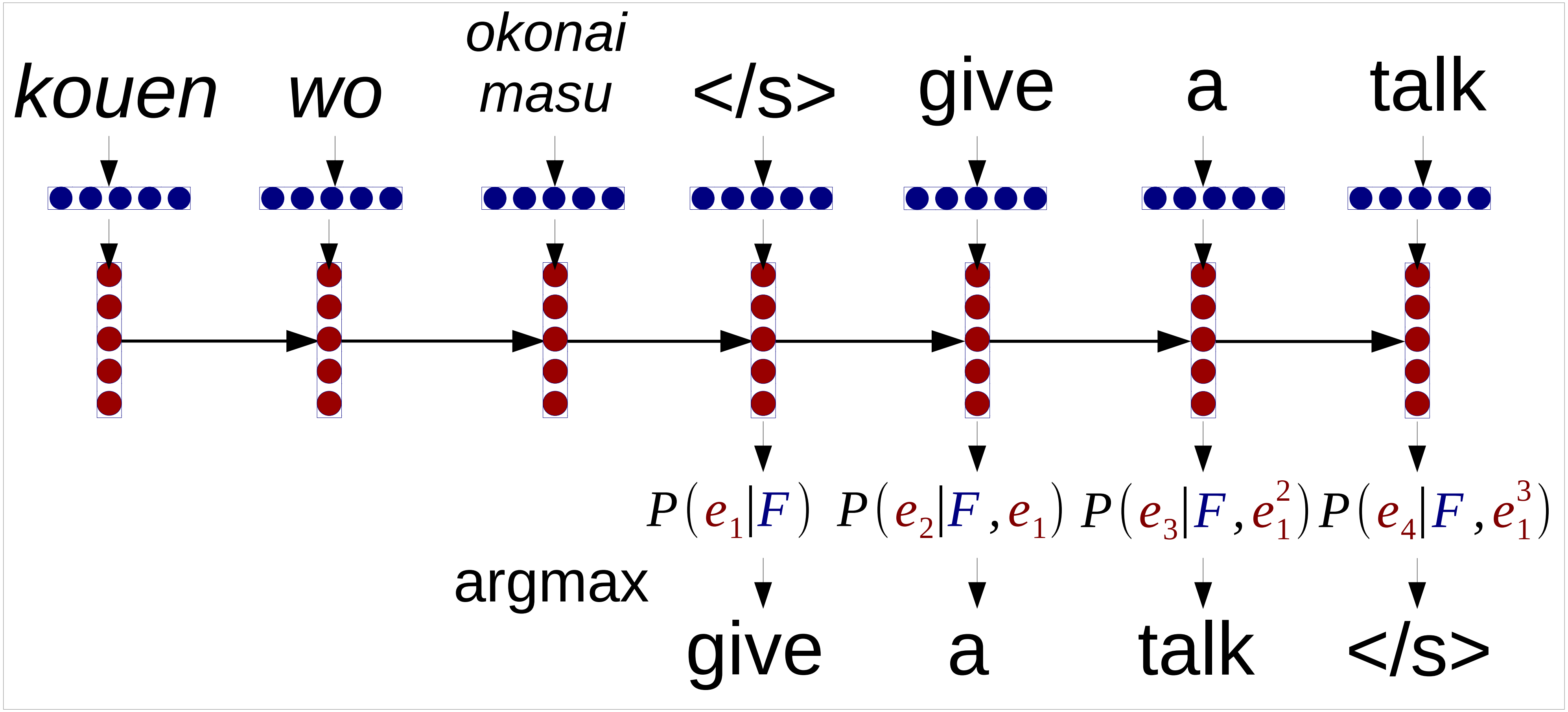
Softmax Alternatives in Neural MT

Graham Neubig
5/24/2017

Neural MT Models



How we Calculate Probabilities

$$p(e_i|\mathbf{h}_i) = \text{softmax}(\ W * \mathbf{h}_i\ + \mathbf{b}\ )$$

Next word prob. Weights Hidden context Bias

W

$[\mathbf{w}_{*,1}, \mathbf{w}_{*,2}, \mathbf{w}_{*,3}, \ldots]$

$\mathbf{b}$

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \ldots \end{pmatrix}$$

In other words, the score is:

$$s(e_i|\mathbf{c}_i) = \mathbf{w}_{*,k} \cdot \mathbf{c}_i + b_k$$

Closeness of output embedding and context + bias. Choose word with highest score

A Visual Example

$$p = \text{softmax}(W h + b)$$

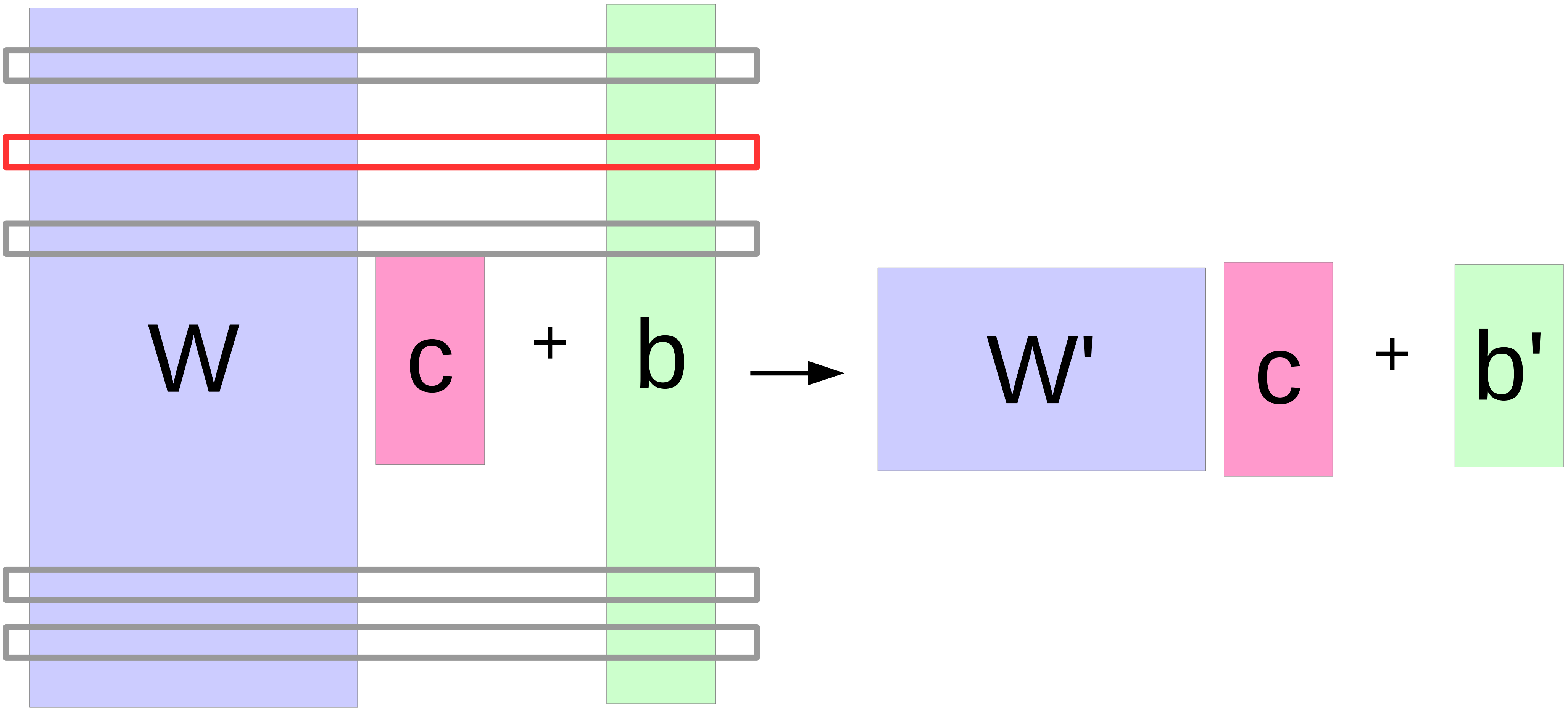
Problems w/ Softmax

- Computationally inefficient at training time
- Computationally inefficient at test time
- Many parameters
- Sub-optimal accuracy

Calculation/Parameter Efficient Softmax Variants

Negative Sampling/ Noise Contrastive Estimation

- Calculate the denominator over a subset



Negative samples according to distribution q

Lots of Alternatives!

- **Noise contrastive estimation:** train a model to discriminate between true and false examples

$$p(D=0 \mid c,w) = \frac{k \times q(w)}{u_\theta(w,c) + k \times q(w)}$$
$$p(D=1 \mid c,w) = \frac{u_\theta(w,c)}{u_\theta(w,c) + k \times q(w)}.$$

$$\sum_{(w,c)\in\mathcal{D}} \left(\log p(D=1 \mid c,w) + \sum_{i=1,\overline{w}\sim q}^{k} \log p(D=0 \mid c,\overline{w}) \right)$$

- **Negative sampling:** e.g. word2vec

$$p(D=0 \mid c,w) = \frac{1}{u_\theta(w,c) + 1}$$
$$p(D=1 \mid c,w) = \frac{u_\theta(w,c)}{u_\theta(w,c) + 1}.$$

- **BlackOut**

$$J^s_{disc}(\theta) = \log \tilde{p}_\theta(w_i|s) + \sum_{j\in S_K} \log(1 - \tilde{p}_\theta(w_j|s)),$$

Used in MT:
Eriguchi et al. 2016: Tree-to-sequence attentional neural machine translation

Ref: Chris Dyer, 2014. Notes on Noise Contrastive Estimation and Negative Sampling

GPUifying Noise Contrastive Estimation

- Creating the negative samples and arranging memory is expensive on GPU
- Simple solution: sample the negative samples once for each mini-batch

Zoph et al. 2016. Simple, Fast Noise-Contrastive Estimation for Large RNN Vocabularies

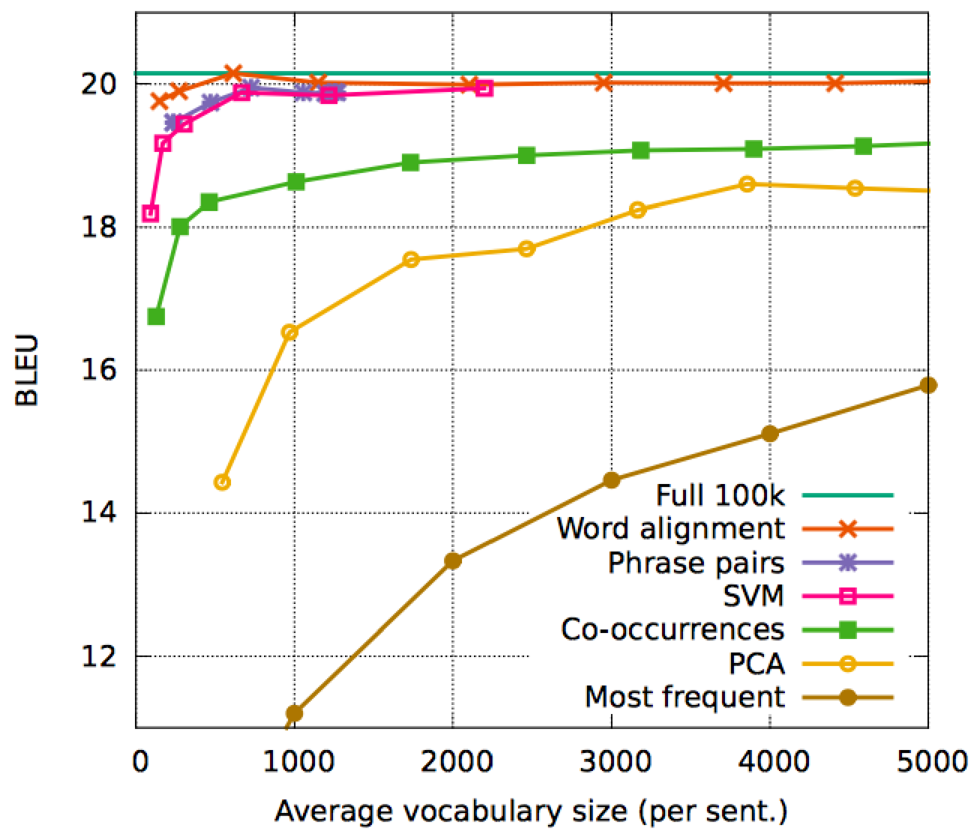
Summary of Negative Sampling Approaches

- Train time efficiency: **Much faster!**
- Test time efficiency: **Same**
- Number of parameters: **Same**
- Test time accuracy: **A little worse?**
- Code complexity: **Moderate**

Vocabulary Selection

- Select the vocabulary on a per-sentence basis

Mi 2016. Vocabulary Manipulation for NMT
L'Hostis et al. 2016. Vocabulary Selection Strategies for NMT



Summary of Vocabulary Selection

- Train time efficiency: **A little faster**
- Test time efficiency: **Much faster!**
- Number of parameters: **Same**
- Test time accuracy: **Better** or **a little worse**
- Code complexity: **Moderate**

Class-based Softmax

- Predict P(class|hidden), then P(word|class,hidden)
- Because P(w|c,h) is 0 for all but one class, efficient computation

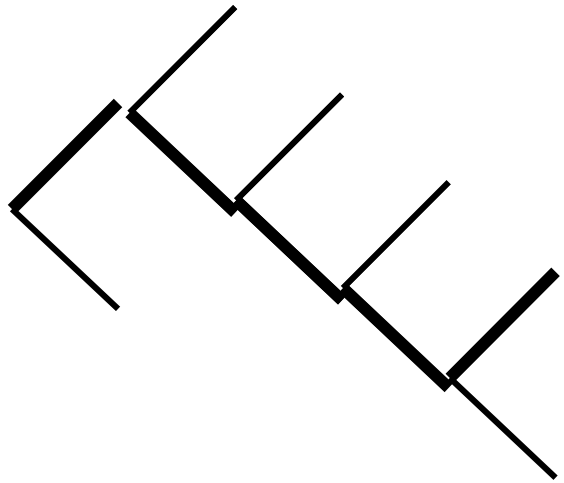
$$\mathrm{softmax}(W_c \, h + b_c)$$

$$\mathrm{softmax}(W_w \, h + b_w)$$

Goodman 2001. Classes for Fast Maximum Entropy Training

Hierarchical Softmax

- Tree-structured prediction of word ID
- Usually modeled as a sequence of binary decisions



0 1 1 1 0 → word 14

Morin and Bengio 2005: Hierarchical Probabilistic NNLM

Summary of Class-based Softmaxes

- Train time efficiency: **Faster on CPU**, **Pain to GPU**
- Test time efficiency: **Worse**
- Number of parameters: **More**
- Test time accuracy: **Slightly worse** to **slightly better**
- Code complexity: **High**

Binary Code Prediction

- Just directly predict the binary code of the word ID

$$\sigma\left(W \; h + b \right) = \begin{matrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{matrix}$$

$$\downarrow$$

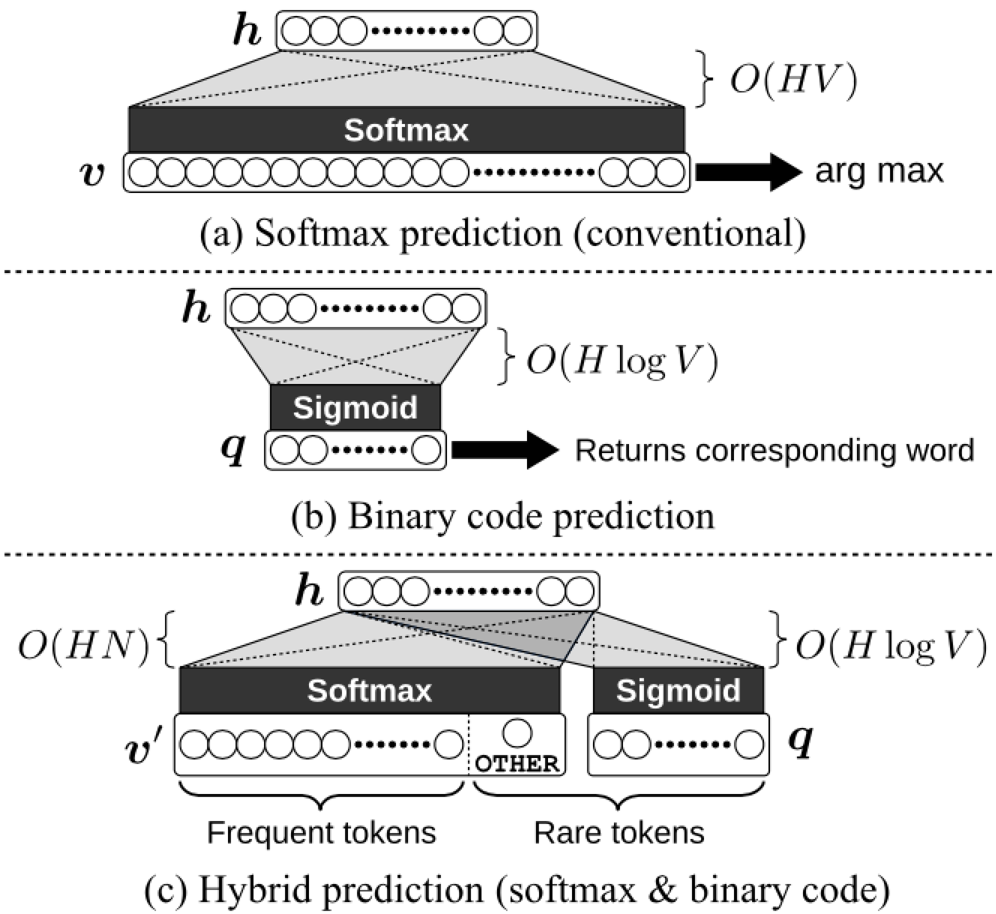
word 14

- Like hierarchical softmax, but with shared weights at every layer → fewer parameters, easy to GPU

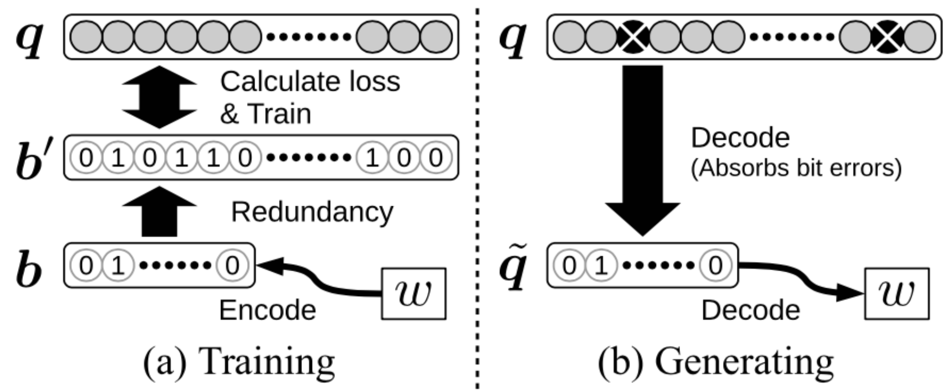
Oda et al. 2017: NMT Via Binary Code Prediction

Two Improvements

Hybrid model



Error correcting codes



Summary of Binary Code Prediction

- Train time efficiency: **Faster**
- Test time efficiency: **Faster (12x on CPU!)**
- Number of parameters: **Fewer**
- Test time accuracy: **Slightly worse**
- Code complexity: **Moderate**

Parameter Sharing

Parameter Sharing

- We have two |V| x |h| matrices in the decoder:
  - Input word embeddings, which we look up and feed into the RNN
  - Output word embeddings, which are the weight matrix **W** in the softmax
- Simple idea: tie their weights together

Press et al. 2016: Using the output embedding to improve language models
Inan et al. 2016: Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling

Summary of Parameter Sharing

- Train time efficiency: **Same**
- Test time efficiency: **Same**
- Number of parameters: **Fewer**
- Test time accuracy: **Better**
- Code complexity: **Low**

Incorporating External Information

Problems w/ Lexical Choice in Neural MT

Input: I come from Tunisia.

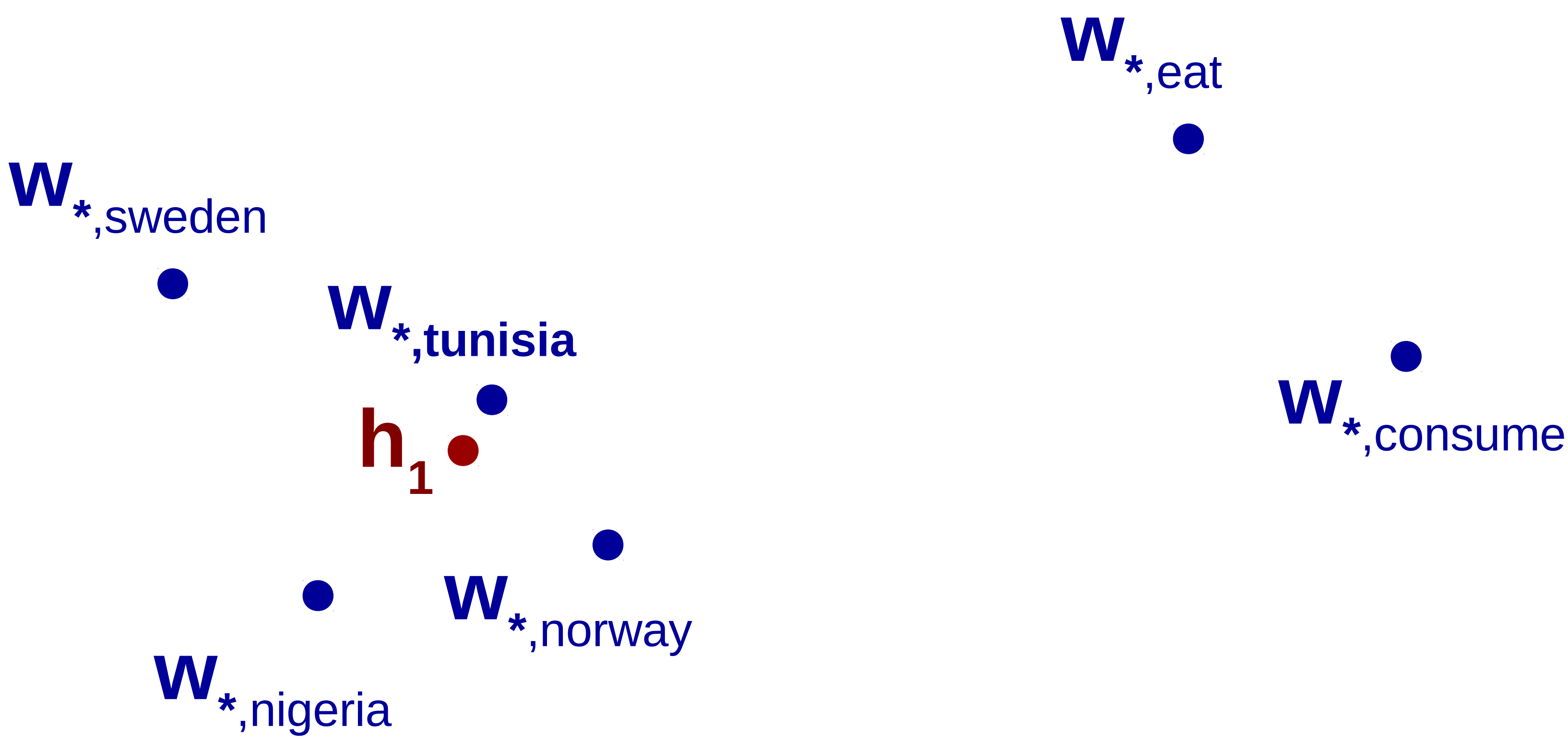
Reference: チュニジアの出身です。
Chunisia no shusshindesu.
(I'm from Tunisia.)

System: ノルウェーの出身です。
Noruue- no shusshindesu.
(I'm from Norway.)

Arthur et al. 2016:
Incorporating Discrete Translation Lexicons in NMT

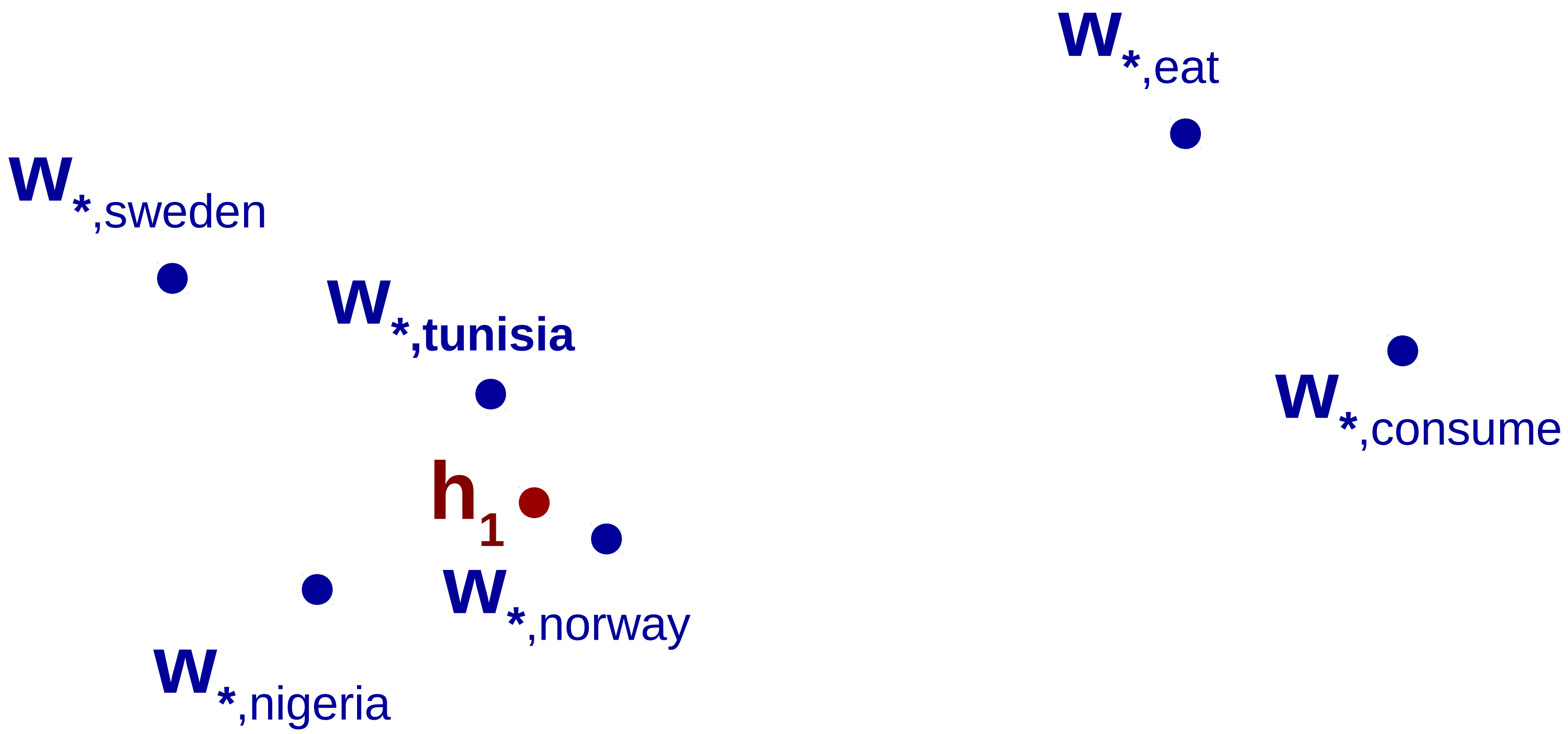
When Does Translation Succeed? (in Output Embedding Space)

I come from Tunisia



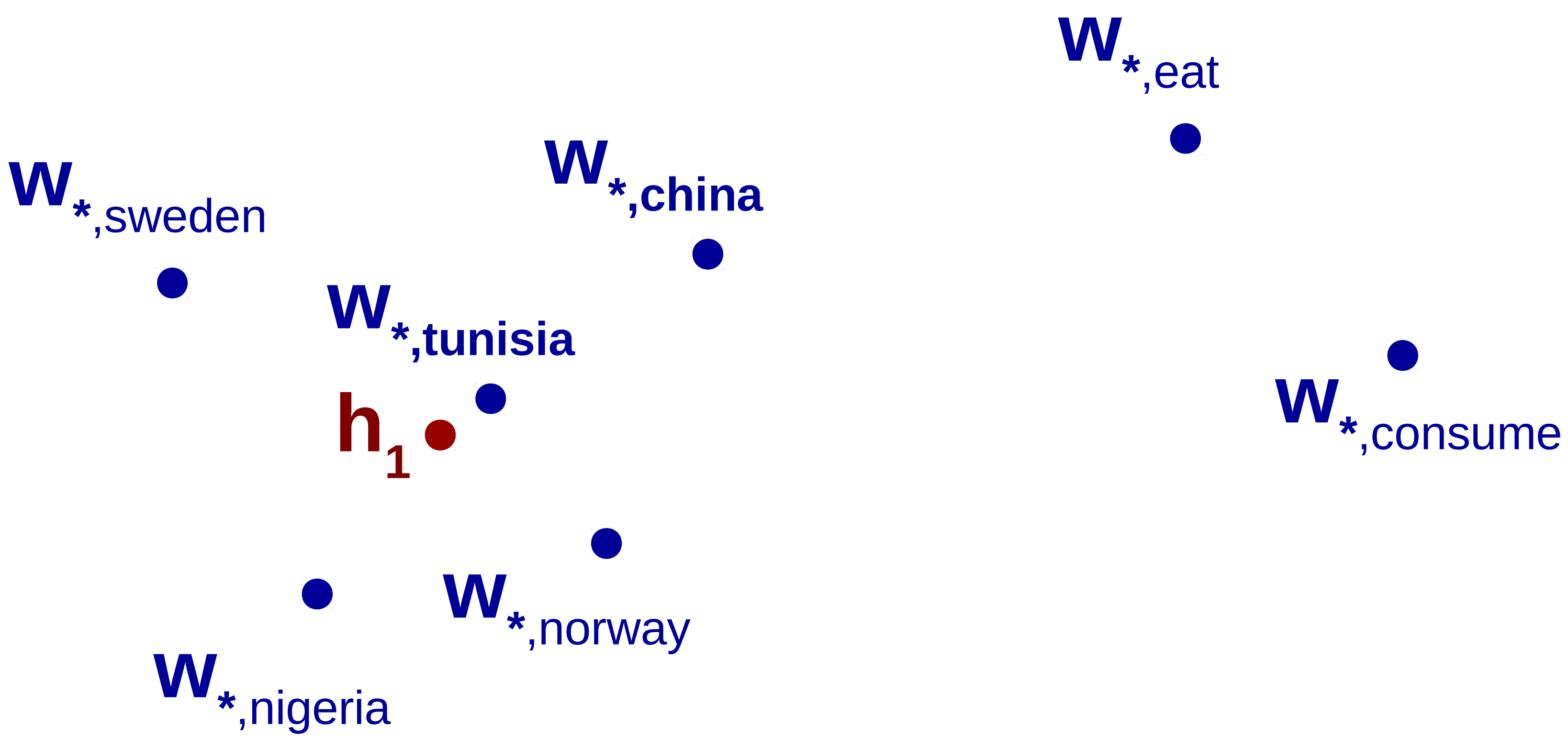
When Does Translation Fail? Embeddings Version

I come from Tunisia



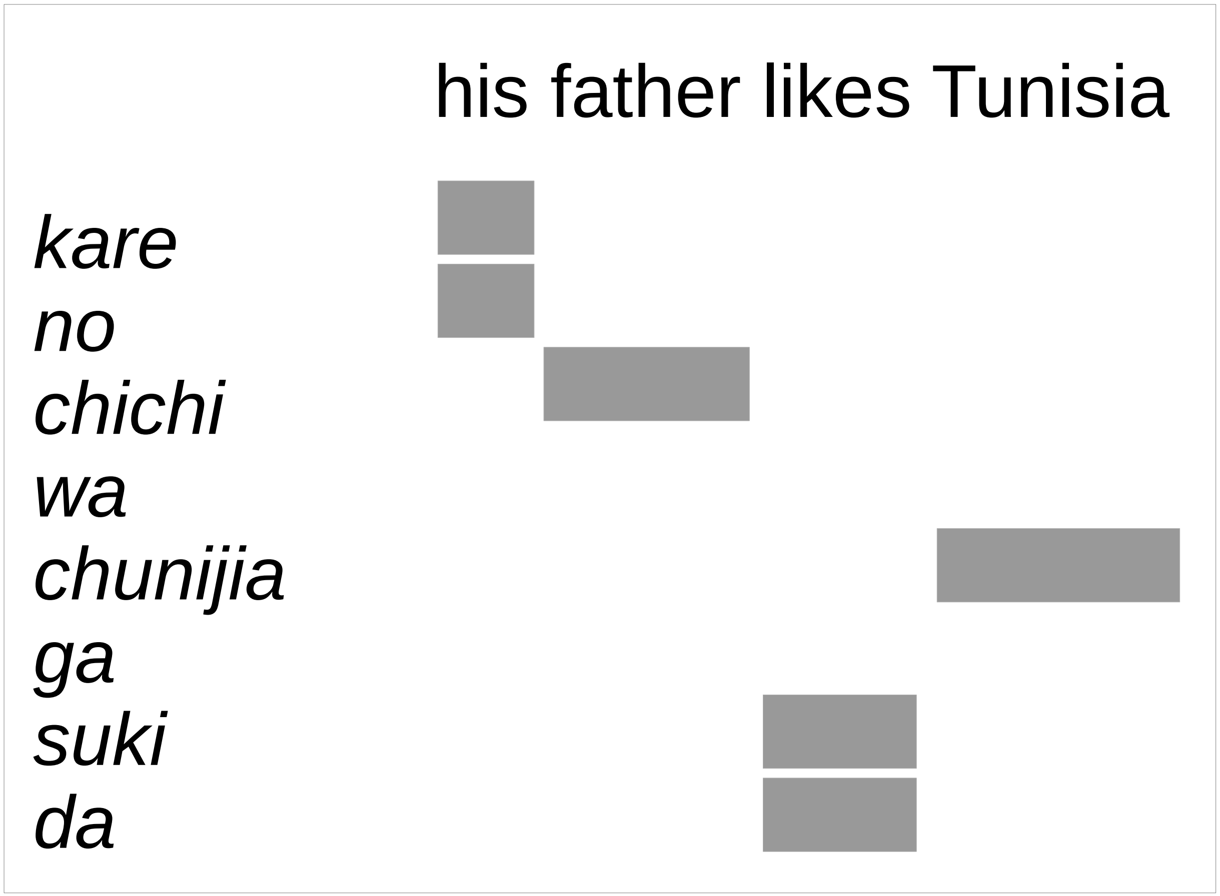
When Does Translation Fail? Bias Version

I come from Tunisia



$\mathbf{b}_{\mathbf{tunisia}} = -0.5$ $\quad$ $\mathbf{b}_{\mathbf{china}} = 4.5$

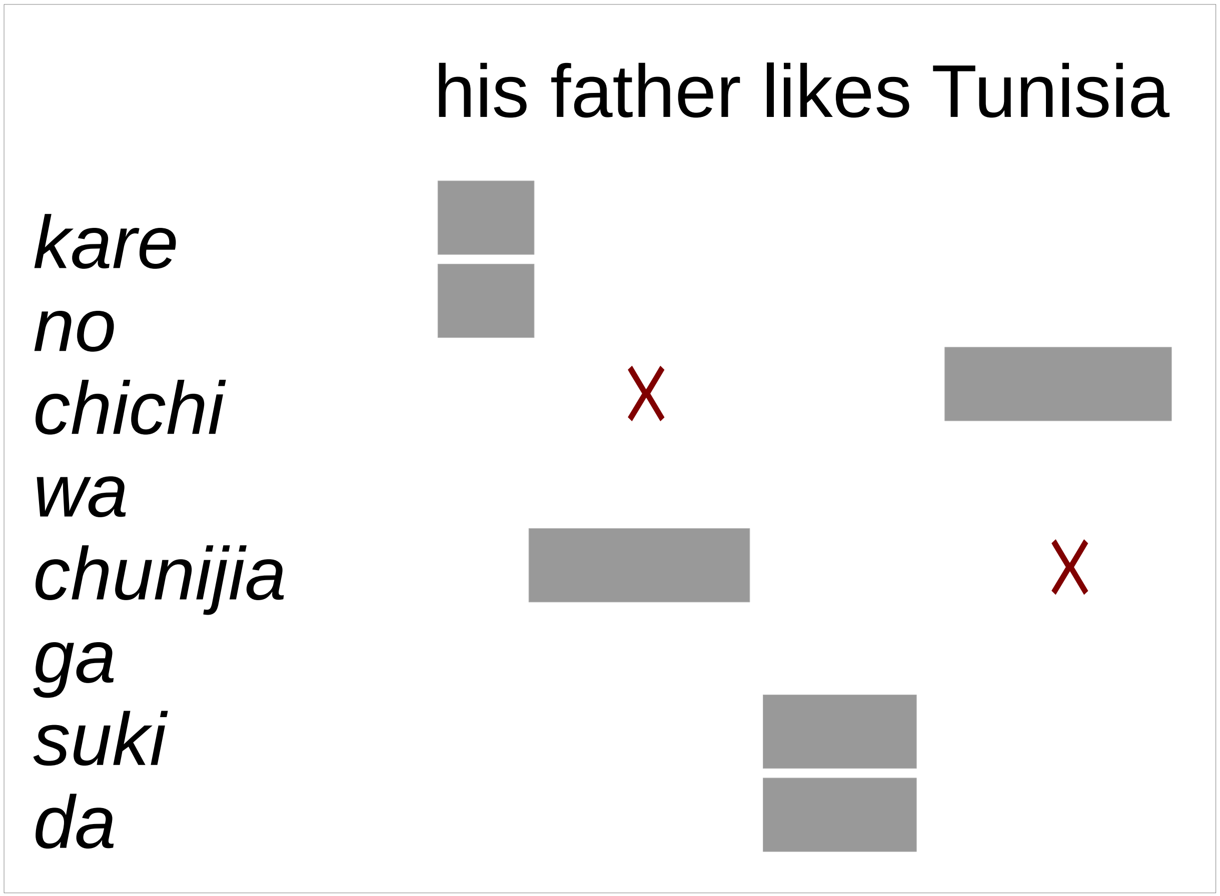
What about Traditional Symbolic Models?



P(*kare*|his) = 0.5
P(*no*|his) = 0.5
P(*chichi*|father) = 1.0
P(*chunijia*|Tunisia) = 1.0
P(*suki*|likes) = 0.5
P(*da*|likes) = 0.5

1-to-1 alignment

Even if We Make a Mistake...



P(*kare*|his) = 0.5
P(*no*|his) = 0.5
P(*chichi*|Tunisia) = 1.0
P(*chunijia*|father) = 1.0
P(*suki*|likes) = 0.5
P(*da*|likes) = 0.5

Different mistakes than neural MT

Soft alignment possible

Calculating Lexicon Probabilities

	I	come	from	Tunisia	
Attention	0.05	0.01	0.02	0.93	
watashi	0.6	0.03	0.01	0.0	0.03
ore	0.2	0.01	0.02	0.0	0.01
…	…	…	…	…	…
kuru	0.01	0.3	0.01	0.0	0.00
kara	0.02	0.1	0.5	0.01	0.02
…	…	…	…	…	…
chunijia	0.0	0.0	0.0	0.96	0.89
oranda	0.0	0.0	0.0	0.0	0.00

Word-by-word lexicon prob

Conditional lexicon prob

Incorporating w/ Neural MT

- softmax bias:

$$p(e_i|\mathbf{h}_i) = \text{softmax}(W * \mathbf{h}_i + \mathbf{b} + \log(\mathbf{lex}_i + \varepsilon))$$

To prevent -∞ scores

- Linear interpolation:

$$p(e_i|\mathbf{h}_i) = \gamma * \text{softmax}(W * \mathbf{h}_i + \mathbf{b}) + (1-\gamma) * \mathbf{lex}_i$$

Summary of External Lexicons

- Train time efficiency: **Worse**
- Test time efficiency: **Worse**
- Number of parameters: **Same**
- Test time accuracy: **Better** to **Much Better**
- Code complexity: **High**

Other Varieties of Biases

- Copying source words as-is

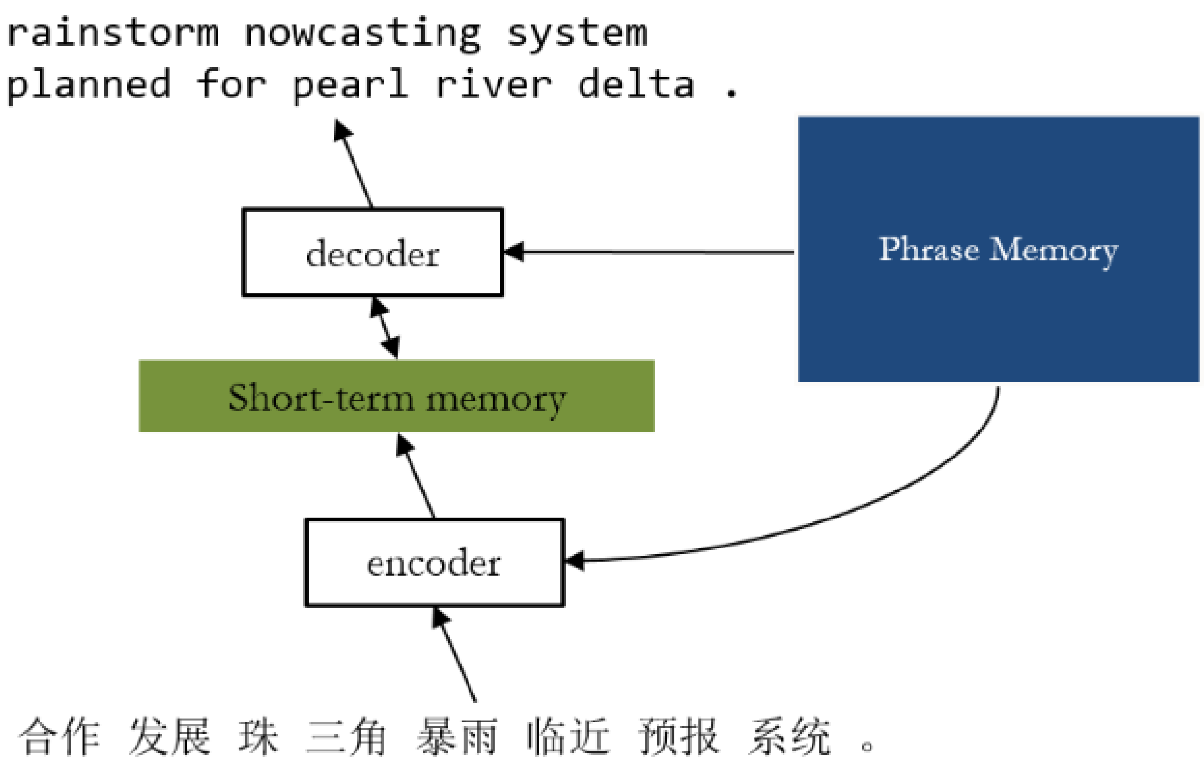
  Gu et al. 2016. Incorporating copying mechanism in sequence-to-sequence learning

  Gulcehre et al. 2016. Pointing the unknown words

- Remembering and copying target words
  Were called cache models, now called **★pointer sentinel models★** :)

  Merity et al. 2016. Pointer Sentinel Mixture Models

Use of External Phrase Tables



Tang et al. 2016. NMT with External Phrase Memory

Conclusion

Conclusion

- Lots of softmax alternatives for neural MT
  → Consider them in your systems!
- But there is no fast at train, fast at test, accurate, small, and simple method
  → Consider making one yourself!