

Play Around with Distillation, Part 1: Forward vs. Reverse KL, On vs. Off Policy, and Hard vs. Soft!

Haryo Akbarianto Wibowo ¹

¹A Humble Researcher

haryo.akbarianto@gmail.com

ABSTRACT

In this era, you must have heard these terms: distillation and on/off-policy in Large Language Model (LLM) training, right? In simple words, there are two models, a teacher and a student. The teacher gives guidance in some form (e.g., probabilities, labels) to the student, and the data used for this comes either from the student's own outputs (on-policy) or from somewhere else, like the teacher's outputs or a dataset (off-policy). That's it, if you only care about the definition (which I simplified, lol). But are you curious to dive deeper? Like, what the loss function actually is, why it works in theory, and how it behaves? Here, I will show you all of that in a way that is (hopefully) beginner friendly, yet still informative for advanced people. Instead of just shouting "LLM DISTILL WORKS, IT'S AWESOME", I will use some toy problems. These toy problems are about matching distributions, which is the main part of distillation anyway. As you can already see, the writing is casual, since this is meant to be a casual blog rather than a hardcore one. Still, I will make the content as informative as possible :).

Keywords Knowledge Distillation, Forward and Reverse KL, On- and Off-Policy, Soft and Hard Targets, Toy Problems

1 Introduction

First of all, as mentioned above, the language in this blog will be casual, but the information I give you will be legit and informative, hopefully! Expect the content to be a mix of my personal ramblings, random unfunny jokes, and my personal experiences :), fused with academic writing. It's been a long time since I last wrote a blog (those were just some fun experiments, like training a model to write recipes), and I'm going to do a fun thing here too.^a

Motivation. Alright, let's start with the motivation. I see lots of people either downplaying knowledge distillation (KD) or just taking it for granted. The first group is

^a Also, some psychology/neuro stuff that I read says that sharing what you learn by writing it down tidily helps you learn better. So, I encourage people to write blogs too :D.



Figure 1 • Us, reading about LLM distillation for the first time. By the end of this blog, every word in this picture should make sense. I promise!

people who don't really keep up with the distillation world, while the second group is people jumping on LLM-world techniques without caring how they work. However, some people do want to know the basic idea behind KD. Since I am in this field and I still keep updating my knowledge^b, I would like to share my insights about it. I will especially focus on the intuition behind distillation, such as its behavior, its theoretical foundation, and so on, while also giving you some practical problems. Hopefully, those who want to understand the KD problem better or want to enter this field will love it.

Structure. This blog will cover KD, on/off-policy, and soft and hard distillation. I will start with the basics of each, and then move on to a toy problem for vanilla (non-sequence) distillation. The sequence version will come in a separate blog post (Part 2), so this one doesn't get too long. For the toy problem, I will not use any modern language models (LSTM included in this list). Instead, I will use a very simple statistical model (a single bell curve over the classes), as this will help you understand better. Sure, there are lots of works with more advanced techniques, but as far as I know, they mostly tinker around with the policy and the objective function. The foundation should be the one I explain in this blog.

Despite being in the KD field, I am, honestly, still learning about it (it keeps evolving!!). I've been here since 2018, back when distillation was just a way to compress models (e.g., Sanh et al., 2019). Nowadays, with the advent of LLMs, people want to make small models work, and distillation is the main key (Gemma Team, 2024; Guo et al., 2025). RL (Reinforcement Learning) for language models also uses KD as a reward (Agarwal et al., 2024; Lu and Thinking Machines Lab, 2025), since the common rewards (e.g., a reward model (Ouyang et al., 2022) or a verifiable reward (Guo et al., 2025)) are sparse, hence not as informative as KD. I will not dive

^b fyi, this is a pain in the current agentic era. People publish too fast..

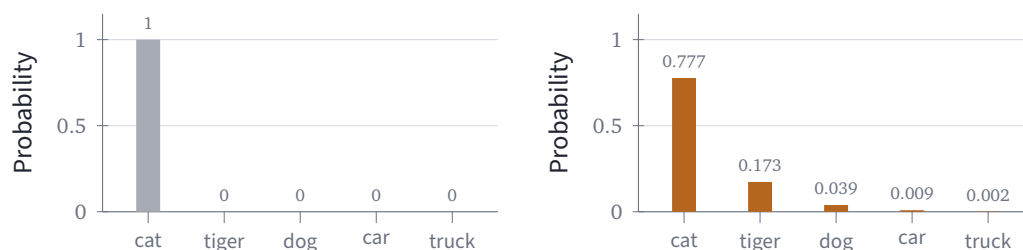
further into it here, but maybe I will in the future, hopefully. Anyway, what I was going to say is: KD is in an exciting era, though there are definitely some problems still unsolved. Mainly, the data used for distillation, and the right objective function that makes the knowledge transfer go the way we want it to.^c

^c Sometimes, KD is also used to change a model's behavior, not only to make it better.

2 What Is Knowledge Distillation?

Core KD Explanation. Let's start with the definition. The term KD was coined by Hinton et al. (2015) (the idea of making a small model copy a big one is a bit older, see Buciluă et al., 2006), who call the extra information from the teacher 'dark knowledge' (sounds cool, huh?). Basically, KD is a method to transfer knowledge from a larger, better model (the teacher) to a smaller model (the student).^d The dark knowledge itself comes from the output probabilities, which give more information than the hard label (see Figure 2). Because of this, the student gets a more precise target, and the training signal has lower variance than training with hard labels only.

^d Nowadays, the student can be as big as the teacher, or even bigger! E.g., a same-size student can beat its teacher (Furlanello et al., 2018), and KD can work as a regularizer that stops a big student from overfitting (Yuan et al., 2020).



(a) Hard label (ground truth).

(b) Teacher's output (soft target).

Figure 2 • Where the 'dark knowledge' lives. Given a picture of a cat, the hard label (a) only says "cat". The teacher (b) says "it's a cat, but it kinda looks like a tiger, a bit like a dog, and it's definitely not a car or a truck". All that extra info about the *wrong* classes is what the student gets from KD.

TL;DR In my own words: you have a teacher model and a student model. Given a task (e.g., "generate a poem") and a proposed answer (it can come from the student's output, the teacher's output, the training data, and so on), let the teacher give its answer in some form (normally logits), and let the student give its answer too. Then, make the student use the teacher as an anchor for how it should answer the task.

KD Objective Function. To understand KD, we need to look at its objective function, since every training in Machine Learning is about optimizing one (people sometimes call it the loss function). Let's start from the intuition: as mentioned above, we want the student's distribution to be anchored to, or mimic, the teacher's. To keep things standard, we follow Hinton et al. (2015) and start by defining the distributions we get

from a neural network. Let z^t and z^s denote the logits of the teacher and the student, respectively. Logits are converted into probabilities with a softmax at temperature T :

$$\text{softmax}(z/T)_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}. \quad (1)$$

T is useful for controlling how much knowledge is transferred to the student. If you want a sharp distribution, use a lower T (close to 0 means it becomes one-hot). A higher T flattens the distribution, so classes with close logits end up with similar probabilities, and the tiny probabilities (the dark knowledge!) become more visible. But if T is too high, everything looks almost the same, and you are mostly transferring noise. See Figure 3 for an illustration. The choice of T depends on your use case. Now, let's define the distributions:

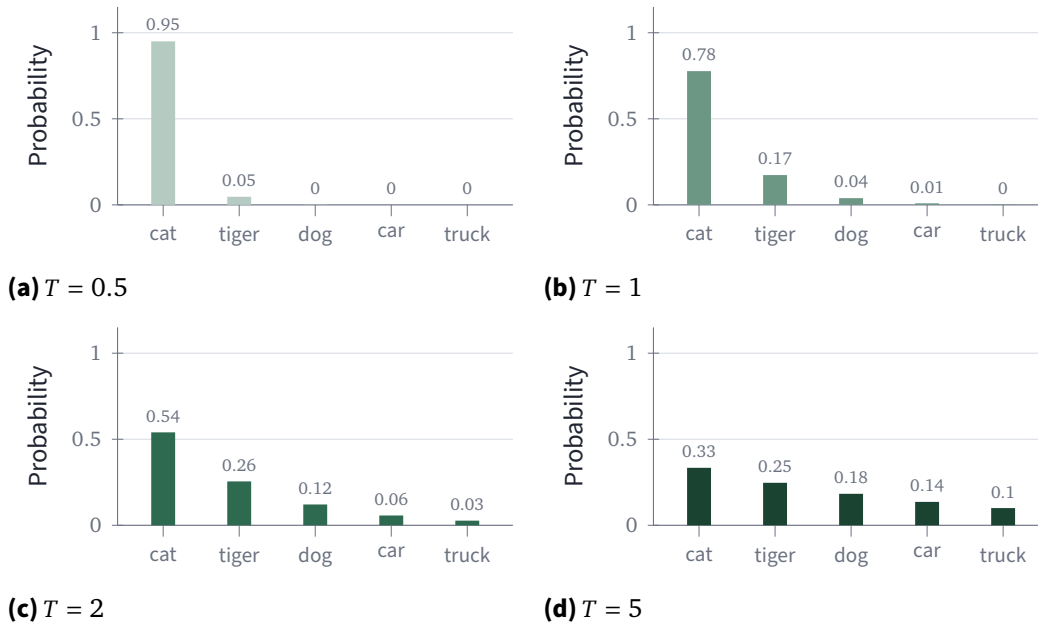


Figure 3 • The teacher's distribution for the same cat picture (same logits, $z^t = [5, 3.5, 2, 0.5, -1]$), only with different temperatures. With a low T (a), it's almost a hard label again. As T goes up (b, c), the small probabilities (tiger, dog, ...) become easier to see. Push T too high, though (d), and even "truck" starts to look almost as likely as "cat".

$$p = \text{softmax}\left(\frac{z^t}{T}\right), \quad q_1^s = \text{softmax}\left(\frac{z^s}{T}\right), \quad q_2^s = \text{softmax}(z^s). \quad (2)$$

Here p is the teacher's 'softened' distribution (the *soft targets*), q_1^s is the student's distribution softened with the same temperature, and q_2^s is the student's ordinary distribution at $T = 1$. Both q_1^s and q_2^s come from the same student logits; they differ only in temperature.

The student parameters θ are trained to minimize a weighted combination of two terms:

$$\mathcal{L}(\theta) = \underbrace{\alpha T^2 \text{KL}(p \parallel q_1^s)}_{\text{KL term}} + (1 - \alpha) \underbrace{\text{CE}(y^{gt}, q_2^s)}_{\text{Anchor to Ground Truth}}, \quad \theta^* = \arg \min_{\theta} \mathcal{L}(\theta), \quad (3)$$

where

$$\text{KL}(p \parallel q_1^s) = \sum_i p_i \log \frac{p_i}{q_{1,i}^s}, \quad \text{CE}(y^{gt}, q_2^s) = - \sum_i y_i^{gt} \log q_{2,i}^s, \quad (4)$$

and y^{gt} is the one-hot ground-truth label. The KL term pulls the student’s softened distribution toward the teacher’s; the cross-entropy term keeps the student anchored to the true labels. $\alpha \in [0, 1]$ balances the two. The factor T^2 keeps the two terms on a comparable scale.^e In modern LLM distillation, $T = 1$ is the common choice, in which case $q_1^s = q_2^s$ and the T^2 disappears.

Note that in some cases, we don’t have any ground truth and can only rely on the teacher, so the second term is dropped ($\alpha = 1$). This is what people do when they use an AI model to ‘distill’ another one: there is no ground truth, so the focus is only on the teacher.

Hard Distillation. The setting above (Eq. (3)), where you have access to the teacher’s logits or probability distribution, is called soft distillation (mimicking soft targets). In reality, though, we often don’t get them. When people publish data generated by an LLM, they usually don’t include the probability distribution of each token due to storage issues.^f The same goes for using an API, which normally doesn’t give them to you (this is a current hot topic in the AI world). Training on this kind of data is still called distillation, since the LLM that generated it acts as the teacher, even though we only have hard targets. This is what we call hard distillation (Kim and Rush, 2016). A famous example is Alpaca (Taori et al., 2023), which was trained on outputs generated through the OpenAI API.

Note that hard distillation is not necessarily the same as using a low T , since it depends on the decoding strategy used to generate the data. With an LLM API, you can still adjust the temperature, but you only get hard targets back. Though, perhaps, these hard targets can approximate the teacher’s output distribution: if you sample one token from the teacher, the one-hot vector of that sample is, *on average*, exactly the teacher’s distribution. So with enough samples, they do approximate it (see Figures 4c and 4d). Whether this works nicely for whole sequences is another story. See Figure 4 for a comparison between hard and soft distillation. Later, I will show you how hard and soft distillation behave differently in the toy problems.

3 Forward and Reverse KL

So, the vanilla KD proposed by Hinton et al. (2015) is what people call forward KL, where they use the Kullback–Leibler (KL) divergence^g to make the student match the teacher’s output distribution. Look at the KL term in Eq. (3): this is the forward KL. People say it has a characteristic called ‘mean-seeking’ (some also call it mode-covering), where the student tries to cover the teacher’s distribution as much as possible.

Note that if the student has a lower capacity (i.e., fewer parameters) than the teacher, it needs to compromise on what it can store. This is a problem: we want to make sure the student does store acceptable information from the teacher to perform

^e The gradient of the KL term with respect to the student logits scales as $1/T^2$, since the softmax divides the logits by T and the difference between the softened distributions shrinks by another factor of T . Multiplying by T^2 cancels this, so changing T does not require retuning α .

^f Imagine you have a vocabulary of 40,000 tokens and each sequence has 1,024 tokens on average. That’s around 41 million numbers (~160MB in float32!) for *one* sequence. The file gets heavy really fast.

^g In reality, there are lots of objective functions that can be used for KD (e.g., MSE, JS divergence, and so on). However, from this distribution point of view, KL divergence or similar divergences feel like the natural choice, since they measure how far apart two distributions are. Some work (Kim et al., 2021) says that MSE (on the logits) may be equal to, or even outperform, KL divergence. Actually, my prior work (Nityasya et al., 2022) also found this.

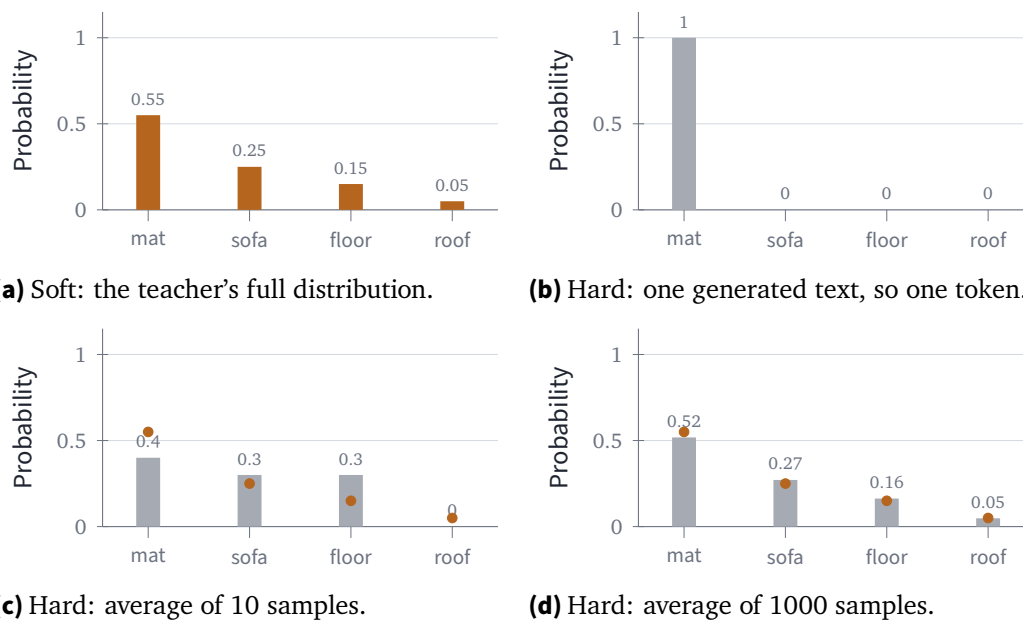


Figure 4 • Soft vs. hard distillation for the next token of “The cat sat on the ____”. With soft distillation (a), the student sees the teacher’s whole distribution. With hard distillation (b), it only sees the token the teacher happened to generate. If we average many generated samples (c, d; dots are the teacher’s real probabilities), we slowly get the soft one back, but we need *a lot* of samples for that.

well, but it doesn’t need to cover every piece of information from the teacher. MiniLLM (Gu et al., 2024) tries the OPPOSITE of forward KL: it uses reverse KL with on-policy data (we will get to this in §4), and finds that it performs better than forward KL with off-policy data. Reverse KL has a ‘mode-seeking’ characteristic, where the student focuses on fitting one (or a few) modes really well instead of trying to cover all modes of the teacher. For those of you who are still confused about these seeking things, please refer to the visualization in Figure 5.

The reverse KL is just the KL term in Eq. (3), but flipped (with $T = 1$ here, so $q^s = q_1^s = q_2^s$):

$$\text{KL}(q^s \parallel p) = \sum_i q_i^s \log \frac{q_i^s}{p_i}. \quad (5)$$

Notice that the average is now taken over the *student’s* distribution q^s , not the teacher’s.^h

Also, note that only the student gets updated! In both forward and reverse KL, the teacher is frozen: it just gives p as the target, and the gradient only flows into the student’s q^s (i.e., only θ in Eq. (3) changes). So when people say “reverse KL makes the student mode-seeking”, it is the student that moves to fit the teacher, never the other way around.

Note that, as I mentioned earlier, there are lots of objective functions, but in this blog I will only focus on these two. I will cover the others in the future.

^h Fun fact: this is why reverse KL and on-policy often come together. To estimate an average over q^s for long sequences, you naturally sample the sequences from the student itself, which is exactly on-policy (§4).

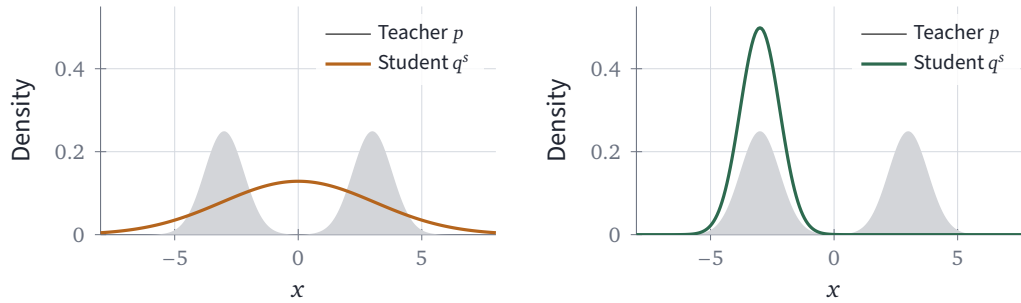
(a) Forward KL: $KL(p \parallel q^s)$.(b) Reverse KL: $KL(q^s \parallel p)$.

Figure 5 • Mean-seeking vs. mode-seeking. The teacher has two bumps (modes), but the student is a simple model that can only make one bump (a Gaussian). With forward KL (a), the student spreads out to cover both bumps, so it puts a lot of probability in the middle, where the teacher puts almost none. With reverse KL (b), the student picks one bump and fits it really well, and just ignores the other one. Both students here are the actual best fits for each loss.

Understanding the Forward and Reverse KL. Now, let's dissect the forward and reverse KL to understand their characteristics more. Look at a single term inside the sums of Eqs. (4) and (5), i.e., one class (or one token) i . Since both p_i and q_i^s can be anything, there are three common cases: p_i is much higher than q_i^s , q_i^s is much higher than p_i , and p_i is close or equal to q_i^s . Let's do this! See Table 1.

Table 1 • What one term of each KL does in each case. Remember that a KL is a sum of terms, one per class; this table only looks at one of those terms.

Case	Forward: $p_i \log \frac{p_i}{q_i^s}$	Reverse: $q_i^s \log \frac{q_i^s}{p_i}$
$p_i \gg q_i^s$ teacher likes it, student ignores it e.g., $p_i = 0.9$, $q_i^s = 0.01$	Huge. Big penalty, the student <i>must</i> put probability here. $0.9 \log \frac{0.9}{0.01} \approx 4.05$	≈ 0 . Doesn't care, the student is allowed to skip it. $0.01 \log \frac{0.01}{0.9} \approx -0.05$
$q_i^s \gg p_i$ student likes it, teacher says nope e.g., $p_i = 0.01$, $q_i^s = 0.9$	≈ 0 . Doesn't care, the student can put probability here. $0.01 \log \frac{0.01}{0.9} \approx -0.05$	Huge. Big penalty, the student <i>must not</i> go here. $0.9 \log \frac{0.9}{0.01} \approx 4.05$
$p_i \approx q_i^s$ e.g., $p_i = 0.5$, $q_i^s = 0.48$	≈ 0 . Happy. $0.5 \log \frac{0.5}{0.48} \approx 0.02$	≈ 0 . Happy. $0.48 \log \frac{0.48}{0.5} \approx -0.02$

The examples use the natural log ($\log = \ln$). Notice the same pair of numbers gives ≈ 4.05 in one KL and ≈ -0.05 in the other, just because we swapped who is p and who is q^s ! Also, one term can be slightly negative, but the whole sum is never below zero, and it is zero only when $p = q^s$. Note that lower is better.

So, forward KL hates it when the student misses something the teacher likes, hence the student spreads out to cover everything, even if that means putting probability in places the teacher doesn't like (mean-seeking). Reverse KL hates it when the student puts probability where the teacher says nope, hence the student plays safe and sits on

the modes it is sure about (mode-seeking). This is exactly what you see in Figure 5: the forward KL student puts a lot of probability in the middle (the second case is not punished), while the reverse KL student ignores the right bump (the first case is not punished).

In the toy problems, I will show you this seeking behavior in action and observe its characteristics more. While at it, I will also show you a mathematical derivation later (§7.1, with the full proofs in App. A).

4 Policy

So here is the last piece before we play with the toy problems: **policy**. The term comes from RL (Sutton and Barto, 2018), where a policy is the thing that decides which action to take. For an LLM, the policy is just the model itself: given the text so far, it picks the next token. So, when people say on-policy or off-policy in distillation, they are basically answering one question: *who writes the sequences used for training?* See Figure 6.

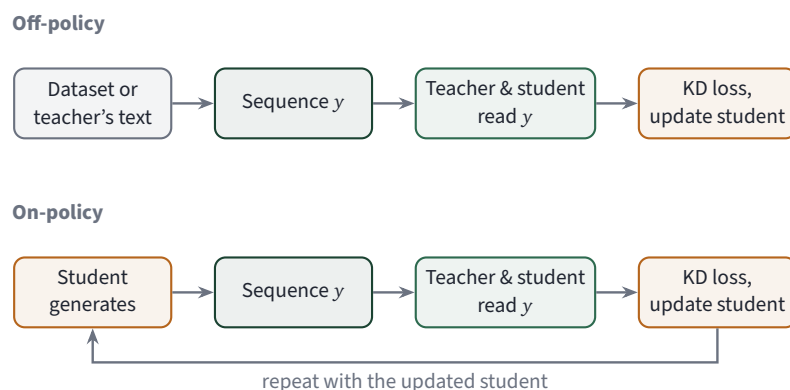


Figure 6 • The only difference between off-policy and on-policy is where the sequence y comes from. Off-policy uses fixed data written by someone else. On-policy lets the (current) student write it, so the student keeps learning on the kind of text it will actually produce.

Off-Policy. The sequences come from someone other than the student: the ground-truth dataset, or text generated by the teacher (like hard distillation above, Figure 4; Kim and Rush, 2016). The student then learns on these sequences, either from the teacher’s distribution at each token (soft) or just from the tokens (hard). This is cheap and easy, since you can prepare the data once and reuse it. The catch: at inference time, the student continues from *its own* outputs, not from the dataset. If it makes a small mistake early, it ends up in a situation it never saw during training, and the mistakes can pile up. People call this the train–inference mismatch (or exposure bias) (Ross et al., 2011; Agarwal et al., 2024). For instance, suppose the training data often contains “I like an apple”, but the student generates “I love ..” instead. If “I love” never appears in the training data, the student has never learned what should come next, so it may make a mistake or hallucinate (see Figure 7).

On-Policy. The student generates the sequences itself, and the teacher gives its distribution for every token the student wrote. So, the teacher is basically grading the student’s own homework and telling it what it should have done at each step (you can think of it as a reward), including how to recover from its own mistakes (Agarwal et al., 2024; Gu et al., 2024). The problem: it is more expensive, since you need to sample from the student and run the teacher during training. Note that you can also mix the two; for example, GKD (Agarwal et al., 2024) uses on-policy data some of the time and off-policy data the rest of the time. See Figure 7 for what the teacher and the student actually see, token by token, in both settings.

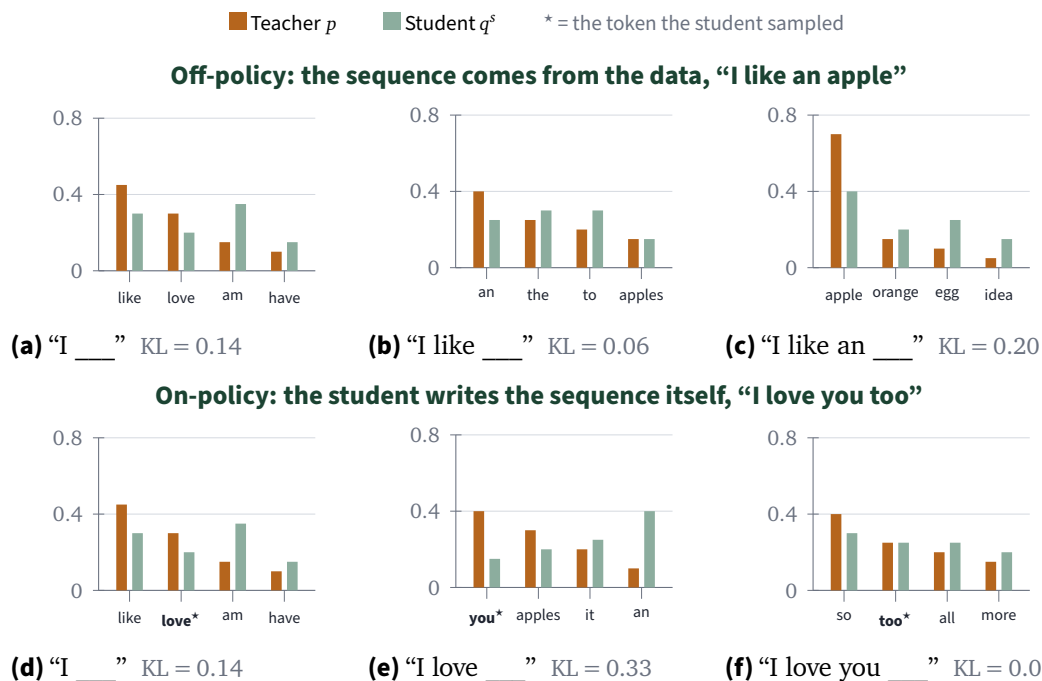


Figure 7 • What the teacher and the student see in a sequence. At every position, both of them read the same prefix (the text so far, e.g., “I like”) and give a distribution over the next token. The KD loss compares the two bars at *every* position (here with forward KL, shown under each panel) and adds them up. Off-policy (a–c): the prefixes come from the data, so the student only learns about “I like ...”. On-policy (d–f): the student happened to sample “love”, a prefix it rarely saw in the data. It is clueless after “I love” and wants to say “an” (copying its “I like an” habit), and this is exactly where the teacher steps in with the biggest correction (KL = 0.33 in e).

To sum it up, distillation roughly has three components: the models (the teacher(s) and the student(s)), the objective function, and the data used for training (i.e., the policy). For the objective, this blog only covers forward KL and reverse KL (Eqs. (4) and (5)). See Figure 8 for the whole picture.

5 Toy Problems

Now, we come to the exciting part: observing how the components above (§§2 and 4) behave! First of all, this blog focuses only on the KL term in Eq. (3), as that’s the

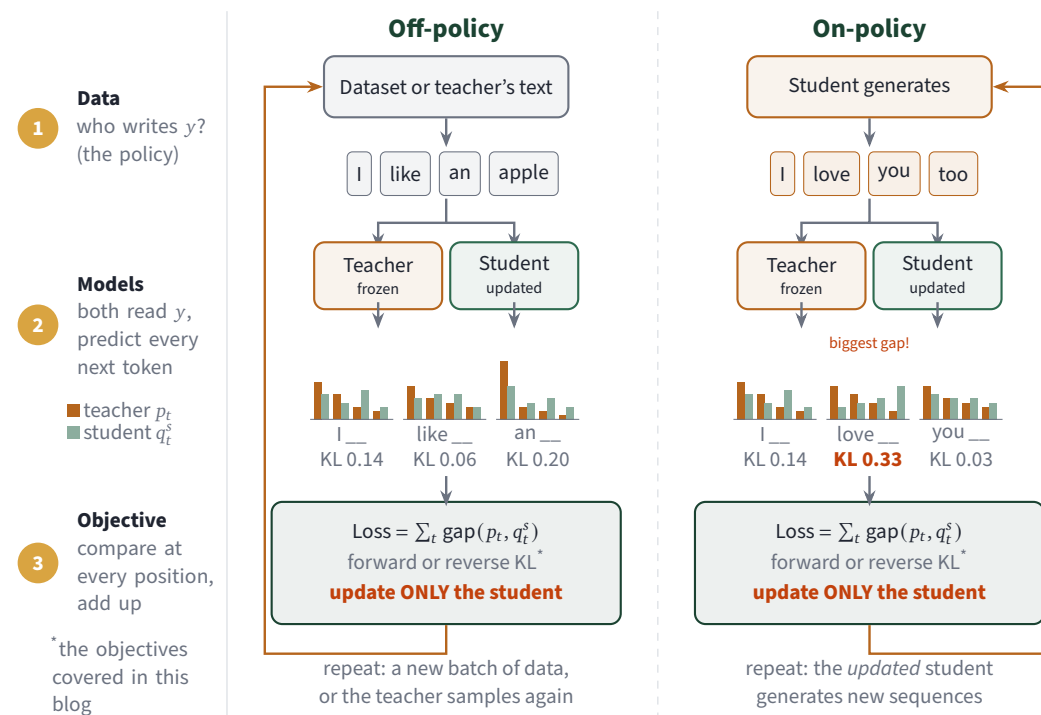


Figure 8 • The whole picture of distillation, with the two branches side by side (same numbers as Figure 7). **(1) Data (policy):** off-policy trains on a sequence written by someone else (“I like an apple”), while on-policy trains on what the student itself writes (“I love you too”). So the two branches give the models *different* sequences, and the teacher ends up correcting different things: on-policy catches the spot where the student is clueless (“love __”, KL = 0.33), which off-policy never shows it. **(2) Models:** in both branches, the teacher and the student read the same y and give p_t and q_t^s at every position. **(3) Objective:** each branch adds up its own gaps into a loss (forward KL or reverse KL in this blog), and **only the student gets updated**; the teacher stays frozen the whole time. Note that you pick *one* branch for training (or mix them, like GKD), and each branch has its own loop: off-policy grabs a new batch (from the dataset, or by letting the teacher sample again), while on-policy lets the *updated* student write new sequences, so its data keeps changing as the student learns.

part we are interested in (so $\alpha = 1$, no ground-truth anchor, and $T = 1$). Here, we look at non-sequence KD (Toy Problem 1, §6) and change different variables: the policy (off vs. on), the objective (forward vs. reverse KL), hard vs. soft targets, and the teacher/student models. (Sequence KD will be in Part 2.)

As mentioned in §1, instead of using advanced neural models (well, I will do that in the future), I will use simple models, since it’s much easier to see what’s going on. The question we want to answer is:

The question Given a low-capacity student and a higher-capacity teacher, how does distillation behave under different settings: policies, models, and the type of distribution passed?

6 Toy Problem 1: Non-Sequence KD

First of all, let me remind you that KD essentially wants the student to mimic the teacher’s distribution. This *DOES NOT NECESSARILY* mean the teacher has to be a neural network. It can also be a statistical model with a pre-defined output distribution. For instance, suppose the teacher has 3 labels; we can just assign the probabilities directly (e.g., label 0: 0.2, label 1: 0.5, label 2: 0.3).

The Teacher. In this toy problem, we let the teacher be a BI-MODAL GAUSSIAN distribution, i.e., it has $M = 2$ hills, each with its own predefined MEAN, VARIANCE, and weight. To make it look like an LLM’s next-token distribution, we put it over K classes $x \in \{0, 1, \dots, K - 1\}$ (think of them as tokens) and normalize it so it sums to 1:

$$\tilde{p}(x) = \sum_{m=1}^M w_m \exp\left(-\frac{(x - \mu_m^t)^2}{2(\sigma_m^t)^2}\right), \quad p(x) = \frac{\tilde{p}(x)}{\sum_{x'} \tilde{p}(x')}, \quad (6)$$

where μ_m^t , σ_m^t , and w_m are the mean (center), standard deviation (width), and weight of hill m . The superscript t means these belong to the *teacher* (same as z^t in Eq. (2)): they are fixed and never learned.

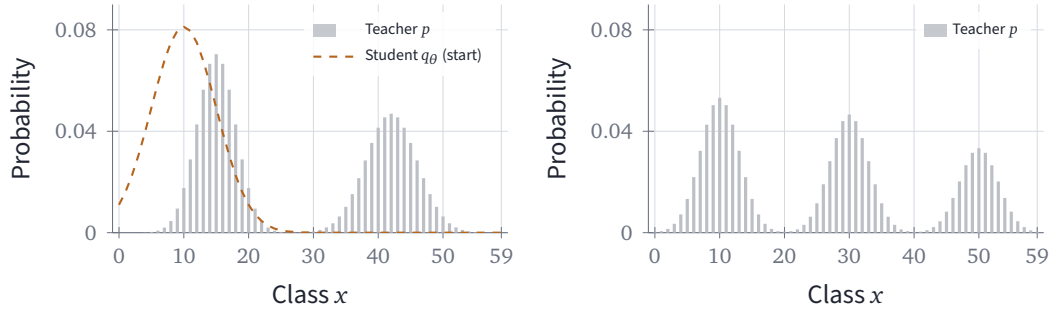
Basically, if you paid attention in your statistics class, you will get it at a glance: it’s just a Gaussian, but double, and we tell the model the mean and variance of each hill. If you didn’t, now you learn it lol. Quick reminder in case you forgot: in a normal/Gaussian distribution, the mean is where the peak is, while sigma is how wide it is.

So, the teacher is fully described by K and the $(\mu_m^t, \sigma_m^t, w_m)$ of each hill, and each class gets its own probability (see Figure 9a). The exact values I use are in Table 2. Note that the classes here are ordinal (e.g., class 5 is ‘more’ than class 4), though I believe this does not mean much here, since the KL still treats each class independently.

Table 2 • The values used in Toy Problem 1.

Symbol	Meaning	Value
K	number of classes	60
M	number of teacher hills	2 (main), 3 (extra)
$(\mu_m^t, \sigma_m^t, w_m)$	teacher’s hill m	$M=2$: (15, 3, 0.6), (42, 4, 0.4) $M=3$: (10, 3, 0.4), (30, 3, 0.35), (50, 3, 0.25)
(μ_0^s, σ_0^s)	student’s start	(10, 5)
N	samples per training step	64
	optimizer, learning rate, steps	Adam, 0.1, 4000
	seeds (hard runs only)	3

The Student. For the student, we use a simple model that can only make ONE GAUSSIAN HILL, so there are only TWO parameters to learn: the mean μ^s and the width σ^s (we actually learn $\log \sigma^s$, so σ^s always stays positive). The superscript s



(a) Two-hill teacher (main setup).

(b) Three-hill teacher (extra setup).

Figure 9 • The setup of Toy Problem 1. The teacher (gray bars) is a fixed distribution over K classes with $M = 2$ hills (a) or $M = 3$ hills (b). The student can only make one hill, so it has to choose how to be wrong. The dashed curve in (a) is where the student starts (μ_0^s, σ_0^s): near the first hill, but far from the second. The exact values are in Table 2.

means *student*: unlike the teacher's μ_m^t and σ_m^t , these two are the ones being learned. Here is the formula of the student:

$$q_\theta(x) = \frac{\exp\left(-\frac{(x-\mu^s)^2}{2(\sigma^s)^2}\right)}{\sum_{x'} \exp\left(-\frac{(x'-\mu^s)^2}{2(\sigma^s)^2}\right)}, \quad \theta = (\mu^s, \log \sigma^s). \quad (7)$$

Fun fact: this is just a softmax (Eq. (1)) whose logits are forced into the shape $-(x - \mu^s)^2/2(\sigma^s)^2$, so you can think of it as a tiny ‘neural network’ with only two weights.

So, the capacity of the student is LIMITED: it can never match the teacher's two hills exactly. This is actually what we want! If the student could match the teacher perfectly, every method would end up in the same place. Here, the objective and the policy decide *how* the student is wrong. So, how does distillation behave given this?

The Policy. In distillation, we need both the teacher's and the student's distributions, and the policy decides *who generates the samples* we train on: the teacher (off-policy) or the student (on-policy). Note that in this toy problem, each sample is only ONE TOKEN (one class x), not a sequence.

Wait, why do we need samples at all? In this toy, we could compute the KL exactly by summing over all K classes. But in real training (especially for sequences), there are way too many possible outputs to sum over, so people estimate the loss with samples instead. To keep things realistic, we do the same here.

What Is Sampling? If you are not familiar with statistics, think of sampling as rolling a weighted die with K sides. Each side is a class, and the weight of each side is its probability. For off-policy, the die is the teacher's distribution p (Eq. (6)), and we write it as

$$x_1, x_2, \dots, x_N \sim p, \quad \text{i.e.,} \quad \Pr(x_i = k) = p(k) \text{ for every class } k. \quad (8)$$

Since p has its hills around μ_1^t and μ_2^t , most rolls land near them (e.g., 15, 15, 43, 12, 38, ...), while classes where p is almost zero (e.g., the valley around 28, or 55)

basically never show up. The same class can also show up more than once (15 shows up twice here!). Each roll gives ONE token, and we roll N times to get one batch. Then, for every sampled token, we ask both models how likely it is, $p(x_i)$ and $q_\theta(x_i)$, and average the gap. For forward KL, this is

$$\widehat{\text{KL}}(p \parallel q_\theta) = \frac{1}{N} \sum_{i=1}^N \left[\log p(x_i) - \log q_\theta(x_i) \right], \quad x_i \sim p, \quad (9)$$

where the hat means it's an estimate of the real KL (Eq. (4)), and it gets closer to the real one as N grows. The whole pipeline is in Figure 10.

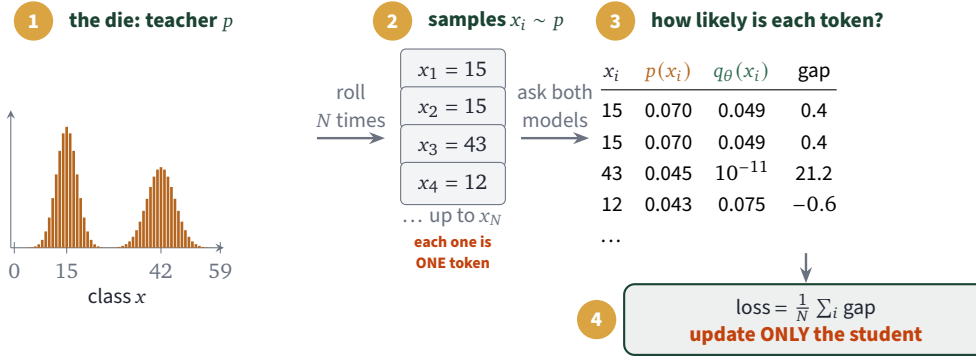


Figure 10 • One off-policy training step in Toy Problem 1. (1) The teacher's distribution p is a weighted die with K sides. (2) We roll it N times; each roll gives ONE token (one class), so tokens near the hills show up a lot, and the same token can show up twice. (3) For each sampled token, we look up how likely the teacher and the (current) student think it is; the gap is $\log p(x_i) - \log q_\theta(x_i)$ (Eq. (9)). Here the student is still at its starting point (around class 10), so it thinks class 43 (from the second hill) is almost impossible, and that token gets a huge gap. A gap can even be negative when the student likes a token more than the teacher does (class 12). (4) The average gap is the loss. **On-policy is the same pipeline**, except that in (1) we roll the student's die q_θ instead of the teacher's.

For **on-policy**, the pipeline is exactly the same, except that we roll the *student's* die instead: $x_i \sim q_\theta$ (Eq. (7)). And since the student keeps changing during training, the die (and so the samples) keeps changing too. With reverse KL, the estimate is just Eq. (9) flipped: $\frac{1}{N} \sum_i [\log q_\theta(x_i) - \log p(x_i)]$ with $x_i \sim q_\theta$.ⁱ

What if the Pair Doesn't Match? Importance Weighting. Say we want forward KL (an average over the teacher's p), but our samples come from the student's q_θ (on-policy). The trick is to give each sample a weight that corrects for how much more (or less) often q_θ rolls it than p would:

Lemma 6.1 (Importance weighting) *For any function f , if $q(x) > 0$ wherever $p(x) > 0$, then*

$$\mathbb{E}_{x \sim p} [f(x)] = \mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right].$$

ⁱ These are the natural pairs: forward KL is an average over p , so it goes well with teacher samples, and reverse KL is an average over q_θ , so it goes well with student samples. You can also do the other two pairs (e.g., forward KL with student samples), but then you need importance weighting (Lemma 6.1).

Proof. $\mathbb{E}_{x \sim q} \left[\frac{p(x)}{q(x)} f(x) \right] = \sum_x q(x) \frac{p(x)}{q(x)} f(x) = \sum_x p(x) f(x) = \mathbb{E}_{x \sim p} [f(x)]$. The q 's just cancel. ■

Think of dice: you want the average face of a *fair* die, but you can only roll a *loaded* one that gives a 6 half the time. The 6s show up too often, so each 6 gets a weight of $\frac{1/6}{1/2} = 0.33$, while the other faces show up too rarely and get $\frac{1/6}{1/10} = 1.67$, and the weighted average comes back to 3.5. So, forward KL with student samples becomes $\frac{1}{N} \sum_i \frac{p(x_i)}{q_\theta(x_i)} [\log p(x_i) - \log q_\theta(x_i)]$ with $x_i \sim q_\theta$, and reverse KL with teacher samples uses the weight q_θ/p instead. The catch: if q almost never rolls a place where p is big, then those rare samples get a *huge* weight, and the estimate jumps around a lot. Keep this in mind for the results!

In our runs, each training step uses N samples, and the student starts from $\theta_0 = (\mu_0^s, \log \sigma_0^s)$, on the left side: near the first hill, but far away from the second one. Also, note that an LLM outputs a token *given an input* (the prompt), i.e., $P_\theta(\text{token} \mid \text{prompt})$, while our toy has no input at all: it's just $P_\theta(x)$, which is our $q_\theta(x)$. Doing the experiment with inputs would be interesting, but that'll be for a future blog, I guess XD.

The setup can be seen in Figure 9. Note that I will also try what happens if the teacher has $M = 3$ hills instead of two, while the student still has only one (Figure 9b).

7 Result of Toy Problem 1

Here are the results for the two-hill teacher (Figure 9a). The soft settings compute the KL exactly over all K classes, so there is no sampling and no randomness at all. That also means soft has no policy here: with only ONE token and no input, there is nothing for the teacher or the student to generate (this changes with sequences, where the prefixes are sampled; more on that in Part 2). The hard (sampled) settings use N sampled tokens per step, and each is run with 3 seeds. Everything is summarized in Table 3. Note that I trained for 4000 steps, since with forward KL, the student was still moving at 1500 steps (see Figure 13).

Wait, Why Doesn't Soft Have a Policy? You might think: teacher samples and student samples are different data, so soft off-policy and soft on-policy should be different, right? Yes, the data is different. But whether that matters depends on whether the sample actually shows up *inside* the loss. Let's write the loss for one sampled token x_i :

$$\ell_{\text{soft}}(x_i) = \sum_{k=1}^K p(k) \log \frac{p(k)}{q_\theta(k)}, \quad \ell_{\text{hard}}(x_i) = \log p(x_i) - \log q_\theta(x_i). \quad (10)$$

Look closely: ℓ_{soft} sums over *all* classes k , so the sampled x_i doesn't appear in it at all. ℓ_{hard} only looks at the one token we sampled, so x_i is right there.

Why Does ℓ_{hard} Look Like That? Hard targets mean that, other than the target label, all the other classes get 0. For example, if the sampled token is class 3 (out of 4), the

Table 3 • Where the student ends up after training (two-hill teacher). Hard rows are the mean over 3 seeds, with the standard deviation as a subscript. “Own KL” is the KL that setting minimizes. “Wobble” is how much μ^s still moves over the last 300 steps (standard deviation); lower means calmer.

Target	Policy	μ^s	σ^s	Own KL	Wobble
FORWARD KL, $\text{KL}(P \parallel Q_\theta)$					
Soft	none (exact)	26.87	16.91	0.66	0.00
Hard	off ($x \sim p$)	26.73 $\pm$ 0.07	17.15 $\pm$ 0.35	0.66 $\pm$ 0.00	0.27 $\pm$ 0.06
Hard	on ($x \sim q_\theta$)	23.51 $\pm$ 2.79	19.30 $\pm$ 2.57	0.67 $\pm$ 0.01	0.64 $\pm$ 0.24
REVERSE KL, $\text{KL}(Q_\theta \parallel P)$					
Soft	none (exact)	15.00	3.00	0.64	0.00
Hard	on ($x \sim q_\theta$)	15.00 $\pm$ 0.00	3.00 $\pm$ 0.00	0.64 $\pm$ 0.00	0.01 $\pm$ 0.01
Hard	off ($x \sim p$)	14.97 $\pm$ 0.05	3.02 $\pm$ 0.04	0.64 $\pm$ 0.00	0.02 $\pm$ 0.01

Soft rows have no policy: they compute the KL exactly by summing over all K classes, so nothing is sampled (with only ONE token and no input, soft off-policy and soft on-policy would be the exact same calculation). For reference, the teacher’s first hill is $(\mu_1^t, \sigma_1^t) = (15, 3)$, and the mean of the whole teacher distribution is about 27.7. Note that μ^s is the *center* of the student’s bell curve; since the curve is cut off at classes 0 and 59, the forward KL student’s actual average is 27.7, matching the teacher (Proposition 7.4).

hard target is $[0, 0, 0, 1]$, while the student might say $q_\theta = [0.3, 0.1, 0.1, 0.5]$. If we plug this one-hot target into the KL (Eq. (4)), every class with target 0 disappears (since $0 \cdot \log 0 = 0$), and only the sampled class is left:

$$\text{KL}(\text{one-hot} \parallel q_\theta) = 1 \cdot \log \frac{1}{q_\theta(x_i)} = -\log q_\theta(x_i), \quad (11)$$

which is just the plain cross-entropy on the sampled token (basic log arithmetic, in case you forgot: $\log \frac{a}{b} = \log a - \log b$). Our ℓ_{hard} in Eq. (10) only adds $\log p(x_i)$ on top. That extra term doesn’t depend on the student at all, so both give exactly the same gradient; the $\log p(x_i)$ is only there so that averaging ℓ_{hard} over many teacher samples gives back the real KL (Eq. (9)).

A Tiny Example. Let’s take only 3 classes, with teacher $p = [0.2, 0.5, 0.3]$ and student $q_\theta = [0.4, 0.4, 0.2]$. The soft loss compares the whole vectors at once:

$$\ell_{\text{soft}} = 0.2 \ln \frac{0.2}{0.4} + 0.5 \ln \frac{0.5}{0.4} + 0.3 \ln \frac{0.3}{0.2} = -0.139 + 0.112 + 0.122 = 0.095,$$

while the hard loss depends on which token we rolled: $\ln \frac{0.2}{0.4} = -0.69$ for class 0, $\ln \frac{0.5}{0.4} = 0.22$ for class 1, and $\ln \frac{0.3}{0.2} = 0.41$ for class 2. Now, let the teacher and the student each roll 4 tokens (Table 4).

So, the teacher and the student still disagree ($p \neq q_\theta$, the soft loss is 0.095, not 0), but *who rolled the tokens* doesn’t change the soft loss at all. That’s why soft has no policy in Toy 1. The hard loss, on the other hand, gives different numbers for the teacher’s batch and the student’s batch, so off- and on-policy really differ. Notice

Table 4 • Soft vs. hard loss on the same samples ($p = [0.2, 0.5, 0.3]$, $q_\theta = [0.4, 0.4, 0.2]$). The soft loss is the same for every sample, so who rolls the tokens doesn't matter. The hard loss changes with every sample, so it does.

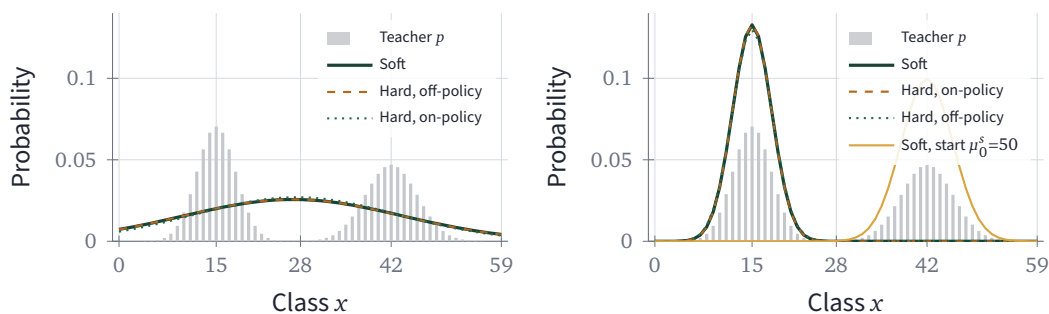
Who rolls	Samples	ℓ_{soft} each	ℓ_{hard} each	Avg. soft	Avg. hard
Teacher (off)	1, 1, 2, 0	0.095	0.22, 0.22, 0.41, -0.69	0.095	0.04
Student (on)	0, 0, 1, 2	0.095	-0.69, -0.69, 0.22, 0.41	0.095	-0.19

that the student's batch even gives a *negative* hard loss, which can't be right for a KL! That's because forward KL should be weighted by the teacher's $p(k)$, but the student rolls with its own $q_\theta(k)$. This is exactly why forward KL with student samples needs importance weights (p/q_θ) to fix the weighting.

In sequences, it's different again: the sampled prefix $y_{<t}$ becomes the *context*, so it shows up inside the soft loss too, through $p(k | y_{<t})$ and $q_\theta(k | y_{<t})$. So there, soft off- and on-policy really give different losses (remember “I like ___” vs. “I love ___” in Figure 7).

7.1 Forward vs. Reverse KL

Mean-Seeking vs. Mode-Seeking. Figure 11 shows where the student ends up. As expected, with forward KL, the student covers both teacher hills ($\mu^s \approx 26.9$, $\sigma^s \approx 16.9$): its center μ^s sits between the two hills, and its width σ^s gets much bigger to reach both. So, it puts a lot of probability in the valley (around class 28), where the teacher has almost none. This is what being mean-seeking means: the student goes for the overall mean of the teacher's distribution, even if the teacher has almost nothing there.



(a) Forward KL: one wide hill over both.

(b) Reverse KL: sits on one hill.

Figure 11 • Where the student ends up (seed 0 for the hard runs). With forward KL (a), the student spreads over both hills and puts a lot of probability in the valley around class 28. With reverse KL (b), it sits exactly on the first hill, because it starts on the left ($\mu_0^s = 10$); starting on the right ($\mu_0^s = 50$, gold) makes it pick the other hill instead. In both panels, the soft and hard students mostly end up on top of each other.

Interestingly, with reverse KL, the student sits exactly on the first hill of the teacher ($\mu^s = 15$, $\sigma^s = 3$, which are exactly that hill's own mean and width) and ignores the

second one. This is what we call mode-seeking: the student picks one mode and fits it really well.

Reverse KL Depends on the Starting Point. Which hill reverse KL picks depends on where the student starts (Figure 11b). Starting from $\mu_0^s = 10$, it picks the hill at 15, but starting from $\mu_0^s = 50$, it picks the hill at 42 ($\mu^s = 42$, $\sigma^s = 4$). Forward KL, on the other hand, always ends at the same place from both starts.

This makes sense if we look at the gradients. Our student is a softmax (Eq. (7)): $q_\theta = \text{softmax}(z)$ with logits $z_k = -(k - \mu^s)^2 / 2(\sigma^s)^2$. So, it's enough to look at how each KL reacts when we nudge one logit z_k (the gradient for μ^s and σ^s then just follows from the chain rule).

Lemma 7.1 (Forward KL gradient) *For $q = \text{softmax}(z)$,*

$$\frac{\partial}{\partial z_k} \text{KL}(p \parallel q) = q_k - p_k.$$

↪ Proof in App. A

Lemma 7.2 (Reverse KL gradient) *For $q = \text{softmax}(z)$,*

$$\frac{\partial}{\partial z_k} \text{KL}(q \parallel p) = q_k \left(\log \frac{q_k}{p_k} - \text{KL}(q \parallel p) \right).$$

↪ Proof in App. A

Comparing the two gradients gives the “bounded vs. not bounded” story:

Corollary 7.3 (Forward is bounded, reverse is not) *Let $q = \text{softmax}(z)$. Then:*

- (i) Forward KL is bounded: *for any teacher p and student q , $\left| \frac{\partial}{\partial z_k} \text{KL}(p \parallel q) \right| \leq 1$.*
- (ii) Reverse KL is not: *keep the student fixed with $0 < q_k < 1$, and let the teacher's p_k go to 0. Then $\frac{\partial}{\partial z_k} \text{KL}(q \parallel p) \rightarrow +\infty$.*
- (iii) Reverse KL ignores classes the student doesn't visit: *if the teacher has $p_j > 0$ for every class and the student's q_k goes to 0, then $\frac{\partial}{\partial z_k} \text{KL}(q \parallel p) \rightarrow 0$, no matter how big p_k is.*

↪ Proof in App. A

In words: with forward KL (i), the gradient stays between -1 and 1 , and every class the teacher likes ($p_k > 0$) pulls the student toward it, no matter how far that class is from where the student currently is. So, the far hill always gets covered.

With reverse KL, two things happen instead. First, (iii): look at Lemma 7.2 again, and notice that the whole signal for class k is *multiplied by the student's own probability q_k* . So, if the student gives a class almost zero probability, the teacher's opinion about that class barely reaches the student, even if the teacher loves that class. Let's check it

on our two-hill teacher, with the student sitting on the first hill ($\mu^s = 15$, $\sigma^s = 3$), and look at class 42, the peak of the far hill. The teacher likes it ($p_{42} \approx 0.047$), but the student almost ignores it ($q_{42} \approx 3 \times 10^{-19}$). With reverse KL, the gradient for class 42 is about -1×10^{-17} , basically nothing, so the student never hears “the teacher wants class 42”. With forward KL, it is $q_{42} - p_{42} \approx -0.047$, a clear pull toward class 42. Think of reverse KL as checking your answers with a flashlight: you only see the teacher’s opinion where your flashlight (your own probability) is shining. The far hill is in the dark, so as far as the student knows, it isn’t there, and the student just rolls into the nearest hill (that’s why the starting point matters). Forward KL, on the other hand, turns on the lights in the whole room.

Second, (ii): where the teacher has almost nothing (the valley) but the student still puts some probability, the gradient blows up, so the student gets pushed out of the valley really hard (mode-seeking).

Proposition 7.4 (Forward KL has one best answer) *For our one-hill student (Eq. (7)), forward KL has only one resting point, no matter where the student starts. At that point, the student copies the teacher’s mean and spread: $\mathbb{E}_{q_\theta}[x] = \mathbb{E}_p[x]$ and $\mathbb{E}_{q_\theta}[x^2] = \mathbb{E}_p[x^2]$.*

↪ Proof in App. A

This is mean-seeking, written as math: the student’s mean lands on the teacher’s overall mean, which, with two far-apart hills, is right in the valley. Checking it on our two-hill teacher: the student’s mean is 27.70 vs. the teacher’s 27.71, and its $\mathbb{E}[x^2]$ is 961.2 vs. 961.5. Reverse KL has no such guarantee (because of Lemma 7.2), so every hill can be its own resting point.

So, Which One Is Better? Neither, really: each one wins on its own KL. The forward KL student gets $\text{KL}(p \parallel q_\theta) = 0.66$, while the reverse KL student gets 18.3 on that same metric. Flip the metric, and the reverse KL student gets $\text{KL}(q_\theta \parallel p) = 0.64$, while the forward KL student gets 1.55. Again, which one you want depends on your use case: covering everything the teacher can do, or doing one thing really well.

What If the Hills Are Close to Each Other? In Figure 12, I move the hills closer (centers at 25 and 35), so the valley between them is shallow. Now, forward and reverse KL give almost the same wide hill ($\mu^s \approx 29.7$ vs. 30.1, $\sigma^s \approx 6.1$ vs. 5.8), so the mode-seeking behavior doesn’t really show. Perhaps, the mode-seeking behavior only shows up when the valley is deep?

7.2 Off- vs. On-Policy

Here, the “natural” pairs are forward KL with teacher samples (off-policy) and reverse KL with student samples (on-policy). The other two pairs need importance weights (Lemma 6.1).

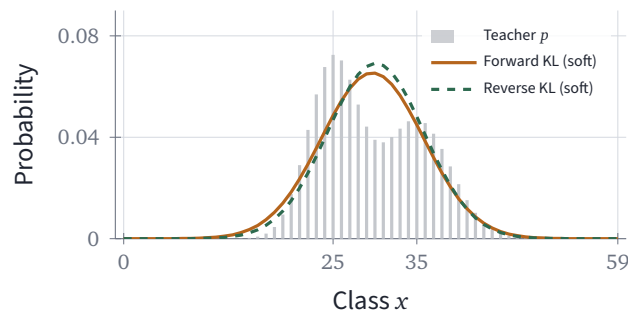


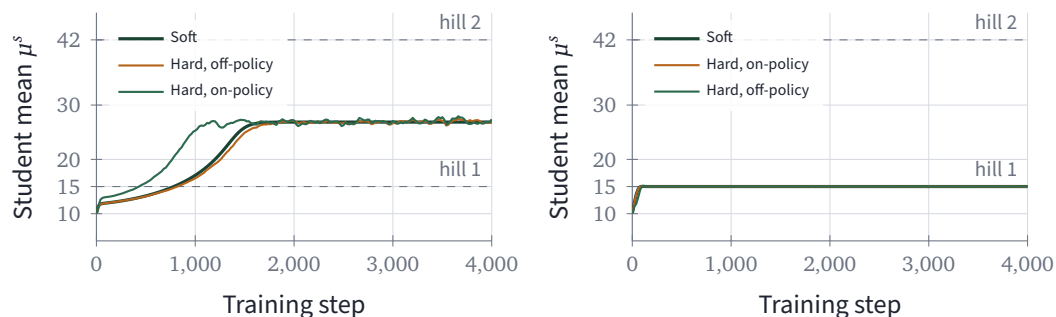
Figure 12 • When the two hills are close (centers at 25 and 35), the valley is shallow, and forward and reverse KL end up with almost the same wide hill.

Final Destination. With the natural pairs, hard training lands at the same place as soft training ($\mu^s = 26.7$ vs. 26.9 for forward KL; 15.0 vs. 15.0 for reverse KL, Table 3).

Forward KL + On-Policy Is the Least Reliable. Across the 3 runs, it ends up in different places ($\mu^s = 23.5 \pm 2.8$, vs. only ± 0.07 for off-policy), and it wobbles the most (0.64). From what I see, the student rarely samples the far hill, so it rarely sees what the teacher wants there. And when it finally does, the importance weight for that sample becomes huge, which makes the gradient jump around (see the bumpy on-policy curve in Figure 13a).

Reverse KL + Off-Policy. This one was okay (all 3 seeds reach the first hill), but it is a bit less stable than reverse KL + on-policy (wobble 0.02 vs. 0.01), and at 1500 steps, one of the three seeds had not made it yet.

Conclusion. For forward KL, off-policy is better, whereas for reverse KL, on-policy is better. Basically, use the policy that matches the KL.



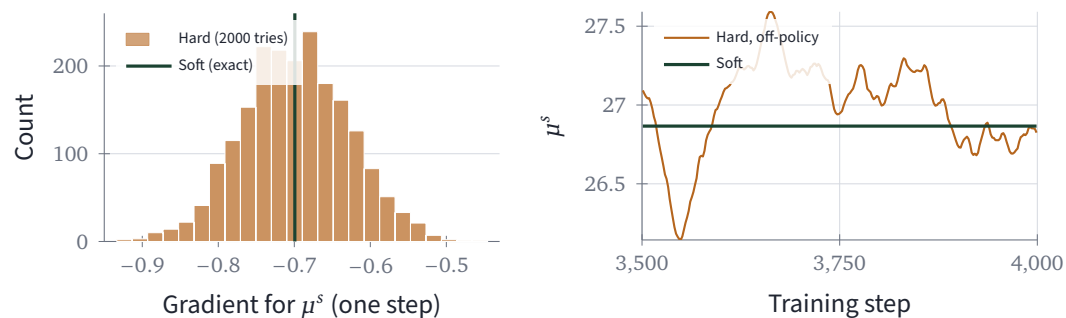
(a) Forward KL.

(b) Reverse KL.

Figure 13 • How the student's mean μ^s moves during training (seed 0). It starts at $\mu_0^s = 10$. With forward KL (a), the student first fits the near hill, then slowly stretches toward the far one, and it is still moving after 1500 steps. With reverse KL (b), it reaches the near hill almost right away and stays there. The hard runs mostly follow the soft one, just less smoothly; the on-policy forward KL run takes its own, bumpier path (and its 3 seeds end up in different places, Table 3).

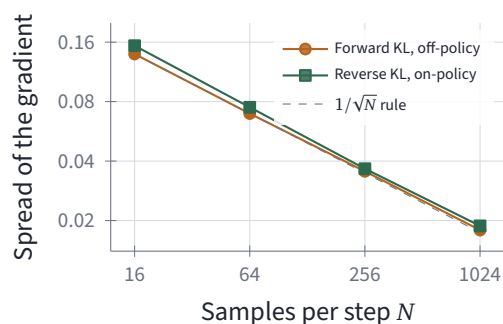
7.3 Hard vs. Soft

Figure 14 zooms in on what happens with hard targets.



(a) One step, at the start.

(b) The last 500 steps.



(c) More samples, less noise.

Figure 14 • Hard targets give a noisy gradient (forward KL). (a) At the start, we compute the gradient for μ^s 2000 times with N fresh samples each time. The hard estimates are centered on the exact (soft) value, but each single one is off by a bit. (b) So at the end of training, the soft student sits still, while the hard student keeps wobbling around the same answer. (c) The spread of the hard gradient (standard deviation over 2000 tries, at the start) for different N : it halves every time N goes up 4 $\times$, following the $1/\sqrt{N}$ rule (dashed).

Final Destination. Basically, both end at the same place (Table 3). On average, the hard gradient also points the same way as the soft one: at the start, the true (soft) gradient for μ^s is -0.70 , and the average of 2000 hard estimates is also -0.70 .

Noisiness. However, each single hard estimate is a noisy guess: those 2000 estimates spread out with a standard deviation of 0.07, which is about 10% of the true value (Figure 14a). Because of this noise, the student never fully stops. Soft training stays still once it arrives (wobble = 0), while hard training keeps wobbling around the answer (wobble ≈ 0.27 for forward KL, off-policy, Figure 14b).

Reducing the Noise. Remember that the hard gradient is just an *average* over the N sampled tokens: each token x_i gives its own gradient g_i , and we use $\hat{g} = \frac{1}{N} \sum_i g_i$. Two basic facts from statistics tell us how good this average is:

Lemma 7.5 (Hard estimates are right on average, with noise $\propto 1/\sqrt{N}$) *If $x_1, \dots, x_N \sim p$ are independent, then (i) on average, the hard estimate equals the real KL: $\mathbb{E}[\widehat{\text{KL}}(p \parallel q_\theta)] = \text{KL}(p \parallel q_\theta)$ (Eq. (9)), and the same holds for its gradient; (ii) the spread of the average shrinks like $1/\sqrt{N}$ (Eq. (12)).*

Proof. (i) An average of samples has the same expected value as one sample, and one sample $x \sim p$ gives $\mathbb{E}[\log p(x) - \log q_\theta(x)] = \sum_k p(k)[\log p(k) - \log q_\theta(k)] = \text{KL}(p \parallel q_\theta)$. (ii) If the g_i are independent and each has variance $\text{Var}[g]$, their variances simply add up, so

$$\text{Var}[\hat{g}] = \text{Var}\left[\frac{1}{N} \sum_{i=1}^N g_i\right] = \frac{1}{N^2} \sum_{i=1}^N \text{Var}[g_i] = \frac{\text{Var}[g]}{N}, \quad \text{so} \quad \text{std}[\hat{g}] = \frac{\text{std}[g]}{\sqrt{N}}. \quad (12)$$

(See, e.g., Wasserman, 2004; and Mohamed et al., 2020 for this in the context of gradient estimators in machine learning.) In words: the spread shrinks like $1/\sqrt{N}$, so to cut the noise in half, you need $4\times$ more samples.^j And indeed, every time N goes up $4\times$, the spread of the gradient is cut in half ($0.14 \rightarrow 0.07 \rightarrow 0.036 \rightarrow 0.018$ for $N = 16, 64, 256, 1024$, Figure 14c), and the wobble at the end also goes down ($0.47, 0.27, 0.18, 0.08$). So, more samples give more stability, but of course, more compute per step.

^j For on-policy reverse KL, the samples share a baseline (the average reward of the batch), so they are not perfectly independent, and Eq. (12) is only approximately true. It still follows the rule nicely in Figure 14c.

7.4 Three Hills: What If the Student Has One or Two Hills?

Now, let's make the teacher harder: three hills at classes 10, 30, and 50, all with the same width ($\sigma_m^t = 3$) and weights 0.40, 0.35, and 0.25 (Figure 9b). And this time, I also give the student a second hill. With two hills, the student is a mix of two copies of Eq. (7):

$$q_\theta(x) = \pi_1 q_1(x) + \pi_2 q_2(x), \quad \pi_1 + \pi_2 = 1, \quad (13)$$

where each q_j has its own μ_j^s and σ_j^s , and π_j says how much weight each student hill gets (also learned).

Setup. To keep things clean, I only use soft (exact) training here, so there is no sampling noise at all. The only randomness is *where the student starts*: for each of 10 seeds, the student's means μ_j^s start at random spots between 0 and 59, and its widths start at random values between 2 and 8. Each run uses Adam (learning rate 0.05) for 8000 steps, which is long enough that every seed stops moving.

A Handy Rule for Reverse KL. Before looking at the results, here's a rule that helps a lot.

Lemma 7.6 (Copying some hills) *Suppose the student copies some of the teacher's hills exactly and puts nothing anywhere else, and those hills hold a total weight m of the teacher's probability. Then $\text{KL}(q_\theta \parallel p) = -\log m$.*

Proof. On the kept hills, the student is just the teacher rescaled to sum to 1, $q_\theta = p/m$, and it is 0 everywhere else (those terms add nothing, since $0 \log 0 = 0$). So

$$\text{KL}(q_\theta \parallel p) = \sum_{x \text{ in kept hills}} \frac{p(x)}{m} \log \frac{p(x)/m}{p(x)} = \log \frac{1}{m} \sum_{x \text{ in kept hills}} \frac{p(x)}{m} = -\log m. \quad (14)$$

■

So, copying only the hill at 10 costs $-\log 0.40 = 0.92$, only the hill at 30 costs 1.05, and only the hill at 50 costs 1.39. Copying two hills costs 0.29 (10 and 30), 0.43 (10 and 50), or 0.51 (30 and 50). Fun fact: this also explains the two-hill result above (§7.1), where the first hill holds about 53% of the teacher, so copying it costs $-\log 0.53 = 0.64$, exactly the reverse KL in Table 3.

One Student Hill. With forward KL, all 10 seeds end up at the exact same place: one very wide, almost flat hill ($\mu^s \approx 20.6$, $\sigma^s \approx 30.5$) that covers everything (Figure 15a). Again, forward KL has only one best answer, no matter where the student starts.

Table 5 • Where the 10 seeds end up with the three-hill teacher. Each student hill is written as (μ^s, σ^s) . “Loss” is the KL that setting minimizes (lower is better); the first row of each group is the best answer the seeds found.

Student, objective	Seeds	Student hills (μ^s, σ^s)	Loss
1 hill, forward	10/10	(21, 30)	0.48
1 hill, reverse	8/10	(25, 28)	0.84
	1/10	(30, 3)	1.05
	1/10	(50, 3)	1.39
2 hills, forward	4/10	(10, 3) + (38, 13)	0.29
	5/10	(18, 13) + (50, 3)	0.37
	1/10	(21, 30)	0.48
2 hills, reverse	3/10	(10, 3) + (40, 16)	0.42
	2/10	(30, 3) + (50, 3)	0.51
	1/10	(18, 40) + (30, 2)	0.55
	4/10	(18, 14) + (50, 3)	0.57
	0/10	(10, 3) + (30, 3) (<i>best possible</i>)	0.29

The surprise is reverse KL: 8 out of 10 seeds *also* end up as a wide hill ($\mu^s \approx 25$, $\sigma^s \approx 28.4$, loss 0.84), and only 2 seeds sit on a single teacher hill, at 30 (loss 1.05) and at 50 (loss 1.39) (Figure 15b). So, reverse KL is not mode-seeking here! Lemma 7.6 tells us why: copying any single hill costs at least 0.92, while spreading out costs only 0.84, so spreading out is simply cheaper. In the two-hill case, it’s the other way around (copying the first hill costs 0.64, spreading costs 1.55), so the student picks a hill. Together with the close-hills result (Figure 12), this tells us that whether reverse KL is mode-seeking depends on the teacher’s shape: it picks one hill only when that hill holds enough of the teacher’s probability.

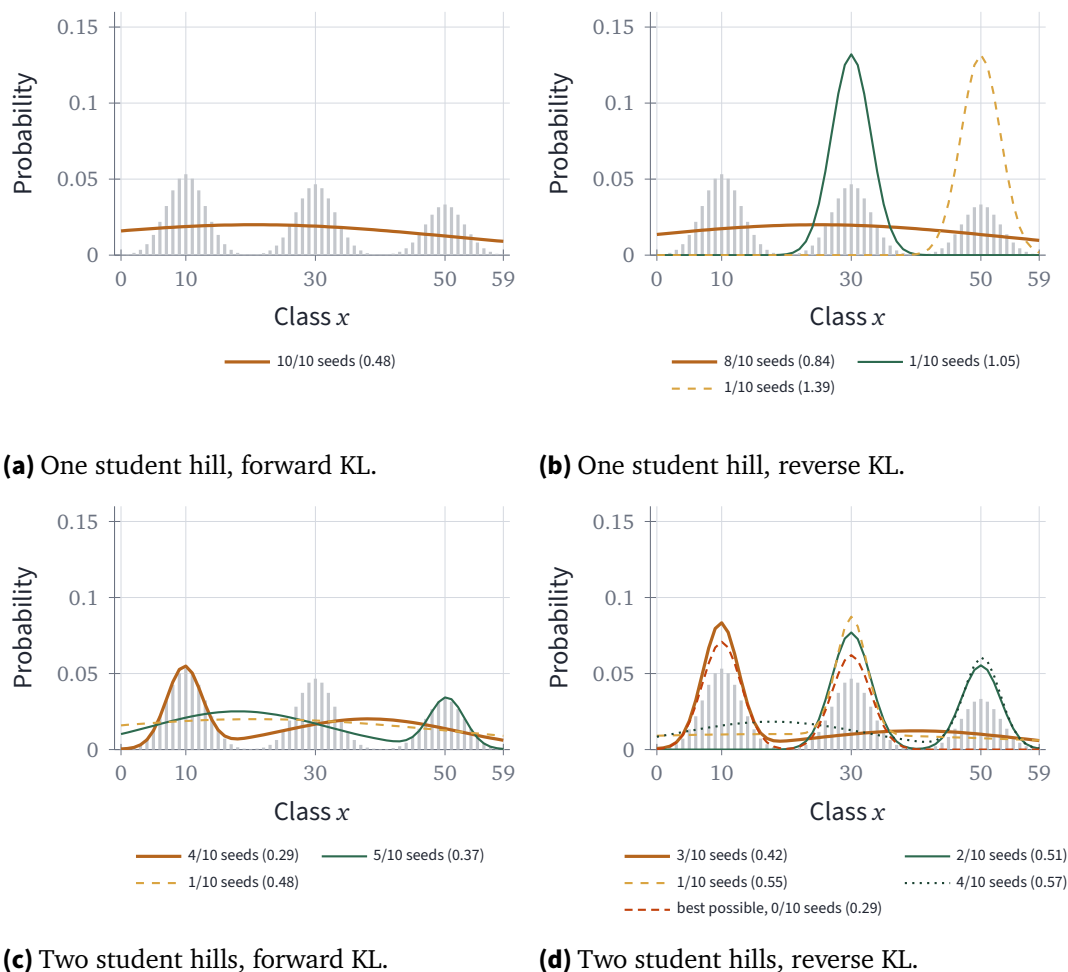


Figure 15 • Three-hill teacher (gray bars), student with one hill (top) or two hills (bottom), each trained from 10 random starting points (seeds). Each line is one place where the seeds ended up; the legend says how many seeds ended there and the final loss (the KL that setting minimizes, lower is better). With one student hill (a, b), most seeds end as one wide, almost flat hill. With two student hills (c, d), the seeds end up in several different places. In (d), the best possible answer for reverse KL (copying the hills at 10 and 30 exactly, red dashed) exists, but none of the 10 seeds found it.

Two Student Hills. Now, the seeds start to disagree (Table 5). With forward KL, they end up in three different places (Figure 15c). The best one (4 seeds, loss 0.29) puts a narrow hill on 10 and stretches a wide hill over 30 and 50; 5 seeds do it the other way around (wide over 10 and 30, narrow on 50, loss 0.37); and 1 seed basically gives up on its second hill (weight ≈ 0) and ends as a single wide hill (loss 0.48). So even forward KL can get stuck now: with a mix of hills, the “only one best answer” property is gone. Note that each student hill still covers, though: a wide student hill happily stretches over two teacher hills.

With reverse KL, it gets even messier: the seeds end up in four different places, and *none of them is the best one* (Figure 15d). By Lemma 7.6, the best possible answer is to copy the hills at 10 and 30 exactly (loss 0.29), and I checked that it really is

a resting point by starting the student right there. But 0 out of 10 random seeds found it. Instead, 3 seeds put a narrow hill on 10 and a wide one over 30 and 50 (loss 0.42), 2 seeds copy the hills at 30 and 50 exactly and ignore the hill at 10 (loss $0.51 = -\log 0.60$, the classic mode-seeking picture), and the rest end up with one wide and one narrow hill (loss 0.55 and 0.57).

Why Seeds Matter. With a simple student (one hill) or a simple teacher (two hills), forward KL always lands at the same place, and the starting point only matters for reverse KL. But once the student gets more flexible (two hills), there are many more places to get stuck, for both objectives, and the result really depends on the seed. So, if you try this yourself, run several seeds (or start the student somewhere smart), and don't trust a single run!

8 Closing Remarks with More Intuition

In this blog, I shared the results of me playing around with distillation, especially the policies (on/off), reverse and forward KL, and hard and soft distillation. The goal was to dive deep into their characteristics through toy problems and mathematical proofs.

How Does This Relate to KD in NLP Now? As you can see, people are leaning toward on-policy distillation, often together with reverse KL (Gu et al., 2024; Agarwal et al., 2024; Lu and Thinking Machines Lab, 2025). As the proofs show, reverse KL has no bound on its gradient, unlike forward KL (Corollary 7.3), so it can push the student very hard; on the other hand, the distribution it gives is quite aligned with what we usually want (high-quality outputs). But since the result depends on where the student starts (Lemma 7.2), and the starting point is the pre-trained model, it would be wise to give the student a proper prior first (for instance, by doing SFT), since otherwise it may get stuck in a bad local minimum (remember the seeds in §7.4). This is also what MiniLLM and GKD do: they start from a student that was already fine-tuned (Gu et al., 2024; Agarwal et al., 2024). How to do this well is still open research. Some papers look at related problems: for instance, Xin et al. (2026) show that during on-policy distillation, the student can drift into a bad prefix (e.g., a reasoning mistake or repeated tokens) where the teacher locally agrees with it, so the reverse KL is low but gives almost no corrective signal (they call it the “low-KL agreement trap”), and they propose stopping such rollouts early. This is very much in the spirit of Corollary 7.3 (iii): with reverse KL, the teacher only gives feedback on the states the student actually visits. Again, it depends on your use case: I think the intuition is to figure out which distribution you want your student model to have.

Another Fun Fact. If you've realized it by now, the teacher that I showed you is basically like training data, which has its own distribution. This is actually how you should see normal training: the data usually comes as hard labels, and we do off-policy training on it. In reality, the data itself was produced by some process (or by another model), and your model maps it into a distribution (e.g., a transformer

outputs a probability distribution), which may not cover every possible pattern in the data, again due to capacity. That’s why crafting appropriate training data is IMPORTANT, so that you get the distribution you want. And since you train with the cross-entropy loss, which is just forward KL with hard targets (see the ℓ_{hard} paragraph in §7), you get mean-seeking behavior toward the whole population of the data. Then, when that model is distilled into a smaller model, the behavior again depends on the loss used. So, based on the experiments, you can pick whichever suits your use case, or mix them like GKD (Agarwal et al., 2024) and DistiLLM (Ko et al., 2024).

Feedback Please! I am still eager to learn about this field, even though I have been in it for a while. Here, I used LLMs to assist me with the proofs, writing (grammar and better phrases), and the tedious stuff (especially making this blog nicer to read and play with). The topics I talk about here are things I was already familiar with, but I had never played around with them before this write-up. I just took it for granted that reverse and forward KL are mode-seeking and mean-seeking, respectively, but I never checked it with my own eyes. So here I am! I revisited distillation and wrote up the conclusions of these fun experiments. I’m happy to get any feedback, as it will help me get better!

What’s Next? As I said, I have actually also played with KD on sequences, but I felt it would make this blog way too long, so I will put it in a separate part. I would like to show you how sequence distillation behaves, since it has some interesting findings, especially for soft on- vs. off-policy. There, I will use a simple Markov chain (think of it as a tiny language model that only looks at the last token), and besides that, I will also try a more language-like sequence problem (spoiler: I will use that old tree-based statistical model, a Probabilistic Context-Free Grammar (PCFG), and check how distillation behaves there). So, stay tuned for Part 2!

The GitHub repository will be uploaded soon, as the code is still messy... please wait for it. Thank you for understanding!



References

- Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=3zKtaqxLhW>. ← cited on pp. 2, 8, 9, 24, and 25
- Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 535–541, 2006. doi: 10.1145/1150402.1150464. ← cited on p. 3

- Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1607–1616, 2018. URL <https://proceedings.mlr.press/v80/furlanello18a.html>. ← cited on p. 3
- Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024. URL <https://arxiv.org/abs/2408.00118>. ← cited on p. 2
- Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=5h0qf7IBZZ>. ← cited on pp. 6, 9, and 24
- Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z. ← cited on pp. 2 and 3
- Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. URL <https://arxiv.org/abs/1503.02531>. ← cited on pp. 3 and 5
- Taehyeon Kim, Jaehoon Oh, NakYil Kim, Sangwook Cho, and Se-Young Yun. Comparing Kullback-Leibler divergence and mean squared error loss in knowledge distillation. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 2628–2635, 2021. doi: 10.24963/ijcai.2021/362. ← cited on p. 5
- Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, 2016. doi: 10.18653/v1/D16-1139. URL <https://aclanthology.org/D16-1139/>. ← cited on pp. 5 and 8
- Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. DistiLLM: Towards streamlined distillation for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 24872–24895, 2024. URL <https://proceedings.mlr.press/v235/ko24c.html>. ← cited on p. 25
- Kevin Lu and Thinking Machines Lab. On-policy distillation. Thinking Machines Lab: Connectionism, October 2025. URL <https://thinkingmachines.ai/blog/on-policy-distillation/>. ← cited on pp. 2 and 24
- Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, 21(132):1–62, 2020. URL <https://jmlr.org/papers/v21/19-346.html>. ← cited on p. 21
- Made Nindyatama Nityasya, Haryo Akbarianto Wibowo, Rendi Chevi, Radityo Eko Prasajo, and Alham Fikri Aji. Which student is best? a comprehensive knowledge distillation exam for task-specific BERT models, 2022. URL <https://arxiv.org/abs/2201.00558>. ← cited on p. 5
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html. ← cited on p. 3

- Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011. URL <https://proceedings.mlr.press/v15/ross11a.html>. ↔ cited on p. 8
- Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019. URL <https://arxiv.org/abs/1910.01108>. ↔ cited on p. 2
- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 2018. URL <http://incompleteideas.net/book/the-book-2nd.html>. ↔ cited on p. 8
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An instruction-following LLaMA model. GitHub repository, 2023. URL https://github.com/tatsu-lab/stanford_alpaca. ↔ cited on p. 5
- Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer, 2004. doi: 10.1007/978-0-387-21736-9. ↔ cited on p. 21
- Haoran Xin, Anhao Zhao, Ying Sun, Jin Li, Xiaoyu Shen, and Hui Xiong. Escaping the KL agreement trap in on-policy distillation, 2026. URL <https://arxiv.org/abs/2606.09471>. ↔ cited on p. 24
- Li Yuan, Francis E. H. Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3903–3911, 2020. doi: 10.1109/CVPR42600.2020.00396. ↔ cited on p. 3

SUPPLEMENTARY MATERIAL

Appendix

A	Proofs	28
B	Citation & Changelog	33

A Proofs

Don't worry, these proofs only use a few basic rules. Let's collect them first, then build one helper fact that every proof below uses.

Toolbox. We only need these:

- Log of a fraction: $\log \frac{a}{b} = \log a - \log b$.
- Log undoes exp: $\log e^z = z$.
- Derivatives: $\frac{d}{dz} e^z = e^z$, $\frac{d}{dx} \log x = \frac{1}{x}$, and the derivative of a constant is 0.
- Chain rule: $\frac{d}{dz} \log f(z) = \frac{1}{f(z)} \cdot \frac{df(z)}{dz}$.
- Product rule: $\frac{d}{dz} [f(z)g(z)] = \frac{df}{dz} g + f \frac{dg}{dz}$.

A Tiny New Symbol. We will write $\mathbb{1}[j = k]$ (“indicator”) for a number that is 1 if $j = k$ and 0 if $j \neq k$. For example, $\mathbb{1}[3 = 3] = 1$ and $\mathbb{1}[2 = 3] = 0$. It shows up naturally when we take the derivative of z_j with respect to z_k : if $j = k$, it's the derivative of z_k with respect to itself, which is 1; if $j \neq k$, then z_j is a different variable (a constant as far as z_k is concerned), so the derivative is 0. In short,

$$\frac{\partial z_j}{\partial z_k} = \mathbb{1}[j = k]. \quad (15)$$

It also has a handy property: in a sum over j , it keeps only the $j = k$ term, so $\sum_j a_j \mathbb{1}[j = k] = a_k$.

Helper: How the Softmax Reacts to One Logit. The student is $q = \text{softmax}(z)$, i.e.,

$$q_j = \frac{e^{z_j}}{S}, \quad \text{where} \quad S = \sum_i e^{z_i}$$

is just the sum that makes everything add up to 1. Let's see how $\log q_j$ changes when we nudge one logit z_k , step by step.

1. Take the log (log of a fraction, then log undoes exp, see the toolbox):

$$\log q_j = \log e^{z_j} - \log S = z_j - \log S. \quad (16)$$

2. *Derivative of the first part:* $\frac{\partial z_j}{\partial z_k} = \mathbb{1}[j = k]$, by Eq. (15).
3. *Derivative of the second part:* by the chain rule, $\frac{\partial \log S}{\partial z_k} = \frac{1}{S} \cdot \frac{\partial S}{\partial z_k}$. Inside $S = e^{z_1} + e^{z_2} + \dots$, only the term e^{z_k} depends on z_k , and its derivative is e^{z_k} . So $\frac{\partial \log S}{\partial z_k} = \frac{e^{z_k}}{S} = q_k$.
4. *Put them together:*

$$\frac{\partial \log q_j}{\partial z_k} = \mathbb{1}[j = k] - q_k. \quad (17)$$

In words: if we push up logit k , then $\log q_k$ goes up by $1 - q_k$ (its own class gains), and every other $\log q_j$ goes down by q_k (they lose, since everything must still add up to 1). We will also need the version without the log. By the chain rule, $\frac{\partial \log q_j}{\partial z_k} = \frac{1}{q_j} \frac{\partial q_j}{\partial z_k}$, so multiplying both sides by q_j :

$$\frac{\partial q_j}{\partial z_k} = q_j(\mathbb{1}[j = k] - q_k). \quad (18)$$

↔ Back to Lemma 7.1 on p. 17

Lemma 7.1 (Forward KL gradient) *For $q = \text{softmax}(z)$,*

$$\frac{\partial}{\partial z_k} \text{KL}(p \parallel q) = q_k - p_k.$$

Proof. Write the KL as two sums: $\text{KL}(p \parallel q) = \sum_j p_j \log \frac{p_j}{q_j} = \sum_j p_j \log p_j - \sum_j p_j \log q_j$. The first sum only has the teacher in it, so its derivative is 0. For the second, the teacher's p_j are just constants, so we only differentiate $\log q_j$, using Eq. (17):

$$\begin{aligned} \frac{\partial}{\partial z_k} \text{KL}(p \parallel q) &= - \sum_j p_j \frac{\partial \log q_j}{\partial z_k} = - \sum_j p_j (\mathbb{1}[j = k] - q_k) \\ &= - \underbrace{\sum_j p_j \mathbb{1}[j = k]}_{= p_k \text{ (only } j=k \text{ survives)}} + q_k \underbrace{\sum_j p_j}_{= 1} = q_k - p_k. \end{aligned} \quad \blacksquare$$

↔ Back to Lemma 7.2 on p. 17

Lemma 7.2 (Reverse KL gradient) *For $q = \text{softmax}(z)$,*

$$\frac{\partial}{\partial z_k} \text{KL}(q \parallel p) = q_k \left(\log \frac{q_k}{p_k} - \text{KL}(q \parallel p) \right).$$

Proof. Now the student appears twice in each term: $\text{KL}(q \parallel p) = \sum_j q_j (\log q_j - \log p_j)$. So, by the product rule, each term gives two pieces:

$$\frac{\partial}{\partial z_k} \text{KL}(q \parallel p) = \underbrace{\sum_j \frac{\partial q_j}{\partial z_k} (\log q_j - \log p_j)}_{\text{piece A}} + \underbrace{\sum_j q_j \frac{\partial \log q_j}{\partial z_k}}_{\text{piece B}}$$

(the teacher's $\log p_j$ is a constant, so it has no derivative of its own).

Piece B is zero. Using Eq. (17): $\sum_j q_j (\mathbb{1}[j = k] - q_k) = \sum_j q_j \mathbb{1}[j = k] - q_k \sum_j q_j = q_k - q_k \cdot 1 = 0$.

Piece A. Using Eq. (18), and writing $r_j = \log q_j - \log p_j$ to keep it short:

$$\sum_j q_j (\mathbb{1}[j = k] - q_k) r_j = \underbrace{\sum_j q_j r_j \mathbb{1}[j = k]}_{\text{only } j=k \text{ survives}} - q_k \sum_j q_j r_j = q_k r_k - q_k \text{KL}(q \parallel p),$$

because $\sum_j q_j r_j = \sum_j q_j (\log q_j - \log p_j)$ is exactly the definition of $\text{KL}(q \parallel p)$. Finally, $r_k = \log q_k - \log p_k = \log \frac{q_k}{p_k}$, so

$$\frac{\partial}{\partial z_k} \text{KL}(q \parallel p) = q_k \left(\log \frac{q_k}{p_k} - \text{KL}(q \parallel p) \right). \quad \blacksquare$$

↔ Back to Corollary 7.3 on p. 17

Corollary 7.3 (Forward is bounded, reverse is not) *Let $q = \text{softmax}(z)$. Then:*

- (i) Forward KL is bounded: for any teacher p and student q , $\left| \frac{\partial}{\partial z_k} \text{KL}(p \parallel q) \right| \leq 1$.
- (ii) Reverse KL is not: keep the student fixed with $0 < q_k < 1$, and let the teacher's p_k go to 0. Then $\frac{\partial}{\partial z_k} \text{KL}(q \parallel p) \rightarrow +\infty$.
- (iii) Reverse KL ignores classes the student doesn't visit: if the teacher has $p_j > 0$ for every class and the student's q_k goes to 0, then $\frac{\partial}{\partial z_k} \text{KL}(q \parallel p) \rightarrow 0$, no matter how big p_k is.

Proof. All three parts come straight from Lemmas 7.1 and 7.2.

(i) By Lemma 7.1, the gradient is $q_k - p_k$. Both q_k and p_k are probabilities, so both are between 0 and 1, and the difference of two numbers between 0 and 1 is always between -1 and 1 .

(ii) Call the teacher's probability on class k simply $p_k = \varepsilon$, and let the other classes keep their relative sizes: $p_j = (1-\varepsilon) r_j$ for $j \neq k$, where the r_j are fixed positive numbers that sum to 1. By Lemma 7.2, the gradient is $q_k \cdot B$ with $B = \log q_k - \log p_k - \text{KL}(q \parallel p)$. Let's expand B . First, write out the KL, splitting it into the class k term and the rest:

$$B = \log q_k - \log \varepsilon - \sum_j q_j \log q_j + q_k \log \varepsilon + \sum_{j \neq k} q_j \log ((1-\varepsilon)r_j).$$

Now collect the two $\log \varepsilon$ terms ($-\log \varepsilon + q_k \log \varepsilon = (1 - q_k) \log \frac{1}{\varepsilon}$) and put everything else in one box C :

$$B = (1 - q_k) \log \frac{1}{\varepsilon} + C, \quad C = \log q_k - \sum_j q_j \log q_j + \sum_{j \neq k} q_j \log r_j + (1 - q_k) \log(1 - \varepsilon),$$

where, for the last term of C , we used $\sum_{j \neq k} q_j = 1 - q_k$. Since the student is fixed, C stays a normal, finite number as $\varepsilon \rightarrow 0$ (and $\log(1 - \varepsilon) \rightarrow 0$). As $\varepsilon \rightarrow 0$, $\log \frac{1}{\varepsilon} \rightarrow \infty$,

so the gradient $q_k B \approx q_k(1 - q_k) \log \frac{1}{\epsilon}$ grows without limit (as long as $0 < q_k < 1$, so that $q_k(1 - q_k) > 0$). So there is no number that bounds it.

(iii) Now the teacher is fixed with $p_j > 0$ everywhere, so $\text{KL}(q \parallel p)$ stays a normal, finite number. Write the gradient from Lemma 7.2 as

$$q_k \log q_k - q_k (\log p_k + \text{KL}(q \parallel p)).$$

As $q_k \rightarrow 0$, the second part goes to 0 (a finite number times something going to 0). The first part also goes to 0: $\log q_k$ goes to $-\infty$, but q_k goes to 0 much faster, so $q_k \log q_k \rightarrow 0$ (e.g., $0.01 \ln 0.01 \approx -0.046$ and $0.0001 \ln 0.0001 \approx -0.0009$). So the whole gradient goes to 0, whatever p_k is. ■

↔ Back to Proposition 7.4 on p. 18

Proposition 7.4 (Forward KL has one best answer) *For our one-hill student (Eq. (7)), forward KL has only one resting point, no matter where the student starts. At that point, the student copies the teacher's mean and spread: $\mathbb{E}_{q_\theta}[x] = \mathbb{E}_p[x]$ and $\mathbb{E}_{q_\theta}[x^2] = \mathbb{E}_p[x^2]$.*

Proof. We go in four steps. The idea: rewrite the student so the loss becomes “bowl-shaped” (a bowl has only one bottom, so there is only one place to rest), and then find where the bottom is.

Step 1: rewrite the student's logit. Expand the square in the logit $z_k = -\frac{(k - \mu^s)^2}{2(\sigma^s)^2}$, using $(k - \mu^s)^2 = k^2 - 2k\mu^s + (\mu^s)^2$:

$$z_k = \underbrace{\frac{\mu^s}{(\sigma^s)^2}}_{\eta_1} k + \underbrace{\left(-\frac{1}{2(\sigma^s)^2}\right)}_{\eta_2} k^2 - \frac{(\mu^s)^2}{2(\sigma^s)^2}. \quad (19)$$

The last term is the same for every class k , and adding the same number to all logits doesn't change a softmax (it cancels between the top and the bottom of the fraction). So we can drop it, and the logit is simply

$$z_k = \eta_1 k + \eta_2 k^2. \quad (20)$$

Note that η_2 is always negative.

Step 2: the loss is bowl-shaped in (η_1, η_2) . Let's start from the normal forward KL and rewrite it line by line (the reason for each line is on the right):

$$\begin{aligned}
 \text{KL}(p \parallel q) &= \sum_k p_k \log \frac{p_k}{q_k} && \text{(definition)} \\
 &= \sum_k p_k \log p_k - \sum_k p_k \log q_k && \text{(log of a fraction)} \\
 &= \text{const} - \sum_k p_k (z_k - \log S) && \text{(Eq. (16))} \\
 &= \text{const} - \sum_k p_k z_k + \log S \sum_k p_k && \text{(split; } \log S \text{ has no } k) \\
 &= \text{const} - \sum_k p_k z_k + \log S && \left(\sum_k p_k = 1 \right) \\
 &= \text{const} - \sum_k p_k (\eta_1 k + \eta_2 k^2) + \log \sum_k e^{\eta_1 k + \eta_2 k^2} && \text{(Eq. (20))} \\
 &= \text{const} - \left(\eta_1 \mathbb{E}_p[x] + \eta_2 \mathbb{E}_p[x^2] \right) + \log \sum_k e^{\eta_1 k + \eta_2 k^2} && \left(\sum_k p_k k = \mathbb{E}_p[x] \right).
 \end{aligned}$$

Here, Eq. (16) is $\log q_k = z_k - \log S$ with $S = \sum_k e^{z_k}$, and “const” is the first sum, the teacher-only part $\sum_k p_k \log p_k$, which doesn't change when the student changes. The middle part is a straight line in (η_1, η_2) , which doesn't bend at all. The last part, a “log of a sum of exps”, is a well-known bowl-shaped (convex) function.^k A straight line plus a bowl is still a bowl, so the loss has only one bottom, and there are no traps.

^k Its curvature is the variance of $\eta_1 k + \eta_2 k^2$ under the student, which is never negative, and it's strictly positive here because k and k^2 really vary across the 60 classes.

Step 3: where is the bottom? By Lemma 7.1, nudging logit z_k changes the loss by $q_k - p_k$. Since $z_k = \eta_1 k + \eta_2 k^2$ (Eq. (20)), nudging η_1 nudges every z_k by k , and nudging η_2 nudges every z_k by k^2 (chain rule). So

$$\frac{\partial \text{KL}}{\partial \eta_1} = \sum_k (q_k - p_k) k = \mathbb{E}_q[x] - \mathbb{E}_p[x], \quad \frac{\partial \text{KL}}{\partial \eta_2} = \sum_k (q_k - p_k) k^2 = \mathbb{E}_q[x^2] - \mathbb{E}_p[x^2].$$

At the bottom, both are zero, i.e., the student's mean and $\mathbb{E}[x^2]$ match the teacher's.

Step 4: back to μ^s and σ^s . We actually train μ^s and $\log \sigma^s$, not η_1 and η_2 . But each pair gives exactly one of the other: from Eq. (19), $\sigma^s = \sqrt{-1/(2\eta_2)}$ and $\mu^s = \eta_1 (\sigma^s)^2$. It's just a change of “units”, like Celsius and Fahrenheit, so it can't create new resting points. One bottom in (η_1, η_2) means one resting point in (μ^s, σ^s) . ■

Remark A.1 Two things break this. If the student is a *mix* of hills (Eq. (13)), its logits are no longer of the simple form $\eta_1 k + \eta_2 k^2$ (Eq. (20)), so the loss is no longer bowl-shaped, and forward KL can also get stuck (as in §7.4). And reverse KL is never of this nice form, because the average is taken over the student itself (Lemma 7.2).

B Citation & Changelog

Cite this paper

```
@misc{wibowo2026playdistillation,  
  title      = {Play Around with Distillation, Part 1: Forward vs. Reverse {KL},  
                On vs. Off Policy, and Hard vs. Soft!},  
  author     = {Wibowo, Haryo Akbarianto},  
  year       = {2026},  
  month      = oct,  
  howpublished = {Blog post}  
}
```

Changelog

v1 Oct 2026 Initial release.