InferPy

Probabilistic Modeling with Tensorflow Made Easy

Rafael Cabañas

PIDSIA

Manno, 28.05.19

UNIVERSIDAD DE ALMERÍA

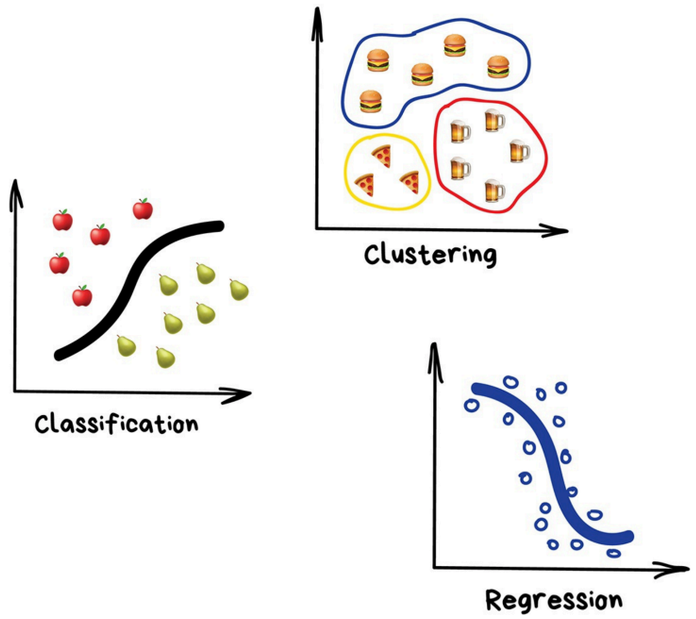
USI/SUPSI
IDSIA

GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

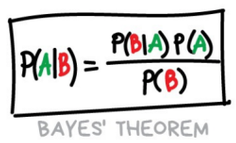
UNIÓN EUROPEA
Fondo Europeo de Desarrollo Regional
"Una manera de hacer Europa"

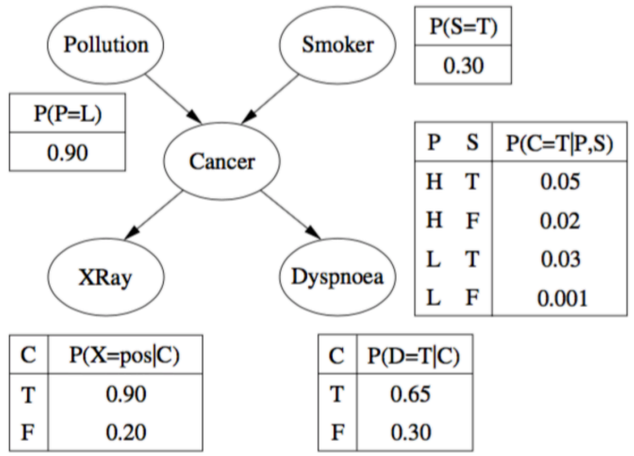
Overview

Machine Learning applications

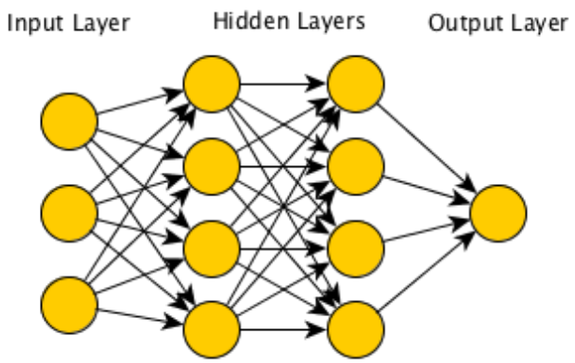


Probabilistic models

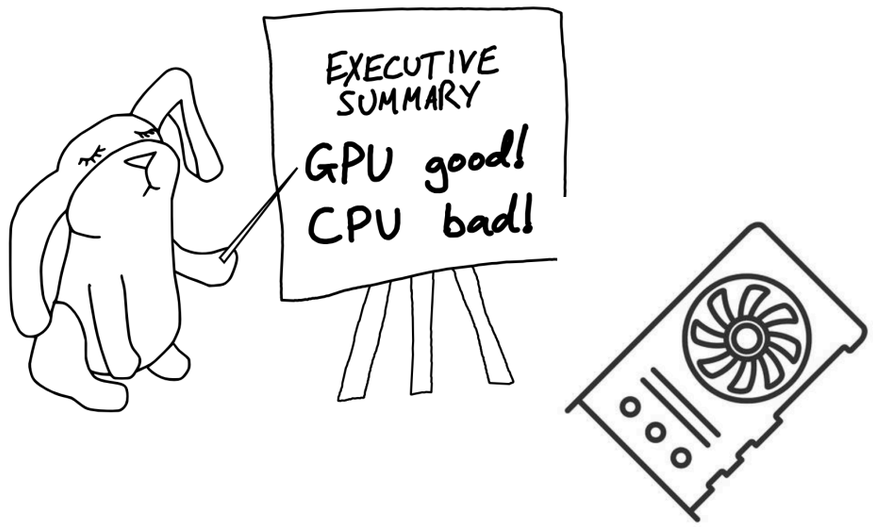




Neural networks



Parallelism



InferPy

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)
```

$p(\theta \mid data)?$

$\mathcal{N}(3.25,1.2)$

data + **model as a program (samples generator)** + **query** → **blackbox methods** → **answer**

(e.g., Variational Inference)

InferPy

Related software

PyTorch
facebook research

TensorFlow Probability
Google

Edward

Pyro
UBER AI Labs

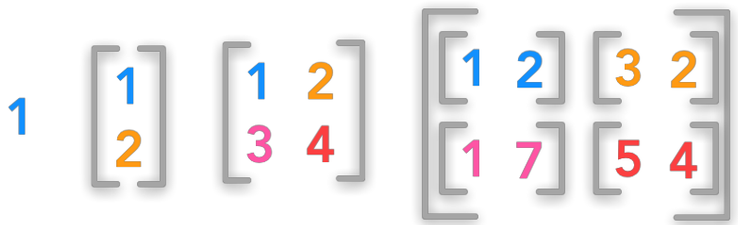
PyMC3

theano

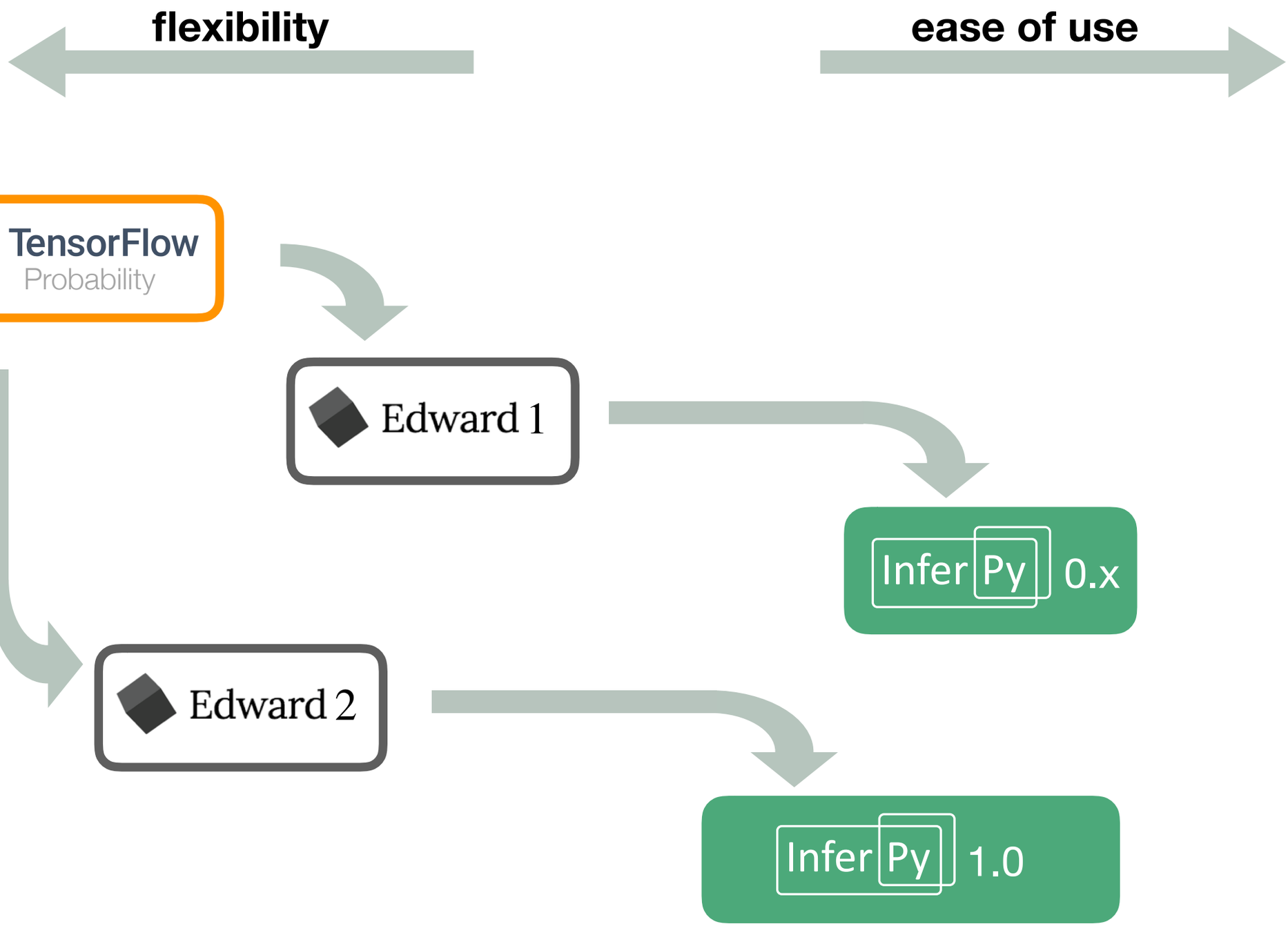
Stan

Advantages of InferPy:

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

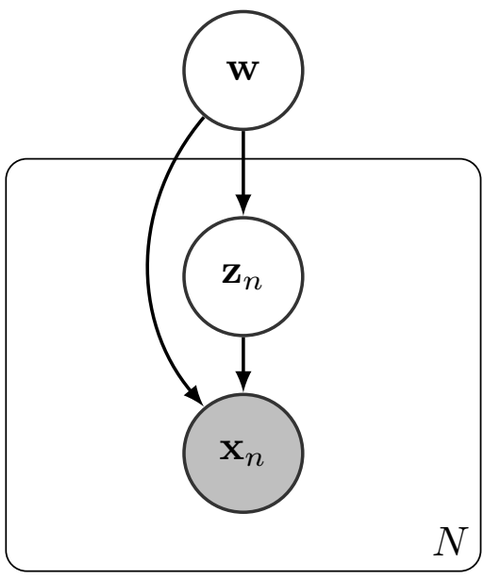


Tensorflow Probability and Edward



InferPy

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

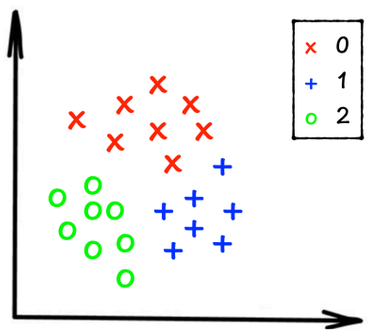
Objective: posterior distribution $p(\mathbf{w}, \mathbf{z}|\mathbf{x})$

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

TensorFlow Keras

Running example

Factor Analysis (PCA)



k = 2

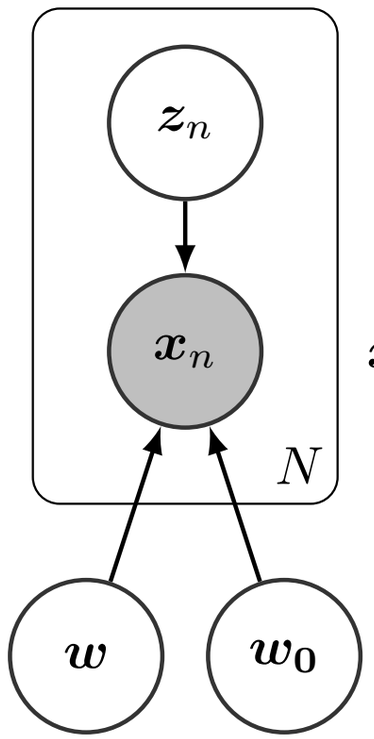
hidden representation

encoding

decoding

d = 28 * 28 = 784

original representation



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

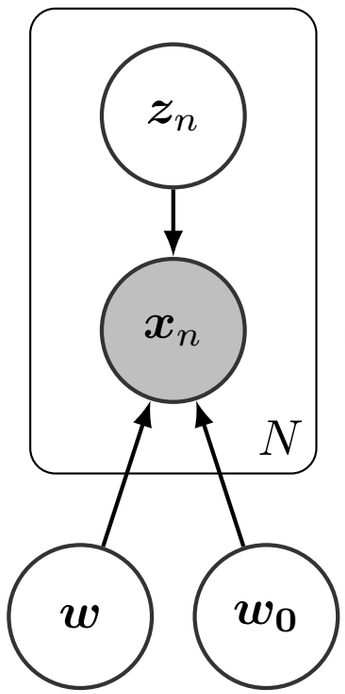
$$x_n \sim \mathcal{N}_d(z_n \cdot w^T + w_0, I)$$

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

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

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]
```



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

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

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

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

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

```
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]])}
```

Model definition

- Random variables in InferPy encapsulate another from Edward 2

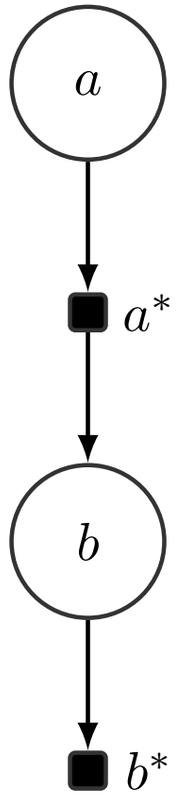
- These can follow the following distributions

```
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']
```

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]
```

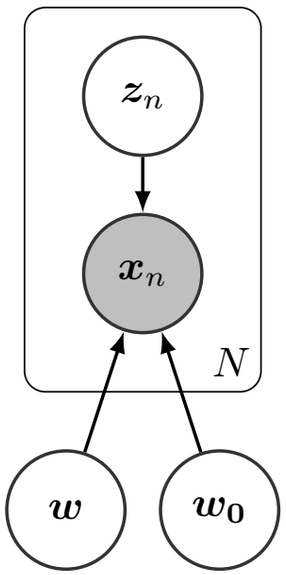



```
# a continuous variable might be parent of a discrete one
 x = inf.Normal(0, 1)
 c = inf.Categorical(probs=tf.case({ x > 0: lambda : [0.0, 1.0],
                                     x <= 0: lambda : [1.0, 0.0]}))
```

Inference (of the parameters)

Variational Inference (VI)

P model

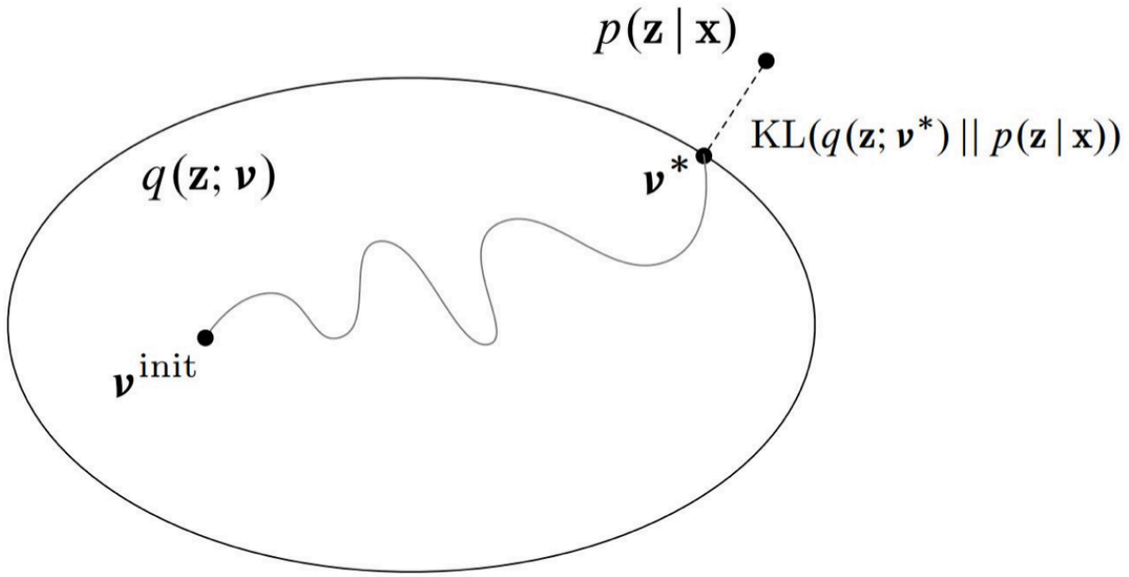


Q model

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

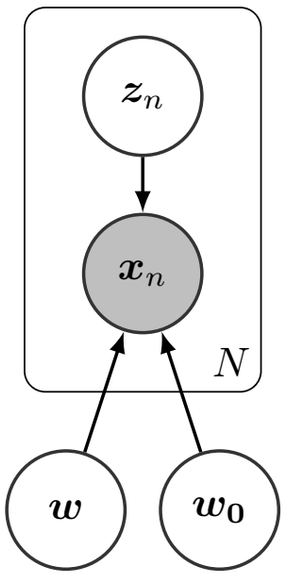
- Inference turns into an optimisation problem



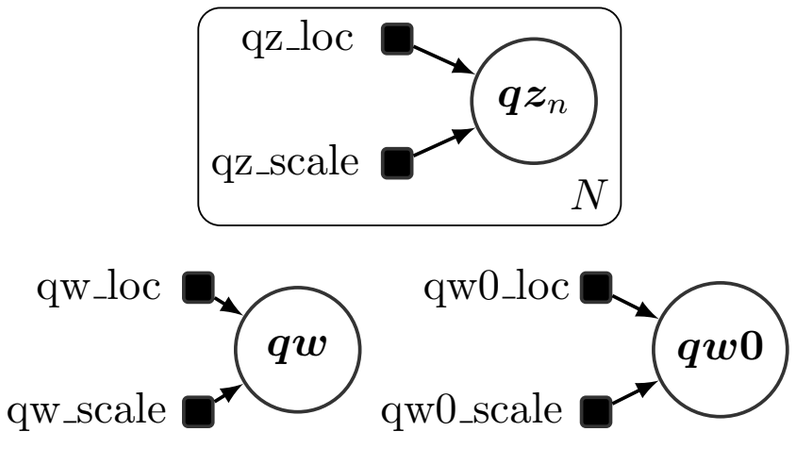
Inference (of the parameters)

Variational Inference (VI)

P model

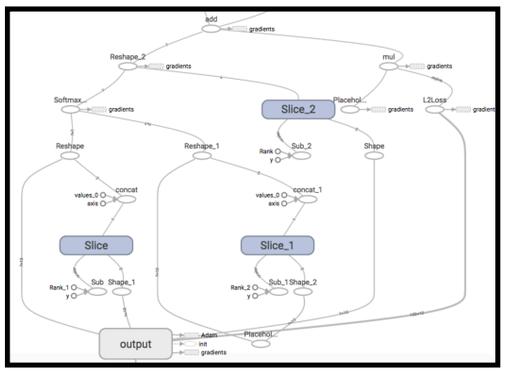


Q model



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



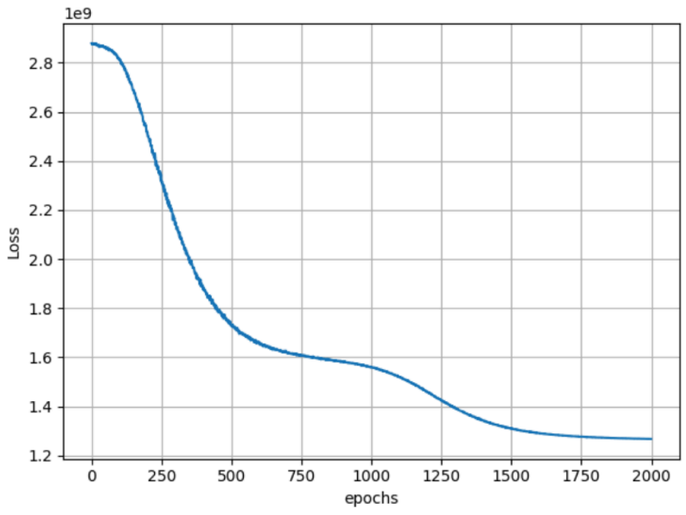
Computational graph of the $ELBO$ **(to maximise)**

$$loss(P, Q, data) = -ELBO(P, Q, data)$$ **(to minimise)**

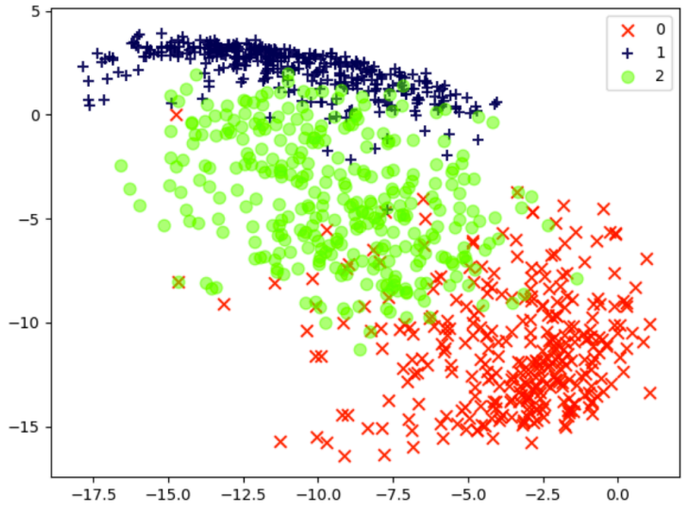
Inference (of the parameters)

```
@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

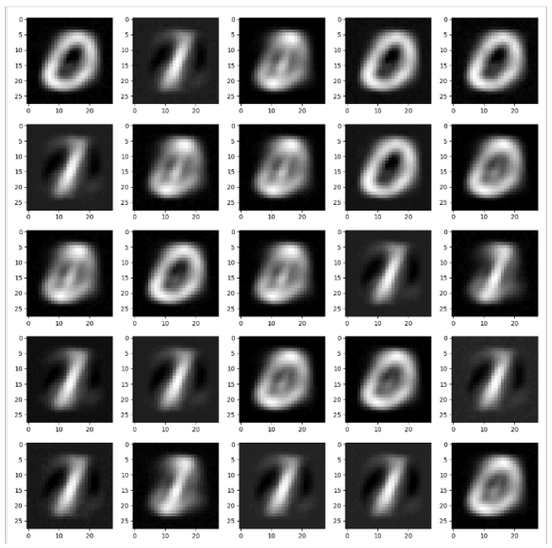



```
# extract the loss evolution
L = VI.losses
```



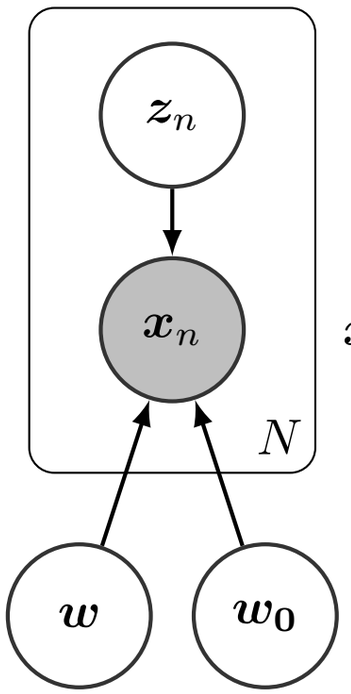

```
# extract the hidden encoding
sess = inf.get_session()
sess.run(m.posterior["z"].loc)
```

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



Models with NNs

Linear PCA



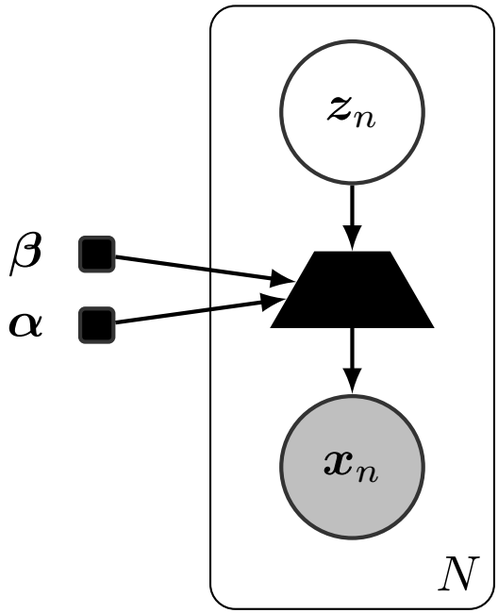
$$z_n \sim \mathcal{N}_k(0, I)$$

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

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

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

Non-Linear PCA

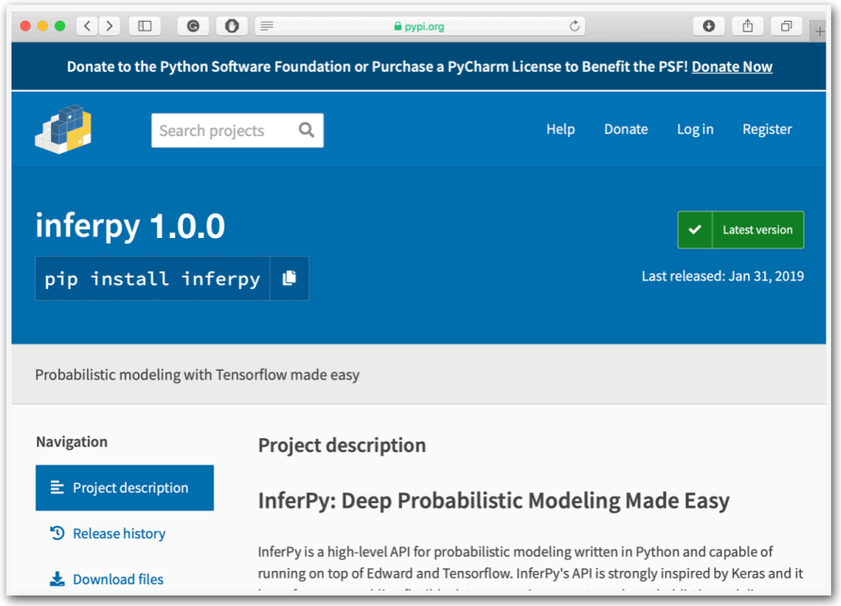


$$z_n \sim \mathcal{N}_k(0, I)$$

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

- The same inference methods can be used
- Demo in jupyter notebook

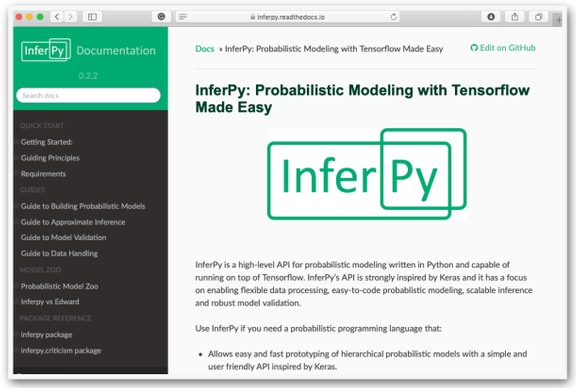
Documentation



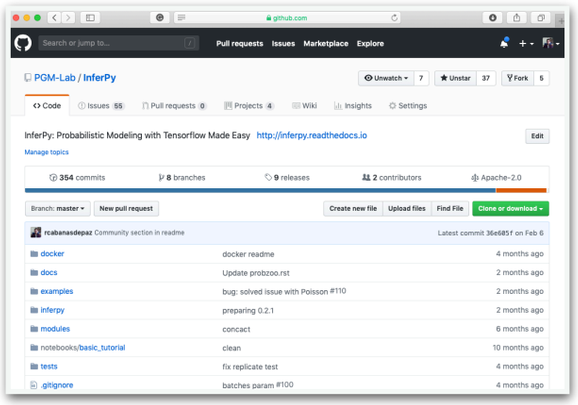
https://pypi.org/project/inferpy/

```
$ pip install inferpy
```

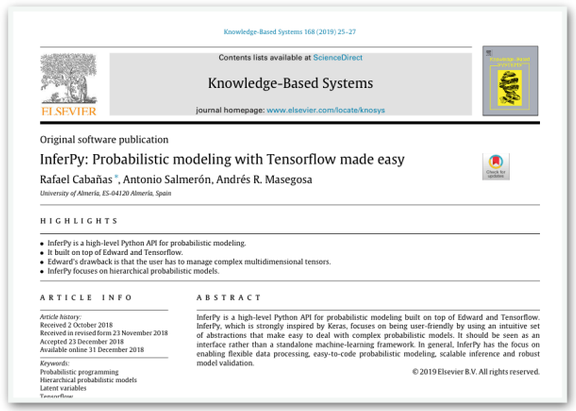
Documentation



https://inferpy.readthedocs.io



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



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