CS224d
Deep NLP

Lecture 8:
Recap, Projects and
Fancy Recurrent Neural Networks
for Machine Translation

Richard Socher

richard@metamind.io

Overview

- Projects
- Recap of most important concepts & equations
- Wrap up: Deep RNNs and F1 evaluation
- Machine translation
- Fancy RNN Models tackling MT:
  - Gated Recurrent Units by Cho et al. (2014)
  - Long-Short-Term-Memories
    by Hochreiter and Schmidhuber (1997)

Advanced, cutting edge, blast from the past

Recap of most important concepts

Word2Vec $$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$$

Glove $$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

Nnet & Max-margin $$s = U^T f(Wx + b)$$

$$J = \max(0, 1 - s + s_c)$$

Recap of most important concepts

Multilayer Nnet

&

Backprop

$$
\begin{aligned}
x &= z^{(1)} = a^{(1)} \\
z^{(2)} &= W^{(1)} x + b^{(1)} \\
a^{(2)} &= f\left(z^{(2)}\right) \\
z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} \\
a^{(3)} &= f\left(z^{(3)}\right) \\
s &= U^T a^{(3)}
\end{aligned}
$$

$$\boxed{\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \circ f'(z^{(l)}),}$$

$$\boxed{\frac{\partial}{\partial W^{(l)}} E_R = \delta^{(l+1)} (a^{(l)})^T + \lambda W^{(l)}}$$

Recap of most important concepts

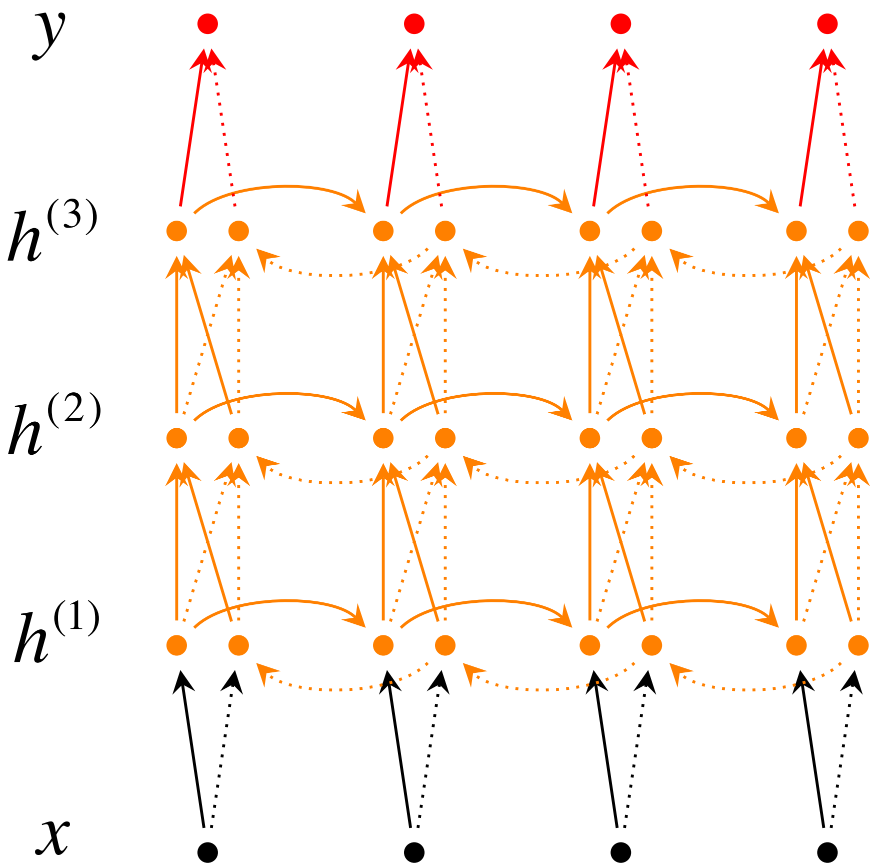
Recurrent Neural Networks

$$\begin{aligned} h_t &= \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right) \\ \hat{y}_t &= \mathrm{softmax}\left(W^{(S)} h_t\right) \end{aligned}$$

Cross Entropy Error $$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

Mini-batched SGD $$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_{t:t+B}(\theta)$$

Deep Bidirectional RNNs by Irsoy and Cardie



$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)};\overleftarrow{h}_t^{(L)}]+c)$$

Each memory layer passes an intermediate sequential representation to the next.

Data

- MPQA 1.2 corpus (Wiebe et al., 2005)
- consists of 535 news articles (11,111 sentences)
- manually labeled at the phrase level
- Evaluation: F1

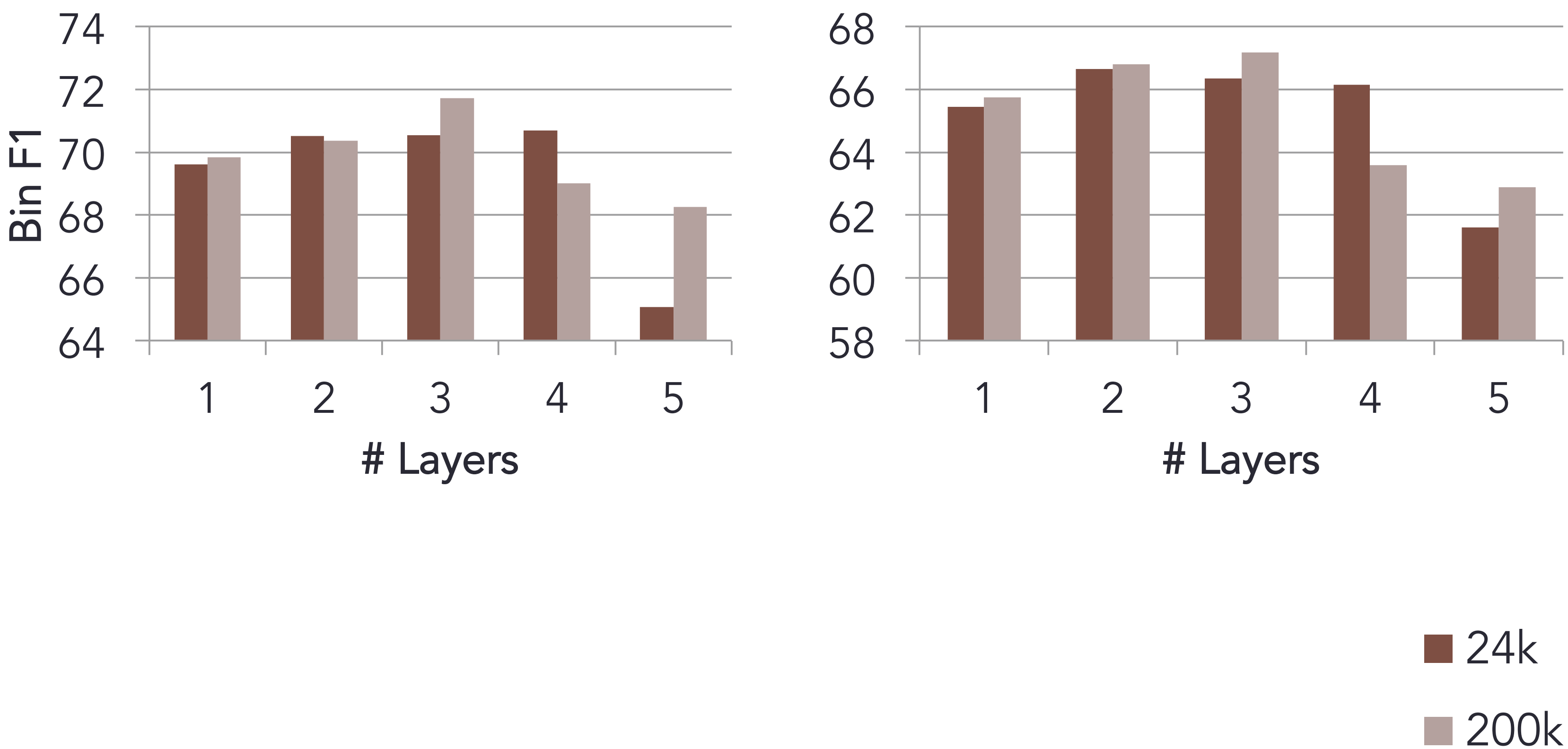
Total population	Condition positive	Condition negative
Test outcome positive	**True positive**	**False positive** (Type I error)
Test outcome negative	**False negative** (Type II error)	**True negative**

$$\text{precision} = \frac{tp}{tp + fp}$$

$$\text{recall} = \frac{tp}{tp + fn}$$

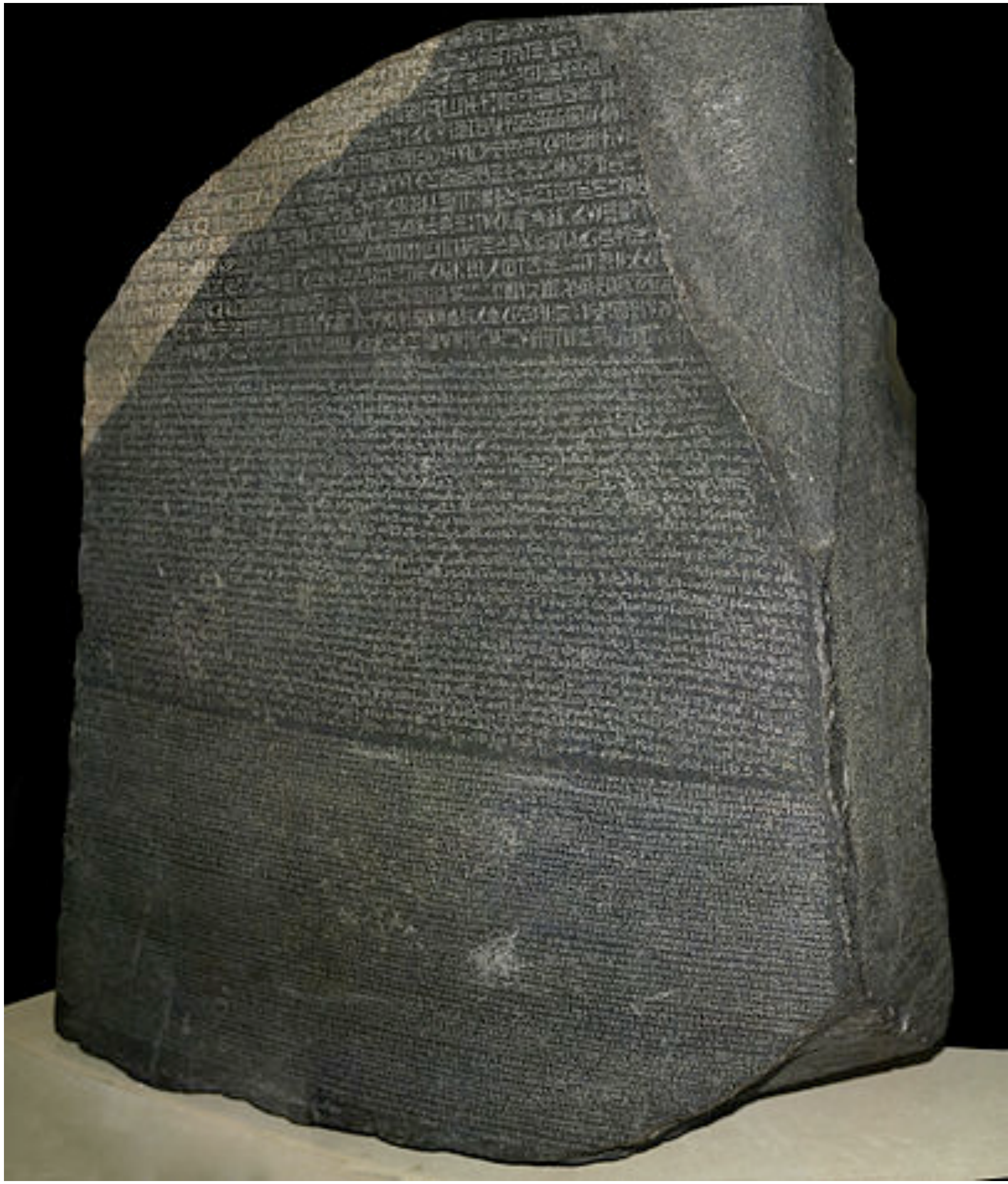
$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Evaluation


Bin F1
74
72
70
68
66
64
1
2
3
4
5
Layers
68
66
64
62
60
58
1
2
3
4
5
Layers
24k
200k

Machine Translation

- Methods are statistical
- Use parallel corpora
  - European Parliament
- First parallel corpus:
  - Rosetta Stone →
- Traditional systems are very complex

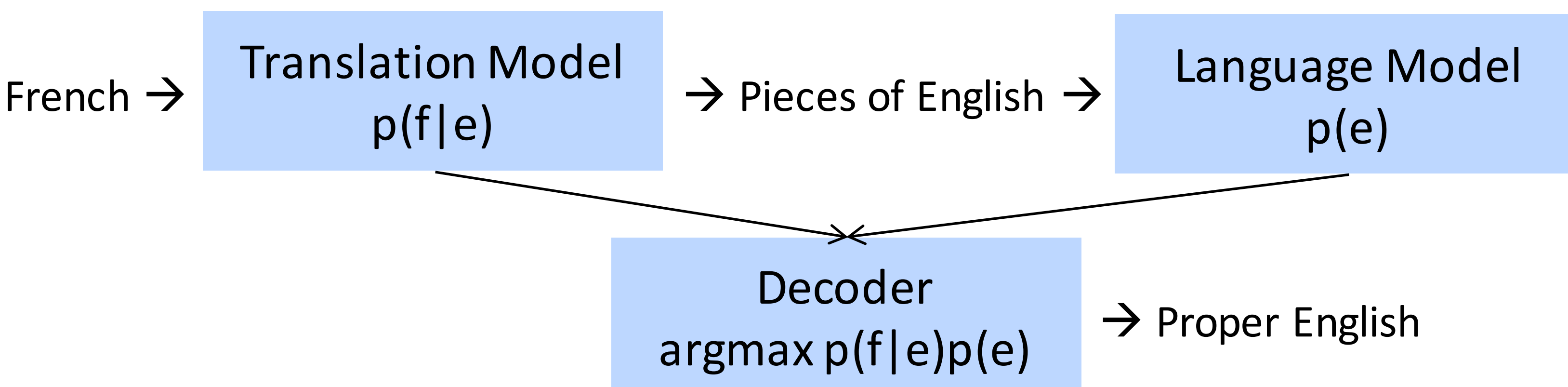


Current statistical machine translation systems

- Source language f, e.g. French
- Target language e, e.g. English
- Probabilistic formulation (using Bayes rule)

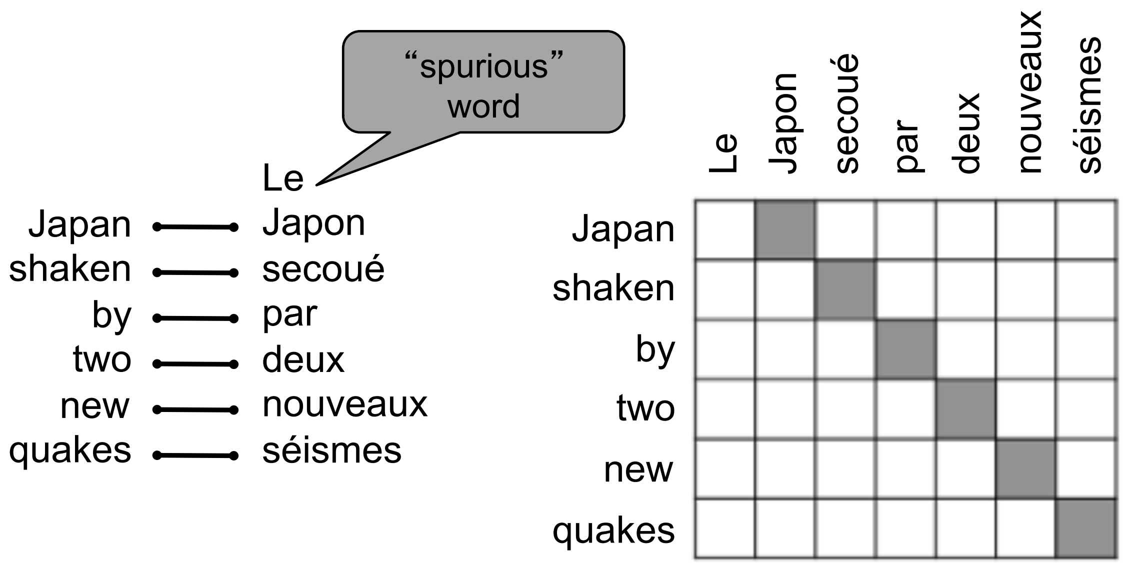
$$\hat{e} = \mathrm{argmax}_e p(e|f) = \mathrm{argmax}_e p(f|e)p(e)$$

- Translation model p(f|e) trained on parallel corpus
- Language model p(e) trained on English only corpus (lots, free!)



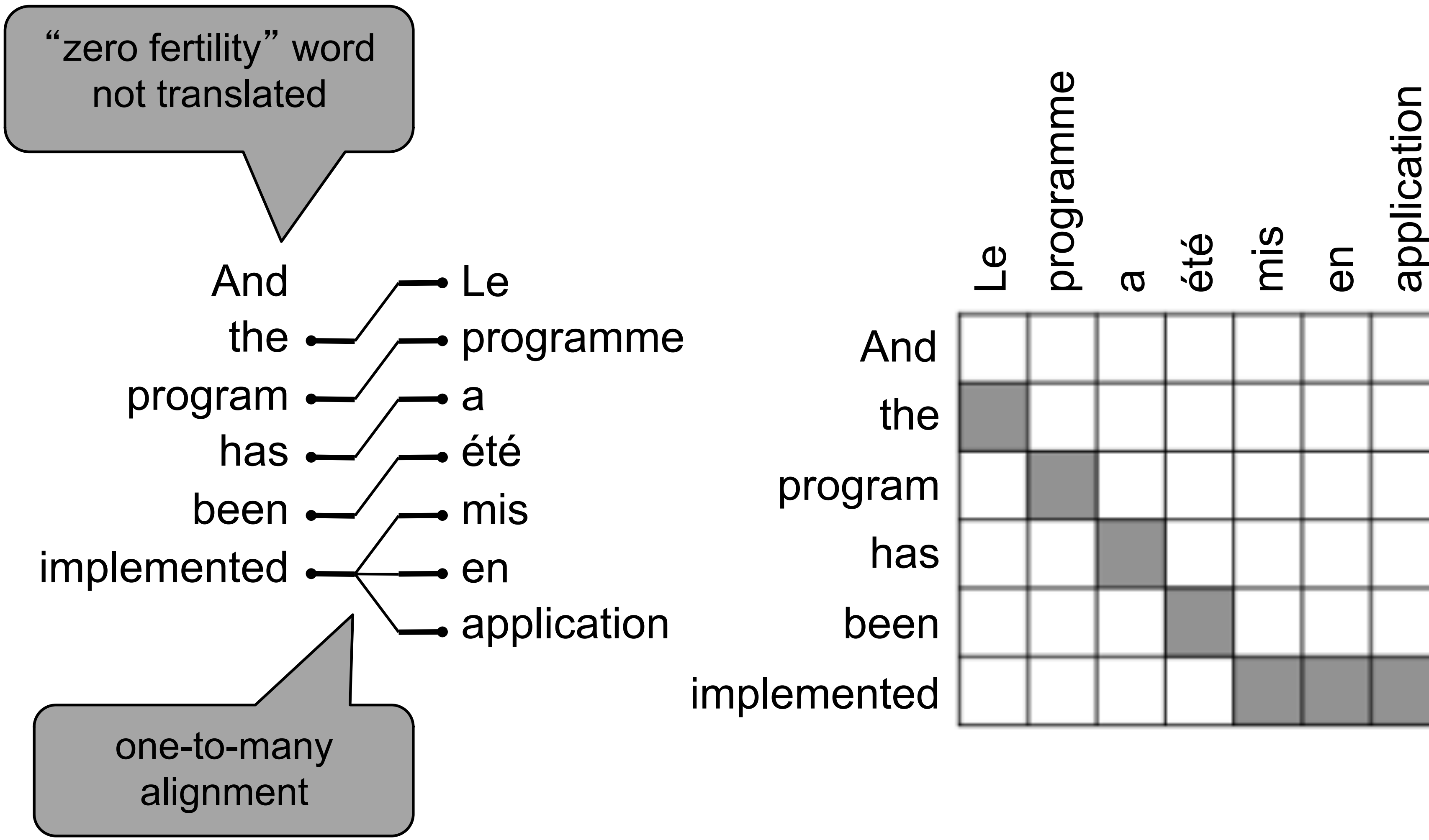
Step 1: Alignment

Goal: know which word or phrases in source language would translate to what words or phrases in target language? → Hard already!



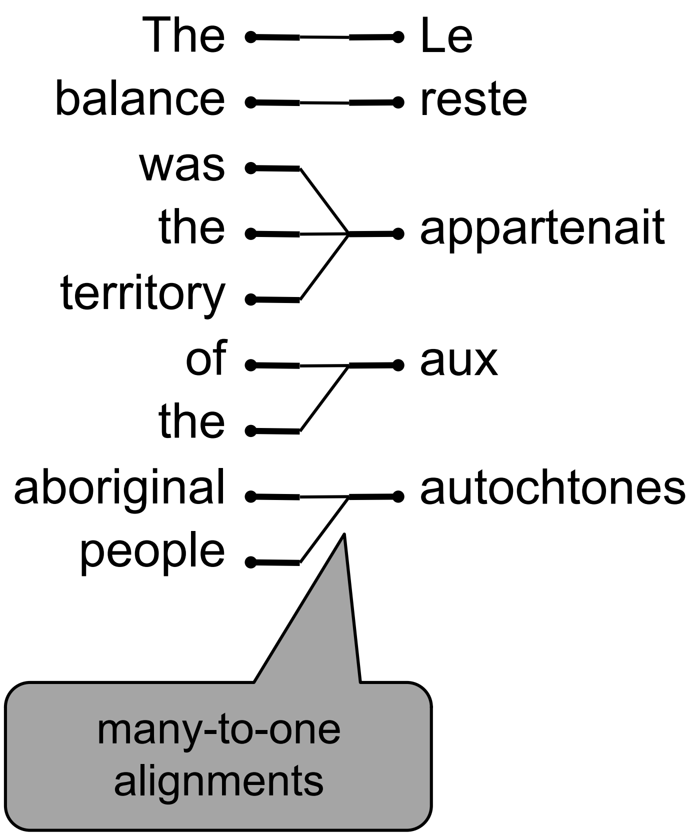
Alignment examples from Chris Manning/CS224n

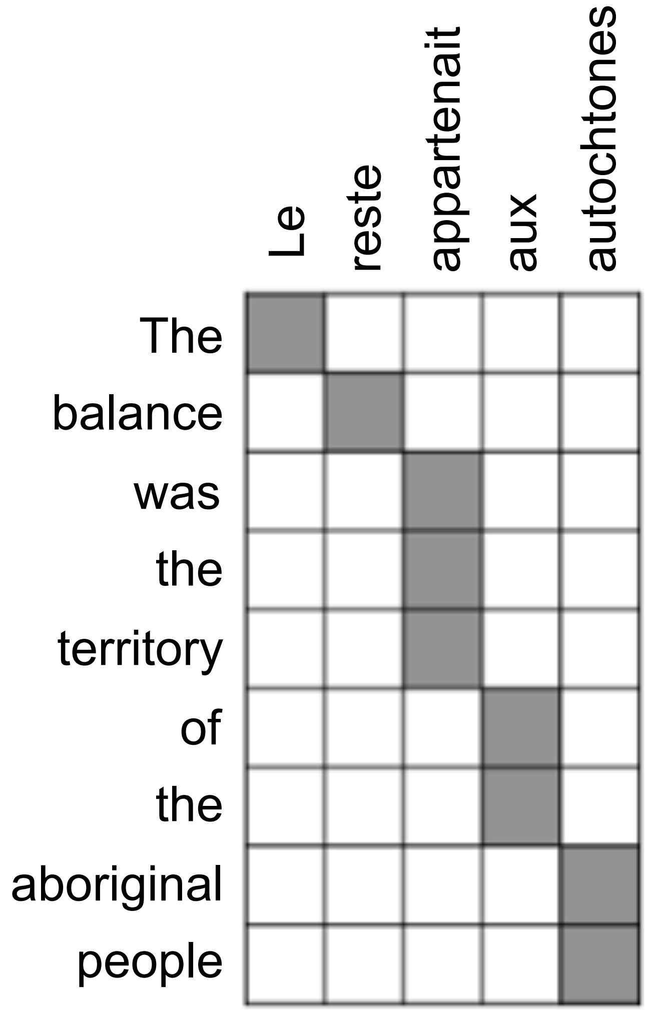
Step 1: Alignment



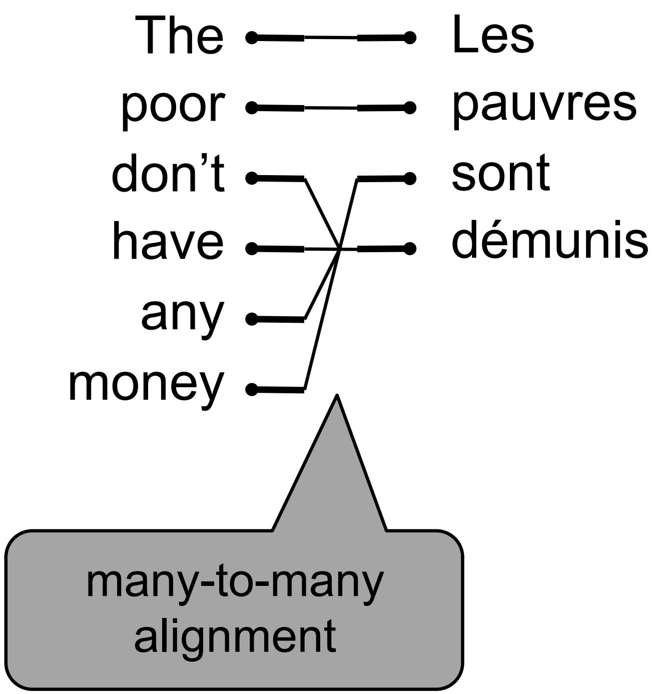
Step 1: Alignment

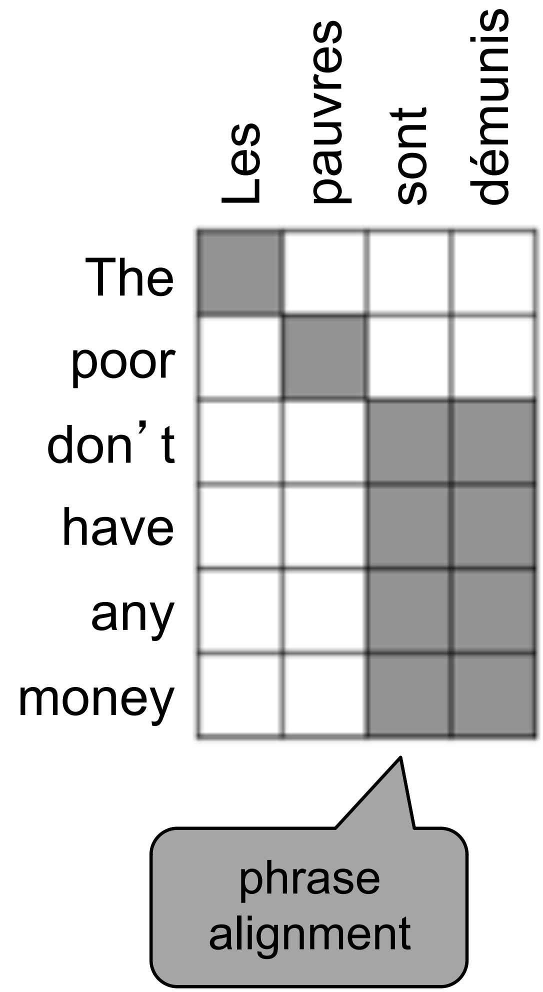
Really hard :/





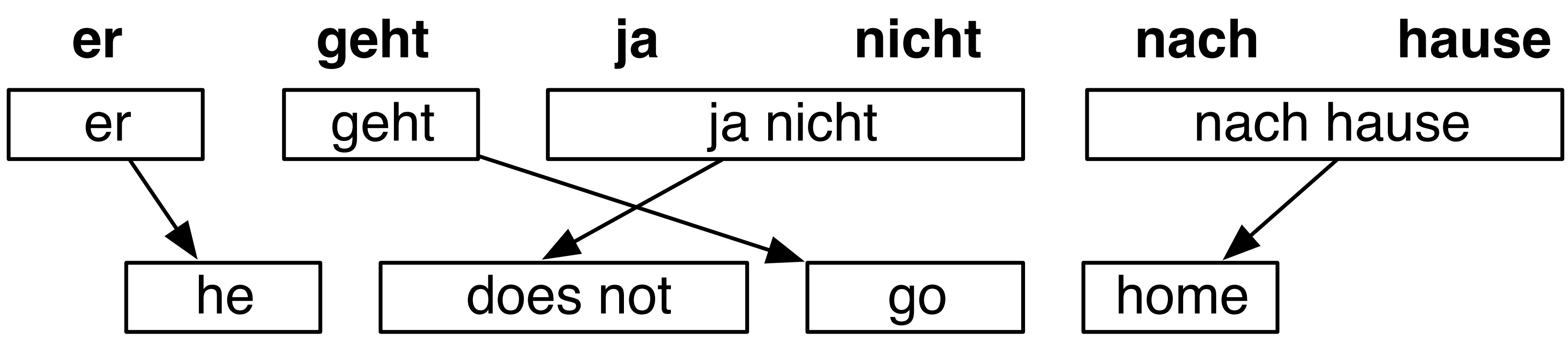
Step 1: Alignment





Step 1: Alignment

- We could spend an entire lecture on alignment models
- Not only single words but could use phrases, syntax
- Then consider reordering of translated phrases

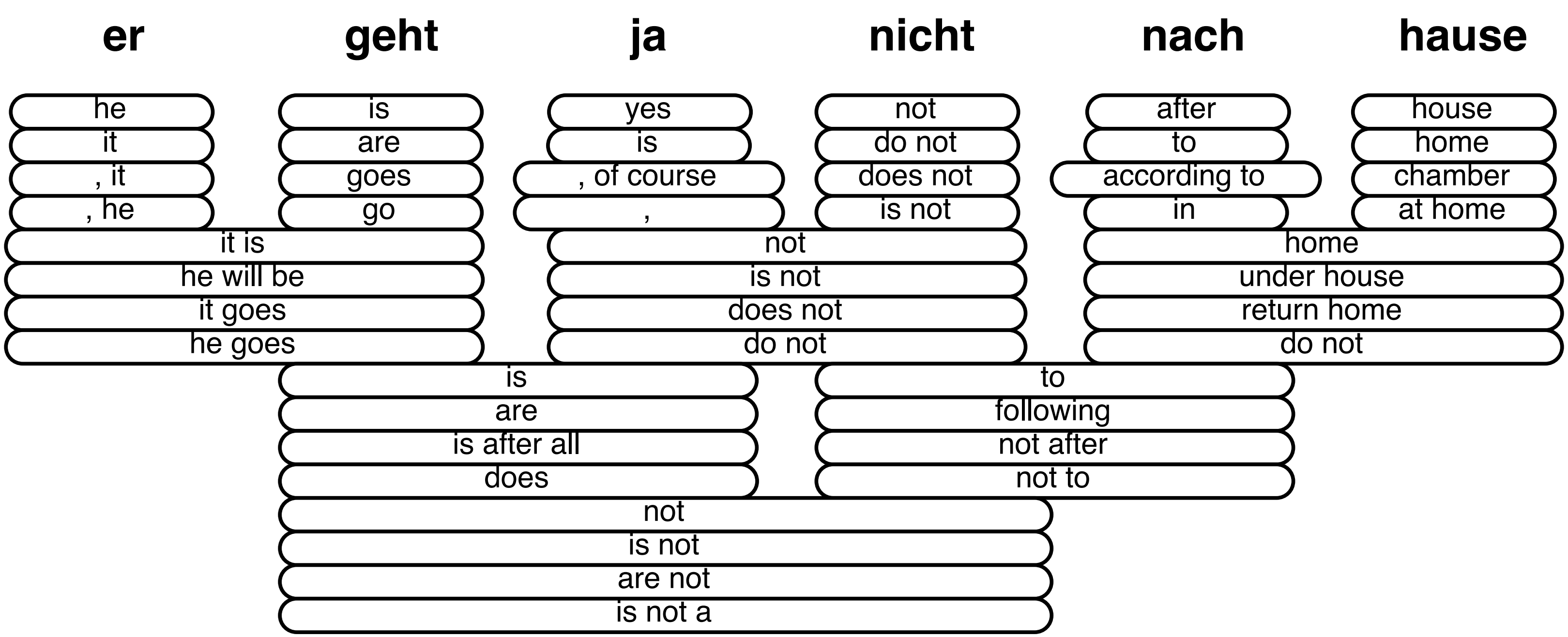


Example from Philipp Koehn

After many steps

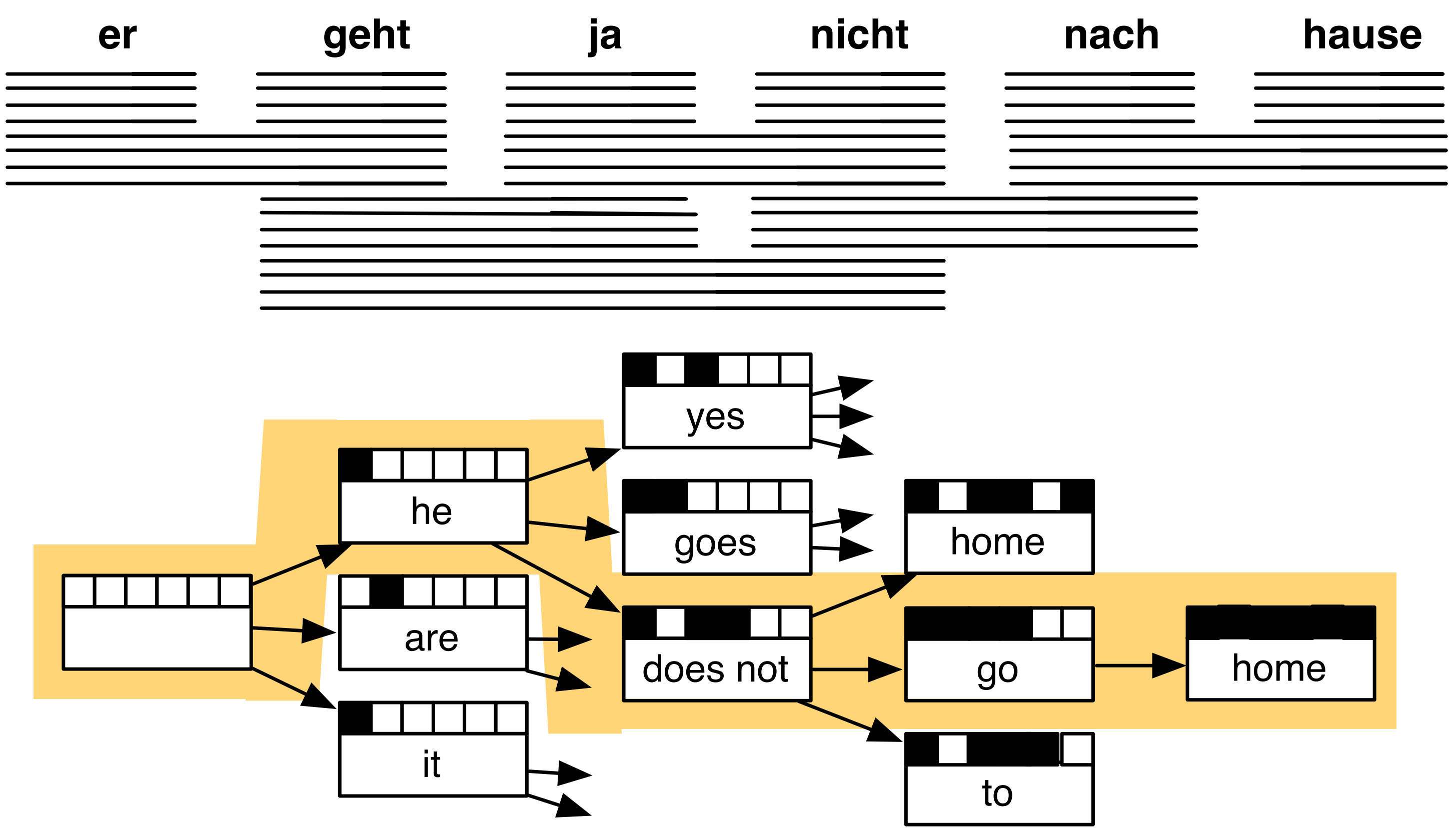
Each phrase in source language has many possible translations resulting in large search space:

Translation Options



Decode: Search for best of many hypotheses

Hard search problem that also includes language model



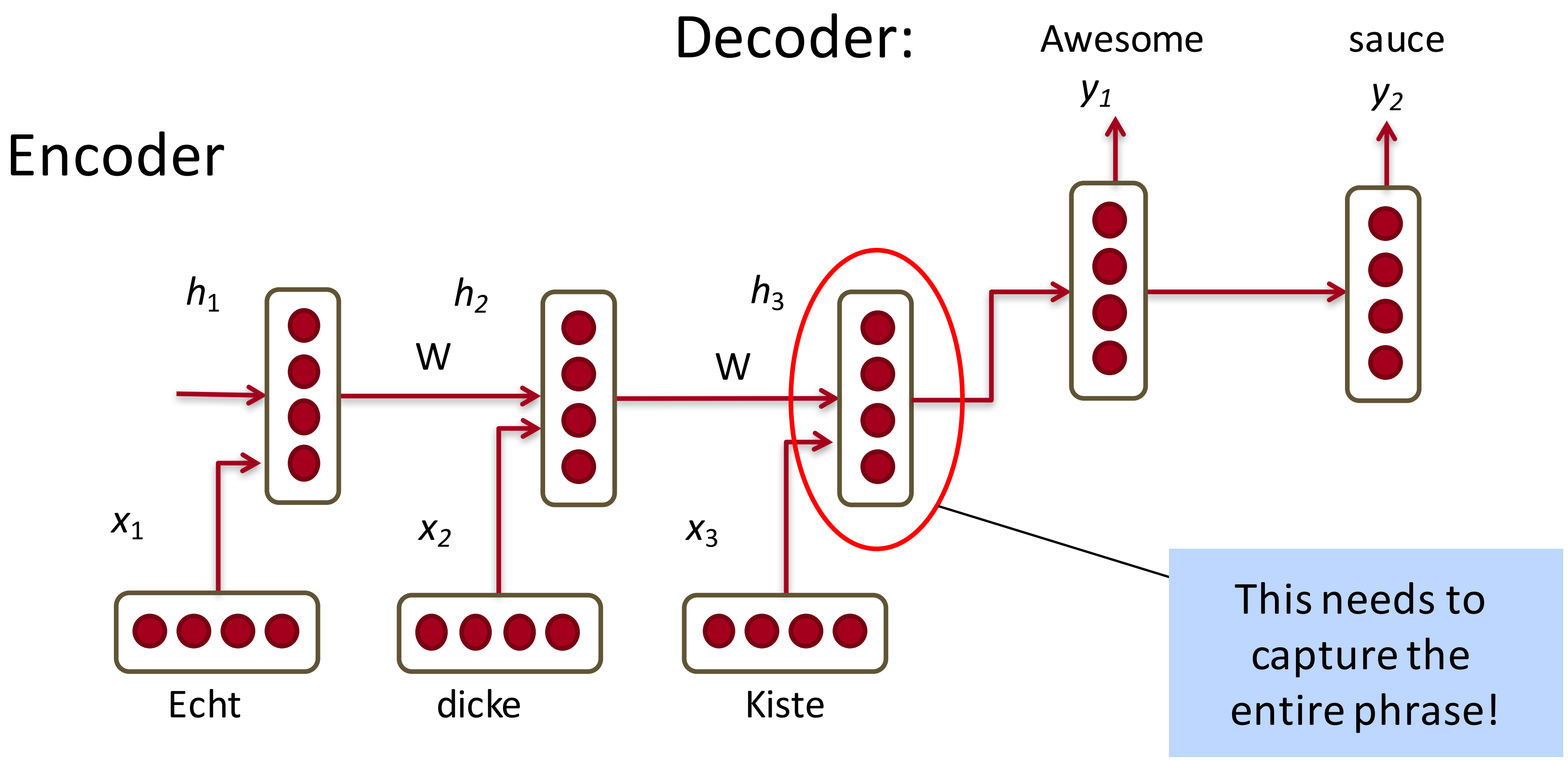
Traditional MT

- Skipped hundreds of important details
- **A lot** of human feature engineering
- Very complex systems

- Many different, independent machine learning problems

Deep learning to the rescue! … ?

Maybe, we could translate directly with an RNN?



MT with RNNs – Simplest Model

Encoder: $$h_t = \phi(h_{t-1}, x_t) = f\left(W^{(hh)} h_{t-1} + W^{(hx)} x_t\right)$$

Decoder: $$h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$$

$$y_t = softmax\left(W^{(S)} h_t\right)$$

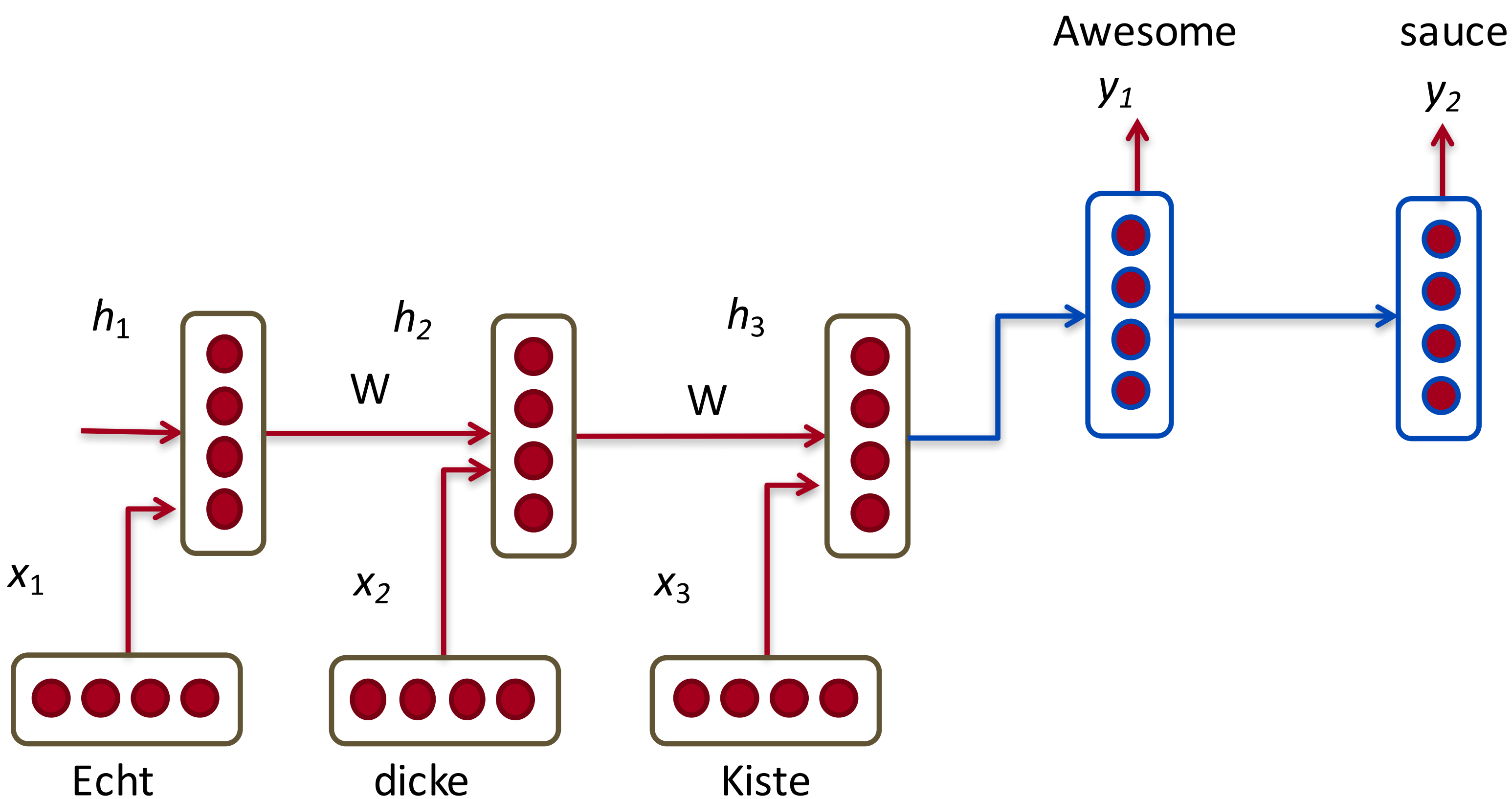
Minimize cross entropy error for all target words conditioned on source words

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log p_{\theta}(y^{(n)} | x^{(n)})$$

It's not quite that simple ;)

RNN Translation Model Extensions

1. Train different RNN weights for encoding and decoding

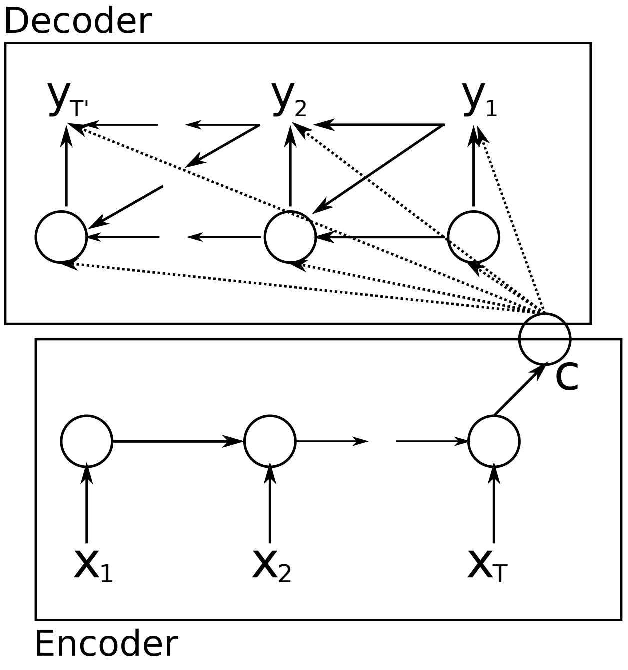


RNN Translation Model Extensions

Notation: Each input of Á has its own linear transformation matrix. Simple: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$

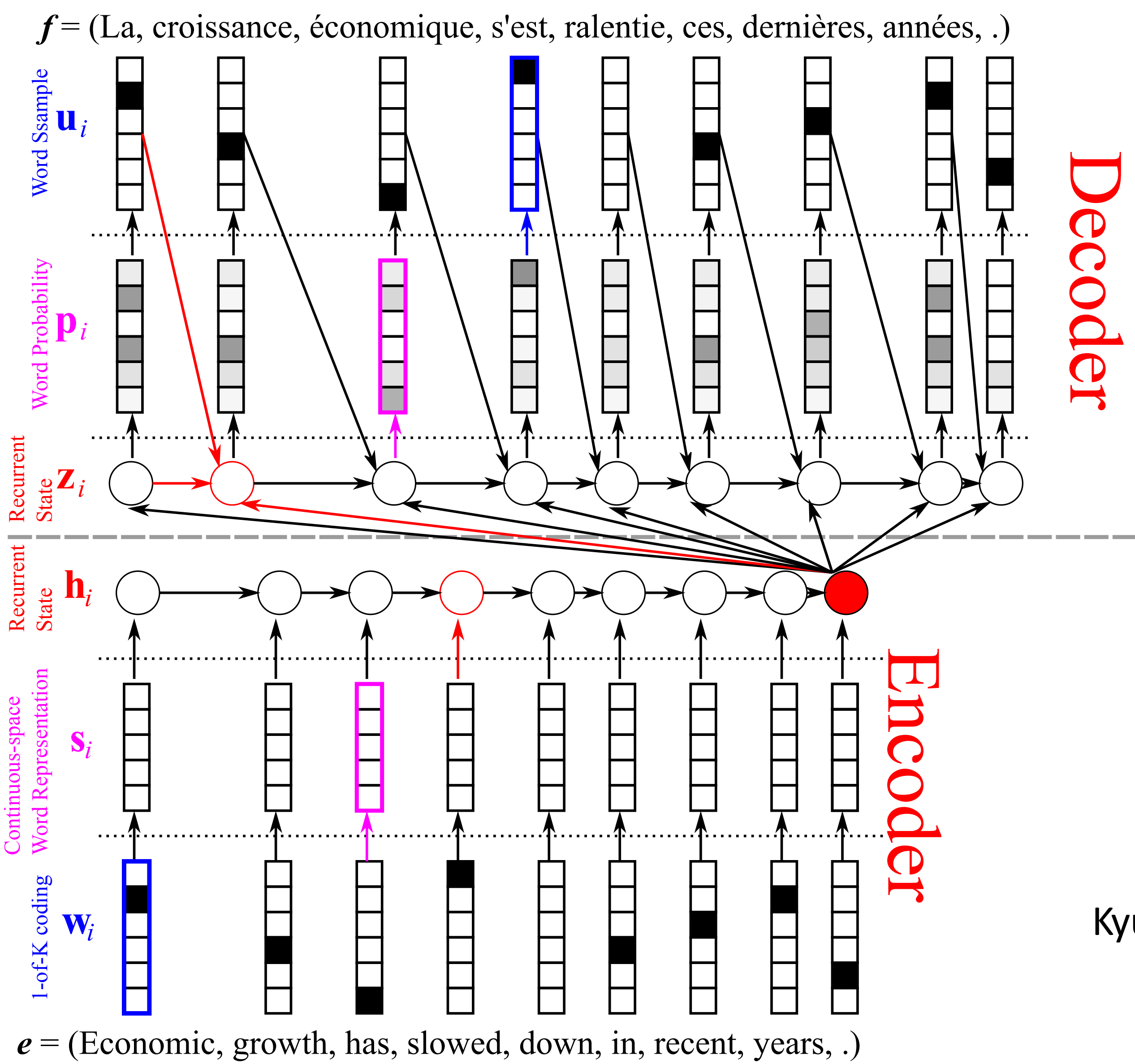
2. Compute every hidden state in decoder from
   - Previous hidden state (standard)
   - Last hidden vector of encoder $c=h_T$
   - Previous predicted output word y_{t-1}

$$h_{D,t} = \phi_D(h_{t-1}, c, y_{t-1})$$



Cho et al. 2014

Different picture, same idea

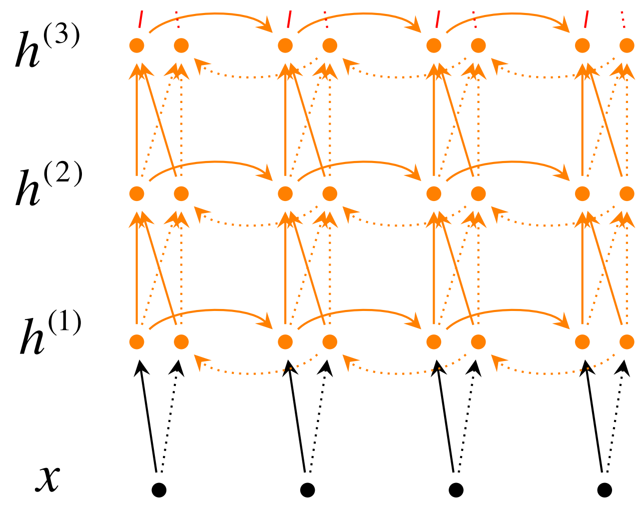


Kyunghyun Cho et al. 2014

RNN Translation Model Extensions

3. Train stacked/deep RNNs with multiple layers

4. Potentially train bidirectional encoder



5. Train input sequence in reverse order for simple optimization problem: Instead of A B C → X Y, train with C B A → X Y

6. Main Improvement: Better Units

- More complex hidden unit computation in recurrence!
- Gated Recurrent Units (GRU)
  introduced by Cho et al. 2014 (see reading list)
- Main ideas:
  - keep around memories to capture long distance dependencies
  - allow error messages to flow at different strengths depending on the inputs

GRUs

- Standard RNN computes hidden layer at next time step directly: $h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$
- GRU first computes an update **gate** (another layer) based on current input word vector and hidden state

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

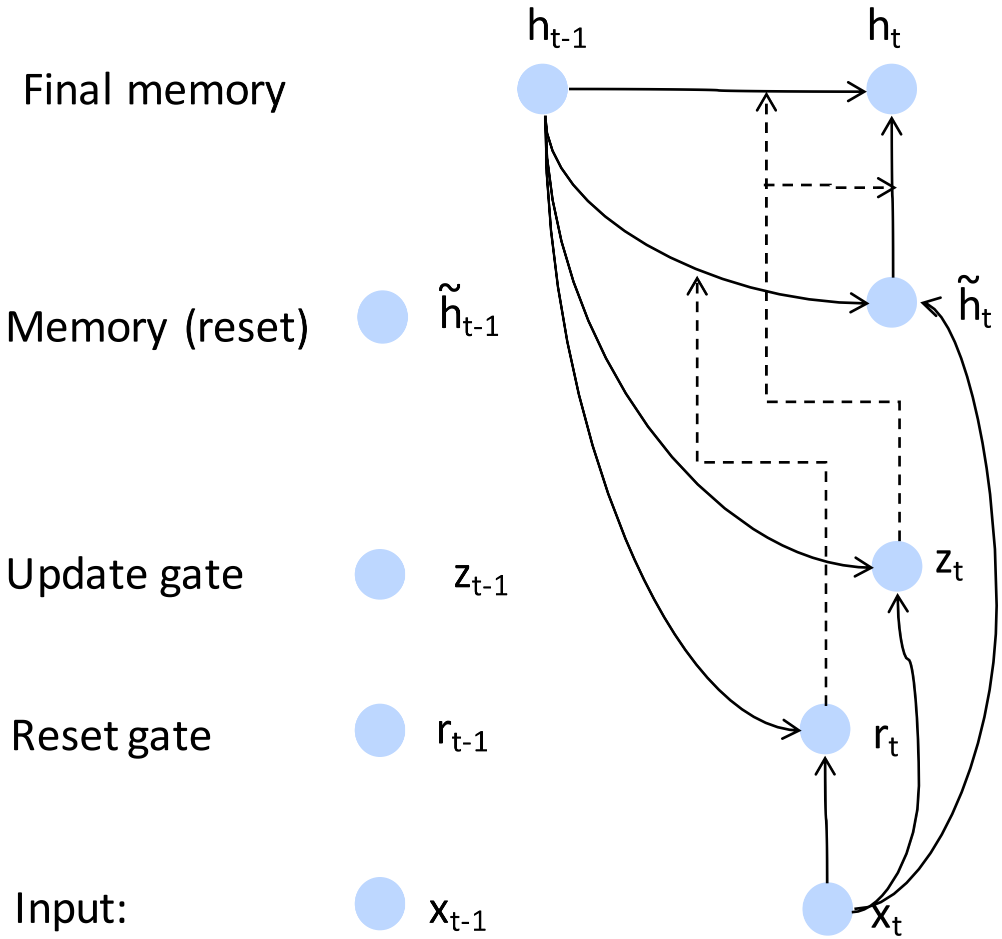
- Compute reset gate similarly but with different weights

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

GRUs

- Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$
- Reset gate $r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$
- New memory content: $\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$
  If reset gate unit is ~0, then this ignores previous memory and only stores the new word information
- Final memory at time step combines current and previous time steps: $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

Attempt at a clean illustration



$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$

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

GRU intuition

- If reset is close to 0, ignore previous hidden state → Allows model to drop information that is irrelevant in the future

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

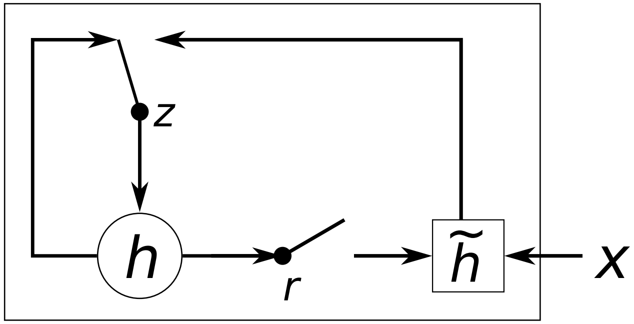
- Update gate z controls how much of past state should matter now.
  - If z close to 1, then we can copy information in that unit through many time steps! **Less vanishing gradient**!
- Units with short-term dependencies often have reset gates very active

GRU intuition

- Units with long term dependencies have active update gates z

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

- Illustration:

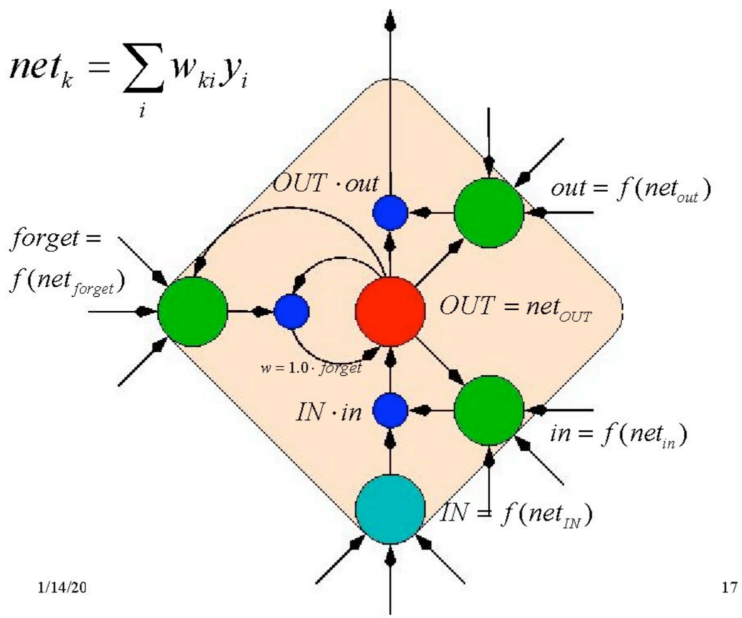


- Derivative of $\frac{\partial}{\partial x_1} x_1 x_2$? → rest is same chain rule, but implement with **modularization** or automatic differentiation

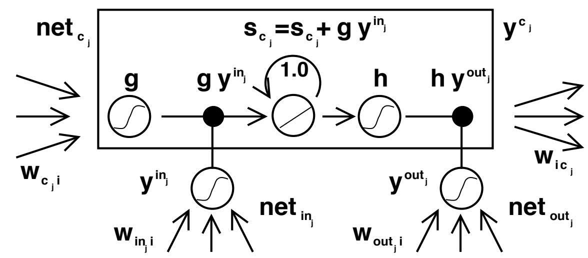
Long-short-term-memories (LSTMs)

- We can make the units even more complex
- Allow each time step to modify
  - Input gate (current cell matters) $i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$
  - Forget (gate 0, forget past) $f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$
  - Output (how much cell is exposed) $o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$
  - New memory cell $\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$
- Final memory cell: $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$
- Final hidden state: $h_t = o_t \circ \tanh(c_t)$

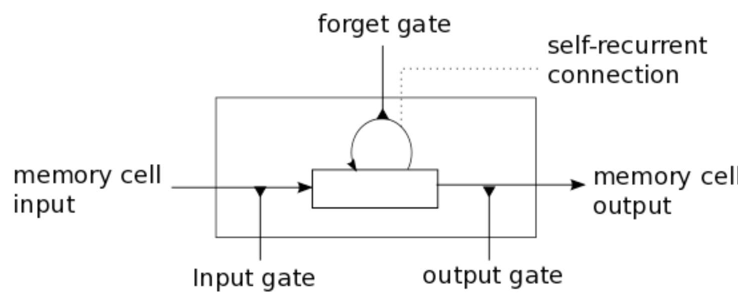
Illustrations all a bit overwhelming ;)



http://people.idsia.ch/~juergen/lstm/sld017.htm



Long Short-Term Memory by Hochreiter and Schmidhuber (1997)



http://deeplearning.net/tutorial/lstm.html

Intuition: memory cells can keep information intact, unless inputs makes them forget it or overwrite it with new input.

Cell can decide to output this information or just store it

LSTMs are currently very hip!

- En vogue default model for most sequence labeling tasks

- Very powerful, especially when stacked and made even deeper (each hidden layer is already computed by a deep internal network)

- Most useful if you have lots and lots of data

Deep LSTMs compared to traditional systems 2015

Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

Table 1: The performance of the LSTM on WMT'14 English to French test set (ntst14). Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than of a single LSTM with a beam of size 12.

Method	test BLEU score (ntst14)
Baseline System [29]	33.30
Cho et al. [5]	34.54
Best WMT'14 result [9]	**37.0**
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	**36.5**
Oracle Rescoring of the Baseline 1000-best lists	~45

Sequence to Sequence Learning by Sutskever et al. 2014

Deep LSTMs (with a lot more tweaks) today

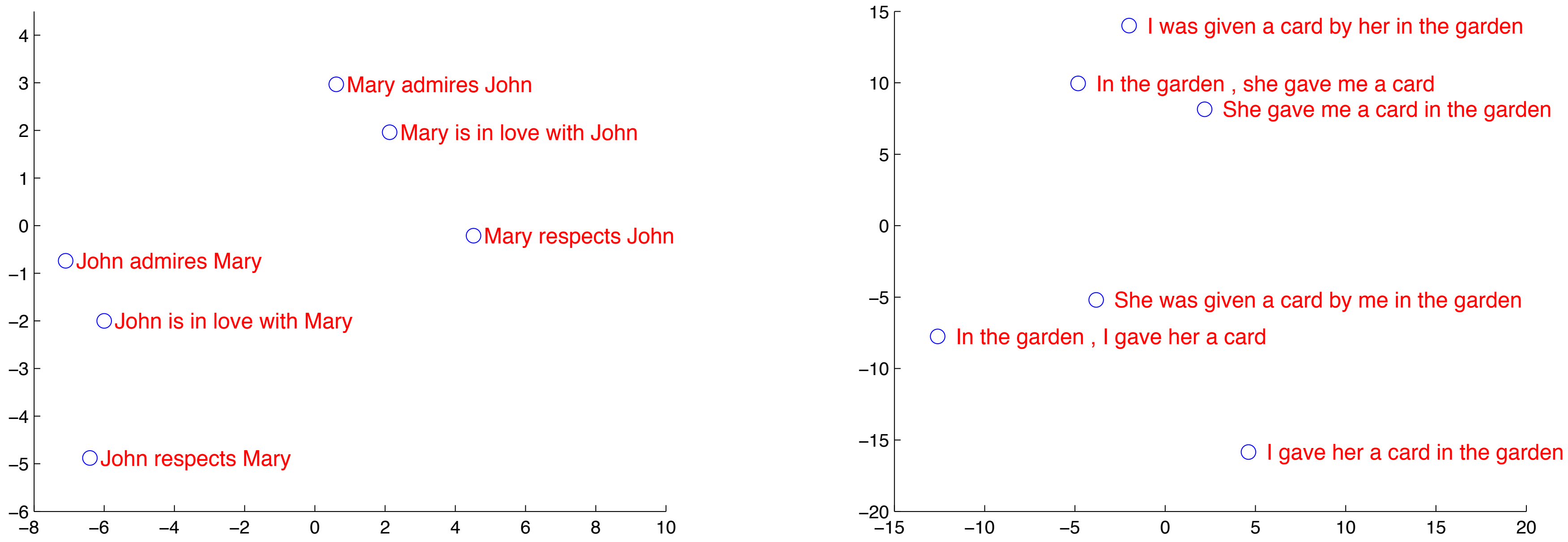
WMT 2016 competition results from yesterday

Scored Systems

System	Submitter	System Notes	Constraint	Run Notes	BLEU
uedin-nmt-ensemble (Details)	rsennrich *University of Edinburgh*	BPE neural MT system with monolingual training data (back-translated). ensemble of 4, reranked with right-to-left model.	yes		34.8
metamind-ensemble (Details)	jekbradbury *Salesforce MetaMind*	Neural MT system based on Luong 2015 and Sennrich 2015, using Morfessor for subword splitting, with back-translated monolingual augmentation. Ensemble of 3 checkpoints from one run plus 1 Y-LSTM (see entry).	yes		32.8
uedin-nmt-single (Details)	rsennrich *University of Edinburgh*	BPE neural MT system with monolingual training data (back-translated). single model. (contrastive)	yes		32.2
KIT (Details)	niehues *KIT*	Phrase-based MT with NMT in rescoring	yes		29.7
uedin-pbt-wmt16-en-de (Details)	Matthias Huck *University of Edinburgh*	Phrase-based Moses	yes		29.1
Moses Phrase-Based (Details)	jhu-smt *Johns Hopkins University*	Phrase-based model, word clusters for all model components (LM, OSM, LR, sparse features), neural network joint model, large cc LM	yes	[26-7]	29.0
uedin-pbt-wmt16-en-de-contrastive (Details)	Matthias Huck *University of Edinburgh*	Phrase-based Moses (contrastive, 2015 system)	yes		29.0

Deep LSTM for Machine Translation

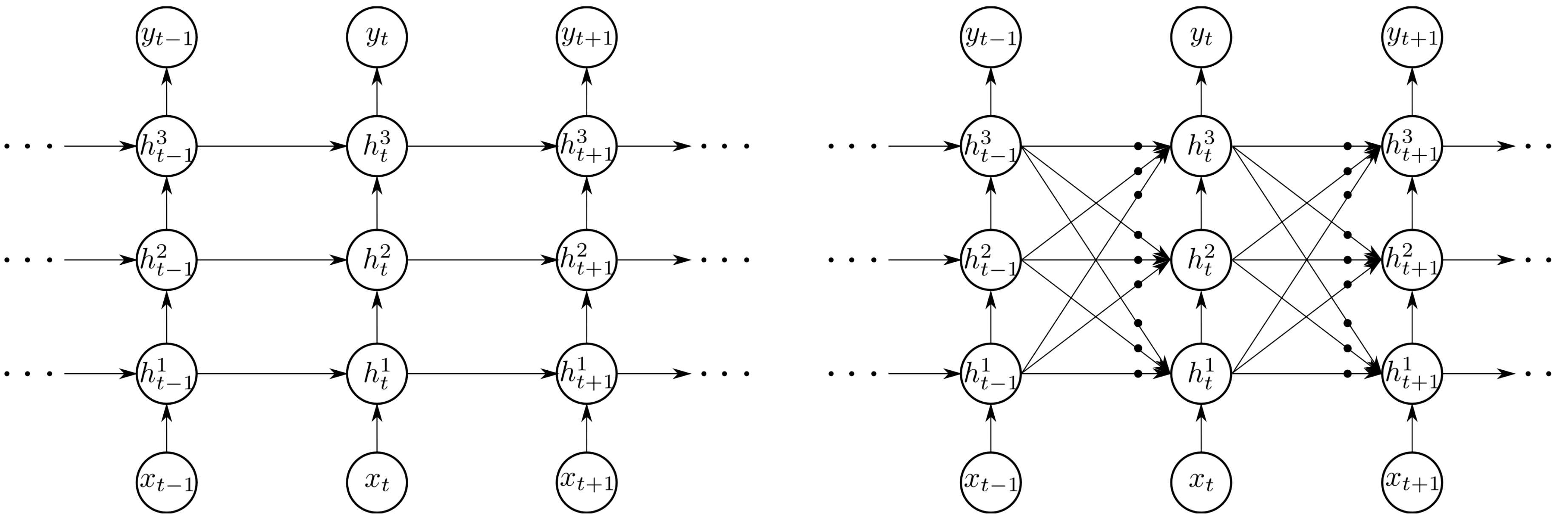
PCA of vectors from last time step hidden layer



Sequence to Sequence Learning by Sutskever et al. 2014

Further Improvements: More Gates!

Gated Feedback Recurrent Neural Networks, Chung et al. 2015



(a) Conventional stacked RNN (b) Gated Feedback RNN

Summary

- Recurrent Neural Networks are powerful
- A lot of ongoing work right now
- Gated Recurrent Units even better
- LSTMs maybe even better (jury still out)
- This was an advanced lecture → gain intuition, encourage exploration
- Next up: Recursive Neural Networks
  simpler and also powerful :)