Margin-Based Methods and Reinforcement Learning for Structured Prediction

Brendon Boldt, Graham Neubig

November 16, 2021

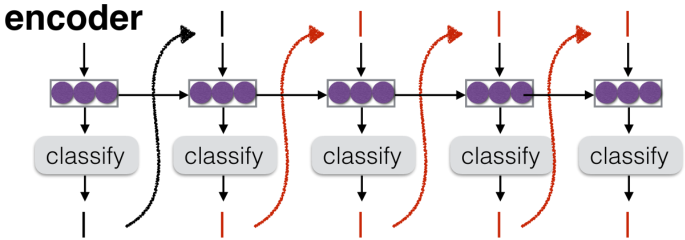
Overview

Types of prediction

- Two discrete classes (binary classification)
  - I hate this movie. → positive, **negative**
- Multiple discrete classes (multi-class classification)
  - I hate this movie. → positive, neutral, **negative**
- Real number(s) (regression)
  - I hate this movie. → Positivity: 0.1
- Everything else (structured prediction)
  - I hate this movie. → Ich hasse diesen Film.
  - I hate this movie. → [S [NP I] [VP [V hate] [NP [DT this] [NN movie]]] .]

Problem 1: Exposure bias

- Teacher forcing assumes correct previous labels
  - No guarantee of this at test time!



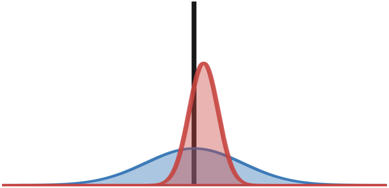
- **Exposure bias:** distribution of previous labels shifts from training to test time

Problem 2: Disregard of evaluation metrics

- Ultimately, we want good outputs
- Goodness of transalations can be measured with metrics, e.g., BLEU, METEOR
  - Maximum likelihood is *not* the ultimate goal
- Some mistakes are worse than others
  - This should be taken into account in training

Bias vs. variance

- Age-old trade-off in machine learning
- Compare two distributions
  - Black - value we are estimating
  - Blue - Low/no bias, high variance
  - Red - High bias, low variance



Varieties of structured prediction

- Models
  - RNN-based decoders
  - Convolutional/self-attentional decoders
  - Conditional random field w/ local factors
- Training algorithms
  - Maximum likelihood w/ teacher forcing
  - Sequence-level likelihood
  - Structured perceptron, structured hinge loss
  - Reinforcement learning, minimum risk training
  - Simpler remedies to exposure bias

Margin-Based Methods

Globally normalized models

- Normalization: sum of all outcome probabilities is 1
- **Locally normalized models:** each step in decoding is normalized separately

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

- **Globally normalized models:** (a.k.a. energy-based models) each sequence has score, normalized over *every possible sequence*

$$P(Y \mid X) = \frac{e^{S(X,Y)}}{\sum_{\tilde{Y} \in V*} e^{S(X,\tilde{Y})}} \tag{2}$$

Difficulties with global normalization

- Normalizing constant is called the *partition function*

$$Z(X) = \sum_{Y \in V*} e^{S(X,Y)} \tag{3}$$

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

Two methods for approximation

- **Direct sampling:**
  - Take k samples according the probability distribution
  - :-) Unbiased estimator: $k \to \infty$, gives the actual distr.
  - :-(High variance—a large k is needed in practice
- **Beam search:**
  - Search for the k-best hypotheses
  - :-(Biased estimator: systematic differences from the true distr.
  - :-) Low variance; high probabilities outputs mean a lower k is needed

Ditching normalization

- For inference, we often just want the *best hypothesis*
  - Division by a positive number is monotonic (preserves order)

$$\hat{Y} = \operatorname*{argmax}_{Y} P(Y \mid X) = \operatorname*{argmax}_{Y} S(Y \mid X) \tag{4}$$

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

Structured perceptron algorithm

- Extremely simple way of training (non-probabilistic) global models
  (5) Find the one-best prediction
  (6) If it is better than the correct prediction...
  (7) Adjust parameters to score the one-best lower, correct higher

$$\hat{Y} = \underset{\tilde{Y} \neq Y}{\operatorname{argmax}}\, S(\tilde{Y} \mid X; \theta) \tag{5}$$

$$\textbf{if } S(\hat{Y} \mid X; \theta) \geq S(Y \mid X; \theta) \tag{6}$$

$$\theta \leftarrow \theta + \alpha \left(\frac{\partial S(Y \mid X; \theta)}{\partial \theta} - \frac{\partial S(\hat{Y} \mid X; \theta)}{\partial \theta} \right) \tag{7}$$

Structured perceptron loss

- Stuctured perceptron can be expressed as a loss function

$$\ell_{\text{percept}}(X, Y) = \max(0, S(\hat{Y} \mid X; \theta) - S(Y \mid X; \theta)) \tag{8}$$

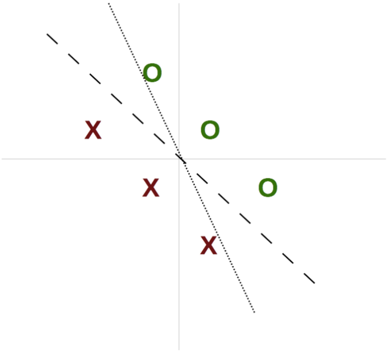
- The resulting gradient recovers the original algorithm
- Normal loss function $\rightarrow$ can be used in neural nets
- But! Requires finding the argmax in addition to the true candidate
  - **You must do prediction during training.**

Structured training and pre-training

- Neural nets have many parameters and a big output space—**training is hard**
- Trade-offs between training algorithms
  - Selecting just one negative example is inefficient
  - Teacher forcing efficiently updates all parameters, but suffers from exposure bias
- Practically, we can:
  1. Pre-train with teacher forcing
  2. Fine-tune with a less biased objective

Perceptron and uncertainty

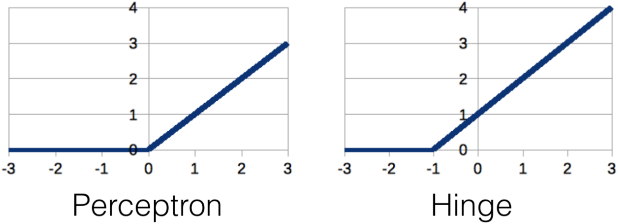
- Which is better: dotted or dashed?



- Both have zero perceptron loss!

Adding a "margin" with hinge loss

- Penalize when incorrect answer is within margin m



$$\ell_{\text{hinge}}(x, y; \theta) = \max(0, m + S(\hat{y} \mid x; \theta) - S(y \mid x; \theta)) \quad (9)$$

Cost-augmented hinge loss

- Some mistakes can be worse than others
  - VB $\rightarrow$ VBP is not so bad
  - VB $\rightarrow$ NN could be bad for downstream apps
- Cost-augmented hinge sets the margin equal to a function of y and $\hat{y}$.

$$\ell_{\text{ca-hinge}} = \max(0, \text{cost}(\hat{y}, y) + S(\hat{y} \mid x; \theta) - S(y \mid x; \theta)) \quad (10)$$

- Cost function has no dependence on θ. Is this good? Bad?

Cost over sequences

- $\text{cost}(\hat{Y}, Y)$ can be basically anything!
- **Zero-one loss:** 1 if sequences differ, 0 otherwise
- **Hamming loss:** 1 for every differing element ($|\hat{Y}| = |Y|$)
- Other losses: edit distance, 1-BLEU, etc.

Structured hinge loss

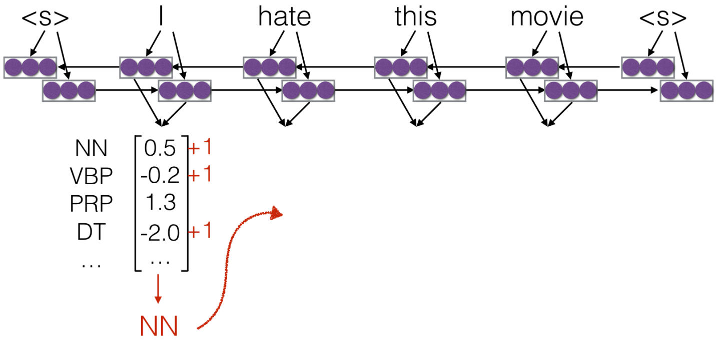
- Hinge loss over sequence with largest margin violation

$$\hat{Y} = \operatorname*{argmax}_{\tilde{Y} \neq Y} \mathrm{cost}(\tilde{Y}, Y) + S(\tilde{Y} \mid X; \theta) \tag{11}$$

- **Problem:** How do we find the argmax above?
- **Solution:** Sometimes we can incorporate the cost in search.

Cost-augmented decoding for Hamming loss

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



Reinforcement Learning

Basics of reinforcement learning (RL)

- Imagine a robot (the agent) that lives in a little world, at each timestep...
  - Environment (the world) has a certain state
  - Agent observes the state and takes an action
  - Environment transitions to a new state based on the action
  - Agent receives reward based on the action and environment state
- More formally
  - State space: S
  - Action space: A
  - Policy (action taking): $\pi : S \to A$
  - Reward function: $R : S \times A \to \mathbb{R}$
  - Transition function: $p(s' \mid s, a)$

Examples of RL

- Pong
  - S: Pixels of the display
  - A: Move up or down
  - R: Win or lose game
- Self-driving car
  - S: Location of cars, street signs
  - A: Steering, gas, brake
  - R: Not crashing, obeying traffic laws
- Image classification
  - S: Image to be classified
  - A: Probability distribution over classes (e.g., cat, dog, emu)
  - R: Probability of correct class

Why RL in NLP?

- Typical reinforcement learning scenarios do appear; e.g., dialog agents, vision-language navigation
- Latent variable selection where the selection process is non-differentiable
- Situations with a sequence-level error function such as BLEU

Supervised maximum likelihood estimation (MLE)

- Correct actions are known at training time

$$\ell_{\text{super}}(Y, X) = -\log P(Y \mid X) \tag{12}$$

- Think of $S = X$, $A = Y$, and $R = -\ell_{\text{super}}$
  - Supervised learning $\subset$ reinforcement learning!
- In RL, this would be called *behavioral cloning*, a subset of *imitation learning*

Self-training

- Sample or argmax according to the current model

$$\hat{Y} \sim P(Y \mid X) \quad \text{or} \quad \hat{Y} = \operatorname*{argmax}_{Y} P(Y \mid X) \tag{13}$$

- Use this sample as the label in MLE

$$\ell_{\text{self}}(X) = -\log P(\hat{Y} \mid X) \tag{14}$$

- No labeled data needed! But is this a good idea?
  - Co-training: only use labels where multiple models agree (Blum and Mitchell 1998)
  - Noising the input, to match output (He et al. 2020)

Policy Gradient and REINFORCE

- Add a term that scales the loss by the reward

$$\ell_{\text{pg}} = R(\hat{Y}, Y) \cdot \ell_{\text{self}}(X) = -R(\hat{Y}, Y) \log P(\hat{Y} \mid X) \qquad (15)$$

- $R(\hat{Y}, Y)$ can be an arbitrary function
- We don't need to know $P(Y \mid X)$

Credit assignment for rewards

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

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 0 & +1 & 0 & -0.5 & +1 & +1.5 \end{array}$$

- Worst scenario, only at end of episode:

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{array}$$

Problems w/ reinforcement learning

- Like other sampling-based methods, RL 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

Minimum risk training

- Shen et al. (2016) propose to minimize expected risk
  - Risk (Δ) can be -BLEU, TER, -NIST (smoothed, sentence-level)

$$L_{\mathrm{MRT}}(Y, X) = \frac{1}{|\mathcal{S}(X)|} \sum_{\hat{Y} \in \mathcal{S}(X)} \Delta(\hat{Y}, Y) P(\hat{Y} \mid X; \theta) \qquad (16)$$

- $\mathcal{S}(X)$ generates the set of candidate translations; $\mathcal{S}(X) = \ldots$
  - A single sample (vanilla REINFORCE) $\rightarrow$ unstable :-(
  - All candidate translations (expected risk) $\rightarrow$ intractable :-(
  - 100 samples from the model $\rightarrow$ slow but stable :-)

Adding a baseline

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

Input	Reward	Baseline	$b - r$
"This is an easy sentence."	0.8	0.9	-0.1
"Buffalo buffalo Buffalo."	0.3	0.1	0.2

- Weighting the likelihood by $r - b$ to reflect when we did **better or worse than expected**

$$\ell_{\text{baseline}}(Y, X) = -(R(\hat{Y}, Y) - B(\hat{Y})) \log P(\hat{Y} \mid X) \quad (17)$$

Calculating baselines

- Choice of baseline is arbitrary
- Option 1: predict final reward using linear layer 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

- Each sample will be high-variance, so we sample many different examples with the same policy
- Increase the number of examples (rollouts) done before an update to stabilize
- We can also save previous rollouts and reuse them to update parameters (experience replay, Lin 1993)
  - Caution! Using rollouts calculated by old policies can also make stability worse

Warm start

- Start training with maximum likelihood, then switch over to REINFORCE
- Works only in the scenarios where we can fall back on MLE
- MIXER (Ranzato et al. 2016) anneal from MLE to RL objective

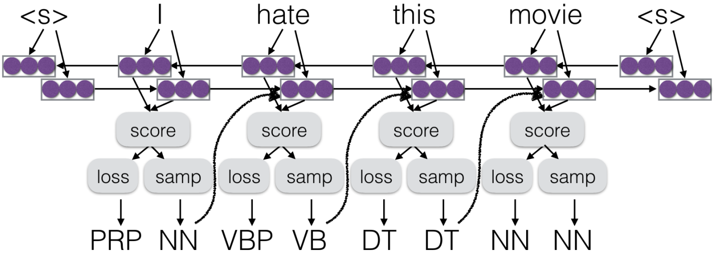
Remedying Exposure Bias

What's wrong with these hinge loss and RL?

- Hinge loss can work, but...
  - Considers few hypotheses, thus *unstable*
  - Requires decoding, thus *slow*
- Reinforcement learning
  - Has similar issues as hinge loss
  - Credit assignment problem means gradient is noisy
- Hinge/RL isn't bad—maximum likelihood is great baseline!
  - Full differentiable $\rightarrow$ find the contribution of each parameter
  - Good option for pre-training
- How do we address exposure bias while still using MLE?

Solution 1: Sample mistakes in training (Ross et al. 2010)

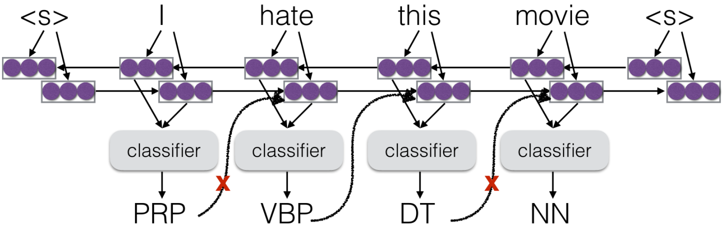
- DAgger, a.k.a. *scheduled sampling*, randomly samples wrong decisions and feeds them in



- Start with no mistakes; gradually introduce them with annealing

Solution 2: Drop out inputs

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



- Decrease dependence 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/ MLE

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

- Sampling probability proportional to goodness of output

Review

Topics covered

- Structured margin-based methods
- Reinforcement learning and minimum risk training
- Simpler remedies to exposure bias

Takeaways

- NLP presents unique problems within machine learning
- Bias-variance trade-off
- No free lunch!
  - But you can get more bang for your buck.
- Differentiable objectives are often preferrable
  - But they are not always preferrable