Nevertheless, the distributions in these two plots are different, and thus we should find a metric that mirrors this in its definition.

Properties of the KL divergence: Observe that the KL divergence, by definition, satisfies:

$$\begin{aligned} D_{\mathrm{KL}}\left(p_1(\cdot)||p_2(\cdot)\right) &= \mathbb{E}_{p_1(\cdot)}\left[\log \tfrac{p_1(\cdot)}{p_2(\cdot)}\right] \\ &= \mathbb{E}_{p_1(\cdot)}\left[\log p_1(\cdot)\right] - \mathbb{E}_{p_1(\cdot)}\left[\log p_2(\cdot)\right]. \end{aligned}$$

To help our discussion, and make connections with our problem so far, we use $p_1(\cdot) = p(x|\theta)$ and $p_2(\cdot) = p(x|\theta')$. Then, the gradient of KL with respect to θ' satisfies:

$$\begin{aligned} &\nabla_{\theta'}\left(D_{\mathrm{KL}}\left(p(x|\theta)||p(x|\theta')\right)\right) \\ &= \nabla_{\theta'}\mathbb{E}_{p(x|\theta)}[\log p(x|\theta)] - \nabla_{\theta'}\mathbb{E}_{p(x|\theta)}[\log p(x|\theta')] \\ &= -\mathbb{E}_{p(x|\theta)}[\nabla_{\theta'}\log p(x|\theta')] \\ &= -\int p(x|\theta)\nabla_{\theta'}\log p(x|\theta')\,\mathrm{d}x. \end{aligned}$$

and the second derivative satisfies:

$$\nabla^2_{\theta'}\left(D_{\mathrm{KL}}\left(p(x|\theta)||p(x|\theta')\right)\right) = -\int p(x|\theta)\,\nabla^2_{\theta'}\log p(x|\theta')\,\mathrm{d}x$$

Then, the Hessian evaluated at $\theta' = \theta$ is:

$$\begin{aligned} \mathrm{H}_{D_{\mathrm{KL}}(p(x|\theta)||p(x|\theta'))} &= -\int p(x|\theta)\ \nabla^2_{\theta'}\log p(x|\theta')\big|_{\theta'=\theta}\,\mathrm{d}x \\ &= -\int p(x|\theta)\,\mathrm{H}_{\log p(x|\theta)}\,\mathrm{d}x \\ &= -\mathop{\mathbb{E}}_{p(x|\theta)}[\mathrm{H}_{\log p(x|\theta)}] \\ &= F, \end{aligned}$$

which is what we have shown above; i.e., the expected Hessian of the log function is the Fisher information matrix.

2nd-oder Taylor expansion of KL divergence and natural gradient: Let us now connect the dots. Following similar reasoning to classical optimization, given an objective function, we can approximate locally the objective with its second-order Taylor approximation (which involves both the gradient and the Hessian information), and then locally minimize that approximation; then, we iterate.

The second-order approximation of the KL divergence metric satisfies:

$$\begin{aligned} &D_{\mathrm{KL}}\left(p(x|\theta)||p(x|\theta+d)\right) \\ &\approx D_{\mathrm{KL}}\left(p(x|\theta)||p(x|\theta)\right) + \left\langle \nabla D_{\mathrm{KL}}\left(p(x|\theta)||p(x|\theta)\right), d\right\rangle + \frac{1}{2}\left\langle Fd, d\right\rangle \\ &= \frac{1}{2}\left\langle Fd, d\right\rangle. \end{aligned}$$

Similar to the Euclidean case, we seek for an update vector d that minimizes the loss function $\mathcal{L}(\theta)$ in the *distribution space*. Analogously to steepest descent:

$$d^\star = \mathrm{argmin}_{D_{\mathrm{KL}}(p(x|\theta)||p(x|\theta+d))=c}\mathcal{L}(\theta+d),$$

where c is some constant. Compare this with the Euclidean case where:

$$d^\star = \mathrm{argmin}_{\|d\|_2\le\epsilon}\mathcal{L}(\theta+d).$$

The purpose of fixing the KL-divergence to some constant is to make sure that we move along the space of distributions with constant speed, regardless the curvature.

How do we solve this part? If we write the above minimization in Lagrangian form (*we have not talked about Lagrange multipliers yet, but accept this for the moment*), we get:

$$\begin{aligned} d^\star &= \arg\min_d\left\{\mathcal{L}(\theta+d) + \lambda\cdot\left(D_{\mathrm{KL}}\left(p(x|\theta)||p(x|\theta+d)\right) - c\right)\right\} \\ &\approx \arg\min_d\left\{\mathcal{L}(\theta) + \left\langle\nabla\mathcal{L}(\theta),\ d\right\rangle + \frac{1}{2}\lambda\left\langle Fd,\ d\right\rangle - \lambda c\right\}. \end{aligned}$$

To solve this minimization, we set its derivative with respect to d to zero; this will lead to the solution:

$$d = -\tfrac{1}{\lambda}F^{-1}\nabla\mathcal{L}(\theta).$$

(The λ constant can be absorbed in the definition of F.)

The above lead to the definition of the *natural gradient* descent method:

- **Repeat:**
  1. Compute the gradient $\nabla\mathcal{L}(\theta_t)$.
  2. Compute the Fisher information matrix F_t.
  3. Compute the natural gradient direction: $d_t = F_t^{-1}\nabla\mathcal{L}(\theta_t)$.
  4. For a step size η, compute $\theta_{t+1} = \theta_t - \eta d_t$.

Take-away messages:

- The natural gradient descent is a generalization of the Newton's method, as there are cases where from the natural gradient descent we can obtain the Newton's iteration.
- Remember that, afterall, the Fisher information matrix is computed per iteration, and inverted; and it turns out that on expectation it corresponds to the expected Hessian of the objective.
- How do we implement the natural gradient method in reality? Remember the definition of the Fisher information:

$$F = -\mathop{\mathbb{E}}_{p(x|\theta)}[H_{\log p(x|\theta)}].$$

In realistic scenarios, we do not have access to the distribution $p(x|\theta)$, but rather have data that come from that distribution. In that case, we refer to the *empirical* Fisher information matrix, defined as:

$$F = \tfrac{1}{n}\sum_{i=1}^{n}\nabla\log p(x_i|\theta)\cdot\nabla\log p(x_i|\theta)^\top$$

where $\{x_i\}_{i=1}^n$ denote the training set of examples. In that case, the main recursion of natural gradient descent becomes:

$$\theta_{t+1} = \theta_t - \eta\left(\tfrac{1}{n}\sum_{i=1}^{n}\nabla\log p(x_i|\theta_t)\cdot\nabla\log p(x_i|\theta_t)^\top\right)^{-1}\cdot\nabla\widehat{\mathcal{L}}(\theta_t),$$

where also the objective and its gradient are evaluated in their empirical form:

$$\nabla\widehat{\mathcal{L}}(\theta_t) := \tfrac{1}{n}\sum_{i=1}^{n}\nabla\log p(x_i|\theta_t).$$

- The main reason we studied natural gradient descent is to motivate our discussion later on, regarding algorithms in training neural networks. The empirical Fisher information matrix appears in almost all modern algorithms in ML, usually further approximated to be easily computed. E.g., one way to get around computing the exact empirical Fisher information matrix is to constrain it to be diagonal matrix; a technique that is heavily used in algorithms such as AdaGrad, AdaDelta, RMSprop, Adam, AMSGrad, Yogi, etc. In other words, in the mostly used algorithms in neural network training.