CS11-747 Neural Networks for NLP

Structured Prediction Basics

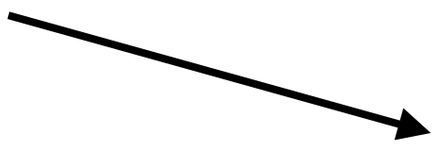
Graham Neubig

Carnegie Mellon University
Language Technologies Institute

Site
https://phontron.com/class/nn4nlp2019/

A Prediction Problem

I hate this movie



very good
good
neutral
bad
very bad

I love this movie

very good
good
neutral
bad
very bad

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*

Why Call it “Structured” Prediction?

- Classes are to numerous to enumerate
- Need some sort of method to exploit the *problem structure* to learn efficiently
- Example of “structure”, the following two outputs are similar:

PRP VBP DT NN

PRP VBP VBP NN

Many Varieties of Structured Prediction!

- **Models:**
  - RNN-based decoders
  - Convolution/self attentional decoders
  - CRFs w/ local factors
- **Training algorithms:**
  - Structured perceptron, structured large margin
  - Sampling corruptions of data
  - Exact enumeration with dynamic programs
  - Reinforcement learning/minimum risk training

An Example Structured Prediction Problem: Sequence Labeling

Sequence Labeling

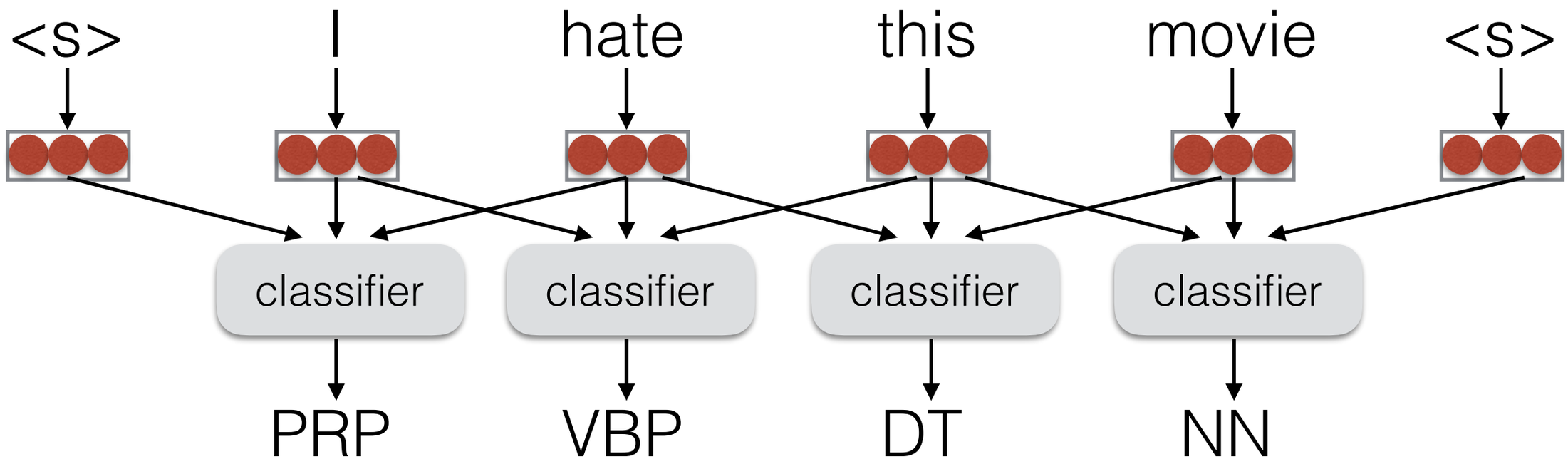
- One tag for one word
- e.g. Part of speech tagging

I	hate	this	movie
↓	↓	↓	↓
PRP	VBP	DT	NN

- e.g. Named entity recognition

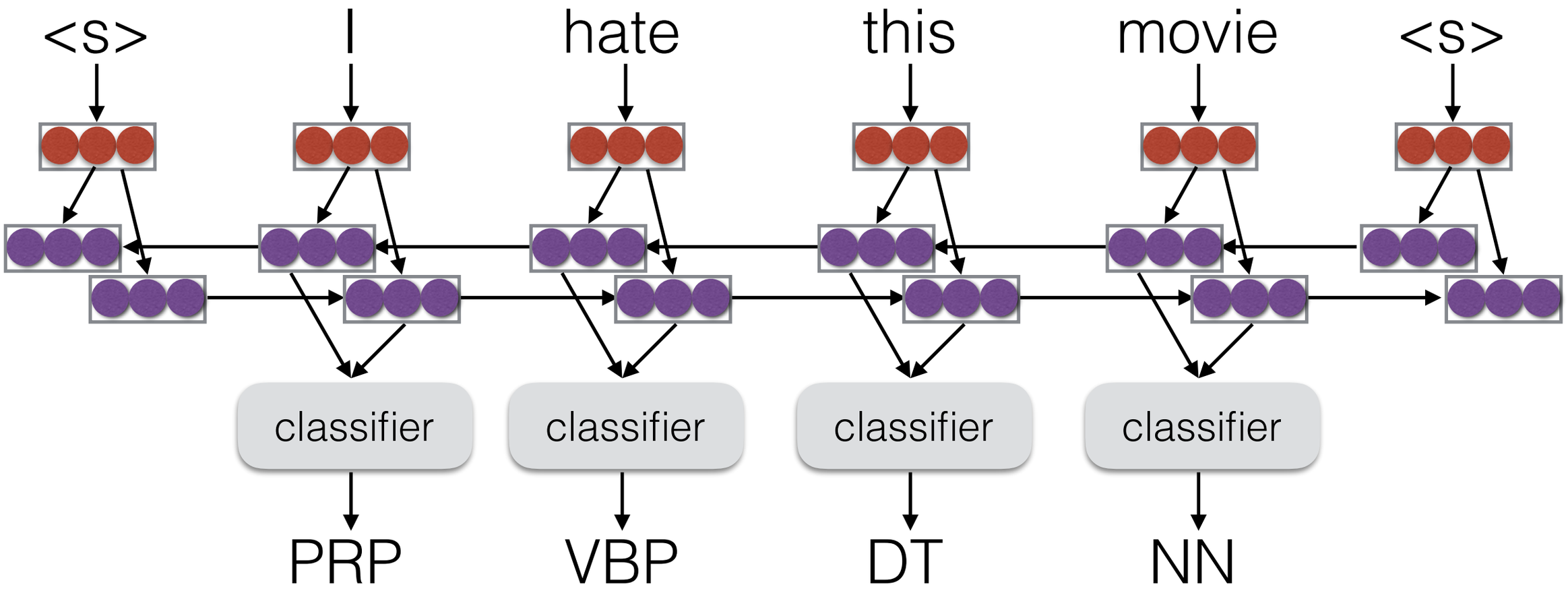
The	movie	featured	Keanu	Reeves
↓	↓	↓	↓	↓
O	O	O	B-PER	I-PER

Sequence Labeling as Independent Classification



- Structured prediction task, but not structured prediction model: multi-class classification

Sequence Labeling w/ BiLSTM



- Still not modeling output structure! Outputs are independent

Why Model Interactions in Output?

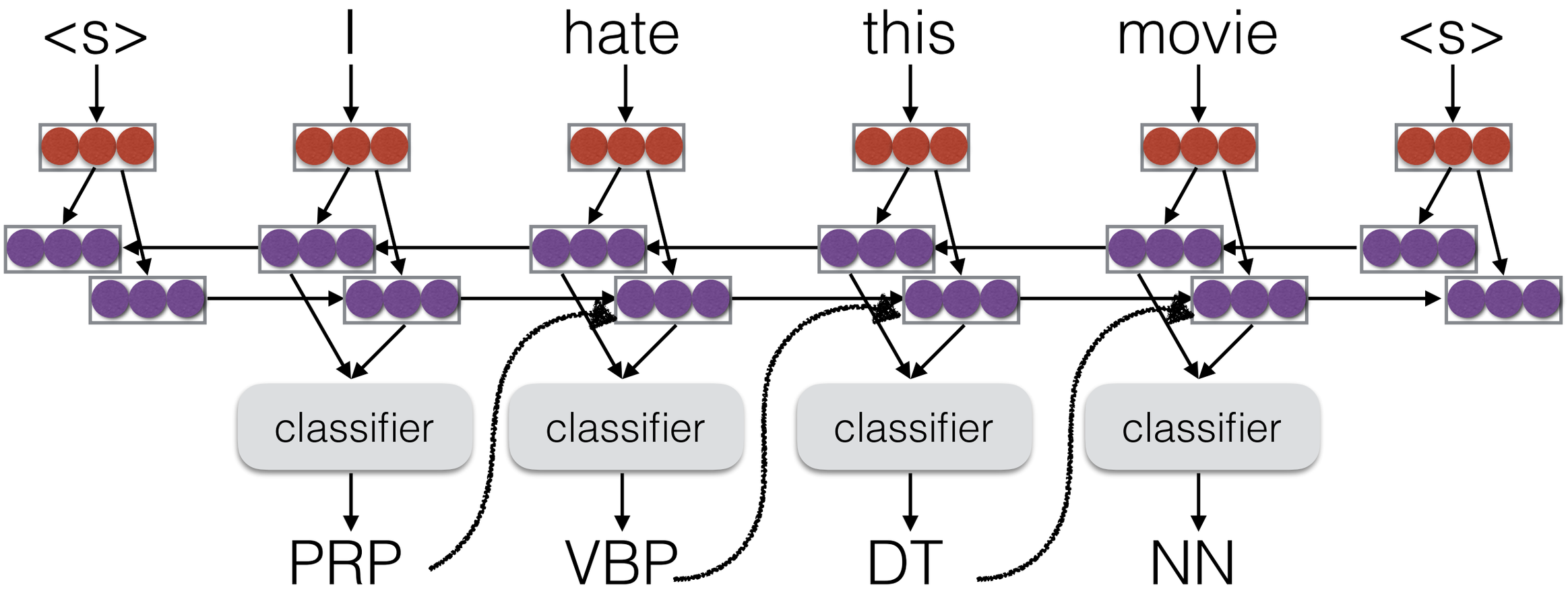
- Consistency is important!

time	flies	like	an	arrow	
NN	VBZ	IN	DT	NN	(time moves similarly to an arrow)
NN	NNS	VB	DT	NN	(“time flies” are fond of arrows)
VB	NNS	IN	DT	NN	(please measure the time of flies similarly to how an arrow would)

↓ max frequency

NN	NNS	IN	DT	NN	(“time flies” that are similar to an arrow)

A Tagger Considering Output Structure



- Tags are inter-dependent
- Basically similar to encoder-decoder model
  (this is like an seq2seq model with hard attention on a single word)

Training Structured Models

- Simplest training method “teacher forcing”
- Just feed in the *correct* previous tag

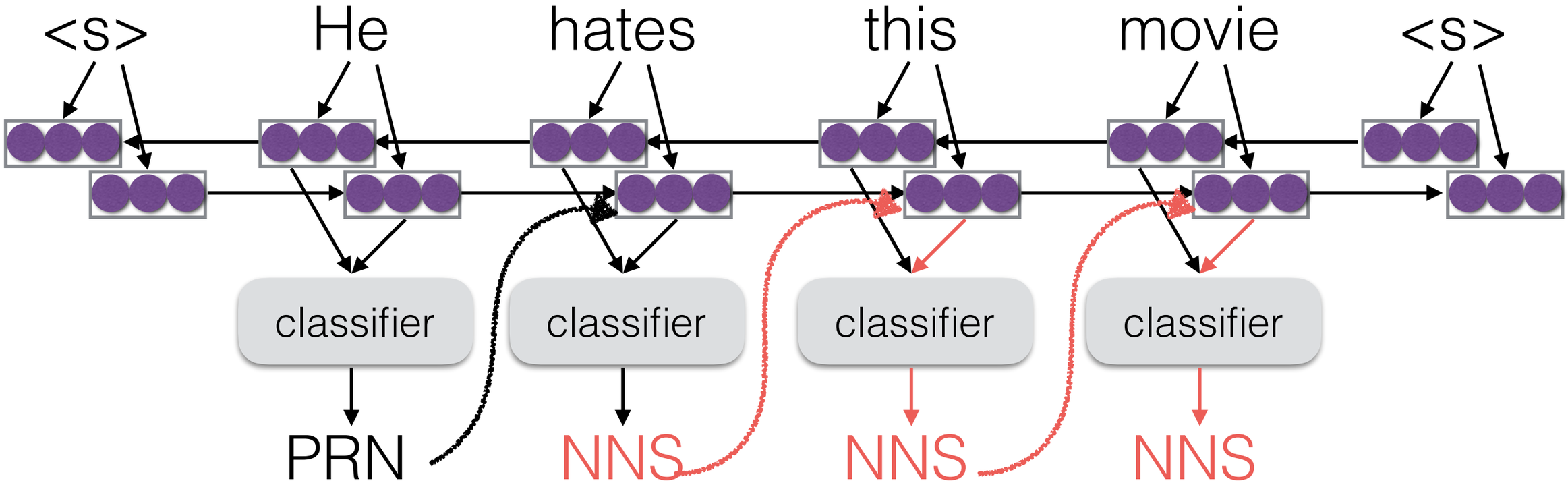
Let’s Try It!

`bilstm-tagger.py`

`bilstm-variant-tagger.py -teacher`

Teacher Forcing and 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

Local Normalization vs. Global Normalization

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

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

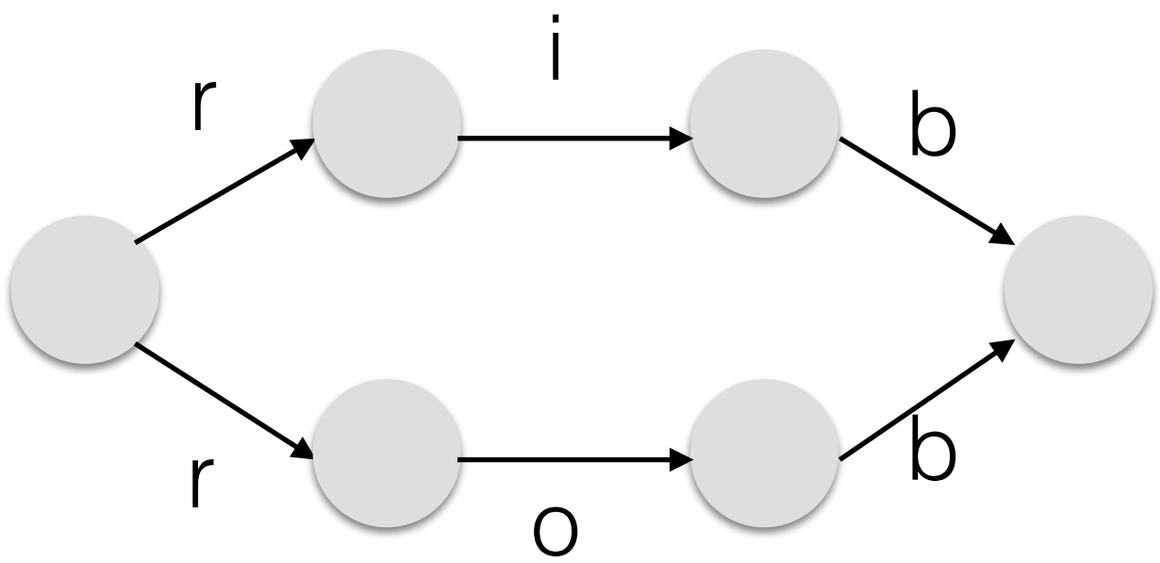
- **Globally normalized models (a.k.a. energy-based models):** each sentence has a score, which is not normalized over a particular decision

$$P(Y \mid X) = \frac{e^{\sum_{j=1}^{|Y|} S(y_j|X,y_1,\ldots,y_{j-1})}}{\sum_{\tilde{Y} \in V*} e^{\sum_{j=1}^{|\tilde{Y}|} S(\tilde{y}_j|X,\tilde{y}_1,\ldots,\tilde{y}_{j-1})}}$$

Local Normalization and Label Bias

- Even if the model detects a “failure state” it cannot reduce its score directly (Lafferty et al. 2001)

Looks ok! P(i|r) = 1



Looks horrible! But no other options so P(o|r) = 1

- **Label bias:** the problem of preferring model states decisions that have few decisions

Problems Training Globally Normalized Models

- **Problem:** the denominator is too big to expand naively
- We must do something tricky:
  - Consider only a subset of hypotheses (this and next time)
  - Design the model so we can efficiently enumerate all hypotheses (in a bit)

Structured Perceptron

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, label bias

- Thus, it is common to **pre-train with teacher forcing, then fine-tune with more complicated algorithm**

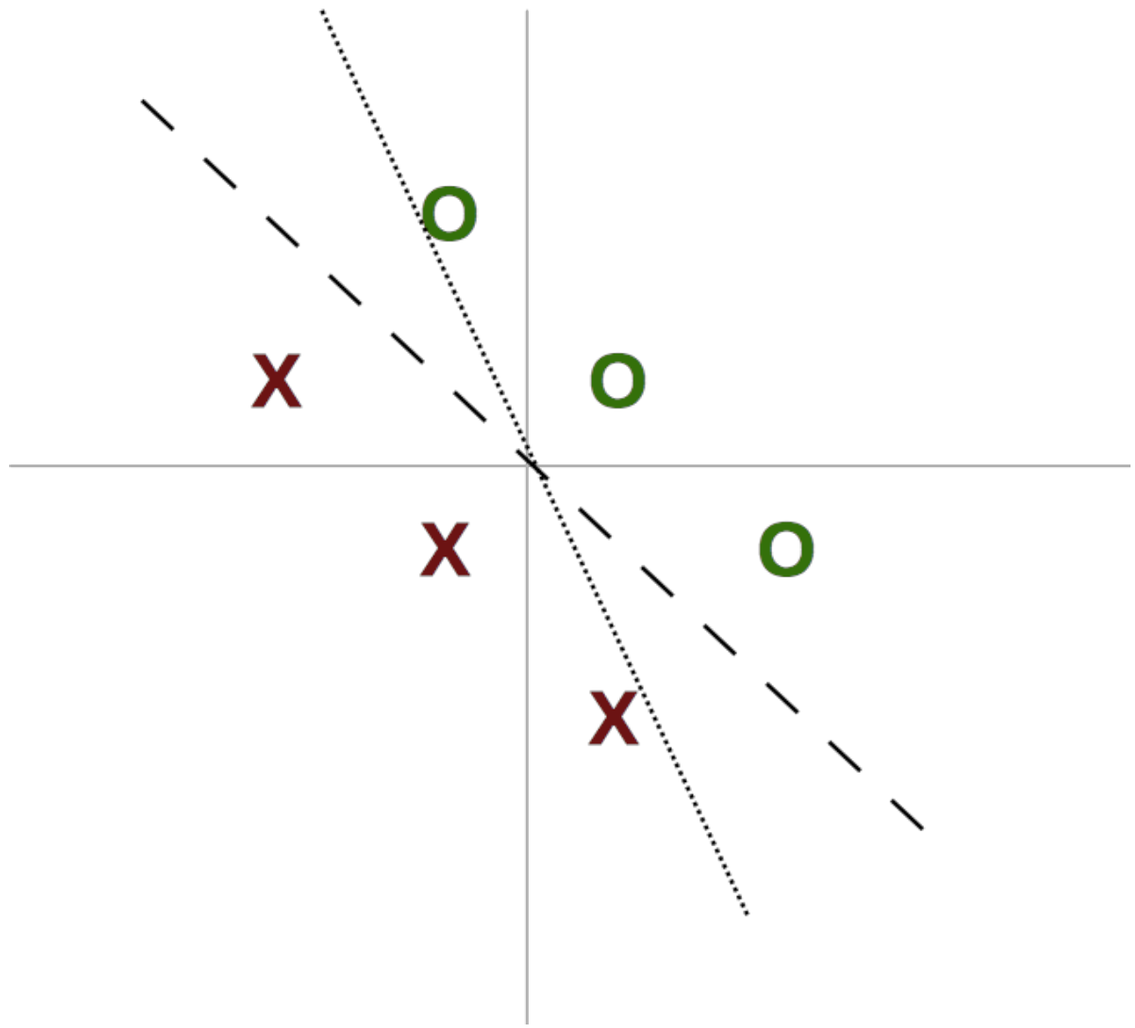
Let’s Try It!

```
bilstm-variant-tagger.py -percep
```

Hinge Loss and Cost-sensitive Training

Perceptron and Uncertainty

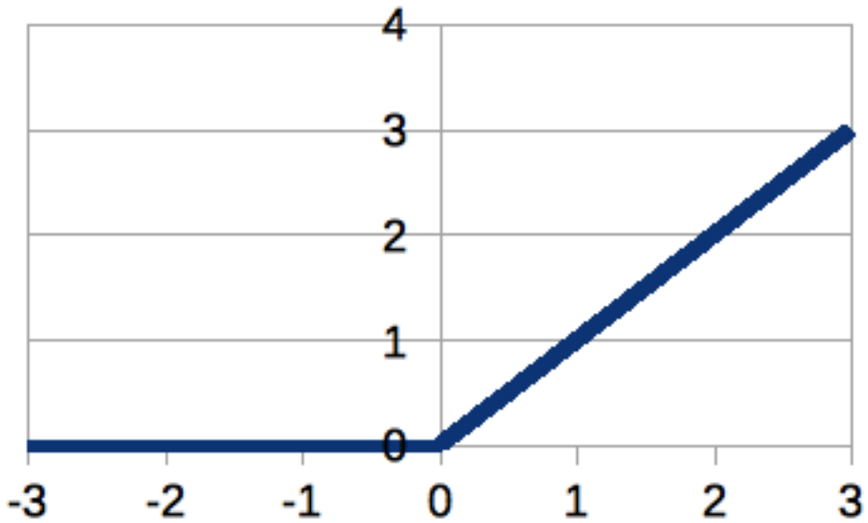
- Which is better, dotted or dashed?



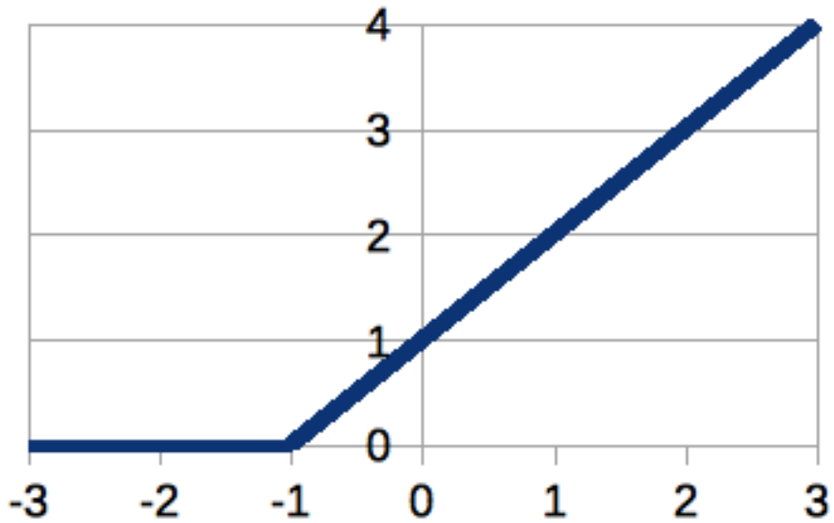
- Both have zero perceptron loss!

Adding a "Margin" with Hinge Loss

- Penalize when incorrect answer is within margin *m*



Perceptron

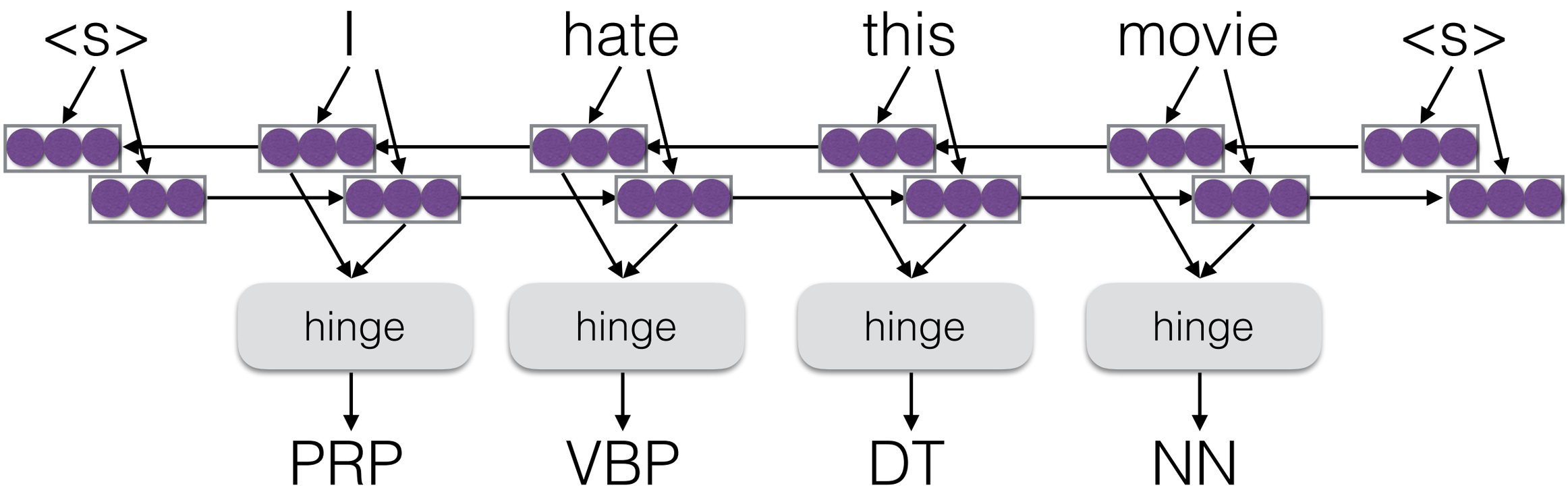


Hinge

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

Hinge Loss for Any Classifier!

- We can swap cross-entropy for hinge loss anytime




```
loss = dy.pickneglogsoftmax(score, answer)
```

↓

```
loss = dy.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

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

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

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

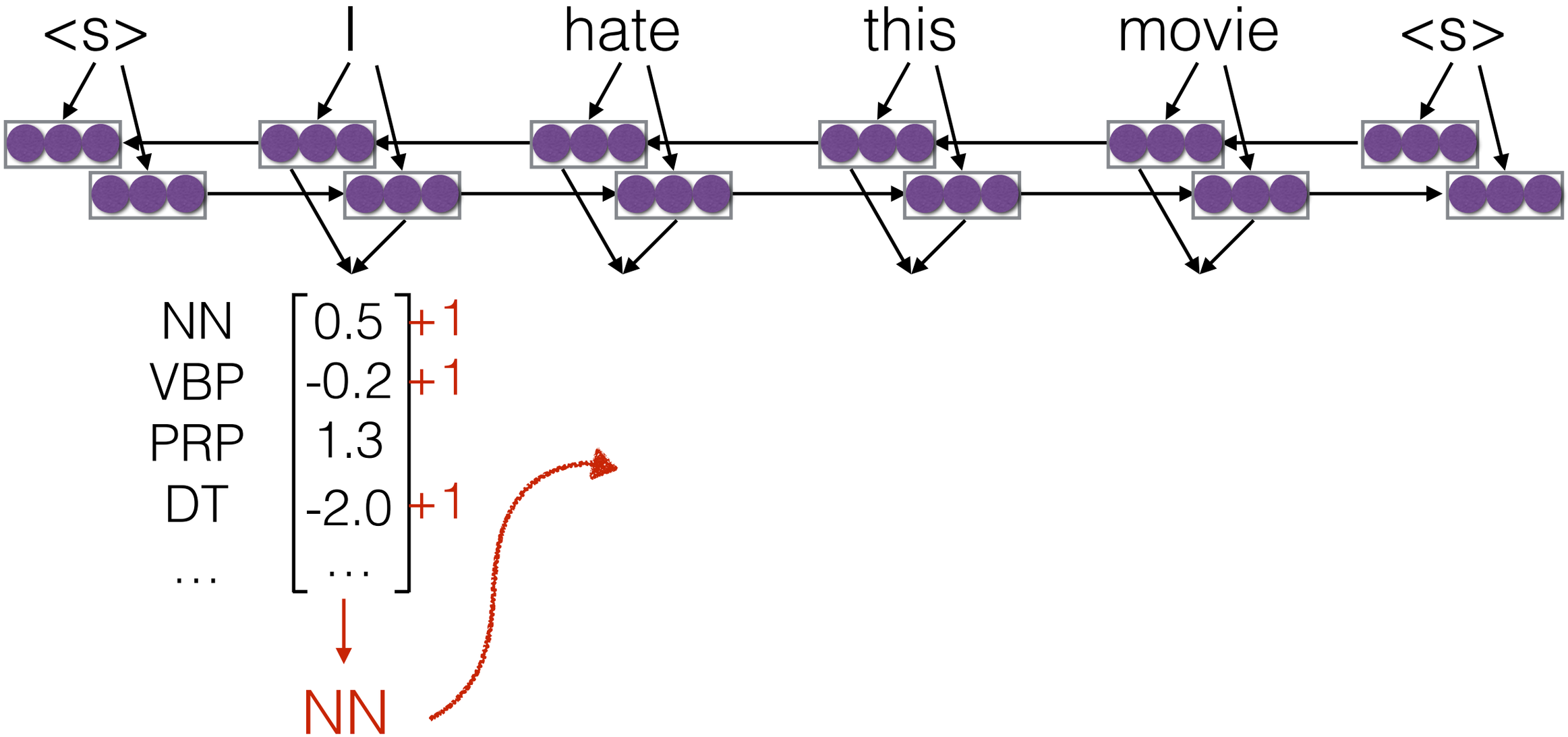
$$\ell_{\text{ca-hinge}}(X, Y; \theta) = \max(0, \text{cost}(\hat{Y}, Y) + S(\hat{Y} \mid X; \theta) - S(Y \mid X; \theta))$$

- **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



Let’s Try It!

```
bilstm-variant-tagger.py -hinge
```

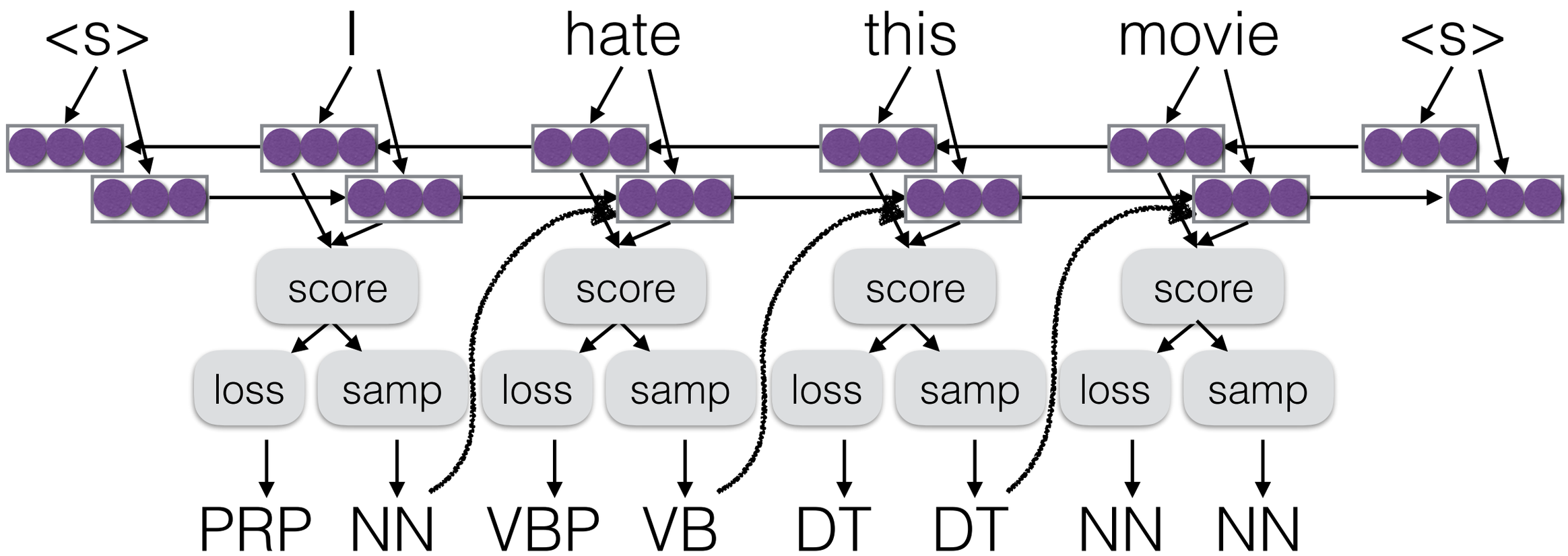
Simpler Remedies to Exposure Bias

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

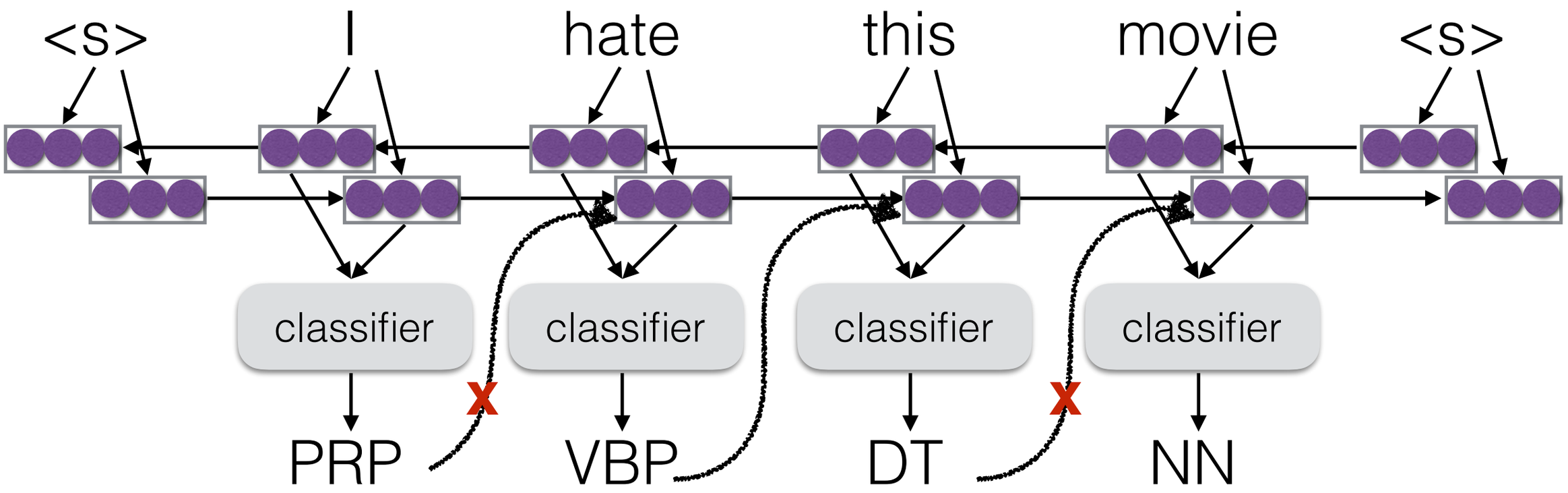


  - Start with no mistakes, and then gradually introduce them using annealing

- 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

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?