

CS11-711 Advanced NLP

Margin-based and Reinforcement Learning for Structured Prediction

Daniel Fried



Carnegie Mellon University

Language Technologies Institute

Site

<https://cmu-anlp.github.io/>

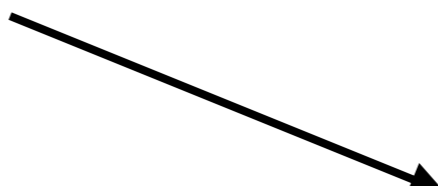
(w/ slides from Graham Neubig)

Types of Prediction


- Two classes (**binary classification**)

I hate this movie  positive
negative

- Multiple classes (**multi-class classification**)

I hate this movie  very good
good
neutral
bad
very bad

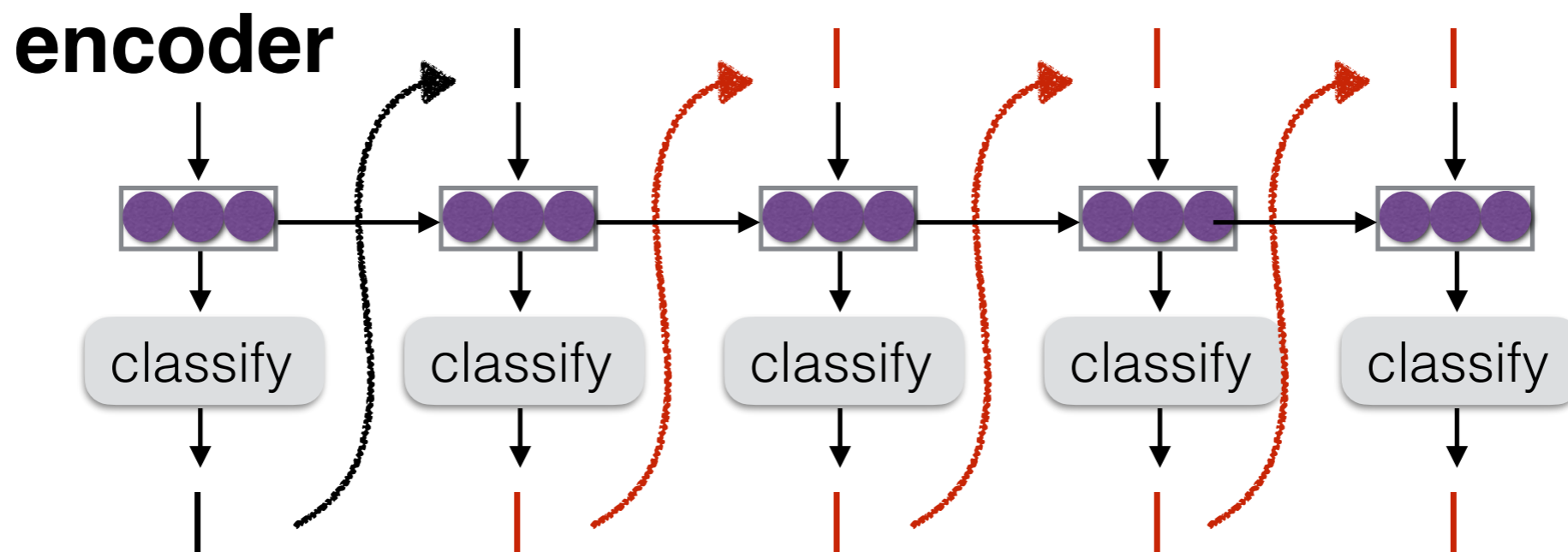
- Exponential/infinite labels (**structured prediction**)

I hate this movie  PRP VBP DT NN

I hate this movie  *kono eiga ga kirai*

Problem 1: Exposure Bias

- *Teacher forcing* assumes feeding correct previous input, but at test time we may make mistakes that propagate



- **Exposure bias:** The model is not exposed to mistakes during training, and cannot deal with them at test

Problem 2: Disregard to Evaluation Metrics

- In the end, we want good outputs
- Good translations can be measured with metrics, e.g. BLEU or METEOR
- Some mistaken predictions hurt more than others, so we'd like to penalize them appropriately

Many Varieties of Structured Prediction!

- **Models:**

- RNN-based decoders
- Self attentional decoders

Covered
already

- **Training algorithms:**

- Maximum likelihood w/ teacher forcing

- Sequence level likelihood
- Structured perceptron, structured large margin
- Reinforcement learning/minimum risk training
- Sampling corruptions of data

Covered
today

Globally Normalized Models

- **Locally normalized models:** each decision made by the model has a probability that adds to one

$$P(Y | X) = \prod_{j=1}^{|Y|} \frac{e^{S(y_j | X, y_1, \dots, y_{j-1})}}{\sum_{\tilde{y}_j \in V} e^{S(\tilde{y}_j | X, y_1, \dots, y_{j-1})}}$$

- **Globally normalized models (a.k.a. energy-based models):** each sentence has a score, which is not normalized over a particular decision

$$P(Y | X) = \frac{e^{S(X, Y)}}{\sum_{\tilde{Y} \in V^*} e^{S(X, \tilde{Y})}}$$

Globally Normalized Models

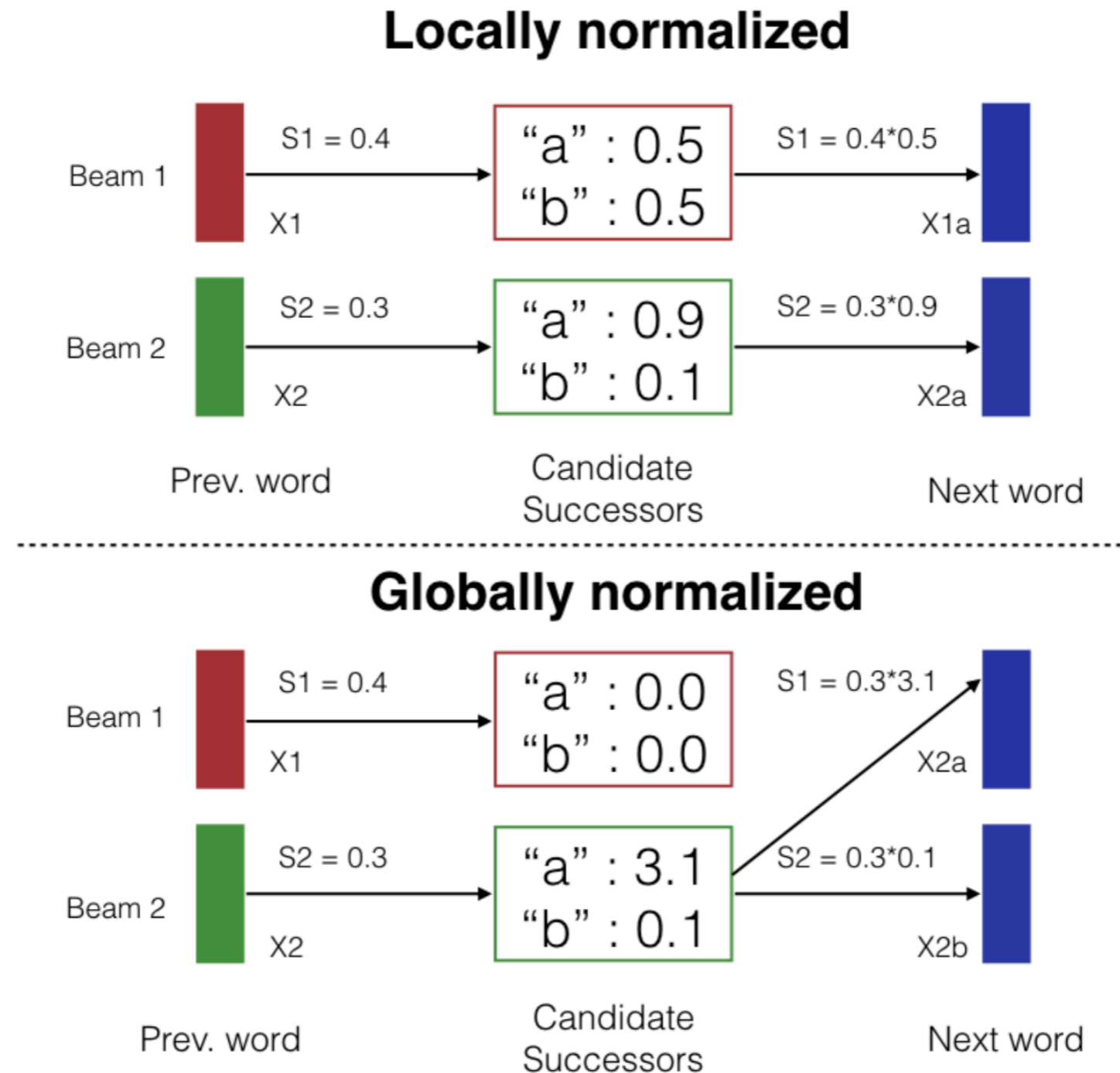


Figure from Goyal et al. 2019

Difficulties Training Globally Normalized Models

- Partition function problematic

$$P(Y | X) = \frac{e^{S(X, Y)}}{\sum_{\tilde{Y} \in V^*} e^{S(X, \tilde{Y})}}$$

- Two options for calculating partition function
 - Structure model to allow enumeration via dynamic programming, e.g. linear chain CRF, CFG
 - Estimate partition function through **sub-sampling** hypothesis space

Two Methods for Approximation

- **Sampling:**
 - Sample k samples according to the probability distribution
 - *+ Unbiased estimator:* as k gets large will approach true distribution
 - *- High variance:* what if we get low-probability samples?
- **Beam search:**
 - Search for k best hypotheses
 - *- Biased estimator:* may result in systematic differences from true distribution
 - *+ Lower variance:* more likely to get high-probability outputs

Un-normalized Models: Structured Perceptron

Normalization often Not Necessary for Inference!

- At inference time, we often just want to (approximately) find the **best hypothesis**

$$\hat{Y} = \operatorname{argmax}_Y P(Y | X)$$

- If that's all we need, no need for normalization!

$$P(Y | X) = \frac{e^{S(X, Y)}}{\sum_{\tilde{Y} \in V^*} e^{S(X, \tilde{Y})}} \quad \hat{Y} = \operatorname{argmax}_Y S(X, Y)$$

The Structured Perceptron Algorithm

- An extremely simple way of training (non-probabilistic) global models
- Approximately find the one-best, and if its score is better than the correct answer, adjust parameters to fix this

$$\hat{Y} = \operatorname{argmax}_{\tilde{Y} \neq Y} S(\tilde{Y} | X; \theta) \quad \leftarrow \text{Search for the best}$$

if $S(\hat{Y} | X; \theta) \geq S(Y | X; \theta)$ **then** \leftarrow If score better than reference

$$\theta \leftarrow \theta + \alpha \left(\frac{\partial S(Y | X; \theta)}{\partial \theta} - \frac{\partial S(\hat{Y} | X; \theta)}{\partial \theta} \right) \quad \leftarrow \text{Increase score of ref, decrease score of one-best (here, SGD update)}$$

end if

Structured Perceptron Loss

- Structured perceptron can also be expressed as a loss function!

$$\ell_{\text{percept}}(X, Y) = \max(0, S(\hat{Y} | X; \theta) - S(Y | X; \theta))$$

- Resulting **gradient looks like perceptron algorithm**

$$\frac{\partial \ell_{\text{percept}}(X, Y; \theta)}{\partial \theta} = \begin{cases} \frac{\partial S(Y|X; \theta)}{\partial \theta} - \frac{\partial S(\hat{Y}|X; \theta)}{\partial \theta} & \text{if } S(\hat{Y} | X; \theta) \geq S(Y | X; \theta) \\ 0 & \text{otherwise} \end{cases}$$

- This is a normal loss function, **can be used in NNs**
- But! Requires finding the argmax in addition to the true candidate: must **do prediction during training**

Contrasting Perceptron and Global Normalization

- **Globally normalized probabilistic model**

$$\ell_{\text{global}}(X, Y; \theta) = -\log \frac{e^{S(Y|X)}}{\sum_{\tilde{Y}} e^{S(\tilde{Y}|X)}}$$

- **Structured perceptron**

$$\ell_{\text{percept}}(X, Y) = \max(0, S(\hat{Y} | X; \theta) - S(Y | X; \theta))$$

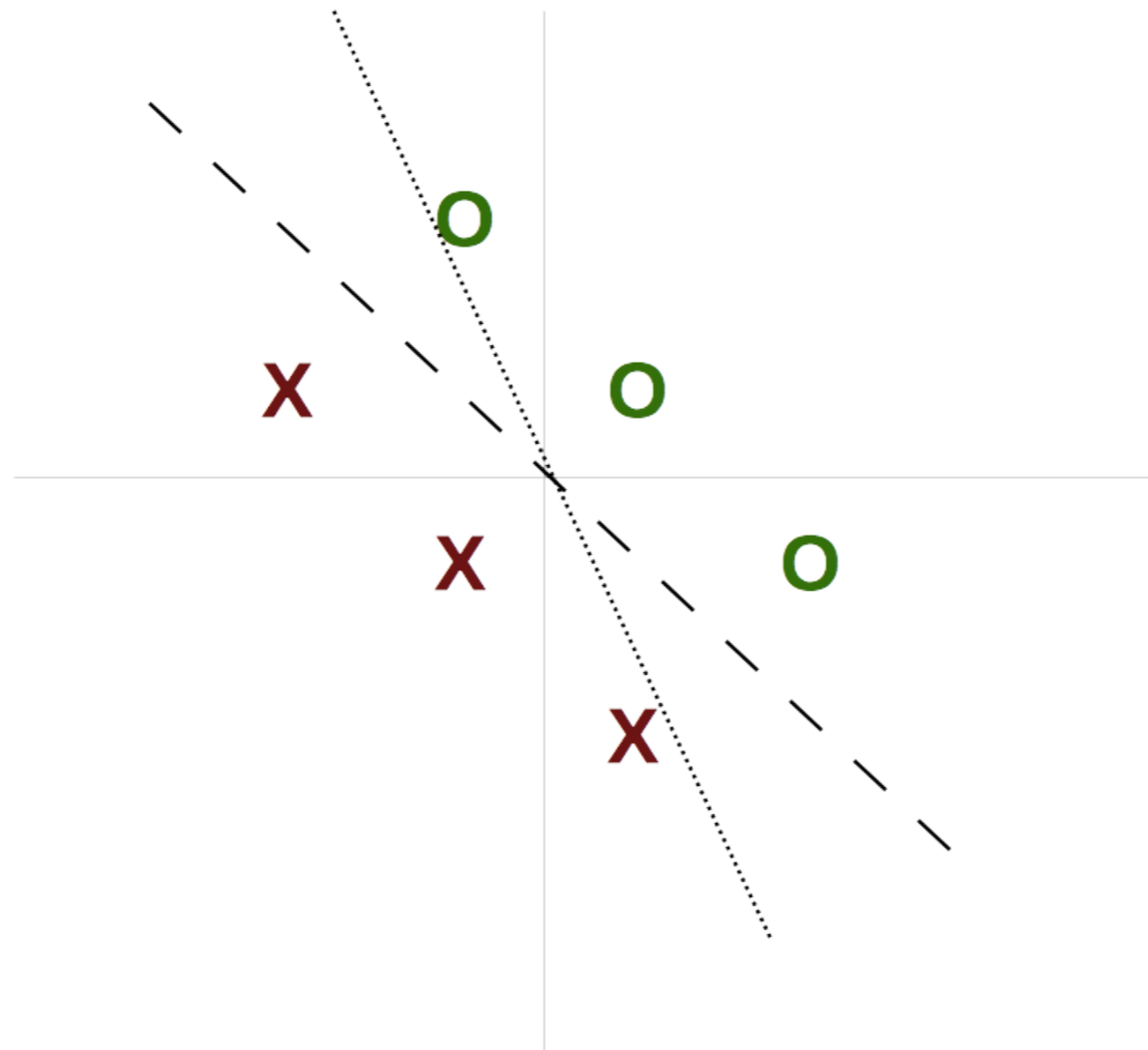
Structured Training and Pre-training

- Neural network models have lots of parameters and a big output space; **training is hard**
- **Tradeoffs** between training algorithms:
 - Selecting just one negative example is inefficient
 - Teacher forcing efficiently updates all parameters, but suffers from exposure bias
- Thus, it is common to **pre-train with teacher forcing, then fine-tune with more complicated algorithm**

Hinge Loss and Cost-sensitive Training

Perceptron and Uncertainty

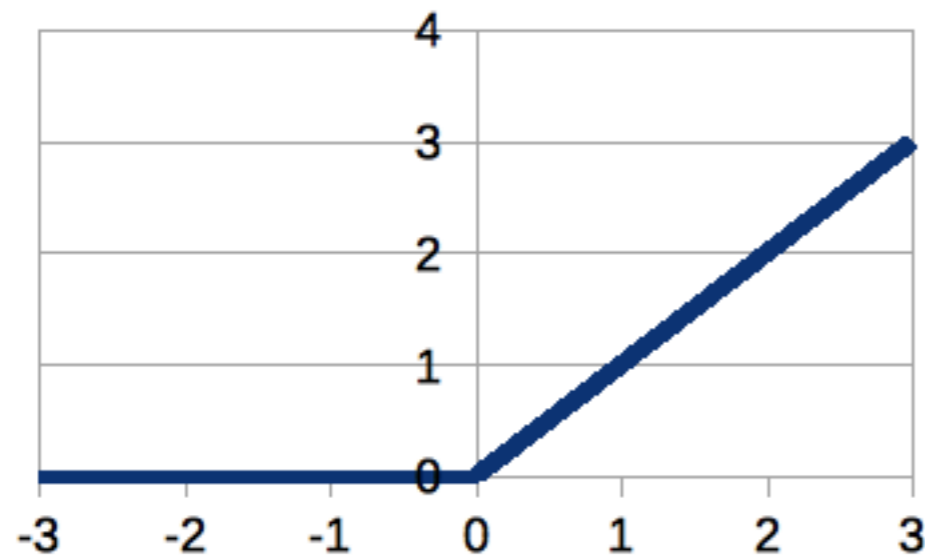
- Which is better, dotted or dashed?



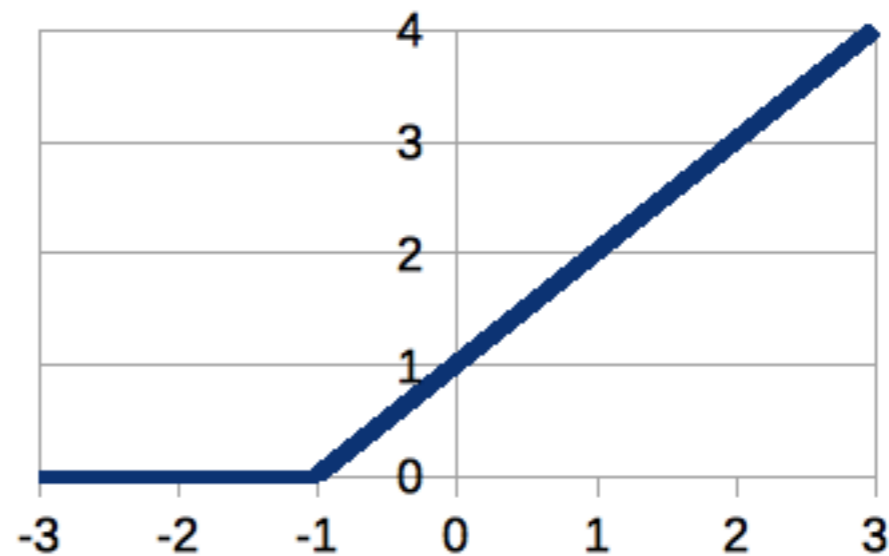
- Both have zero perceptron loss!

Adding a “Margin” with Hinge Loss

- Penalize when incorrect answer is within margin m .
i.e. the reference has to be better than the incorrect by at least m .



Perceptron

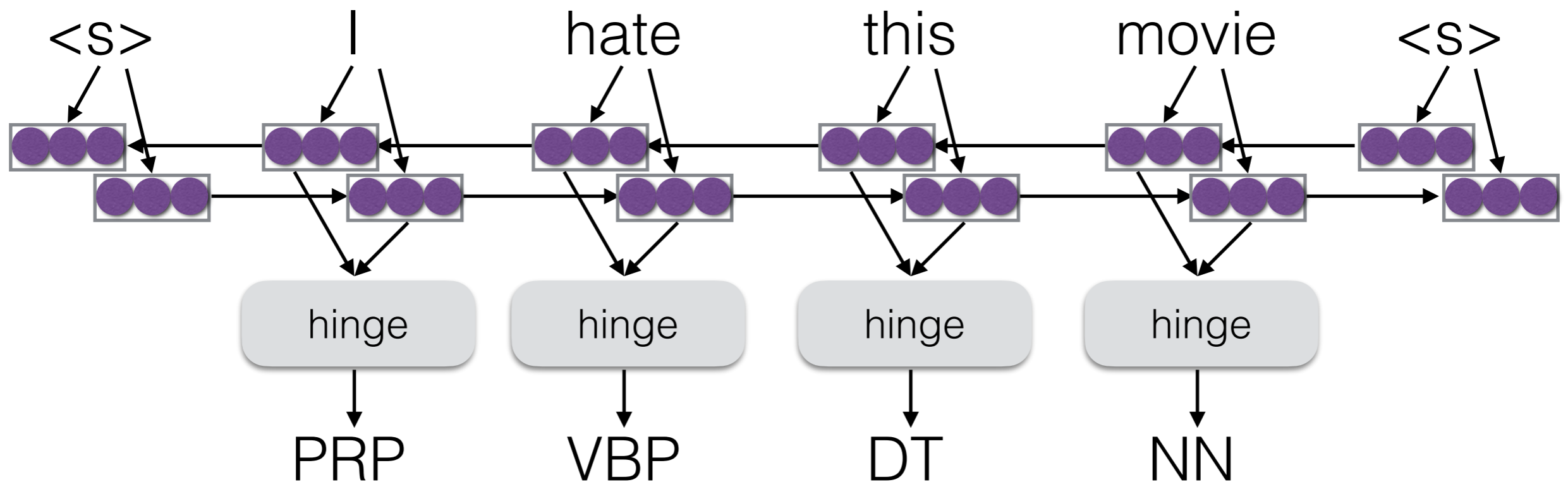


Hinge

$$\ell_{\text{hinge}}(x, y; \theta) = \max(0, m + S(\hat{y} | x; \theta) - S(y | x; \theta))$$

Hinge Loss for Any Classifier!

- We can swap cross-entropy for hinge loss anytime



e.g.

```
loss = nn.CrossEntropyLoss()
```

↓

in
PyTorch

```
loss = nn.MultiMarginLoss(margin=1.0)
```

Cost-augmented Hinge

- Sometimes some decisions are worse than others
 - e.g. VB -> VBP mistake not so bad, VB -> NN mistake much worse for downstream apps
- Cost-augmented hinge defines a cost for each incorrect decision, and sets margin equal to this

$$\ell_{\text{ca-hinge}}(x, y; \theta) = \max(0, \text{cost}(\hat{y}, y) + S(\hat{y} | x; \theta) - S(y | x; \theta))$$

Costs over Sequences

- **Zero-one loss:** 1 if sentences differ, zero otherwise

$$\text{cost}_{\text{zero-one}}(\hat{Y}, Y) = \delta(\hat{Y} \neq Y)$$

- **Hamming loss:** 1 for every different element (lengths are identical)

$$\text{cost}_{\text{hamming}}(\hat{Y}, Y) = \sum_{j=1}^{|Y|} \delta(\hat{y}_j \neq y_j)$$

- **Other losses:** edit distance, 1-BLEU, etc.

Structured Hinge Loss

- Hinge loss over sequence with the largest margin violation

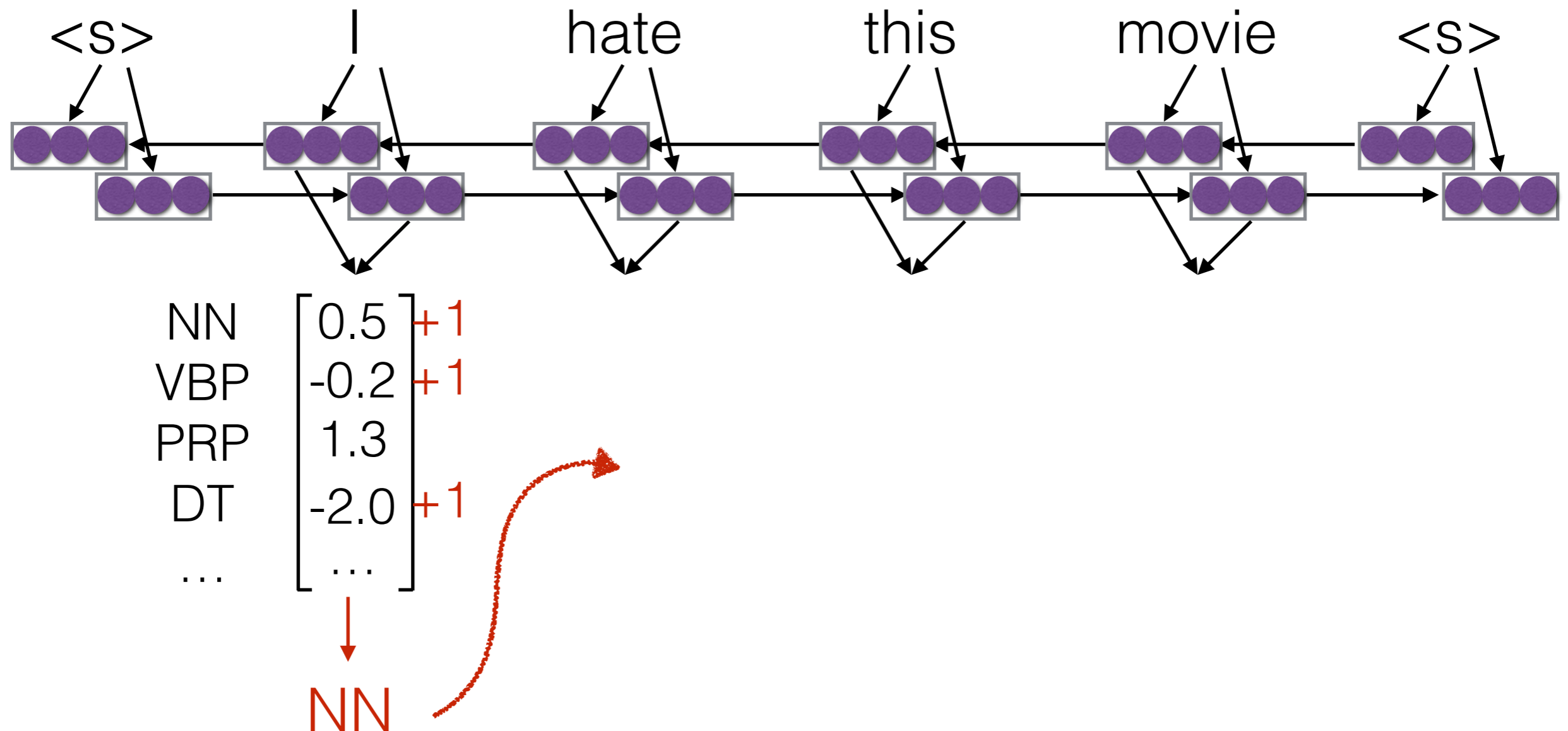
$$\hat{Y} = \operatorname{argmax}_{\tilde{Y} \neq Y} \operatorname{cost}(\tilde{Y}, Y) + S(\tilde{Y} | X; \theta)$$

$$\ell_{\text{ca-hinge}}(X, Y; \theta) = \max(0, \operatorname{cost}(\hat{Y}, Y) + S(\hat{Y} | X; \theta) - S(Y | X; \theta))$$

- **Problem:** How do we find the argmax above?
- **Solution:** In some cases, where the loss can be calculated easily, we can consider loss in search.

Cost-Augmented Decoding for Hamming Loss

- Hamming loss is decomposable over each word
- **Solution:** add a score = cost to each incorrect choice during search



Objective Zoo

- Edunov and Ott et al. 2018 have a nice overview of many seq-level losses, and evaluations on machine translation

$$\mathcal{L}_{\text{TokNLL}} = - \sum_{i=1}^n \log p(t_i | t_1, \dots, t_{i-1}, \mathbf{x})$$

$$\mathcal{L}_{\text{TokLS}} = - \sum_{i=1}^n \log p(t_i | t_1, \dots, t_{i-1}, \mathbf{x}) - D_{KL}(f || p(t_i | t_1, \dots, t_{i-1}, \mathbf{x}))$$

$$\mathcal{L}_{\text{SeqNLL}} = - \log p(\mathbf{u}^* | \mathbf{x}) + \log \sum_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} p(\mathbf{u} | \mathbf{x})$$

$$\mathcal{L}_{\text{Risk}} = \sum_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} \text{cost}(\mathbf{t}, \mathbf{u}) \frac{p(\mathbf{u} | \mathbf{x})}{\sum_{\mathbf{u}' \in \mathcal{U}(\mathbf{x})} p(\mathbf{u}' | \mathbf{x})}$$

Reinforcement
learning
(coming soon)

$$\mathcal{L}_{\text{MaxMargin}} = \max [0, \text{cost}(\mathbf{t}, \hat{\mathbf{u}}) - \text{cost}(\mathbf{t}, \mathbf{u}^*) - s(\mathbf{u}^* | \mathbf{x}) + s(\hat{\mathbf{u}} | \mathbf{x})]$$

Objective Zoo

- Edunov and Ott et al. 2018 have a nice overview of many seq-level losses, and evaluations on machine translation

TokNLL [T]	31.78	0.07
TokLS [T]	32.23	0.10
<hr/>		
SeqNLL [S]	32.68	0.09
Risk [S]	32.84	0.08
MaxMargin [S]	32.55	0.09
MultiMargin [S]	32.59	0.07
SoftmaxMargin [S]	32.71	0.07
<hr/>		

Table 1: Test accuracy in terms of BLEU on IWSLT'14 German-English translation with various loss functions cf. Figure 1. W & R (2016) refers to [Wiseman and](#)

Reinforcement Learning Basics:

Policy Gradient

(Review of Karpathy 2016)

What is Reinforcement Learning?

- Learning where we have an
 - environment X
 - ability to make actions A
 - get a delayed reward R
- **Example of pong:** X is our observed image, A is up or down, and R is the win/loss at the end of the game

Why Reinforcement Learning in NLP?

- We may have a **typical reinforcement learning scenario**: e.g. a dialog where we can make responses and will get a reward at the end. **This need not use a reference!**
- We may have **latent variables**, where we decide the latent variable, then get a reward based on their configuration.
- We may have a **sequence-level error function** such as BLEU score that we cannot optimize without first generating a whole sentence.

Supervised MLE

- We are given the correct decisions

$$\ell_{\text{super}}(Y, X) = -\log P(Y | X)$$

- In the context of reinforcement learning, this is also called “imitation learning,” imitating a teacher (although imitation learning is more general)

Self Training

- Sample or argmax according to the current model

$$\hat{Y} \sim P(Y | X) \quad \text{or} \quad \hat{Y} = \operatorname{argmax}_Y P(Y | X)$$

- Use this sample (or samples) to maximize likelihood

$$\ell_{\text{self}}(X) = -\log P(\hat{Y} | X)$$

- No correct answer needed! But is this a good idea?
- *One successful alternative:* co-training, only use sentences where multiple models agree (Blum and Mitchell 1998)
- *Another successful alternative:* noising the input, to smooth the data distribution (He et al. 2020)

Policy Gradient/REINFORCE

- Sample outputs \hat{Y} from the current model P
- Add a term that scales the loss by the reward

$$\ell_{\text{self}}(X) = -R(\hat{Y}, Y) \log P(\hat{Y} | X)$$

- Outputs that get a bigger reward will get a higher weight
- Quiz: Under what conditions is this equal to MLE?

Credit Assignment for Rewards

- How do we know which action led to the reward?
- Best scenario, immediate reward:

a_1	a_2	a_3	a_4	a_5	a_6
0	+1	0	-0.5	+1	+1.5

- Worst scenario, only at end of roll-out:

a_1	a_2	a_3	a_4	a_5	a_6
					+3

- Often assign decaying rewards for future events to take into account the time delay between action and reward

Problems w/ Reinforcement Learning

- Like other sampling-based methods, reinforcement learning is unstable
- It is particularly unstable when using bigger output spaces (e.g. words of a vocabulary)
- Can lead to language drift / reward over-optimization (Gao et al. 2023)
- A number of strategies can be used to stabilize

Stabilizing RL

- Add a value-estimation baseline (Dayan 1990; Ranzato et al. 2016)
- Increase the batch size / replay data
- Warm-start the model with maximum likelihood
- Use KL-regularization to ensure the model doesn't "drift" from language
- Use a more sophisticated approach, like Proximal Policy Optimization (Schulman et al. 2017), Quark (Lu et al. 2022), or DPO (Rafailov et al. 2023)

RL for LLMs

RL from Human Feedback


- Use human quality judgements as the reward. Reference-free training
- **Problem 1:** human-in-the-loop is expensive
- **Solution:** model human preferences as a separate NLP problem. Train an LM to predict reward R from an annotated dataset. Then use R in RL


SAN FRANCISCO,
California (CNN) --
A magnitude 4.2
earthquake shook the
San Francisco

...
overturn unstable
objects.

An earthquake hit
San Francisco.
There was minor
property damage,
but no injuries.

The Bay Area has
good weather but is
prone to
earthquakes and
wildfires.

$$R(s_1) = 8.0$$


$$R(s_2) = 1.2$$


RL from Human Feedback

- **Problem 2:** human judgements are noisy and miscalibrated!
- **Solution:** instead of direct ratings, ask for **pairwise comparisons**, which can be more reliable (Phelps et al. 2015; Clark et al. 2018)

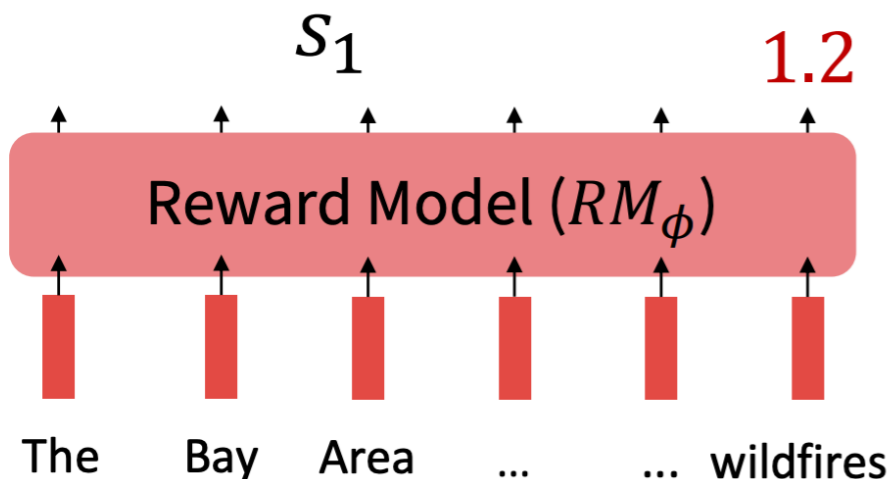
An earthquake hit San Francisco. There was minor property damage, but no injuries.

>

A 4.2 magnitude earthquake hit San Francisco, resulting in massive damage.

>

The Bay Area has good weather but is prone to earthquakes and wildfires.

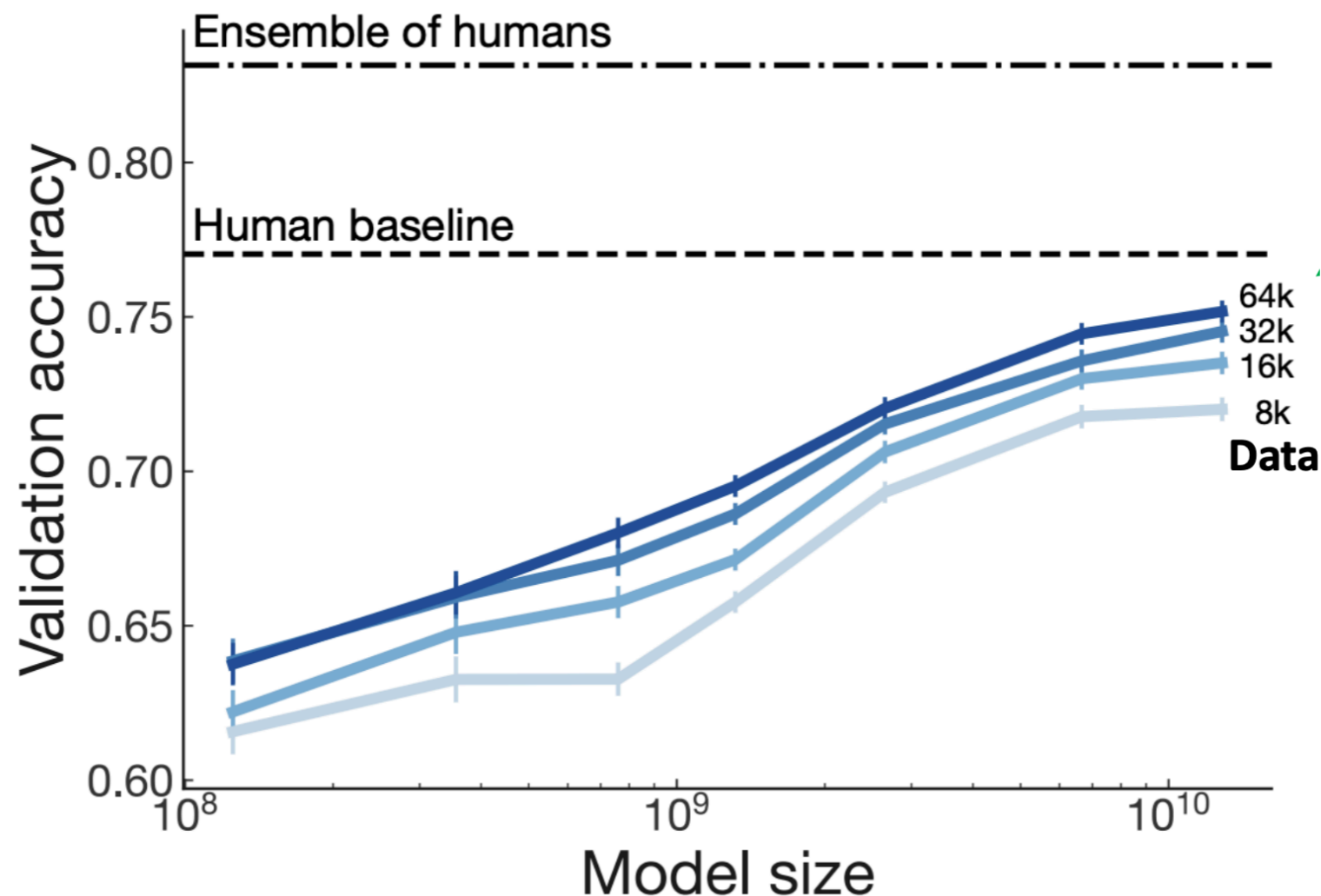


S_3

S_2

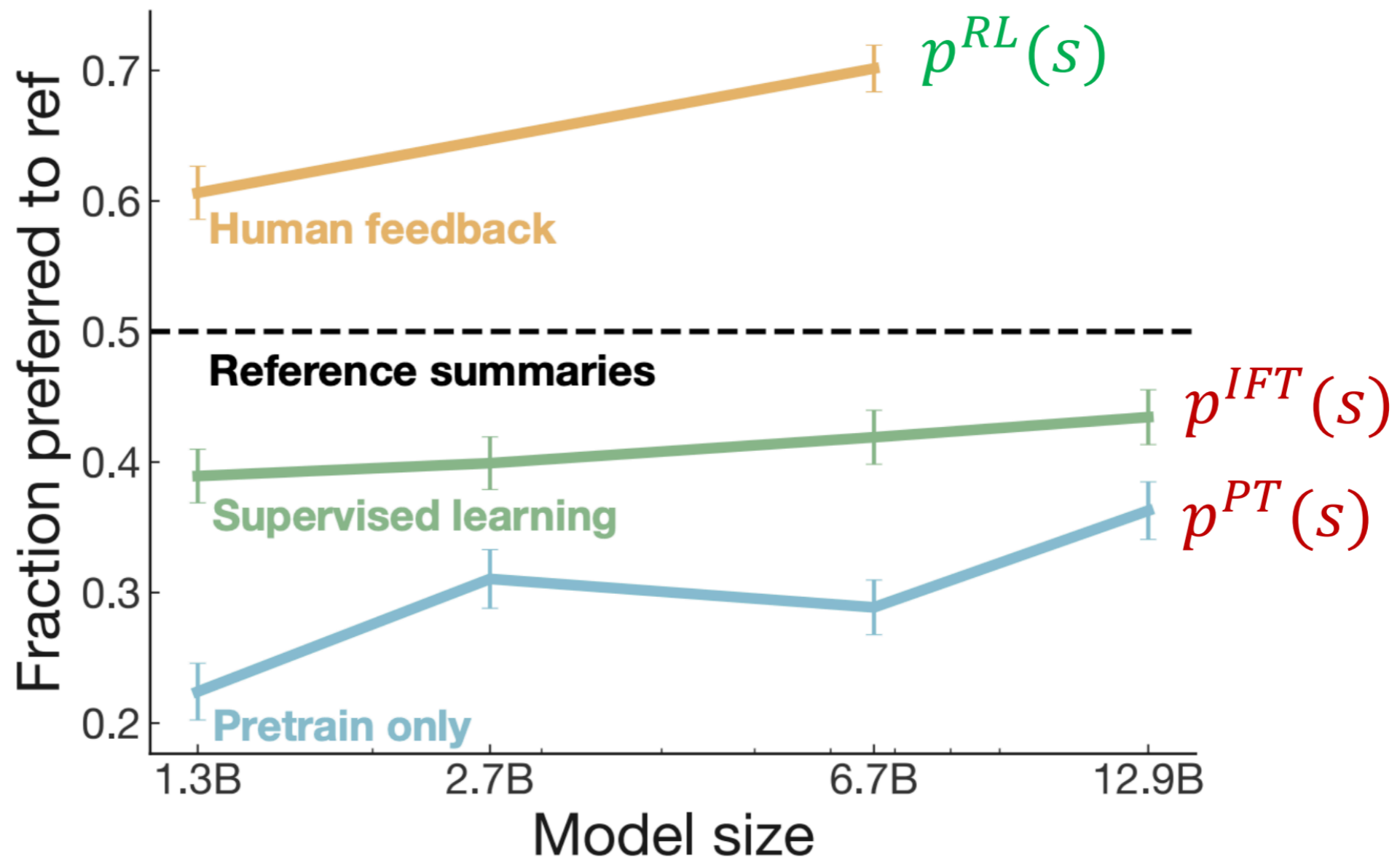
RL from Human Feedback

- Stiennon et al. 2020: RL for summarization.
- Large enough reward model trained on enough data approaches single-human performance



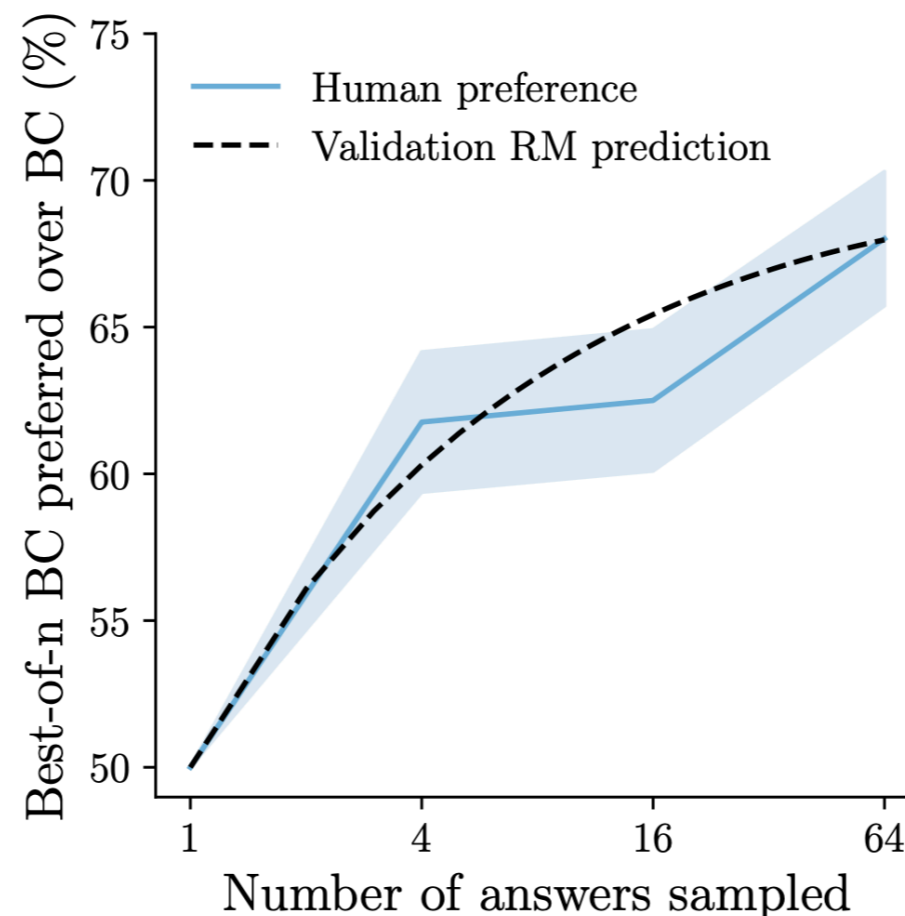
RL from Human Feedback

- Stiennon et al. 2020: RL for summarization.
- Once the reward model is trained, freeze it and use it to update a large pre-trained/fine-tuned LM using RL



Reranking with a Reward Model

- An alternative to RL: once the reward model is trained, use it at inference time to rerank!
- Sample many possible generations, and choose the highest scoring



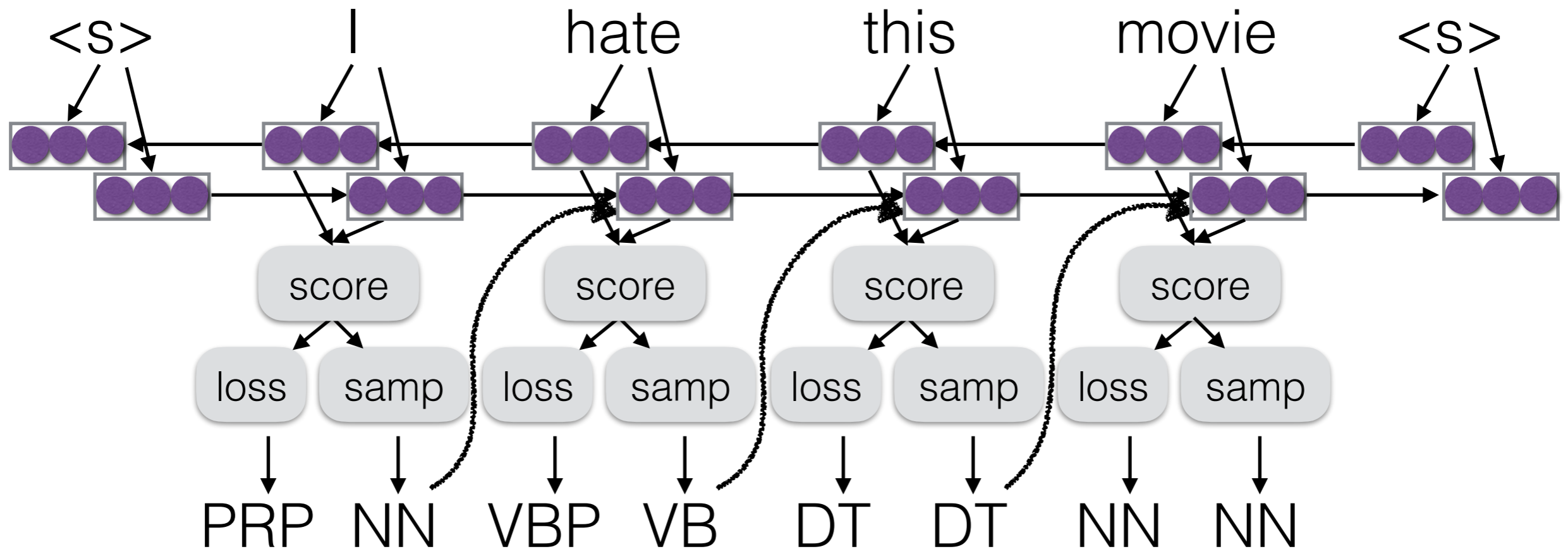
Simpler Remedies to Exposure Bias

What's Wrong w/ Structured Hinge Loss?

- It may work, but...
 - Considers fewer hypotheses, so **unstable**
 - Requires decoding, so **slow**
- Generally must resort to pre-training (and even then, it's not as stable as teacher forcing w/ MLE)

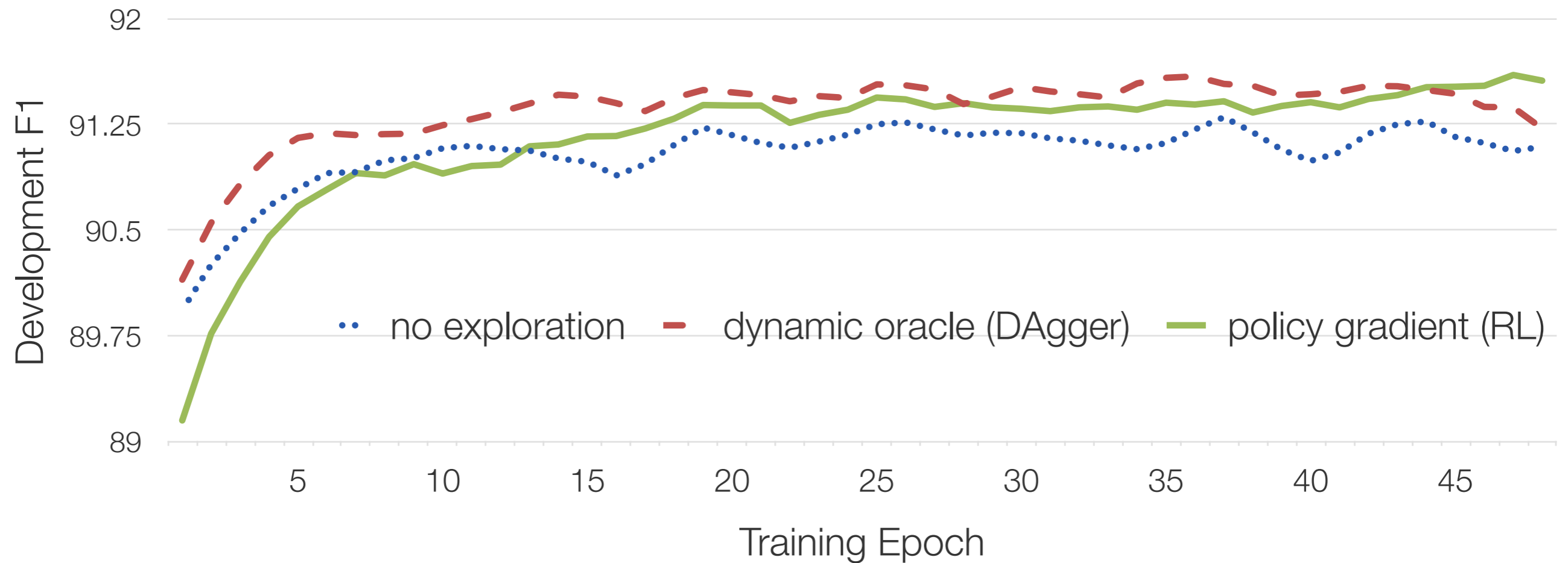
Solution 1: Sample Mistakes in Training (Ross et al. 2010)

- DAgger, also known as “scheduled sampling”, “student forcing” etc., randomly samples wrong decisions and feeds them in



- How to choose the next tag? Use the gold standard, or create a “dynamic oracle” (e.g. Goldberg and Nivre 2013)

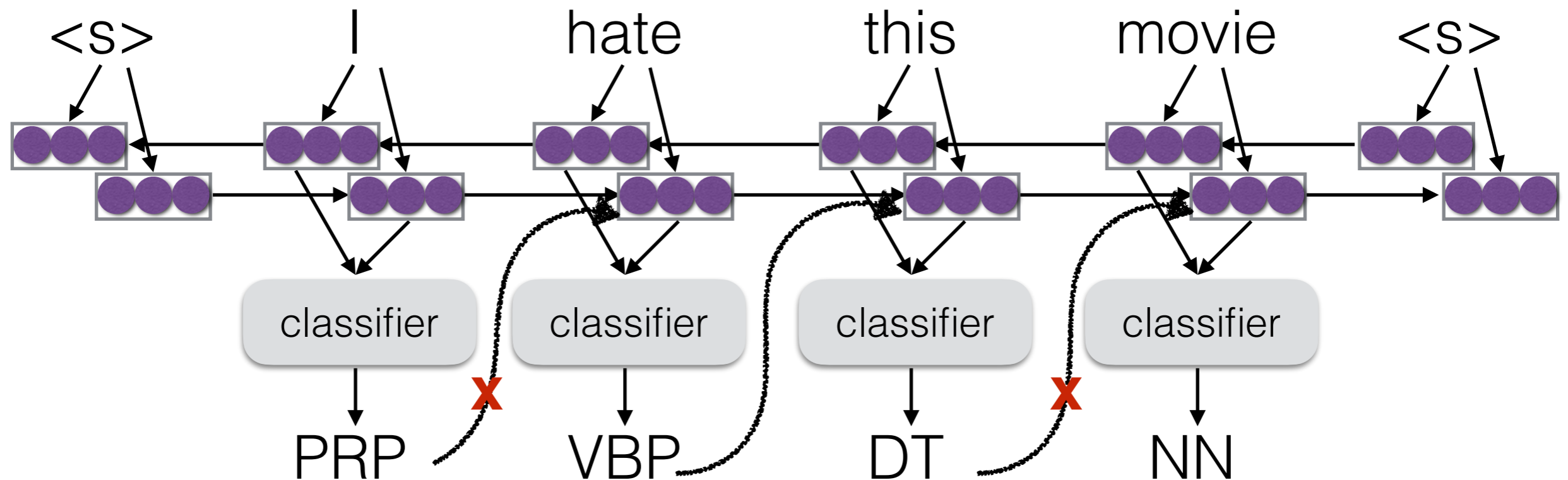
DAgger vs RL



- Fried and Klein 2018: comparison for constituency parsing
- DAgger requires designing an “oracle” which gives best actions to take
- RL often performs as well but takes longer to train

Solution 2: Drop Out Inputs

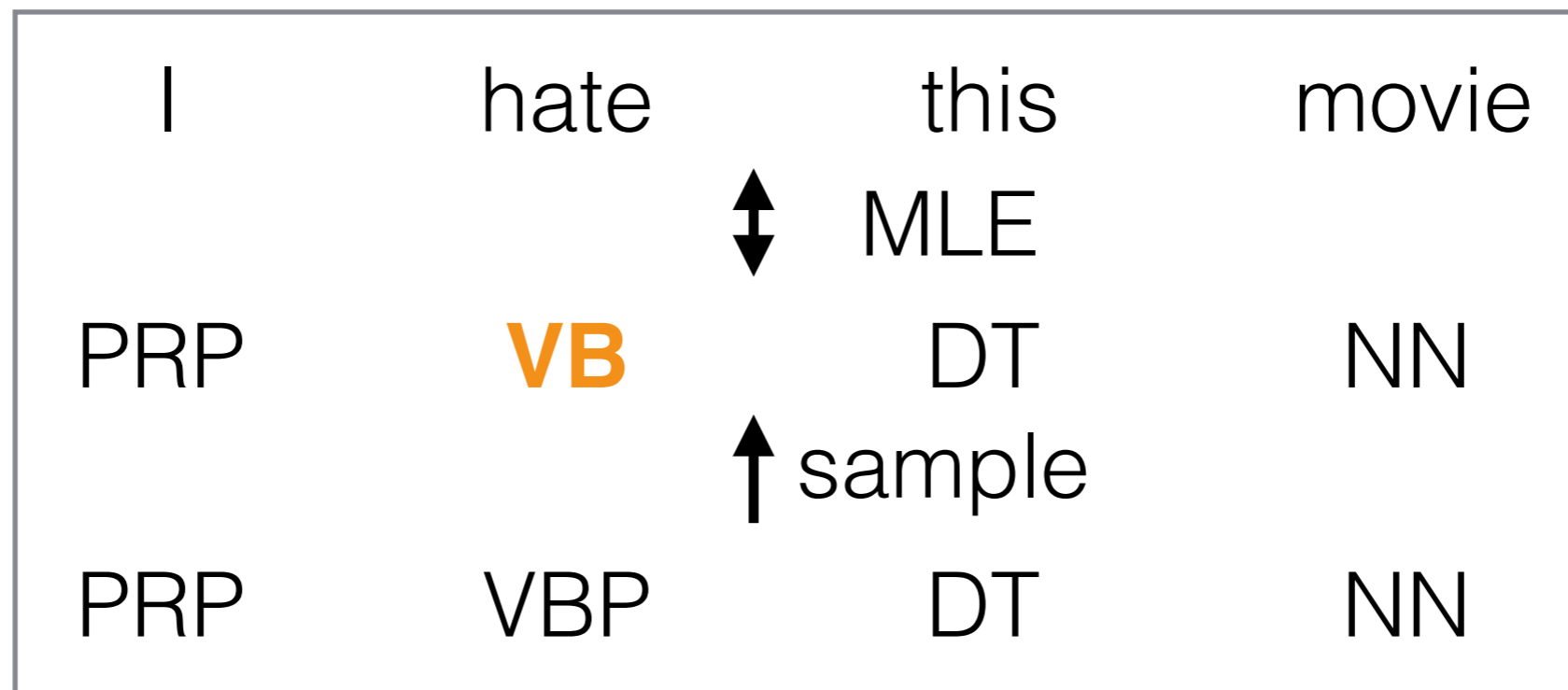
- **Basic idea:** Simply don't input the previous decision sometimes during training (Gal and Ghahramani 2015)



- Helps ensure that the model doesn't rely too heavily on predictions, while still using them

Solution 3: Corrupt Training Data

- Reward augmented maximum likelihood (Nourozi et al. 2016)
- **Basic idea:** randomly sample incorrect training data, train w/ maximum likelihood



- Sampling probability proportional to goodness of output
- Can be shown to approximately minimize risk

Questions?