# Neural Machine  Translation by Jointly Learning to Align and Translate


Dzmitry Bahdanau
KyungHyun Cho
Yoshua Bengio
@ICLR 2014

Presented By
Roee Aharoni

# Machine Translation is Everywhere




# But there’s still much work to do…


Sixi roasted husband

Meat Muscle Stupid Bean Sprouts

# Lets start with a Live Demo

http://104.131.78.120/

# “Traditional” Statistical Machine Translation

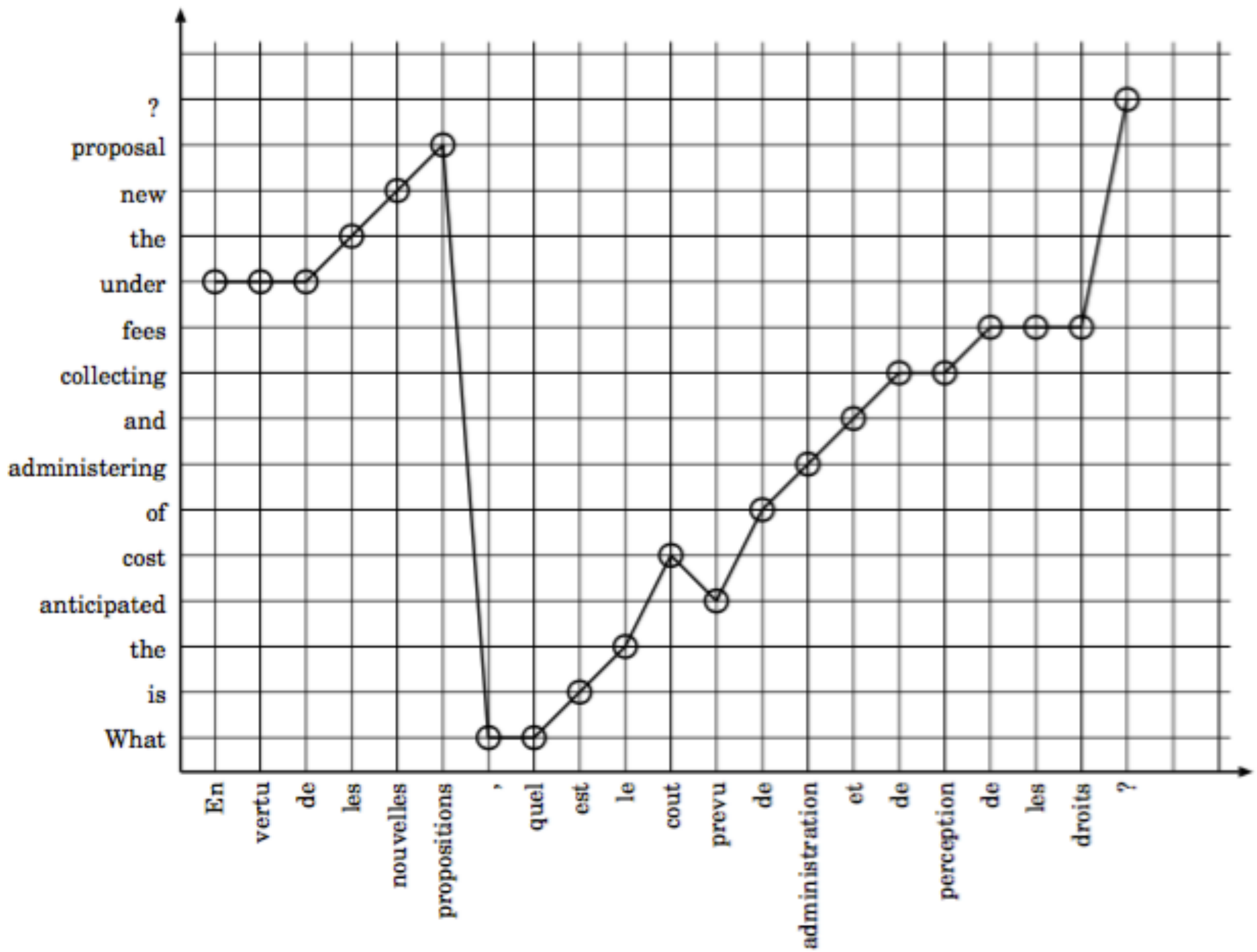
Start with (lots) of parallel text:

```
1a.  ok-voon ororok sprok .

1b.  at-voon bichat dat .

2a.  ok-drubel ok-voon anok plok sprok .

2b.  at-drubel at-voon pippat rrat dat .

3a.  erok sprok izok hihok ghirok .

3b.  totat dat arrat vat hilat .

4a.  ok-voon anok drok brok jok .

4b.  at-voon krat pippat sat lat .

5a.  wiwok farok izok stok .

5b.  totat jjat quat cat .
```

```
6a.  lalok sprok izok jok stok .

6b.  wat dat krat quat cat .

7a.  lalok farok ororok lalok sprok izok enemok .

7b.  wat jjat bichat wat dat vat eneat .

8a.  lalok brok anok plok nok .

8b.  iat lat pippat rrat nnat .

9a.  wiwok nok izok kantok ok-yurp .

9b.  totat nnat quat oloat at-yurp .

10a. lalok mok nok yorok ghirok clok .

10b. wat nnat gat mat bat hilat .
```

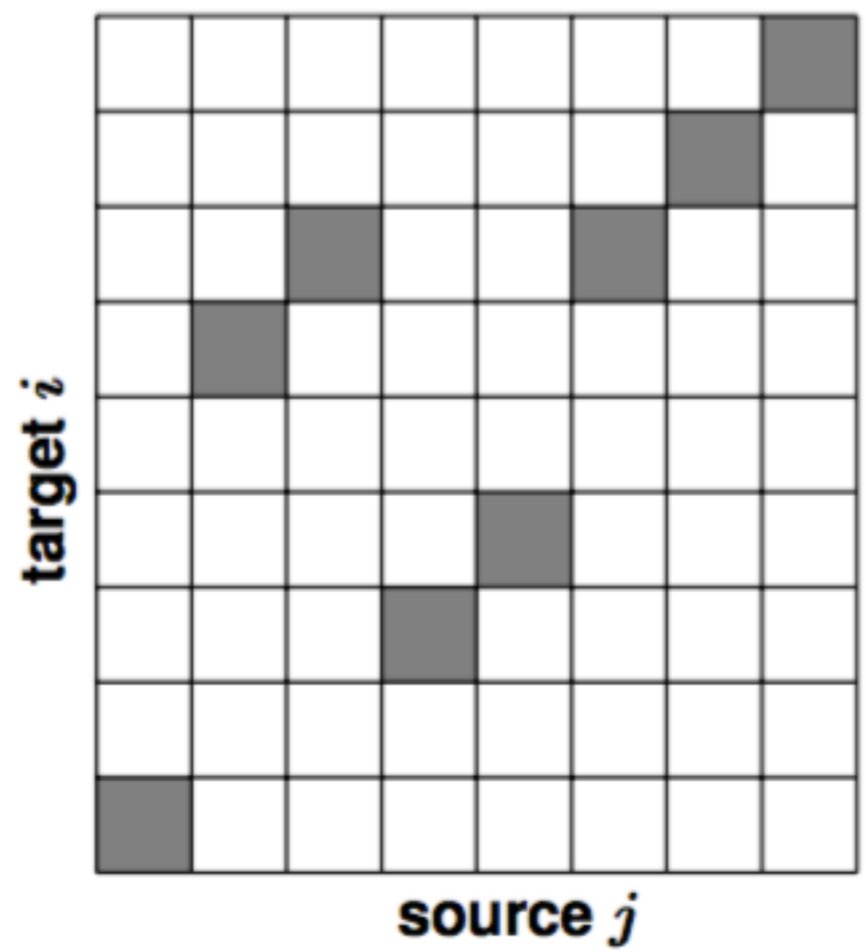
# “Conventional” Statistical Machine Translation

Learn the alignments (using the EM algorithm):



# “Conventional” Statistical Machine Translation

Learn the alignments (using the EM algorithm):




**mapping:** $j \rightarrow i = a_j$

# “Conventional” Statistical Machine Translation

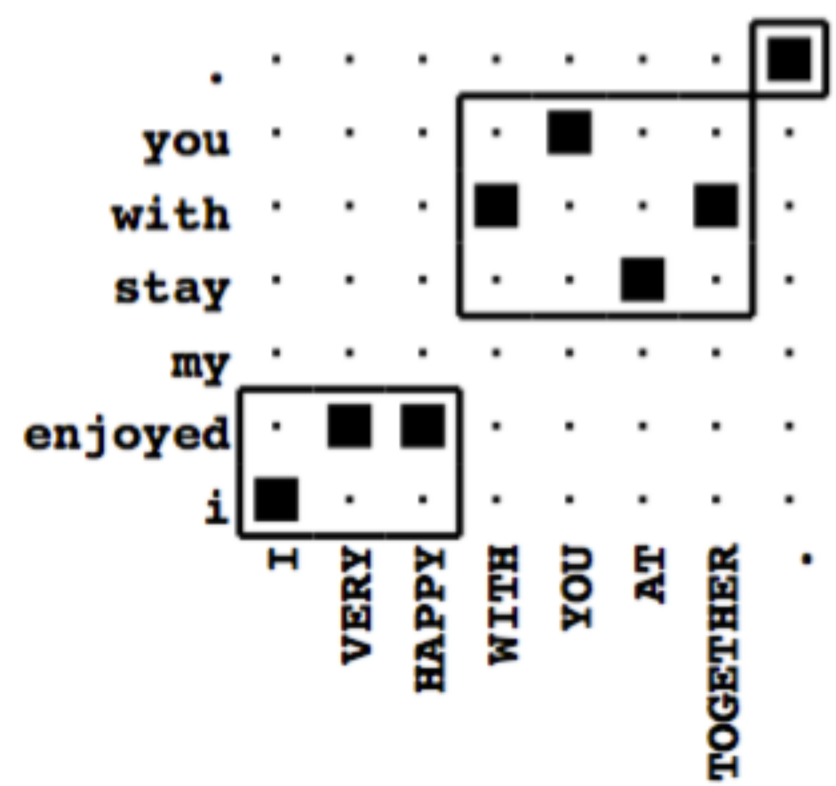
Extract phrase pairs:

**source sentence** 我 很 高兴 和 你 在 一起 .

**gloss notation** **I VERY HAPPY WITH YOU AT TOGETHER .**

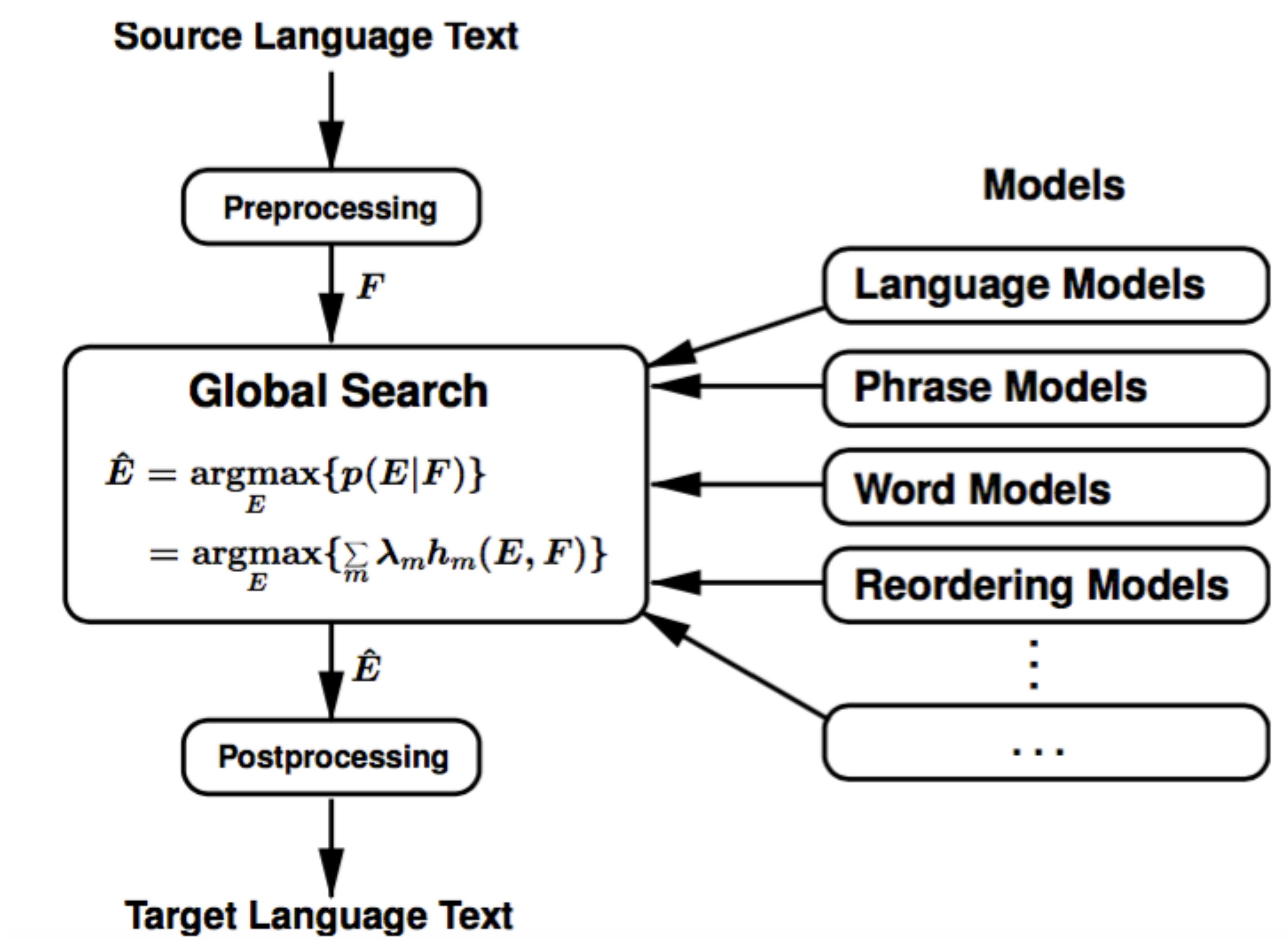
**target sentence** **I enjoyed my stay with you .**

**Viterbi alignment for $F \to E$:**



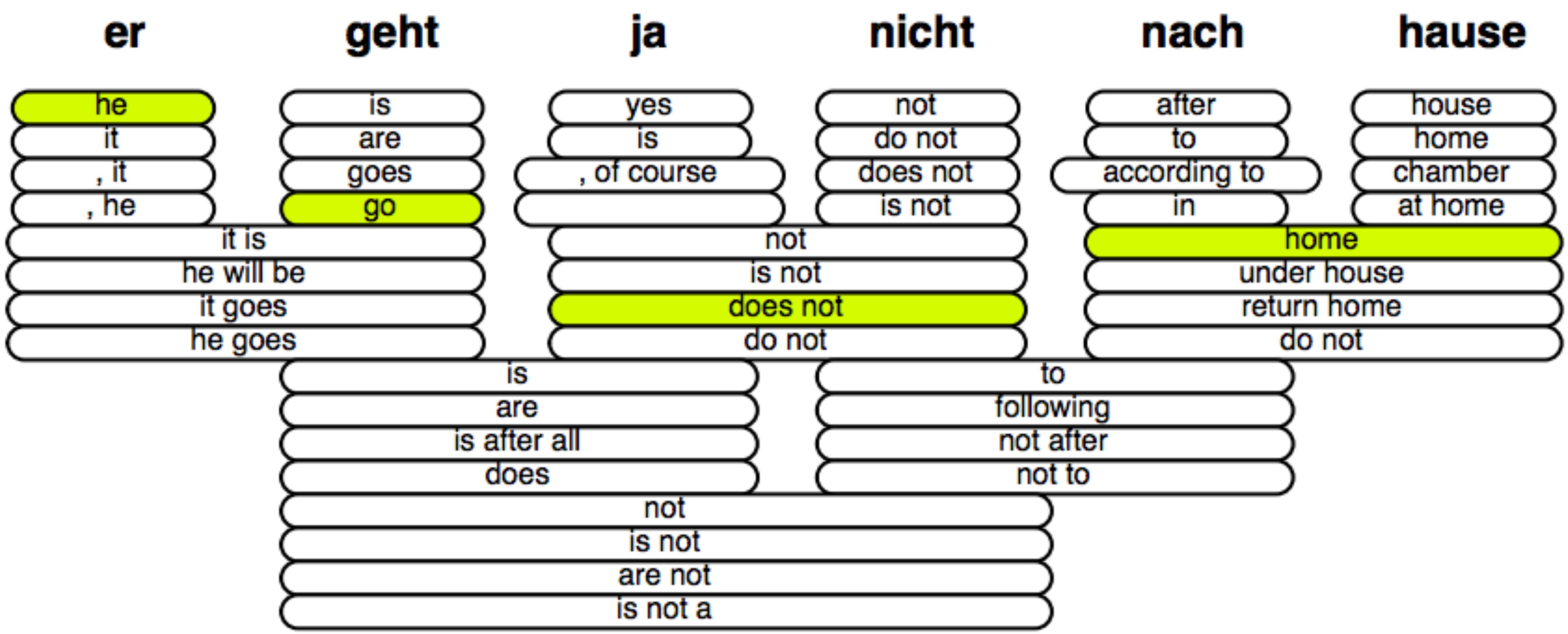
# “Conventional” Statistical Machine Translation

Use a log linear model combination to **score** possible hypotheses:



# “Conventional” Statistical Machine Translation

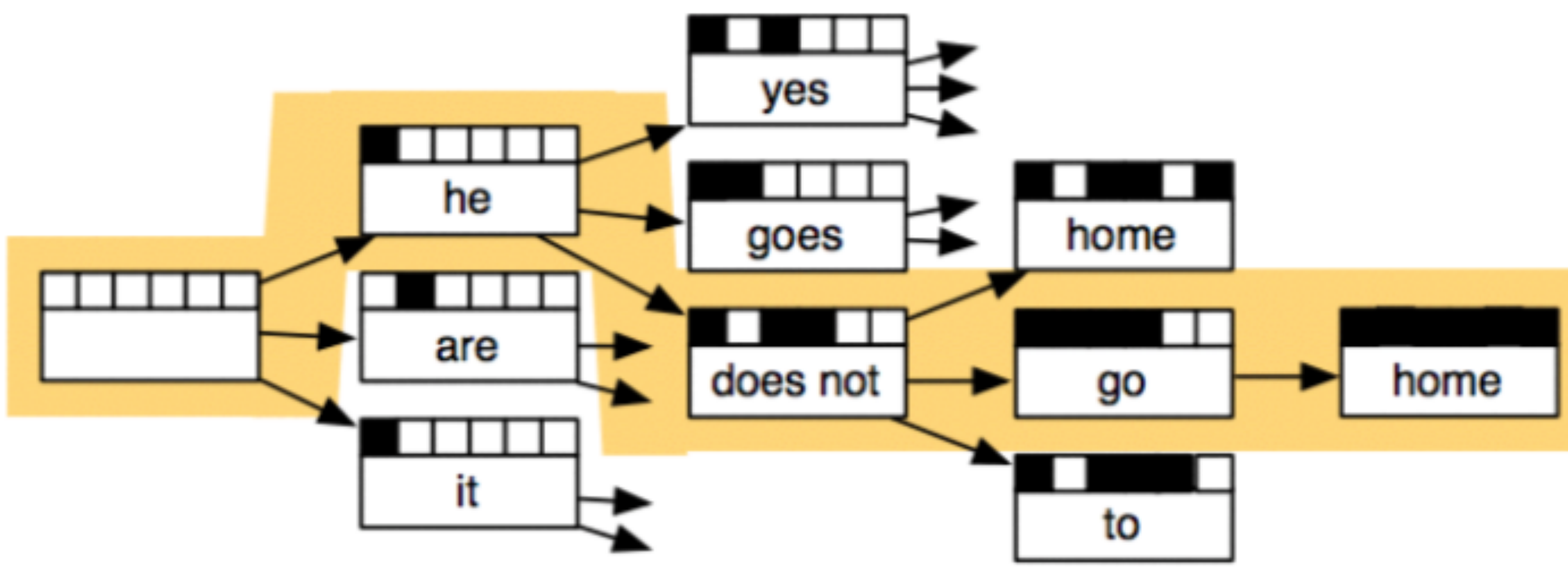
Use a log linear model combination to score **possible hypotheses**:




- Many translation options to choose from
  - in Europarl phrase table: 2727 matching phrase pairs for this sentence
  - by pruning to the top 20 per phrase, 202 translation options remain

# “Conventional” Statistical Machine Translation

Use beam search to output the best hypothesis:




backtrack from highest scoring complete hypothesis

- Learn Alignment

- Extract phrases

- Extract some mo

- (And then some


- Train one combined model using all the above + large language model: estimate **p(f|e) as p(e|f)p(f)**

translation model language model

- Run search on top of that

# So let's try another approach

- Maybe we can just estimate **p(f|e)** directly?
- For that, we need to know -

# What is deep learning?

“A family of learning methods that use deep architectures to learn high-level feature representations”

# What is deep learning?

“A family of **learning methods** that use **deep architectures** to learn **high-level** feature representations”

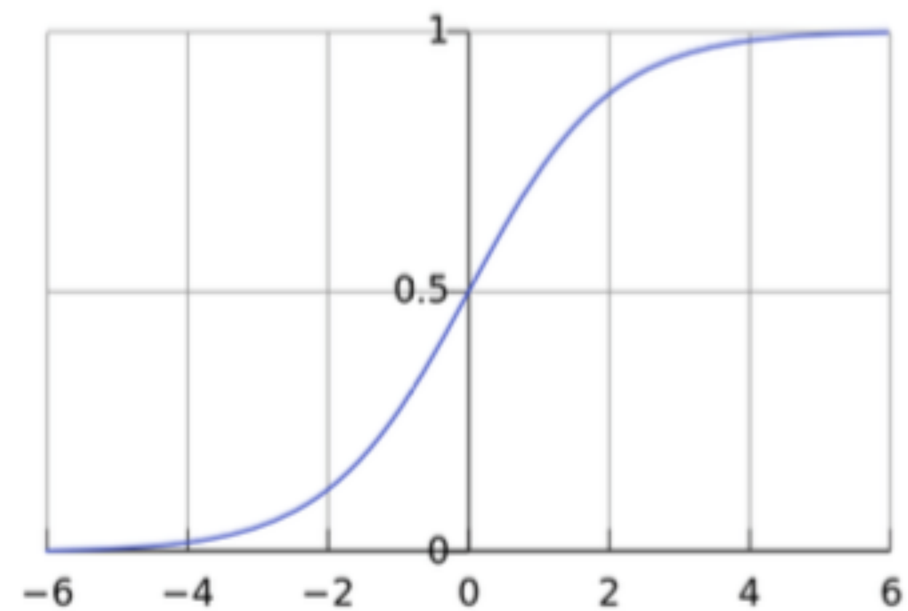
# A basic machine learning setup

- Given a dataset of: $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$ training examples,
  - input: $x^{(m)} \in R^d$
  - output: $y^{(m)} = \{0, 1\}$
- Learn a function $f : x \rightarrow y$ to predict correctly on new inputs.
  - step I: pick a learning algorithm (SVM, log. reg., NN…)
  - step II: optimize it w.r.t a loss, i.e: $\mathrm{min}_w \sum_{m=1}^{M} (f_w(x^{(m)}) - y^{(m)})^2$

# Logistic regression - the “1-layer” network

- Model the classier as: $f(x) = \sigma(w^T \cdot x)$
- Learn the weight vector: $w \in R^d$ using gradient-descent (next slide)
- $\sigma$ is a non-linearity, e.g. the sigmoid function:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



# Training (log. regression) with gradient-descent

- Define the loss-function (squared error, cross entropy…):

$$Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$$

- Derive the loss-function w.r.t. the weight vector, w:

$$\nabla_w Loss = \sum_m \left[\sigma(w^T x^{(m)}) - y^{(m)}\right] \sigma'(w^T x^{(m)}) x^{(m)}$$

- Perform gradient-descent:
    - start with a random weight vector
    - repeat until convergence: $w \leftarrow w - \gamma(\nabla_w Loss)$

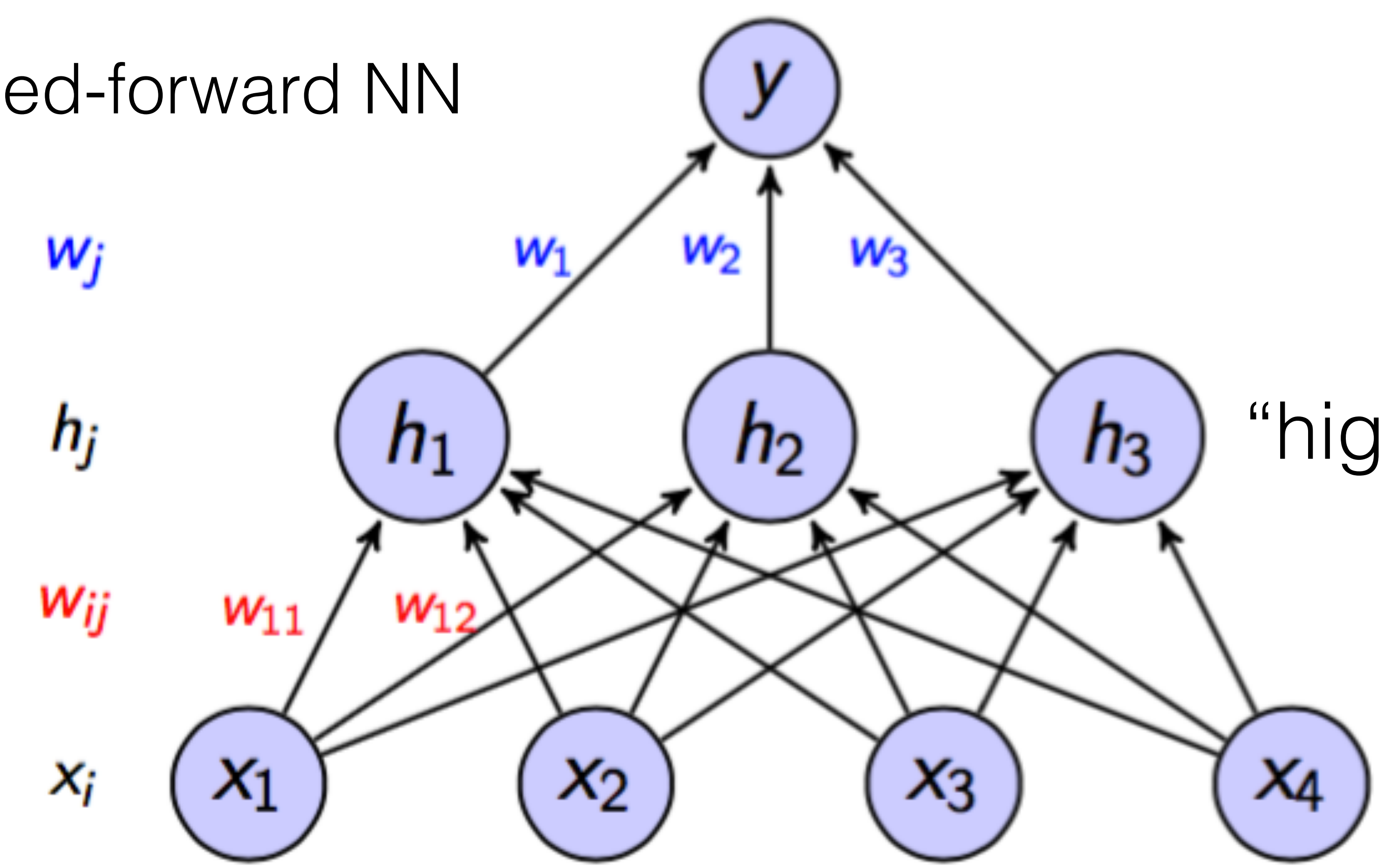
# Multi layer perceptron (MLP) - a multi-layer NN

- Model the classifier as:

$$f(x) = \sigma(\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

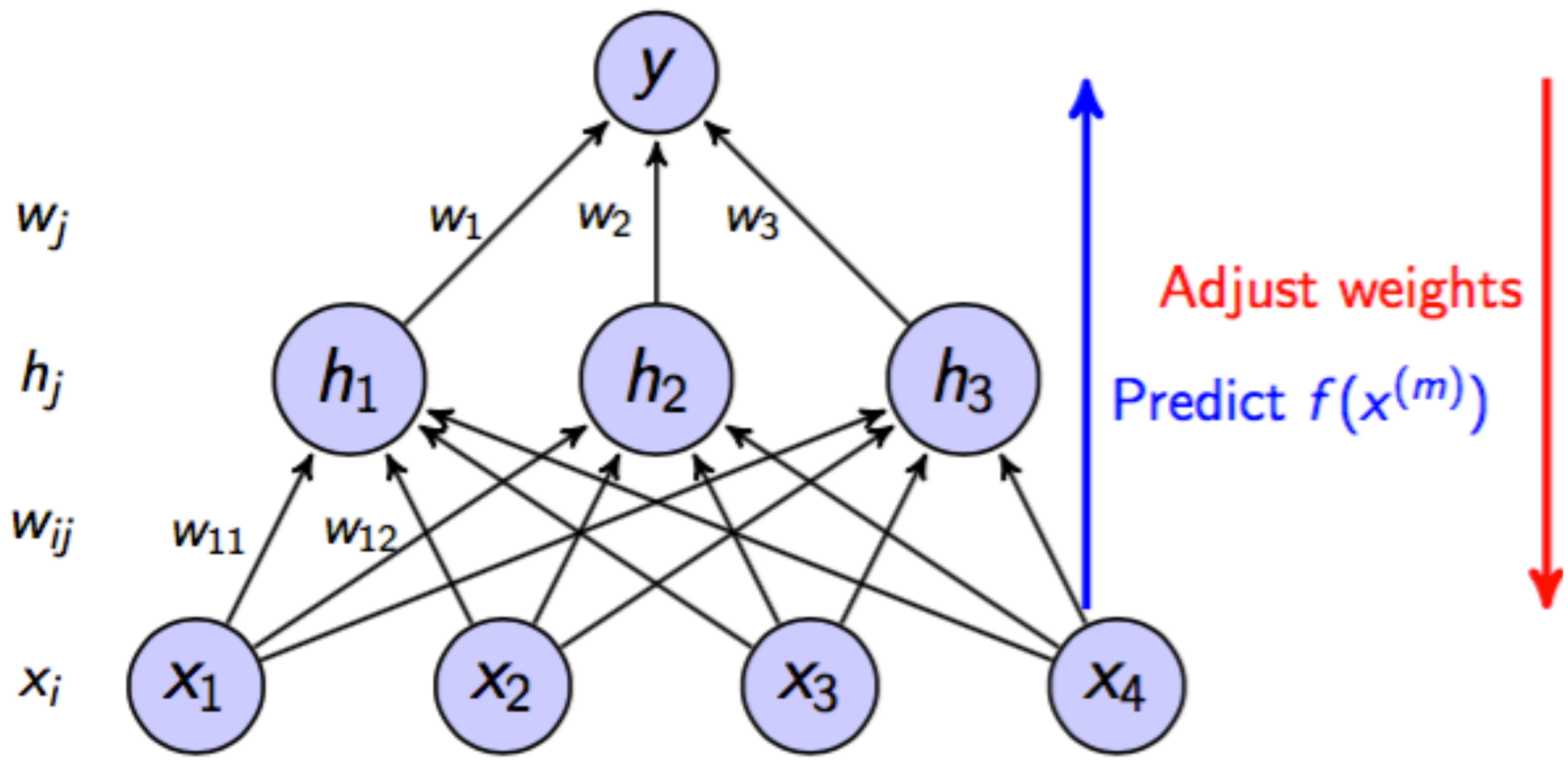
- Can be seen as multilayer logistic regression

- a.k.a feed-forward NN

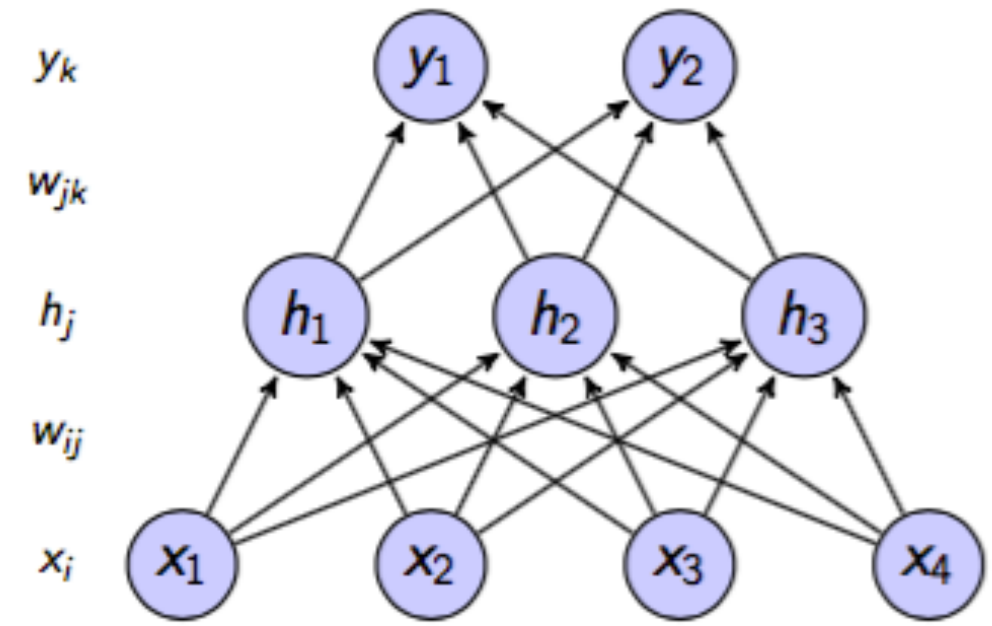



"high level" features

# Training (an MLP) with Backpropagation:



# Training (an MLP) with Backpropagation:




- Assume two outputs per input:
- Define the loss-function per example:

$$Loss = \sum_k \frac{1}{2} \left[\sigma(in_k) - y_k\right]^2$$

- Derive the loss-function w.r.t. the last layer:

$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k} \frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial(\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$

- Derive the loss function w.r.t. the first layer:

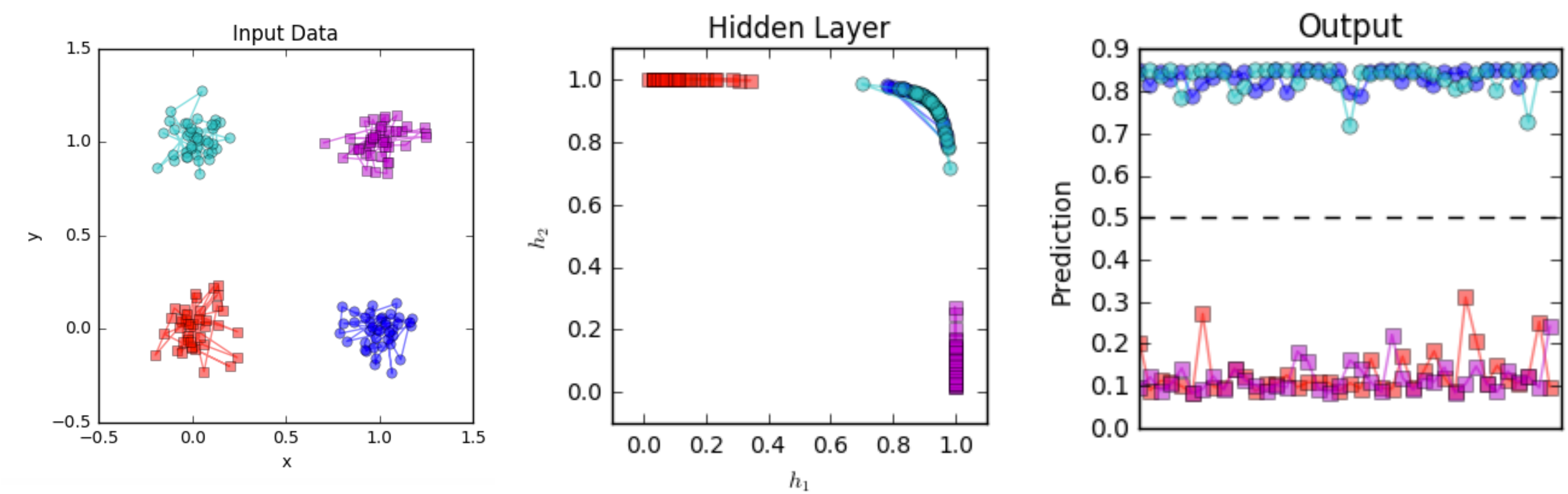
$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial(\sum_i w_{ij} x_i)}{\partial w_{ij}} = \delta_j x_i$$

- Update the weights: $w \leftarrow w - \gamma(\nabla_w Loss)$

# Why deeper is better?

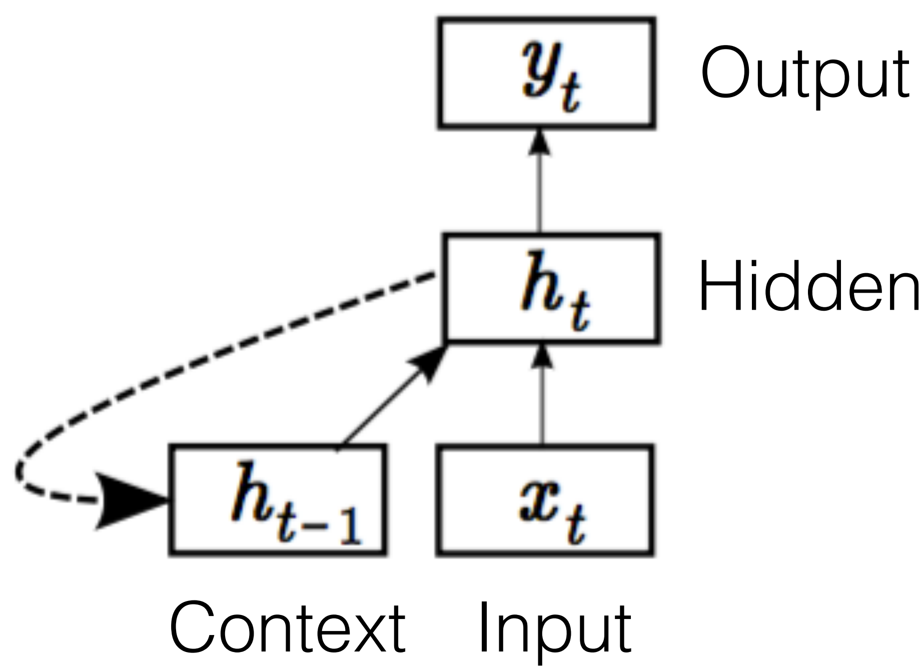
- A deeper architecture is more expressive than a shallow one given same number of nodes [Bishop, 1995]
  - 1-layer nets (log. regression) can only model linear hyperplanes
  - 2-layer nets can model any continuous function (given sufficient nodes)
  - >3-layer nets can do so with fewer nodes

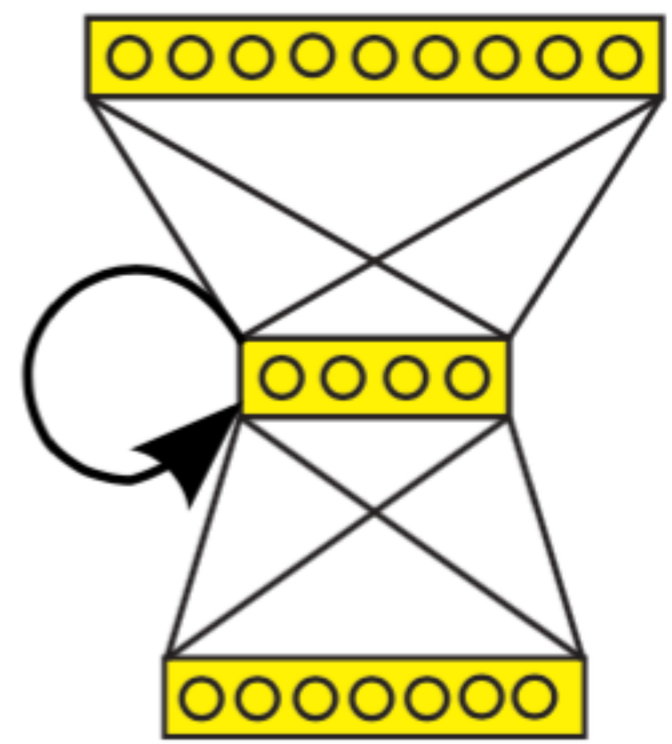
**Example - the XOR problem:**



# Recurrent Neural Networks (RNN's)

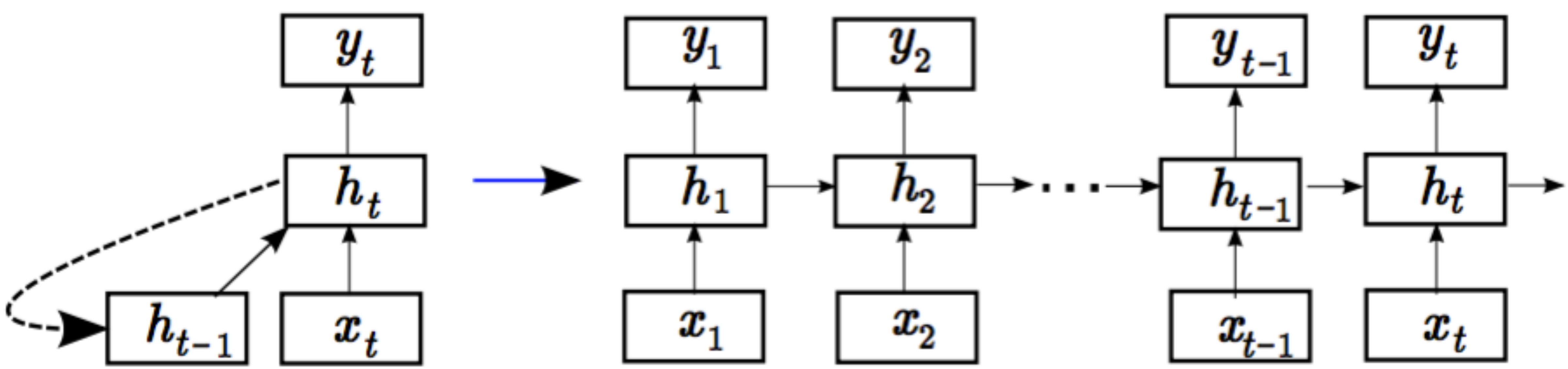
- Enable variable length inputs (sequences)
- Modeling internal structure in the input or output
- Introduce a "memory/context" component to utilize history



# Recurrent Neural Networks (RNN's)

- "Horizontally deep" architecture
- Recurrence equations:
  - Transition function: $h_t = H(h_{t-1}, x_t) = tanh(Wx_{t-1} + Uh_{t-1} + b)$
  - Output function: $y_t = Y(h_t)$, usually implemented as softmax



# The Softmax Function

- Enables to output a probability distribution over k possible classes
- can be seen as trying to minimize the cross-entropy between the predictions and the truth
- $y_i$ usually holds log-likelihood values

$$p(x = i) = \frac{e^{y_i}}{\sum_{j=1}^{k} e^{y_j}}$$

# Training (RNN's) with Backpropagation Through Time

- As usual, define a loss function (per sample, through time $t = 1, 2, ..., T$):

$$Loss = J(\Theta, x) = -\sum_{t=1}^{T} J_t(\Theta, x_t)$$

- Derive the loss function w.r.t. parameters $\Theta$, starting at $t = T$:

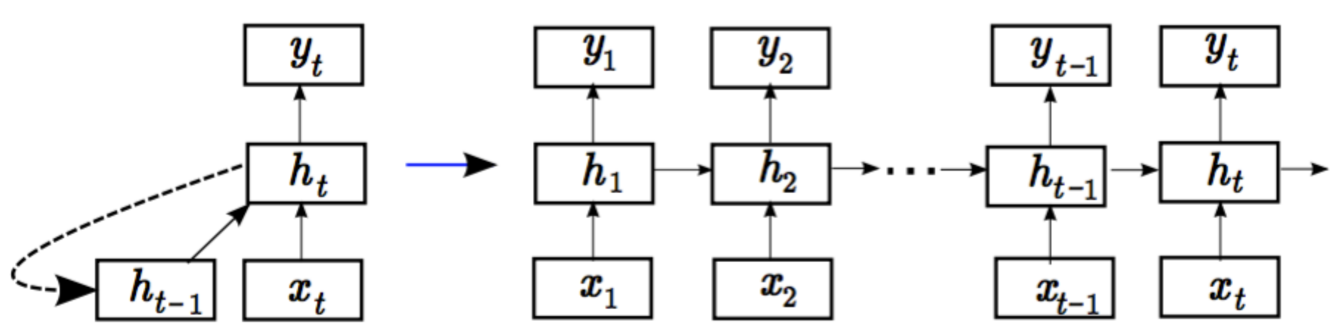
$$\nabla\Theta = \frac{\partial J_t}{\partial \Theta}$$

- Backpropagate through time - update and repeat for $t - 1$, until $t = 1$:

$$\nabla\Theta = \nabla\Theta + \frac{\partial J_t}{\partial \Theta}$$

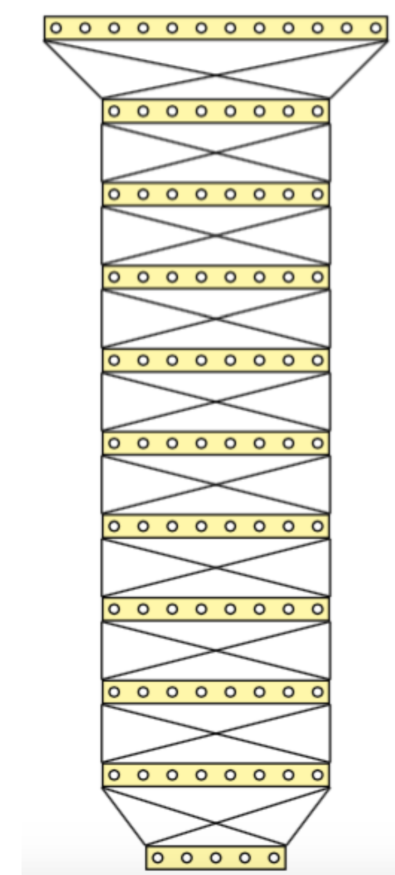
- Eventually, update the weights:

$$\Theta = \gamma \nabla\Theta$$



# Why now? Today vs. 80’s-90’s

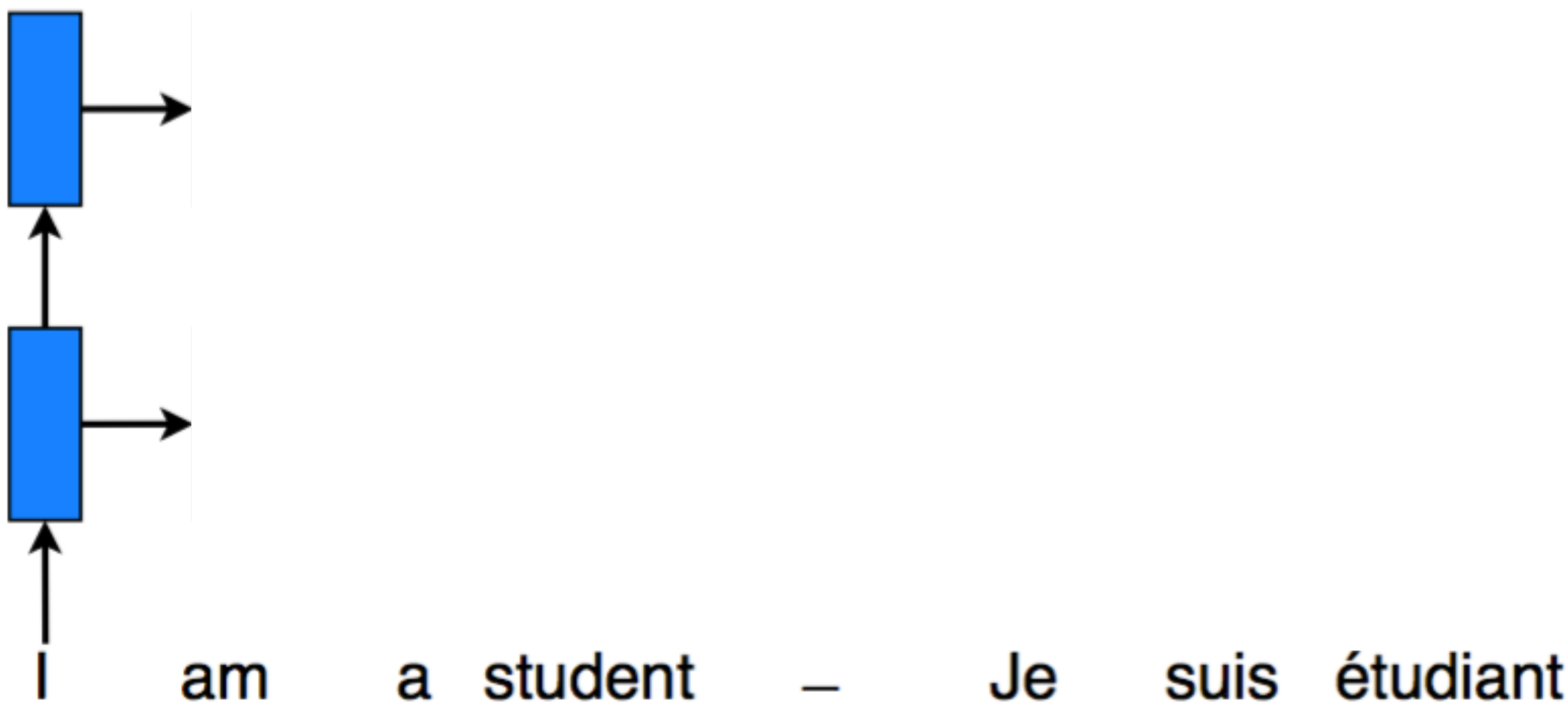
- Number of hidden layers: 10 (or more) rather than 2-3
- Number of output nodes: 5000 (or more) rather then 50
- Better optimization strategies, heuristics (layer-by-layer pre-training, dropout…)
- Much more **computation power**



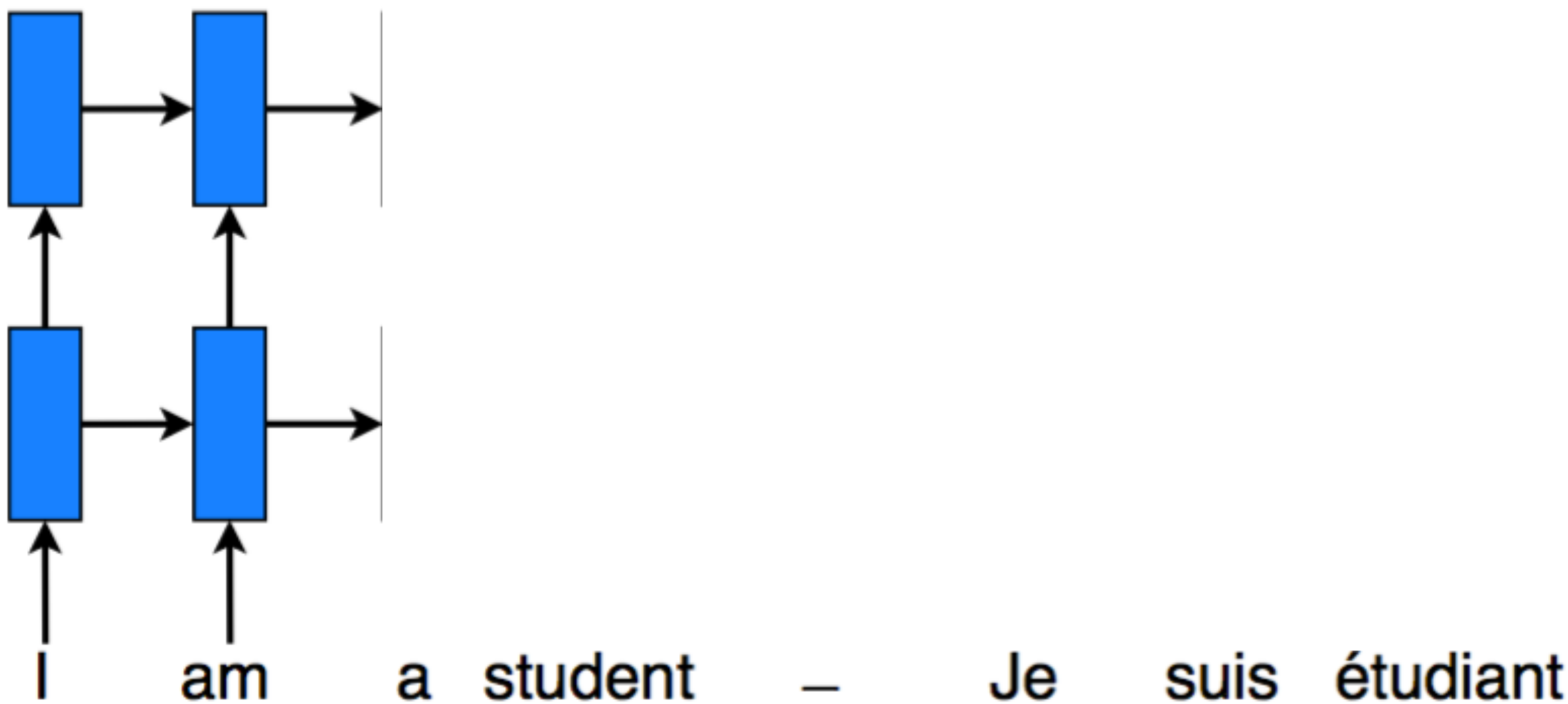
# And back to Machine Translation…

I am a student _ Je suis étudiant

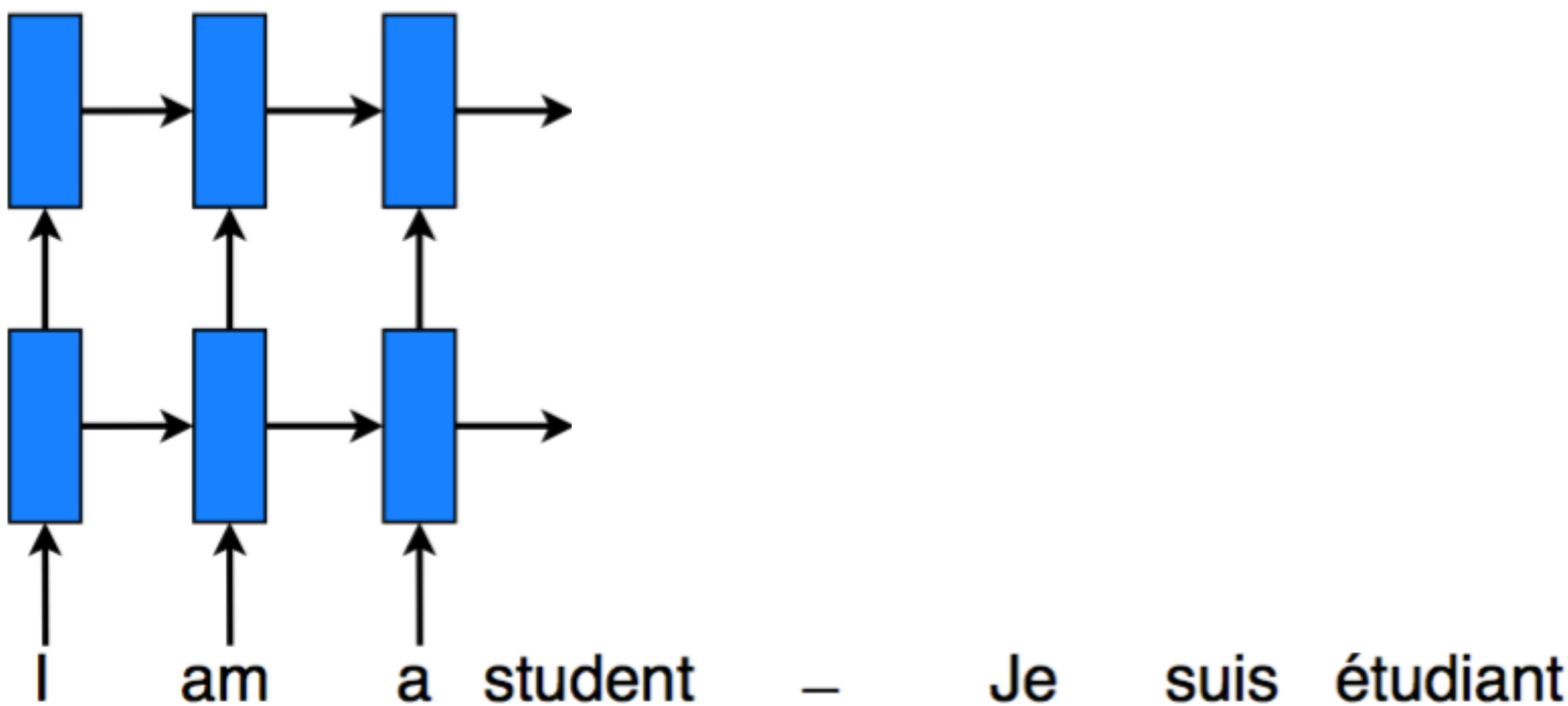
# And back to Machine Translation…



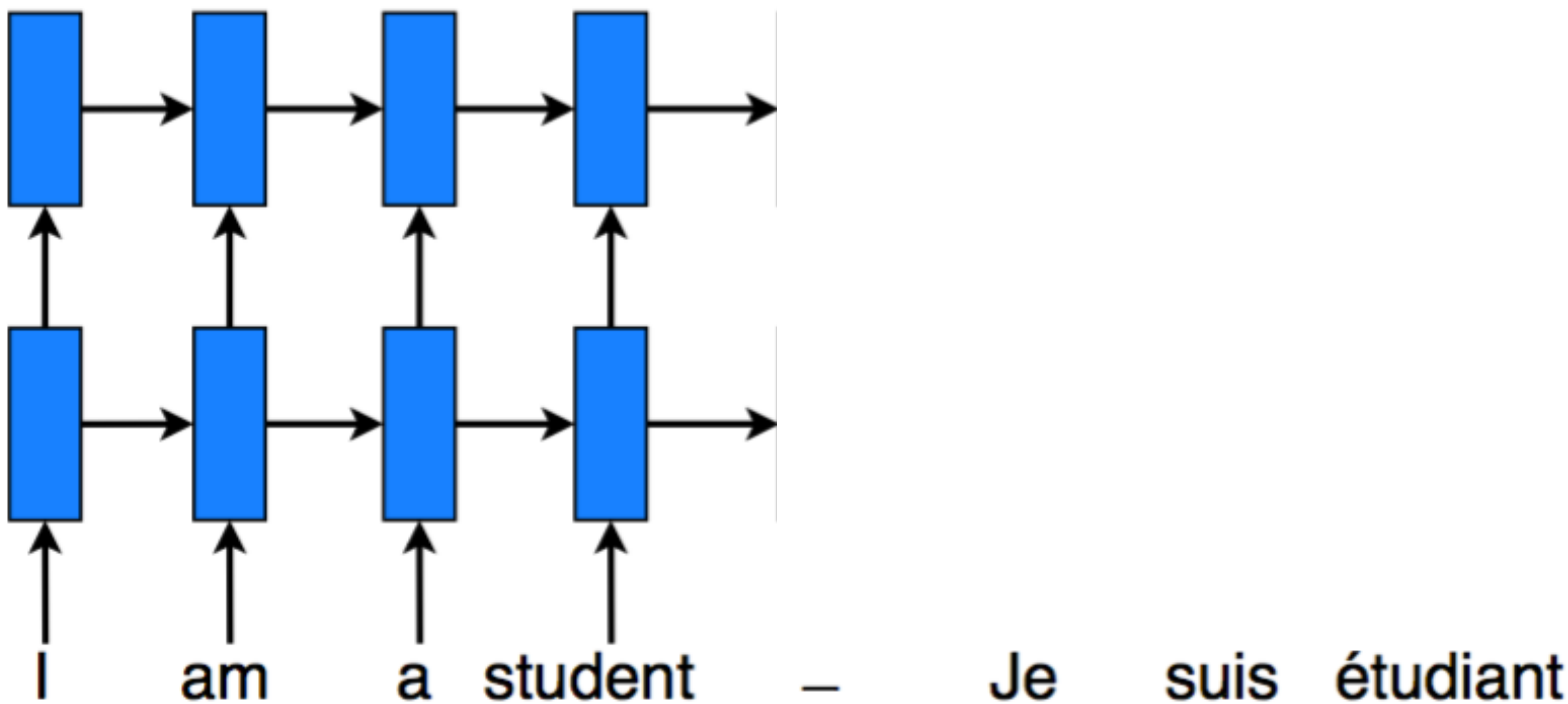
# And back to Machine Translation…



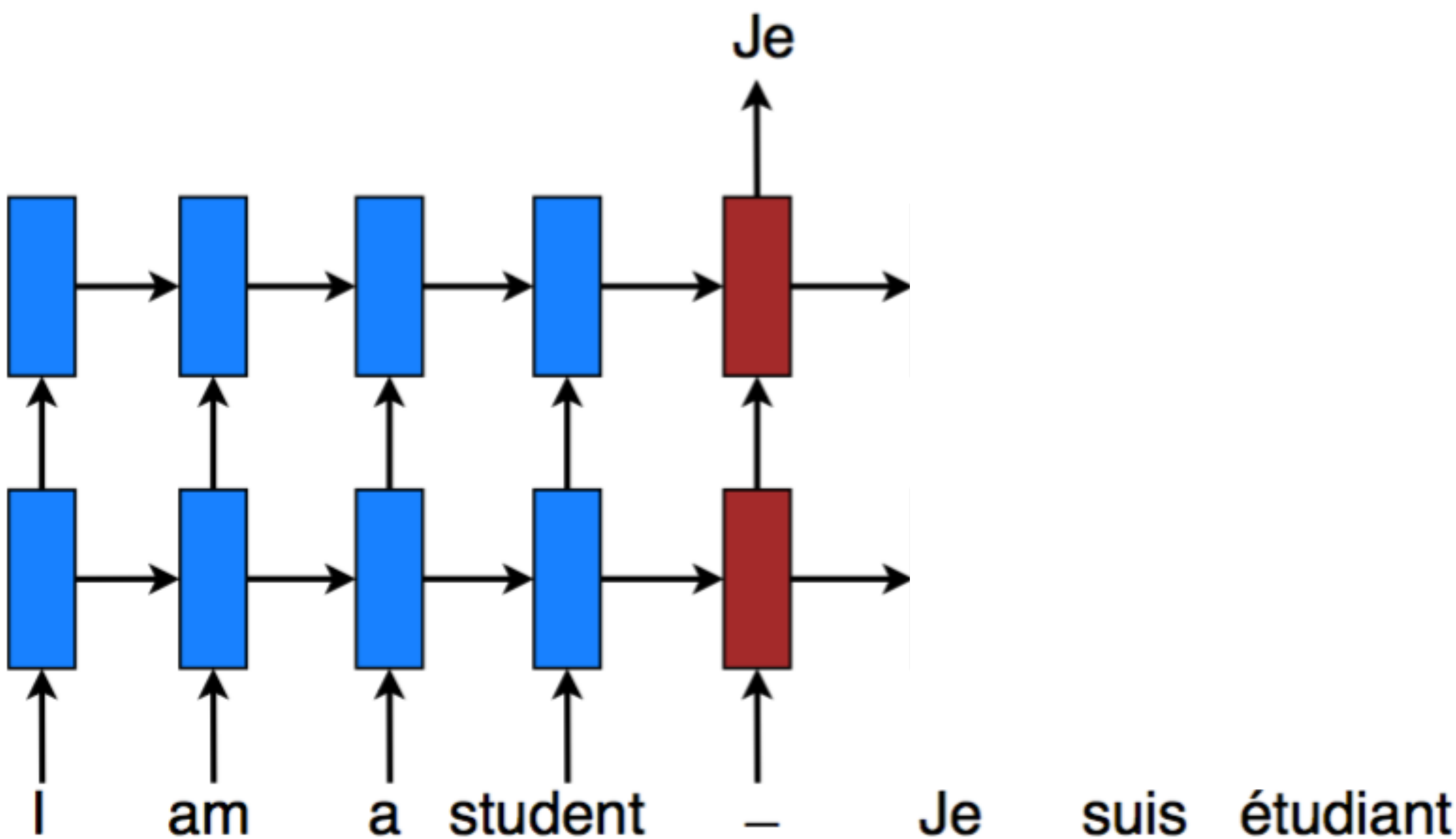
# And back to Machine Translation…



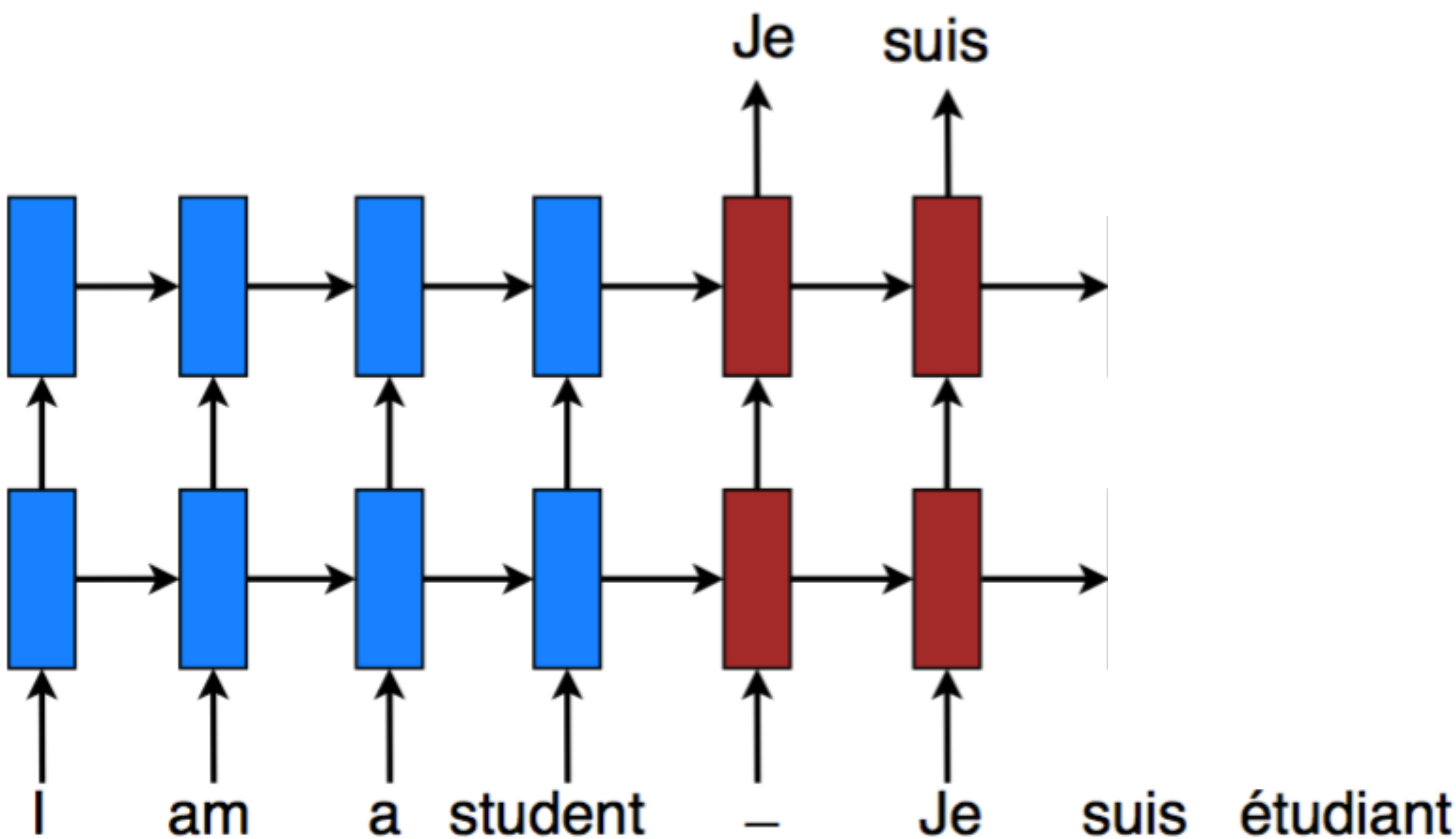
# And back to Machine Translation…



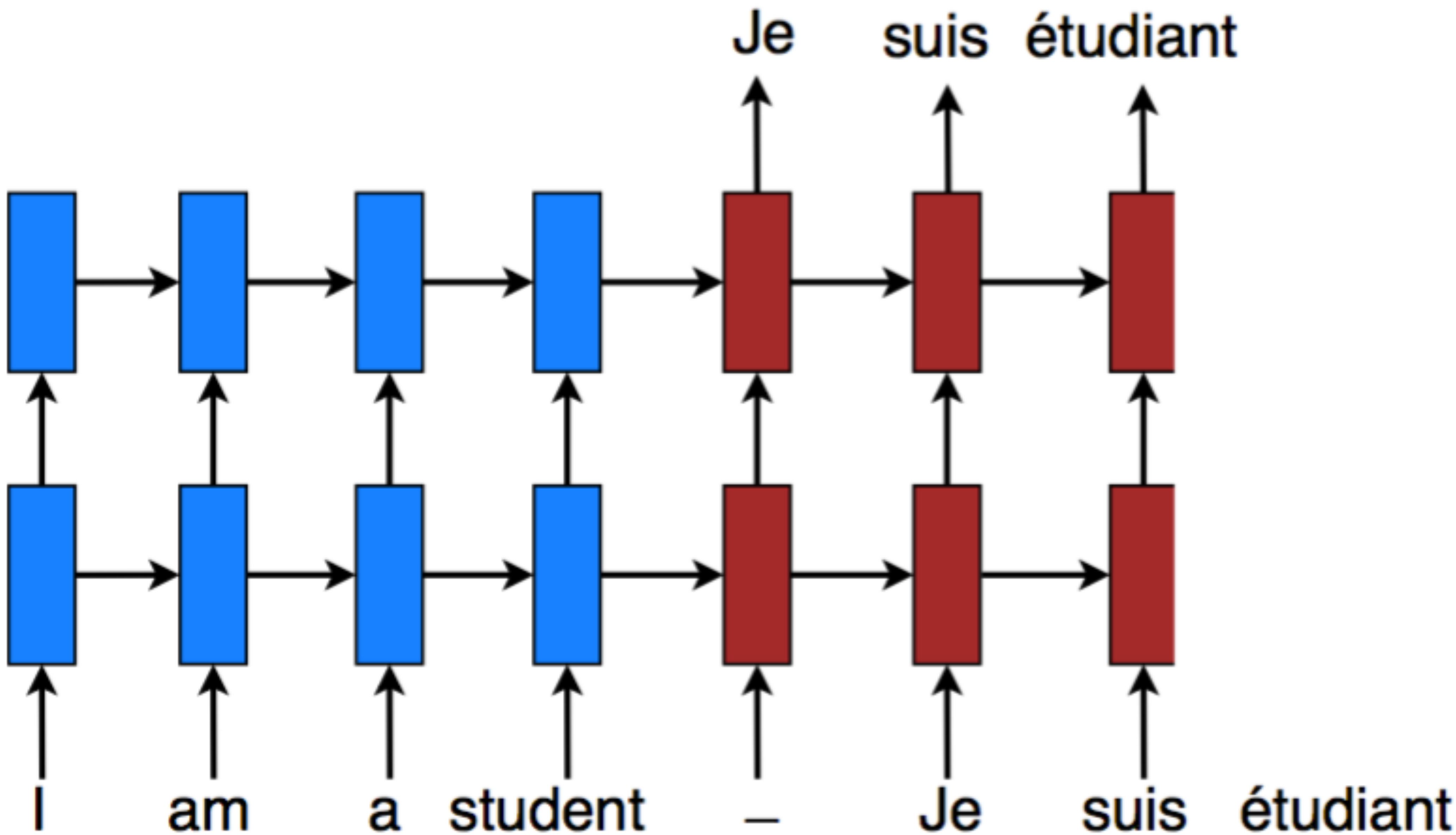
# And back to Machine Translation…



# And back to Machine Translation…

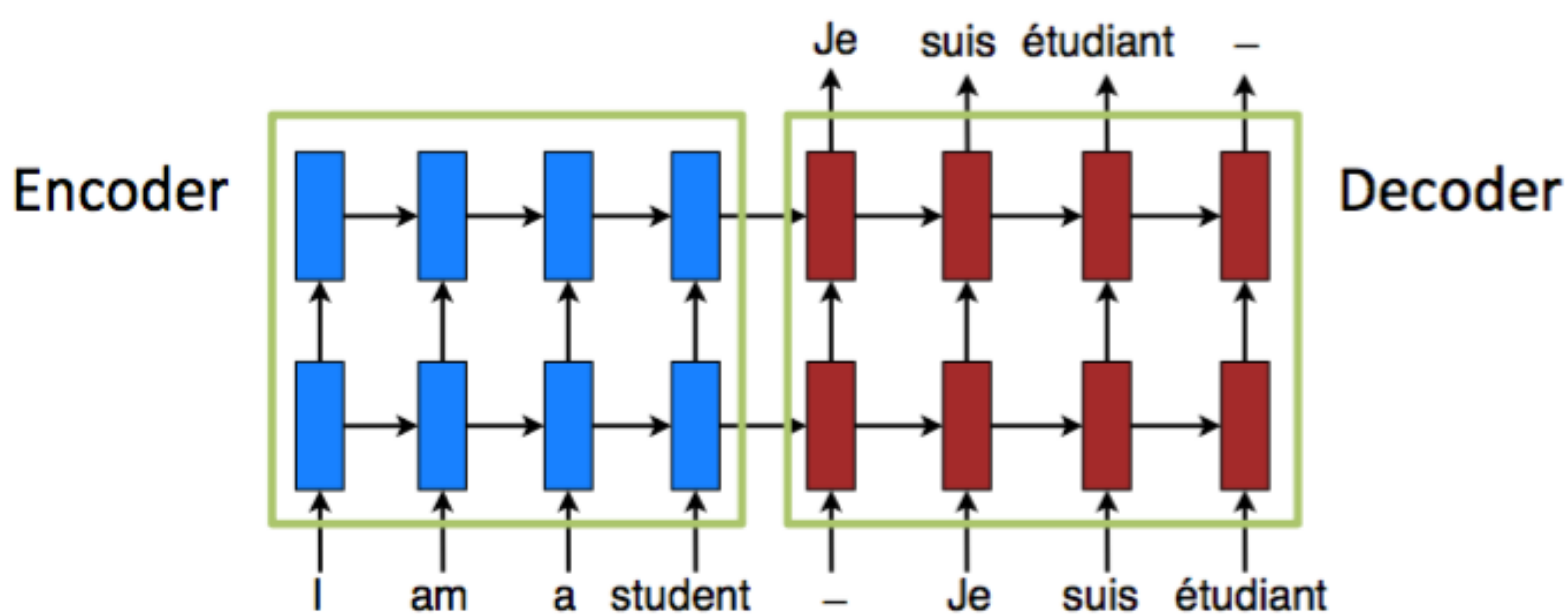


# And back to Machine Translation…



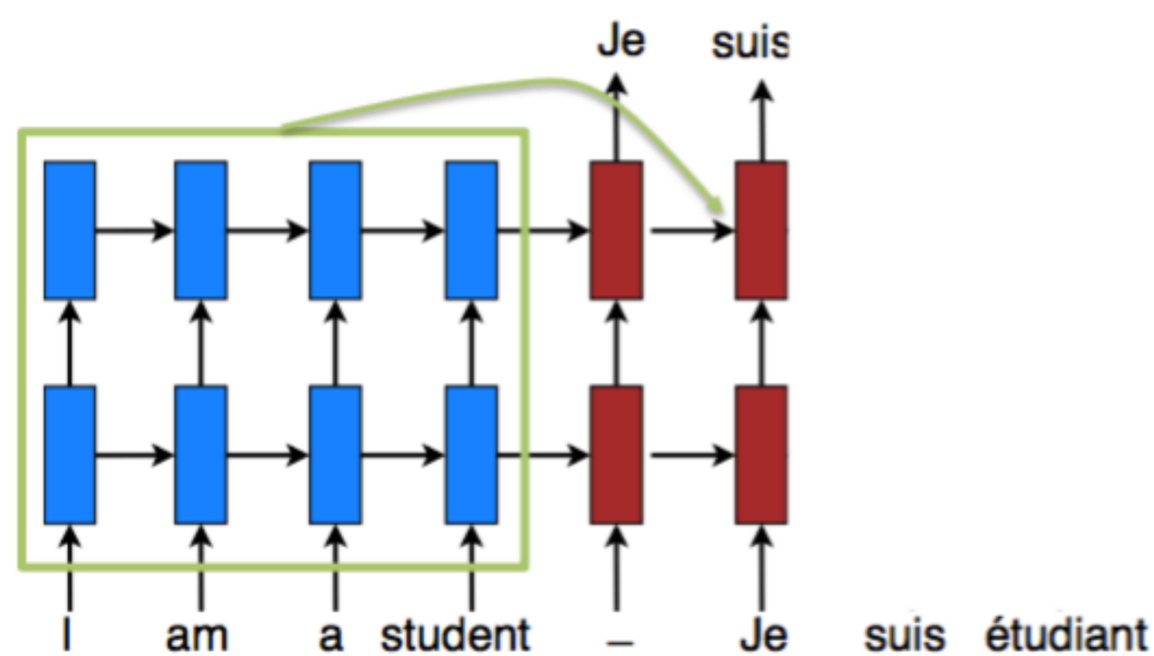
# Neural Machine Translation

- Sequence2Sequence/Encoder-Decoder model (Sutskever et al, 2014)
- Much simpler model the traditional one
- But is it too simple? Where are the alignments?



# The Attention Mechanism

- The previous model tries to decode the entire translation from one “compressed” vector
- But at each step we would like to focus on a **specific part** of the sentence



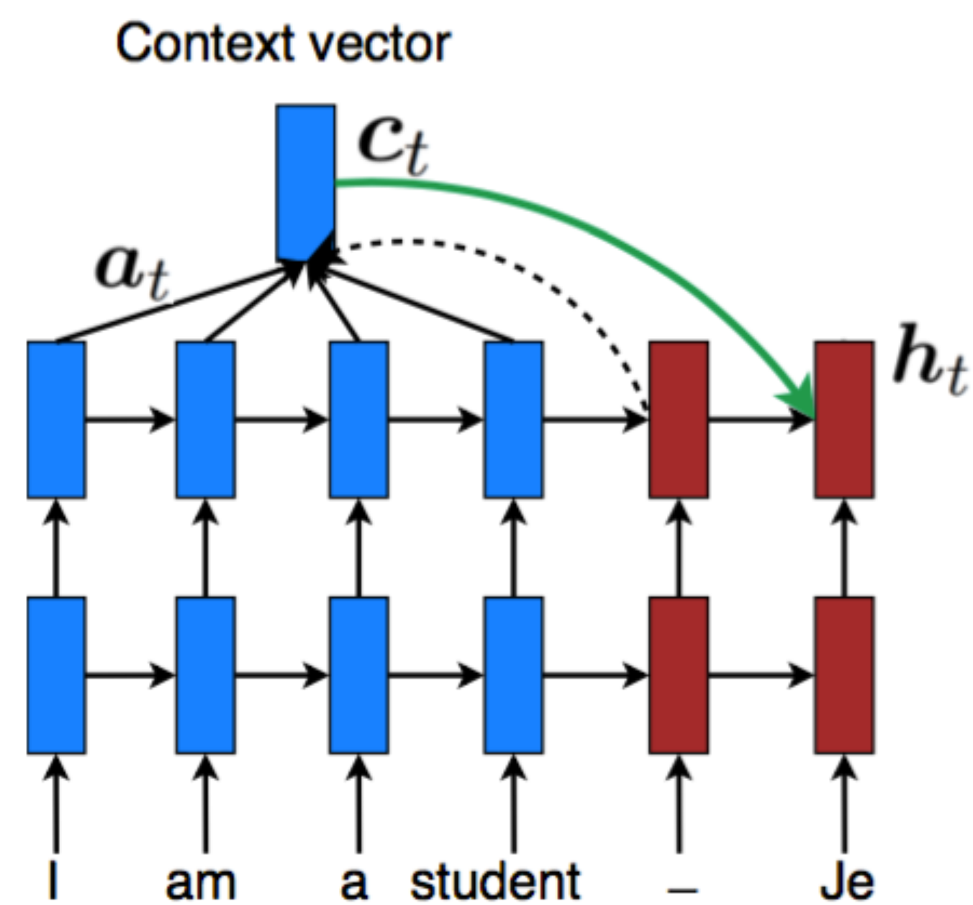
# The Attention Mechanism - This work

- The solution: enable the network to pay attention to specific areas of the input by adding new (weighted) connections

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

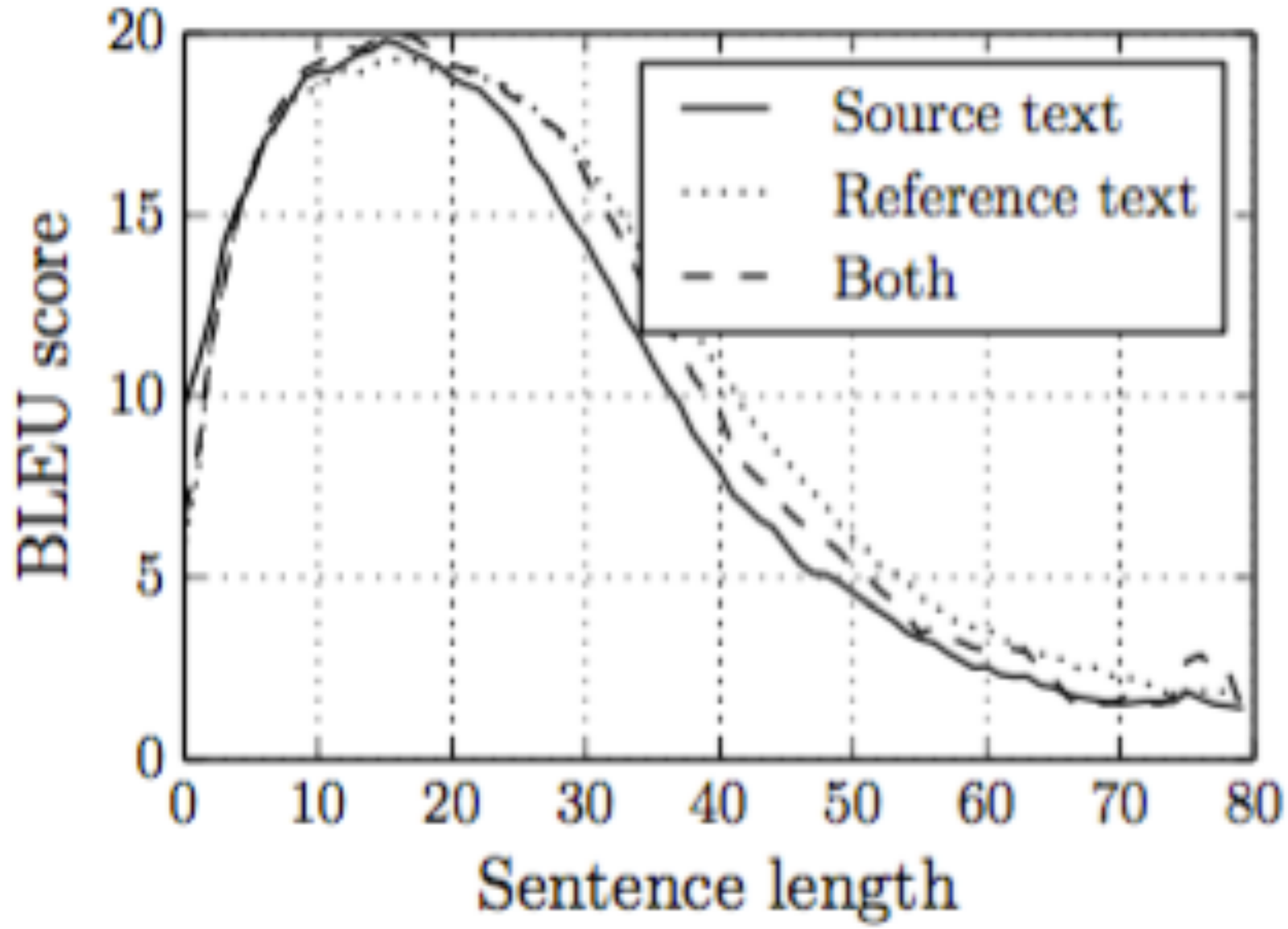
$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)},$$

$$e_{ij} = a(s_{i-1}, h_j)$$



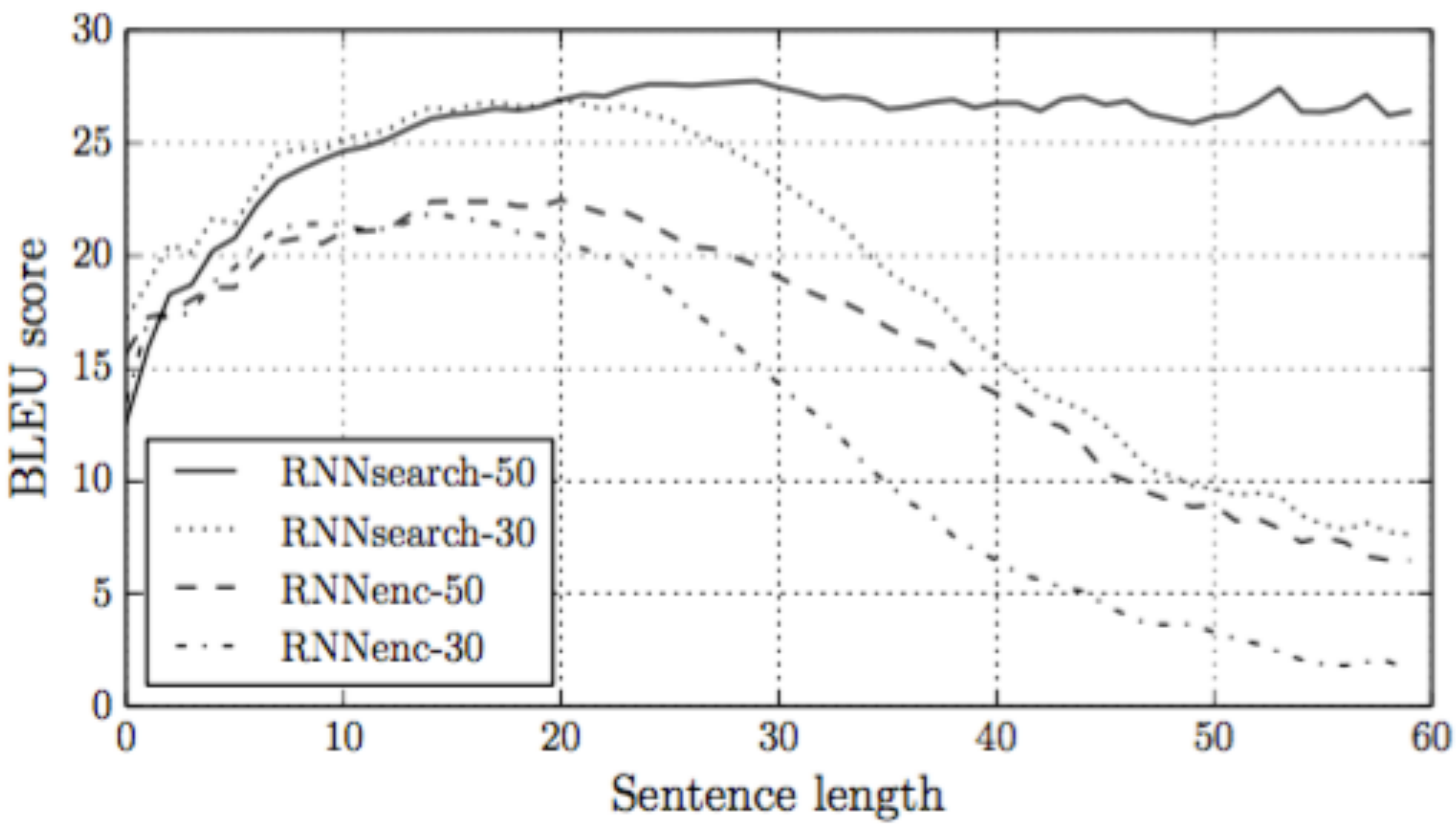
# Results - Before Attention

- Long sentences are very hard as they are compressed to a fixed length vector



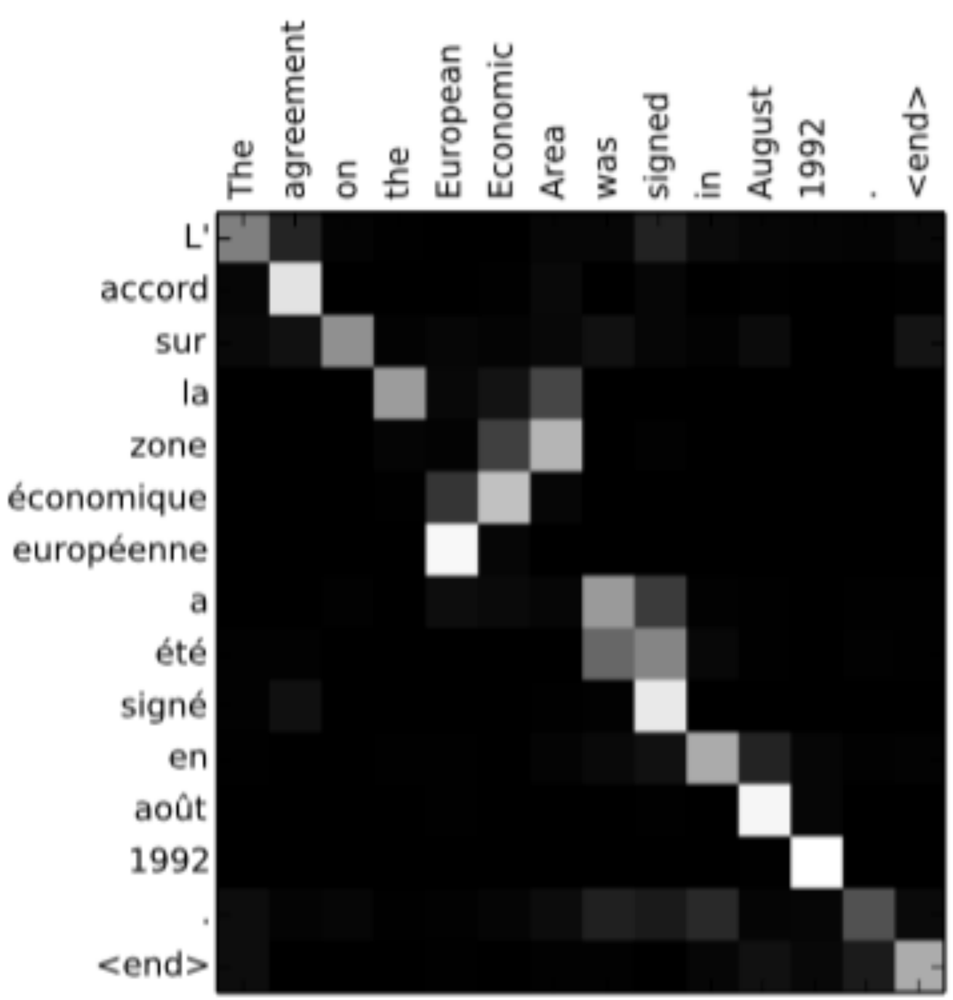
# Results - After Attention

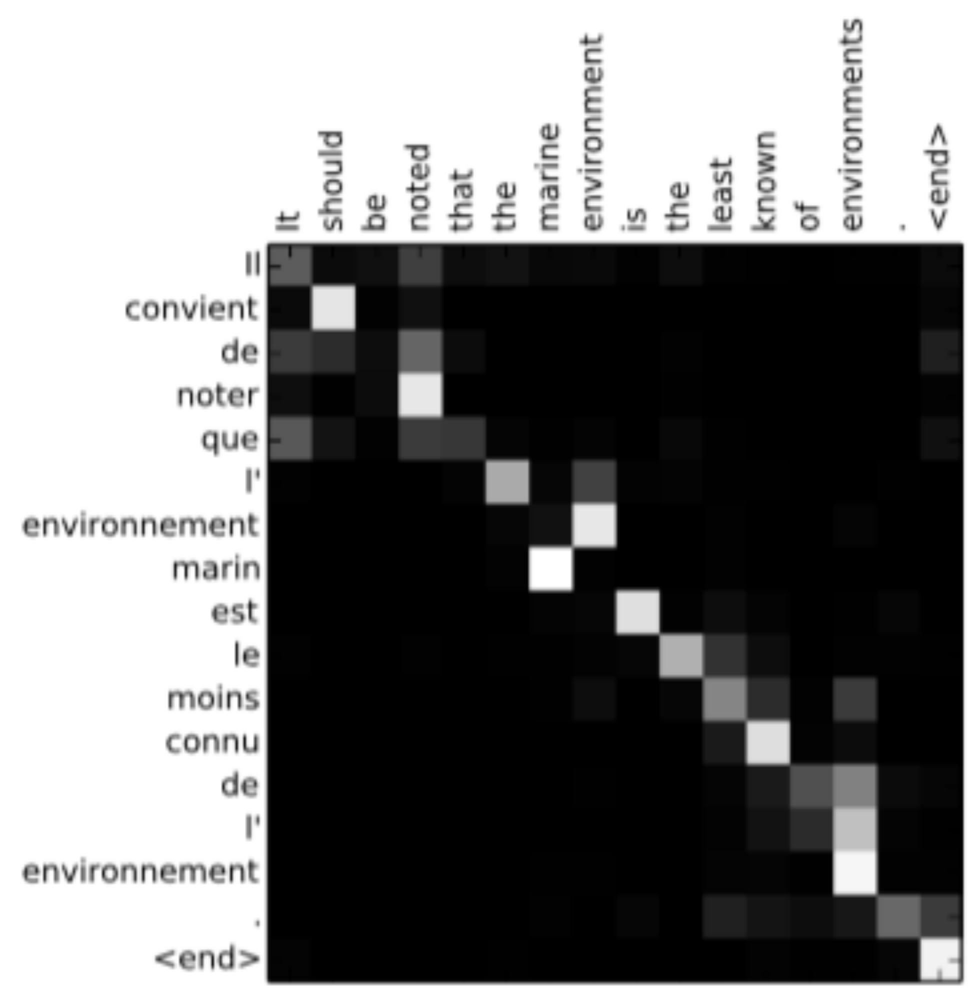
- The attention mechanism helps to overcome the issue



# Results - After Attention

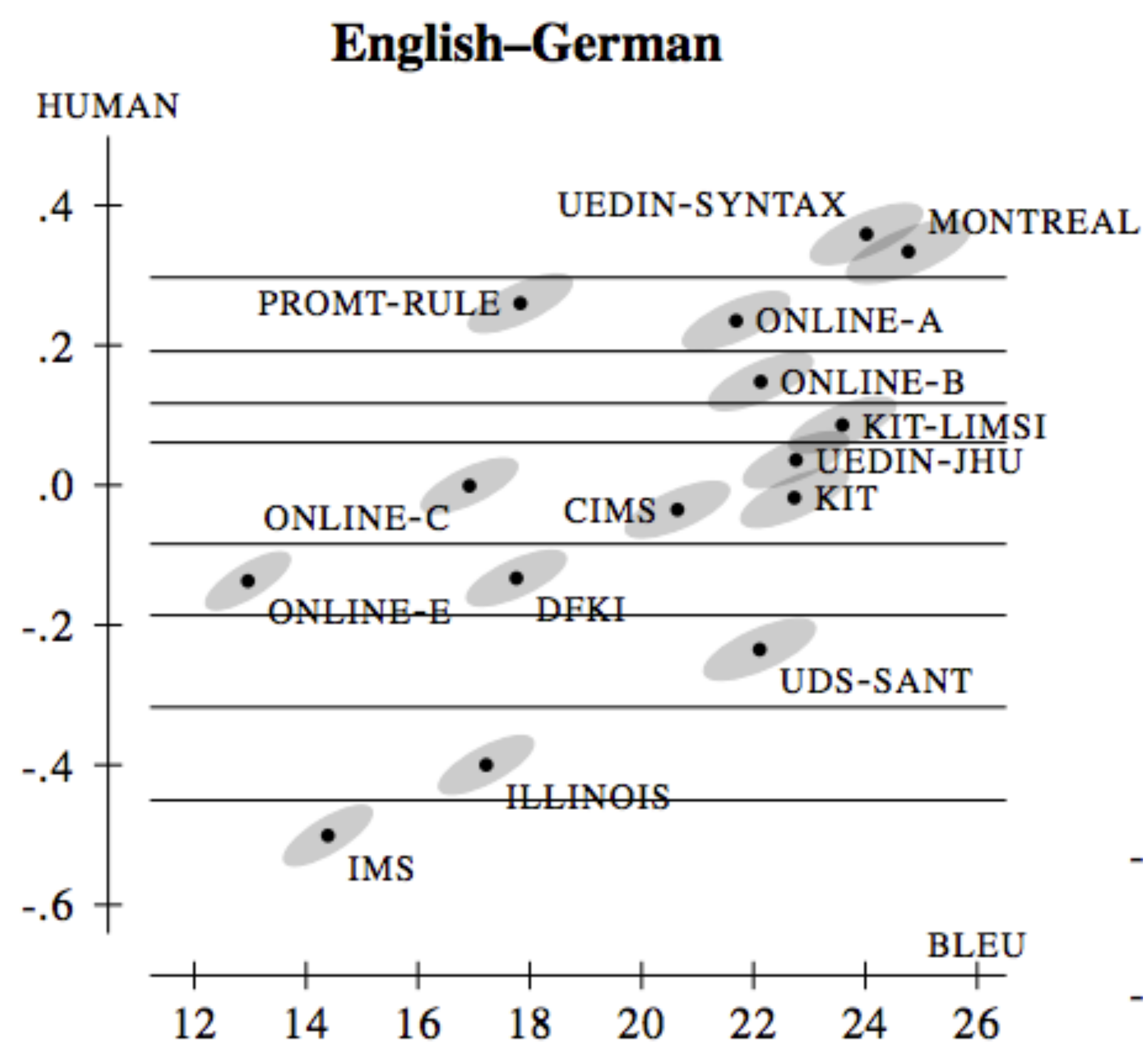
- The attention mechanism helps to overcome the issue
- The model is able to learn nice alignments:



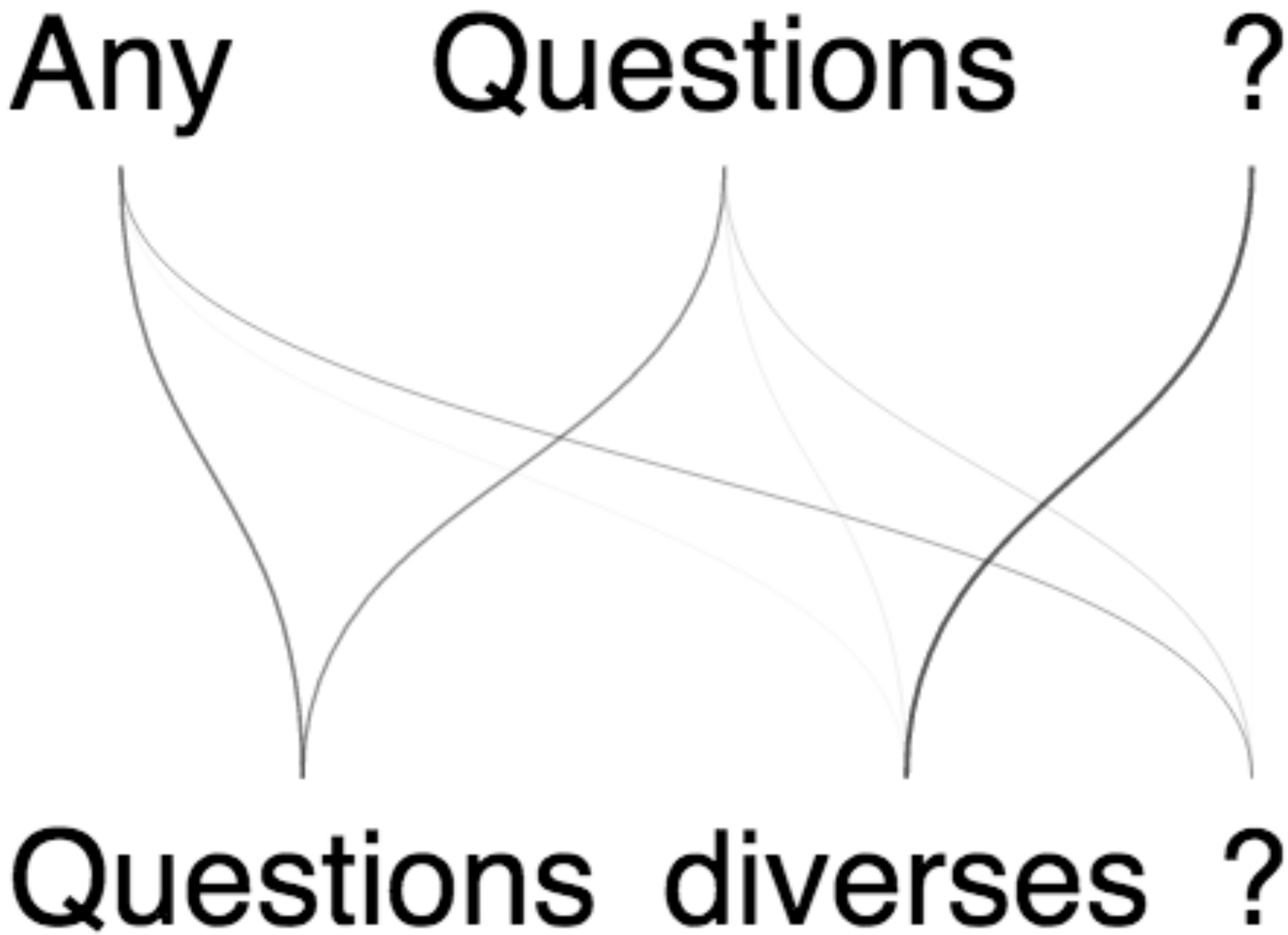
# Results - WMT 15’

- WMT 15’ - “MT olympics”
- MONTREAL - this work
- First time that a neural system gets the highest BLEU score in the competition!



# Summary

- Machine translation is hard!
- The traditional models work well (google translate) but are very complex
- Neural Machine Translation is very promising
- The attention mechanism is essential to make it work well


Any
Questions
?
Questions
diverses
?

# References


- A Primer on Neural Network Models for Natural Language Processing (Yoav Goldberg)

- Neural Machine Translation by Jointly Learning to Align and Translate (Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio)

- Stanford CS224N - Neural Machine Translation Talk (Thang Luong)

- K. Duh, Deep Learning Tutorial at DL4MT winter school