to matter much.
- Can help even if the de sentences are already in an existing en-de parallel corpus!

system	EN→DE		DE→EN	
	dev	test	dev	test
baseline	22.4	26.8	26.4	28.5
+synthetic	25.8	31.6	29.9	36.2
+ensemble	27.5	33.1	31.5	37.5
+r2l reranking	**28.1**	**34.2**	**32.1**	**38.6**

Table 2: English↔German translation results (BLEU) on dev (newstest2015) and test (newstest2016). Submitted system in bold.

(Sennrich et al., 2015) Improving Neural Machine Translation Models with Monolingual Data
(Sennrich et al., 2016) Edinburgh Neural Machine Translation Systems for WMT 16