InferPy: Probabilistic Modeling with Deep Neural Networks Made Easy

Javier Córza, Rafael Cabañas, Antonio Salmerón, Andrés R. Masegosa

aDepartment of Mathematics, University of Almería, Spain
bCDTIME, University of Almería, Spain
cIstituto Dalle Molle di Studi sull’Intelligenza Artificiale (IDSIA), Switzerland

Abstract

InferPy is a Python package for probabilistic modeling with deep neural networks. It defines a user-friendly API that trades-off model complexity with ease of use, unlike other libraries whose focus is on dealing with very general probabilistic models at the cost of having a more complex API. In particular, this package allows to define, learn and evaluate general hierarchical probabilistic models containing deep neural networks in a compact and simple way. InferPy is built on top of Tensorflow Probability and Keras.

Keywords: Deep Probabilistic modeling, Hierarchical probabilistic models, Neural networks, Bayesian layers, Tensorflow, User-friendly

1. Introduction

Advances in variational methods [1] have made possible the development of a new formalism, namely deep probabilistic modeling [2], which combines probabilistic models within deep neural networks (DNNs) to capture complex non-linear relationships among random variables. The release of multiple libraries for deep probabilistic modeling [3, 4] are greatly expanding the adoption of these powerful probabilistic modeling techniques. However, these libraries are usually difficult to use, especially when defining distributions containing NNs, which requires dealing with multidimensional matrices (i.e. tensors).
This paper presents a new version of InferPy as a high-level Python API for probabilistic modeling with DNNs with a strong focus on ease of use. (A major released of the code has been performed: from 0.2.x to 1.2.x.) The main differences with the previous version [5] are the following ones. Models can now contain DNNs to model non-linear relationships among random variables. The API has been significantly changed to make it compatible with the use of DNNs. InferPy relies now on Tensorflow Probability (TFP) [3] (Inferpy’s previous version relied on Edward [6], which is deprecated).

2. Background

Probabilistic models with DNNs are usually found in the literature under the name of deep generative models or Bayesian deep learning [7]. The former focuses on density estimation using DNNs, while the latter focuses on treating the parameters of a DNN as random variables to capture the model uncertainty. See [2] for a recent review of all these models. Along these lines, new software tools have appeared to deal with probabilistic models containing DNNs [5, 3, 4]. These tools usually fall under the umbrella term probabilistic programming languages (PPLs) [8], and provide support for methods for reasoning about complex probabilistic models. Some examples are TFP [3], Pyro [4], etc.

3. Software Framework

The main features of InferPy are: (i) Its simple API allows easy prototyping of probabilistic models including DNNs; (ii) Unlike TFP, it is not required to have a strong background in the inference methods available (Variational Inference [1] [2] and Monte Carlo methods [9]) as many details are hidden to the user; (iii) InferPy runs seamlessly on CPUs and GPUs. InferPy can be seen as an upper layer for working with TFP. Thus, models that can be defined in InferPy are those that can be defined using TFP. InferPy is distributed as open-software (Apache-2.0) using Pypi and its source code is available at GitHub (see Tab. 1 and 2).

4. Illustrative Example

For illustrating the usage of InferPy, we will consider a variational autoencoder (VAE) [10], as it is one of the most widely used probabilistic models containing deep NNs. In a VAE, every object has a unknown latent representation (a code), modeled with a multivariate Gaussian (a distribution over
possible codes). This latent representation gives rise to a multivariate Gaussian distribution over the observed representation (the decoded observation of this object) by passing the latent representation through a NN called the *decoder*. This part of the model is defined in Fig. 1 (lines 1 to 8) and the creation of an instance (line 9), which is an object of class `inf.models.probmodel`. A probabilistic model in InferPy is defined as a function with the decorator `@inf.probmodel`. Following the statistical inference terminology, we refer to this part of the model as the *p model*.

Random variables are objects of class `inf.models.RandomVariable`. Variables composing a probabilistic model are those instantiated during the execution of its decorated function. The `with inf.datamodel()` syntax is used to indicate the InferPy variables contained within this construct are replicated for every data sample. Every replicated variable is conditionally independent given the previous random variables (if any) defined outside this `with` statement[1]. This construct enormously simplify the code of the model.

```python
@inf.probmodel
def vae(k, d0, d):
    with inf.datamodel():
        z = inf.Normal(tf.ones(k), 1, name="z")
        decoder = inf.layers.Sequential([
            tfp.layers.DenseFlipout(d0, activation=tf.nn.relu),
            tf.keras.layers.Dense(d)]))
        x = inf.Normal(decoder(z), 1, name="x")
    p = vae(k=2, d0=100, d=28*28)
```

Figure 1: p-model and decoder

The *encoder* part of a VAE defines the inference part of the model: given the observed representation of an object we need to find the posterior probability over possible latent representations (codes) of this object, in the form a multivariate Gaussian. In a VAE, this inference part is defined using an *amortized variational inference* [1][2] scheme, which relies on an *encoder network*. Following the variational inference terminology, we call this part of the model as the *q model*. As shown in Fig. 2, this part is similarly defined with the same decorator. The correspondence between the variables in the *decoder* part and the *encoder* part of the model is done by the argument *name*, i.e.,

[1] In contrast to other libraries, the number of replications will be automatically calculated just before the inference.
they should be the same. For the NNs definition, standard Keras code can be used. However, in case of Bayesian NNs (e.g., `tfp.layers.DenseFlipout` defines a Bayesian layer), the `Sequential` model provided by InferPy must be used. This provides a Bayesian treatment of the decoder network.

```python
@inf.probmodel
def qmodel(k, d0, d):
    with inf.datamodel():
        x = inf.Normal(tf.ones(d), 1, name=\"x\")
        encoder = tf.keras.Sequential([tf.keras.layers.Dense(d0, activation=tf.nn.relu),
                                        tf.keras.layers.Dense(2 * k)])
        output = encoder(x)
        qz_loc = output[:, :k]
        qz_scale = tf.nn.softplus(output[:, k:]) + 0.01
        qz = inf.Normal(qz_loc, qz_scale, name=\"z\")
        q = qmodel(k=2, d0=100, d=28*28)
```

Figure 2: q-model and encoder

A minimal example using (stochastic) variational inference \[1, 2\] as a learning engine is given in Fig. 3. Even though the learning algorithm can be further configured, in this case, an object of class `inf.inference.SVI` is created with the q-model, the epochs (number of iterations) and batch size as input arguments. The optimization starts when the method `fit()` is invoked. Finally, we might sample from the posterior of z (latent representation) or from the posterior predictive (generating new samples).

```python
SVI = inf.inference.SVI(q, epochs=1000, batch_size=100)
p.fit({\"x\": x_train}, SVI)
postz = p.posterior(\"z\", data={\"x\": x_train[:M,:]}).sample()
x_gen = p.posterior_predictive(\"x\", data={\"z\": postz}).sample()
```

Figure 3: Inference from the posterior distributions

The analogous TFP code for this model is far more complex. This can be found in the online documentation (see Tab. \[1\] and \[2\]), together with other examples and a complete user manual.

5. Conclusions

InferPy can now represent probabilistic models containing DNNs using a simple and compact API. As most of the inference details are hidden, this package can be used for users without a strong probabilistic background.
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.

References


Metadata

Current executable software version

<table>
<thead>
<tr>
<th>Nr.</th>
<th>(executable) Software metadata description</th>
<th>Software version</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Current software version</td>
<td>1.3.0</td>
</tr>
<tr>
<td>S2</td>
<td>Permanent link to executables of this version</td>
<td><a href="https://pypi.org/project/inferpy/1.3.0/">https://pypi.org/project/inferpy/1.3.0/</a></td>
</tr>
<tr>
<td>S3</td>
<td>Legal Software License</td>
<td>Apache 2.0</td>
</tr>
<tr>
<td>S4</td>
<td>Computing platform/Operating System</td>
<td>Linux, OS X, Microsoft Windows, Unix-like</td>
</tr>
<tr>
<td>S5</td>
<td>Installation requirements &amp; dependencies</td>
<td>Pip, Python 3.5-3.7, tensorflow 1.12.1-1.15.0 tensorflow-probability 0.7.0, networkx 2.2.0-3.0</td>
</tr>
<tr>
<td>S6</td>
<td>Link to user manual</td>
<td><a href="https://inferpy.readthedocs.io/">https://inferpy.readthedocs.io/</a></td>
</tr>
<tr>
<td>S7</td>
<td>Support email for questions</td>
<td><a href="mailto:inferpy.api@gmail.com">inferpy.api@gmail.com</a></td>
</tr>
</tbody>
</table>

Table 1: Software metadata

Current code version

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Code metadata description</th>
<th>Code version</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Current code version</td>
<td>1.3.0</td>
</tr>
<tr>
<td>C2</td>
<td>Permanent link to code/repository used of this code version</td>
<td><a href="https://github.com/PGM-Lab/InferPy/tree/1.3.0">https://github.com/PGM-Lab/InferPy/tree/1.3.0</a></td>
</tr>
<tr>
<td>C3</td>
<td>Legal Code License</td>
<td>Apache 2.0</td>
</tr>
<tr>
<td>C4</td>
<td>Code versioning system used</td>
<td>github</td>
</tr>
<tr>
<td>C5</td>
<td>Software code languages, tools, and services used</td>
<td>Python</td>
</tr>
<tr>
<td>C6</td>
<td>Compilation requirements, operating environments</td>
<td>Python 3.5-3.7, tensorflow 1.12.1-1.15.0 tensorflow-probability 0.7.0, networkx 2.2.0-3.0</td>
</tr>
<tr>
<td>C7</td>
<td>Link to developer documentation/Manual</td>
<td><a href="https://inferpy.readthedocs.io/">https://inferpy.readthedocs.io/</a></td>
</tr>
<tr>
<td>C8</td>
<td>Support email for questions</td>
<td><a href="mailto:inferpy.api@gmail.com">inferpy.api@gmail.com</a></td>
</tr>
</tbody>
</table>

Table 2: Code metadata