Prepackaging the Variational Auto-Encoder

Evan Poworoznek Michael Valancius

April 2019

Abstract

We focus on the paper *Auto-Encoding Variational Bayes* by Kingma and Welling. We explore the methodological concepts introduced in the paper, and the modeling scheme it is famous for. Using the Keras architecture, we develop python code to easily fit the example model from AEVB to an arbitrary dataset with straightforward input of model parameters. We package this code for installation from PyPI and provide use case examples on the dataset in AEVB and an extended version.

1 Background

Auto-Encoding Variational Bayes is an often cited paper in the ML literature [4]. It serves the dual purpose of introducing a method for performing inference, the Stochastic Gradient Variational Bayes estimator, and a model that can take great advantage of that method, the variational auto-encoder. The influence of both the VAE model and the SGVB bound lead to a resurgence of interest in generative models succeeding the Restricted Boltzmann

Machine [2]. One product of that interest is Generative Adversarial Networks [1], which share many similarities with the VAE, and remain extremely popular.

An auto-encoder takes as input a high dimensional vector. The encoder maps this input to a latent space of lower dimension, and the decoder remaps a low dimensional latent vector to an output of the same dimension as the input. The full auto-encoder then closely reproduces a high dimensional input after it has been passed through a low dimensional space. A loss function for this model then would be defined over the high dimensional inputs and outputs, and desire accuracy in the reproduction. These models function well for highly structured data, where the high dimensional input does not represent a lot of information, and thus translation to the lower dimensional space can be performed without destroying much information.

The variational auto-encoder uses a neural network (a single layer dense network) as the encoder and decoder, and assumes a standard isotropic multivariate normal prior for the latent space. This makes it a flexible tool capable of approximating many data-generating functions. The output of the encoder for a single datapoint are the parameters for the variational approximation to the latent space distribution of the datapoint. The SGVB algorithm allows such a model to be fit efficiently where some other estimators have high variance or do not allow such an intractable likelihood.

The data for which such a model can be considered is evident from the basic structure. The model can learn a complicated way in which a small amount of information can be mapped to a large dimensional space, so it will be useful for data of high dimension resulting from a possibly complicated

function of a small amount of information. Image data, especially simple images such as letters and numbers are a clear case of this data-type. Letters are designed to be discernable, but images of a letter contain nearly the same amount of information at different resolutions once the lines of the letter or digit are visible. The high dimensional input of pixel intensities for images of letters are then mapped to a lower dimensional representation without loss of much information. Other natural data such as musical compositions are similarly reducible.

The problems that can be solved with a general auto-encoder take three forms once the model is trained. The encoder can be used for dimension reduction, either for visualization dissimilarity or conceivably for lossy compression of complicated, structured data. The decoder can be used for generating new, never-observed cases in the original datatype based on an input in the latent space. Together the entire auto-encoder can be used for generating nearest representations based on the training data on new inputs of the original datatype. All of these problems can be solved with the variational auto-encoder, and the class of data-generating functions that can be closely approximated by the neural network encoder and decoder is very large.

The AEVB algorithm is separately applicable. The VAE, despite its massive popularity, is included in the AEVB paper as an example, and the primary focus is performing variational inference for an arbitrary auto-encoder using the SGVB estimator. We include more technical discussion in *Section 3* on the function and novel advancement of this algorithm. Practically, this algorithm is a flexible mini-batch algorithm for performing a first order gradient based optimization in an arbitrary parameter space using an estimator

of the likelihood with low variance. This allows its application to many more models than just the variational auto-encoder.

2 AEVB Algorithm

The Auto-Encoding Variational Bayes algorithm is primarily just the use-case of the Stochastic Gradient Variational Bayes estimator and is thus flexibly defined for all other elements. The version presented in the paper is a mini-batch version, but potentially the entire dataset could be used in the optimization at every step. The algorithm is defined for complete input data, a set of generative model parameters which in the VAE case represents the weight parameters in the neural net, and a set of variational parameters. The variational parameters, forming the latent representations of the training data, are optimized at the same time as the generative model parameters, and there are not separate steps for the two.

The algorithm is short and can be directly summarized. We begin at an initial state for all the parameters, and using some subset of the data along with realizations of some stochastic element, we compute the gradient of the SGVB lower bound estimator with respect to every parameter, variational and generative. Using this gradient information we update the values of the parameters using an arbitrary gradient-based optimization method. We repeat this process until we reach convergence in every parameter. This algorithm is standard and accessible, except for the introduction of a stochastic element and the use of the SGVB bound instead of a direct likelihood or posterior. We discuss both of these ideas in the following section.

2.1 SGVB estimator

Given that sampling the posterior is infeasible, we might just wish to maximize parameters for variational inference. Unfortunately not all problems are easily optimizable using the regular or parameter expanded expectation maximization algorithm, and in such cases the integration required for computing the expectations in a mean field approach can be difficult or impossible. In these cases a Monte-Carlo approximation of such integrals can be substituted, but these approximations are either time consuming to compute or exhibit high variance. The case of such complicated data generating models is a useful one to consider, though, as they include many useful and highly flexible tools like neural and deep networks. This is the motivation for the SGVB estimator, and is important for understanding its function.

The estimator itself is designed to be useful for training models and thus exhibits relatively low variance, is appropriately maximized near the posterior maximum, and is once differentiable almost everywhere with respect to all the parameters, assuming that the model and prior is valid for such optimization. The estimator is shown below:

$$\tilde{\mathcal{L}}(\theta, \phi; \mathbf{x}^{(i)}) = \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i,l)}) - \log q_\phi(\mathbf{z}^{(i,l)} \mid \mathbf{x}^{(i)}) \tag{1}$$

Where the parameters of the prior p are θ, the parameters of the data generating model q are ϕ, the observable data are $\mathbf{x}$ and the latent representations of these data are $\mathbf{z}$. By restricting the class of estimable approximate posteriors, the latent variable $\mathbf{z}^{(i)}$ is decomposed into a differentiable function of the generative parameters, the datapoint it represents $\mathbf{x}^{(i)}$, and a stochastic element. This stochastic element is the stochastic element sampled in the

AEVB algorithm and is the source of the restriction. This restriction therefore allows all distributional families that can be written as a differentiable parameterized transformation of a standard distribution. The sum over l is then a Monte-Carlo approximation over realizations of this stochastic element.

This version of the SGVB estimator, though very general, includes additional variance from the generative model term that is not necessary in certain cases, and is not the one used to fit the variational auto-encoder. For convenient choices of p and q the KL divergence term q to p can be computed analytically and performing a Monte-Carlo approximation adds additional unnecessary variance. For such a case the second version of the SGVB estimator is introduced;

$$\tilde{\mathcal{L}}(\theta, \phi; \mathbf{x}^{(i)}) = -D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i,l)}) \qquad (2)$$

This estimator (2) is the one used in the AEVB algorithm to fit the VAE.

3 Optimization

The flexibility of the algorithm in all areas except the estimator allows certain choices to minimize the computational cost of a fitting the algorithm. The choices that are immediately available are the choices of batch size, the number of stochastic element realizations, the method of computing the estimator gradients, the optimization method used for updating the parameter values, and what defines convergence. We are optimizing for fitting a variational auto-encoder, so we make choices to speed up that process. We do not

consider the compiled code optimization methods such as `pybind11`, `cython`, or `numba` because the entirety of the computationally intensive portion of the code depends on the fitting of two large neural nets connected by a latent space with a loss function defined over the reconstruction error. Manually writing the back-propogation to speed up with JIT compilation (for a variable number of nodes) is out of our scope, and very much reinventing the wheel.

To avoid this massive exercise in inefficiency we take advantage of platforms that are designed for extremely efficient computation in this type of model structure. We work in a probabilistic programming environment to avoid computing the back-propagation gradients ourselves. We chose the Keras package in Python to provide a simple and extremely efficient platform for defining and training an arbitrary neural network architecture, using the TensorFlow backend. This backend is built on Python controlled C++ functions, and is highly optimized.

For our replication of the data example given in the paper, we use the same batch size of 100, but in the package implementation of this algorithm we make this a flexible parameter for cases in which 100 datapoints are not an appropriate amount of information for parameter updates. The number of stochastic element samples is suggested at 1 in the paper, and if the resulting Monte-Carlo estimate in (2) is a decent estimator of the targeted expectation then this is the computationally optimal choice.

In general the choice of optimization method is dependent on both the data and the model. We are constrained to first order methods, but that is the focus of a lot of work in the field of optimization. Given that our

motivating example is image analysis, we choose an optimization method that performs well in similar cases. The AdamMax method has shown good behaviour in training neural networks to classify handwritten letters, as in our example case, so we use that method as a default but allow the user to specify a different optimizer Keras allows to perform parameter updates. We also allow the user to specify the number of epochs, allowing them to decide what constitutes convergence.

4 Examples

Please see the Jupyter notebooks for examples of the packaged variational auto-encoder in practice.

`https://poworoznek.github.io/website/MNIST.html`

`https://poworoznek.github.io/website/Fashion.html`

5 Conclusion

The paper *Auto-Encoding Variational Bayes* has had a large impact in generative modeling both for the model presented in the paper, and the models inspired by it. The example model introduced in the paper, the variational auto-encoder, was made possible to train efficiently by the SGVB lower bound estimator similarly introduced. The use of this model and simple augmentations, such as a deep or convolutional neural net, fit using the AEVB algorithm have made both auto-encoding and generative models well known to almost all of the machine learning community. The successors of these

models, such as GANs, represent some of the most interesting and creative models in use today, producing artwork and extremely believable fake photographs [3].

The implementation we have provided is efficient, and flexible within the bounds of the model as it was introduced. Such a model is suitable for attempting dimension reduction for visualization, or to compare relative distances between inputs in a more euclidean space. It is also useful for generating nice pictures, as a generative model should be. It can generate representative output for latent input, or a closest representative output given a new input. These problems can be solved well for any data that have the characteristics described in *Section 1* using the Python package.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[2] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[3] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018.

[4] D. Kingma and M. Welling. Auto-Encoding Variational Bayes. *e-Print arXiv*, pages 1–14, December 2013.