Structured prediction

Day 2

ESSLLI 2016

Overview

Today

1. What's a structured y?.
2. Gallery of structured prediction problems.
3. Structured prediction as search in state space.
4. Architecture caricature
    - Discrete state with Markov property: CRF.
    - Discrete state without Markov property: L2S.
    - Latent continous state: RNN.
5. Time permitting. The dependency parser controller problem.

Problem representation

Structured output

Yesterday we mainly looked at models where the prediction could be represented as a natural number.

Classification: $\mathcal{Y} = \{1, \ldots, k\}$, *for* k *choices.*

In structured prediction, the output is a vector of decisions.

Structured prediction: $\mathcal{Y} = \{1, \ldots, k\}^n$, *with* k *choices and* n *components.*

The output space often depends on the input.

$$\mathcal{Y}(x)$$

Structured prediction conditions

There must be some interesting interaction *between the labels* of y.

- Is assignment sentiment to a bunch of sentences (y is a vector) structured prediction?

Stronger: The loss $L(y, \tilde{y})$ should not *decompose* as a sum of component-wise losses (Daumé III 2006).

Eight structured tasks

Named entity recognition

Input: a sequence of m words.

Max Weber was born in 1864, in Erfurt. His father, Max Weber Sr., was a member of the National Liberal Party.

Potential tags:

- ORGANIZATION
- LOCATION
- PERSON

Stanford NER

(Also grounded version of task)

Output: a sequence of m tags.

Atari game playing

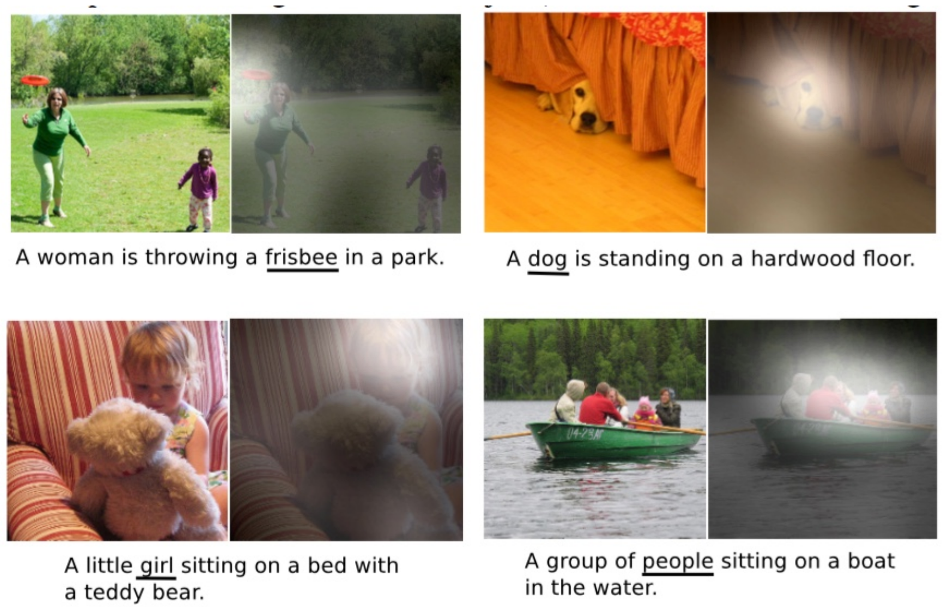
Input: a sequence of m frames.

From Mnih et al. (2015)

Output: a sequence of $l \leq m$ decisions (e.g. LEFT, RIGHT).

Image caption generation

Input: an $m \times n$ pixel image.

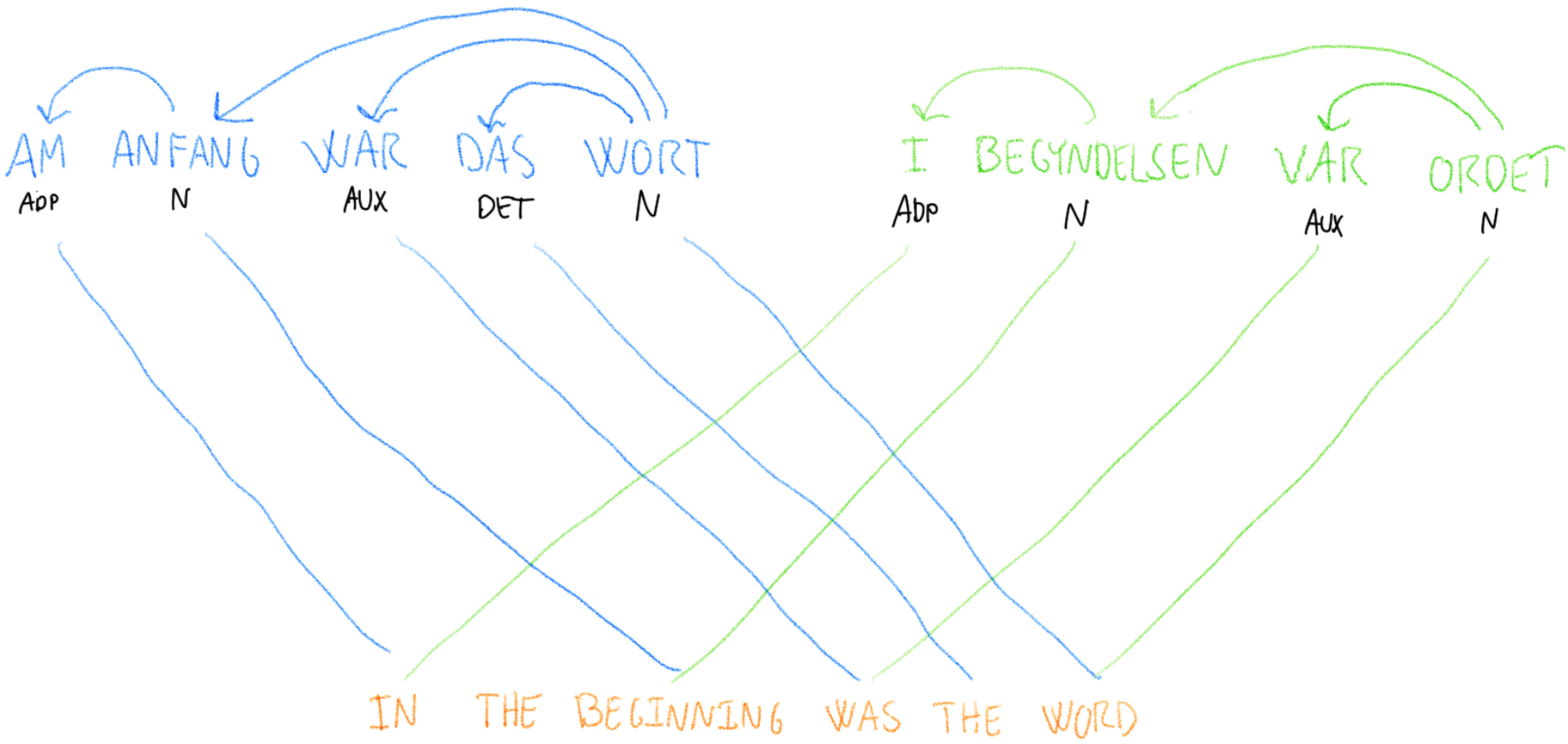


From Xu et al. (2015)

Output: a sequence of words.

Word alignment

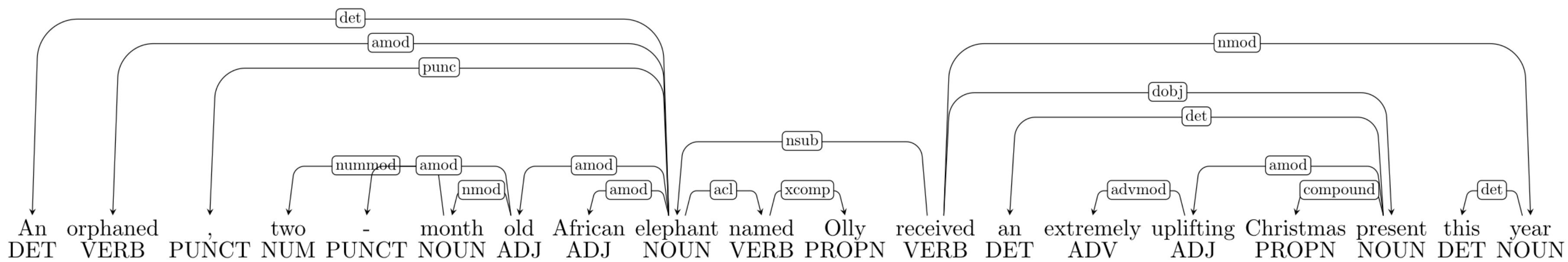
Input: two sentences of m and n tokens.



Output: an $m \times n$ adjacency matrix.

Dependency parsing

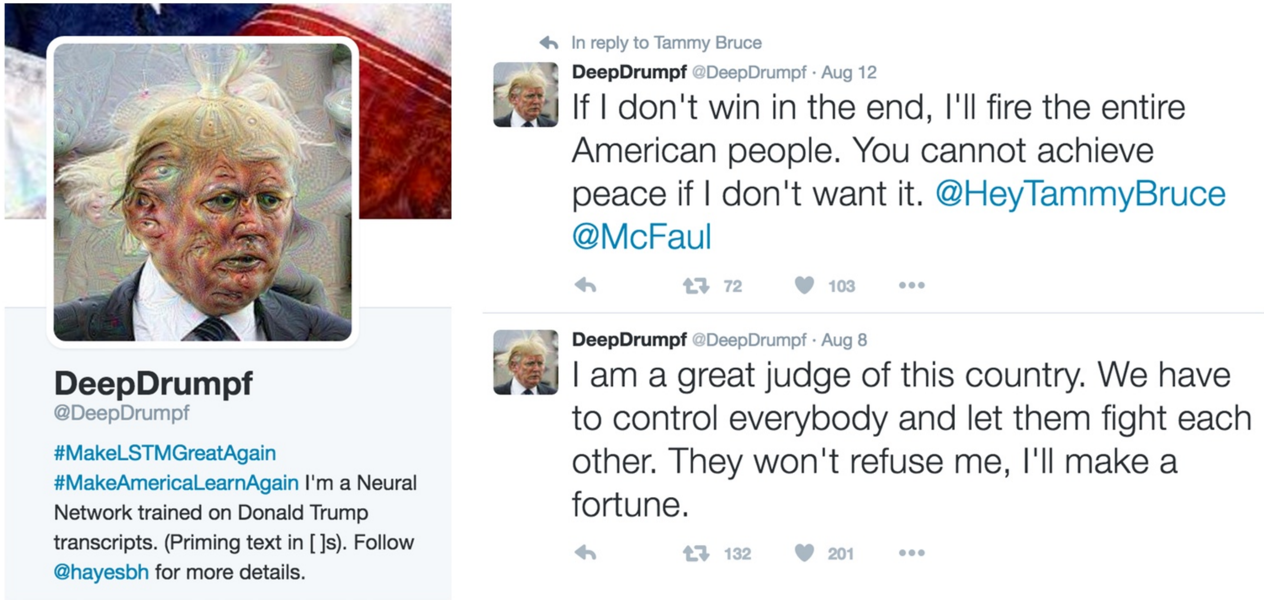
Input: a sequence of m words.



Output: a labeled tree structure with m leaf nodes.*

Donald Trump wisdom generation

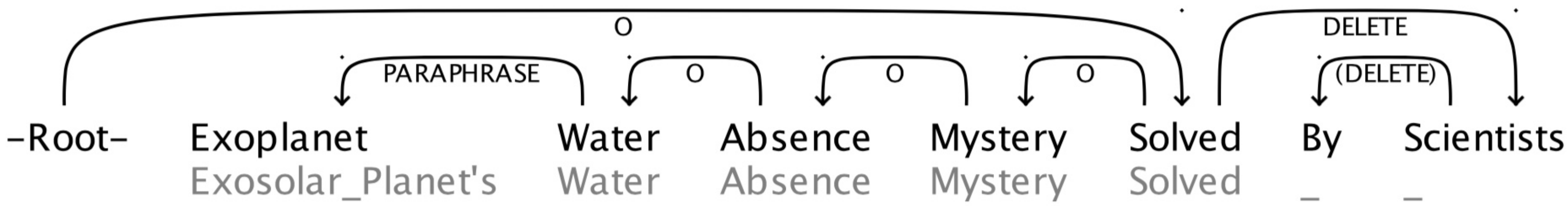
Input: An optional prime text.



Output: a sampled sequence of words.

Text simplification

Input: a parsed sentences with m edges.

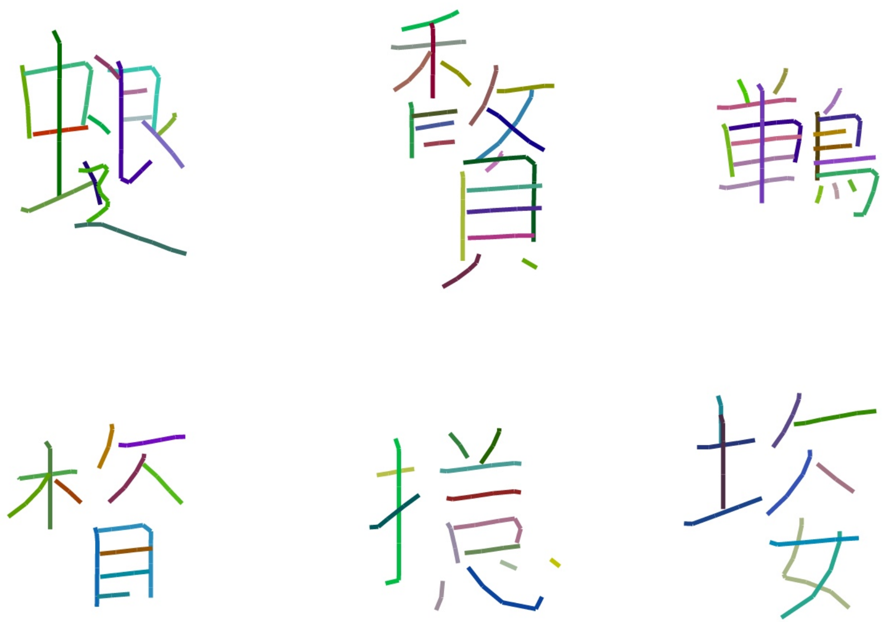


From Bingel and Søgaard (2016)

Output: m new labels, one for each edge.

Fake Chinese character generation

Input: End-of-character symbol.



Fake Kanji Generation

Output: a sampled list of strokes* that combine to a non-existing Chinese character.

Approaches

Some problems

Three problems of structured prediction:

- **Variable-sized output**. Output space not fixed; may depend on x.
- **Exponential output space**. Number of possible label sequences is exponential in the length of y.
- **Label dependence**. Label components depend on each other.

A framework

The goal of prediction is to find a function h

$$y = h(x)$$

We define h by introducing "helper" problem H, such that

$$h(x) = \arg \max_{y' \in \mathcal{Y}(x)} H(x, y', \theta)$$

Solution: score y', pick best as prediction $\tilde{y}$.

However...

After a day of exhaustive search

A compromise

We must either

- seek an **approximate solution** to the arg max,
- **restrict the label interactions** in H to make the search efficient,
- or **not search at all**.

Non-searchy options are very popular now. (We're going to ignore a big chunk of interesting work in approximate methods.)

Order of decisions

We'll assume a sensible decision order. This will follow the y vector.

Does order matter? (Vinyals, Bengio, and Kudlur 2015)

- MT results improve dramatically when input sentence is reversed (Sutskever, Vinyals, and Le 2014).
- Parsing performance improve from same transformation (Vinyals et al. 2014).

Controller problem

If original problem is hard to solve directly, maybe there's an easier problem we can attempt?

This works if the **controller problem** produces a vector y', such that

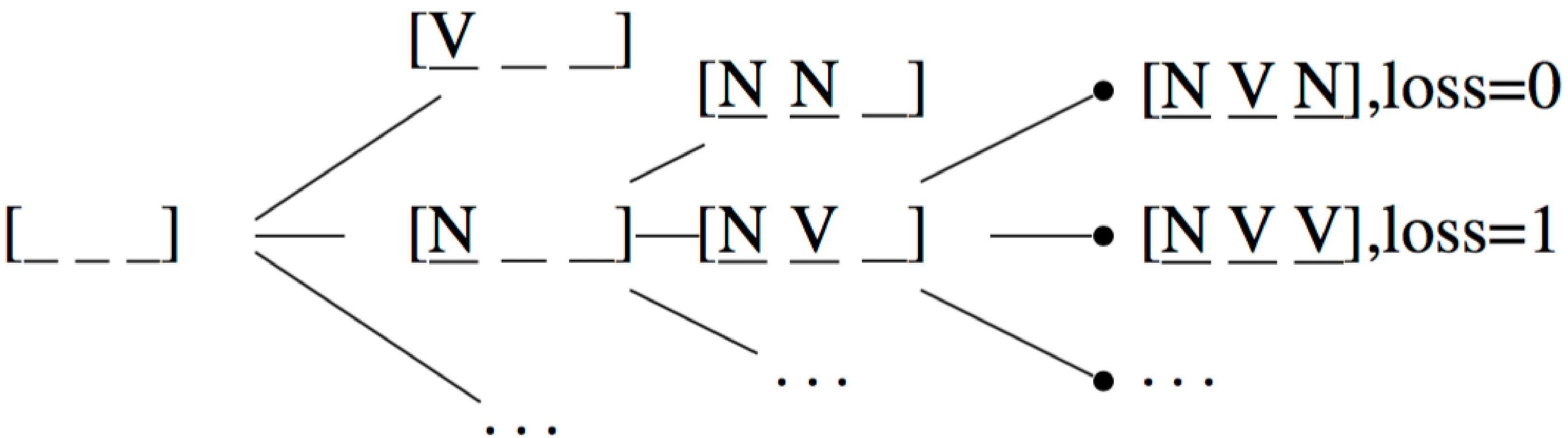
$$y = T(y'),$$

where T is a deterministic transformation.

Parsing typically uses a controller problem.

The state space

fish	drain	flood
NOUN	NOUN	NOUN
VERB	VERB	VERB
ADJ	ADJ	ADJ



State space exploration. Figure Chang et al. (2015)

What if: we used a classifier?

An unstructured solution.

Pros and cons? How does it fare wrt. the problems from before?

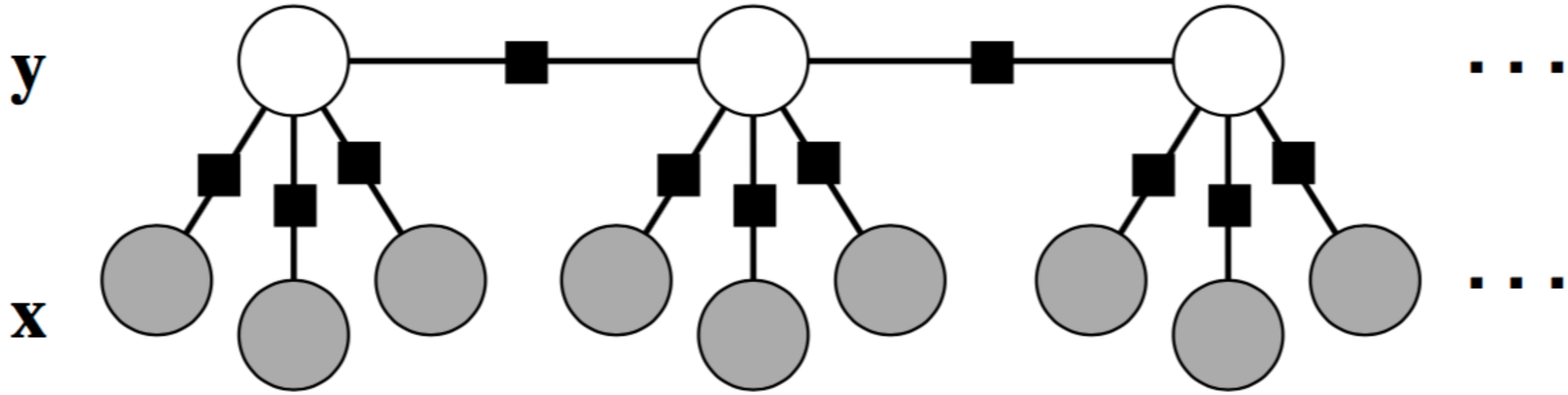
- Variable-sized output.
- Exponential output space.
- Label dependence.

What if: the classifier could depend on just the previous label

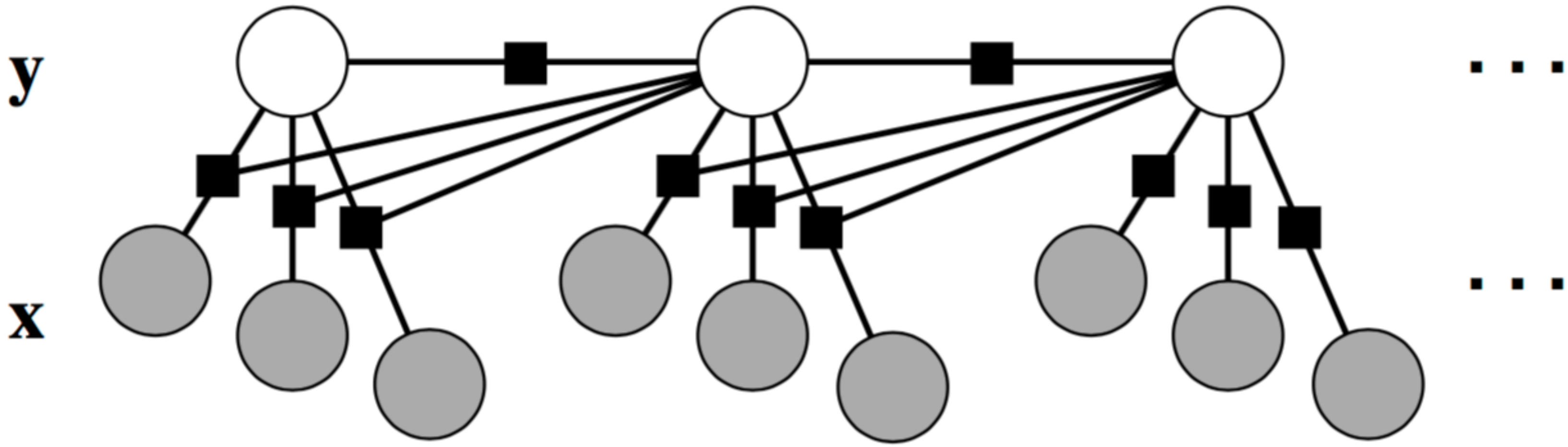
CRFs

The **conditional random field (CRF)** is a principled way of implementing a classifier with limited memory.

Any labeling decision can only depend on the previous label. (**Markov property**).



CRF factor graph.



CRF factor graph, more deps.

CRF probability function

$$P(y|x) = \frac{1}{Z(x,\theta)} \prod_{i=1}^{n} \exp\left\{\theta^\top \Phi(x, i, y_i, y_{i-1})\right\}$$

where,

$$Z(x,\theta) = \sum_{y \in \mathcal{Y}(x)} \prod_{i=1}^{n} \exp\left\{\theta^\top \Phi(x, i, y_i, y_{i-1})\right\}$$

Decoding is not approximate: it faithfully recovers the best y.

What if: the classifier could depend on the whole history

Learning to search

In **learning to search** we can condition on complete history because inference no longer involves search. Instead we train a classifier to navigate the state space in a loss-minimising way.

Like in the CRF, we'll have a feature function over the state. The main difference is that we have access all past decision, in addition to the whole input:

$$\Phi(x, i, y, y_{1:i-1})$$

$$\Phi(x, i, y, y_{1:i-1})$$

How do we generate training data? Note that the history is sparse.

Training data for L2S

Learning to search (L2S) is a form of **imitation learning** and requires that we have a **reference policy** π_{ref}.

A reference policy can be **optimal** (π^*) if it tells us what the best thing (leading to lowest loss) is to do at any given state. The reference policy is usually derived using labeled data.

We wish to learn a policy π that imitates the reference policy π_{ref}.

First idea for training data

1. Set $i = 0$ and $s = ()$ to an empty list.
2. Use $a = \arg\max_a \pi^*(s, a))$ to get the optimal action from state s.
3. Generate a multi-class example $(\Phi(x, i, a, s), a)$.
4. Move to next state by appending the action to the current state $s = s \oplus (a)$. Increment i.
5. Repeat steps 2-5 until the end of the sequence.

Would this work?

Problem 1: No error exploration

Error exploration

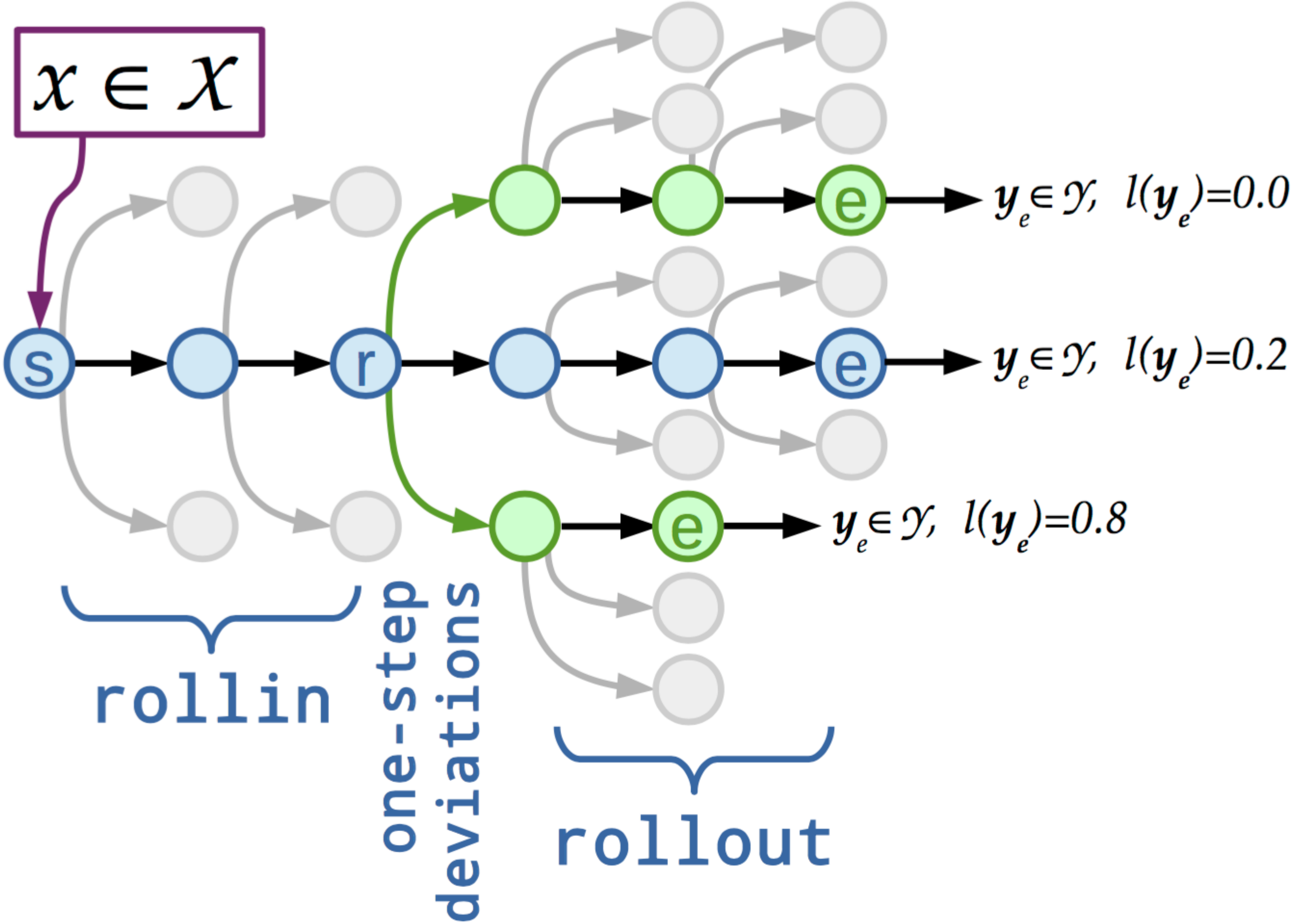
The policy only learns what to from states that are in the optimal trajectory.

Problem 2: Refining the loss

The policy only knows about **good** actions (one per state) and **bad** actions (the rest). In reality we may have **better** or **worse** actions, each of which has an associated cost.

The final cost of an action only becomes known when we reach the end state.

Roll-in, roll-out



What works, when

roll-out → ↓ roll-in	**Reference**	**Mixture**	**Learned**
Reference	Inconsistent		
Learned	Not locally opt.	Good	RL

What if: the classifier also modelled the state?

Recurrent neural networks.

In L2S the state is accessed only through the feature function Φ.

The onus is on the implementor to decide how to present the history of decisions to the classifier, including how to compress a variable-sized history into a fixed-length feature vector.

Recurrent neural networks model the state as a continous latent vector.

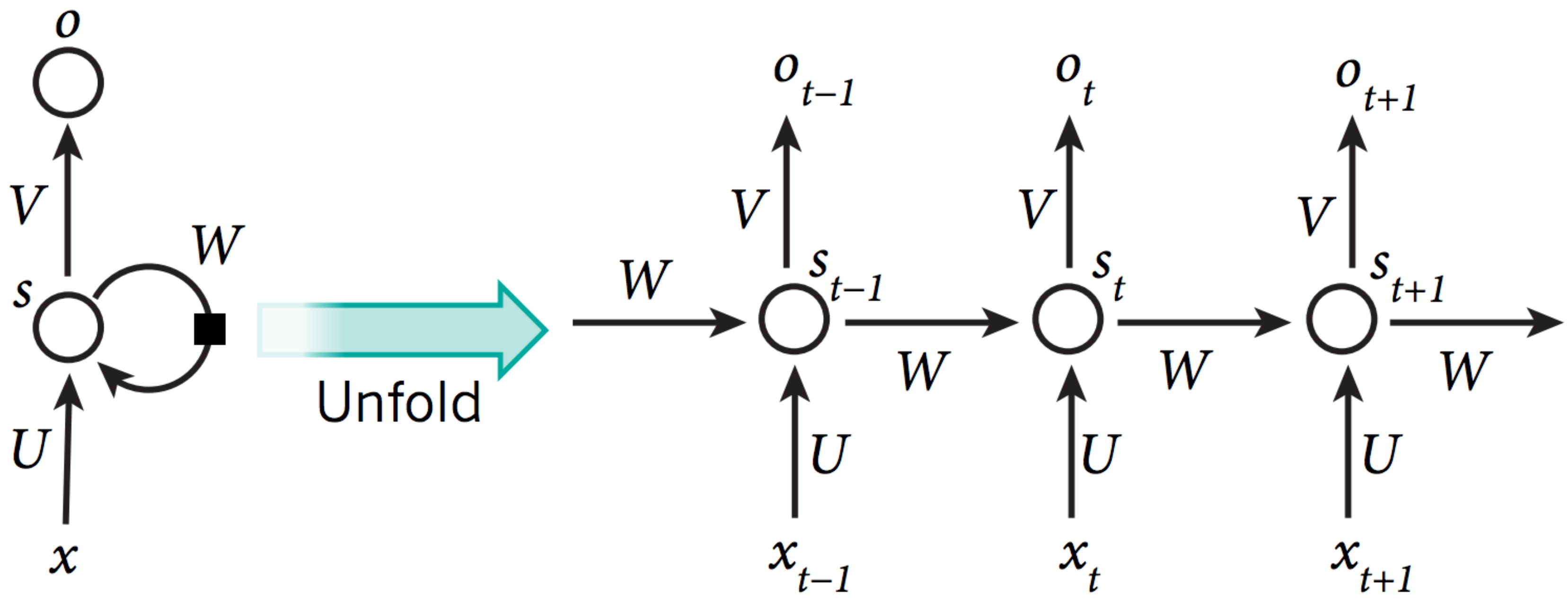


Figure from LeCun, Bengio, and Hinton (2015–5AD)

Probability function of an seq2seq model

Example from Vinyals et al. (2014):

$$P(y|x) = \prod_{i=1}^{n} P(y_i|x_1, \ldots, x_n, y_1, \ldots, y_{i-1})$$

Rewritten as function

$$P(y|x) = \prod_{i=1}^{n} \mathrm{RNN}(\Phi(x, i, y_{1:i-1}), \mathbf{h}_{i-1})_{y_i}$$

Flexible input-output

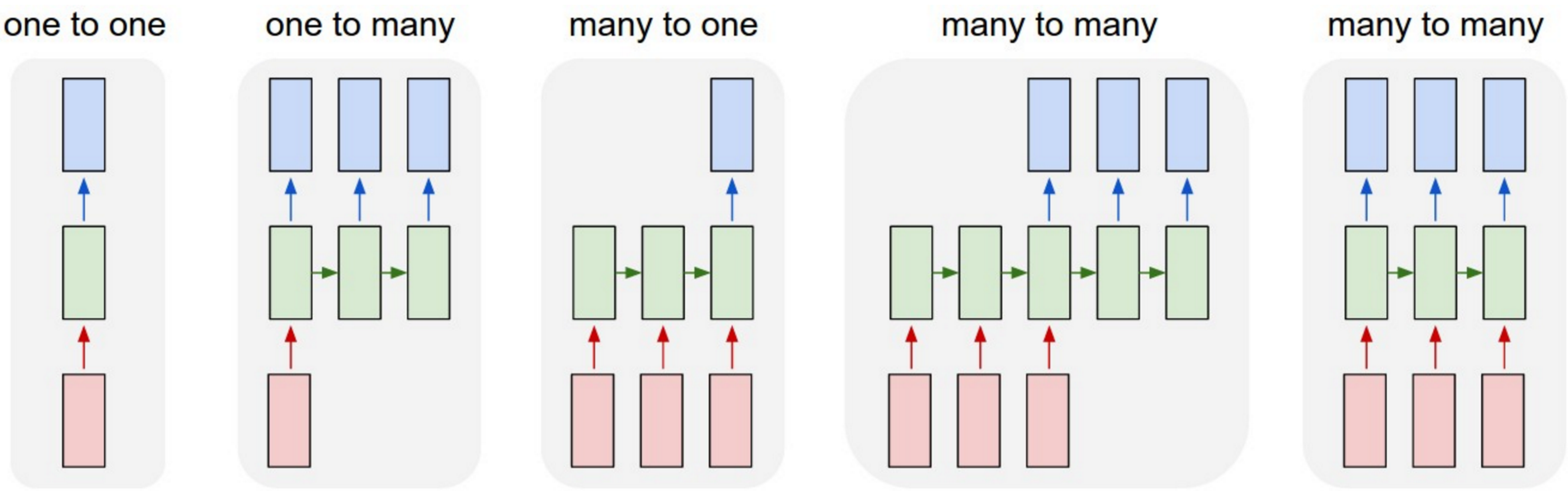
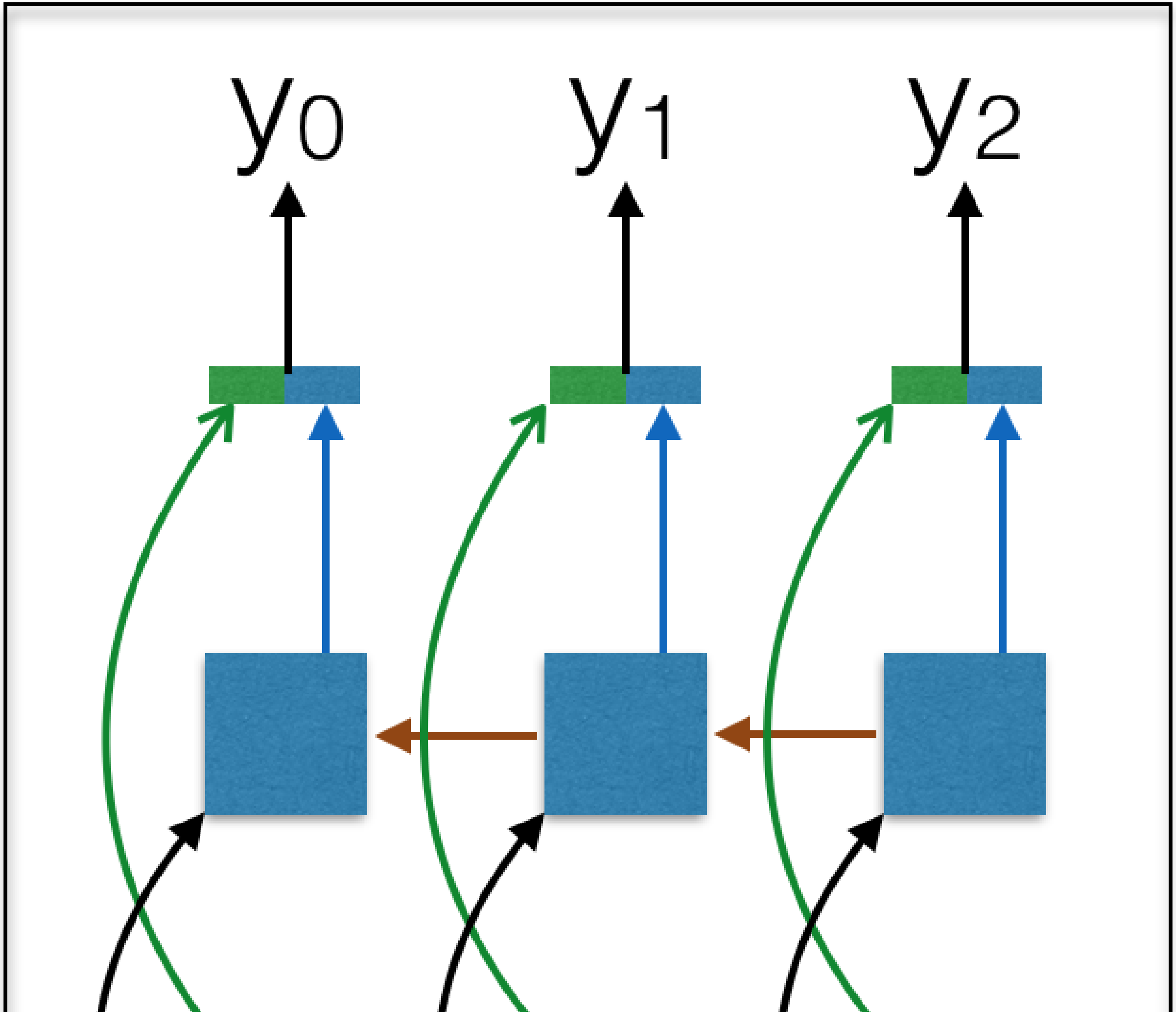


Figure from *The Unreasonable Effectiveness of Recurrent Neural Networks*

BiRNN



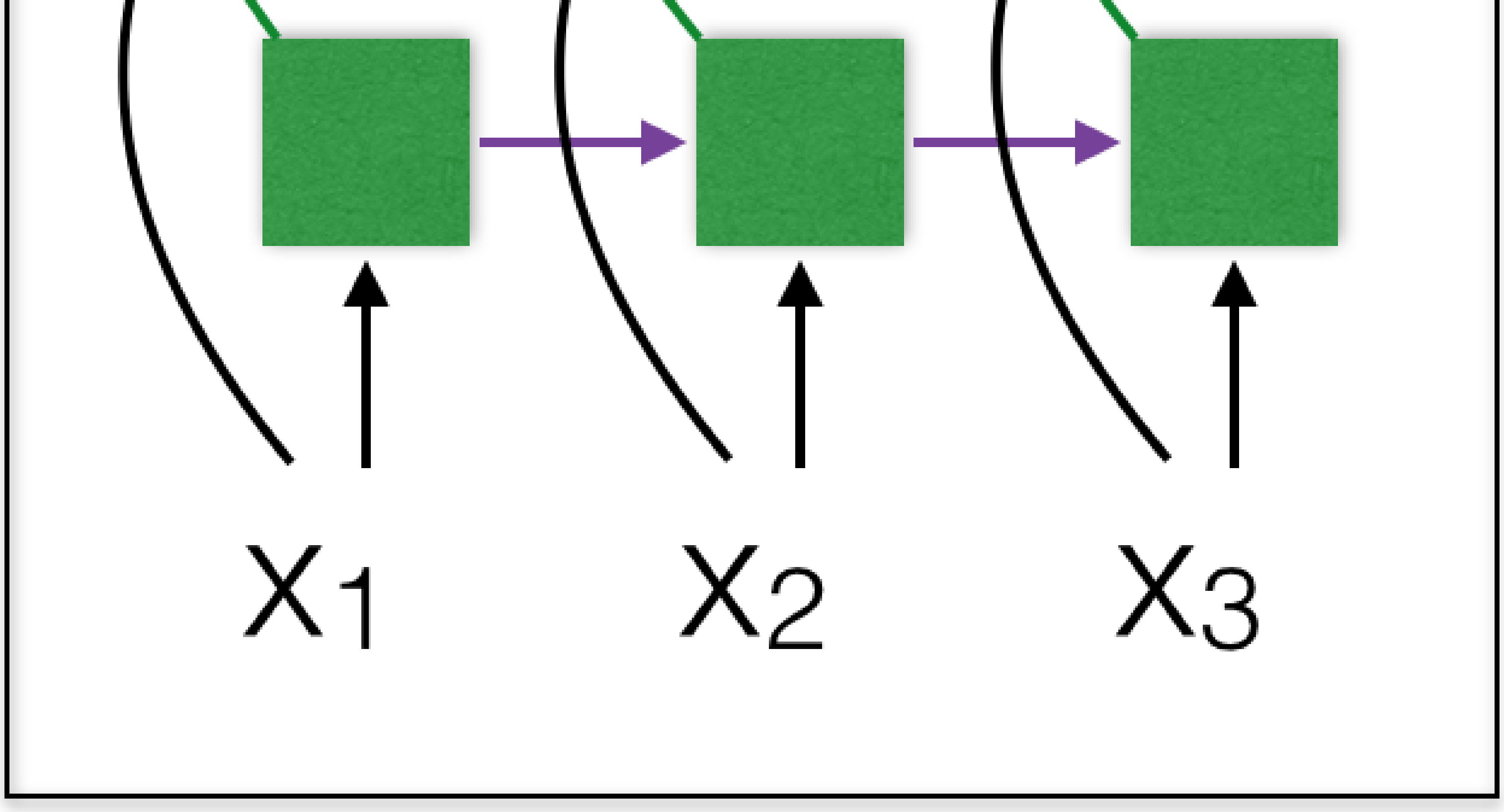


Image credit

Encoder-decoder

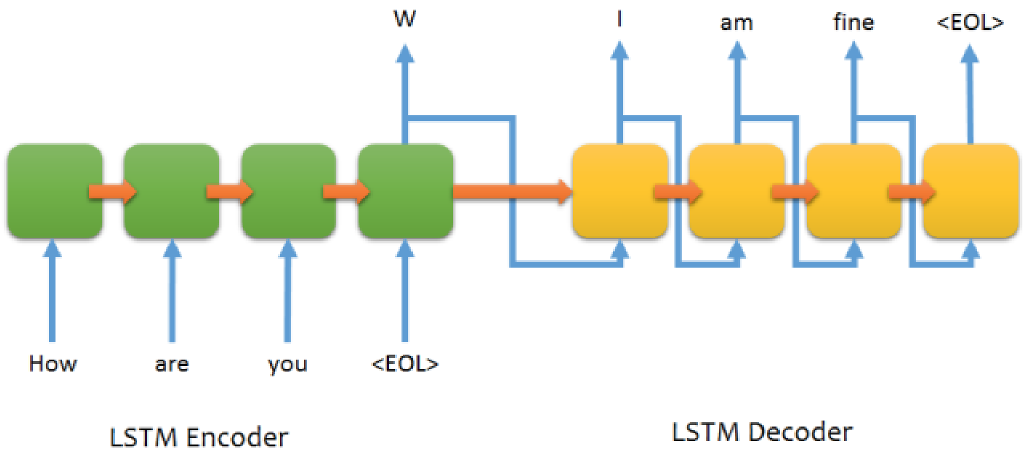


Image credit

(Vinyals et al. 2014)

What deep learning can learn from CRFs?

Label bias problem in beam search (Andor et al. 2016).

References

References

Andor, Daniel, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. "Globally Normalized Transition-Based Neural Networks." *Association for Computational Linguistics*. https://arxiv.org/abs/1603.06042.

Bingel, Joachim, and Anders Søgaard. 2016. "Text Simplification as Tree Labeling." In *The 54th Annual Meeting of the Association for Computational Linguistics*, 337.

Chang, K.-W., A. Krishnamurthy, A. Agarwal, H. Daumé III, and J. Langford. 2015. "Learning to Search Better Than Your Teacher." *ArXiv E-Prints*, February.

Daumé III, Hal. 2006. "Practical Structured Learning Techniques for Natural Language Processing." Ph.d.-afhandling, Los Angeles, CA: University of Southern California. http://pub.hal3.name/#daume06thesis.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015–5AD. "Deep Learning." *Nature* 521 (7553). Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.: 436–44. http://dx.doi.org/10.1038/nature14539.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, et al. 2015. "Human-Level Control Through Deep Reinforcement Learning." *Nature* 518 (7540). Nature Publishing Group: 529–33.

Sutskever, I., O. Vinyals, and Q. V. Le. 2014. "Sequence to Sequence Learning with Neural Networks." *ArXiv E-Prints*, September.

Vinyals, O., S. Bengio, and M. Kudlur. 2015. "Order Matters: Sequence to Sequence for Sets." *ArXiv E-Prints*, November.

Vinyals, O., L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. 2014. "Grammar as a Foreign Language." *ArXiv E-Prints*, December.

Xu, K., J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. 2015. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention." *ArXiv E-Prints*, February.