From words to sentences

L101: Machine Learning for Language Processing

Andreas Vlachos, Pushkar Mishra



Language modelling reminder

Decompose the probability of the sentence ***x*** into conditional probabilities of each word given the previous ones:

$$P(\mathbf{x}) = P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)\dots P(x_N|x_1, \dots, x_{N-1})$$

A.k.a., next-word prediction.

This is how we estimate the probability of a sentence given a model.

But how do we decide which **sentence** to generate next, given some input?

Structured prediction in a nutshell

Given an input **x** (e.g. a sentence) predict **y**:

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

Where $\boldsymbol{Y}$ is rather large and often depends on the input

A well-studied problem in NLP and ML:

- Linear models (structured perceptron)
- Probabilistic linear models (conditional random fields)
- Applications such as part-of-speech tagging, named entity recognition, object recognition

Decoding

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

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

Dynamic programming to the rescue?

Yes! But we need to make assumptions on the structure:

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

What if we need greater flexibility?

Incremental structured prediction

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

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

Examples:

- Predicting the PoS tags word-by-word
- Generating a sentence via **next-word prediction**

Incremental structured prediction

Pros:

✓ No need to enumerate all possible outputs

✓ No modelling restrictions on features

Cons:

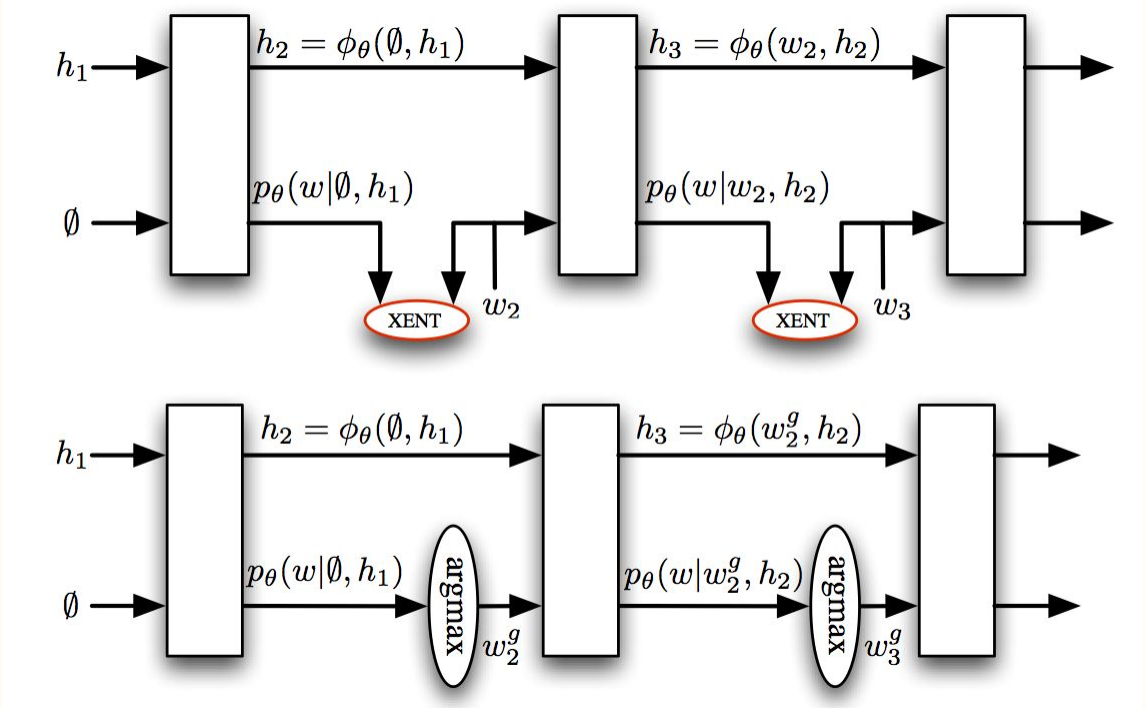
x Prone to error propagation

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

Error propagation

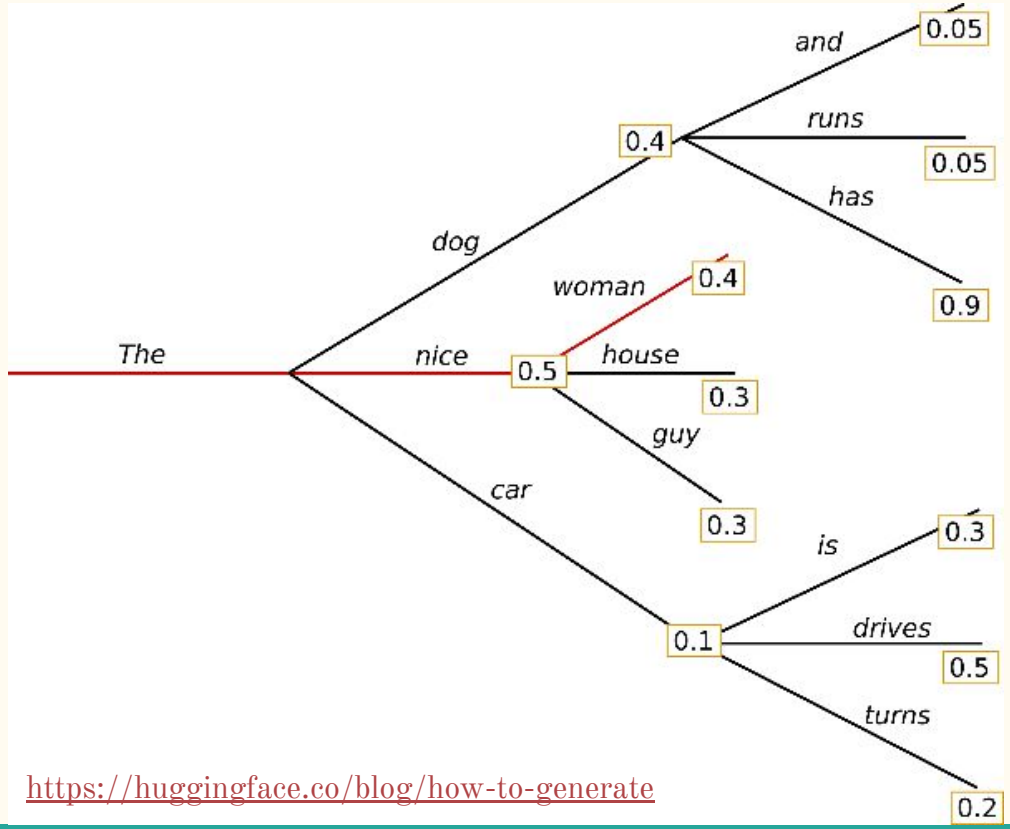
We do not score complete outputs:

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



Ranzato et al. (ICLR2016)

Incremental basics: Greedy and Beam search



https://huggingface.co/blog/how-to-generate

Greedy: pick the most likely action (“the nice woman”)

Beam: keep the top-k paths alive (“the dog has” with k=2)

Overcome locally optimal decisions that are not globally optimal **according to the model**

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 extensions

Reranking:

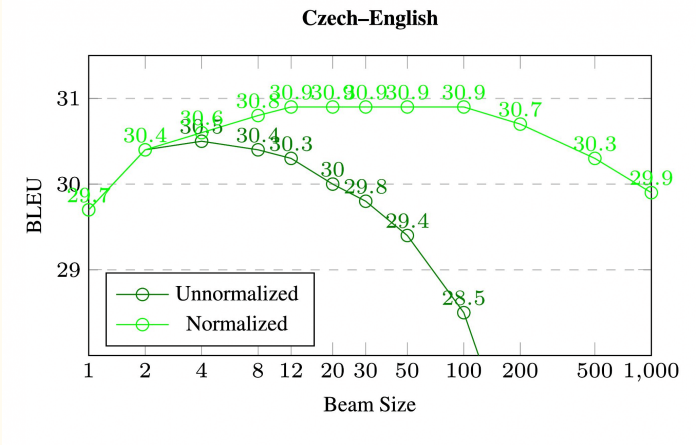
- Adjust probabilities to normalise for sentence length
- Model to pick outputs that are likely to have better global score (e.g. BLEU)
- Re-rank intermediate beams, a.k.a. incremental beam manipulation

We still rely on beam search to generate good hypotheses (for good reasons?)

Training decoders for beam search:

- Penalize the model when the correct hypothesis falls of the beam (beam search optimization, beam-aware training)
- Train a greedy decoder to approximate beam search to maximize a sentence-level score

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!
  - Also MAP decoding might not be doing justice to our models
- Part of the problem at least is that we train word-level models but the task makes (a lot more!) sense at the sentence-level really...

Training for incremental 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 incremental structured prediction, what do we train our classifier to do?

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

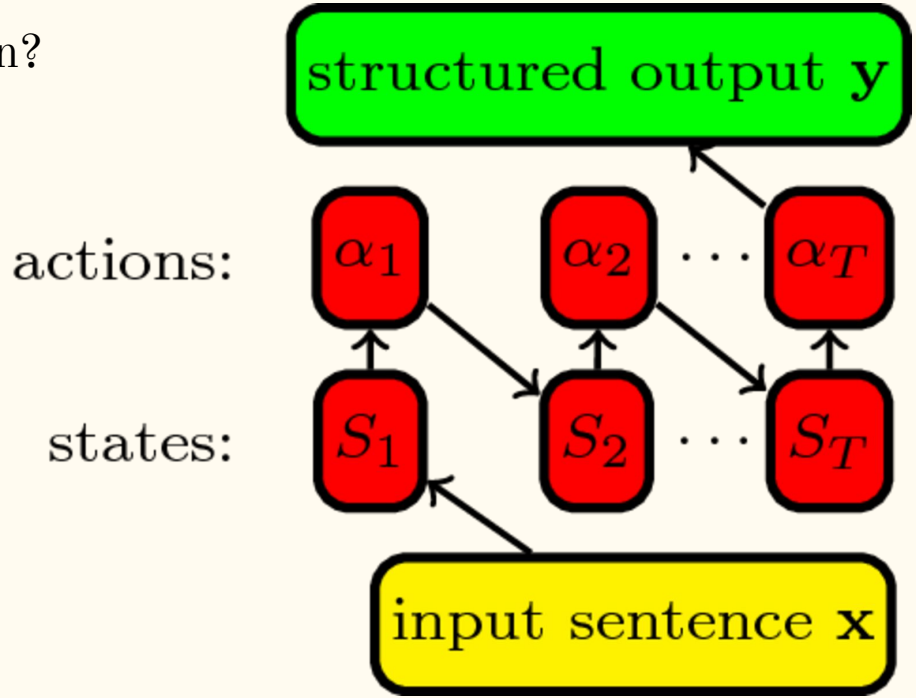
- Hamming loss: number of incorrect part of speech tags in a sentence
- Reduction in BLEU score (n-gram overlap) in generation tasks, e.g. machine translation

Loss and decomposability

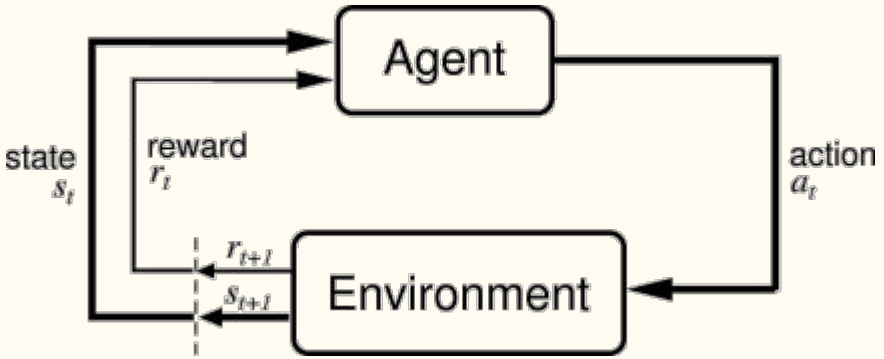
Can we assess the goodness of each action?

- 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**

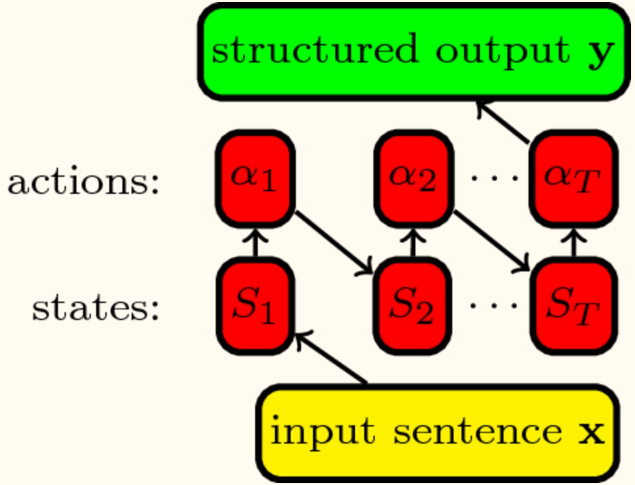
BLEU score doesn't decompose over the actions defined by the transition system



Reinforcement learning



Sutton and Barto (2018)



Incremental structured prediction can be viewed as (degenerate) RL:

- No environment dynamics
- No need to worry about physical wear and tear (e.g. robots damaged)

Policy gradient

Learn the parameters $\boldsymbol{\theta}$ of policy/classifier $\boldsymbol{\pi}$ that optimize rewards/task loss v:

$$J(\theta) = \sum_{s \in S} d^{\pi_\theta}(s) v^{\pi_\theta}(s)$$
$$= \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_{\alpha \in \mathcal{A}} \pi_\theta(\alpha|s) Q^{\pi_\theta}(s, \alpha)$$

- on-policy learning: the policy affects the distributions of states visited d
- the reward from reaching a state s is its expectation according to the policy

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...

See Choshen et al. (2020), and Kiegeland and Kreutzer (2021) for a recent debate
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 Q to evaluate the outcome of the action (actor-critic)

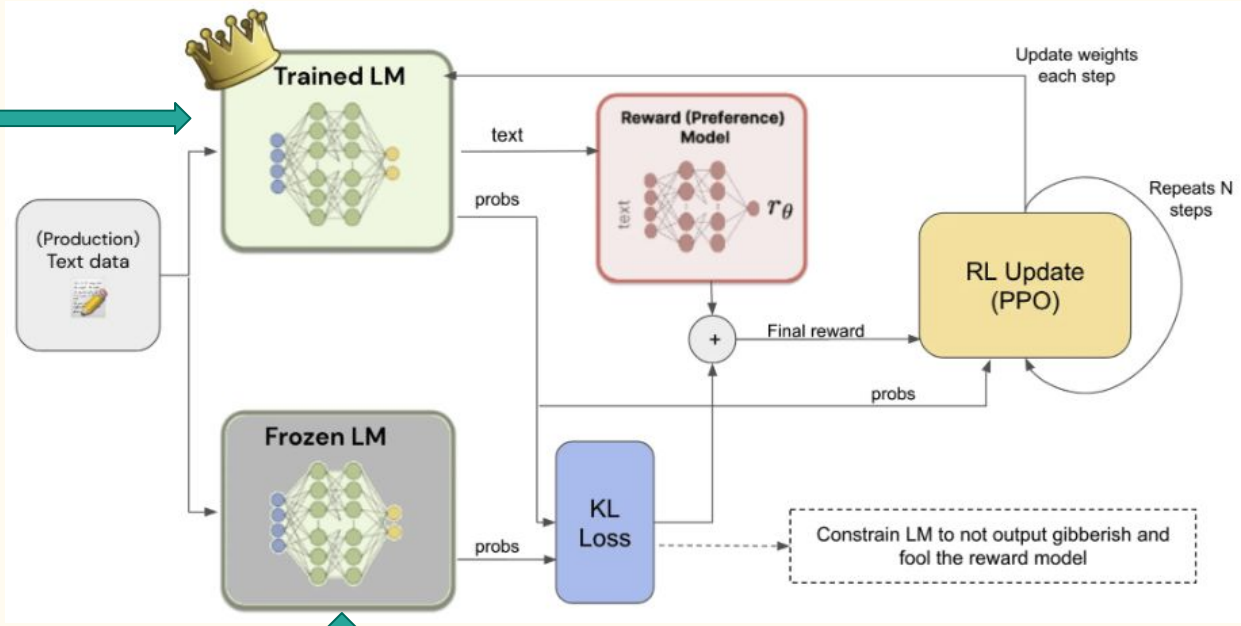
In NLP, often the models are trained initially in the standard supervised way and then fine-tuned with RL (e.g. for summarization)

- Hard to tune the balance between the two
- Constrains the benefits of RL

Reinforcement learning is hard... but what if

- We had a good starting point

Pretrained / SFTed model



Staying close to the starting point (regularizer)

Credits: Joao Lages

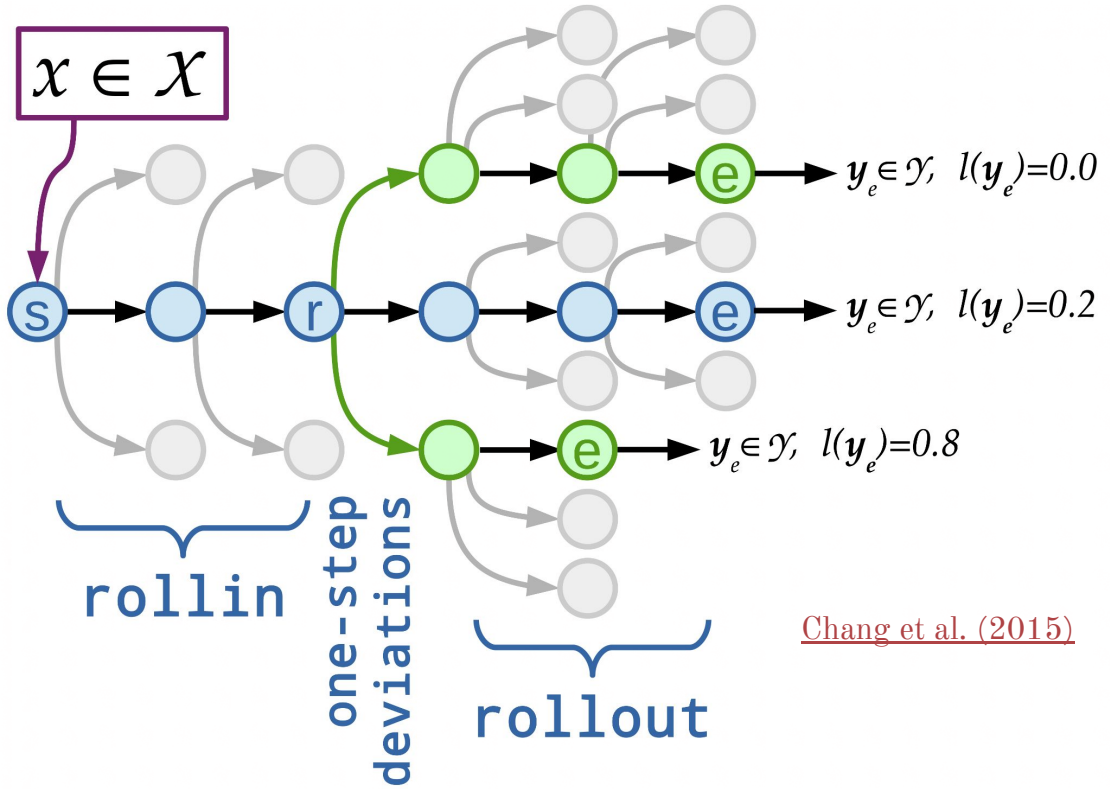
Imitation learning





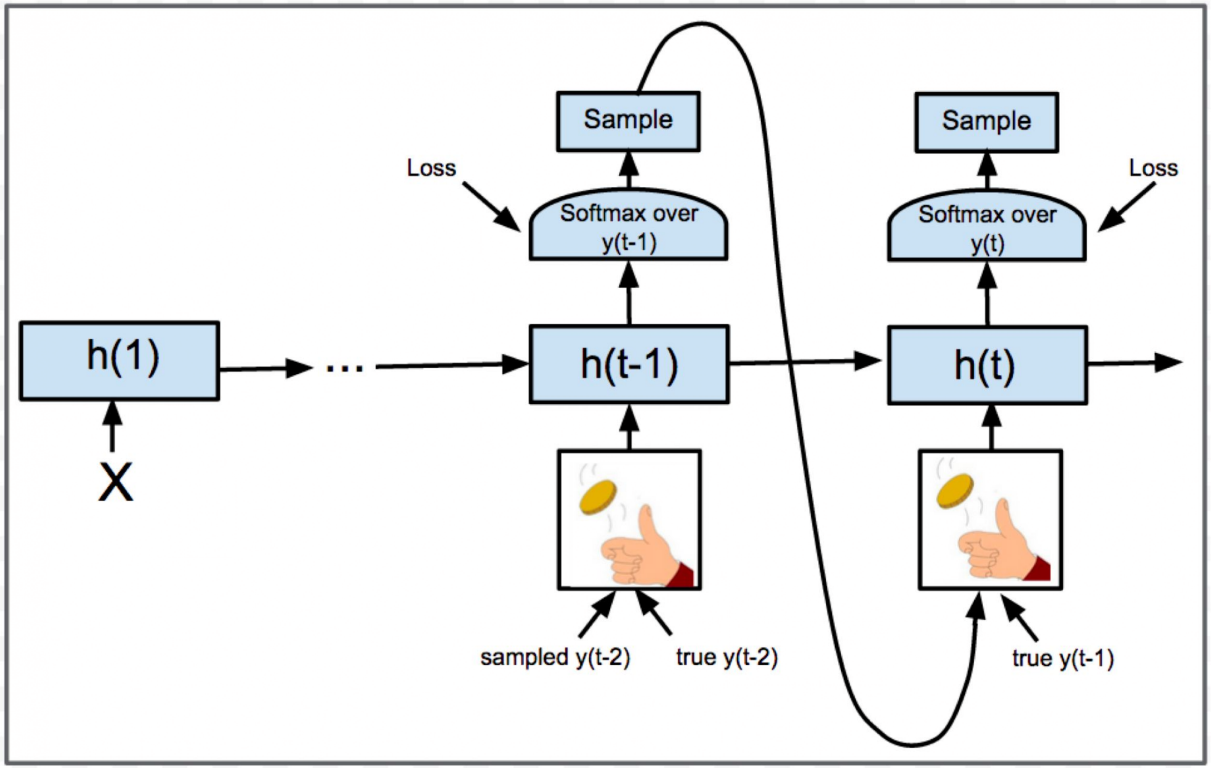
- Both reinforcement and imitation learning learn a classifier/policy to maximize reward
- Learning in imitation learning is facilitated by an **expert**
- Basic form: supervised learning using expert demonstrations, a.k.a behavioural cloning; IL algorithms go beyond this

Imitation learning in a nutshell



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

Scheduled sampling



Train without assuming that all previous words are correctly predicted

This idea was first introduced as the DAgger algorithm in robotics

Works, but make sure you don't forget when the previous words are correct!

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 “dynamic” expert, we are back to RL

- While expert demonstrations make learning more efficient, it is still difficult to handle large numbers of actions

- Recent work has shown that LLMs can be used for generating on-policy expert corrections

Identifying experts in the new world

- Larger models usually do tasks much better than smaller models
  - Knowledge distillation with larger models as the “dynamic” experts
  - Soft distillation, i.e., distilling not just the chosen action, but possibility of other actions too
- Why not get smaller models to imitate the actions of larger model?
  - **Greater variability**: many different trajectories can be explored
  - **Richer feedback**: larger models can provide rationales for picking actions
- See any problems?
  - **Bias amplification**: if the larger model has biases, smaller models will
  - **Saturation**: the capabilities of smaller models are inherently capped at those of the larger model

Bibliography

- Noah Smith's book
- Kevin Murphy's tutorial
- This blog on policy gradient methods
- Imitation learning tutorial