InferPy: Probabilistic Modeling Made Easy

Rafael Cabañas, Andrés R. Masegosa, Antonio Salmerón

University of Almería, ES-04120 Almería, Spain

Abstract

InferPy is a high-level Python API for probabilistic modeling built on top of Edward and Tensorflow. InferPy, which is strongly inspired by Keras, focuses on being user-friendly by using an intuitive set of abstractions that make easy to deal with complex probabilistic models. It should be seen as an interface rather than a standalone machine-learning framework. In general, InferPy has the focus on enabling flexible data processing, easy-to-code probabilistic modeling, scalable inference and robust model validation.

Keywords: keyword 1, keyword 2, keyword 3

1. Introduction

Machine learning (ML) [1] is a fundamental part of many artificial intelligence techniques [2], and the key of many innovative applications. Unfortunately, for a company or an institution, the development of ML models specific to their problems requires enormous efforts [3]. For this reason, probabilistic programming languages (PPLs) [4] are an active area of research. PPLs offer the same advantages to the ML community that high-level programming languages offered to software developers fifty years ago [5]. Programmers could specialize in model development while ML experts could focus their efforts on developing reusable inference engines. Thus, the number of non-experts who can create applications using a PPL could increase. Special attention requires those PPLs which exploit recent advances in probabilisitic inference for defining probabilistic models containing deep neural networks [6, 7]. These PPLs rely on deep learning libraries like Tensorflow

Email addresses: `rcabanas@ual.es` (Rafael Cabañas), `andresmasegosa@ual.es` (Andrés R. Masegosa), `antonio.salmeron@ual.es` (Antonio Salmerón)

[8]. The main drawback of these approaches is the high complexity of the provided abstractions, specially those centered around the definition of probability distributions over multidimensional Tensors.

InferPy[1] tries to address these issues by defining a user-friendly API which trades-off model complexity with ease of use. Complex operations over Tensor objects are hidden to the user. Similarly, Edward's flexible approach to probabilistic inference demands to provide specific details such as the variational family. Again, InferPy gives the possibility to hide all this information and make inference with a single line of code. As InferPy uses Tensorflow as computing engine, all the parallelization details are hidden to the user.

2. Background

InferPy focuses on *hierarchical probabilistic models* structured in two layers: (i) a *prior model* defining a joint distribution $p(\mathbf{w})$ over the global parameters of the model ($\mathbf{w}$ can be a single random variable or a bunch of random variables with any given dependency structure); (ii) a *data or observation model* defining a joint conditional distribution $p(\mathbf{x}, \mathbf{z}|\mathbf{w})$ over the observed quantities $\mathbf{x}$, and the the local hidden variables $\mathbf{z}$ governing the observation $\mathbf{x}$. As a running example, Figure 1 shows a model of this type.

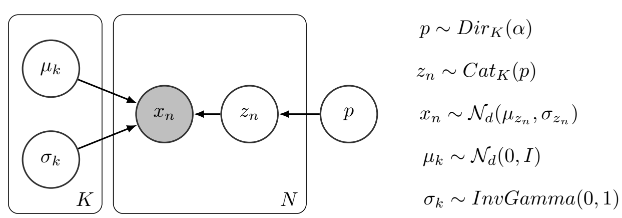

Figure 1: Mixture of K d-dimensional Gaussian distributions learned from N observations.

3. Software Framework

3.1. Model Definition

In InferPy, models are specified using a simple language of random variables, which are grouped in a *probabilistic model* object (i.e., defined using the construct `with inf.ProbModel() as m:`) defining a joint distribution over observable and hidden variables $p(\mathbf{w}, \mathbf{z}, \mathbf{x})$. As an example, we provide in Figure 2 how the model of Figure 1 would be defined in InferPy.

[1]Home: inferpy.readthedocs.io; Source: github.com/PGM-Lab/InferPy

```
1  ## model definition ##
2  with inf.ProbModel() as model:
3
4      # prior distributions
5      with inf.replicate(size=K):
6          mu = inf.models.Normal(loc=0, scale=1, dim=d)
7          sigma = inf.models.InverseGamma(1, 1, dim=d)
8      p = inf.models.Dirichlet(np.ones(K)/K)
9
10     # define the generative model
11     with inf.replicate(size=N):
12         z = inf.models.Categorical(probs = p)
13         x = inf.models.Normal(mu[z], sigma[z], observed=True, dim=d)
```

Figure 2: InferPy code for the Mixture of Gaussians model of Figure 1.

InferPy allows to specify our model in a single sample-basis, resembling the standard *plateau notation*, with the `with inf.replicate(size=N)` construct (Line 5). The dimension N is the number of *replicas* of this part of the model. The dimension of each variable can be specified either using the input parameter `dim` (Line 6), or by the length of the distribution parameters (e.g., other InferPy variable, NumPy's ndarray [9], a tensor or a Python list). For example, variable `x` in the previous code contains N replicas of d independent Gaussian distributions and, in consequence, has two dimensions (i.e., $shape = [N, d]$).

Like in Edward, each random variable y is associated to a tensor y^* representing a sample from its distribution. Note that when operating on y, the operation is indeed done on y^*. In the previous code, the mean (i.e., loc) of `x` is a sample from the distribution obtained by indexing `mu` with a sample from `z`. Any variable defined in InferPy encapsulates an equivalent one in Edward, which can be obtain by accessing the property `dist`. For simplicity, the user does not deal with tensor objects unless it is explicitly specified, e.g.: `z.sample()` returns an array of samples while `z.sample(tf_run=False)` allows to obtain the equivalent (lazily evaluated) Tensor object.

3.2. Approximate Inference

InferPy directly relies on top of Edward's inference engine. In particular, InferPy inherits Edward's approach and considers approximate inference solutions in which the task is to approximate the posterior with a simpler distribution q. Unlike Edward, InferPy offers the possibility to hide all these details about the definition of this q distribution, making the inference more simple for non-advanced users. Figure 4 shows the code for making inference in the model defined in the previous section.

```
# compile and fit the model with training data
data = {x: x_train}
model.compile(infMethod="MCMC")
model.fit(data)
# print the posterior
print(model.posterior(mu))
```

Figure 3: Code for making inference in the Mixture of Gaussian model of Figure 2.

4. Comparison with Edward

The analogous Edward code for defining and making inference in a mixture of Gaussians, which can be found in our online documentation[2], has some drawbacks compared to the code in InferPy (Figures 2 and 4). First, the model definition is more complex because this is not done in a single-sample basis. This can be specially problematic when defining the dependencies among variables. For example, the mean of `x` is specified using the function `tf.gather` which is not always intuitive, i.e. `loc=tf.gather(mu,z)`. Secondly, Edward requires to have a strong knowledge about the inference algorithms for specifying all its parameters. For the running example, a q and g variable is defined for each latent variable in the model. For variable `mu`, this is done as follows.

```
qmu = ed.models.Empirical(params=tf.get_variable("qmu/prm", [T,K,d],
                                initializer=tf.zeros_initializer()))
gmu = ed.models.Normal(loc=tf.ones([K,d]), scale=tf.ones([K,d]))
```

Figure 4: Edward's code for defining the q distribution for the model of Figure 2.

5. Conclusions

We have briefly presented InferPy, a high-level API for probabilistic modeling built on top of Edward and Tensorflow. The use of intuitive abstractions

[2]https://inferpy.readthedocs.io/en/latest/notes/inf_vs_ed.html

such as the *plateau notation* simplifies the task of defining complex hirearchical probabilistic models. In the future, we aim to fully integrate InferPy with Keras, allowing simple probabilistic modeling with deep neural networks.

Acknowledgements

Authors have been jointly supported by the Spanish Ministry of Science, Innovation and Universities and by the FEDER under the projects TIN2015-74368-JIN, and TIN2016-77902-C3-3-P.

[1] K. P. Murphy, Machine learning: A probabilistic perspective. adaptive computation and machine learning (2012).

[2] S. J. Russell, P. Norvig, Artificial intelligence: a modern approach, Malaysia; Pearson Education Limited,, 2016.

[3] Z. Ghahramani, Probabilistic machine learning and artificial intelligence, Nature 521 (7553) (2015) 452.

[4] A. D. Gordon, T. A. Henzinger, A. V. Nori, S. K. Rajamani, Probabilistic programming, in: Proceedings of the on Future of Software Engineering, FOSE 2014, ACM, 2014, pp. 167–181.

[5] R. L. Wexelblat, History of programming languages, Academic Press, 2014.

[6] D. Tran, A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, D. M. Blei, Edward: A library for probabilistic modeling, inference, and criticism, arXiv preprint arXiv:1610.09787.

[7] I. Uber Technologies, Pyro deep universal probabilistic programming, `http://pyro.ai`, accessed 2017-07-31 (2017-2018).

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: a system for large-scale machine learning., in: OSDI, Vol. 16, 2016, pp. 265–283.

[9] S. v. d. Walt, S. C. Colbert, G. Varoquaux, The numpy array: a structure for efficient numerical computation, Computing in Science & Engineering 13 (2) (2011) 22–30.

Required Metadata

Current executable software version

Ancillary data table required for sub version of the executable software: (x.1, x.2 etc.) kindly replace examples in right column with the correct information about your executables, and leave the left column as it is.

Nr.	**(executable) Software metadata description**	**Please fill in this column**
S1	Current software version	for example 1.1, 2.4 etc.
S2	Permanent link to executables of this version	example: $https://github.com/combogenomics/DuctApe/releases/tag/DuctApe-0.16.4$
S3	Legal Software License	List one of the approved licenses
S4	Computing platform/Operating System	for example Android, BSD, iOS, Linux, OS X, Microsoft Windows, Unix-like , IBM z/OS, distributed/web based etc.
S5	Installation requirements & dependencies	
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	Example: $http://mozart.github.io/documentation/$
S7	Support email for questions	

Table 1: Software metadata (optional)

Current code version

Ancillary data table required for subversion of the codebase. Kindly replace examples in right column with the correct information about your current code, and leave the left column as it is.

Nr.	**Code metadata description**	**Please fill in this column**
C1	Current code version	For example v42
C2	Permanent link to code/repository used of this code version	For example: $https://github.com/mozart/mozart2$
C3	Legal Code License	List one of the approved licenses
C4	Code versioning system used	For example svn, git, mercurial, etc. put none if none
C5	Software code languages, tools, and services used	For example c++, python, r, etc.
C6	Compilation requirements, operating environments & dependencies	
C7	If available Link to developer documentation/manual	For example: $http://mozart.github.io/documentation/$
C8	Support email for questions	

Table 2: Code metadata (mandatory)