



Deep Probabilistic Modeling (with Tensorflow) made Easy

Rafael Cabañas
PIDSIA

Manno, 28.05.19





Javier Cózar

jcozar87@ual.es



Rafael Cabañas

rcabanas@idsia.ch



Antonio Salmerón

asalmero@ual.es



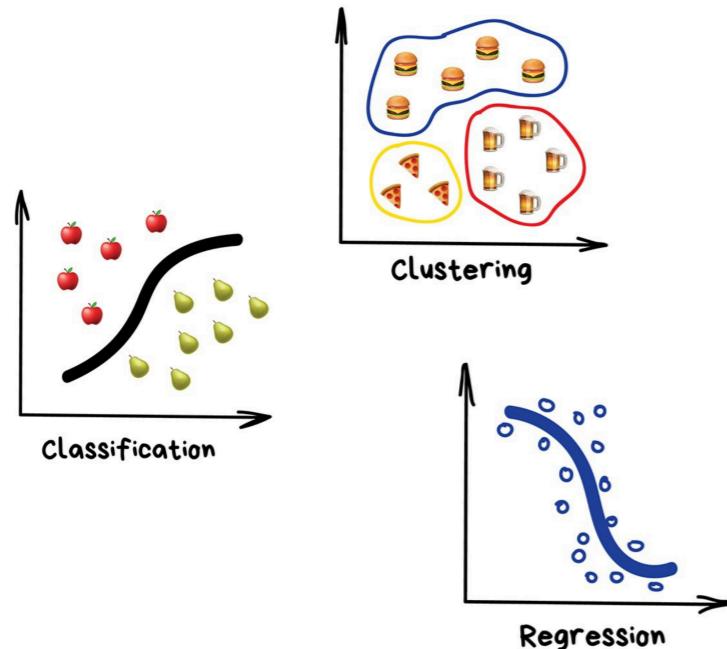
Andrés R. Masegosa

andresmasegosa@ual.es

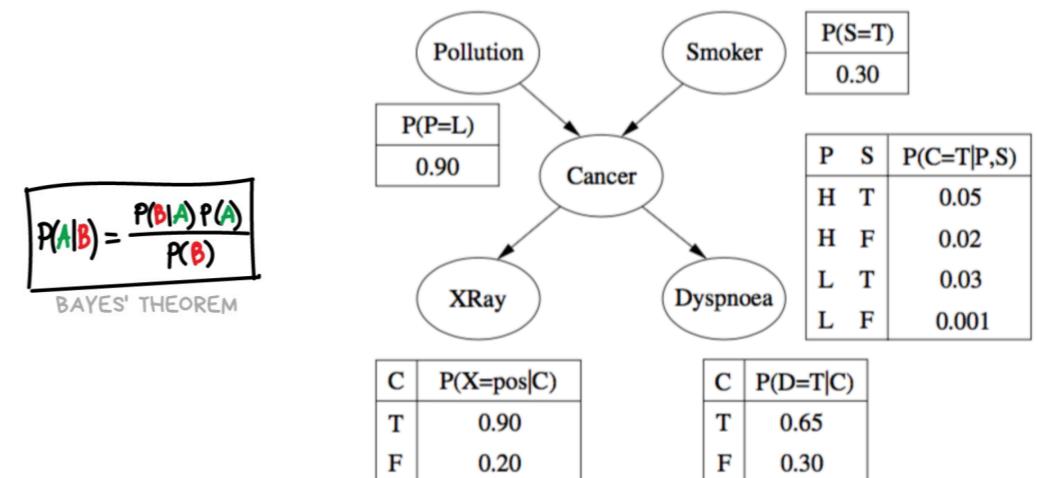
Overview



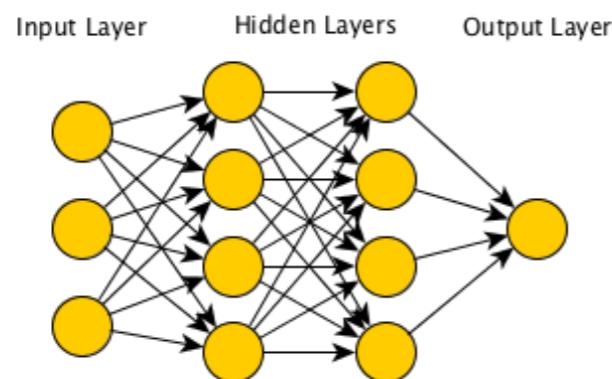
Machine Learning Software



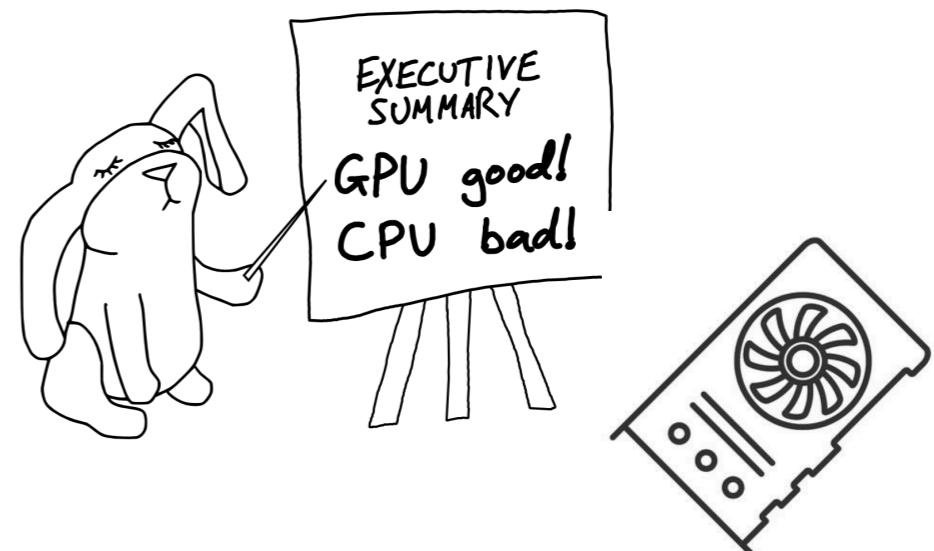
Probabilistic models



Neural networks



Parallelism



Probabilistic programming languages (PPLs)

- PPLs aim to apply the ideas of high-level programming languages to machine learning
- Probabilistic models are defined as programs

```

Sepal.Length Sepal.Width Petal.Length
1          5.1         3.5      1.4
2          4.9         3.0      1.4
3          4.7         3.2      1.3
4          4.6         3.1      1.5
5          5.0         3.6      1.4
6          5.4         3.9      1.7
7          4.6         3.4      1.4
8          5.0         3.4      1.5
9          4.4         2.9      1.4
10         4.9         3.1      1.5
11         5.4         3.7      1.5
12         4.8         3.4      1.6
13         4.8         3.0      1.4
14         4.3         3.0      1.1
15         5.8         4.0      1.2
16         5.7         4.4      1.5
17         5.4         3.9      1.3
18         5.1         3.5      1.4
19         5.7         3.8      1.7
20         5.1         3.8      1.5
21         5.4         3.4      1.7
22         5.1         3.7      1.5
23         4.6         3.6      1.0
24         5.1         3.3      1.7
25         4.8         3.4      1.9

```

```

K, d, N = 5, 10, 200

# model definition
with inf.ProbModel() as m:
    #define the weights
    with inf.replicate(size=K):
        w = inf.models.Normal(0, 1, dim=d)

    # define the generative model
    with inf.replicate(size=N):
        z = inf.models.Normal(0, 1, dim=K)
        x = inf.models.Normal(inf.matmul(z,w),
                             1.0, observed=True, dim=d)

```

data

+

prior model as
a program
(sample generator)

 $p(\theta | \text{data})?$ $\mathcal{N}(3.25, 1.2)$

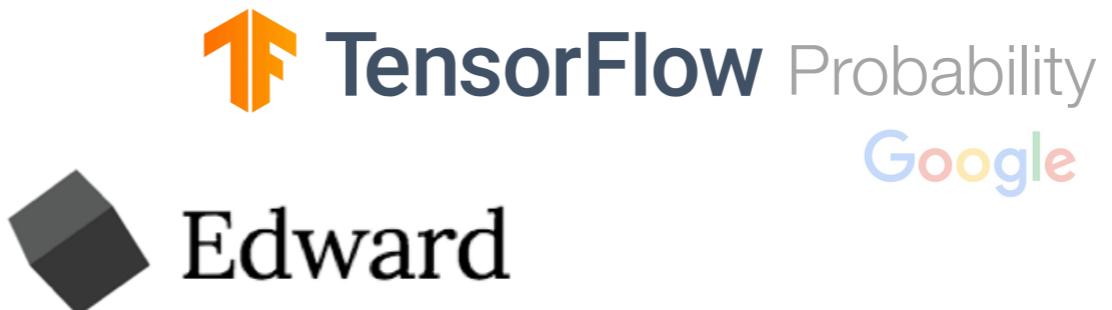
query

blackbox
methods

answer

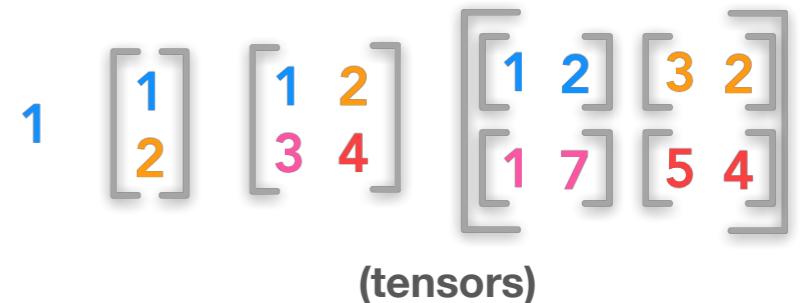
(e.g., Variational Inference)

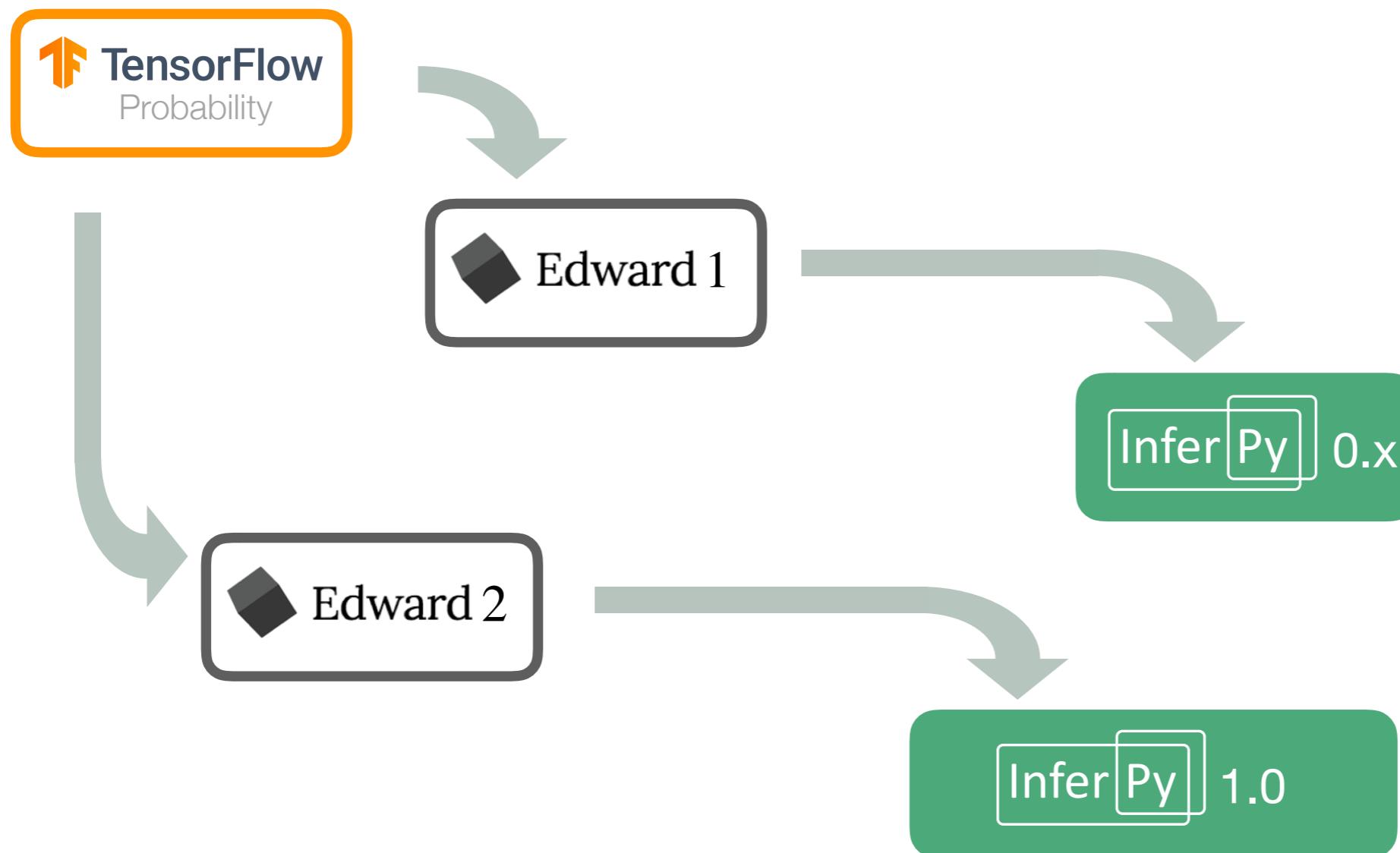
Related software



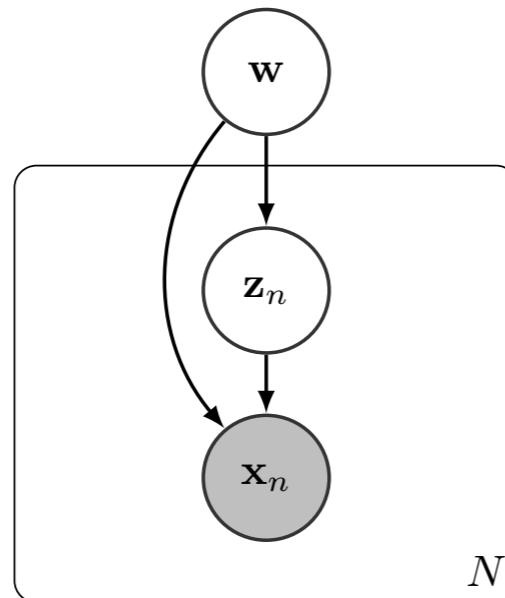
Advantages of InferPy:

- The definition of distributions over tensors is simplified
- No need to have a strong background in inference algorithms





Hierarchical Probabilistic Models



w : global parameters

z : local hidden variables

x : observations

N : number of observations

$p(w)$: prior model

$p(x, z|w)$: data model

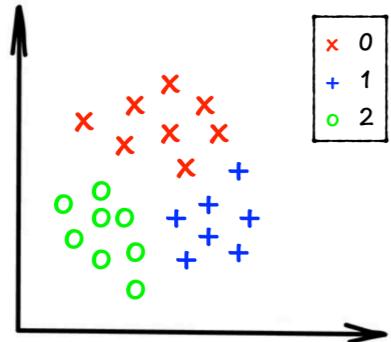
Objective: posterior distribution $p(w, z|x)$

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



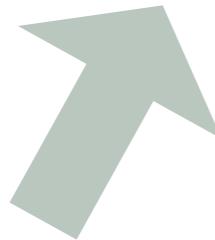
Principal Component Analysis (PCA)

$k = 2$

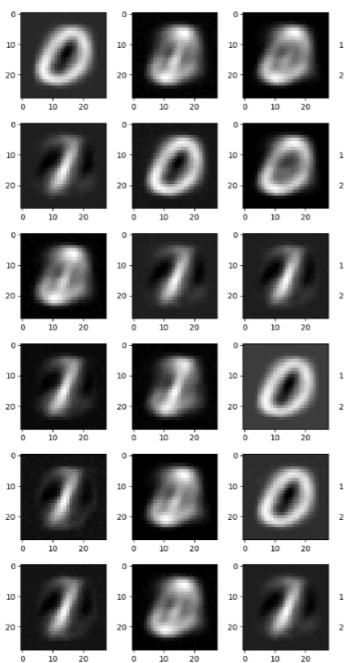
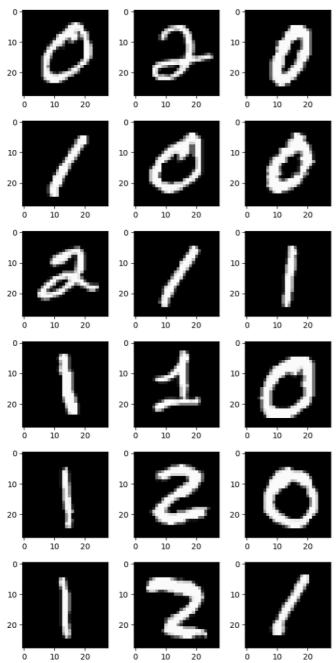


hidden representation

encoding

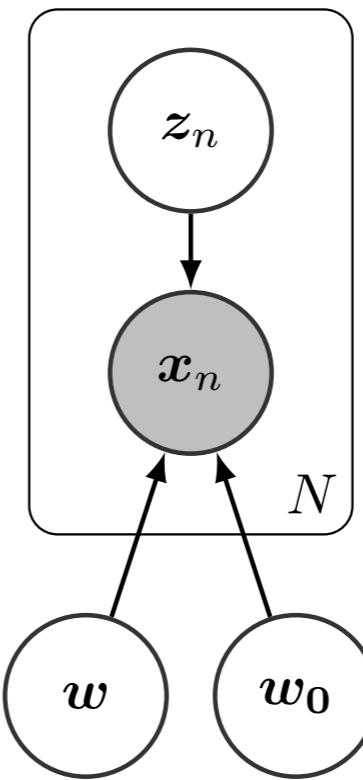


decoding



$$d = 28 * 28 = 784$$

original representation



$$z_n \sim \mathcal{N}_k(0, I)$$

$$x_n \sim \mathcal{N}_d(z_n \cdot w^T + w_0, I)$$

$$w \sim \mathcal{N}_{d \times k}(0, I)$$

$$w_0 \sim \mathcal{N}(0, I)$$

Model definition



```
import inferpy as inf
import tensorflow as tf

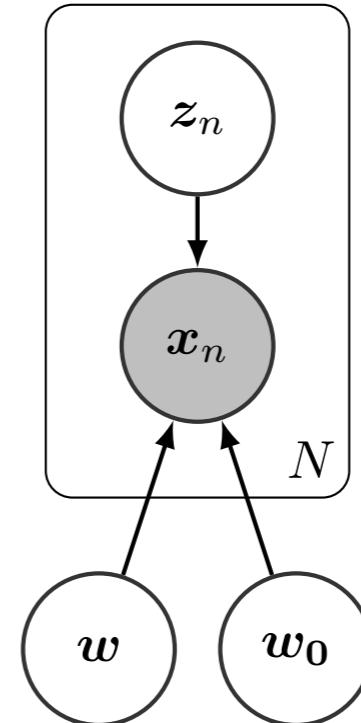
# definition of a generic model
@inf.probmodel
def pca(k,d):
    w = inf.Normal(loc=tf.zeros([k,d]),
                   scale=1, name="w")      # [k,d]

    w0 = inf.Normal(loc=tf.zeros([d]),
                    scale=1, name="w0")    # [d]

    with inf.datamodel():

        z = inf.Normal(tf.zeros([k]),1, name="z") # [N,k]
        x = inf.Normal(z @ w + w0, 1, name="x")  # [N,d]

# create an instance of the model
m = pca(k=2,d=28*28)
```



$$z_n \sim \mathcal{N}_k(0, I)$$

$$x_n \sim \mathcal{N}_d(z_n \cdot w^T + w_0, I)$$

$$w \sim \mathcal{N}_{d \times k}(0, I)$$

$$w_0 \sim \mathcal{N}(0, I)$$

```
In[*]: m.log_prob(m.sample())
```

```
Out[*]:
```

```
{'w': array([-0.96455336, -1.51091952, ..., -1.37483748]),
 'w0': array([-0.84543316]),
 'z': array([-2.04824067, ..., -0.98934147]),
 'x': array([-99.06889862, -16.68617233, ..., -1.40110224])}
```

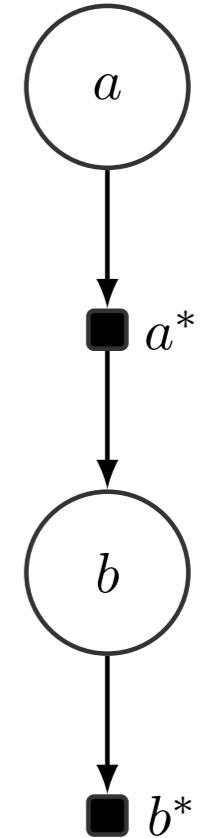
- Random variables in InferPy encapsulate another from Edward 2
- These can follow the following distributions

```
In[*]: inf.models.random_variable.distributions_all  
Out[*]: ['Autoregressive', 'Bernoulli', 'Beta', 'Binomial',  
'Categorical', 'Cauchy', 'Chi2', 'ConditionalTransformedDistribution',  
'Deterministic', 'Dirichlet', 'DirichletMultinomial',  
'ExpRelaxedOneHotCategorical', 'Exponential', 'Gamma', 'Geometric',  
'HalfNormal', 'Independent', 'InverseGamma', 'Kumaraswamy', 'Laplace',  
'Logistic', 'Mixture', 'MixtureSameFamily', 'Multinomial',  
'MultivariateNormalDiag', 'MultivariateNormalFullCovariance',  
'MultivariateNormalTriL', 'NegativeBinomial', 'Normal',  
'OneHotCategorical', 'Poisson', 'PoissonLogNormalQuadratureCompound',  
'QuantizedDistribution', 'RelaxedBernoulli', 'RelaxedOneHotCategorical',  
'SinhArcsinh', 'StudentT', 'TransformedDistribution', 'Uniform',  
'VectorDeterministic', 'VectorDiffeomixture', 'VectorExponentialDiag',  
'VectorLaplaceDiag', 'VectorSinhArcsinhDiag', 'Wishart']
```

```
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]
```

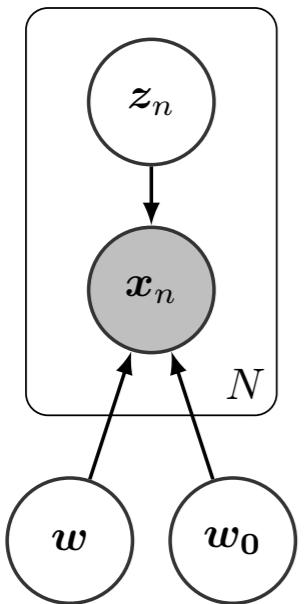


a continuous variable might be parent of a discrete one

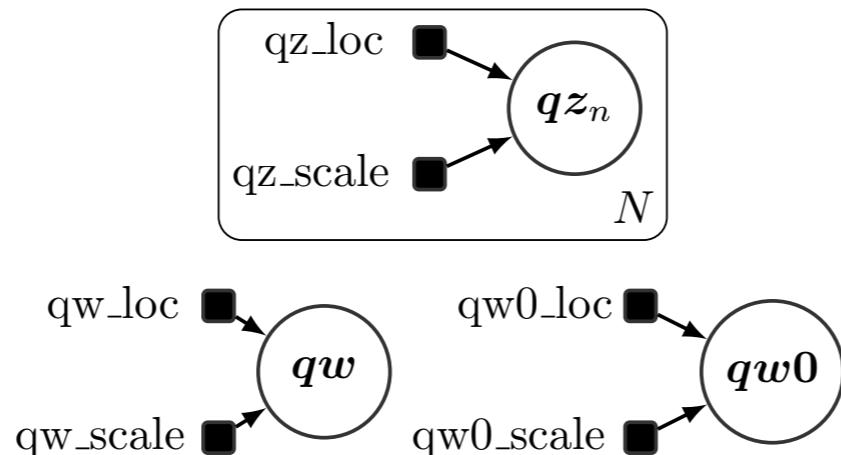
```
x = inf.Normal(0, 1)  
c = inf.Categorical(probs=tf.case({ x > 0: lambda : [0.0, 1.0],  
                                     x <= 0: lambda : [1.0, 0.0]}))
```

Variational Inference (VI)

P model



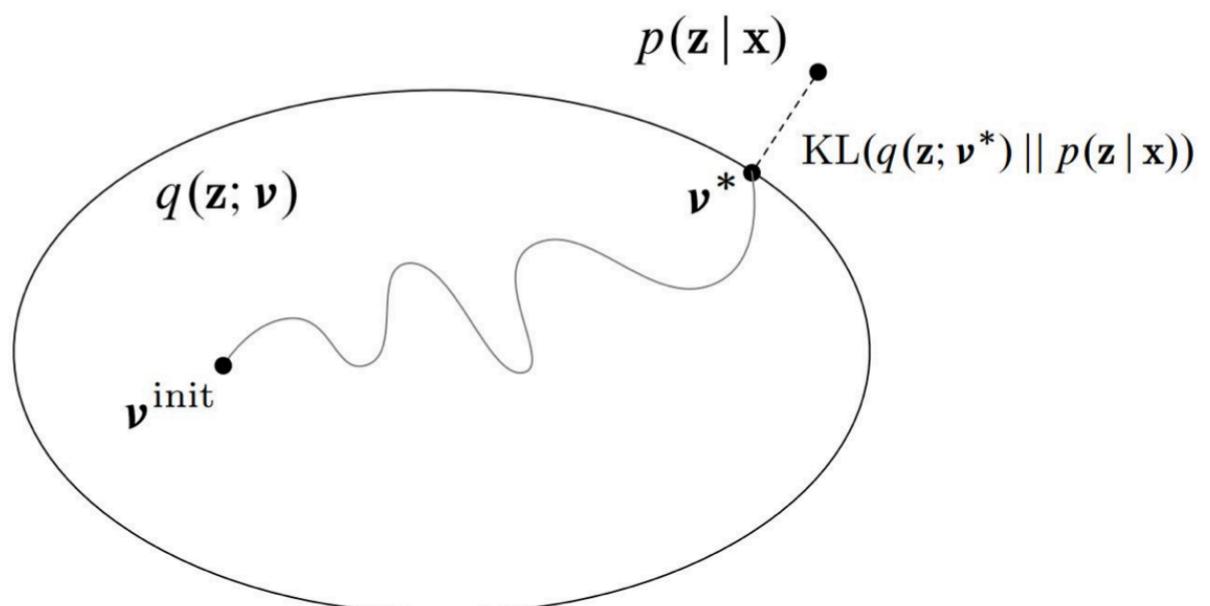
Q model



data

	Sepal.Length	Sepal.Width	Petal.Length
1	5.1	3.5	1.4
2	4.9	3.0	1.4
3	4.7	3.2	1.3
4	4.6	3.1	1.5
5	5.0	3.6	1.4
6	5.4	3.9	1.7
7	4.6	3.4	1.4
8	5.0	3.4	1.5
9	4.4	2.9	1.4
10	4.9	3.1	1.5
11	5.4	3.7	1.5
12	4.8	3.4	1.6
13	4.8	3.0	1.4
14	4.3	3.0	1.1
15	5.8	4.0	1.2
16	5.7	4.4	1.5
17	5.4	3.9	1.3
18	5.1	3.5	1.4
19	5.7	3.8	1.7
20	5.1	3.8	1.5
21	5.4	3.4	1.7
22	5.1	3.7	1.5
23	4.6	3.6	1.0
24	5.1	3.3	1.7
25	4.8	3.4	1.9

- Inference turns into an optimisation problem

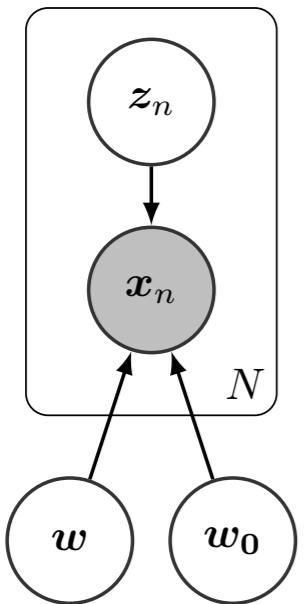


Inference (of the parameters)

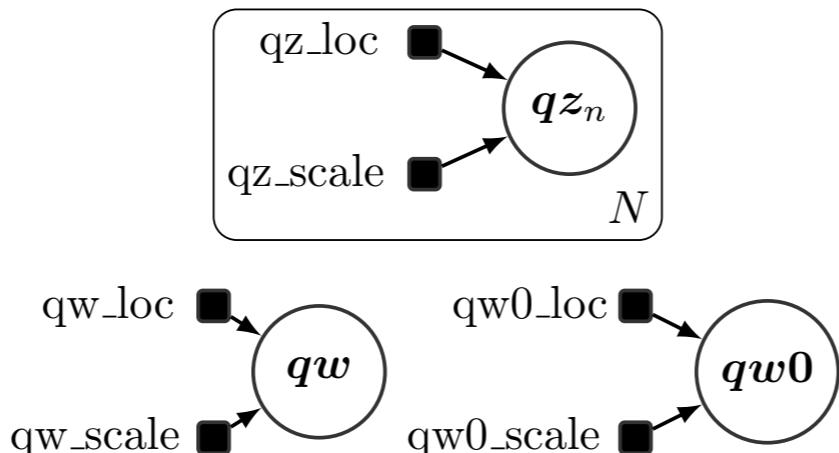


Variational Inference (VI)

P model

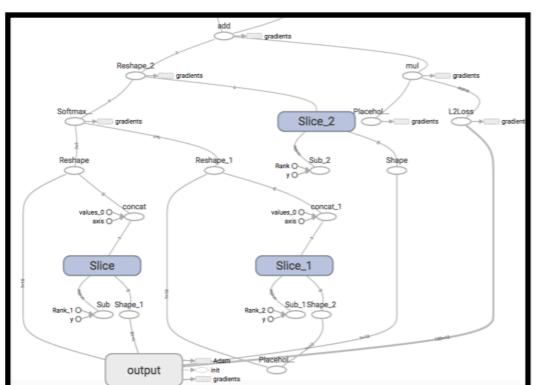


Q model



data

	Sepal.Length	Sepal.Width	Petal.Length
1	5.1	3.5	1.4
2	4.9	3.0	1.4
3	4.7	3.2	1.3
4	4.6	3.1	1.5
5	5.0	3.6	1.4
6	5.4	3.9	1.7
7	4.6	3.4	1.4
8	5.0	3.4	1.5
9	4.4	2.9	1.4
10	4.9	3.1	1.5
11	5.4	3.7	1.5
12	4.8	3.4	1.6
13	4.8	3.0	1.4
14	4.3	3.0	1.1
15	5.8	4.0	1.2
16	5.7	4.4	1.5
17	5.4	3.9	1.3
18	5.1	3.5	1.4
19	5.7	3.8	1.7
20	5.1	3.8	1.5
21	5.4	3.4	1.7
22	5.1	3.7	1.5
23	4.6	3.6	1.0
24	5.1	3.3	1.7
25	4.8	3.4	1.9



Computational graph of the *ELBO*

(to maximise)

$$\text{loss}(P, Q, \text{data}) = -\text{ELBO}(P, Q, \text{data}) \quad (\text{to minimise})$$

Inference (of the parameters)



```
@inf.probmodel
def qmodel(k,d):

    qw_loc = inf.Parameter(tf.zeros([k,d]), name="qw_loc")
    qw_scale = tf.math.softplus(inf.Parameter(tf.ones([k,d]), name="qw_scale"))
    qw = inf.Normal(qw_loc, qw_scale, name="w")

    qw0_loc = inf.Parameter(tf.ones([d]), name="qw0_loc")
    qw0_scale = tf.math.softplus(inf.Parameter(tf.ones([d]), name="qw0_scale"))
    qw0 = inf.Normal(qw0_loc, qw0_scale, name="w0")

    with inf.datamodel():

        qz_loc = inf.Parameter(np.zeros([k]), name="qz_loc")
        qz_scale = tf.math.softplus(inf.Parameter(tf.ones([k]), name="qz_scale"))
        qz = inf.Normal(qz_loc, qz_scale, name="z")

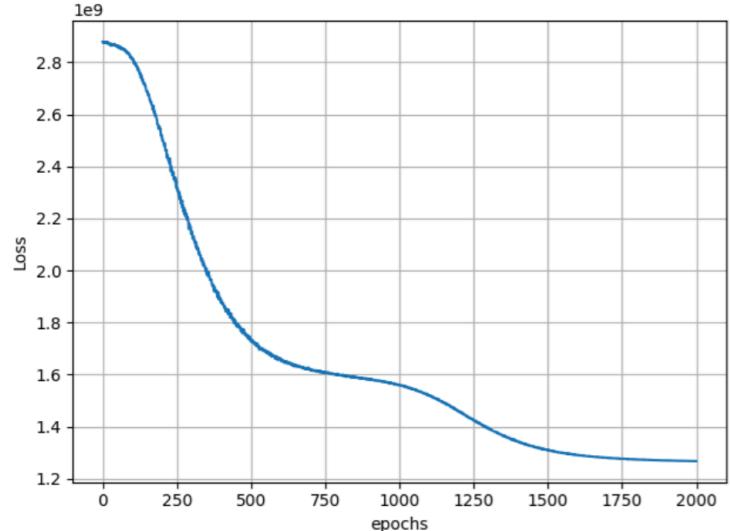
# instance of the Q model
q = qmodel(k=2,d=28*28)

# inference
optimizer = tf.train.AdamOptimizer(learning_rate=0.01)

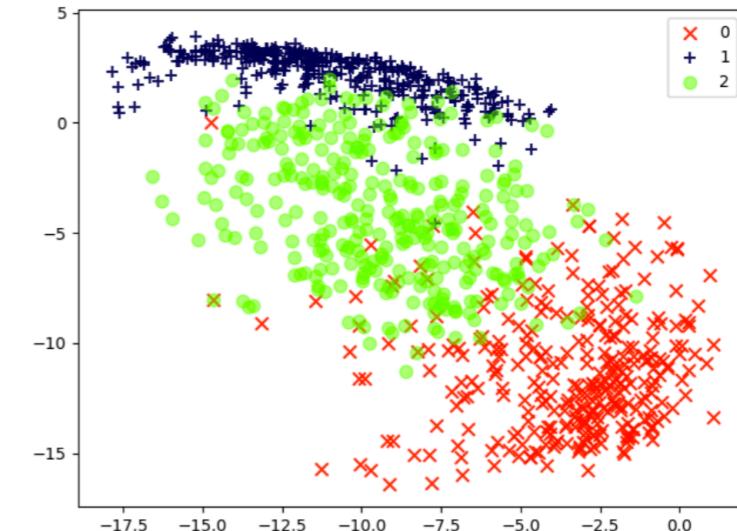
VI = inf.inference.VI(q, optimizer=optimizer, epochs=2000)

m.fit({"x": x_train}, VI)
```

After the inference

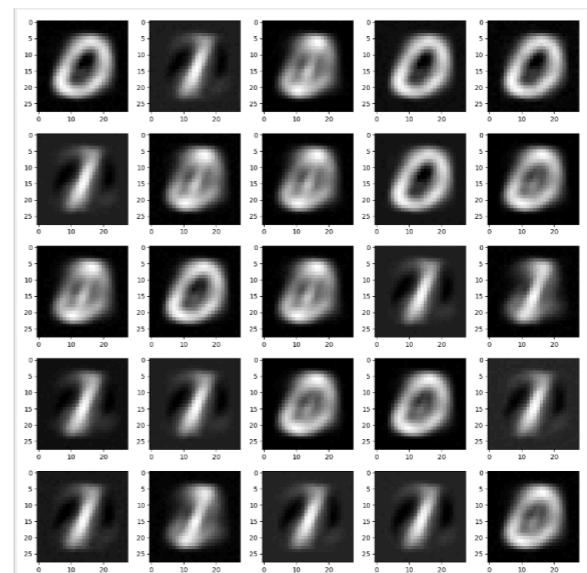


```
# extract the loss evolution  
L = VI.losses
```

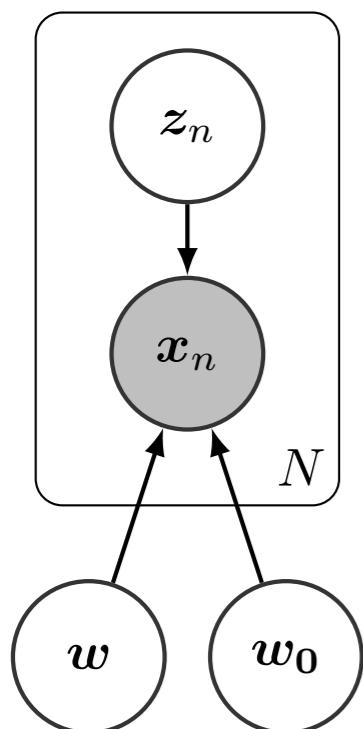


```
# extract the hidden encoding  
sess = inf.get_session()  
sess.run(m.posterior["z"].loc)
```

```
post_samples = m.post_predictive_sample()  
mnist.plot_digits(post_samples["x"], [5,5])
```



Linear PCA

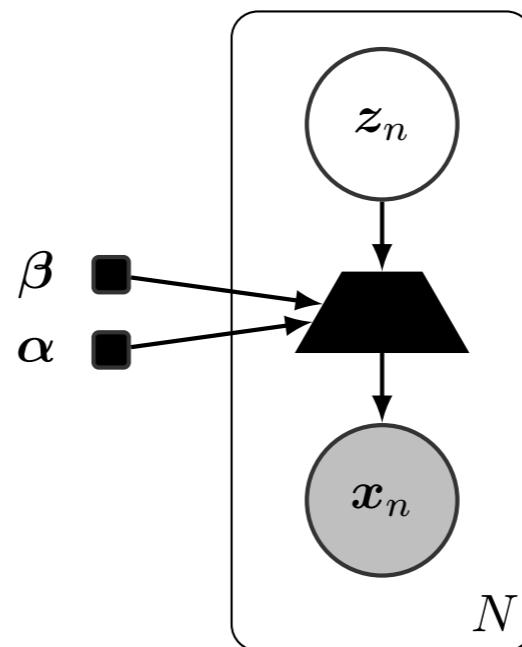


$$z_n \sim \mathcal{N}_k(0, I)$$

$$x_n \sim \mathcal{N}_d(z_n \cdot w^T + w_0, I)$$

$$\begin{aligned} w &\sim \mathcal{N}_{d \times k}(0, I) \\ w_0 &\sim \mathcal{N}(0, I) \end{aligned}$$

Non-Linear PCA



$$z_n \sim \mathcal{N}_k(0, I)$$

$$x_n \sim \mathcal{N}_d(f(z_n : \beta, \alpha), I)$$

- Simply change the functions generating the parameters
- Demo in jupyter notebook

Documentation



A screenshot of a web browser displaying the Python Package Index (PyPI) page for the "inferpy" package. The page header shows the URL "pypi.org" and the package name "inferpy 1.0.0". Below the header, there is a search bar labeled "Search projects" and navigation links for "Help", "Donate", "Log in", and "Register". A prominent blue button with the text "pip install inferpy" and a download icon is visible. To the right of the button, a green button indicates it is the "Latest version". Below the main header, the text "Probabilistic modeling with Tensorflow made easy" is displayed. On the left side, there is a "Navigation" sidebar with links for "Project description", "Release history", and "Download files". The "Project description" link is currently selected and highlighted in blue. The main content area contains the title "InferPy: Deep Probabilistic Modeling Made Easy" and a brief description: "InferPy is a high-level API for probabilistic modeling written in Python and capable of running on top of Edward and Tensorflow. InferPy's API is strongly inspired by Keras and it".

<https://pypi.org/project/inferpy/>

```
$ pip install inferpy
```



The screenshot shows the InferPy Documentation page at version 0.2.2. The left sidebar contains links for 'QUICK START', 'Getting Started', 'Guiding Principles', 'Requirements', 'GUIDES', 'Guide to Building Probabilistic Models', 'Guide to Approximate Inference', 'Guide to Model Validation', 'Guide to Data Handling', 'MODULE ZOO', 'Probabilistic Model Zoo', 'InferPy vs Edward', 'PACKAGE REFERENCE', 'inferpy package', and 'inferpy.criticism package'. The main content area features the InferPy logo and a brief introduction: 'InferPy is a high-level API for probabilistic modeling written in Python and capable of running on top of Tensorflow. InferPy's API is strongly inspired by Keras and it has a focus on enabling flexible data processing, easy-to-code probabilistic modeling, scalable inference and robust model validation.' It also lists reasons to use InferPy: 'Use InferPy if you need a probabilistic programming language that: • Allows easy and fast prototyping of hierarchical probabilistic models with a simple and user friendly API inspired by Keras.'

<https://inferpy.readthedocs.io>

The screenshot shows the GitHub repository page for 'PGM-Lab / InferPy'. It displays basic repository statistics: 354 commits, 8 branches, 9 releases, and 2 contributors. The 'Code' tab is selected. A list of recent commits is shown, including: 'roberasdepaz Community section in readme' (latest commit 36e6851 on Feb 6), 'docker docker realtime' (4 months ago), 'docs Update protzoo.rst' (2 months ago), 'examples bug: solved issue with Poisson #110' (2 months ago), 'inferpy preparing 0.2.1' (2 months ago), 'modules concat' (6 months ago), 'notebooks/basic_tutorial clean' (10 months ago), 'tests fix replicate test' (4 months ago), and 'ignore batches param #100' (4 months ago).

<https://github.com/PGM-Lab/InferPy>

The screenshot shows the article 'InferPy: Probabilistic modeling with Tensorflow made easy' published in 'Knowledge-Based Systems' (Volume 168, 2019) pages 25-27. The article is authored by Rafael Cabañas, Antonio Salmerón, and Andrés R. Masegosa from the University of Almería, Spain. The abstract states: '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 layer that provides 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.' The article history includes: Received 2 October 2018, Revised 13 November 2018, Accepted 23 December 2018, Available online 31 December 2018. The journal homepage is www.elsevier.com/locate/knbsys.

Cabañas, R., Salmerón, A., & Masegosa, A. R. (2019). **InferPy: Probabilistic modeling with Tensorflow made easy.** *Knowledge-Based Systems*, 168, 25-27.