Black-Box Stochastic Variational Inference in Five Lines of Python

David Duvenaud
dduvenaud@seas.harvard.edu
Harvard University

Ryan P. Adams
rpa@seas.harvard.edu
Harvard University

Abstract

Several large software engineering projects have been undertaken to support black-box inference methods. In contrast, we emphasize how easy it is to construct scalable and easy-to-use automatic inference methods using only automatic differentiation. We present a small function which computes stochastic gradients of the evidence lower bound for any differentiable posterior. As an example, we perform stochastic variational inference in a deep Bayesian neural network.

1 Introduction

Automatic variational inference procedures are maturing. Ranganath et al. (2013) showed how to do black-box stochastic variational inference (BBSVI) in models with continuous parameterizations, requiring only gradients of the log-posterior. Titsias & Lázaro-Gredilla (2014) applied this method to large-scale problems by allowing stochasticity due to data subsampling. In this context, 'black-box' means that the only information required about the model is the gradient of its posterior, which can be computed using automatic differentiation.

This method has great potential to speed up the prototyping and evaluation of statistical models. However, the most commonly used automatic differentiation packages in machine learning, Theano (Bastien et al., 2012), Torch (Collobert et al., 2002) and TensorFlow(Abadi et al., 2016) require learning a new syntax in which to express operations, each acting as a compiler for a restricted mini-language.

In contrast, the Autograd package provides automatic differentiation for standard Python, Numpy (Oliphant, 2007) and Scipy (Jones et al., 2001) code. Autograd can handle Python code containing control flow primitives such as `for` loops, `while` loops, recursion, `if` statements, closures, classes, list indexing, dictionary indexing, arrays, array slicing and broadcasting.

Using autograd means that both the model and inference method can be specified in a small amount of pure Python, allowing rapid development, debugging, and simple deployment. It also exposes the simplicity of inference methods such as BBSVI. A similar approach is possible in C++ using the automatic differentiation library of Stan (2015).

2 Very short automatic inference code

Below, we give code for doing black-box stochastic variational inference in any differentiable model whose parameters can be transformed into an unconstrained space.

Autograd's `grad` takes in a function, and returns a function computing its gradient. This gradient function can be fed into any standard stochastic-gradient-based optimizer. In our experiments, we used Adam (Kingma & Ba, 2014), a variant of RMSprop with momentum.

```
def lower_bound(variational_params, logprob_func, D, num_samples):
    # variational_params: the mean and covariance of approximate posterior.
    # logprob_func:       the unnormalized log-probability of the model.
    # D:                  the number of parameters in the model.
    # num_samples:        the number of Monte Carlo samples to use.

    # Unpack mean and covariance of diagonal Gaussian.
    mu, cov = variational_params[:D], np.exp(variational_params[D:])

    # Sample from multivariate normal using the reparameterization trick.
    samples = npr.randn(num_samples, D) * np.sqrt(cov) + mu

    # Lower bound is the exact entropy plus a Monte Carlo estimate of energy.
    return mvn.entropy(mu, np.diag(cov)) + np.mean(logprob(samples))

# Get gradient with respect to variational params using autograd.
gradient_func = grad(lower_bound)
```

Figure 1: Black-box stochastic variational inference in five lines of Python, using automatic differentiation. The variational objective gradient can be used with any stochastic-gradient-based optimizer.

3 A simple example

As an example that's easy to visualize, we used BBSVI to fit a Guassian to a simple 2D target distribution:

```
def log_density(x):
    # An example unnormalized 2D density
    mu, log_sigma = x[:, 0], x[:, 1]
    sigma_density = norm.logpdf(log_sigma, 0, 1.35)
    mu_density    = norm.logpdf(mu,        0, np.exp(log_sigma))
    return sigma_density + mu_density
```

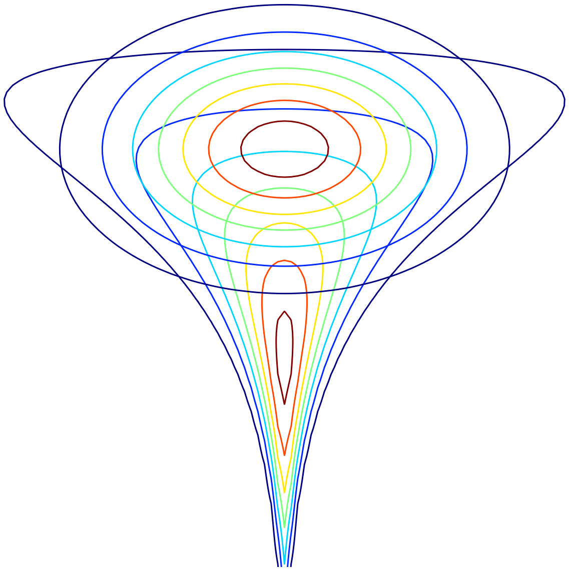

Figure 2: Isocontours of a non-Gaussian target distribution, and contours of the corresponding Gaussian approximate posterior. The full code listing for this figure is available at github.com/HIPS/autograd/tree/master/examples/black_box_svi.py

In this example we use a diagonal Gaussian as the approximating distribution, but it is straightforward to use a full-covariance parameterization, as in Kucukelbir et al. (2014). This parameterization would have $\mathcal{O}(D^2)$ parameters. If the Cholesky factorization is used, both sampling and the entropy calculation can be computed in $\mathcal{O}(D^2)$ time.

4 Variational inference in a deep Bayesian neural network

To demonstrate the power of this method, we implement Bayesian inference in an arbitrarily deep neural network, simply by expressing its forward pass in Numpy:

```
def make_nn_funs(layer_sizes, L2_reg, noise_variance, nonlinearity=np.tanh):
    # These functions implement a standard multi-layer perceptron,
    # vectorized over both training examples and weight samples.
    shapes = zip(layer_sizes[:-1], layer_sizes[1:])
    num_weights = sum((m+1)*n for m, n in shapes)

    def unpack_layers(params):
        # Convert parameter vector into appropriately-sized weight matrices.
        num_param_sets = len(params)
        for m, n in shapes:
            weights = params[:, :m*n]     .reshape((num_param_sets, m, n))
            biases  = params[:, m*n:m*n+n].reshape((num_param_sets, 1, n))
            yield weights, biases
            params = params[:, (m+1)*n:]

    def predictions(params, inputs):
        # weights is shape (num_weight_samples x num_weights)
        # inputs  is shape (num_datapoints x D)
        inputs = np.expand_dims(inputs, 0)
        for weights, biases in unpack_layers(params):
            # Doubly-vectorized matrix multiplication.
            outputs = np.einsum("mnd,mdo->mno", inputs, weights) + biases
            inputs = nonlinearity(outputs)
        return outputs

    def logprob(weights, inputs, targets):
        log_prior = -L2_reg * np.sum(weights**2, axis=1)
        preds = predictions(weights, inputs)
        log_lik = -np.sum((preds - targets)**2, axis=1)[:, 0] / noise_variance
        return log_prior + log_lik

    return num_weights, predictions, logprob

# Specify inference problem by its unnormalized log-posterior.
rbf = lambda x: norm.pdf(x, 0, 1)
num_weights, predictions, logprob = \
    make_nn_funs(layer_sizes=[1, 10, 10, 1], L2_reg=0.01,
                 noise_variance = 0.01, nonlinearity=rbf)

inputs, targets = build_toy_dataset()
log_posterior = lambda weights, t: logprob(weights, inputs, targets)
```

In this example, `einsum` is used to do the matrix multiplication in a way that vectorizes both over datapoints and weight samples. This avoids a separate evaluation of the neural network for every weight sample. This example re-creates some of the work of Blundell et al. (2015) automatically.

5 Source Code

Autograd's source code and documentation is available at `github.com/HIPS/autograd`.

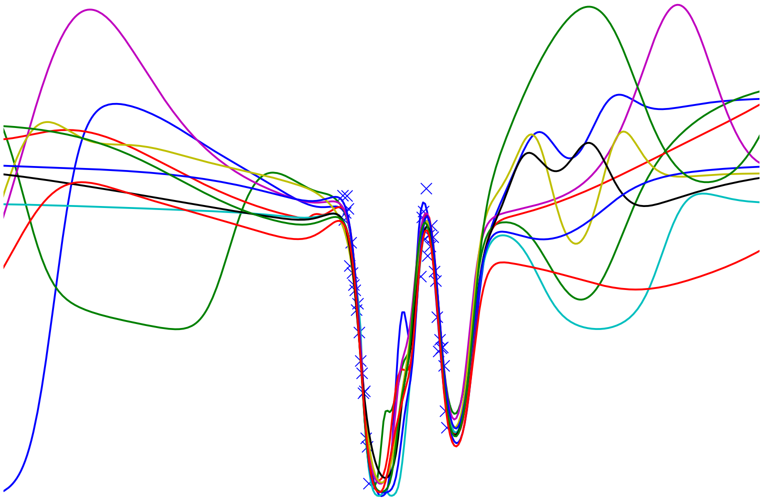

Figure 3: Draws from the variational posterior on a two-layer Bayesian neural network model trained using BBSVI. Blue crosses denote data, and colored lines show individual functions draw from the posterior. The full code listing for this demo is available at `github.com/HIPS/autograd/tree/master/examples/bayesian_neural_net.py`

References

Abadi, Martın, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian J., Bergeron, Arnaud, Bouchard, Nicolas, and Bengio, Yoshua. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Blundell, Charles, Cornebise, Julien, Kavukcuoglu, Koray, and Wierstra, Daan. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Collobert, Ronan, Bengio, Samy, and Mariéthoz, Johnny. Torch: a modular machine learning software library. Technical report, IDIAP, 2002.

Jones, Eric, Oliphant, Travis, Peterson, Pearu, et al. SciPy: Open source scientific tools for Python, 2001. URL `http://www.scipy.org/`.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kucukelbir, Alp, Ranganath, Rajesh, Gelman, Andrew, and Blei, David. Fully automatic variational inference of differentiable probability models. In *NIPS Workshop on Probabilistic Programming*, 2014.

Oliphant, Travis E. Python for scientific computing. *Computing in Science & Engineering*, 9(3): 10–20, 2007.

Ranganath, Rajesh, Gerrish, Sean, and Blei, David M. Black box variational inference. *arXiv preprint arXiv:1401.0118*, 2013.

Stan. A c++ library for probability and sampling, version 2.8.0, 2015. URL `http://mc-stan.org/`.

Titsias, Michalis and Lázaro-Gredilla, Miguel. Doubly stochastic variational bayes for non-conjugate inference. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1971–1979, 2014.