

CS769 Advanced NLP

Structure Prediction

Junjie Hu



Slides adapted from Graham

<https://junjiehu.github.io/cs769-spring23/>

Goals for Today

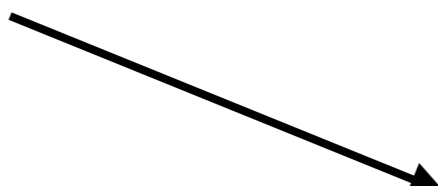
- Problems of structure predictions w.r.t. sentence classification
- Locally (MLE) v.s. Globally normalized likelihood methods
- **Structure perceptron** (hinge loss with margin)
- **Policy Gradient (REINFORCE)**
- **Other simpler solutions**

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 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
- Convolution/self attentional decoders
- CRFs w/ local factors

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

Reminder: 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

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

Globally Normalized Likelihood

Difficulties Training Globally Normalized Models

- Partition function problematic

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

- 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 the **best hypothesis**

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

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

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

Structured Perceptron Algorithm

- An extremely simple way of training (non-probabilistic) global models
- Find the **one-best output** according to the model score, and if its score is better than **the correct answer**, update parameters to fix this
- The one-best output may not be as good as the correct answer, but scores higher! We should update models to either decrease its score or increase the correct answer's score.

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

if $S(\hat{Y} | X; \theta) \geq S(Y | X; \theta)$ **then**

$$\theta \leftarrow \theta + \alpha \left(\frac{\partial S(Y|X;\theta)}{\partial \theta} - \frac{\partial S(\hat{Y}|X;\theta)}{\partial \theta} \right)$$

end if

← Find one best

← If score better than reference

← Increase score of ref, decrease score of one-best (here, SGD update)

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 one-best by 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**

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

- **Global structured perceptron?**

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

- Same computational problems as globally normalized probabilistic models

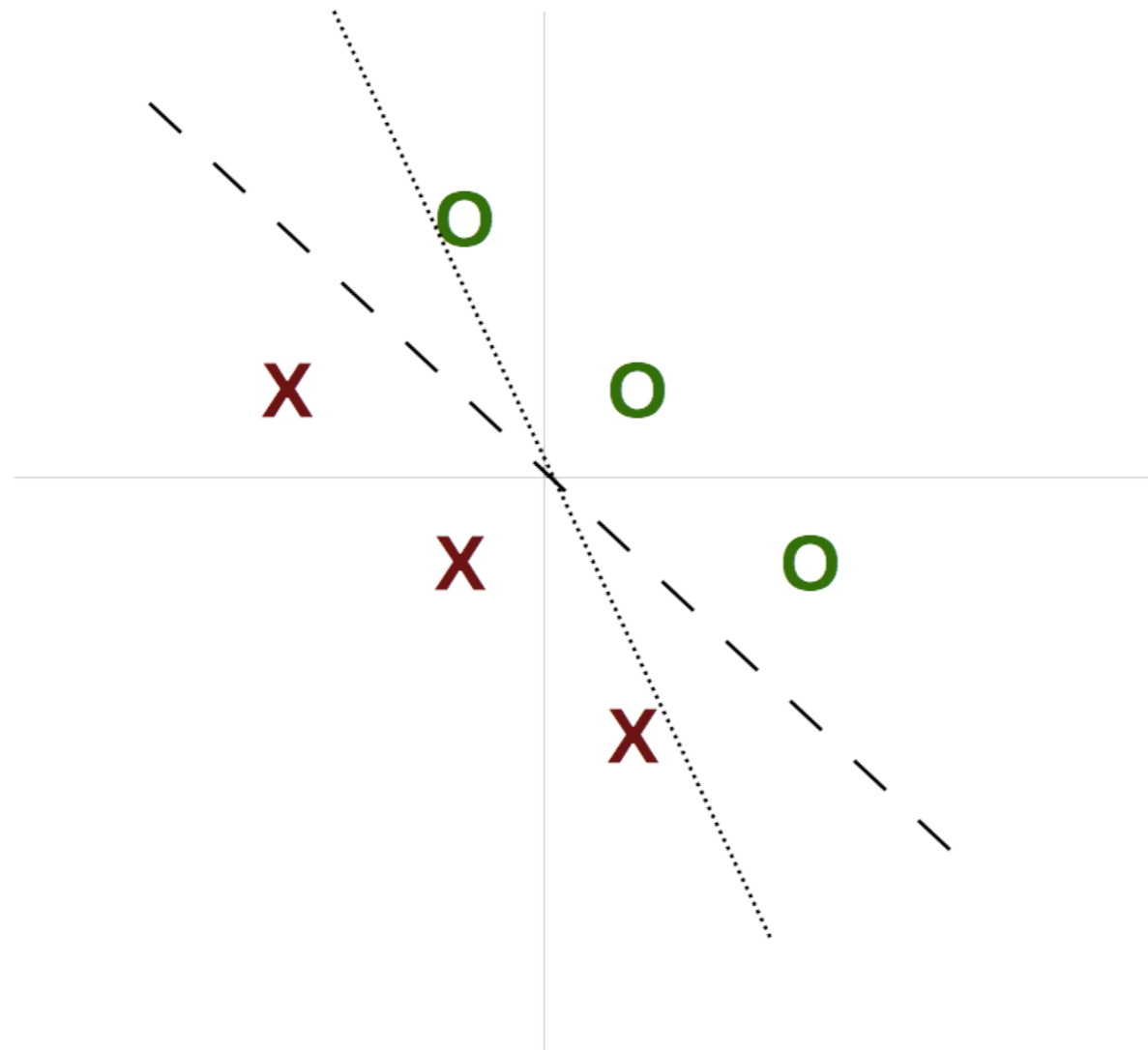
Structured Training and Pre-training

- 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

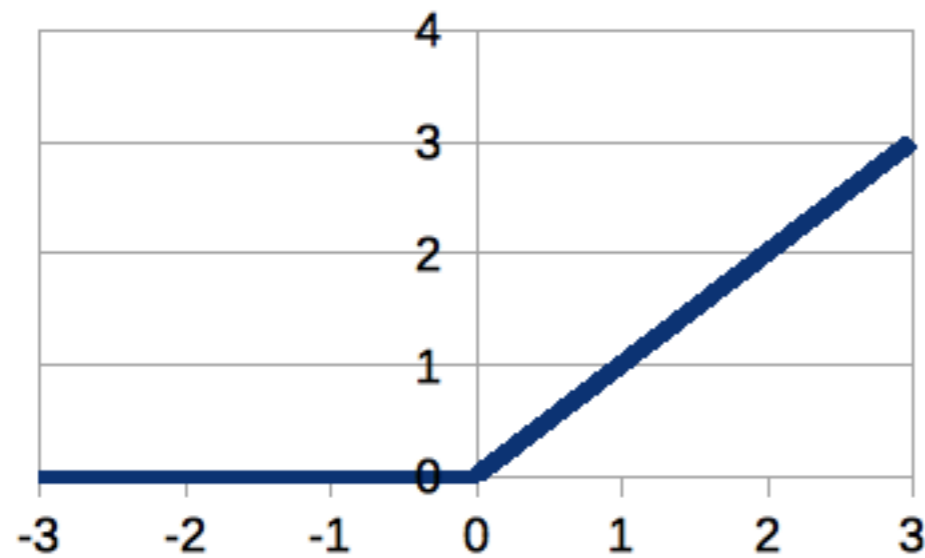
- Which is better, dotted or dashed?



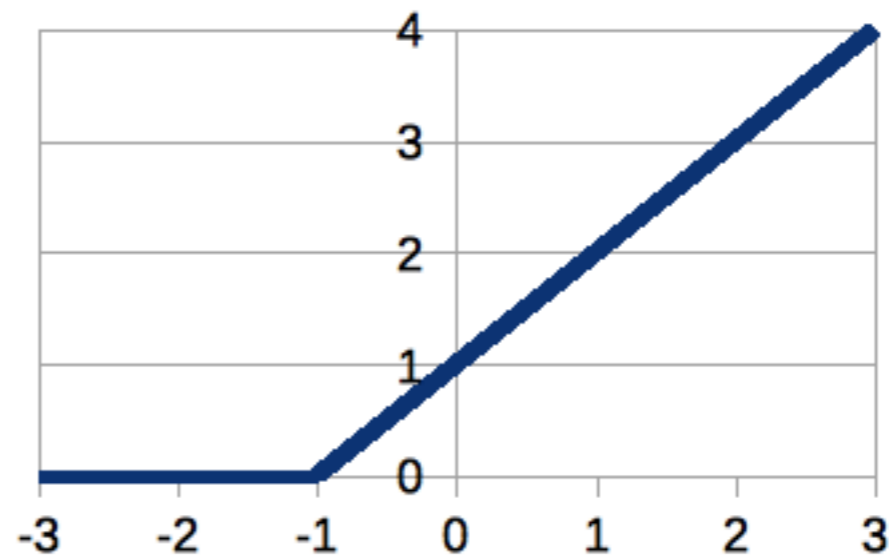
- Both have zero perceptron loss!

Adding a “Margin” with Hinge Loss

- Penalize when incorrect answer is within margin m



Perceptron

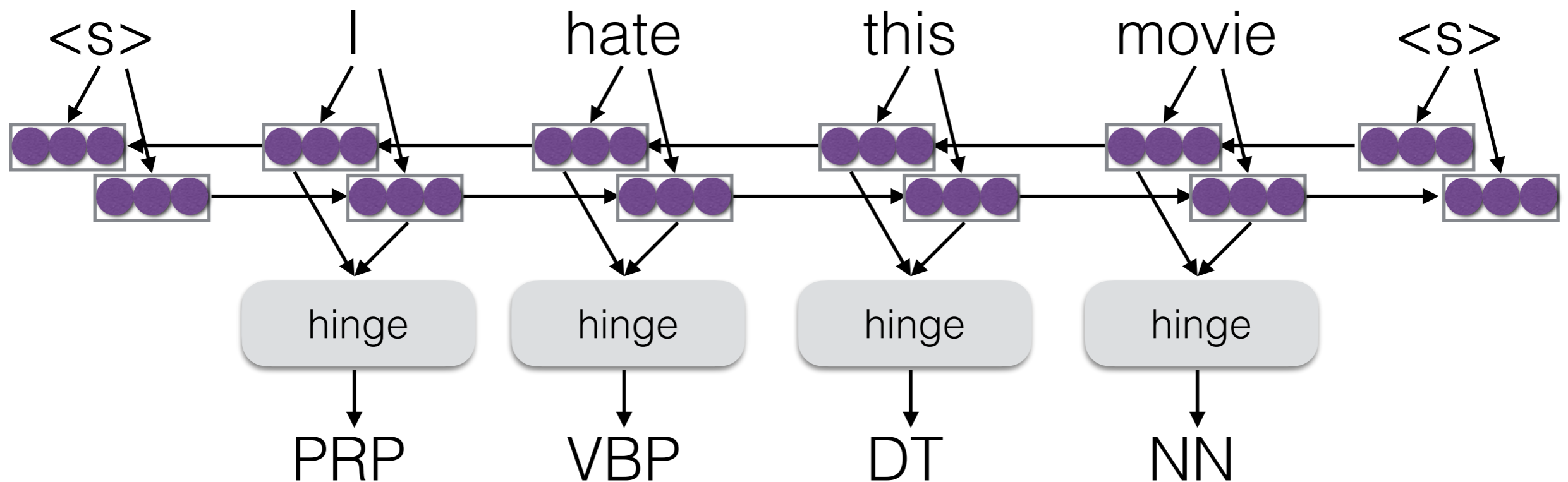


Hinge

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

Hinge Loss for Any Classifier!

- We can swap cross-entropy for hinge loss anytime



e.g.

`loss = logsoftmax(score, answer)`

↓

`loss = hinge(score, answer, m=1)`

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

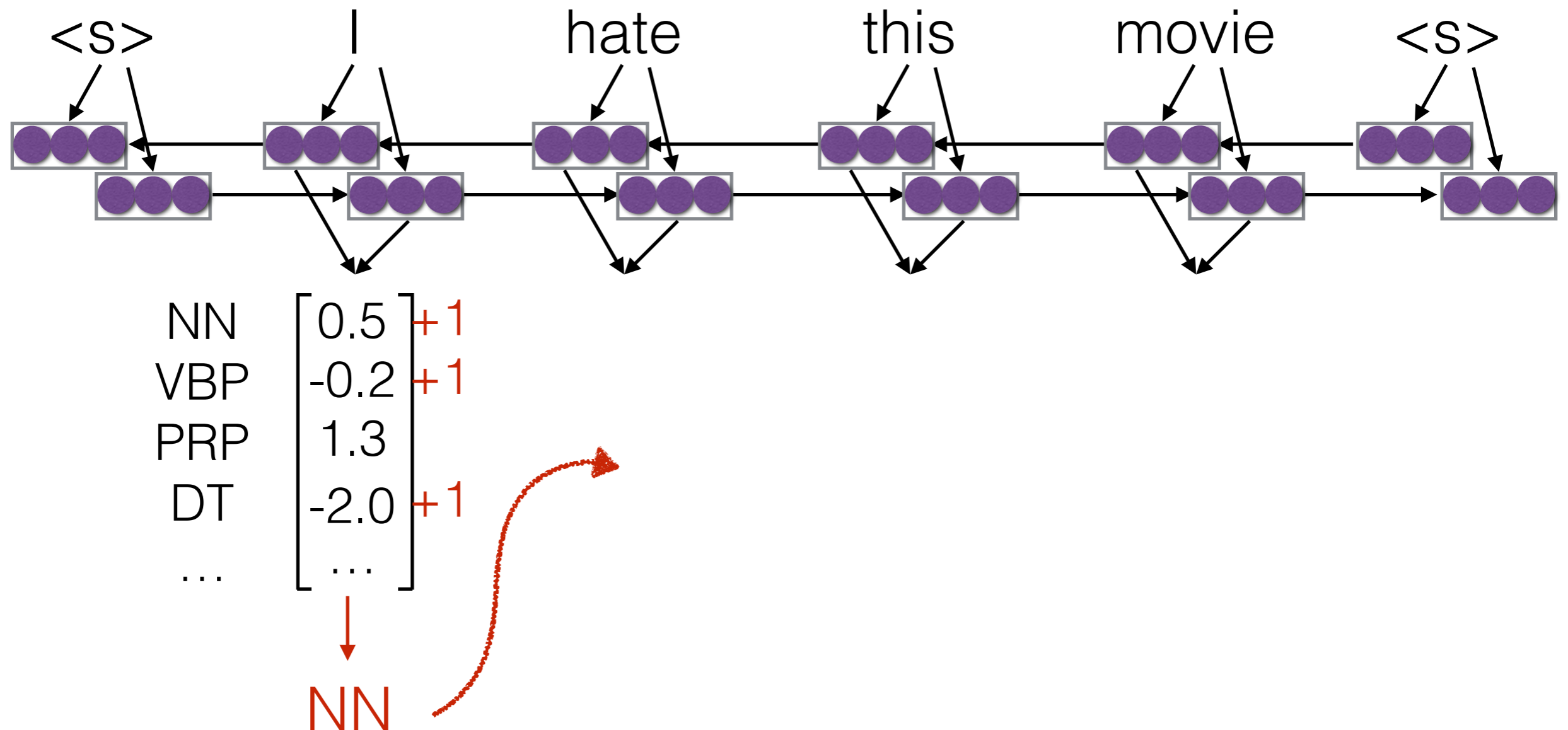
$$\hat{Y} = \operatorname{argmax}_{\tilde{Y} \neq Y} \operatorname{cost}(\tilde{Y}, Y) + S(\tilde{Y} | X; \theta)$$

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

- **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



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.
- 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 (exploration) or argmax (exploitation)** 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 match output (He et al. 2020)

Policy Gradient/REINFORCE

- Add a term that scales the loss by the reward

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

$$\hat{Y} = \arg \max_{\hat{Y} \neq Y} \log P(Y|X)$$

- Outputs that get a bigger reward will get a higher weight
- Quiz: Under what conditions is this equal to MLE?
- \hat{Y} can be obtained by **sampling or argmax (greedy decoding)**, in the same way as self-training (c.f. exploration-exploitation trade-off).

$$\ell_{\text{policy}} = -\frac{1}{N} \sum_{\hat{Y} \sim \log P(Y|X)} R(\hat{Y}, Y) \log P(\hat{Y}|X)$$

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

Stabilizing Reinforcement Learning

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)
- A number of strategies can be used to stabilize

Adding a Baseline

- Basic idea: we have expectations about our reward for a particular sentence

	<u>Reward</u>	<u>Baseline</u>	<u>R-B</u>
“This is an easy sentence”	0.8	0.95	-0.15
“Buffalo Buffalo Buffalo”	0.3	0.1	0.2

- We can instead weight our likelihood by R-B to reflect when we did **better or worse than expected**

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

- (Be careful to not backprop through the baseline)

Calculating Baselines

- Choice of a baseline is arbitrary (often heuristics)
- Option 1: predict final reward using linear from current state (e.g. Ranzato et al. 2016)
 - **Sentence-level:** one baseline per sentence
 - **Decoder state level:** one baseline per output action
- Option 2: use the mean of the rewards in the batch as the baseline (e.g. Dayan 1990)

Increasing Batch Size

- Because each sample will be high variance, we can sample many different examples before performing update
- We can increase the number of examples (roll-outs) done before an update to stabilize
- We can also save previous roll-outs and re-use them when we update parameters (experience replay, Lin 1993)

Warm-start

- Start training with maximum likelihood, then switch over to REINFORCE
- Works only in the scenarios where we can run MLE (not latent variables or standard RL settings)
- MIXER (Ranzato et al. 2016) gradually transitions from MLE to the full objective

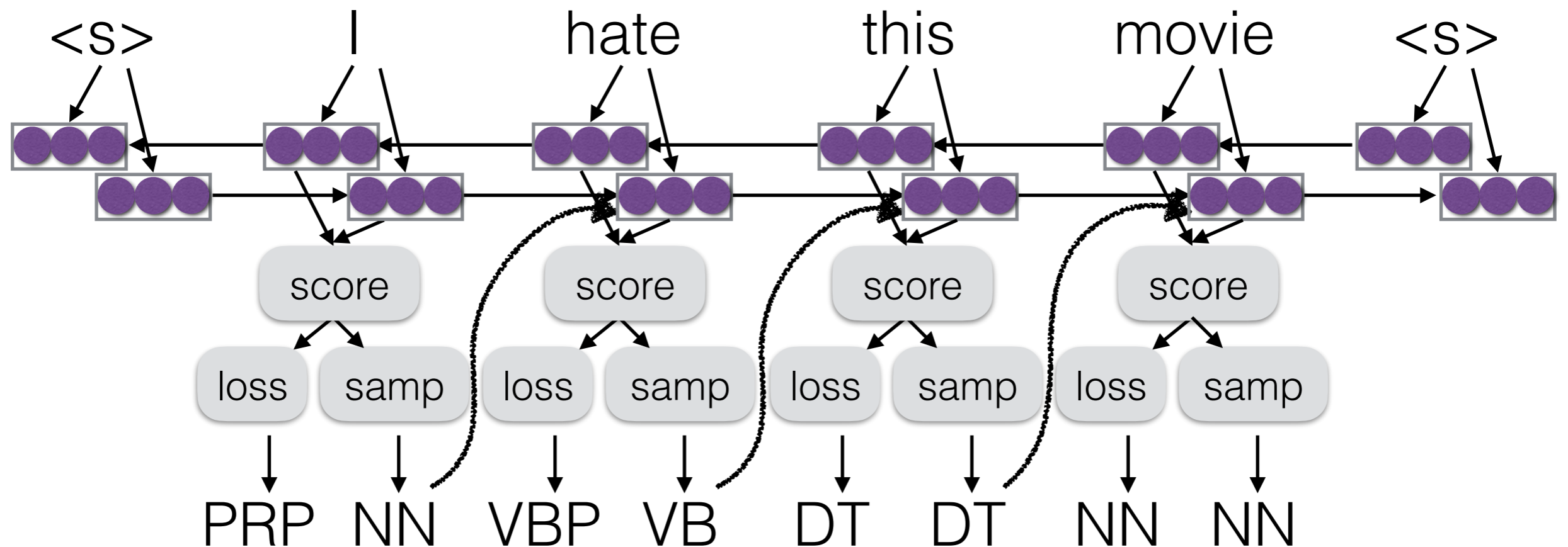
Other Simpler Remedies to Exposure Bias

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)

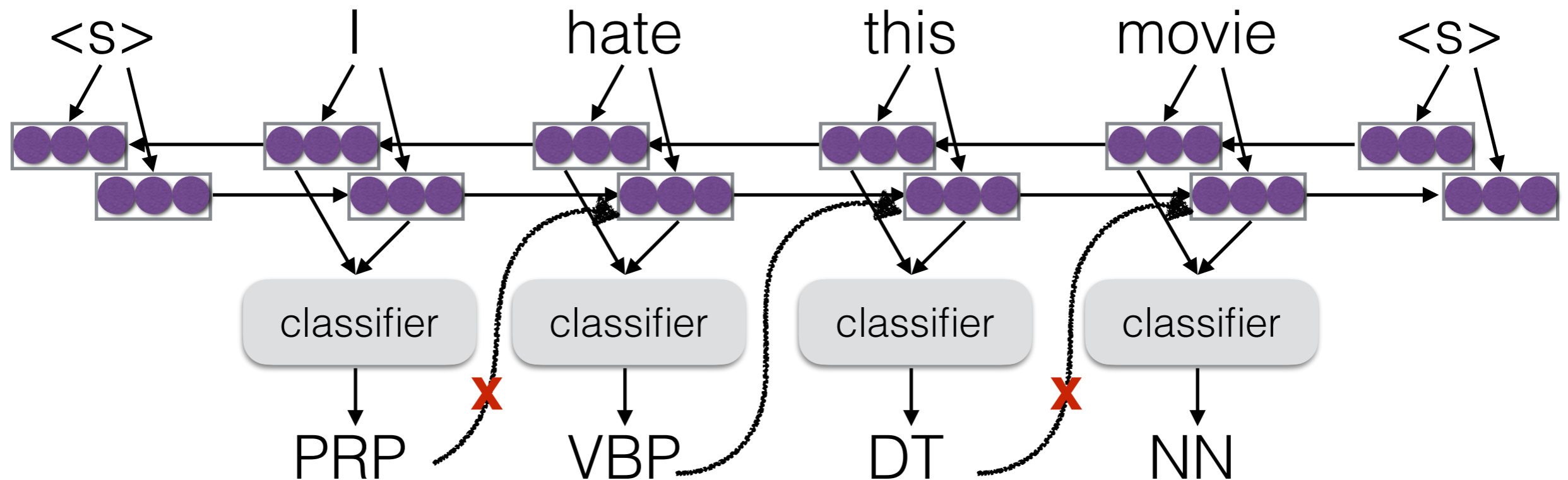
- DAgger, also known as “scheduled sampling”, etc., randomly samples wrong decisions and feeds them in



- Start with no mistakes, and then gradually introduce them using annealing

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

$$\mathcal{L}_{\text{RAML}}(\theta; \tau, \mathcal{D}) = \sum_{(\mathbf{x}, \mathbf{y}^*) \in \mathcal{D}} \left\{ - \sum_{\mathbf{y} \in \mathcal{Y}} q(\mathbf{y} | \mathbf{y}^*; \tau) \log p_{\theta}(\mathbf{y} | \mathbf{x}) \right\}$$

I	hate	this	movie
		↕ MLE	
PRP	NN	DT	NN
		↑ sample	
PRP	VBP	DT	NN

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

Questions?