8 Regularization

Throughout the RLHF optimization, many regularization steps are used to prevent over-optimization of the reward model. Over-optimization in these contexts looks like models that output nonsensical text. Some examples of optimization "off the rails" are that models can output followable math reasoning with extremely incorrect answers, repeated text, switching languages, or excessive special characters.

The most popular variant, used in most RLHF implementations at the time of writing, is a KL Distance from the current policy to a reference policy across the generated samples. Many other regularization techniques have emerged in the literature to then disappear in the next model iteration in that line of research. That is to say that regularization outside the core KL distance from generations is often used to stabilize experimental setups that can then be simplified in the next generations. Still, it is important to understand tools to constrain optimization in RLHF.

The general formulation, when used in an RLHF framework with a reward model, r_θ is as follows:

$$r = r_\theta - \lambda r_{\text{reg.}} \tag{18}$$

With the reference implementation being:

$$r = r_\theta - \lambda_{\text{KL}} \mathcal{D}_{\text{KL}} \left(\pi^{\text{RL}}(y \mid x) \,\|\, \pi^{\text{Ref.}}(y \mid x) \right) \tag{19}$$

8.1 KL Distances in RL Optimization

For mathematical definitions, see Chapter 5 on Problem Setup. Recall that KL distance is defined as follows:

$$D_{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{20}$$

In RLHF, the two distributions of interest are often the distribution of the new model version, say $P(x)$, and a distribution of the reference policy, say $Q(x)$.

8.1.1 Reference Model to Generations

The most common implementation of KL penalities are by comparing the distance between the generated tokens during training to a static reference model. The intuition is that the model you're training from has a style that you would like to stay close to. This reference model is most often the instruction tuned model, but can also be a previous RL checkpoint. With simple substitution, the model we are sampling from becomes $P^{\text{RL}}(x)$ and $P^{\text{Ref.}}(x)$, shown above in eq. 19. Such KL distance was first applied to dialogue agents well before the popularity of large language models [114], yet KL control was quickly established as a core technique for fine-tuning pretrained models [115].

8.1.2 Implementation Example

In practice, the implementation of KL distance is often approximated [116], making the implementation far simpler. With the above definition, the summation of KL can be converted to an expectation when sampling directly from the distribution $P(X)$. In this case, the distribution $P(X)$ is the generative distribution of the model currently being trained (i.e. not the reference model). Then, the computation for KL distance changes to the following:

$$D_{\mathrm{KL}}(P \,||\, Q) = \mathbb{E}_{x \sim P} \left[\log P(x) - \log Q(x) \right]. \tag{21}$$

This mode is far simpler to implement, particularly when dealing directly with log probabilities used frequently in language model training.

```
import torch.nn.functional as F
# Step 1: Generate tokens using the trained model's policy
generated_tokens = model.generate(inputs)

# Step 2: Get logits for both models using the generated tokens as
    context
logits = model.forward(inputs) # technically redundant
ref_logits = ref_model.forward(inputs)
logprobs = convert_to_logpbs(logits) # softmax and normalize
ref_logprobs = convert_to_logpbs(ref_logits)

kl_approx = logprob - ref_logprob
kl_full = F.kl_div(ref_logprob, logprob) # alternate computation
```

Some example implementations include TRL and Hamish Ivison's Jax Code

8.2 Pretraining Gradients

Another way of viewing regularization is that you may have a *dataset* that you want the model to remain close to, as done in InstructGPT [3] ''in order to fix the performance regressions on public NLP datasets''. To implement this, they modify the training objective for RLHF. Taking eq. 18, we can transform this into an objective function to optimize by sampling from the RL policy model, completions y from prompts x, which yields:

$$\text{objective}(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}_{\pi_\theta^{\mathrm{RL}}}} \left[r_\theta(x, y) - \lambda r_{\text{reg.}} \right] \tag{22}$$

Then, we can add an additional reward for higher probabilities on pretraining accuracy:

$$\text{objective}(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}_{\pi_\theta^{\mathrm{RL}}}} \left[r_\theta(x, y) - \lambda r_{\text{reg.}} \right] + \gamma \mathbb{E}_{x \sim \mathcal{D}_{\text{pretrain}}} \left[\log(\pi_\theta^{\mathrm{RL}}(x)) \right] \tag{23}$$

Recent work proposed using a negative log likelihood term to balance the optimization of Direct Preference Optimization (DPO) [117]. Given the pairwise nature of the DPO loss, the same loss modification can be made to reward model training, constraining the model to predict accurate text (rumors from laboratories that did not publish the work).

The optimization follows as a modification to DPO.

$$\mathcal{L}_{\text{DPO+NLL}} = \mathcal{L}_{\text{DPO}}(c_i^w, y_i^w, c_i^l, y_i^l \mid x_i) + \alpha \mathcal{L}_{\text{NLL}}(c_i^w, y_i^w \mid x_i) \tag{24}$$

$$= -\log \sigma \left(\beta \log \frac{M_\theta(c_i^w, y_i^w \mid x_i)}{M_t(c_i^w, y_i^w \mid x_i)} - \beta \log \frac{M_\theta(c_i^l, y_i^l \mid x_i)}{M_t(c_i^l, y_i^l \mid x_i)} \right) - \alpha \frac{\log M_\theta(c_i^w, y_i^w \mid x_i)}{|c_i^w| + |y_i^w|}. \quad (25)$$

8.3 Other Regularization

Controlling the optimization is less well defined in other parts of the RLHF stack. Most reward models have no regularization beyond the standard contrastive loss function. Direct Alignment Algorithms handle regularization to KL distances differently, through the β parameter (see the chapter on Direct Alignment).

Llama 2 proposed a margin loss for reward model training [43]:

$$\mathcal{L}(\theta) = -\left[\log\left(\sigma\left(r_\theta(x, y_w) - r_\theta(x, y_l) - m(r)\right)\right)\right] \quad (26)$$

Where $m(r)$ is the numerical difference in delta between the ratings of two annotators. This is either achieved by having annotators rate the outputs on a numerical scale or by using a quantified ranking method, such as Likert scales.

Reward margins have been used heavily in the direct alignment literature, such as Reward weighted DPO, ''Reward-aware Preference Optimization'' (RPO), which integrates reward model scores into the update rule following a DPO loss [24], or REBEL [118] that has a reward delta weighting in a regression-loss formulation.