L101: Incremental structured prediction

Structured prediction reminder

Given an input **x** (e.g. a sentence) predict **y** (e.g. a PoS tag sequence, cf lecture 6):

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} score(x, y)$$

Where $\boldsymbol{Y}$ is rather large and often depends on the input (e.g. $L^{|\boldsymbol{x}|}$ in PoS tagging)

Various approaches:

- Linear models (structured perceptron)
- Probabilistic linear models (conditional random fields)
- Non-linear models

Decoding

Assuming we have a trained model, decode/predict/solve the argmax/inference:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} score(x, y; \theta)$$

Isn't finding θ meant to be the slow part (training)?

Decoding is often necessary for training; you need to predict to calculate losses

Do you know a model where training is faster than decoding?

Hidden Markov Models (especially if you don't do Viterbi)

Dynamic programming to the rescue?

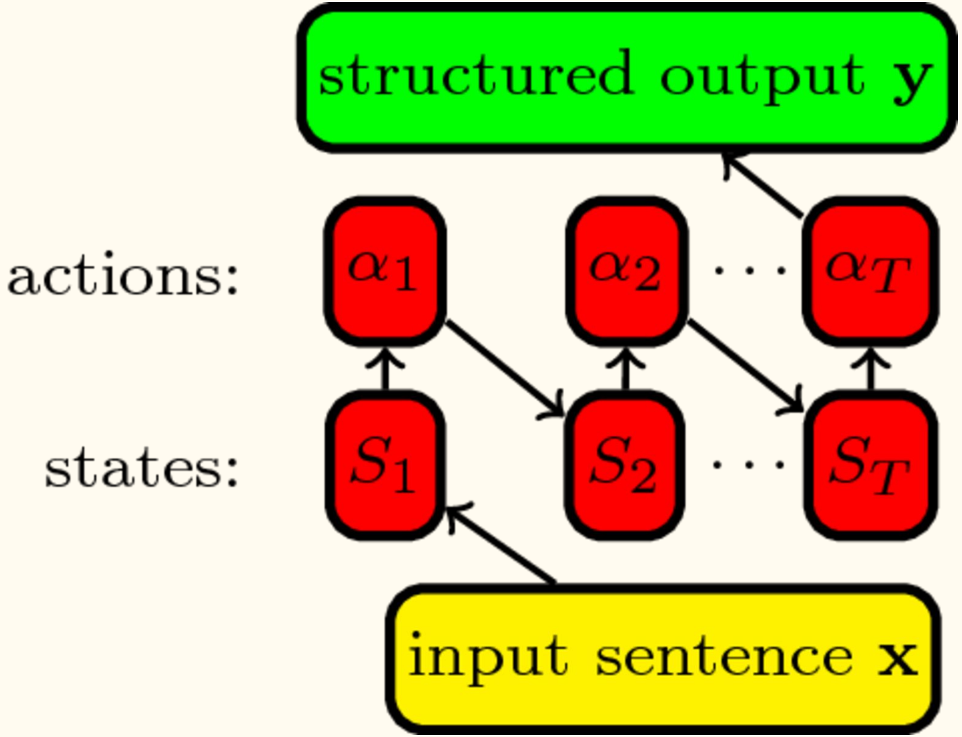
In many cases, yes!

But we need to make assumptions on the structure:

- 1st order Markov assumption (linear chains), rarely more than 2nd
- The scoring function must decompose over the output structure

What if we need greater flexibility?

Incremental structured prediction



Incremental structured prediction

A classifier **f** predicting actions to construct the output:

$$\hat{\mathbf{y}} = output\left(\begin{aligned} \hat{\alpha}_1 &= \arg\max_{\alpha \in \mathcal{A}} f(\alpha, \mathbf{x}), \\ \hat{\alpha}_2 &= \arg\max_{\alpha \in \mathcal{A}} f(\alpha, \mathbf{x}, \hat{\alpha}_1), \cdots \\ \hat{\alpha}_N &= \arg\max_{\alpha \in \mathcal{A}} f(\alpha, \mathbf{x}, \hat{\alpha}_1 \ldots \hat{\alpha}_{N-1}) \end{aligned} \right)$$

Examples:

- Predicting the PoS tags word-by-word
- Generating a sentence word-by-word

Incremental structured prediction

Pros:

✓ No need to enumerate all possible outputs

✓ No modelling restrictions on features

Cons:

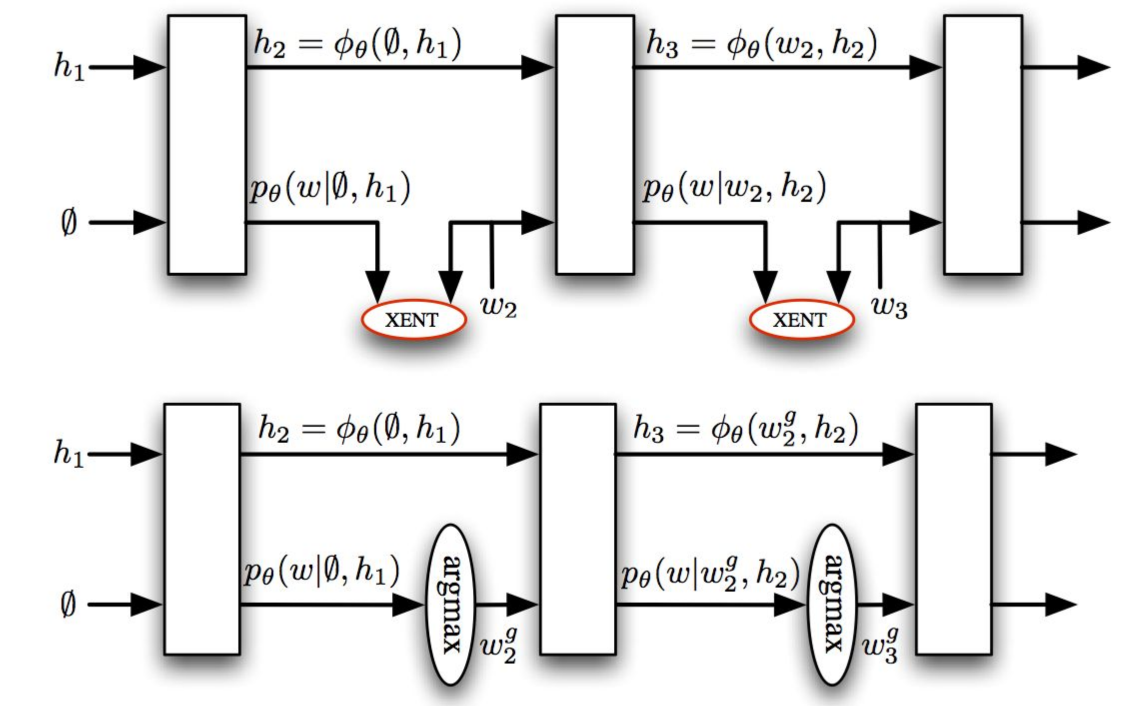
x Prone to error propagation

x Classifier not trained w.r.t. task-level loss

Error propagation

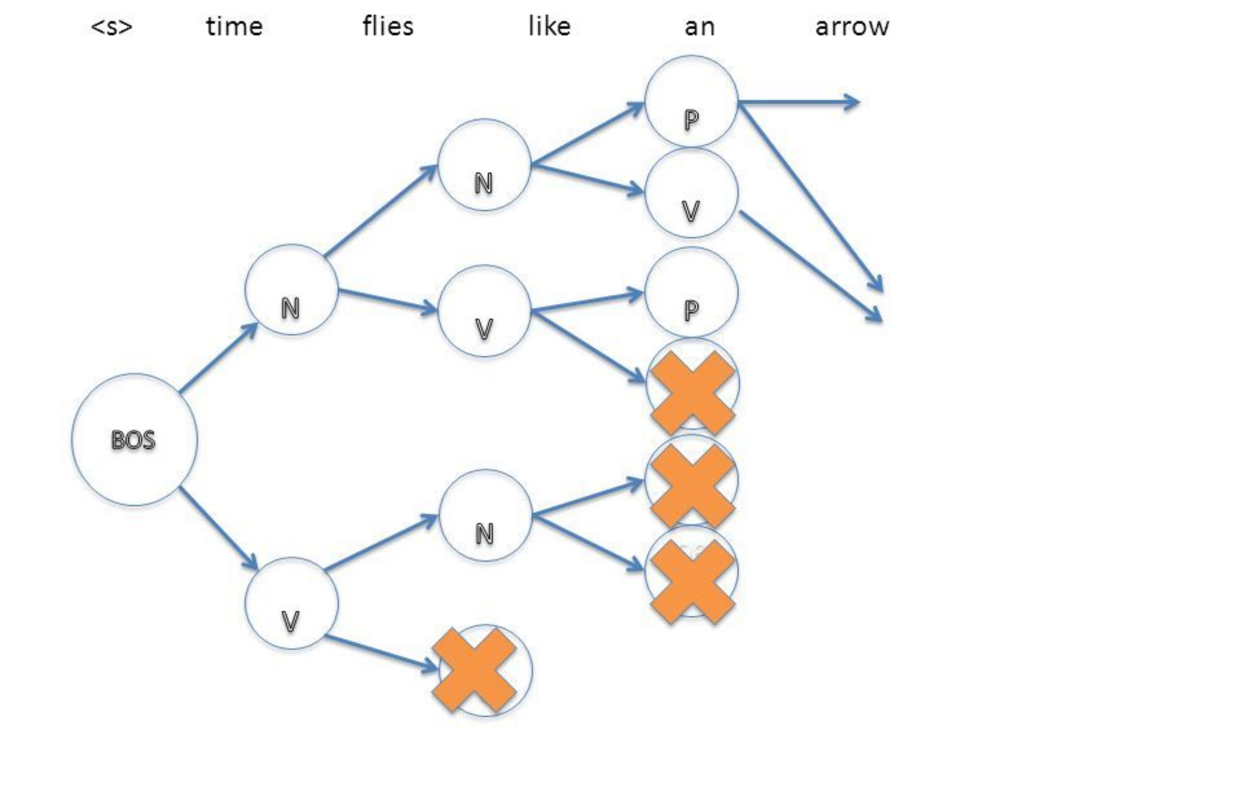
We do not score complete outputs:

- early predictions do not know what follows
- cannot be undone if purely incremental/monotonic
- we are training with gold standard predictions for previous predictions, but test with predicted ones (**exposure bias**)



Ranzato et al. (ICLR2016)

Beam search intuition



Beam size 3

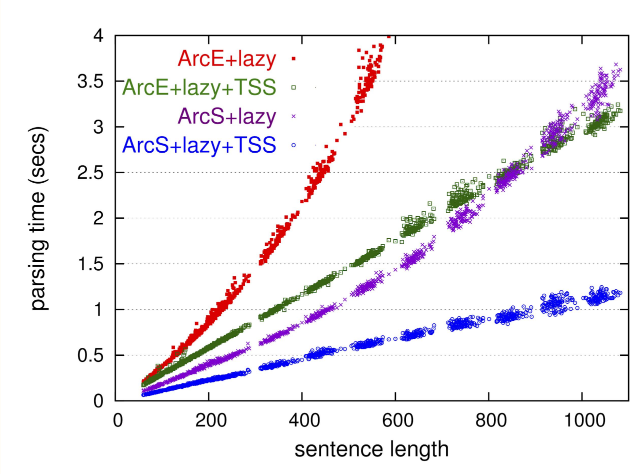
http://slideplayer.com/slide/8593664/

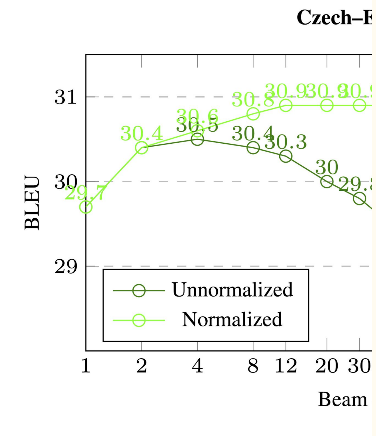
Beam search algorithm

Input: word sequence $x = [x_1, \ldots, x_N]$, tags $\mathcal{Y}$, parameters θ
Initialize beam $B = \{y_{temp} = ([START], score = 0)\}$, size k
for $n = 1 \ldots N$ **do**
  $B' = \{\}$
  for $b \in B$ **do**
    for $y \in \mathcal{Y}$ **do**
      $s = score(\mathbf{x}, [b.y_{temp}; y]); \theta)$
      $B' = B' \cup ([b.\mathbf{y_{temp}}; y], s)$
    end for
  end for
  $B = B'[1 : k]$
end for
return $B[1]$

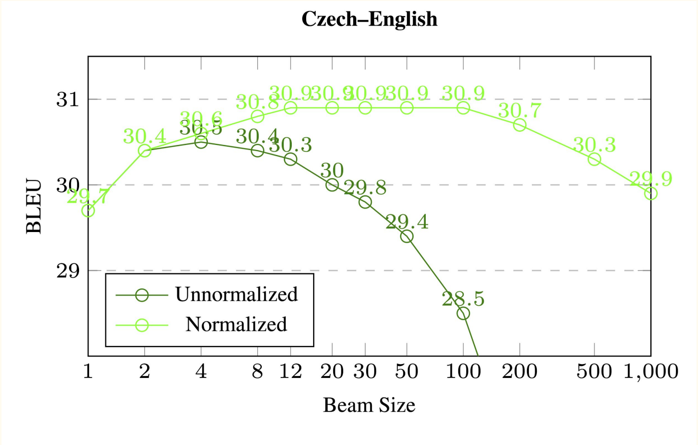
Beam search in practice

- It works, but implementation matters
  - Feature decomposability is key to reuse previously computed scores
  - Sanity check: on small/toy instances large enough beam should find the exact argmax
- Need to normalise for sentence length
- Take care of bias due to action types with different score ranges: picking among all English words is not comparable with picking among PoS tags





Being less exact helps?



Search	BLEU	Ratio	#Search errors	#Empty
Greedy	29.3	1.02	73.6%	0.0%
Beam-10	30.3	1.00	57.7%	0.0%
Exact	2.1	0.06	0.0%	51.8%

Table 1: NMT with exact inference. In the absence of search errors, NMT often prefers the empty translation, causing a dramatic drop in length ratio and BLEU.

- In Neural Machine Translation performance degrades with larger beams...
- Search errors save us from model errors!
- Part of the problem at least is that we train word-level models but the task is at the sentence-level...

Training losses for structured prediction

In supervised training we assume a loss function e.g. negative log likelihood against gold labels in classification with logistic regression/ feedforward NNs.

In structured prediction, what do we train our classifier to do?

Predict the action leading the correct output. Losses over **structured outputs**:

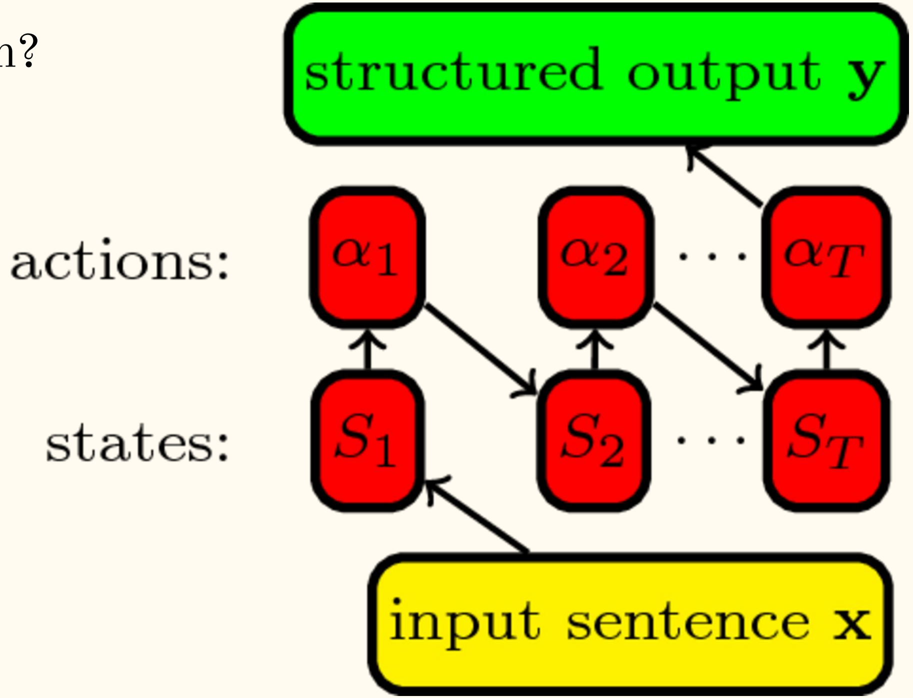
- Hamming loss: number of incorrect part of speech tags in a sentence
- False positive and false negatives: e.g. named entity recognition
- 1-BLEU score (n-gram overlap) in generation tasks, e.g. machine translation

Loss and decomposability

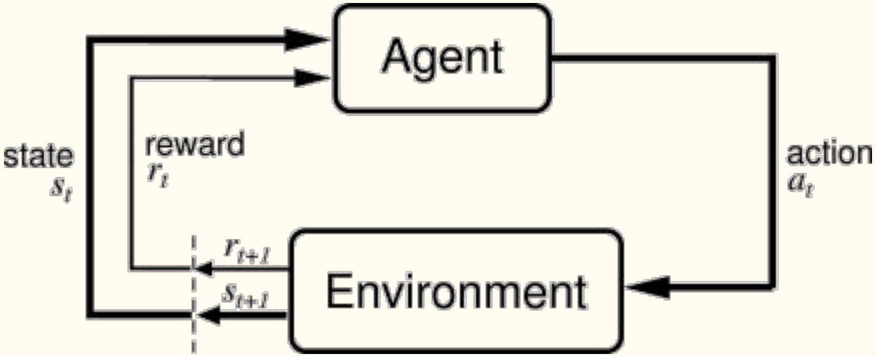
Can we assess the goodness of each action?

- In PoS tagging, predicting a tag at a time with Hamming loss?
  - **YES**
- In machine translation predicting a word at a time with BLEU score?
  - **NO**

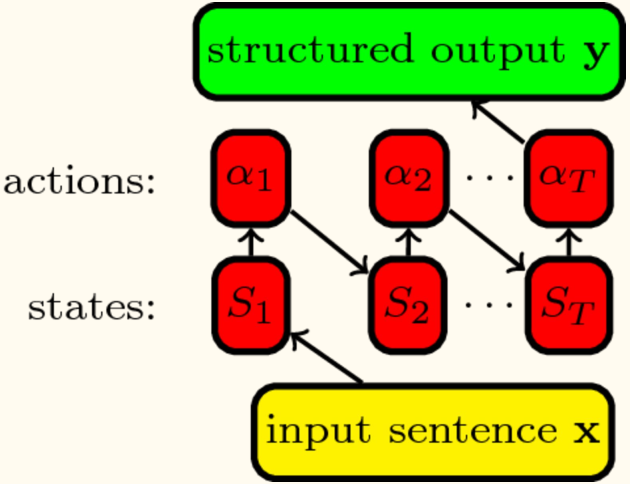
BLEU score doesn't decompose over the actions defined by the transition system



Reinforcement learning



[Sutton and Barto (2018)]



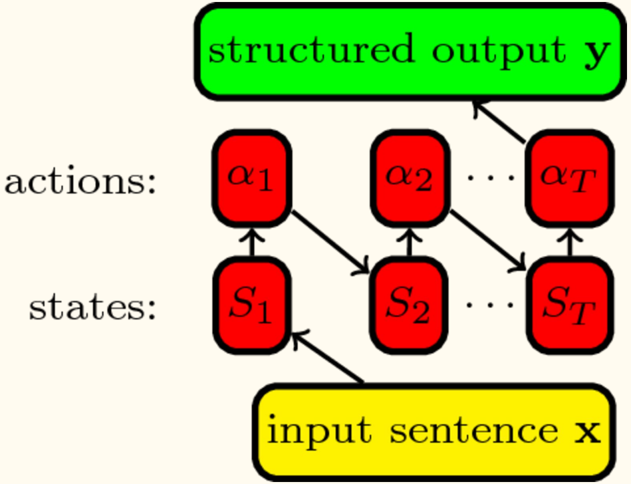
- Incremental structured prediction can be viewed as (degenerate) RL:
  - No environment dynamics
  - No need to worry about physical costs (e.g. robots damaged)

Policy gradient

We want to optimize this objective (per instance):

$$J(\theta) = v_{\pi(\theta)}(s_1)$$

- task level loss to min is the **value** v to max
- θ are the parameters of the **policy** (classifier)



We can now do our stochastic gradient (ascent) updates:

$$\theta_{t+1} = \theta + \alpha \nabla J(\theta_t)$$

What could go wrong?

Reinforcement learning is hard…

To obtain training signal we need complete trajectories

- Can sample (REINFORCE) but inefficient in large search spaces
- High variance when many actions are needed to reach the end (credit assignment problem)
- Can learn a function to evaluate at the action level (actor-critic)

In NLP, often the models are trained initially in the standard supervised way and then fine-tuned with RL

- Hard to tune the balance between the two
- Takes away some of the benefits of RL

Imitation learning

- Both reinforcement and imitation learning learn a classifier/policy to maximize reward
- Learning in imitation learning is facilitated by an **expert**

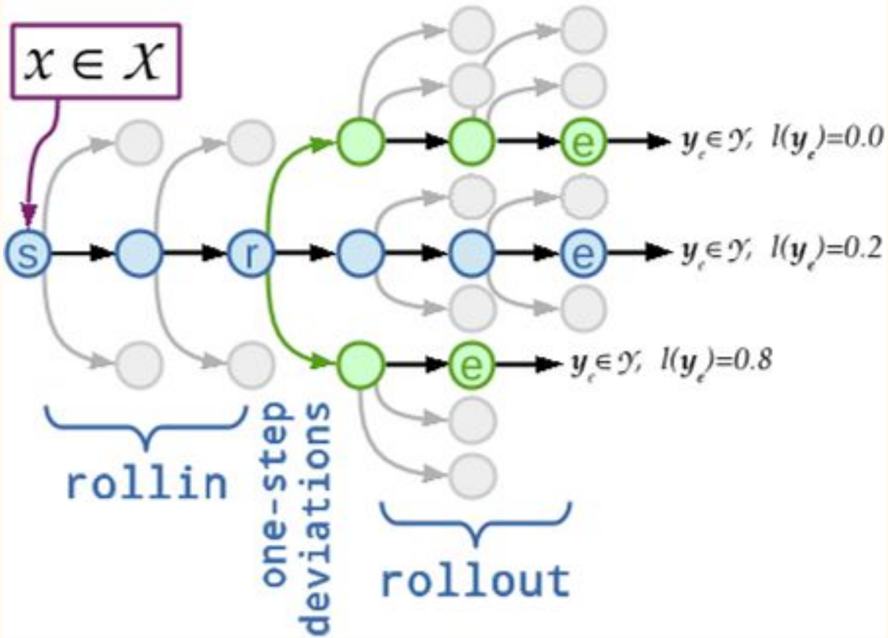
Expert policy

Returns the best action at the current state by looking at the gold standard assuming future actions are also optimal:

$$\alpha^{\star} = \pi^{\star}(S_t, \mathbf{y}) = \arg\min_{\alpha \in \mathcal{A}} L(S_t(\alpha, \pi^{\star}), \mathbf{y})$$

Only available for the training data: an expert demonstrating how to perform the task

Imitation learning in a nutshell



Chang et al. (2015)

- First iteration trained on expert, later ones increasingly use the trained model
- Exploring one-step deviations from the rollin of the classifier

Imitation learning is hard too!

- Defining a good expert is difficult
  - How to know all possible correct next words to add given a partial translation and a gold standard?
  - Without a better than random expert, we are back to RL
  - ACL 2019 best paper award was about a decent expert for MT
- While expert demonstrations make learning more efficient, it is still difficult to handle large numbers of actions
- Iterative training can be computationally expensive with large dataset
- The interaction between learning the feature extraction and learning the policy/classifier is not well understood in the context of RNNs

Bibliography

- [Kai Zhao's survey]
- [Noah Smith's book]
- [Sutton and Barton Reinforcement learning book]
- [Imitation learning tutorial]