Supervised Learning with Black Box Loss Functions

David S. Rosenberg

NYU: CDS

March 31, 2021

Contents

Black box and non-differentiable losses

Supervised learning

- Input space $\mathcal{X}$; Label space $\mathcal{Y}$
- Hypothesis space of functions $x \mapsto f_\theta(x) \in \mathcal{Y}$
- For $(X, Y) \sim P$, loss function $\ell(f_\theta(X), Y)$ tells us how we did.
- For gradient-based learning methods, we compute

$$\nabla_\theta \ell(f_\theta(X), Y) = \frac{\partial \ell}{\partial f_\theta(X)} \ell(f_\theta(X), Y) \nabla_\theta f_\theta(X)$$

and iteratively update θ as $\theta \leftarrow \theta - \eta \nabla_\theta \ell(f_\theta(X), Y)$

What about non-differentiable or black-box losses?

- For gradient-based learning methods, we compute

$$\nabla_\theta \ell(f_\theta(X), Y) = \frac{\partial \ell}{\partial f_\theta(X)} \ell(f_\theta(X), Y) \nabla_\theta f_\theta(X)$$

- What if the loss function ℓ is not differentiable? not continuous?
- What if the loss function ℓ is a black box?
- Examples:
  - 0/1 loss for classification
  - BLEU score – used in machine translation for scoring a candidate sentence against a reference set of translations
  - ROUGE metrics – used in automatic summarization for scoring a summaries against a reference set of summaries
  - a human annotator evaluating predictions made by a model

The essence of the issue

- Gradient methods are iterative and "local".
- Suppose our current model is f_θ.
- By examining $\ell(f_\theta(X), Y)$ in a small neighborhood of θ, we can figure out how to make a small change to θ to improve performance.
- We can do this conveniently with the gradient.
- But with discrete label spaces
  - $\theta \mapsto \ell(f_\theta(X), Y)$ will be piecewise constant, and so.
  - we get no information about how to change θ by looking in a small neighborhood of θ.
- So the situation can be much worse than just "non-differentiable".

Randomization and policy gradient

Using randomness to smooth our objective

- Let's move from deterministic predictions $f_\theta(x) \in \mathcal{Y}$
- to randomized actions $A \in \mathcal{Y}$ drawn from $\pi_\theta(a \mid x)$.
- Let's consider the expected loss as our performance measure:

$$\mathbb{E}_{A \sim \pi_\theta(a|X)} \ell(A, Y).$$

- We want a conditional probability model on actions that gets small expected loss.
- How has this helped?

Expected loss for randomized actions

- Expected loss is

$$\mathbb{E}_{A \sim \pi_\theta(a|X)} \ell(A, Y) \quad = \quad \sum_{a \in \mathcal{A}} \pi_\theta(a \mid X) \ell(a, Y).$$

- Before, we were evaluating the loss at $\ell(f_\theta(X), Y)$, where
  - $f_\theta(X)$ may change discontinuously as a function of θ
  - Can't figure out what direction to move θ from a local neighborhood of θ
- For expected loss, we're always evaluating the loss on all possible actions
  - As θ varies, we're changing the relative weights on losses from each action
- Expected loss changes smoothly as θ changes (so long as $\pi_\theta(a \mid X)$ changes smoothly).

Gradient of expected loss

- Gradient of expected loss:

$$\begin{aligned}\nabla_\theta\left[\mathbb{E}_{A\sim\pi_\theta(y|X)}\ell(A,Y)\right] &= \nabla_\theta\left[\sum_{a\in\mathcal{A}}\pi_\theta(a\mid X)\ell(a,Y)\right]\\ &= \sum_{a\in\mathcal{A}}\ell(a,Y)\,\nabla_\theta\pi_\theta(a\mid X)\end{aligned}$$

- We can compute the gradient of expected loss so long as the CPM is differentiable.
- We don't need to differentiate w.r.t. the loss.
- What if the action space $\mathcal{A}$ is too large to sum over?

Clever trick again

- We have

$$\begin{aligned}\nabla_\theta\left[\mathbb{E}_{A\sim\pi_\theta(a|X)}\ell(A,Y)\right] &= \sum_{a\in\mathcal{A}}\ell(a,Y)\,\nabla_\theta\pi_\theta(a\mid X)\\ &= \sum_{a\in\mathcal{A}}\ell(a,Y)\,\pi_\theta(a\mid X)\nabla_\theta\log\pi_\theta(a\mid X)\\ &= \mathbb{E}_{A\sim\pi_\theta(a|X)}\ell(A,Y)\,\nabla_\theta\log\pi_\theta(A\mid X)\end{aligned}$$

- Now we can use Monte Carlo to estimate this.
- If we have samples $A_1,\ldots,A_n$ from $\pi_\theta(a\mid X)$, our action-generating distribution, then

$$\frac{1}{n}\sum_{i=1}^{n}\ell(A_i,Y)\,\nabla_\theta\log\pi_\theta(A_i\mid X)$$

- is an unbiased estimate of $\nabla_\theta\left[\mathbb{E}_{A\sim\pi_\theta(a|X)}\ell(A,Y)\right]$.
- Look familiar?

Comparison to policy gradient for contextual bandits

- We have rederived policy gradient for contextual bandits.
- Are there any differences?
- We get the ground truth label Y.
- If we also have access to the loss function (at least as a black box),
  - we can feed in lots of possible actions for the given X.
- This will give us a much better estimate of $\nabla_\theta \left[\mathbb{E}_{A \sim \pi_\theta(a|X)} \ell(A, Y)\right]$.
- Compared to policy gradient, where we observe the reward for only a single action.
- We can also directly apply policy gradient, using just a single action to estimate the gradient.

Comparison to maximum likelihood

- We're using conditional probability models.
- And we observe ground truth labels.
- Why not just fit the CPMs with maximum likelihood?
- That's typical, but it ignores our loss function!
- Perhaps some errors are worse than others.
- Maximum likelihood focuses on putting as much weight as possible on the correct label.
- But maybe the loss function is fairly indifferent between multiple labels.
    - e.g. There may be multiple translations that are essentially equivalent, but our "label" is just one of them.
    - Maximum likelihood tries to put all the weight on the "correct" one.
    - Maximizing w.r.t. a loss function that's indifferent to these variations won't do this.
- On the negative side, policy gradient estimates are very high variance, and
    - optimization is very slow.

Policy gradient for sequence prediction

Application: Sequence-to-Sequence Models

- Consider machine translation.
- e.g. Conditioned on sentence in English, produce a distribution on sentences in French.
- Model is $\pi_\theta(a \mid x)$, where x is an English sentence and a is a French sentence.
- Typically trained on ground truth pairs (X_i, Y_i) using maximum likelihood:

$$\theta^* = \arg\max_\theta \prod_{i=1}^{n} \pi_\theta(Y_i \mid X_i).$$

- Seems reasonable...
- But how do we actually measure performance for machine translation?

Application: Sequence-to-Sequence Models

- Suppose we are assessing performance of our MT model on a test set.
- We get input X.
- We use our current model $\pi_\theta(\cdot \mid X)$ and produce a sequence A.
  - (e.g. by sampling or beam search or something)
- Suppose A is a perfect translation of X, but it's different from the ground truth Y.
- We'd like to give credit for this translation.
- I don't think there's a great way to do this in an automated way.

But there is BLEU score

- A frequent measure of translation quality is BLEU score.
- Let's not discuss the details of BLEU score.
- For our purposes, sufficient to know that
  - BLEU takes a proposed translation and a ground truth and gives a numerical score
- BLEU score is computed by an algorithm and is **not differentiable.**
- Perhaps it would make sense to train a model to optimize directly for BLEU score?

Exposure Bias

- Sequence models are frequently **autoregressive**.
- We condition on previously predicted tokens to predict the next token in a sequence.
- During MLE training, we're always conditioning on the gold labels in Y.
- During test, we're conditioning on our own predicted labels.
- **Our model never trains using its own predictions as input.**

Exposure Bias

- This is a known issue with maximum likelihood training of sequence models.
- There is a family of approaches called "learning to search" that address this issue.
- e.g. SEARN, DAgger, AggreVaTe, LOLS, etc.
- We can use policy gradient as well.
- Sampling from $\pi_\theta(\cdot \mid X)$ doesn't involve the ground truth sequence Y at all.
- We don't even have the ground truth label to use, except as part of the reward function.

Usually we pre-train with maximum likelihood

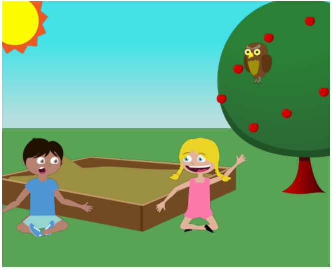
- Suppose we want to train a sequence-to-sequence model
  - with BLEU score as reward.
- Policy gradient is sufficient for this task.
- In practice, we usually pretrain our model with maximum likelihood.
  - It's much faster than policy gradient.
- Then switch to policy gradient to optimize to our particular loss/reward.

Self-Critical Baseline

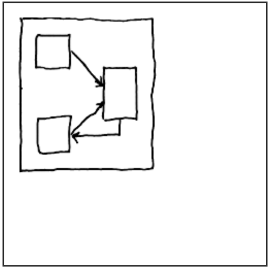
- When we have access
  - to the loss function $\ell(a, y)$ and
  - the ground truth labels $Y_1, Y_2, \ldots,$
- there's another clever way to set a baseline:
  - Find (or approximate) the action that is optimal under our policy: $A_t^* \approx \arg\max_a \pi_{\theta_t}(a|X_t)$
  - Use the loss $\ell(A_t^*, Y_t)$ as a baseline.
- If the current action performs better than the action our policy says is best, then we should make the current action more likely.
- If it performs worse than what the policy says is best, let's make it less likely.

Image to Sequence

Image to Sequence Problems[1]



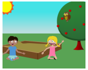
<object>
<supercategory>C-1</supercategory>
<category>CS-3<\category>
<x-coordinate>120</x-coordinate>
<y-coordinate>240</y-coordinate>
<depth>1</depth>
<flip>0</flip>
</object>
<object>....



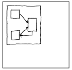
<object>
<category>Rectangle<\category>
<x1-coordinate>7</x1-coordinate>
<y1-coordinate>1</y1-coordinate>
<x2-coordinate>11</x2-coordinate>
<y2-coordinate>16</y2-coordinate>
</object>
<object>....

[1] Results and images for this section are taken from [PRMB21]

Image to Sequence Problems[1]



```
<object>
  <supercategory>C-1</supercategory>
  <category>CS-3<\category>
  <x-coordinate>120</x-coordinate>
  <y-coordinate>240</y-coordinate>
  <depth>1</depth>
  <flip>0</flip>
</object>
<object>....
```



```
<object>
  <category>Rectangle<\category>
  <x1-coordinate>7</x1-coordinate>
  <y1-coordinate>1</y1-coordinate>
  <x2-coordinate>11</x2-coordinate>
  <y2-coordinate>16</y2-coordinate>
</object>
<object>....
```

[1]Results and images for this section are taken from [PRMB21]

- In these image to sequence tasks, there is a domain-specific language (DSL) for specifying images. For the example on the left, the image is rendered directly from the specification. For the example on the right, the image is rendered from the specification, and then a person traced the image, which accounts for the waviness of the lines. The objective is to take the image as input and produce the appropriate DSL specification.

How to Evaluate?

- One obvious idea is to re-render and check for an exact match.
- This is a very challenging metric.
- We only get positive feedback when we get the image exactly correct.
- Will take a **very long time** to learn this way (at least starting from scratch).
- Doesn't work for hand-drawn shapes

Evaluation metrics

- Two specifications can be very different, yet render to very similar things.
  - e.g. by reordering objects
- Two images may look very different (e.g. at the pixel level), but have similar specifications
  - e.g. by changing a color
- We can evaluate performance in **image space** and in **specification space**.

Image Space Measure

- We can measure performance in image space with

$$d_{img} = \|I - \Psi(I^R)\|_2^2,$$

  where I is the original image vector and I^R is the rendering of the predicted image.
- For the noisy shapes dataset, Ψ is a Gaussian blurring function.
- For the abstract scene dataset, Ψ is identity function.
- Why isn't this differentiable?
- Computing I^R uses a graphics renderer...
- (There are differentiable renderers now... but that's another story.)

Specification Space Measure: IOU Reward

- Our specifications break down into "objects".
- We can look for exact matches between prediction and ground truth at the object level.
- For numeric attributes, we divide range into 20 bins of equal size
  - consider it a match if the bin is correct
- Can summarize matches with precision, recall, F1, etc.
- A common summary in this scenario is **intersection-over-union** (IOU)....

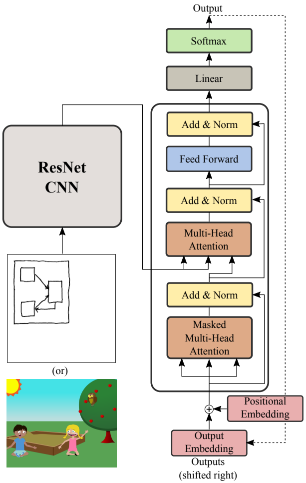
Intersection over Union

- Let $\{o_i\}_{i=1}^m$ and $\{o_j^*\}_{j=1}^n$ represent the objects in predicted and ground-truth specifications, respectively.
- Then the IOU reward is defined as follows:

$$r_{iou} = \frac{\text{count}(\{o_i\}_{i=1}^m \cap \{o_j^*\}_{j=1}^n)}{\text{count}(\{o_i\}_{i=1}^m \cup \{o_j^*\}_{j=1}^n)}$$

- Roughly speaking, IOU gives credit for predicting objects that exactly match objects in the ground truth
- Penalizes both for predicting objects that do not match ground truth objects and for failing to predict objects that are part of the ground truth.

Model: ResNet to Transformer Decoder



Results: cross-entropy Loss (i.e. maximum likelihood)

Model	Recons. Error	IOU
Cross-Entropy Loss		
Image2LSTM+atten.	15.70	32.06
Image2Transformer	10.92	58.54

- reconstruction error corresponds to the image distance
- average error across test set

Results: policy gradient

Model	Recons. Error	IOU
Cross-Entropy Loss		
Image2LSTM+atten.	15.70	32.06
Image2Transformer	10.92	58.54
Image2Transformer with Reinforce Loss		
IOU Reward	10.50	61.29
Recons. Reward	**9.99**	62.44
IOU + Recons.	10.04	**62.45**

References

Resources

- The idea of the self-critical baseline comes from [RMM+17].

References I

[PRMB21] Ramakanth Pasunuru, David S. Rosenberg, Gideon Mann, and Mohit Bansal, *Dual reinforcement-based specification generation for image de-rendering*, Proceedings of the Workshop on Scientific Document Understanding co-located with 35th AAAI Conference on Artificial Inteligence, SDU@AAAI 2021, Virtual Event, February 9, 2021 (Amir Pouran Ben Veyseh, Franck Dernoncourt, Thien Huu Nguyen, Walter Chang, and Leo Anthony Celi, eds.), CEUR Workshop Proceedings, vol. 2831, CEUR-WS.org, 2021.

[RMM+17] Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel, *Self-critical sequence training for image captioning*, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 7 2017, p. nil.