**Variational Inference: The Basics**

[[Hylke C. Donker](https://medium.com/@hylke.donker?source=post_page-----f70ac511bcea--------------------------------)](https://medium.com/@hylke.donker?source=post_page-----f70ac511bcea--------------------------------)

[[Towards Data Science](https://towardsdatascience.com/?source=post_page-----f70ac511bcea--------------------------------)](https://towardsdatascience.com/?source=post_page-----f70ac511bcea--------------------------------)

[Hylke C. Donker](https://medium.com/@hylke.donker?source=post_page-----f70ac511bcea--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fea1bfe4db7a8&operation=register&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fvariational-inference-the-basics-f70ac511bcea&user=Hylke+C.+Donker&userId=ea1bfe4db7a8&source=post_page-ea1bfe4db7a8----f70ac511bcea---------------------post_header-----------)

Published in

[Towards Data Science](https://towardsdatascience.com/?source=post_page-----f70ac511bcea--------------------------------)

·

9 min read

·

Jun 16

235

1

We live in the era of quantification. But rigorous quantification is easier said then done. In complex systems such as biology, data can be difficult and expensive to collect. While in high stakes applications, such as in medicine and finance, it is crucial to account for uncertainty. Variational inference — a methodology at the forefront of AI research — is a way to address these aspects.

This tutorial introduces you to the basics: the when, why, and how of variational inference.

**When is variational inference useful?**

Variational inference is appealing in the following three closely related usecases:

1. if you have little data (i.e., low number of observations),

2. you care about uncertainty,

3. for generative modelling.

We will touch upon each usecase in our worked example.

**1. Variational inference with little data**



Fig. 1: Variational inference allows you to trade-of domain knowledge with information from examples. Image by Author.

Sometimes, data collection is expensive. For example, DNA or RNA measurements can easily cost a few thousand euros per observation. In this case, you can hardcode domain knowledge in lieu of extra samples. Variational inference can help to systematically “dial down” the domain knowledge as you gather more examples, and more heavily rely on the data (Fig. 1).

**2. Variational inference for uncertainty**

For safety critical applications, such as in finance and healthcare, uncertainty is important. Uncertainty can affect all aspects of the model, most obviously the predicted output. Less obvious are the model’s parameters (e.g., weights and biases). Instead of the usual arrays of numbers — the weights and biases — you can endow the parameters with a distribution to make them fuzzy. Variational inference allows you to infer the range(s) of reasonable values.

**3. Variational inference for generative modelling**

Generative models provide a complete specification how the data was generated. For example, how to generate an image of a cat or a dog. Usually, there is a latent representation ***z*** that carries semantic meaning (e.g., ***z*** descibes a siamese cat). Through a set of (non-linear) transformations and sampling steps, ***z*** is transformed into the actual image ***x*** (e.g., the pixel values of the siamese cat). Variational inference is a way to infer, and sample from, the latent semantic space ***z***. A well known example is the [variational auto encoder](https://en.wikipedia.org/wiki/Variational_autoencode).

**What is variational inference?**

At its core, variational inference is a Bayesian undertaking [1]. In the Bayesian perspective, you still let the machine learn from the data, as usual. What is different, is that you give the model a hint ([a prior](https://en.wikipedia.org/wiki/Prior_probability)) and allow the solution ([the posterior](https://en.wikipedia.org/wiki/Posterior_probability)) to be more fuzzy. More concretely, say you have a training set ***X =*[*x***₁, ***x***₂,..,***x****ₘ*]ᵗ of *m* examples. We use Bayes’ theorem:

*p*(***Θ***|***X***)***=****p*(***X***|***Θ***)*p*(***Θ***) /*p*(***X****),*

to infer a range — a distribution — of solutions ***Θ***. Contrast this with the conventional machine learning approach, where we minimise a loss *ℒ(****Θ, X****) =*ln *p*(***X***|***Θ***) to find one specific solution ***Θ****.* Bayesian inference revolves around finding a way to determine *p*(***Θ***|***X***): the *posterior* distribution of the parameters ***Θ*** given the training set ***X***. In general, this is a difficult problem. In practice, two ways are used to solve for *p*(***Θ***|***X***): (i) using simulation ([Markov chain Monte Carlo](https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo)) or (ii) through optimisation.

Variational inference is about option (ii).

**The evidence lower bound (ELBO)**

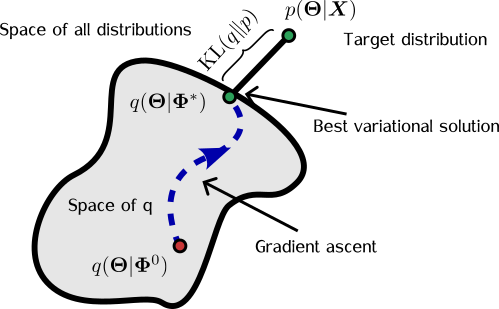


Fig. 2: **Sketch of variational inference.** We look for a distribution q(Θ) that is close to p(Θ|X). Image by Author.

The idea behind variational inference is to look for a distribution *q*(***Θ***) that is a stand-in (a surrogate) for *p*(***Θ***|***X***). We then try to make *q*[***Θ|Φ***(***X***)] look similar to *p*(***Θ***|***X***) by changing the values of ***Φ***(Fig. 2). This is done by maximising the evidence lower bound (ELBO):

*ℒ*(***Φ***)*=*E[ln*p*(***X***,***Θ***) — ln *q*(***Θ|Φ)***],

where the expectation E[·] is taken over *q*(***Θ|Φ***). (Note that ***Φ***implicitly depends on the dataset ***X***, but for notational convenience we'll drop the explicit dependence.)

For gradient based optimisation of *ℒ* it looks, at first sight, like we have to be careful when taking derivatives (with respect to ***Φ***) because of the dependence of E[·] on *q*(***Θ|Φ***). Fortunately, autograd packages like[JAX](https://jax.readthedocs.io/en/latest/) support reparameterisation tricks [2] that allow you to directly take derivatives from random samples (e.g., of the gamma distribution) instead of relying on high variance black box variational approaches [3]. Long story short: estimate ∇ℒ(Φ) with a batch **[*Θ***₁, ***Θ***₂,..] ~ *q*(***Θ|Φ***) and let your autograd package worry about the details.

**Variational inference from scratch**

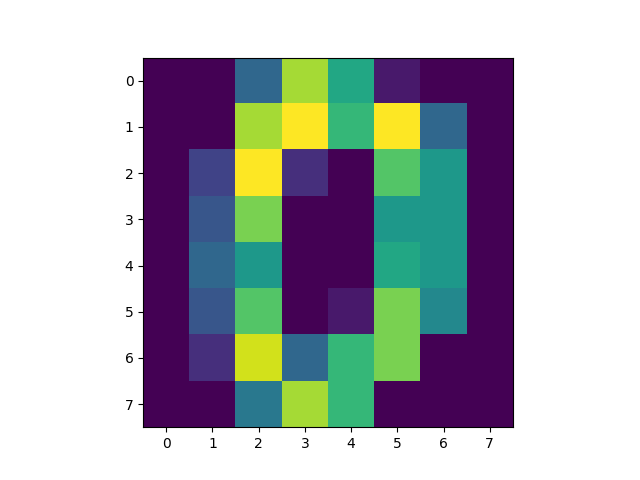


Fig. 3: Example image of a handwritten “zero” from sci-kit learn’s digits dataset. Image by Author.

To solidify our understanding let us implement variational inference from scratch using [JAX](https://jax.readthedocs.io/en/latest/). In this example, you will train a generative model on handwritten digits from [sci-kit learn](https://scikit-learn.org/). You can follow along with the [Colab notebook](https://colab.research.google.com/drive/1NPZfRkhrFfX8flbUJqfgKZ2z1w5JXs1R?usp=sharing).

To keep it simple, we will only analyse the digit “zero”.

from sklearn import datasets  
  
digits = datasets.load\_digits()  
is\_zero = digits.target == 0  
X\_train = digits.images[is\_zero]  
  
# Flatten image grid to a vector.  
n\_pixels = 64 # 8-by-8.  
X\_train = X\_train.reshape((-1, n\_pixels))

Each image is a 8-by-8 array of discrete pixel values ranging from 0–16. Since the pixels are count data, let’s model the pixels, ***x***, using the [Poisson distribution](https://en.wikipedia.org/wiki/Poisson_distribution) with a gamma *prior* for the rate ***Θ***. The rate ***Θ*** determines the average intensity of the pixels. Thus, the *joint distribution* is given by:

*p*(***x***,***Θ***)***=***Poisson(***x***|***Θ***)Gamma(***Θ***|***a***, ***b***),

where ***a*** and ***b*** are the shape and rate of the [gamma distribution](https://en.wikipedia.org/wiki/Gamma_distribution).

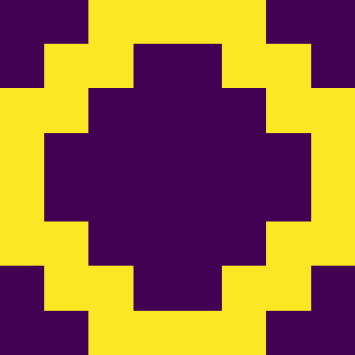


Fig. 4: Domain knowledge of the digit “zero” is used as prior. Image by Author.

The prior — in this case, Gamma(***Θ***|***a***, ***b***) — is the place where you infuse your domain knowledge (usecase 1.). For example, you may have some idea what the “average” digit zero looks like (Fig. 4). You can use this *a priori* information to guide your choice of ***a***and ***b***. To use Fig. 4 as prior information — let’s call it ***x***₀ — and weigh its importance as two examples, then set ***a***= 2***x***₀; **b**= 2.

Written down in Python this looks like:

import jax.numpy as jnp  
import jax.scipy as jsp  
  
# Hyperparameters of the model.  
a = 2. \* x\_domain\_knowledge  
b = 2.  
  
def log\_joint(θ):  
 log\_likelihood = jnp.sum(jsp.stats.gamma.logpdf(θ, a, scale=1./b))  
 log\_likelihood += jnp.sum(jsp.stats.poisson.logpmf(X\_train, θ))  
 return log\_likelihood

Note that we’ve used the [JAX](https://jax.readthedocs.io/) implementation of numpy and scipy, so that we can take derivatives.

Next, we need to choose a surrogate distribution *q*(***Θ|Φ***). To remind you, our goal is to change ***Φ*** so that the surrogate distribution *q*(***Θ|Φ***) matches *p*(***Θ|X)***. So, the choice of *q*(***Θ***) determines the level of approximation (we suppress the dependence on ***Φ*** where context permits). For illustration purposes, lets choose a variational distribution that is composed of (a product of) gamma’s:

*q*(***Θ|Φ***) = Gamma(***Θ***|***α***,***β***),

where we used the shorthand ***Φ***= {***α***,***β***}.

Next, to implement the evidence lower bound *ℒ*(***Φ***) = E[ln*p*(***X***,***Θ***) — ln *q*(***Θ|Φ***)], first write down the term inside the expectation brackets:

@partial(vmap, in\_axes=(0, None, None))  
def evidence\_lower\_bound(θ\_i, alpha, inv\_beta):  
 elbo = log\_joint(θ\_i) - jnp.sum(jsp.stats.gamma.logpdf(θ\_i, alpha, scale=inv\_beta))  
 return elbo

Here, we used JAX’s [vmap](https://jax.readthedocs.io/en/latest/_autosummary/jax.vmap.html) to vectorise the function so that we can run it on a batch **[*Θ***₁, ***Θ***₂,..,***Θ***₁₂₈]ᵗ.

To complete the implementation of *ℒ*(***Φ***), we average the above function over realisations of the variational distribution ***Θ****ᵢ* ~ *q*(***Θ***):

def loss(Φ: dict, key):  
 """Stochastic estimate of evidence lower bound."""  
 alpha = jnp.exp(Φ['log\_alpha'])  
 inv\_beta = jnp.exp(-Φ['log\_beta'])  
  
 # Sample a batch from variational distribution q.  
 batch\_size = 128  
 batch\_shape = [batch\_size, n\_pixels]  
 θ\_samples = random.gamma(key, alpha , shape=batch\_shape) \* inv\_beta  
   
 # Compute Monte Carlo estimate of evidence lower bound.  
 elbo\_loss = jnp.mean(evidence\_lower\_bound(θ\_samples, alpha, inv\_beta))  
   
 # Turn elbo into a loss.  
 return -elbo\_loss

A few things to notice here about the arguments:

* We’ve packed ***Φ*** as a dictionary (or technically, a [pytree](https://jax.readthedocs.io/en/latest/pytrees.html)) containing ln(***α***), and ln(***β***). This trick guarantees that ***α***>0 and ***β***>0 — a requirement imposed by the gamma distribution — during optimisation.
* The *loss* is a random estimate of the ELBO. In JAX, we need a new pseudo random number generator (PRNG) *key* every time we sample. In this case, we use *key* to sample **[*Θ***₁, ***Θ***₂,..,***Θ***₁₂₈]ᵗ.

This completes the specification of the model *p*(***x***,***Θ)***, the variational distribution *q*(***Θ***), and the loss *ℒ*(***Φ***).

**Model training**

Next, we minimise the loss *ℒ*(***Φ***)by varying ***Φ***= {***α***,***β***}so that *q*(***Θ|Φ***) matches the posterior *p*(***Θ***|***X***). How? Using old fashioned gradient descent! For convenience, we use the Adam optimiser from [Optax](https://github.com/deepmind/optax) and initialise the parameters with the prior ***α = a***, and ***β = b*** [remember, the prior wasGamma(***Θ***|***a***, ***b***) and codified our domain knowledge].

# Initialise parameters using prior.  
Φ = {  
 'log\_alpha': jnp.log(a),  
 'log\_beta': jnp.full(fill\_value=jnp.log(b), shape=[n\_pixels]),  
}  
  
loss\_val\_grad = jit(jax.value\_and\_grad(loss))  
optimiser = optax.adam(learning\_rate=0.2)  
opt\_state = optimiser.init(Φ)

Here, we use [*value\_and\_grad*](https://jax.readthedocs.io/en/latest/_autosummary/jax.value_and_grad.html) to simultaneously evaluate the ELBO and its derivative. Convenient for monitoring convergence! We then just-in-time compile the resulting function(with [*jit*](https://jax.readthedocs.io/en/latest/jax-101/02-jitting.html)*)*to make it snappy.

Finally, we’Il train the model for 5000 steps. Since *loss* is random, for each evaluation we need to supply it a pseudo random number generator (PRNG) key. We do this by allocating 5000 keys with [random.split](https://jax.readthedocs.io/en/latest/_autosummary/jax.random.split.html).

n\_iter = 5\_000  
keys = random.split(random.PRNGKey(42), num=n\_iter)  
  
for i, key in enumerate(keys):  
 elbo, grads = loss\_val\_grad(Φ, key)  
 updates, opt\_state = optimiser.update(grads, opt\_state)  
 Φ = optax.apply\_updates(Φ, updates)

Congrats! You’ve succesfully trained your first model using variational inference!

You can access the notebook with the full code [here on Colab](https://colab.research.google.com/drive/1NPZfRkhrFfX8flbUJqfgKZ2z1w5JXs1R?usp=sharing).

**Results**

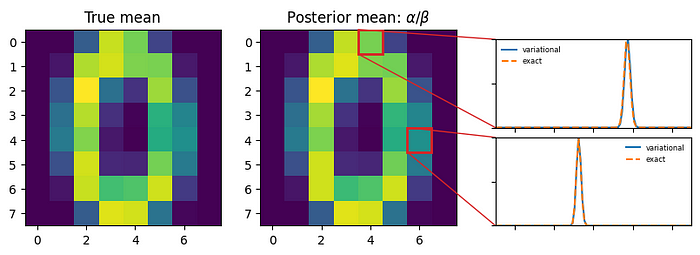


Fig. 5: Comparison of variational distribution with exact posterior distribution. Image by Author.

Let’s take a step back and appreciate what we’ve built (Fig. 5). For each pixel, the surrogate *q*(***Θ***) describes the uncertainty about the average pixel intensity (usecase 2.). In particular, our choice of *q*(***Θ***) captures two complementary elements:

* The typical pixel intensity.
* How much the intensity varies from image to image (the variability).

It turns out that the joint distribution *p*(***x***,***Θ***) we chose has an exact solution:

*p*(***Θ|X) =***Gamma(***Θ***|***a +***Σ***x***ᵢ, *m* + ***b***),

where *m* are the number of samples in the training set ***X***. Here, we see explicitly how the domain knowledge—codified in **a** and **b**— is dialed down as we gather more examples ***x***ᵢ.

We can easily compare the learned shape ***α*** and rate ***β***with the true values ***a +***Σ***x***ᵢ and *m* + ***b***. In Fig. 5 we compare the distributions — *q*(***Θ|Φ***) versus *p*(***Θ|X) —***for two specific pixels. Lo and behold, a perfect match!

**Bonus: generating synthetic images**

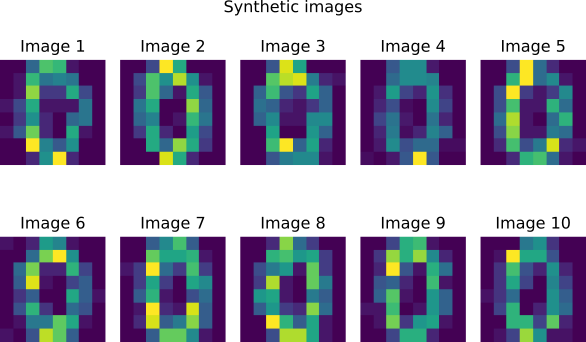


Fig. 6: Synthetically generated images using variational inference. Image by Author.

Variational inference is great for generative modelling (usecase 3.). With the stand-in posterior *q*(***Θ***) in hand, generating new synthetic images is trivial. The two steps are:

* Sample pixel intensities ***Θ ~****q*(***Θ***).

# Extract parameters of q.  
alpha = jnp.exp(Φ['log\_alpha'])  
inv\_beta = jnp.exp(-Φ['log\_beta'])  
  
# 1) Generate pixel-level intensities for 10 images.  
key\_θ, key\_x = random.split(key)  
m\_new\_images = 10  
new\_batch\_shape = [m\_new\_images, n\_pixels]  
θ\_samples = random.gamma(key\_θ, alpha , shape=new\_batch\_shape) \* inv\_beta

* Sample images using ***x*** ~ Poisson(***x***|***Θ***).

# 2) Sample image from intensities.  
X\_synthetic = random.poisson(key\_x, θ\_samples)

You can see the result in Fig. 6. Notice that the “zero” character is slightly less sharp than expected. This was part of our modelling assumptions: we modelled the pixels as mutually independent rather than correlated. To account for pixel correlations, you can expand the model to cluster pixel intensities: this is called Poisson factorisation [4].

**Summary**

In this tutorial, we introduced the basics of variational inference and applied it to a toy example: learning a handwritten digit zero. Thanks to autograd, implementing variational inference from scratch takes only a few lines of Python.

Variational inference is particularly powerful if you have little data. We saw how to infuse and trade-of domain knowledge with information from the data. The inferred surrogate distribution *q*(***Θ***) gives a “fuzzy” representation of the model parameters, instead of a fixed value. This is ideal if you are in a high-stakes application where uncertainty is important! Finally, we demonstrated generative modelling. Generating synthetic samples is easy once you can sample from *q*(***Θ***).

In summary, by harnessing the power of variational inference, we can tackle complex problems, enabling us to make informed decisions, quantify uncertainties, and ultimately unlock the true potential of data science.

**Acknowledgements**

I would like to thank Dorien Neijzen and

[Martin Banchero](https://medium.com/u/3c052f2ab36b?source=post_page-----f70ac511bcea--------------------------------)

 for proofreading.

**References:**

[1] Blei, David M., Alp Kucukelbir, and Jon D. McAuliffe. “[Variational inference: A review for statisticians.](https://doi.org/10.1080/01621459.2017.1285773)” *Journal of the American statistical Association* 112.518 (2017): 859–877.

[2] Figurnov, Mikhail, Shakir Mohamed, and Andriy Mnih. “[Implicit reparameterization gradients.](https://proceedings.neurips.cc/paper/2018/file/92c8c96e4c37100777c7190b76d28233-Paper.pdf)” Advances in neural information processing systems 31 (2018).

[3] Ranganath, Rajesh, Sean Gerrish, and David Blei. “[Black box variational inference](http://proceedings.mlr.press/v33/ranganath14.pdf).” *Artificial intelligence and statistics*. PMLR, 2014.

[4] Gopalan, Prem, Jake M. Hofman, and David M. Blei. “[Scalable recommendation with poisson factorization.](https://arxiv.org/abs/1311.1704)” *arXiv preprint arXiv:1311.1704* (2013).