10-418/10-618 Machine Learning for Structured Data

Machine Learning Department
School of Computer Science
Carnegie Mellon University

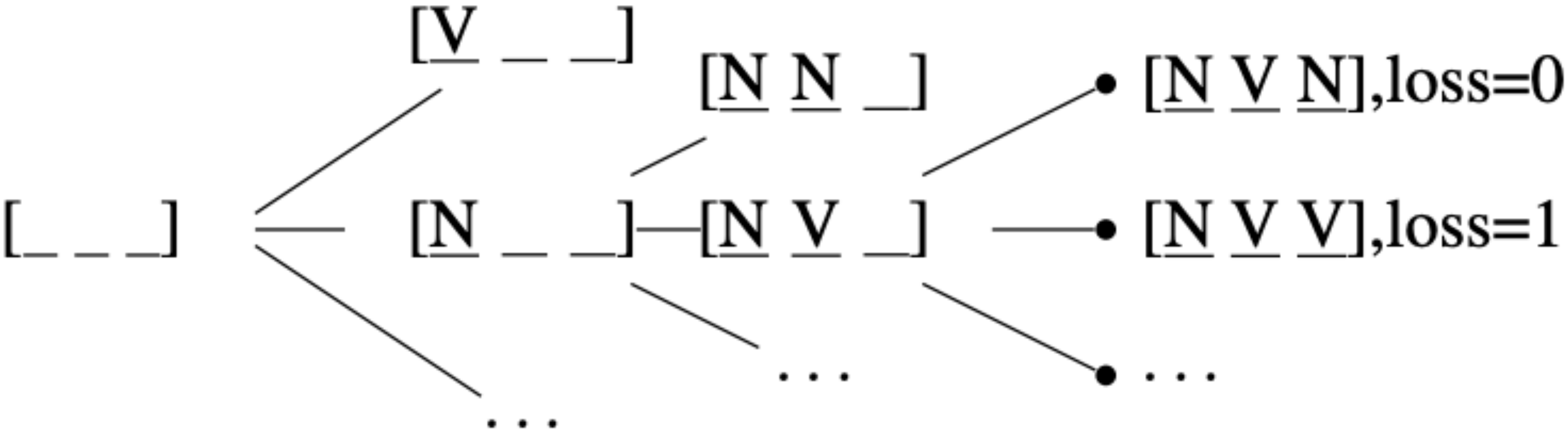
Learning to Search (Part III)

Matt Gormley
Lecture 6
Sep. 18, 2022

STRUCTURED PREDICTION AS SEARCH

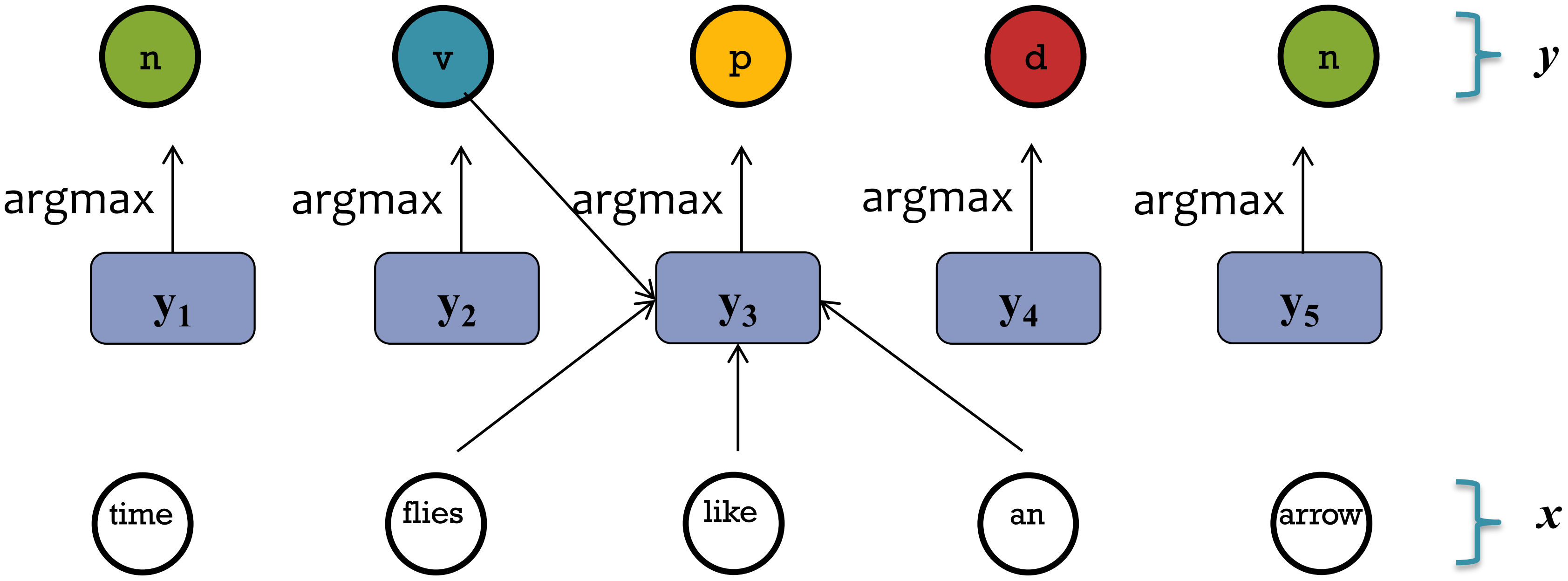
Structured Prediction as Search

- **Key idea:** convert your structured prediction problem to a search problem!
- **Example:** for POS tagging, each node in the search space corresponds to a partial tag sequence



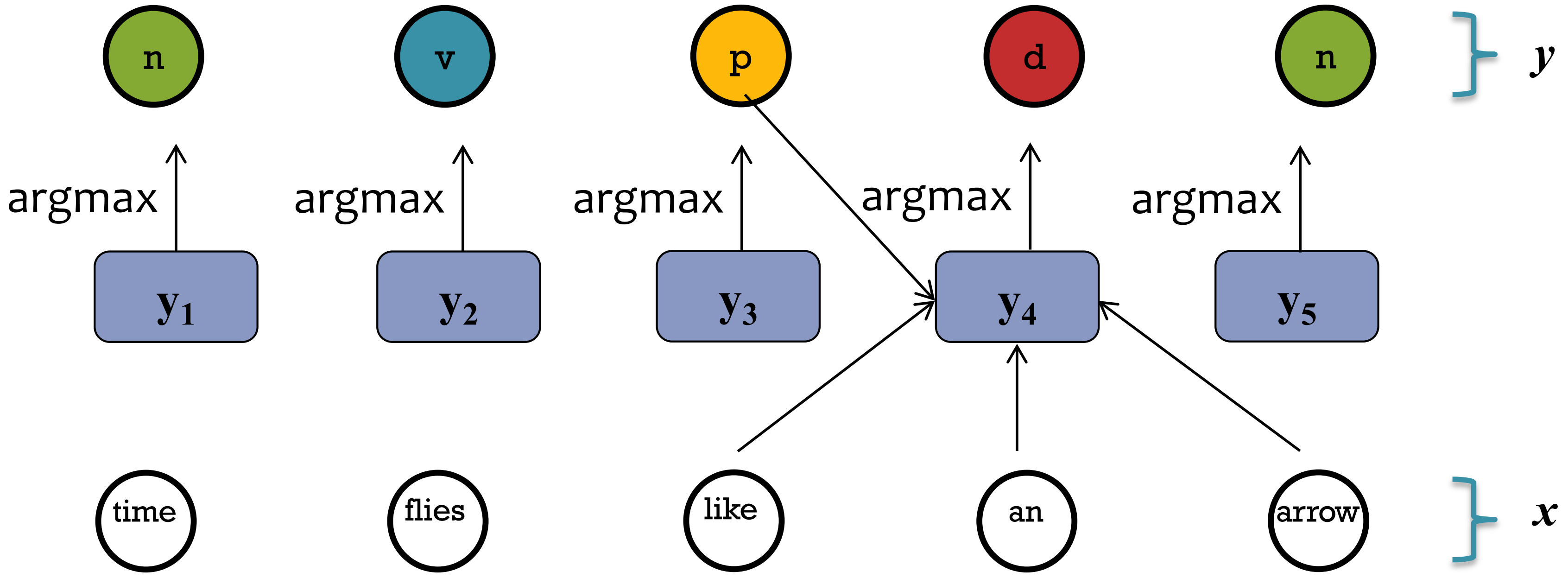
Basic Neural Network

- Suppose we wish to predict the tags **greedily left to right**
- Simple neural network looks at the previous word, the previous tag **prediction**, the current word, and the next word
- From these it builds a **probability distribution over output tags**
- Then it **selects the argmax**



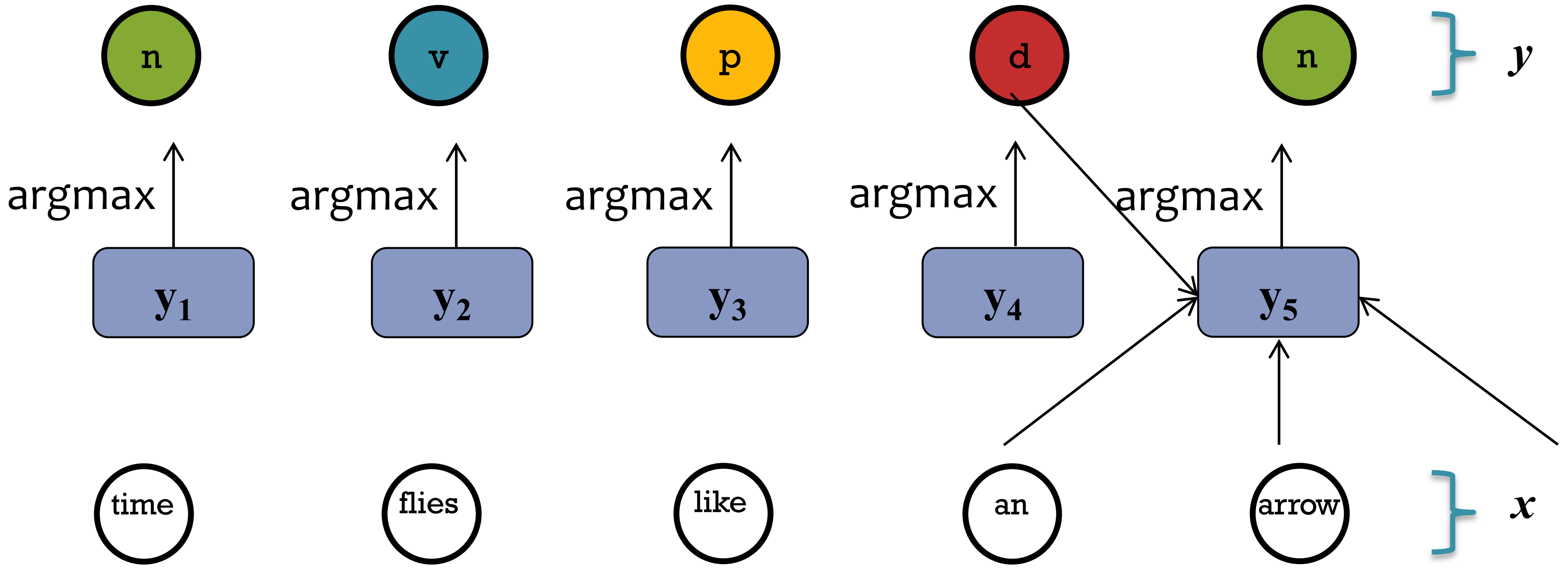
Basic Neural Network

- Suppose we wish to predict the tags **greedily left to right**
- Simple neural network looks at the previous word, the previous tag **prediction**, the current word, and the next word
- From these it builds a **probability distribution over output tags**
- Then it **selects the argmax**



Basic Neural Network

- Suppose we wish to predict the tags **greedily left to right**
- Simple neural network looks at the previous word, the previous tag **prediction**, the current word, and the next word
- From these it builds a **probability distribution over output tags**
- Then it **selects the argmax**



Learning to Search

Whiteboard:

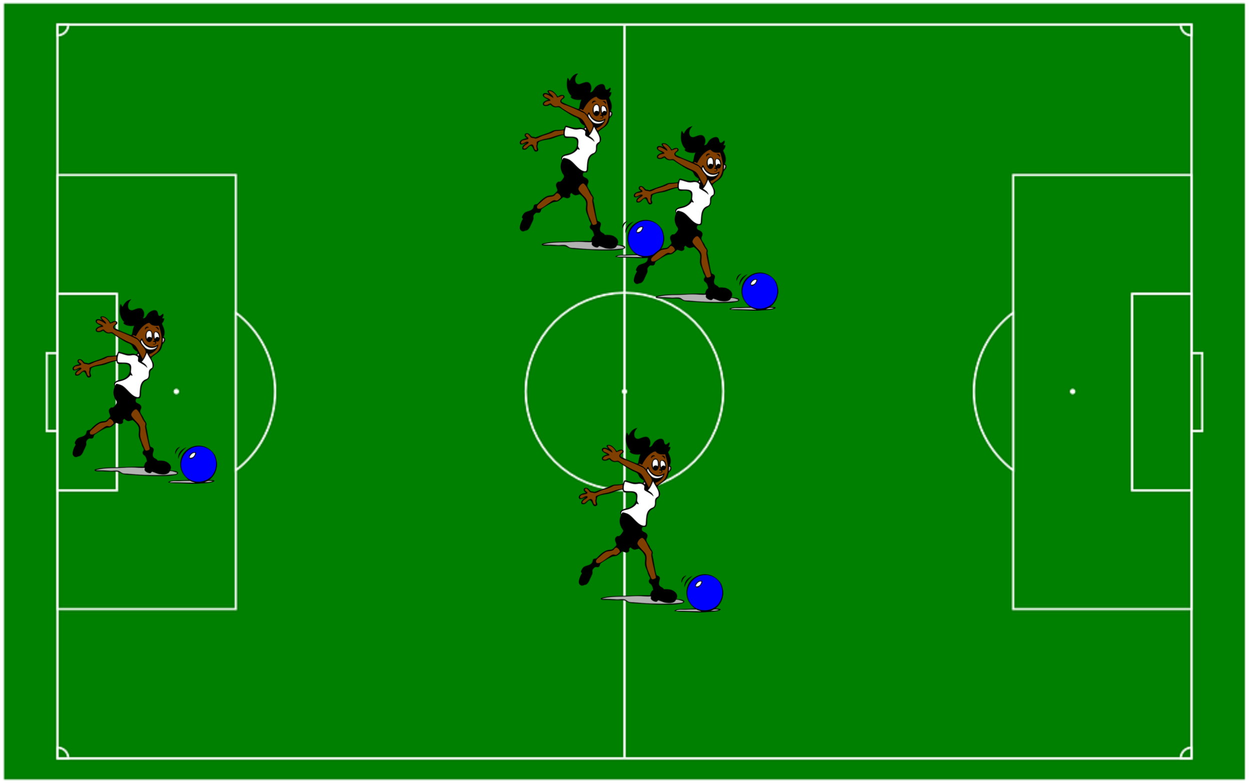
- Problem Setting
- Ex: POS Tagging
- Other Solutions:
  - Completely Independent Predictions
  - Sharing Parameters / Multi-task Learning
  - Graphical Models
- Today's Solution: Structured Prediction to Search
  - Search spaces
  - Cost functions
  - Policies

EXPOSURE BIAS

The Exposure Bias Problem

Imagine you join (for the first time ever) a intramural soccer team.

- **Position, 1st practice:** midfielder
- **Position, 2nd practice:** midfielder
- **Position, 3rd practice:** midfielder
- **Position, 1st game:** goalie



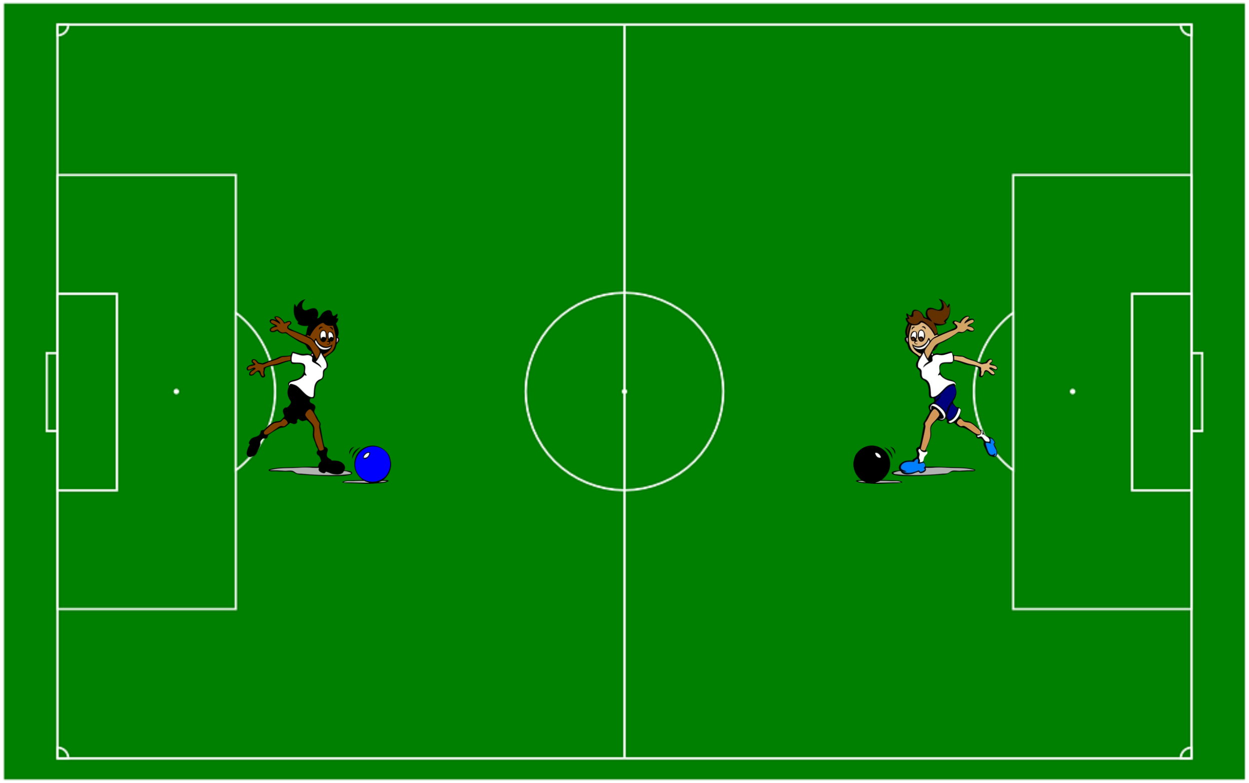
This could end badly! After all, you've never had any training on what to do as a goalie.

Your training is **biased** to the parts of the field that you were **exposed** to (midfield).

The Exposure Bias Problem

To really make this analogy work, we should adjust it slightly:

- **1st, 2nd, 3rd, practice**: your coach assigns each of you to mimic the exact steps of a specific player in the 1986 world cup final; every practice each of you make the same maneuvers every time
- **1st game**: you play a real team, that makes creative autonomous decisions, you are forced to be creative as well



This could end badly! After all, you've never had any training on what to do when you are exposed to the outcomes of your own decisions!

Your training is **biased** to the 1986 behaviors to which you were **exposed**.

The Exposure Bias Problem

Consider two (related) explanations for why maximum likelihood training (aka. fully supervised imitation learning) is a poor choice for a seq2seq model.

Mismatched Test-Time Inference Algorithm

MLE training assumes that at test time we will do exact inference to find the highest probability output string (e.g. for an HMM, using the Viterbi algorithm)

However, in general, exact inference over the space of exponentially many output strings is intractable for a seq2seq model (e.g. unlike an HMM, seq2seq makes no Markov assumption)

beam search

Exposure Bias

At training time, MLE trains to generate the next token conditioned on the ground truth prefix sequence.

At test time, the model generates the next token conditioned on model's prefix sequence (obtained by greedy decoding or sampling).

greedy policy

So the model is exposed to only real prefixes at training time, and is therefore biased towards proper behavior only on those ground truth prefixes.

LEARNING TO SEARCH

Imitation Learning

Algorithm 1: Supervised Imitation Learning (for structured prediction)

```
def trainSupervised($\pi^*$, E, $x_{1:S}$, $y_{1:T}$):
    initialize policy $\pi_\theta$
    for i in 1… N:
        for t in 1… T:
            observe state $s_t = (\boxed{y_{1:t}}, x_{1:S})$
            take action $a_t = \boxed{y_{t+1}} = \pi^*(s_t)$
            $\tau^{(i)} = \tau^{(i)} + [(s_t, a_t)]$
        for $(s_t, a_t)$ in $\tau^{(i)}$:
            update policy $\pi_\theta$ with
            one step of SGD on
            example $(s_t, a_t)$
    repeat for E epochs
    return $\pi_\theta$
```

```
def predict($\pi_\theta$):
    for t in 1… T:
        observe state $s_t = (y_{1:t}, x_{1:S})$
        take action $a_t = \hat{y}_{t+1} = \pi_\theta(s_t)$
        incur loss $\ell_t = \text{loss}(y_{1:t}, \hat{y}_{1:t})$
```

The state is the full input sequence and the partial output sequence, up to step t

The action is the next output token we ~~predict~~ at step t+1 from our ground truth

This is equivalent to accumulating the gradient of the full computation graph for input $x_{1:S}$

Imitation Learning

Algorithm 2: DAgger for Imitation Learning (for structured prediction)

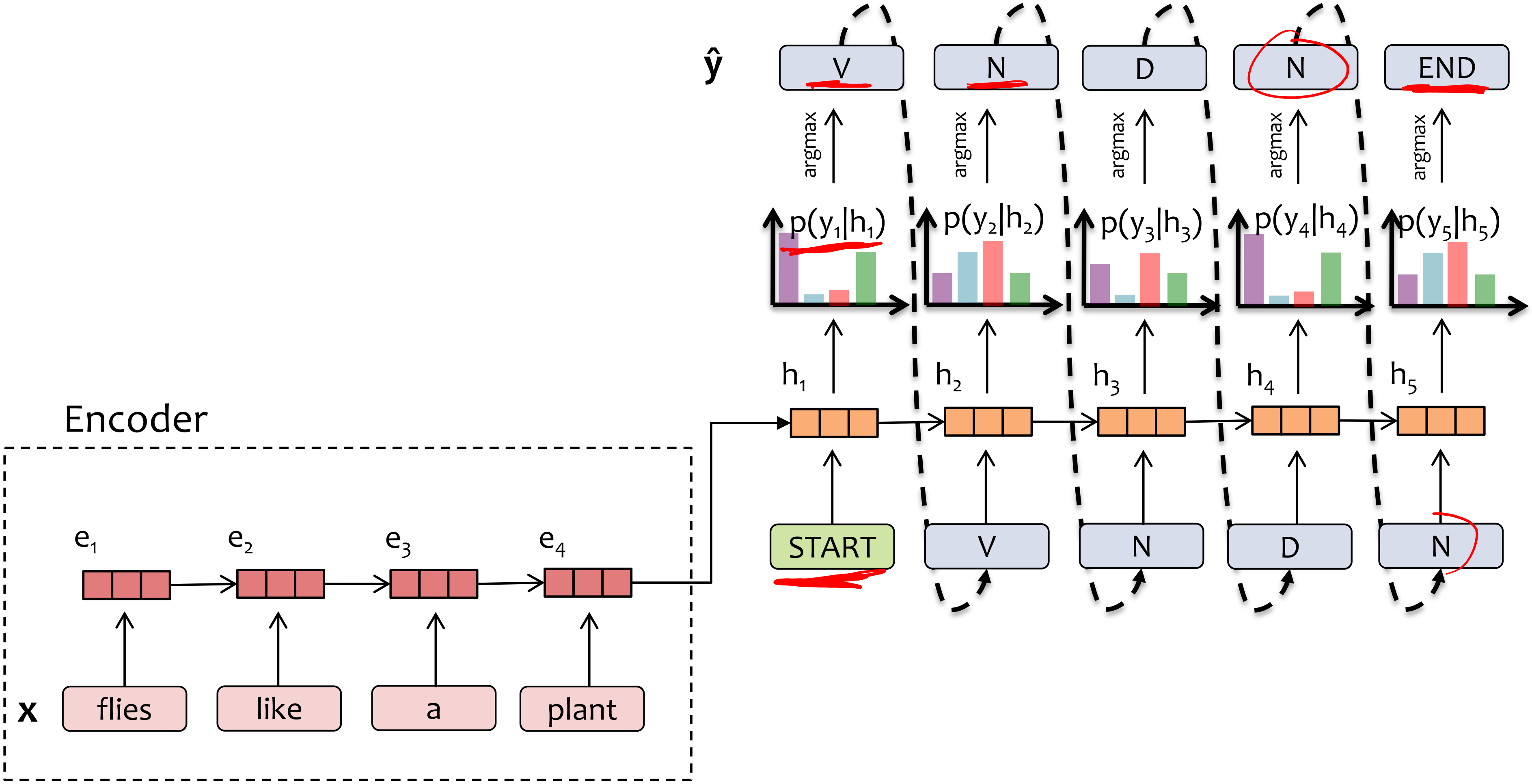
```
def trainDAgger($\pi^*$, E, $\boldsymbol{\beta} = [\beta_1,\dots,\beta_N]$, $x_{1:S}$, $y_{1:T}$):
    initialize policy $\pi_\theta$
    for i in 1… N:
        $\pi_i = \beta_i \pi^* + (1 - \beta_i)\pi_\theta$
        for t in 1… T:
            observe state $s_t = (y_{1:t}, x_{1:S})$
            sample action $\hat{a}_t = \hat{y}_{t+1} \sim \pi_i(s_t)$
            store action $a_t = y_{t+1} = \pi^*(s_t)$
            $\boldsymbol{\tau}^{(i)} = \boldsymbol{\tau}^{(i)} + [(s_t, a_t)]$
        for $(s_t, a_t)$ in $\boldsymbol{\tau}^{(i)}$:
            update policy $\pi_\theta$ with
            one step of SGD on
            example $(s_t, a_t)$

    repeat for E epochs
    return $\pi_\theta$
```

Now the prediction for the next time step $\hat{y}_{t+1}$ comes from the the model policy. + oracle policy

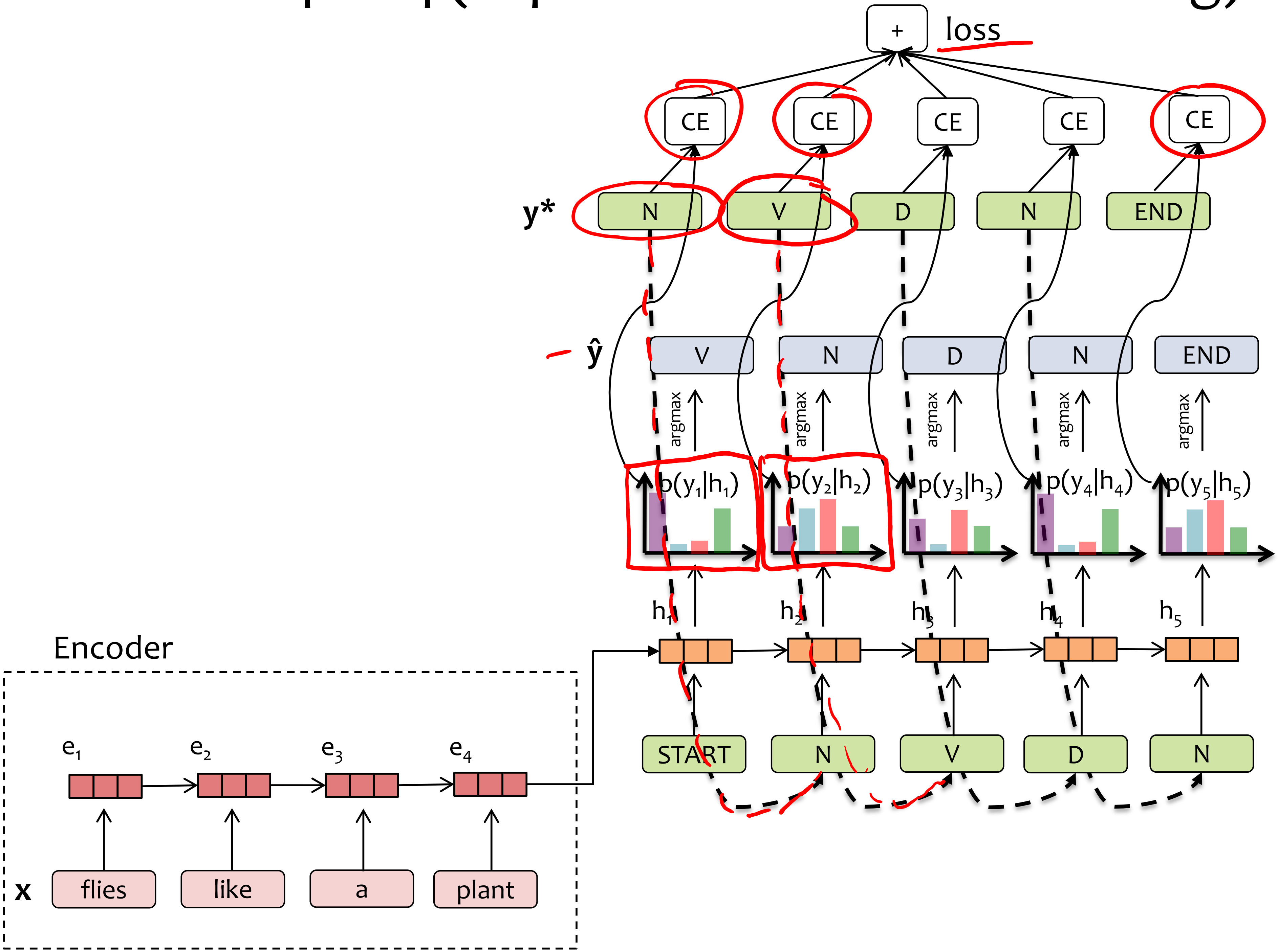
This will typically be fed back into the model.

We still train by updating on the expert policy's prediction of y_{t+1} – this is the output of the "dynamic oracle"

Decoding for seq2seq (test time)



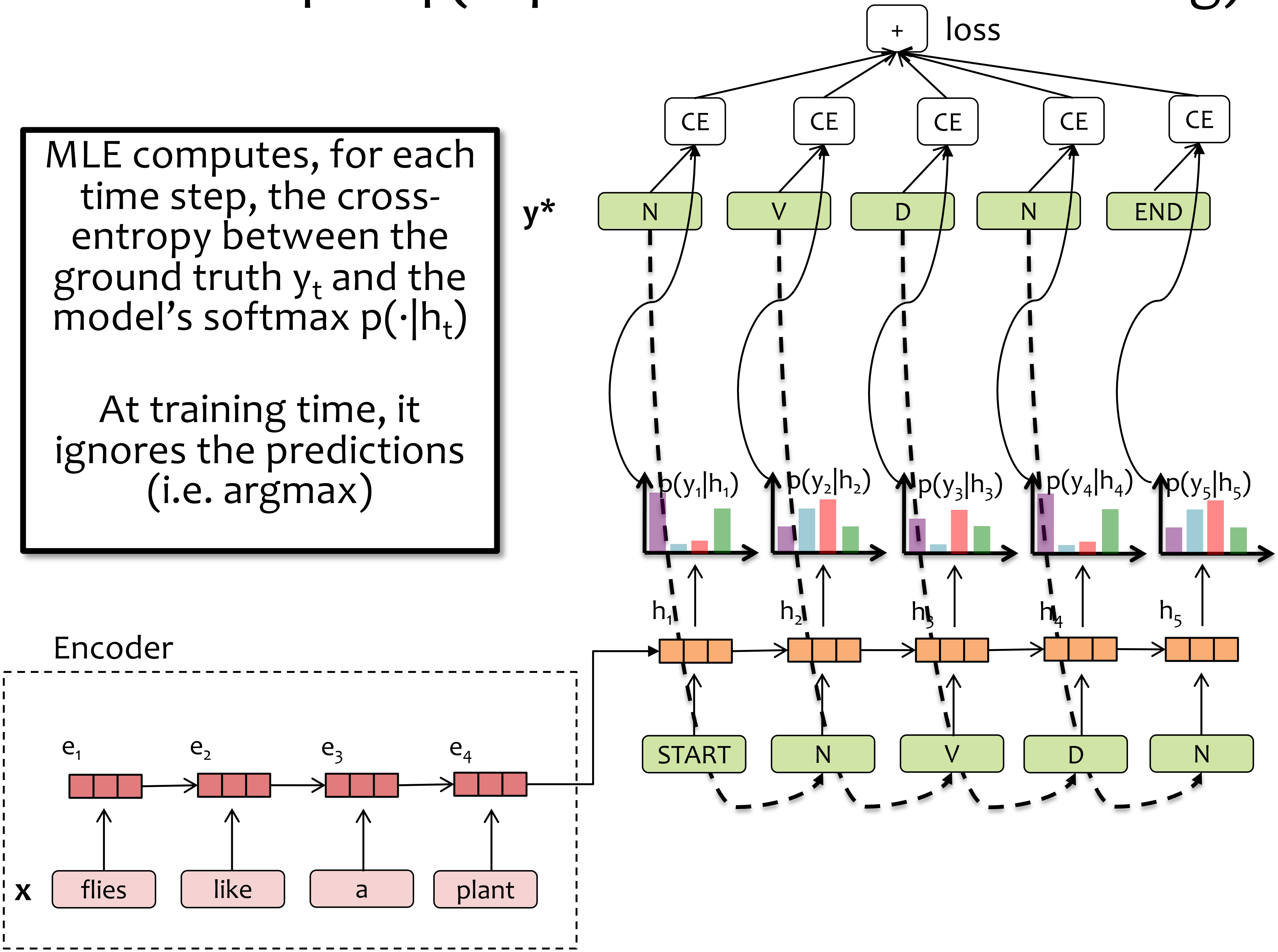
MLE for seq2seq (supervised imitation learning)



MLE for seq2seq (supervised imitation learning)

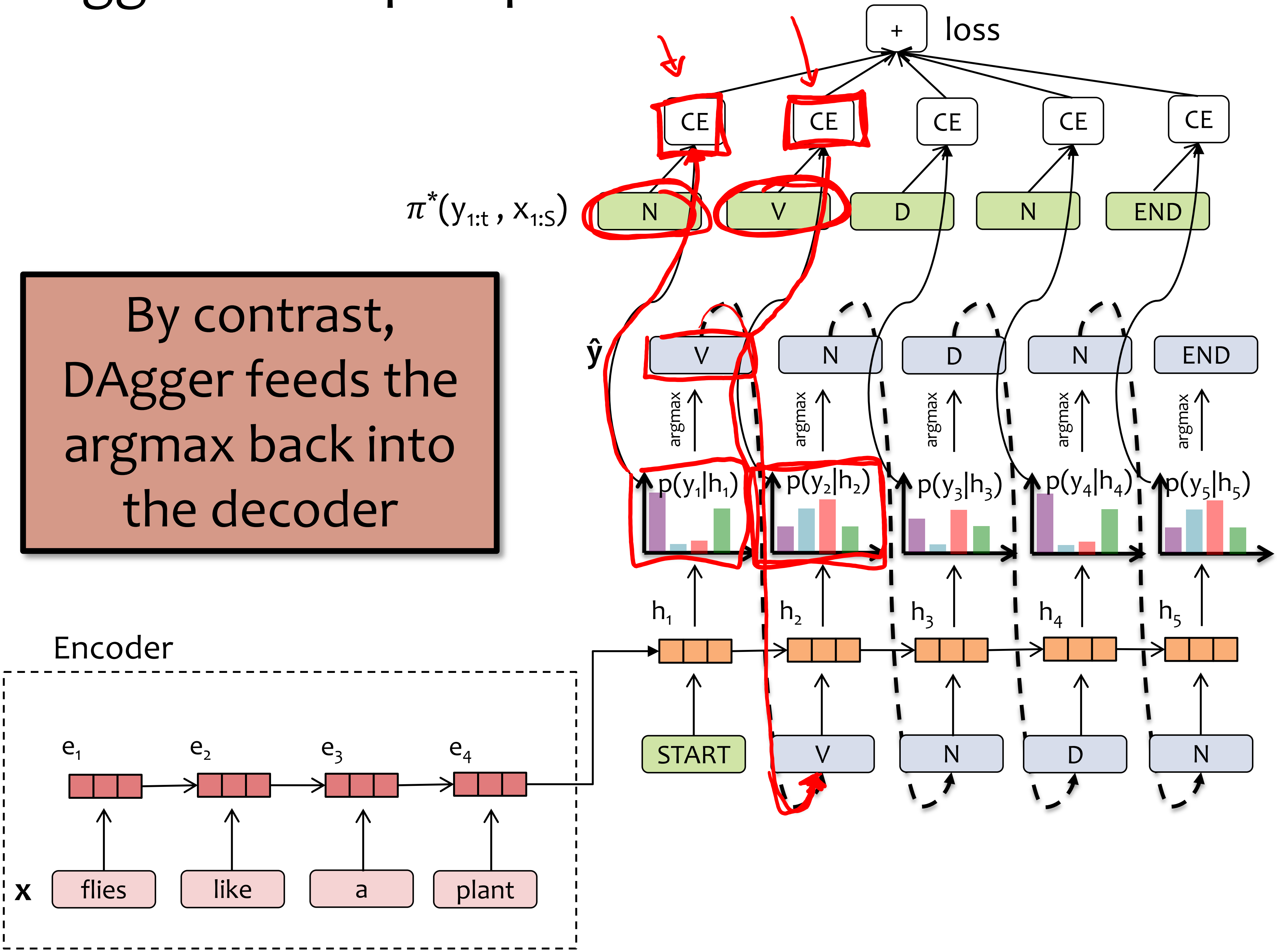
MLE computes, for each time step, the cross-entropy between the ground truth y_t and the model's softmax $p(\cdot|h_t)$

At training time, it ignores the predictions (i.e. argmax)



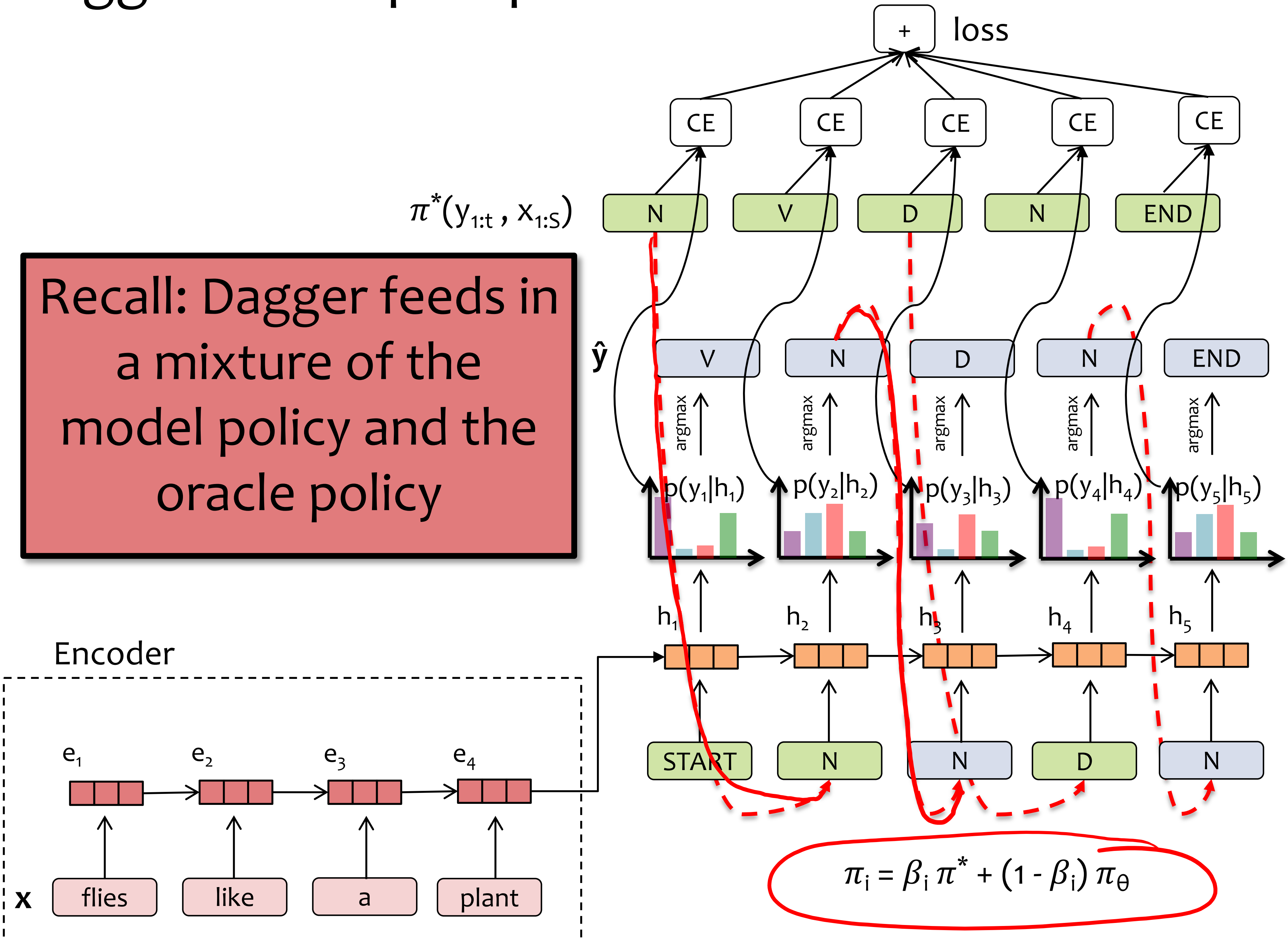
DAgger for seq2seq

By contrast, DAgger feeds the argmax back into the decoder



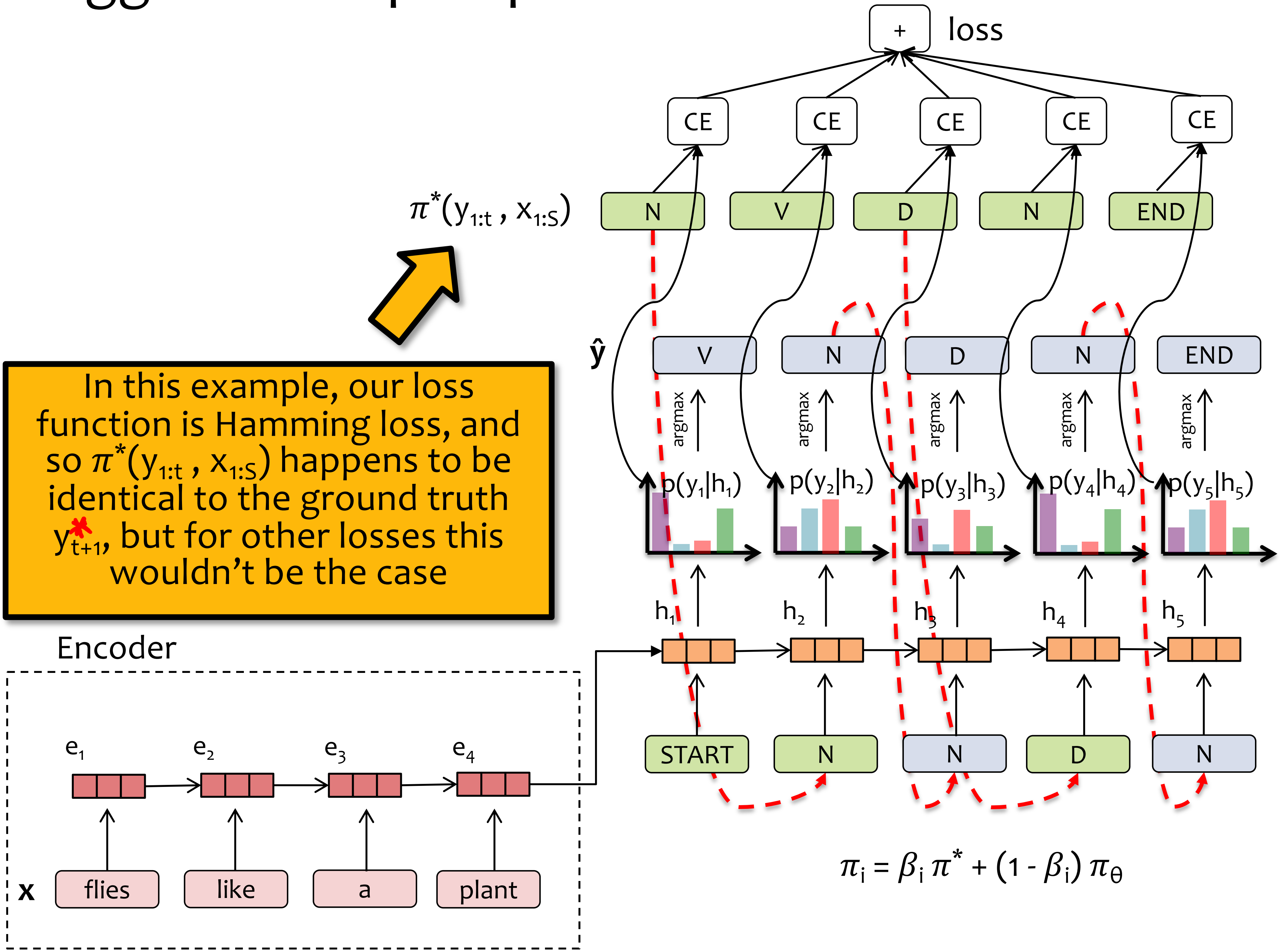
DAgger for seq2seq

Recall: Dagger feeds in a mixture of the model policy and the oracle policy



$$\pi_i = \beta_i \pi^* + (1 - \beta_i)\, \pi_\theta$$

DAgger for seq2seq



In this example, our loss function is Hamming loss, and so $\pi^*(y_{1:t}, x_{1:S})$ happens to be identical to the ground truth y^*_{t+1}, but for other losses this wouldn't be the case

$$\pi_i = \beta_i \pi^* + (1 - \beta_i) \pi_\theta$$

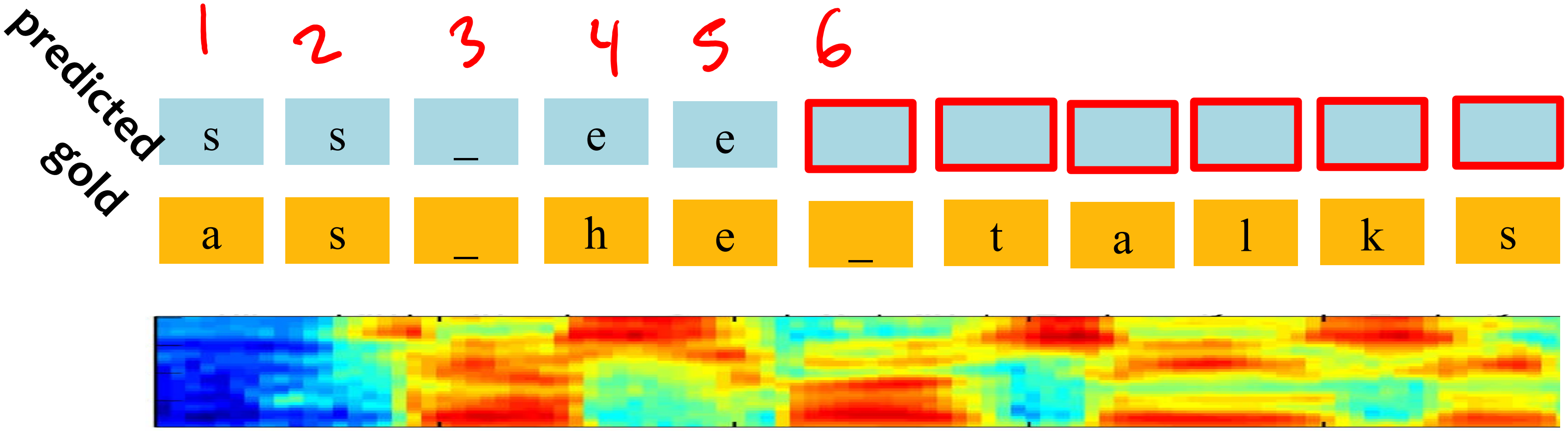
aka. Expert Policies for structured prediction problems

DYNAMIC ORACLES

Dynamic Oracles

Definition: a **dynamic oracle** is a function that answers this question:

Given a partial output sequence, what is the completion that minimizes loss w.r.t the gold output?



predicted

gold

1 2 3 4 5 6

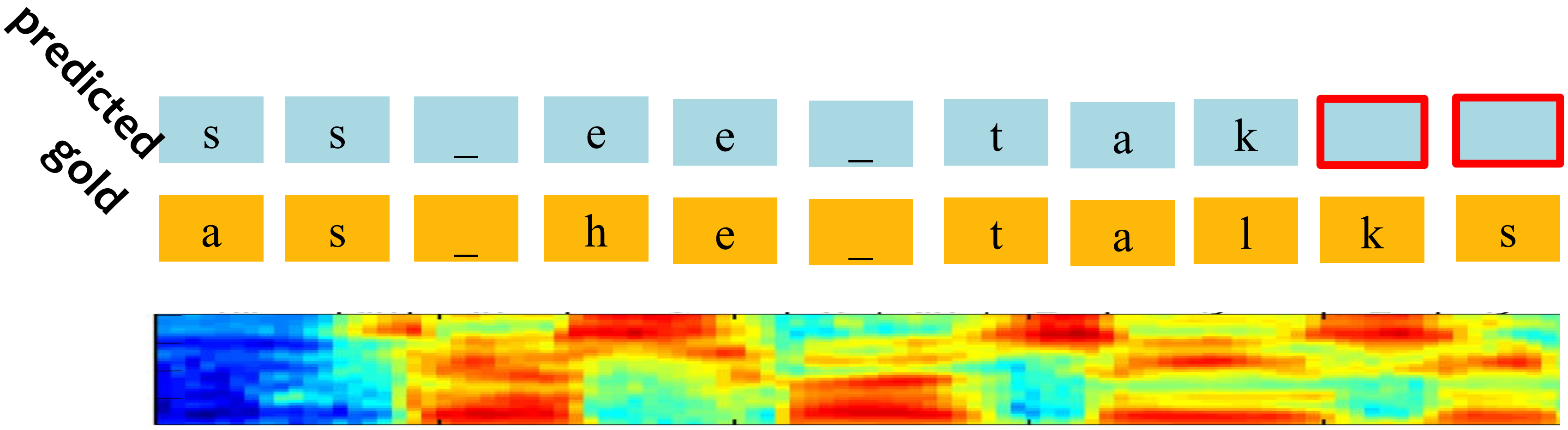
s s _ e e

a s _ h e _ t a l k s

Dynamic Oracles

Definition: a **dynamic oracle** is a function that answers this question:

Given a partial output sequence, what is the completion that minimizes loss w.r.t the gold output?



Background: Levenshtein Distance

- **Given:** two strings, a source string and a target string
- **Definition:** Levenshtein distance (aka. edit distance) is the smallest number of insertions, deletions, and substitutions to transform the source string into the target string
- **Example Transformations:**
  - leaning → learning (insertion of ‘r’)
  - reading → leading (substitution of ‘r’ for ‘l’)
  - learning → earning (deletion of ‘l’)
- **Recursive Definition:** (but not quite how it’s implemented)

$$\mathrm{dist}(\overset{\text{source}}{a_{1:A}}, \overset{\text{target}}{b_{1:B}}) = \begin{cases} \max(A, B) & \text{if } A = 0 \text{ or } B = 0 \\ \mathrm{dist}(a_{2:A}, b_{2:B}) & \text{if } a_1 = b_1 \\ 1 + \min \begin{cases} \mathrm{dist}(a_{2:A}, b_{1:B}) \\ \mathrm{dist}(a_{1:A}, b_{2:B}) \\ \mathrm{dist}(a_{2:A}, b_{2:B}) \end{cases} & \text{otherwise} \end{cases}$$

Background: Levenshtein Distance

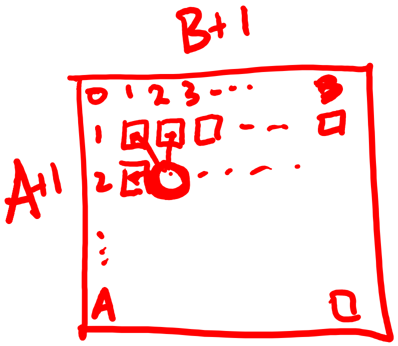
- **Given:** two strings, a source string and a target string
- **Definition:** Levenshtein distance (aka. edit distance) is the smallest number of insertions, deletions, and substitutions to transform the source string into the target string
- **Example Transformations:**
  - leaning → learning (insertion of 'r')
  - **r**eading → **l**eading (substitution of 'r' for 'l')
  - **l**earning → earning (deletion of 'l')
- **Recursive Definition:** (but not quite how it's implemented)

$$\mathrm{dist}(a_{1:A}, b_{1:B}) = \begin{cases} \max(A, B) & \text{if } A = 0 \text{ or } B = 0 \\ \min \begin{cases} \mathrm{dist}(a_{2:A}, b_{1:B}) \underline{+ 1} \\ \mathrm{dist}(a_{1:A}, b_{2:B}) \underline{+ 1} \\ \underline{\mathrm{dist}(a_{2:A}, b_{2:B})} + \underline{\mathbb{1}(a_1 \neq b_1)} \end{cases} & \text{otherwise} \end{cases}$$

Background: Levenshtein Distance

- **Algorithm:** use dynamic programming; store the distance of every pair of substrings in table so that we don't recompute any distances

```
for i = 0…A: table[i][0] = i
for j = 0…B: table[0][j] = j

for i = 1…A:
    for j = 1…B:
        if a[i] != b[j]:
            sub = 1
        else:
            sub = 0
        table[i][j] = min( table[i-1, j] + 1,
                           table[i, j-1] + 1,
                           table[i-1, j-1] + sub )
```



return table[A][B]

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M’ = [b_{m+1} for m in M]

$b_{1:B}$

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1								

$a_{1:A}$

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M' = [b_{m+1} for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0							

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M’ = [b_{m+1} for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1						

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M' = $[b_{m+1}$ for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2					

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M' = $[b_{m+1}$ for m in M$]$

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3				

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M’ = [b_{m+1} for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4			

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M' = [b_{m+1} for m in M]

Edit Distance Table							OCD Targets	min. dist.
		1	2	3	4	5	6		
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4	5		

$M = \{1\}$

$M' = \{b_{1+1}\} = \{U\}$

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are $M' = [b_{m+1} \text{ for m in } M]$

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4	5	U	

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M’ = [b_{m+1} for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4	5	U	0

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M' = [b_{m+1} for m in M]

Edit Distance Table		1	2	3				OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4	5	U	0
A	2	1	1	2	3	3	4		1

$M = \{1, 2\}$

$M' = \{b_2, b_3\} = \uparrow$

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M' = [b_{m+1} for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4	5	U	0
A	2	1	1	2	3	3	4	U,N	1

UNDAY

NDAY

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M' = [b_{m+1} for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4	5	U	0
A	2	1	1	2	3	3	4	U,N	1
T	3	2	2	2	3	4	4	U,N,D	2

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M’ = [b_{m+1} for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4	5	U	0
A	2	1	1	2	3	3	4	U,N	1
T	3	2	2	2	3	4	4	U,N,D	2
R	4	3	3	3	3	4	5	U,N,D,A	3

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M' = [b_{m+1} for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4	5	U	0
A	2	1	1	2	3	3	4	U,N	1
T	3	2	2	2	3	4	4	U,N,D	2
R	4	3	3	3	3	4	5	U,N,D,A	3
A	5	4	4	4	4	3	4	Y	3

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M' = [b_{m+1} for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4	5	U	0
A	2	1	1	2	3	3	4	U,N	1
T	3	2	2	2	3	4	4	U,N,D	2
R	4	3	3	3	3	4	5	U,N,D,A	3
A	5	4	4	4	4	3	4	Y	3
P	6	5	5	5	5	4	4	Y, </s>	4

Dynamic Oracle for CER

Algorithm:

1. Build edit distance table from current prefix, $a_{1:A}$, and ground truth, $b_{1:B}$
2. Let M denote the set of column indices that are minima in the last row
3. The optimal next characters for the last row are M' = [b_{m+1} for m in M]

Edit Distance Table								OCD Targets	min. dist.
		S	U	N	D	A	Y		
	0	1	2	3	4	5	6	S	0
S	1	0	1	2	3	4	5	U	0
A	2	1	1	2	3	3	4	U,N	1
T	3	2	2	2	3	4	4	U,N,D	2
R	4	3	3	3	3	4	5	U,N,D,A	3
A	5	4	4	4	4	3	4	Y	3
P	6	5	5	5	5	4	4	Y, </s>	4
Y	7	6	6	6	6	5	4	</s>	4

Dynamic Oracles

Definition: a **dynamic oracle** is a function that answers this question:

Given a partial output sequence, what is the completion that minimizes loss w.r.t the gold output?

Depending on your loss function, there **may or may not** exist an efficient dynamic oracle.

For example, these metrics readily admit an efficient dynamic oracle:

- CER for phoneme recognition
- chunking F1 for named entity
- labeled attachment score for arc-eager dependency parsing

But it's unlikely that there exists an efficient dynamic oracle for:

- BLEU for machine translation
- ROUGE for summarization

TEACHER FORCING, SCHEDULED SAMPLING, & OCD

Algorithms for Seq2Seq

Here we consider three algorithms that are closely related to DAgger and appropriate for training a seq2seq model:

1. Teacher Forcing
2. Scheduled Sampling
3. Optimal Completion Distillation

Teacher Forcing

Another name for **supervised imitation learning** when it is applied to training RNNLM/seq2seq models is **teaching forcing**. This is equivalent to **maximum likelihood estimation** if cross-entropy is the loss function.

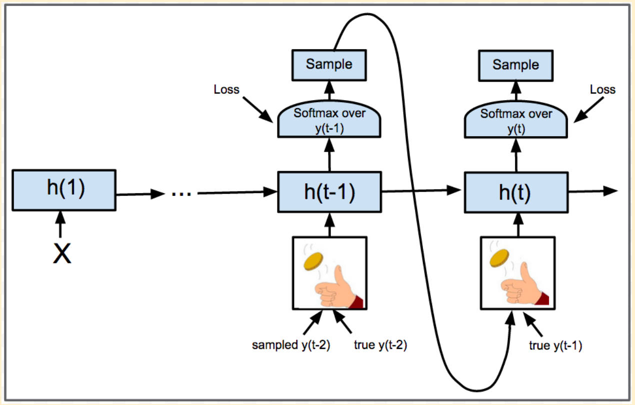
The name ‘teacher forcing’ was coined by Williams & Zipser (1989) *A Learning Algorithm for Continually Running Fully Recurrent Neural Networks*.

The only difference between training an RNNLM or seq2seq model is whether the decoder conditions on the output of an encoder or not.

Algorithm:

1. at training time, feed the **ground truth** from the **previous** time step in as the decoder input for the **next** time step
2. at each timestep **minimize cross entropy** (or some other loss) of the **ground truth** for that time step

Scheduled Sampling



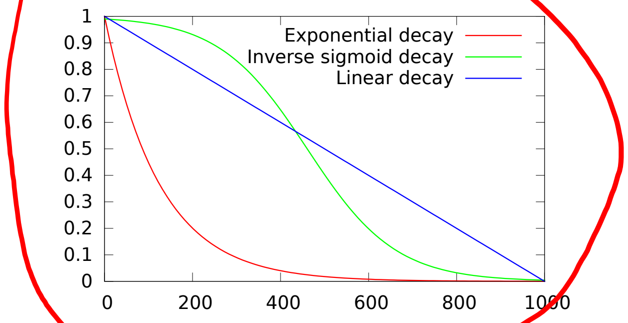


Figure 2: Examples of decay schedules.

The Scheduled Sampling (SS) algorithm is another learning technique for seq2seq models, introduced by Bengio et al. (2015) *Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks*.

length of decoder is identical to ground truth.

Algorithm:

1. at training time, flip a weighted coin to decide whether to feed the **model's prediction or the ground truth** from the previous time step as the decoder input for the next time step
2. at each timestep **minimize cross entropy** (or some other loss) of the **ground truth** for that time step
3. **gradually decrease the probability of** feeding in the **ground truth** with each iteration of training

Comments:

- SS is just like Teacher Forcing except that with some probability we feed in the model's prediction from the previous time step.
- SS has the same motivation as DAgger: addressing the problem of exposure bias (i.e. making the states visited at training time similar to those that will be visited at test time)
- SS is not DAgger because it only relies on the ground truth sequence, not a dynamic oracle
- SS does not come with any theoretical gaurantees (DAgger does)

Figure from Bengio et al. (2015) "Scheduled Sampling…"

Optimal Completion Distillation

Optimal Completion Distillation (OCD) was introduced in Sabour et al. (2019) *Optimal Completion Distillation for Sequence Learning*.

Algorithm:

1. at training time, feed the **model's prediction** from the previous time step as the decoder input for the next time step
2. at each timestep **minimize cross entropy** (or some loss) of the ~~ground truth~~ for that time step
   dynamic oracle (expert policy, π^*)

Comments:

Optimal Completion Distillation (OCD) is a special of the DAgger meta-algorithm to speech recognition in which:

1. the loss function at each time-step is the KL Divergence between the softmax output from the model and a uniform distribution over the dynamic oracle completions
2. the dynamic oracle is a dynamic programming algorithm (similar to edit distance) for character error rate (CER)

Optimal Completion Distillation

Optimal Completion Distillation (OCD) was introduced in Sabour et al. (2019) *Optimal Completion Distillation for Sequence Learning*.

Algorithm:

1. at training time, feed the **model's prediction** from the previous time step as the decoder input for the next time step
2. at each timestep **minimize cross entropy** (or some loss) of the **ground truth** for that time step

In the original OCD paper, they **only** feed in model predictions.

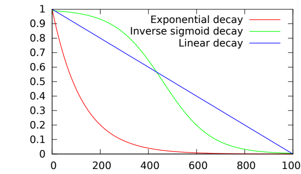
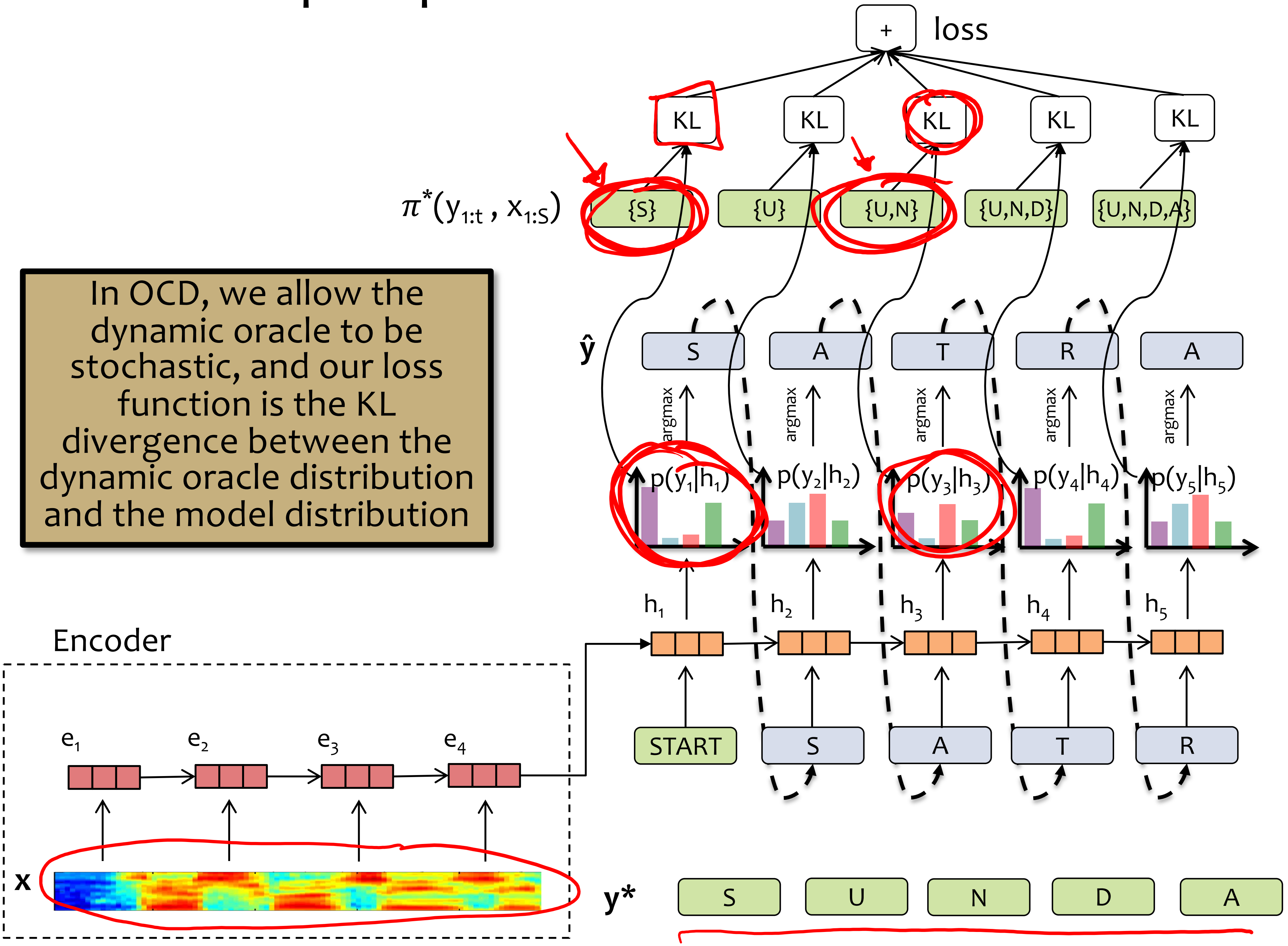


Figure 2: Examples of decay schedules.

However, because OCD is a special case of DAgger, we can generalize OCD to use a beta schedule that gives a mixture of the oracle policy with the model policy, and feed in a sample from the mixture policy to the decoder.

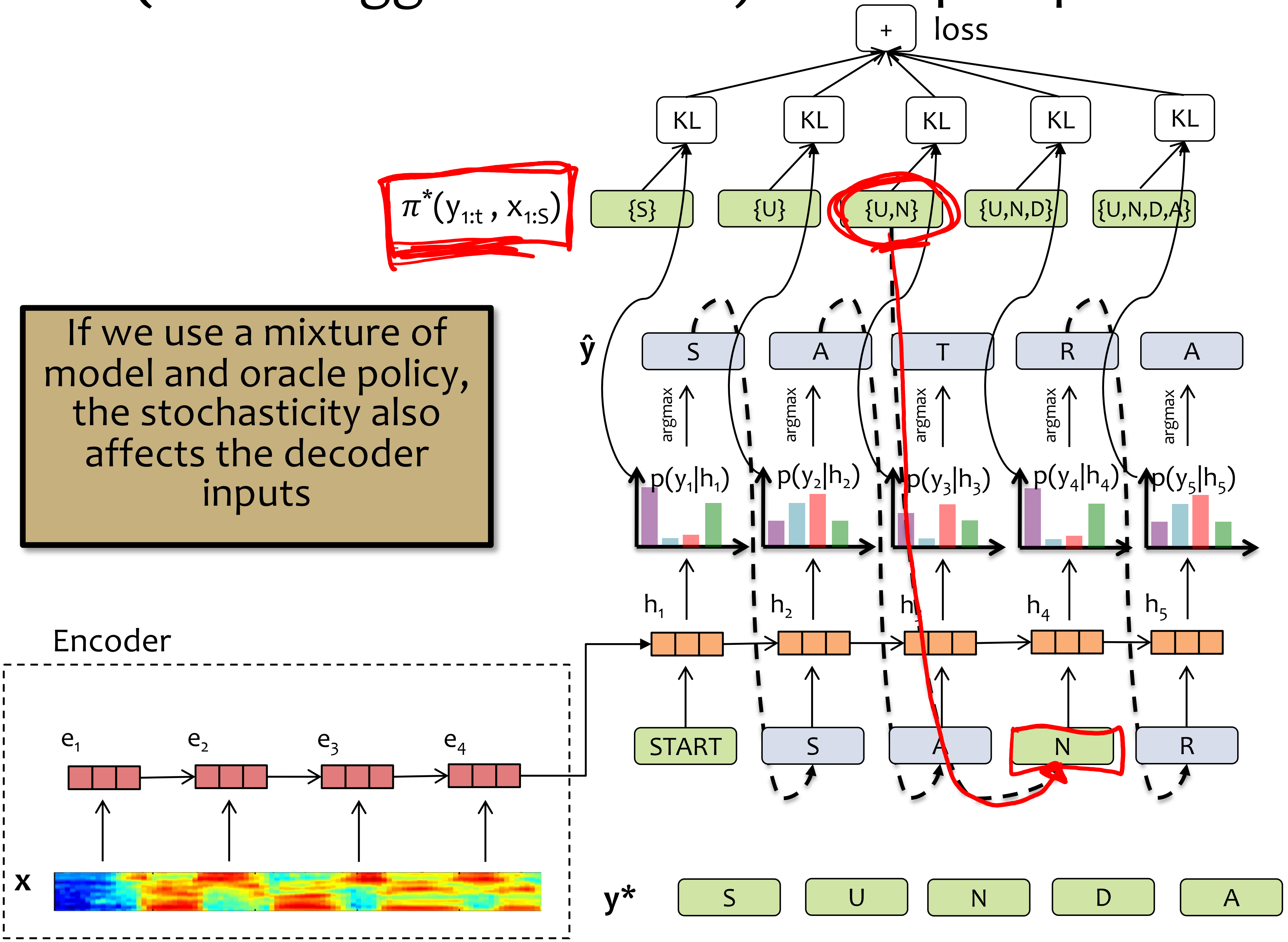
OCD for seq2seq

In OCD, we allow the dynamic oracle to be stochastic, and our loss function is the KL divergence between the dynamic oracle distribution and the model distribution



OCD (with DAgger's mixture) for seq2seq

If we use a mixture of model and oracle policy, the stochasticity also affects the decoder inputs



Imitation Learning vs. RL

Q: What is the difference between imitation learning and reinforcement learning?

A: There are lots of differences but they all stem from one fundamental difference:

Imitation learning assumes that it has access to an **oracle policy** π*, reinforcement learning does not.

Interesting contrast: Q-Learning vs. DAgger.

- both have some notion of explore/exploit (very loose analogy)
- but Q-learning's exploration is random, and its exploitation relies on the model's policy
- whereas DAgger exploration uses the model's policy, and its exploitation follows the oracle

COMPARISON OF LEARNERS

Comparison of Learners

Learning to Search Terminology

- **roll-in:** how we got to the current state
- **one-step deviations:** immediate neighbors in the search space under consideration
- **roll-out:** completions of deviations (might be optimal, might not)
- The **learner** considers each of the one-step deviations and (typically) chooses the one with **lowest loss**

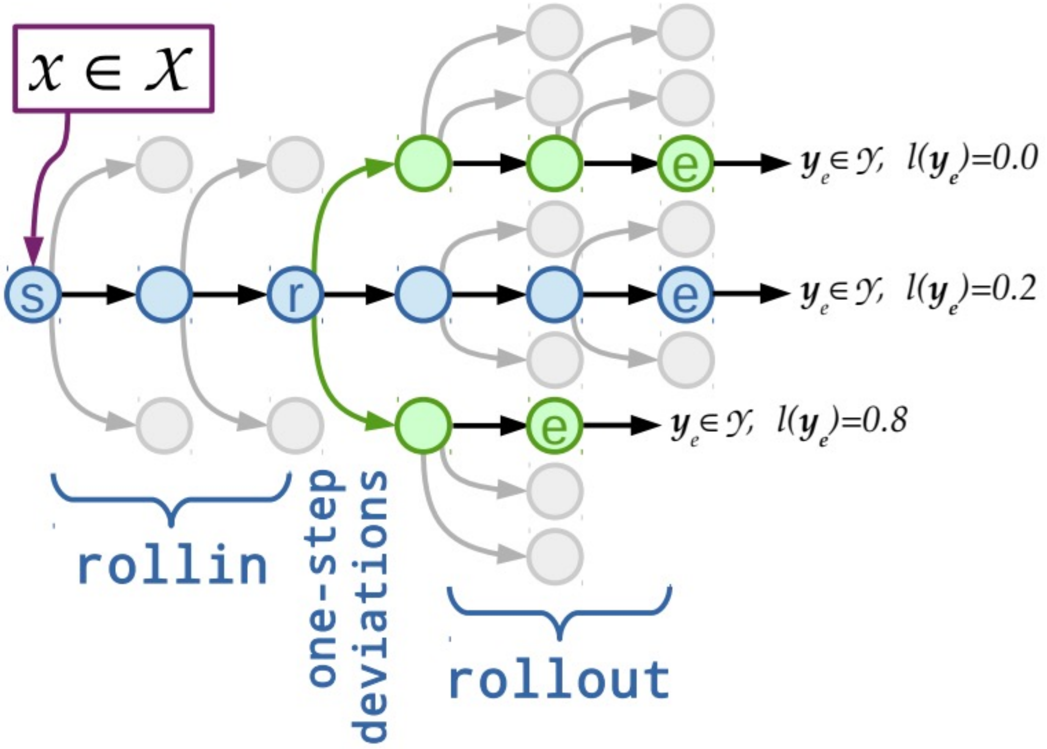
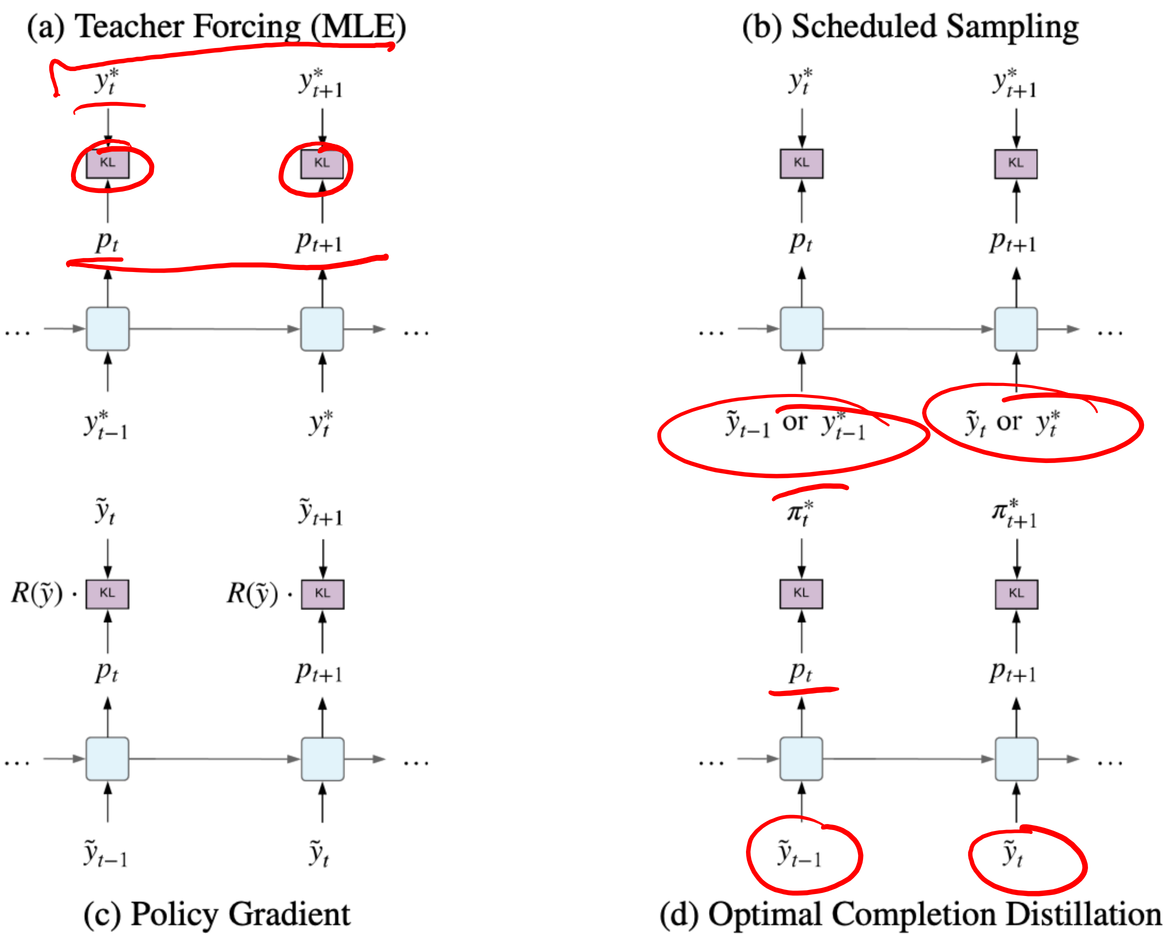

$x \in \mathcal{X}$
s
r
e
$y_e \in \mathcal{Y},\ l(y_e)=0.0$
e
$y_e \in \mathcal{Y},\ l(y_e)=0.2$
e
$y_e \in \mathcal{Y},\ l(y_e)=0.8$
rollin
one-step deviations
rollout

Figure from Chang et al. (2015)

Comparison of Learners

Learner	Roll-in	Target	Objective (per time step)
MLE	ground truth **y***	ground truth **y***	cross-entropy of ground truth
Scheduled Sampling	mixture of model predictions $\hat{\mathbf{y}}$ and ground truth **y***	ground truth **y***	cross-entropy of ground truth
Dagger	mixture of expert policy π^* and model policy $\hat{\pi}$	expert policy π^*	cross-entropy of dynamic oracle completion
OCD (special case of DAgger for CER)	model policy $\hat{\pi}$ = model predictions $\hat{\mathbf{y}}$	expert policy π^*	KL Divergence between uniform distribution over set of dynamic oracle completions and model's softmax
Policy Gradient (RL)	model policy $\hat{\pi}$ = model predictions $\hat{\mathbf{y}}$	model policy $\hat{\pi}$	increase score of sequences with high reward R($\hat{\mathbf{y}}$)

Comparison of Learners



(a) Teacher Forcing (MLE)

(b) Scheduled Sampling

(c) Policy Gradient

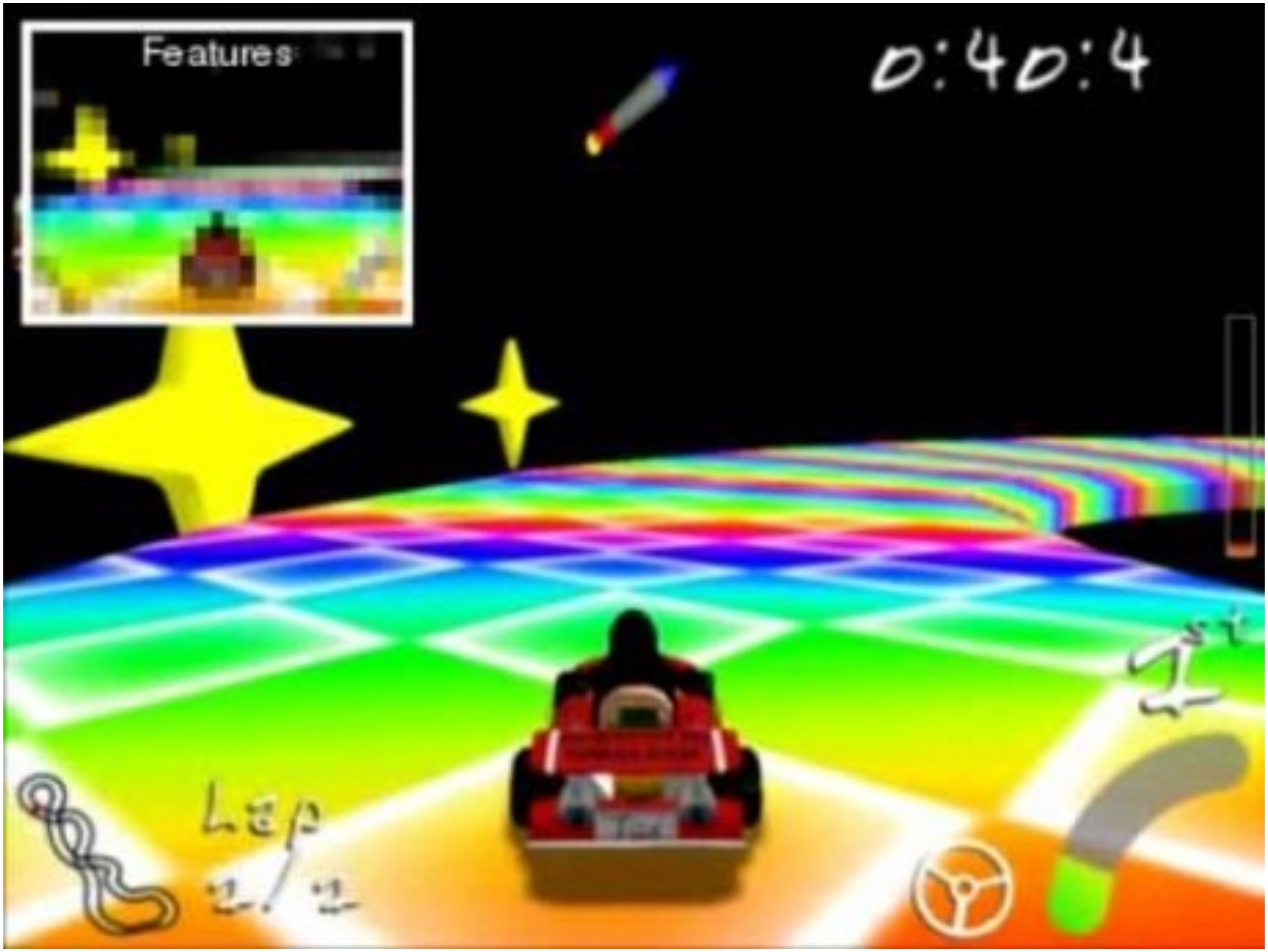
(d) Optimal Completion Distillation

Figure B.1: Illustration of different training strategies for autoregressive sequence models. (a) Teacher Forcing: the model conditions on correct prefixes and is taught to predict the next ground truth token. (b) Scheduled Sampling: the model conditions on tokens either from ground truth or drawn from the model and is taught to predict the next ground truth token regardless. (c) Policy Gradient: the model conditions on prefixes drawn from the model and is encouraged to reinforce sequences with a large sequence reward $R(\tilde{y})$. (d) Optimal Completion Distillation: the model conditions on prefixes drawn from the model and is taught to predict an optimal completion policy π^* specific to the prefix.

Figure from Sabour et al. (2019)

LEARNING TO SEARCH: EMPIRICAL RESULTS

Dagger for Mario Tux Cart



Video from Stéphane Ross (https://www.youtube.com/watch?v=V00npNnWzSU)

Experiments: Vowpal Wabbit L2S

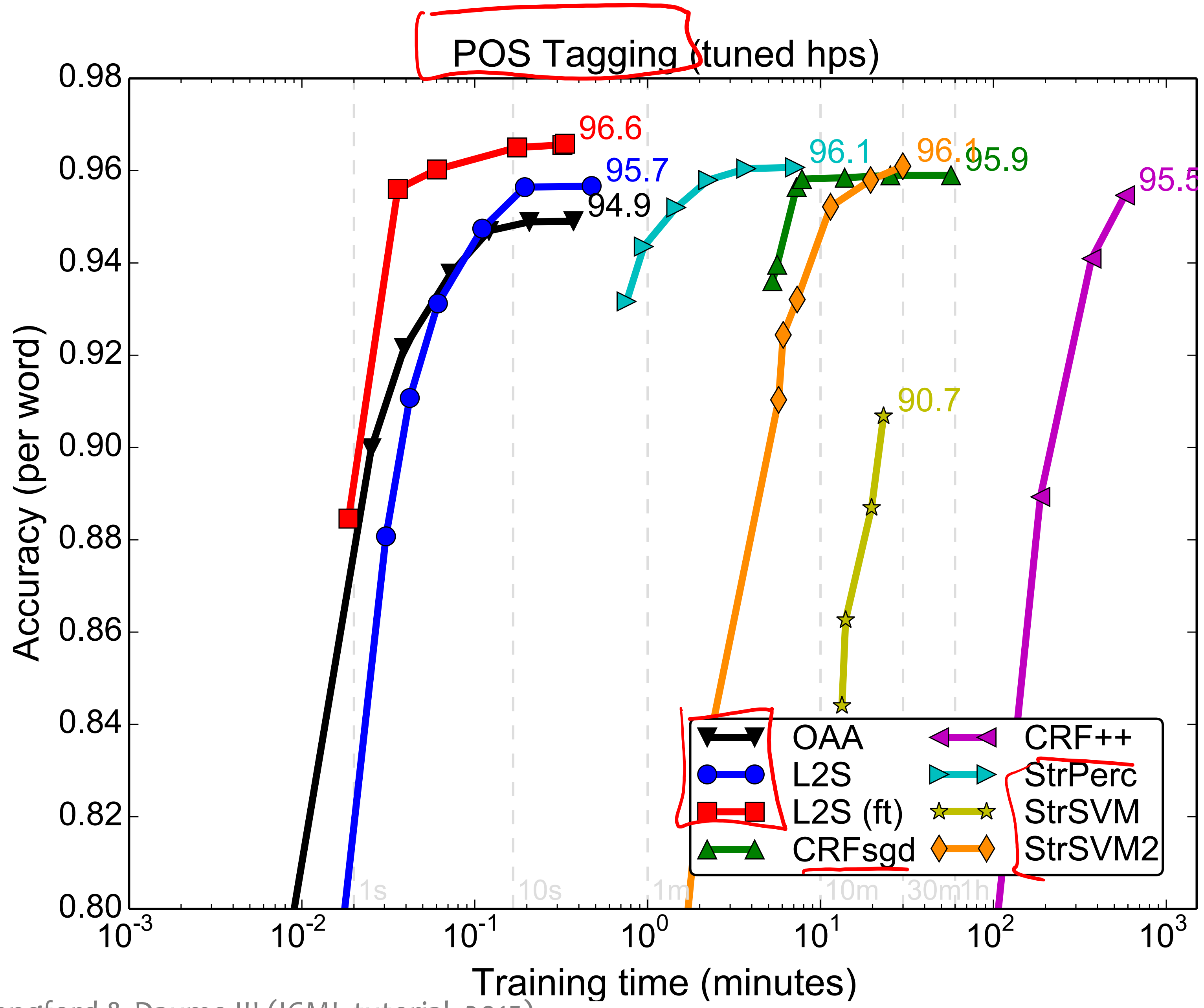


Figure from Langford & Daume III (ICML tutorial, 2015)

Experiments: Vowpal Wabbit L2S

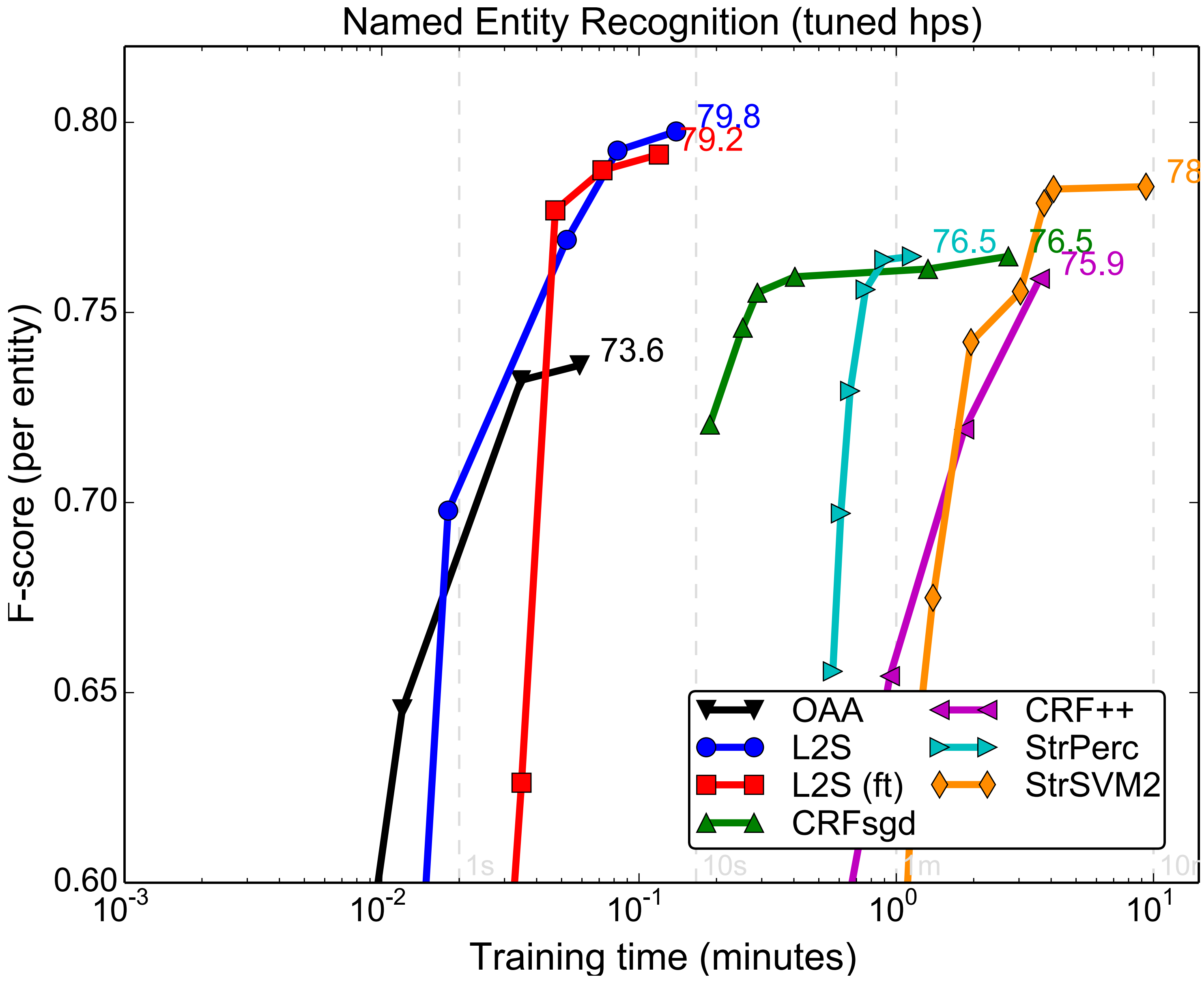


Figure from Langford & Daume III (ICML tutorial, 2015)

Experiments: Vowpal Wabbit L2S

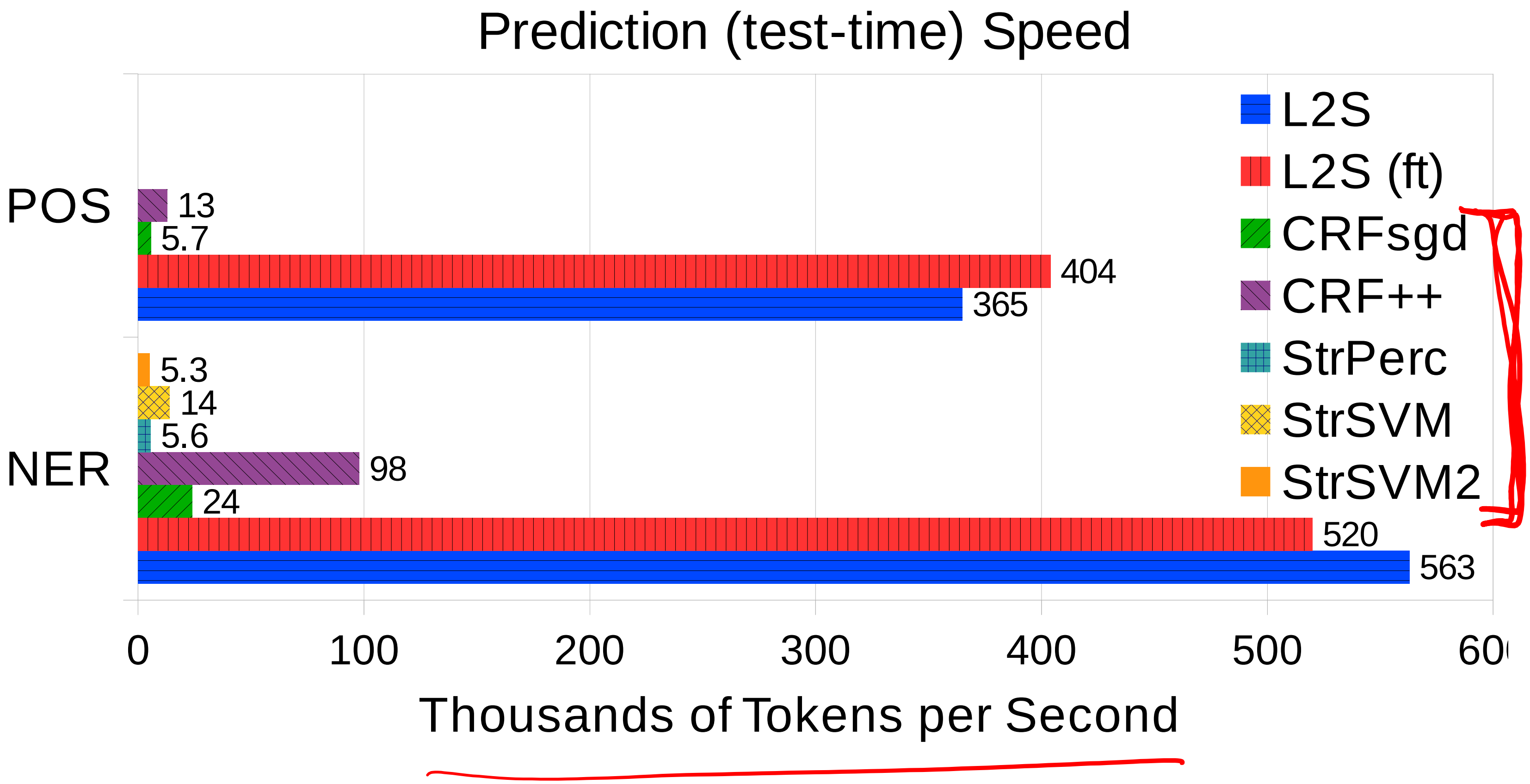


Figure from Langford & Daume III (ICML tutorial, 2015)

Results: Dynamic Oracle for Dependency Parsing

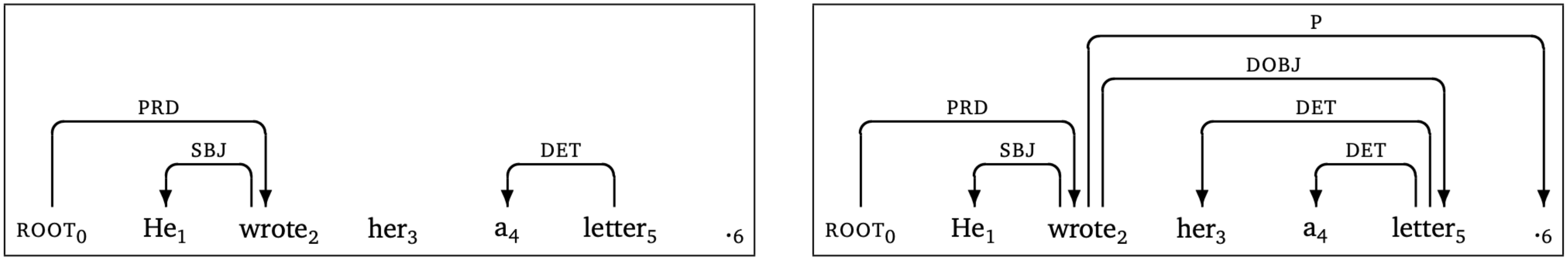


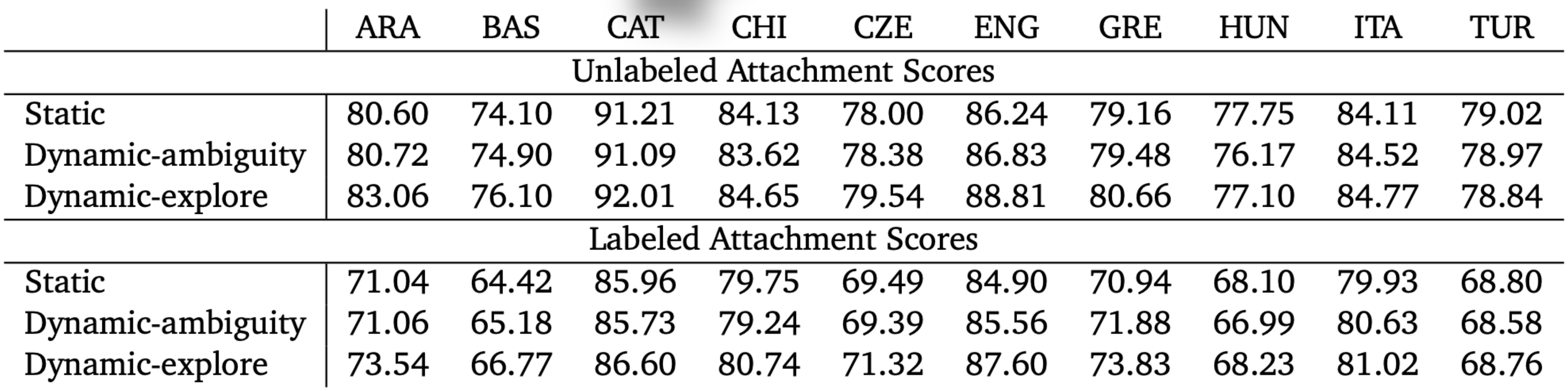
Figure 3: Dependency graphs with loss 3 (left) and loss 1 (right)

	ARA	BAS	CAT	CHI	CZE	ENG	GRE	HUN	ITA	TUR
				Unlabeled Attachment Scores						
Static	80.60	74.10	91.21	84.13	78.00	86.24	79.16	77.75	84.11	79.02
Dynamic-ambiguity	80.72	74.90	91.09	83.62	78.38	86.83	79.48	76.17	84.52	78.97
Dynamic-explore	83.06	76.10	92.01	84.65	79.54	88.81	80.66	77.10	84.77	78.84
				Labeled Attachment Scores						
Static	71.04	64.42	85.96	79.75	69.49	84.90	70.94	68.10	79.93	68.80
Dynamic-ambiguity	71.06	65.18	85.73	79.24	69.39	85.56	71.88	66.99	80.63	68.58
Dynamic-explore	73.54	66.77	86.60	80.74	71.32	87.60	73.83	68.23	81.02	68.76

Table 2: Results on the CoNLL 2007 data sets

Results: Dynamic Oracle for Dependency Parsing

Training with a dynamic oracle (Dynamic-ambiguity & Dynamic-explore) consistently outperforms training to produce the ground truth (Static), across many different languages.



	ARA	BAS	CAT	CHI	CZE	ENG	GRE	HUN	ITA	TUR
					Unlabeled Attachment Scores					
Static	80.60	74.10	91.21	84.13	78.00	86.24	79.16	77.75	84.11	79.02
Dynamic-ambiguity	80.72	74.90	91.09	83.62	78.38	86.83	79.48	76.17	84.52	78.97
Dynamic-explore	83.06	76.10	92.01	84.65	79.54	88.81	80.66	77.10	84.77	78.84
					Labeled Attachment Scores					
Static	71.04	64.42	85.96	79.75	69.49	84.90	70.94	68.10	79.93	68.80
Dynamic-ambiguity	71.06	65.18	85.73	79.24	69.39	85.56	71.88	66.99	80.63	68.58
Dynamic-explore	73.54	66.77	86.60	80.74	71.32	87.60	73.83	68.23	81.02	68.76

Table 2: Results on the CoNLL 2007 data sets

DAGGER: THEORETICAL RESULTS

DAgger Policy During Training

- DAgger assumes that we follow a **stochastic policy** that flips a weighted coin (with weight β_i at timestep i) to decide between the oracle policy and the model's policy

$$\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$$

- We require that (β_1, β_2, β_3, …) is chosen to be a sequence such that:

$$\overline{\beta}_N = \underline{\tfrac{1}{N} \textstyle\sum_{i=1}^{N} \beta_i \rightarrow 0} \quad \text{as} \quad N \rightarrow \infty.$$

Q: What are examples of such sequences?

DAgger Theoretical Results

- The theory mirrors the intuition that **Exposure Bias is bad**
- The Supervised Approach to Imitation performs **not-so-well** even on the oracle (training time) distribution over states (i.e. ~~quadratically~~ number of mistakes grows **quadratically** in task horizon T and classification cost ϵ)
- DAgger yields an algorithm that performs **well** on the test-time distribution over states (i.e. number of mistakes grows **linearly** in task horizon T and classification cost ϵ)

$$J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s\sim d_\pi^t}\left[C_\pi(s)\right]$$

Algo #1: Supervised Approach to Imitation

Theorem 2.1. *(Ross and Bagnell, 2010) Let $\mathbb{E}_{s\sim d_{\pi^*}}[\ell(s,\pi)] = \epsilon$, then $J(\pi) \leq J(\pi^*) + T^2\epsilon$.*

Algo #2: DAgger

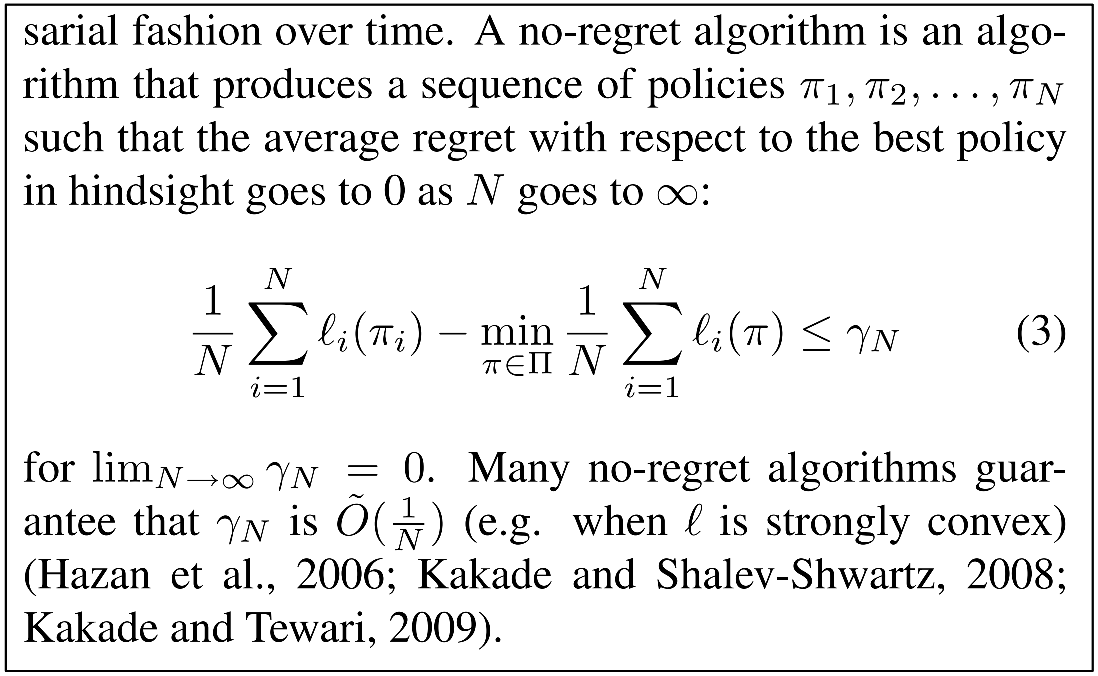
Theorem 3.2. *For* DAGGER, *if N is $\tilde{O}(uT)$ there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ s.t. $J(\hat{\pi}) \leq J(\pi^*) + uT\epsilon_N + O(1)$.*

$$\epsilon_N = \min_{\pi\in\Pi} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{s\sim d_{\pi_i}}\left[\ell(s,\pi)\right]$$

DAgger Theoretical Results

- The proof of the results for DAgger relies on a reduction to no-regret online learning

From Ross et al. (2011) “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning”...

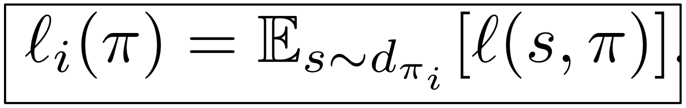


sarial fashion over time. A no-regret algorithm is an algorithm that produces a sequence of policies $\pi_1, \pi_2, \ldots, \pi_N$ such that the average regret with respect to the best policy in hindsight goes to 0 as N goes to ∞:

$$\frac{1}{N}\sum_{i=1}^{N} \ell_i(\pi_i) - \min_{\pi\in\Pi} \frac{1}{N}\sum_{i=1}^{N} \ell_i(\pi) \leq \gamma_N \tag{3}$$

for $\lim_{N\to\infty} \gamma_N = 0$. Many no-regret algorithms guarantee that γ_N is $\tilde{O}(\frac{1}{N})$ (e.g. when ℓ is strongly convex) (Hazan et al., 2006; Kakade and Shalev-Shwartz, 2008; Kakade and Tewari, 2009).

- The key idea is to choose the loss function to be that of the loss on the distribution over states given by the current policy chosen by the online learner



$$\ell_i(\pi) = \mathbb{E}_{s\sim d_{\pi_i}}\left[\ell(s,\pi)\right].$$

Learning 2 Search

Some key challenges:

- performance depends heavily on search order, but have to pick this by hand
- expert policy is critical, but what if it's too difficult to design one
- not always easy to make efficient on a GPU

Adapted from Langford & Daume III (ICML tutorial, 2015)

Learning Objectives

Structured Prediction as Search

You should be able to…

1. Reduce a structured prediction problem to a search problem
2. Implement Dagger, a learning to search algorithm
3. (If you already know RL…) Contrast imitation learning with reinforcement learning
4. Explain the reduction of structured prediction to no-regret online learning
5. Contrast various learning2search algorithms based on their properties