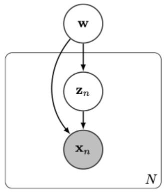
Probabilistic Graphical Models with Neural Networks in InferPy

PGM 2020 AALBORG September 23-25, 2020

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

Models

Hierarchical Probabilistic Models



$\mathbf{w}$: global parameters

$\mathbf{z}$: local hidden variables

$\mathbf{x}$: observations

N : number of observations

$p(\mathbf{w})$: prior model $p(\mathbf{x},\mathbf{z}|\mathbf{w})$: data model

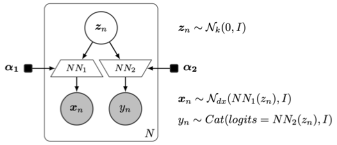
Objective: posterior distribution $p(\mathbf{w},\mathbf{z}|\mathbf{x})$

- Dependencies between variables might be defined with TF functions or even NNs

TensorFlow Keras

Model definition

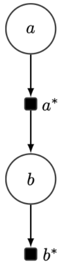
```
import inferpy as inf
import tensorflow as tf

@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)
```




```
In[*]: p.prior().sample()
Out[*]:
OrderedDict([('z', array([[ 0.8503272 , -0.40765837]], dtype=float32)),
             ('x', array([[-2.68360198e-01,  3.11490864e-01, -6.55998230e-01,
                     1.80848286e-01,  5.62604547e-01,  1.11705911e+00,
                     2.10047036e-01, -6.50202155e-01, -6.62622333e-01,
                     . . .
                     2.39737108e-02]], dtype=float32)),
             ('y', array([1], dtype=int32))])
```


Generative models

```
a = inf.Normal(0, 100)

b = inf.Normal(a, 5)
```



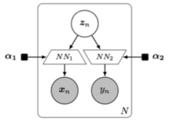

```
In[*]:
   ...: sess = inf.get_session()
   ...: for i in range(5):
   ...:     print(sess.run([a,b]))

[-7.2810316, -6.471646]
[29.092255, 37.471718]
[74.87469, 62.43242]
[44.46464, 39.6697]
[169.10535, 173.74834]
```

```
# a continuous variable might be parent of a discrete one
x = inf.Normal(0, 1)
c = inf.Categorical(probs=tf.case({ x > 0: lambda : [0.0, 1.0],
                                    x <= 0: lambda : [1.0, 0.0]}))
```

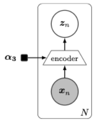

Inference (of the parameters)

Stochastic Variational Inference (SVI)

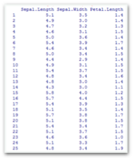
P model



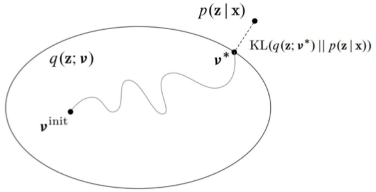
Q model



data



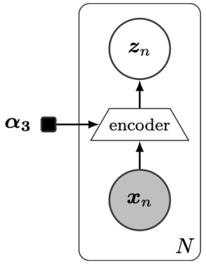
- Inference turns into an optimisation problem



Inference (of the parameters)

- For making inference, the Q model approximating the P model is defined

```
@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)
```

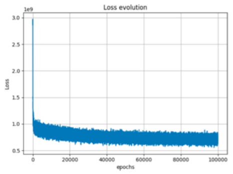



```
# set the inference algorithm
SVI = inf.inference.SVI(q, epochs=10000, batch_size=M)

# fit the model to the data
p.fit({"x": x_train, "y":y_train}, SVI)
```


After the inference

- We can extract and plot the loss function evolution

```
# extract the loss evolution
L = SVI.losses
```



- A function for making predictions

```
# 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])

# compute the accuracy
acc = np.sum(y_test[:M] == y_gen)/M
print(f"accuracy: {acc}")
```