CS11-711 Advanced NLP

Margin-based and Reinforcement Learning for Structured Prediction

Graham Neubig

Carnegie Mellon University
Language Technologies Institute

Site
https://phontron.com/class/anlp2022/

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

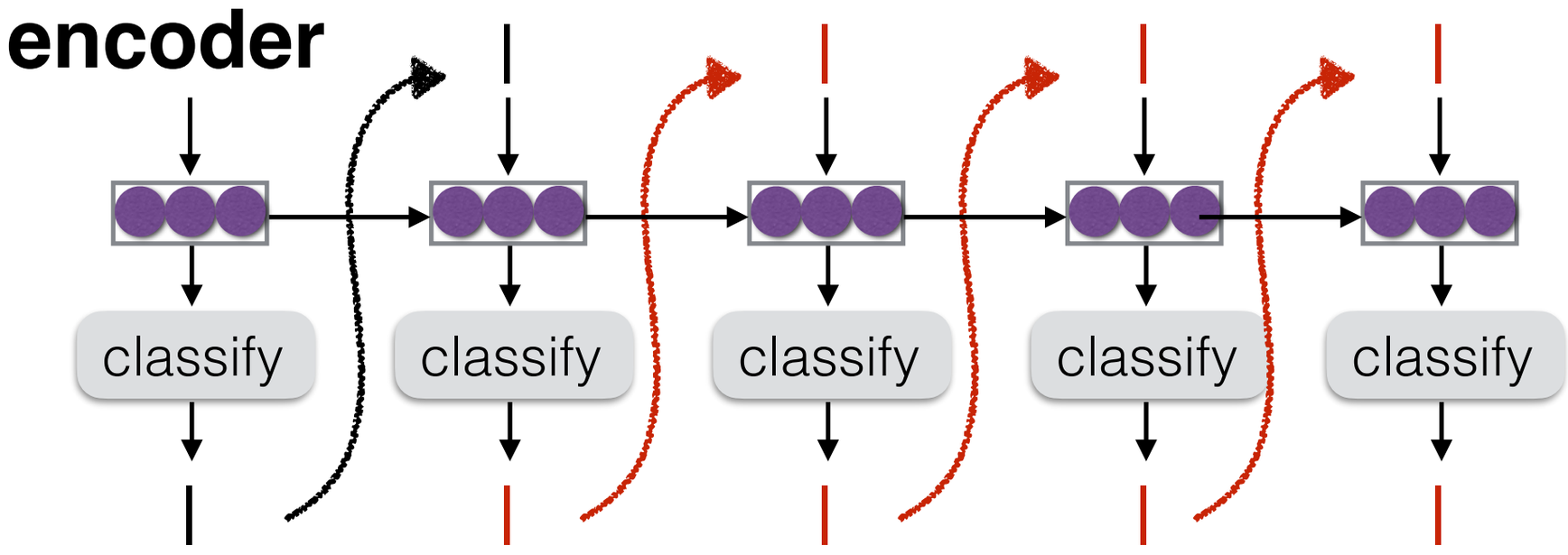
- Exponential/infinite labels (**structured prediction**)

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
  - 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

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

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

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

Difficulties Training Globally Normalized Models

- Partition function problematic

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

- 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 the **best hypothesis**

$$\hat{Y} = \underset{Y}{\text{argmax}}\ P(Y \mid X)$$

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

$$P(Y \mid X) = \frac{e^{S(X,Y)}}{\sum_{\tilde{Y} \in V*} e^{S(X,\tilde{Y})}} \qquad \hat{Y} = \underset{Y}{\text{argmax}}\ S(X,Y)$$

The Structured Perceptron Algorithm

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

$$\hat{Y} = \text{argmax}_{\tilde{Y} \neq Y} S(\tilde{Y} \mid X; \theta)$$ ← Find one best

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

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

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

- Resulting **gradient looks like perceptron algorithm**

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

- 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**

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

- **Global structured perceptron?**

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

  - 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**

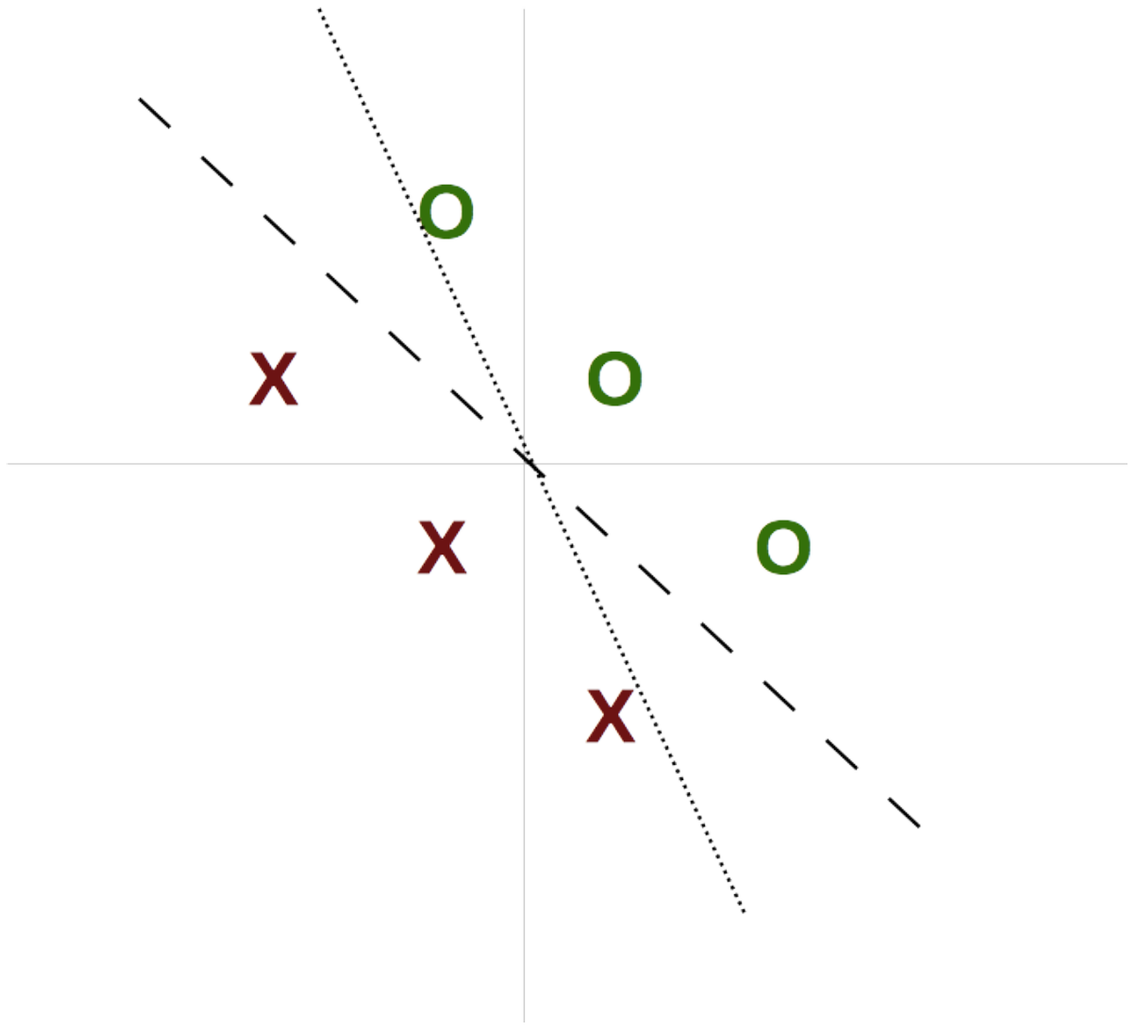
- **Tradeoffs** between training algorithms:
  - Selecting just one negative example is inefficient
  - Teacher forcing efficiently updates all parameters, but suffers from exposure bias

- 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

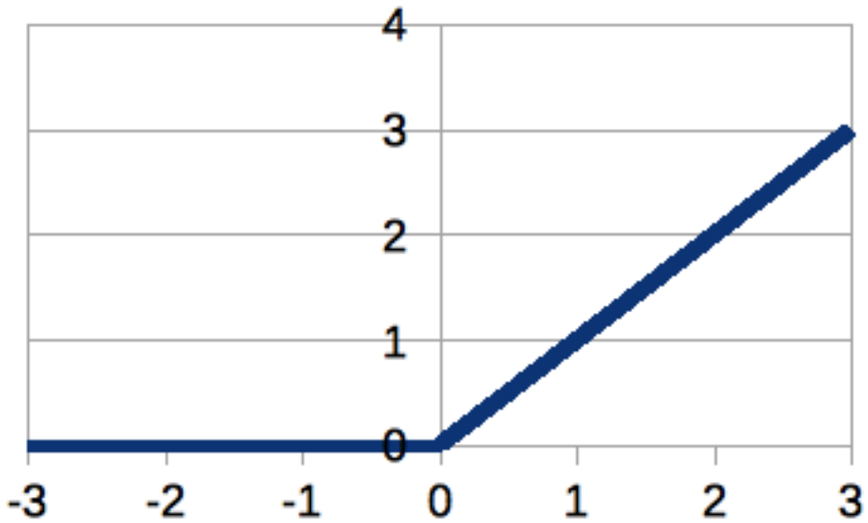
- Which is better, dotted or dashed?



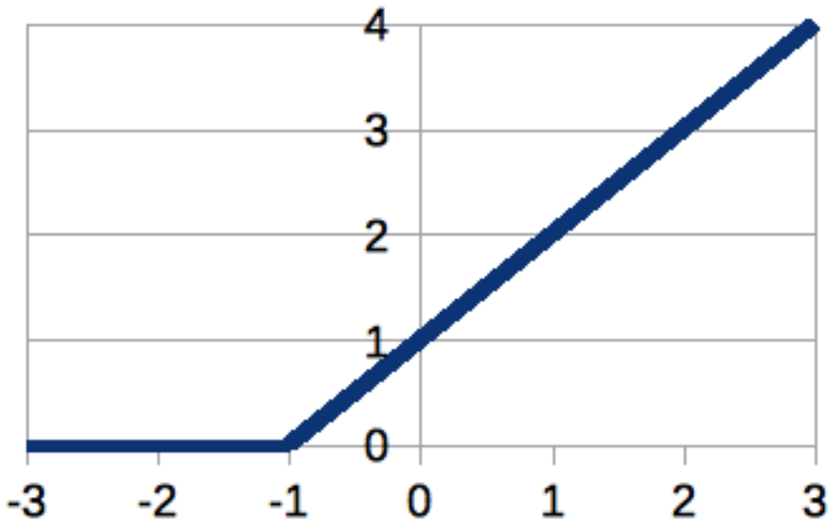
- Both have zero perceptron loss!

Adding a "Margin" with Hinge Loss

- Penalize when incorrect answer is within margin *m*



Perceptron

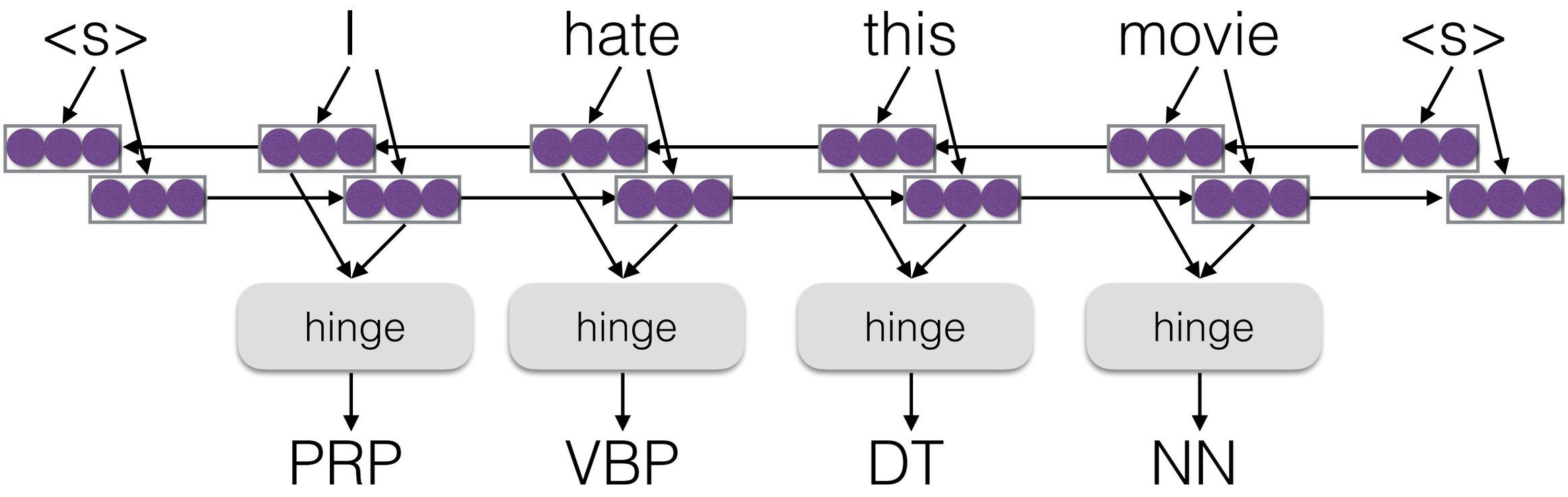


Hinge

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

Hinge Loss for Any Classifier!

- We can swap cross-entropy for hinge loss anytime



e.g. in PyTorch

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

↓

```
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

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

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

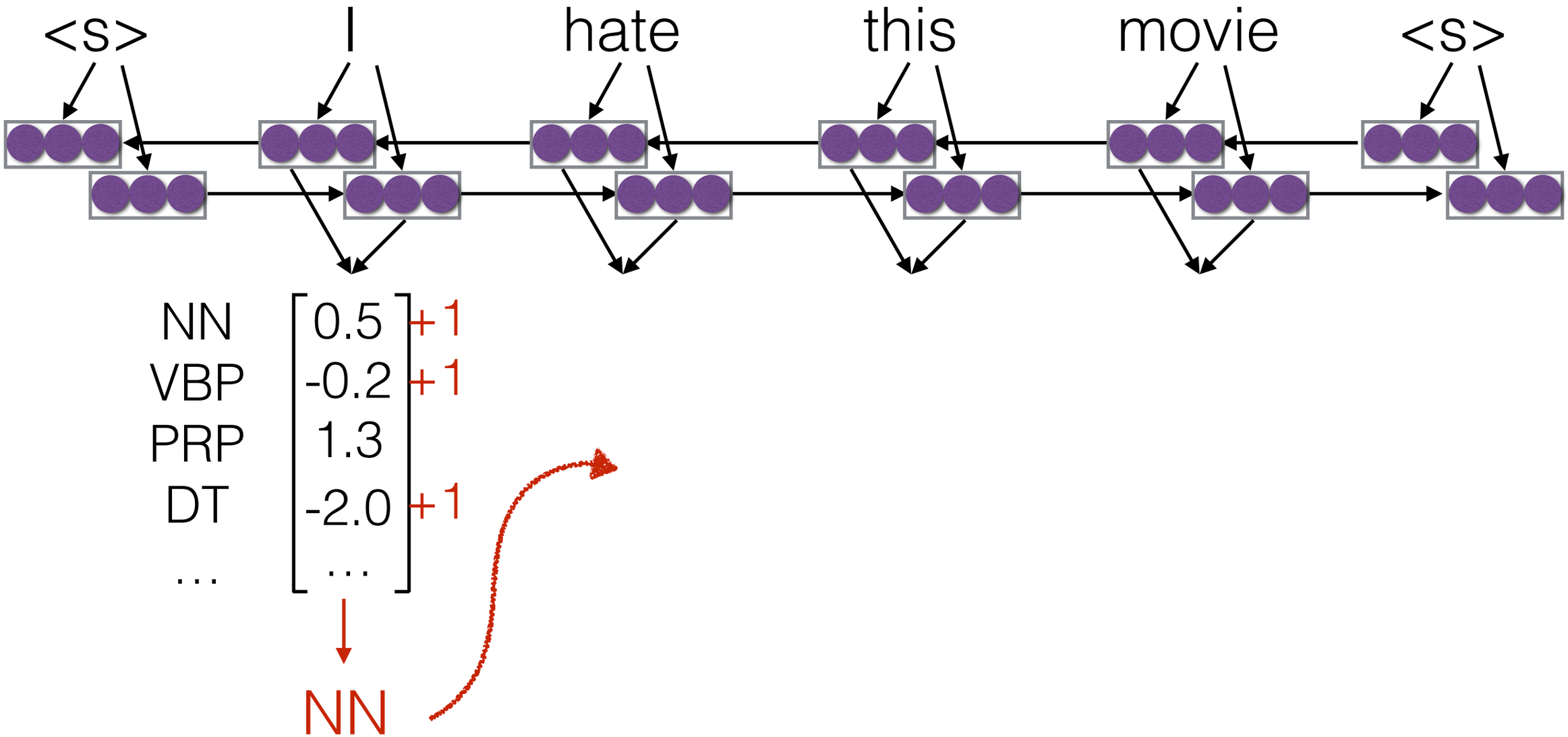
$$\hat{Y} = \text{argmax}_{\tilde{Y} \neq Y} \text{cost}(\tilde{Y}, Y) + S(\tilde{Y} \mid X; \theta)$$

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

- **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



Reinforcment 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

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

- 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

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

- Use this sample (or samples) to maximize likelihood

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

- 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

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

- 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

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

	Reward	Baseline	B-R
“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 B-R to reflect when we did **better or worse than expected**

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

- (Be careful to not backprop through the baseline)

Calculating Baselines

- Choice of a baseline is arbitrary
- 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

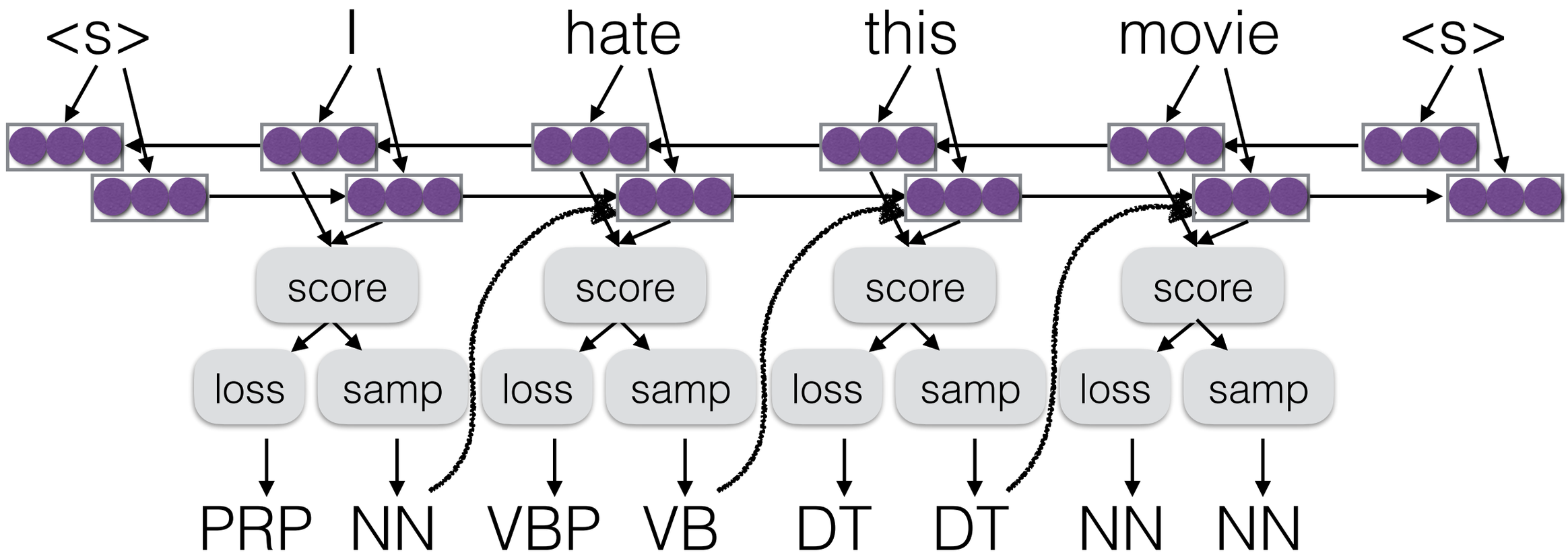
Simpler Remedies to Exposure Bias

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

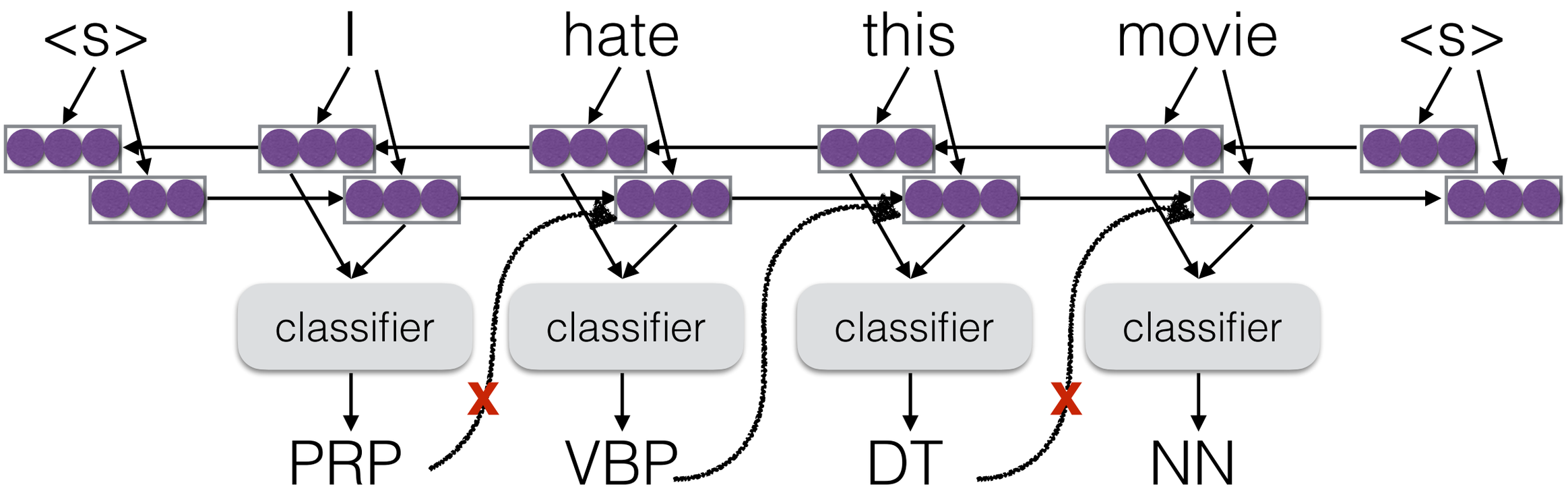


  - Start with no mistakes, and then gradually introduce them using annealing

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

Solution 2: Drop Out Inputs

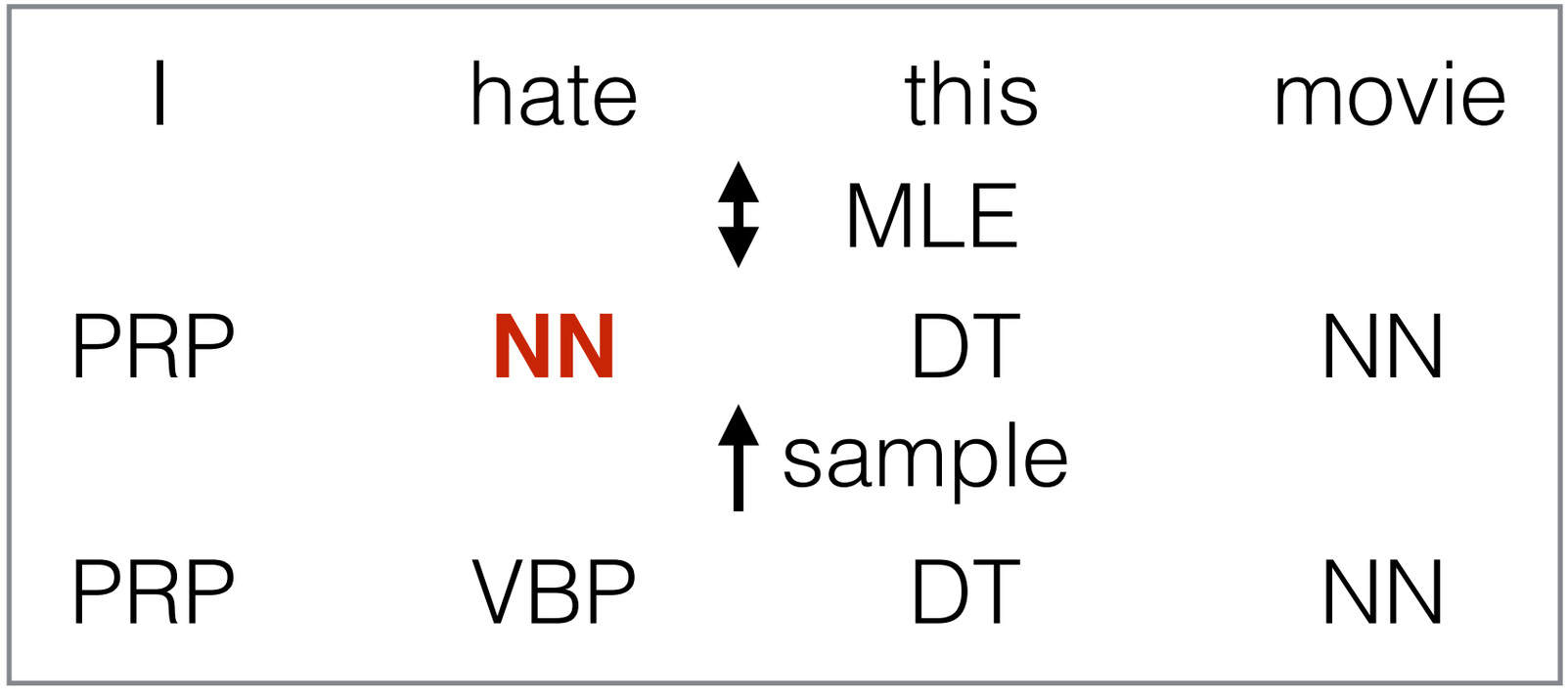
- **Basic idea:** Simply don't input the previous decision sometimes during training (Gal and Ghahramani 2015)



- 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 (next class)

Questions?