Probabilistic Graphical Models with Neural Networks in InferPy

Rafael Cabañas RCABANAS@IDSIA.CH
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Switzerland

Javier Cózar JCOZAR87@UAL.ES

Antonio Salmerón ANTONIO.SALMERON@UAL.ES

Andrés R. Masegosa ANDRESMASEGOSA@UAL.ES
Department of Mathematics and Center for the Development and Transfer of Mathematical Research to Industry (CDTIME), University of Almería, Spain

Abstract

InferPy is an open-source Python package for variational inference in probabilistic models containing neural networks. Other similar libraries are often difficult for non-expert users. InferPy provides a much more compact and simple way to code such models, at the expense of slightly reducing expressibility and flexibility. The main objective of this package is to permit its use without having a strong theoretical background or thorough knowledge of the deep learning frameworks.

Keywords: Deep probabilistic modeling; Hierarchical probabilistic models; Variational Inference; Bayesian learning; TensorFlow; Keras; User-friendly.

1. Introduction

Probabilistic graphical models (PGMs) have been widely used for decades in reasoning under uncertainty and applied to a variety of problems: self-driving cars (Weidl et al., 2018), financial data analysis (Masegosa et al., 2020), environmental modeling (Maldonado et al., 2020), etc. Their advantage resides on their intuitive nature, which makes them suitable to work with experts from other fields who are usually not familiar with machine learning methods. However traditional PGM approaches often fail to model non-linear relations among the variables in the problem.

Probabilistic programming languages over deep learning frameworks, unlike traditional software for PGMs (Scutari, 2010; Masegosa et al., 2019b), allow the definition of probabilistic models containing neural networks (NN). Some examples are TFP/Edward2 (Tran et al., 2018), Pyro (Bingham et al., 2019), Stan (Carpenter et al., 2017), etc. The problem of these tools is that the model specification becomes complex or a strong theoretical background is required. InferPy (Cabañas et al., 2019; Cózar et al., 2019) by contrast provides a user friendly API built on top of TFP/Edward2 for defining and making inference in probabilistic models with NNs. In this paper we briefly describe the package. For further details we refer to the online documentation [1].

2. Theoretical background

InferPy focuses on *hierarchical probabilistic graphical models* (Masegosa et al., 2019a) which are a kind of models defining a joint conditional distribution $p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{w})$ where $\boldsymbol{z}$ are the local hidden variables governing the observable variables $\boldsymbol{x}$ and where $\boldsymbol{w}$ are the global parameters. These models can employ NNs to define the parameters of the distribution of a random variable conditional

1. Home: `http://inferpy.readthedocs.io`; Source: `https://github.com/PGM-Lab/InferPy`

on their parents. For instance, Fig. 1 depicts a model of this kind for classification of hand-written digits: the images and true classes $\{\boldsymbol{x_n}, y_n\}$ correspond to the observable variables while $\boldsymbol{z}_n$ is the hidden representation of the images in a lower-dimensional space. Two neural networks define the conditional distribution of $\boldsymbol{x_n}|\boldsymbol{z_n}$ and $y_n|\boldsymbol{z_n}$, respectively. In this case, the global parameters correspond to $\boldsymbol{\alpha_1}$ and $\boldsymbol{\alpha_2}$ which are the trainable parameters of the two NNs.

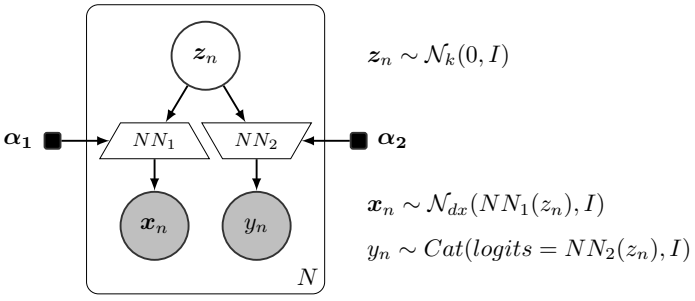

Figure 1: Probabilistic model with NNs for classification of hand-written digits where dx is the number of pixels, dy the number of classes and k the size of the hidden representation.

3. Code example

Here, the usage of InferPy is illustrated with the code for defining and making inference in the digit classification model described above. The probabilistic model with the prior probabilities, namely the P-model, is defined as a function decorated with `@inf.probmodel` and will be made of the random variable defined inside the function. Fig 2 shows the code for defining the P-model of the running example. Dimensions of variables inside `inf.datamodel()` are specified in a single instance basis, resembling the so-called *plateau* notation. Note that NNs are defined with regular Keras code.

```
@inf.probmodel
def digit_classifier(k, d0, dx, dy):
    with inf.datamodel():
        z = inf.Normal(tf.ones(k) * 0.1, 1., name="z")

        nn1 = tf.keras.Sequential([
            tf.keras.layers.Dense(d0, tf.nn.relu),
            tf.keras.layers.Dense(dx)
        ])
        nn2 = tf.keras.Sequential([
            tf.keras.layers.Dense(dy)
        ])
        x = inf.Normal(nn1(z), 1., name="x")
        y = inf.Categorical(logits=nn2(z), name="y")

p = digit_classifier(k=2, d0=100, dx=28*28, dy=3)
```

Figure 2: P-model of the digit classification model depicted in Fig. 1.

For stochastic variational inference (SVI) (Zhang et al., 2018), a variational approximating family should be defined, namely the Q-model. As shown in in the code in Fig. 3, each observation $\boldsymbol{x}_n$ is passed through a *encoder* NN to obtain the parameters of $\boldsymbol{z}_n$.

```python
@inf.probmodel
def qmodel(k, d0, dx):
    with inf.datamodel():
        x = inf.Normal(tf.ones(dx), 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, dx=28*28)
```

Figure 3: Q-model for SVI associated to model in Fig. 2.

Once the P and Q models are defined, inference can be done. Fig. 4 shows how to load the MNIST dataset, set up the inference engine and fit the model to the data with SVI. Finally, the

```python
# get the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data(
num_instances=N, digits=[0, 1, 2])
# set the inference algorithm
SVI = inf.inference.SVI(q, epochs=10000, batch_size=M)
# fit the model to the data
p.fit({"x": x_train, "y":y_train}, SVI)
```

Figure 4: Stochastic variational inference with MNIST data.

hidden representation of all the images can be obtained by sampling from the posterior of $\boldsymbol{z_n}$. For predicting the class of a set of test images, first the posterior of $\boldsymbol{z}$ given the test data is calculated and, secondly, the posterior predictive of y is computed. This is shown in Fig. 5.

```python
# extract the posterior of z given the training data
postz = np.concatenate([
    p.posterior("z", data={"x": x_train[i:i + M, :]}).sample()
    for i in range(0,N,M)])
# predict a set of images
def predict(x):
  postz = p.posterior("z", data={"x": x}).sample()
  return p.posterior_predictive("y", data={"z":postz}).sample()
y_gen = predict(x_test[:M])
```

Figure 5: Sampling from posterior distribution and classification of test data.

4. Conclusions

We have briefly presented InferPy, a high-level API that allows to define hirearchical probabilistic models containing NNs. The use of different intuitive abstractions greatly simplifies the task of defining complex models.

Acknowledgements

Authors have been jointly supported by the Spanish Ministry of Science and Innovation and by the FEDER under the projects TIN2015-74368-JIN, TIN2016-77902-C3-3-P, PID2019-106758GB-C31 and C32.

References

E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019.

R. Cabañas, A. Salmerón, and A. R. Masegosa. Inferpy: Probabilistic modeling with Tensorflow made easy. *Knowledge-Based Systems*, 168:25–27, 2019.

B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

J. Cózar, R. Cabañas, A. R. Masegosa, and A. Salmerón. Inferpy: Probabilistic modeling with deep neural networks made easy. *arXiv preprint arXiv:1908.11161*, 2019.

A. D. Maldonado, M. Morales, P. A. Aguilera, and A. Salmerón. Analyzing uncertainty in complex socio-ecological networks. *Entropy*, 22(1):123, 2020.

A. Masegosa, R. Cabañas, H. Langseth, T. Nielsen, and A. Salmerón. Probabilistic modeling with deep neural networks. *arXiv preprint arXiv:1908.03442*, 2019a.

A. R. Masegosa, A. M. Martinez, D. Ramos-López, R. Cabañas, A. Salmerón, H. Langseth, T. D. Nielsen, and A. L. Madsen. AMIDST: A Java toolbox for scalable probabilistic machine learning. *Knowledge-Based Systems*, 163:595–597, 2019b.

A. R. Masegosa, A. M. Martínez, D. Ramos-López, H. Langseth, T. D. Nielsen, and A. Salmerón. Analyzing concept drift: A case study in the financial sector. *Intelligent Data Analysis*, 24(3): 665–688, 2020.

M. Scutari. Learning bayesian networks with the bnlearn r package. *Journal of Statistical Software, Articles*, 35(3):1–22, 2010. ISSN 1548-7660. doi: 10.18637/jss.v035.i03.

D. Tran, M. W. Hoffman, D. Moore, C. Suter, S. Vasudevan, and A. Radul. Simple, distributed, and accelerated probabilistic programming. In *Advances in Neural Information Processing Systems*, pages 7598–7609, 2018.

G. Weidl, A. L. Madsen, S. Wang, D. Kasper, and M. Karlsen. Early and accurate recognition of highway traffic maneuvers considering real world application: a novel framework using bayesian networks. *IEEE Intelligent Transportation Systems Magazine*, 10(3):146–158, 2018.

C. Zhang, J. Butepage, H. Kjellstrom, and S. Mandt. Advances in variational inference. *IEEE TPAMI*, 2018.