InferPy

Probabilistic Graphical Models with Neural Networks in InferPy

PGM 2020 - Aalborg, 23 - 25 September

Rafael Cabañas

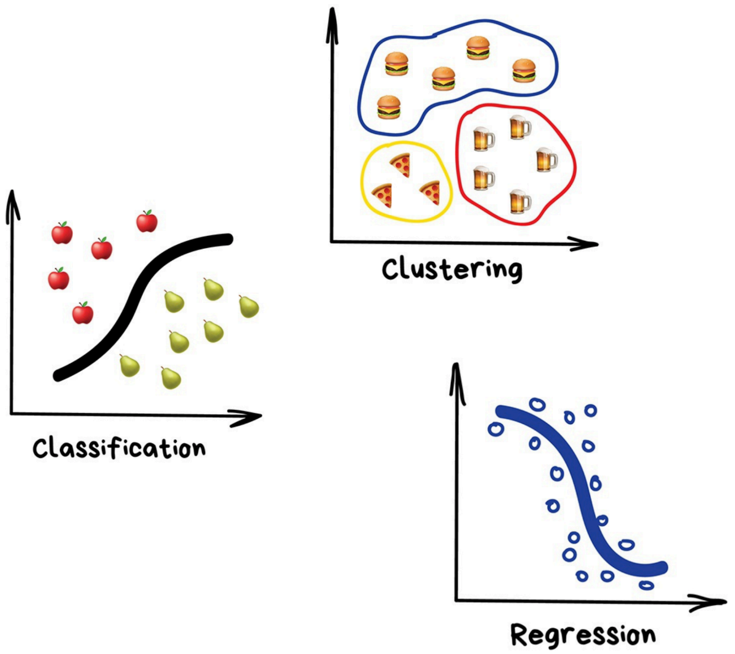
USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

Javier Cózar
Antonio Salmerón
Andrés R. Masegosa

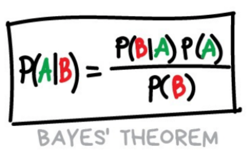
Universidad de Almería

Gobierno de España
Ministerio de Ciencia, Innovación y Universidades

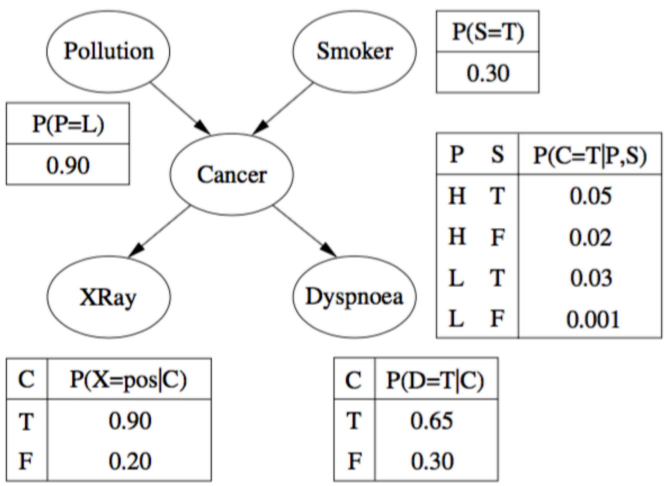
Unión Europea
Fondo Europeo de Desarrollo Regional
"Una manera de hacer Europa"



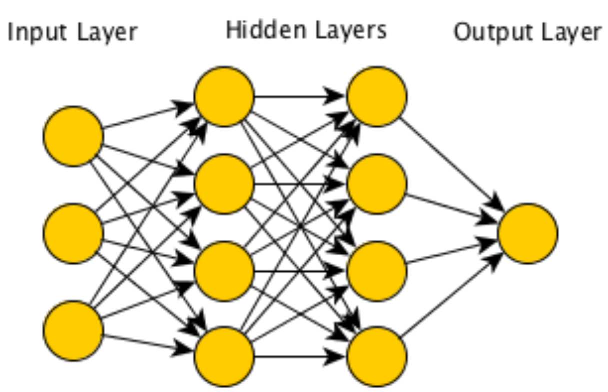
Clustering
Classification
Regression



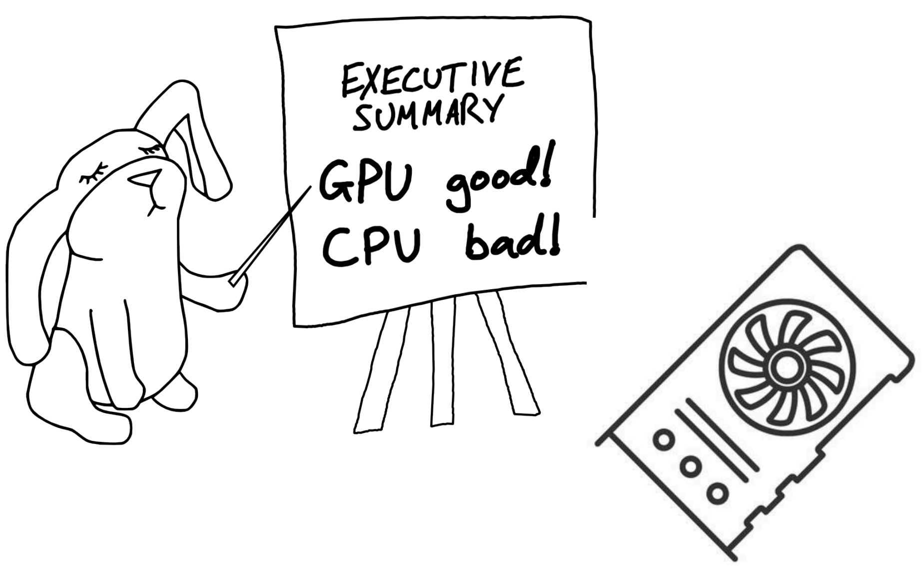
P(A|B) = P(B|A) P(A) / P(B)
BAYES' THEOREM



Pollution
Smoker
P(S=T)
0.30
P(P=L)
0.90
Cancer
P S P(C=T|P,S)
H T 0.05
H F 0.02
L T 0.03
L F 0.001
XRay
Dyspnoea
C P(X=pos|C)
T 0.90
F 0.20
C P(D=T|C)
T 0.65
F 0.30



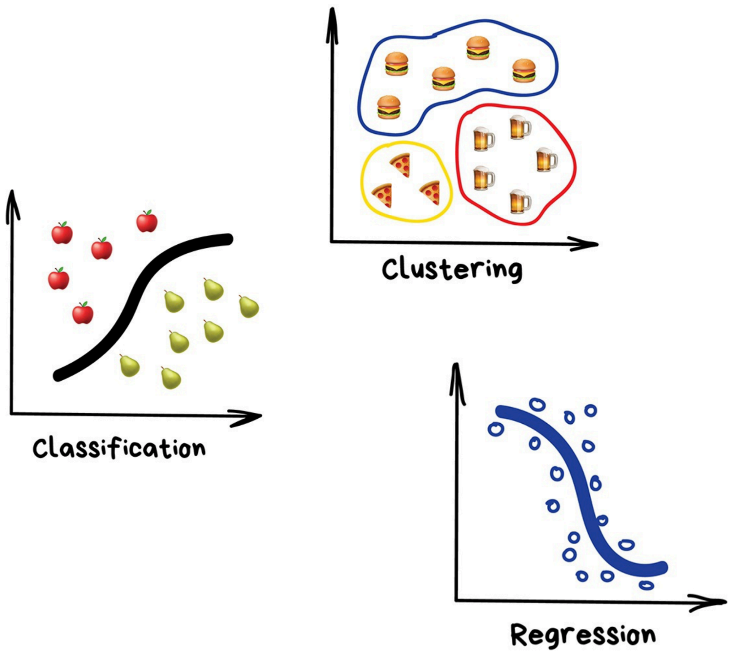
Input Layer
Hidden Layers
Output Layer

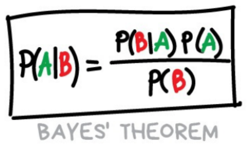


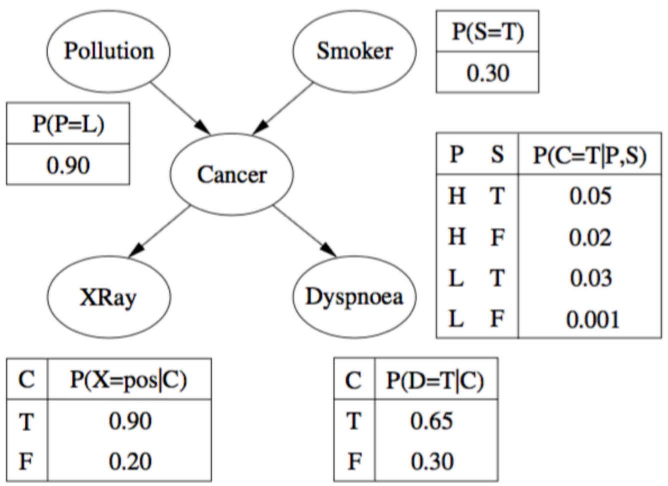
EXECUTIVE SUMMARY
GPU good!
CPU bad!

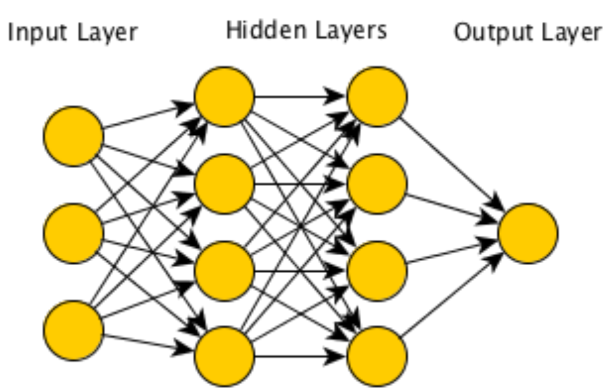
Overview

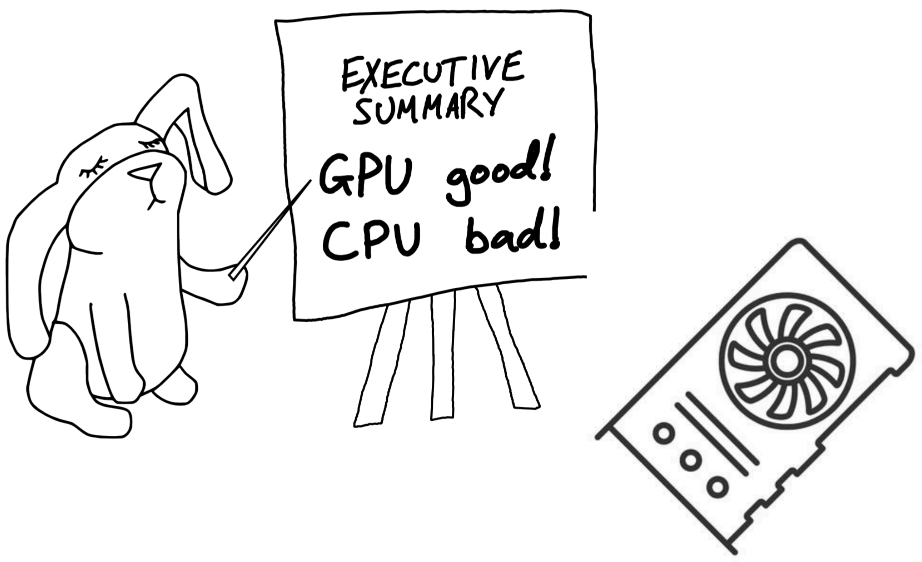
Machine Learning Software





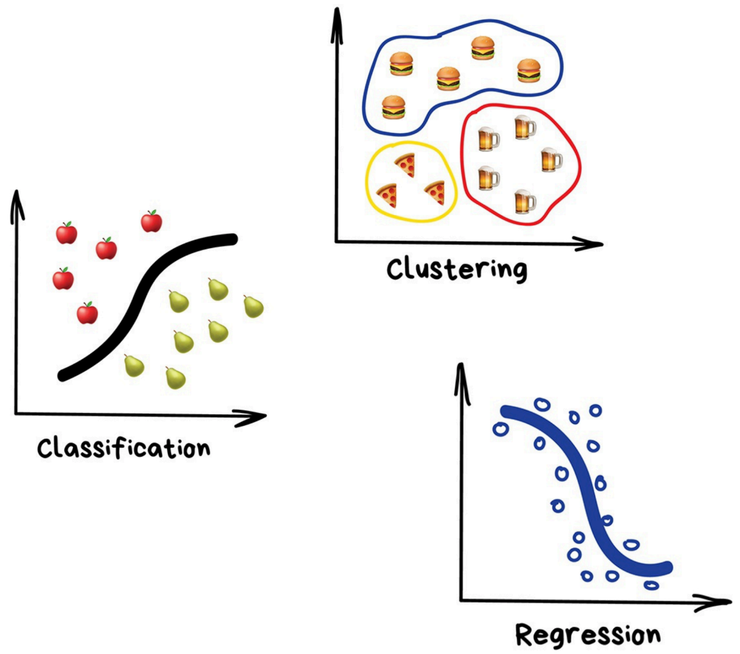


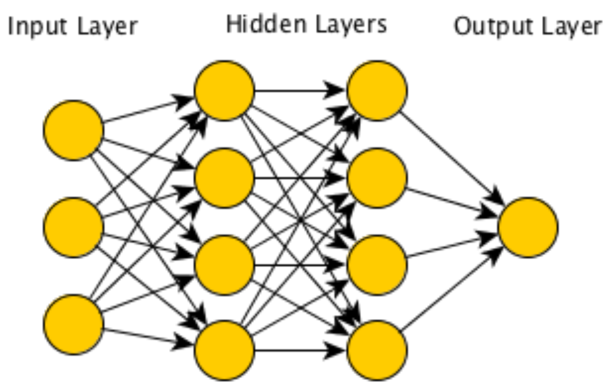




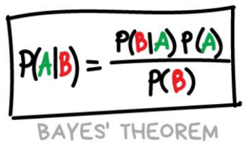
Overview

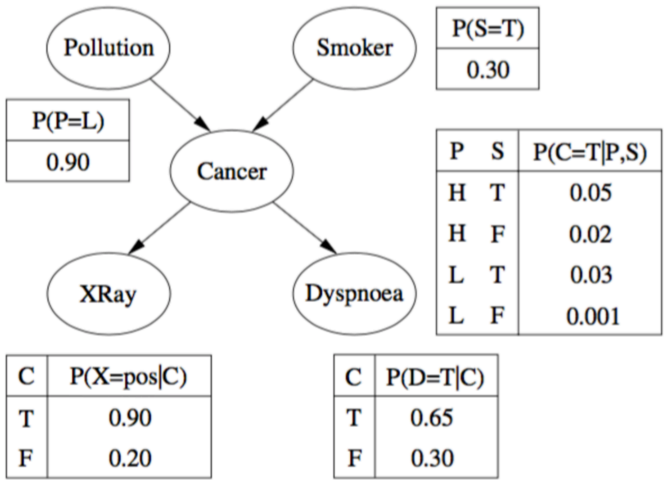
Machine Learning Software

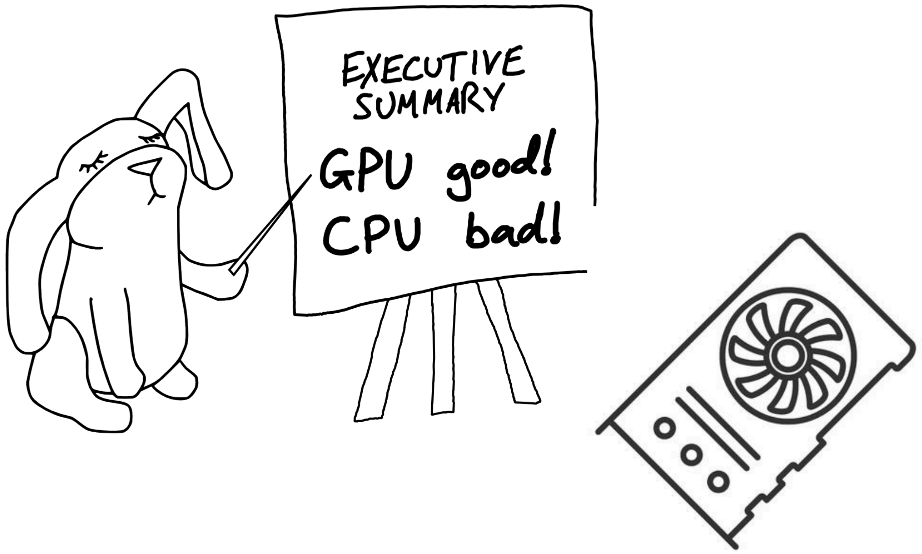




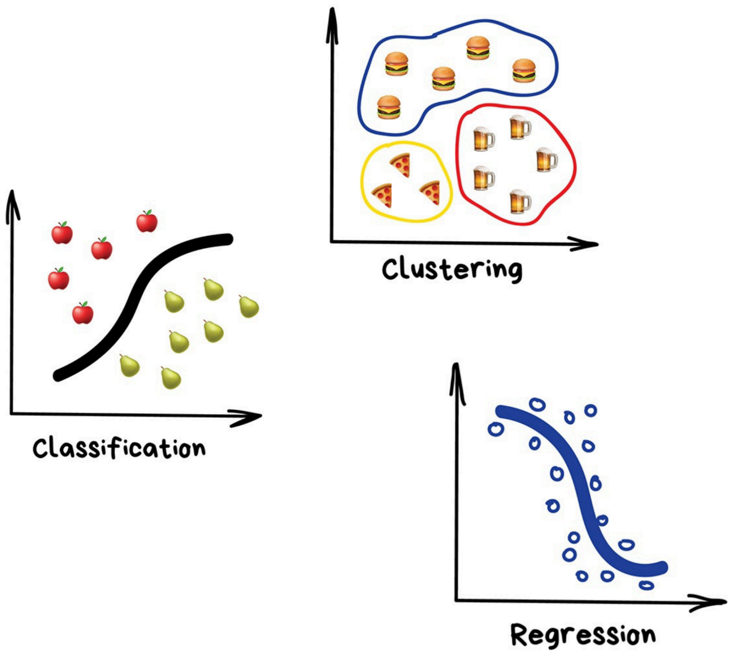
Probabilistic models



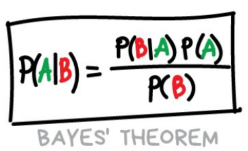


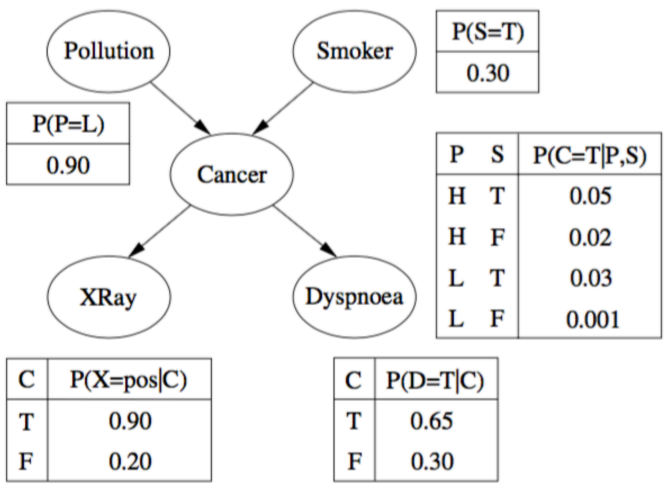


Machine Learning Software

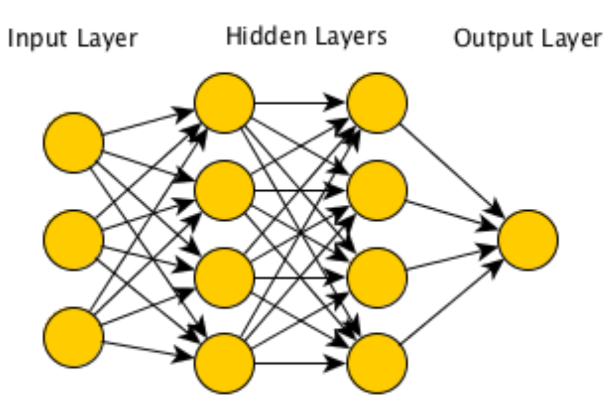


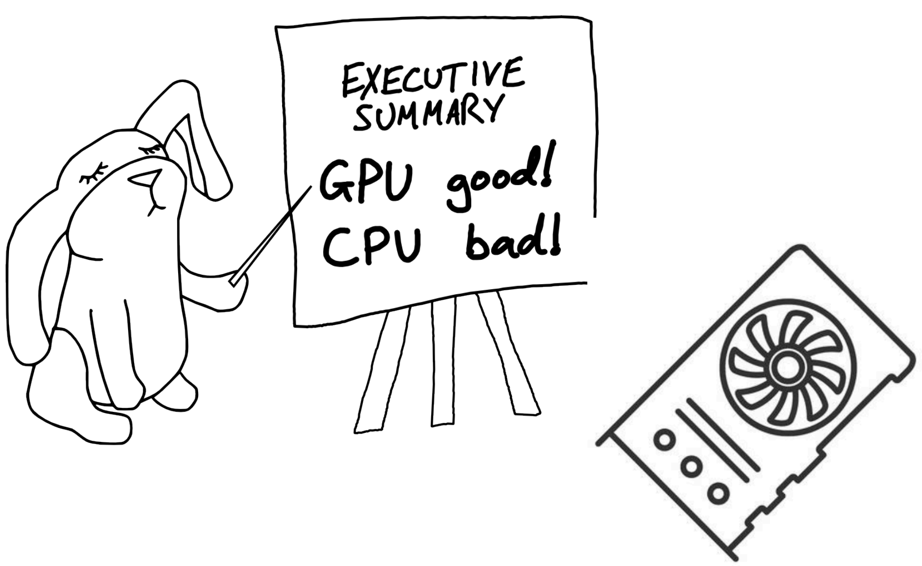
Probabilistic models



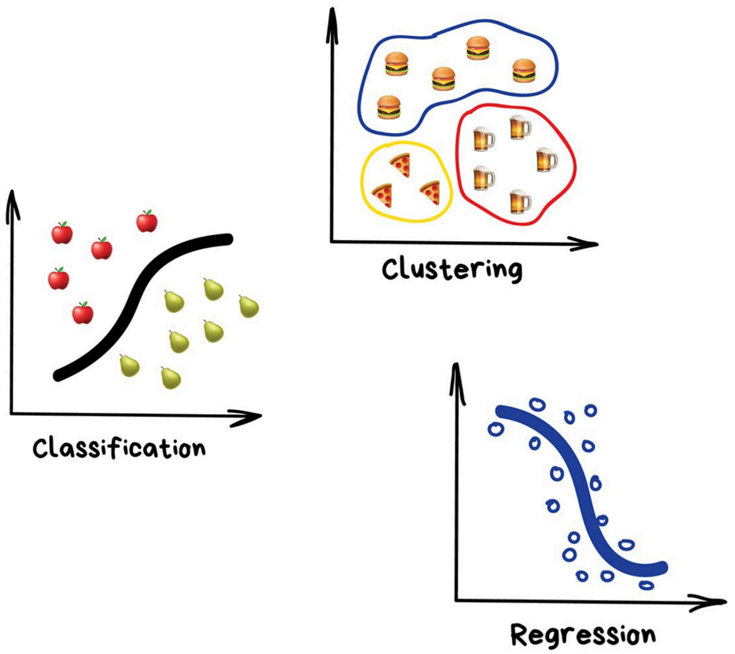


Neural networks

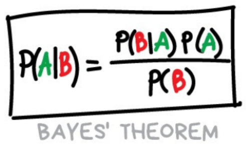


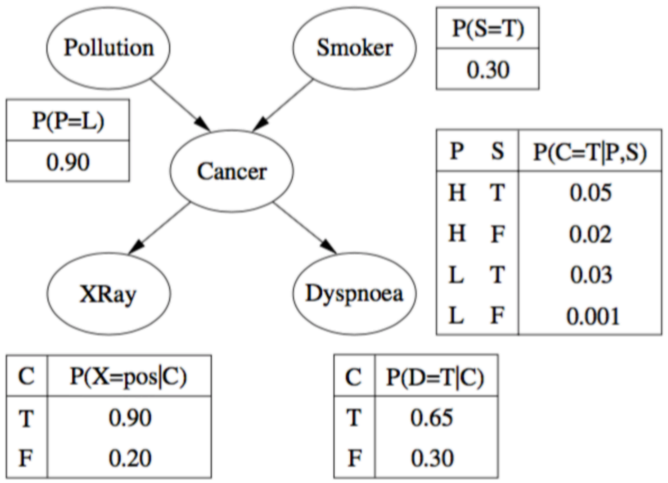


Machine Learning Software

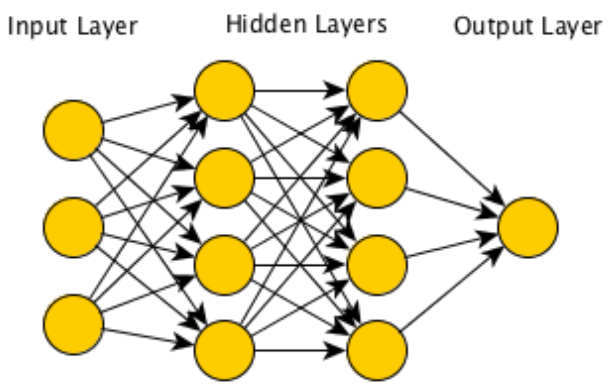


Probabilistic models

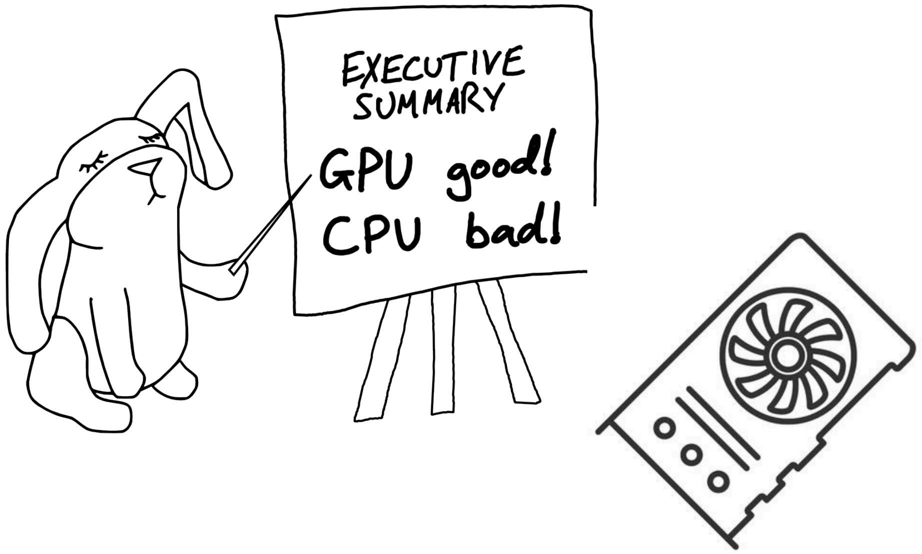




Neural networks



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

```python
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 + **prior model as a program (sample generator)** + **query** → **blackbox methods** → **answer**

(e.g., Variational Inference)

Related software

TensorFlow Probability
Google

Edward

PyTorch
facebook research

Pyro
UBER AI Labs

Stan

PyMC3

theano

InferPy

Related software

TensorFlow Probability
Google

Edward

PyTorch
facebook research

Pyro
UBER AI Labs

Stan

PyMC3

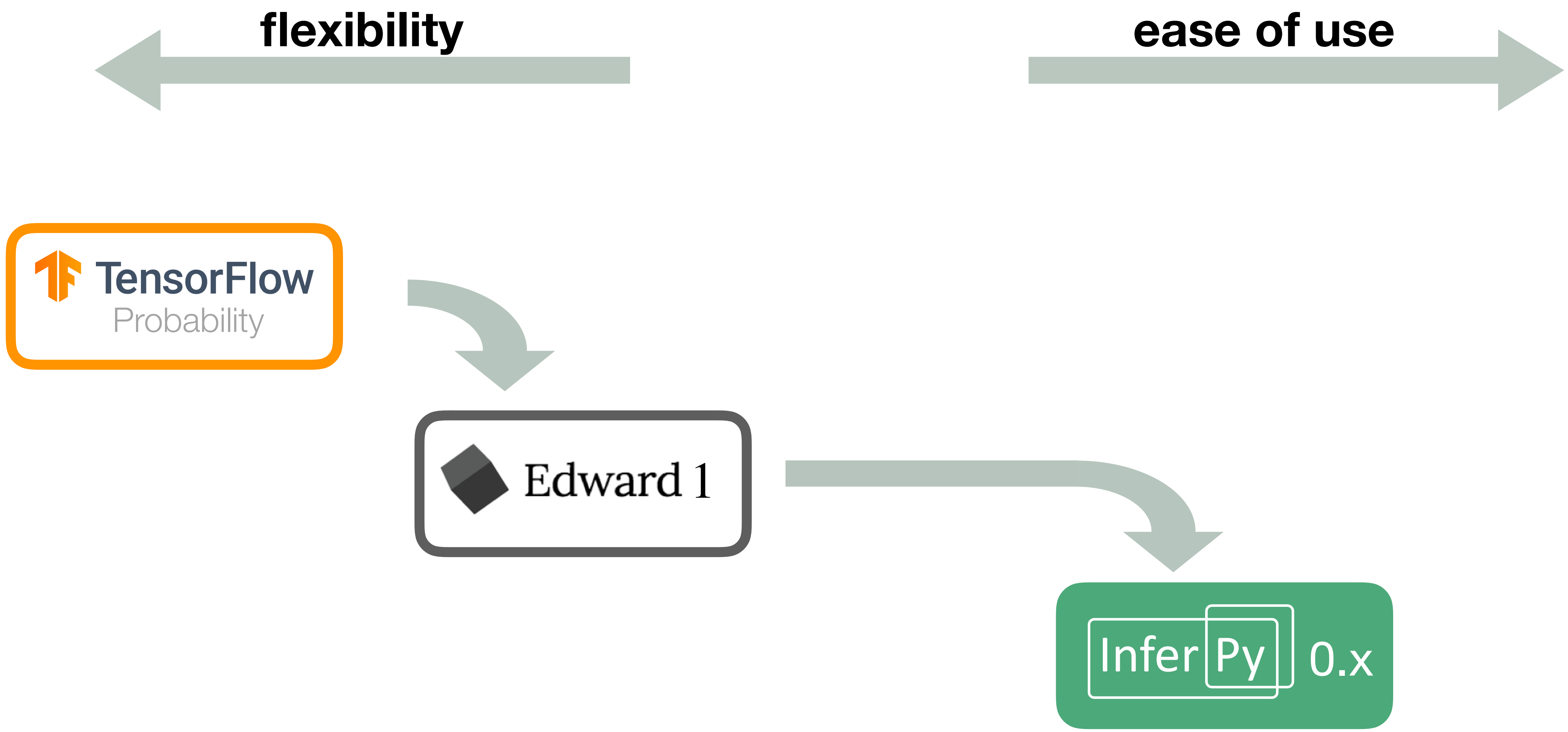
theano

Advantages of InferPy:

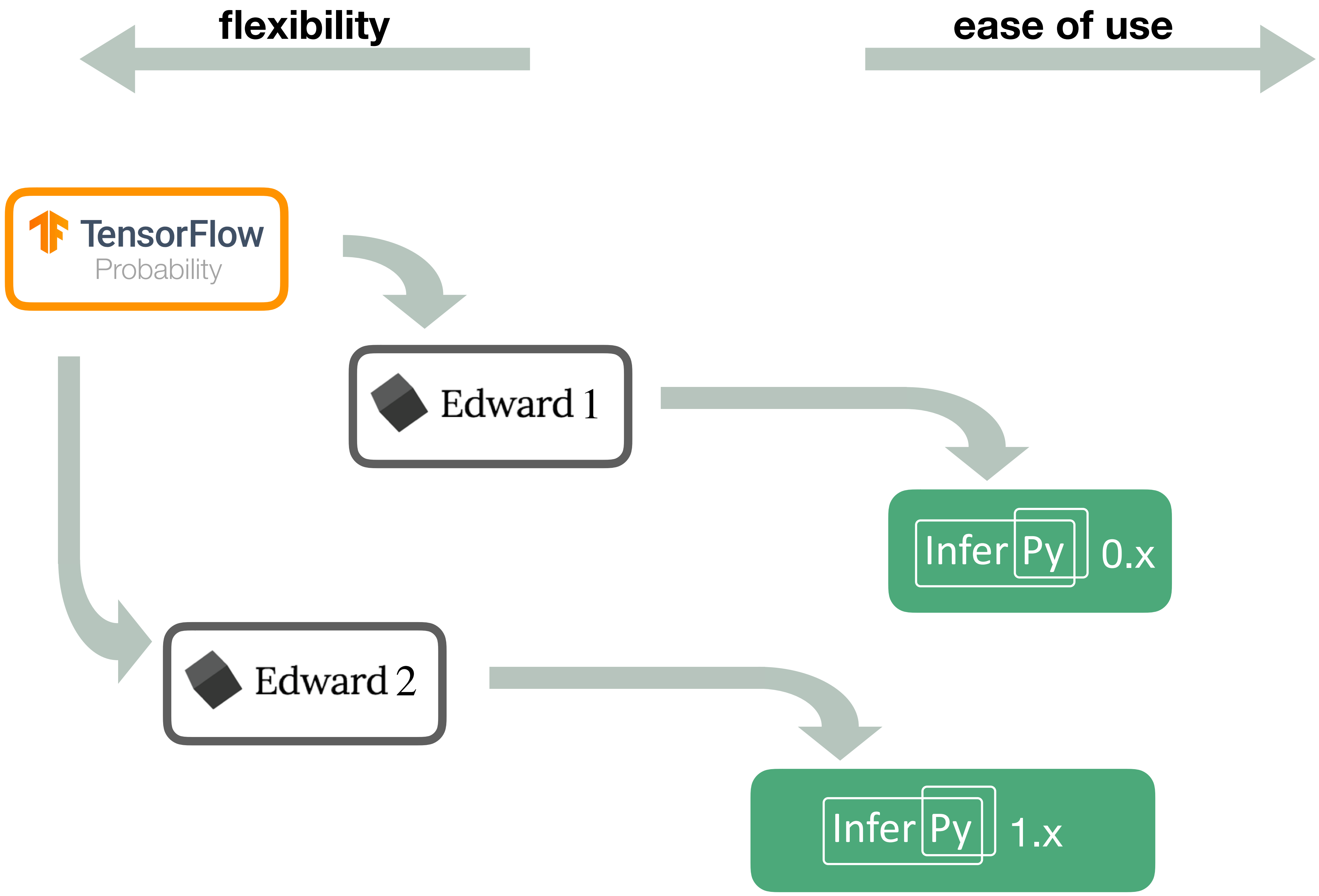
- The definition of distributions over tensors is simplified
- No need to have a strong background in inference algorithms

$$1 \quad \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 1 & 7 \end{bmatrix} & \begin{bmatrix} 3 & 2 \\ 5 & 4 \end{bmatrix} \end{bmatrix}$$

(tensors)

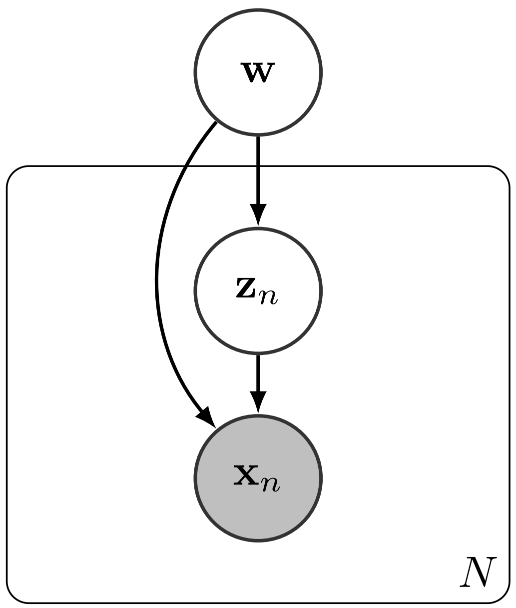

flexibility
ease of use
TensorFlow
Probability
Edward 1
InferPy 0.x

Tensorflow Probability and Edward



InferPy

Hierarchical Probabilistic Models



$\mathbf{w}$: global parameters

$\mathbf{z}$: local hidden variables

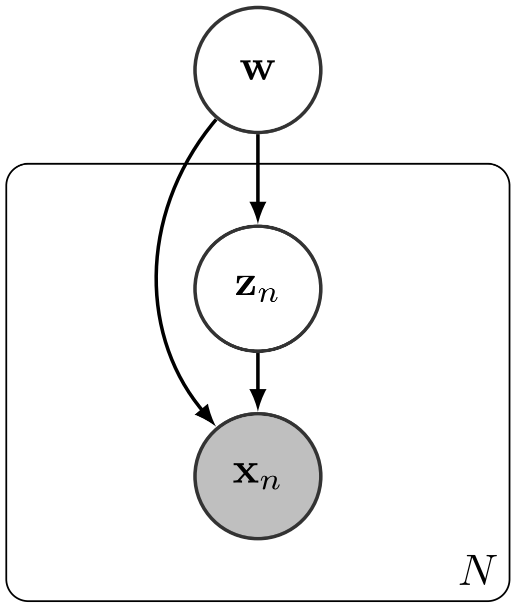
$\mathbf{x}$: observations

N : number of observations

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

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

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

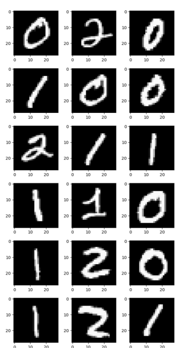
TensorFlow Keras

InferPy

Classification of hand-written digits

images

x_n

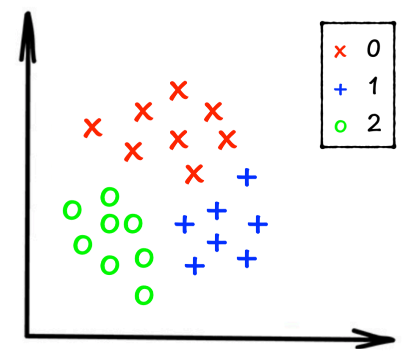


d = 28 * 28 = 784

encoding (NN1)

hidden representation

z_n



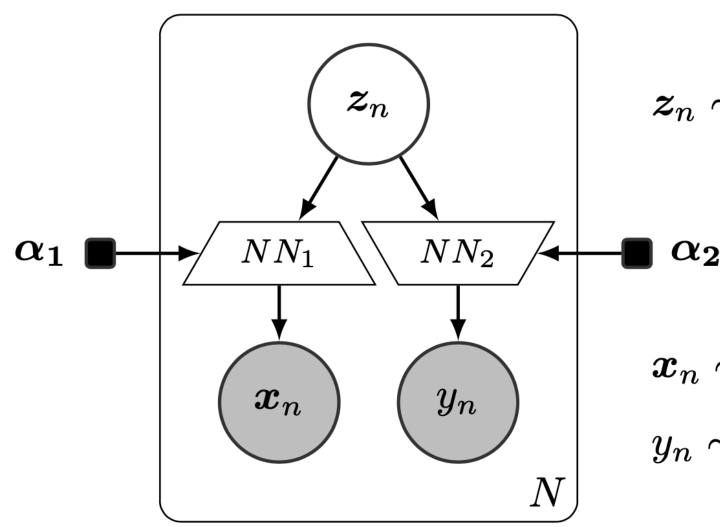
k = 2

classification (NN2)

class

y_n

0 2 0 1 0 0 … 1 2 1



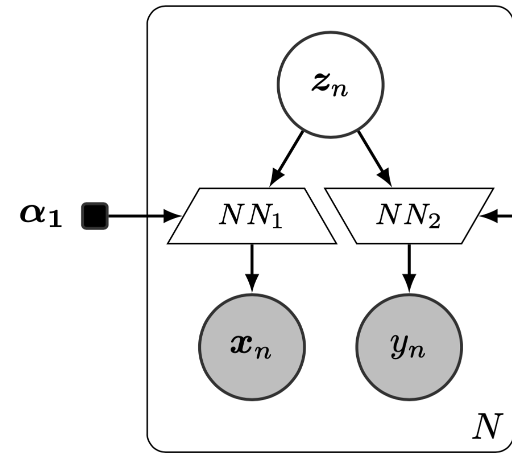
$z_n \sim \mathcal{N}_k(0, I)$

$x_n \sim \mathcal{N}_{dx}(NN_1(z_n), I)$

$y_n \sim Cat(logits = NN_2(z_n), I)$

Model definition

```python
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)
```



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

$x_n \sim \mathcal{N}_{dx}(NN_1(z_n), I)$

$y_n \sim Cat(logits = NN_2(z_n), I)$

```
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))])
```

Model definition

- Random variables in InferPy encapsulate another from Edward 2

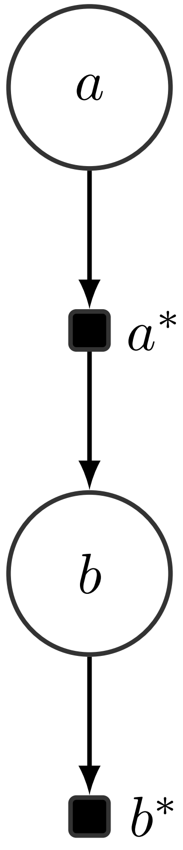
- These can follow the following distributions

```
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

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

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

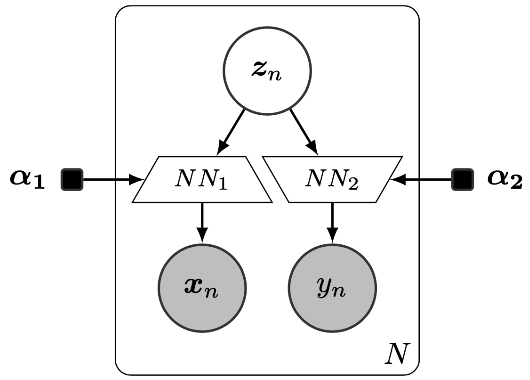
```
In[*]:
   ...: sess = inf.get_session()
   ...: for i in range(5):
   ...:     print(sess.run([a,b]))

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



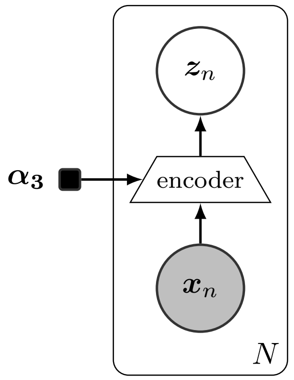

```python
# a continuous variable might be parent of a discrete one

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]}))
```

Stochastic Variational Inference (SVI)

P model



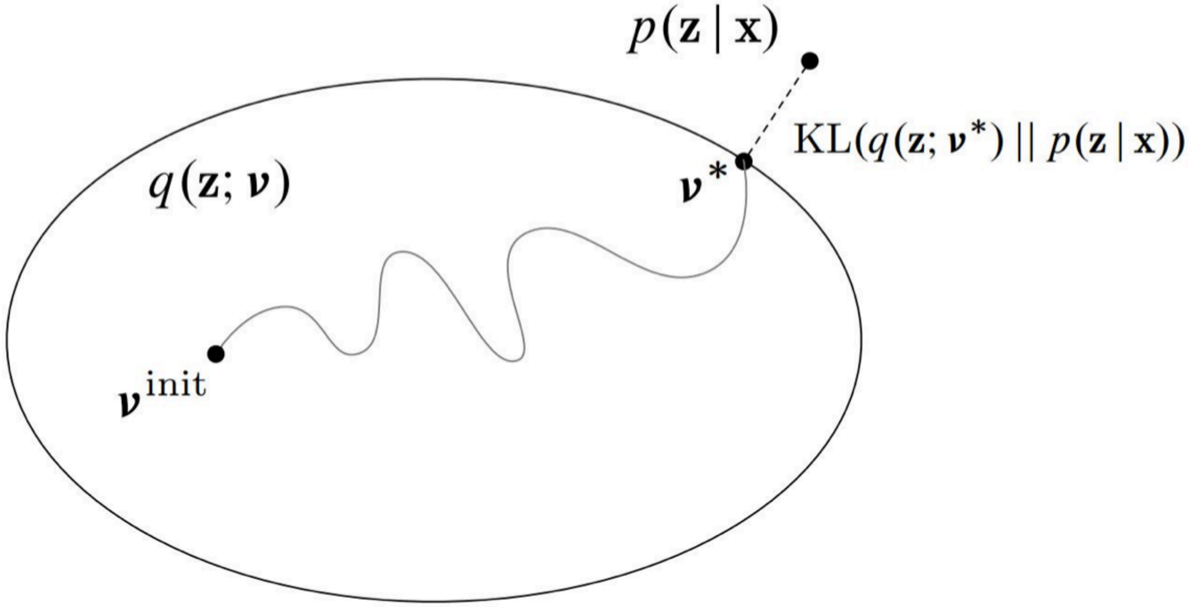
Q model



data

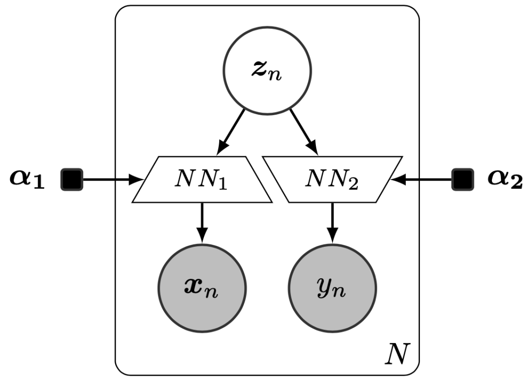
	Sepal.Length	Sepal.Width	Petal.Length
1	5.1	3.5	1.4
2	4.9	3.0	1.4
3	4.7	3.2	1.3
4	4.6	3.1	1.5
5	5.0	3.6	1.4
6	5.4	3.9	1.7
7	4.6	3.4	1.4
8	5.0	3.4	1.5
9	4.4	2.9	1.4
10	4.9	3.1	1.5
11	5.4	3.7	1.5
12	4.8	3.4	1.6
13	4.8	3.0	1.4
14	4.3	3.0	1.1
15	5.8	4.0	1.2
16	5.7	4.4	1.5
17	5.4	3.9	1.3
18	5.1	3.5	1.4
19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

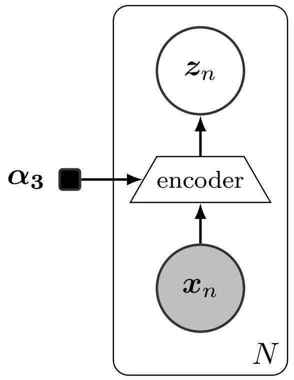


Variational Inference (VI)

P model

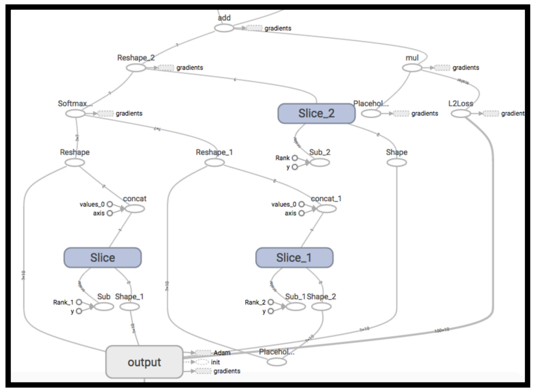


Q model



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



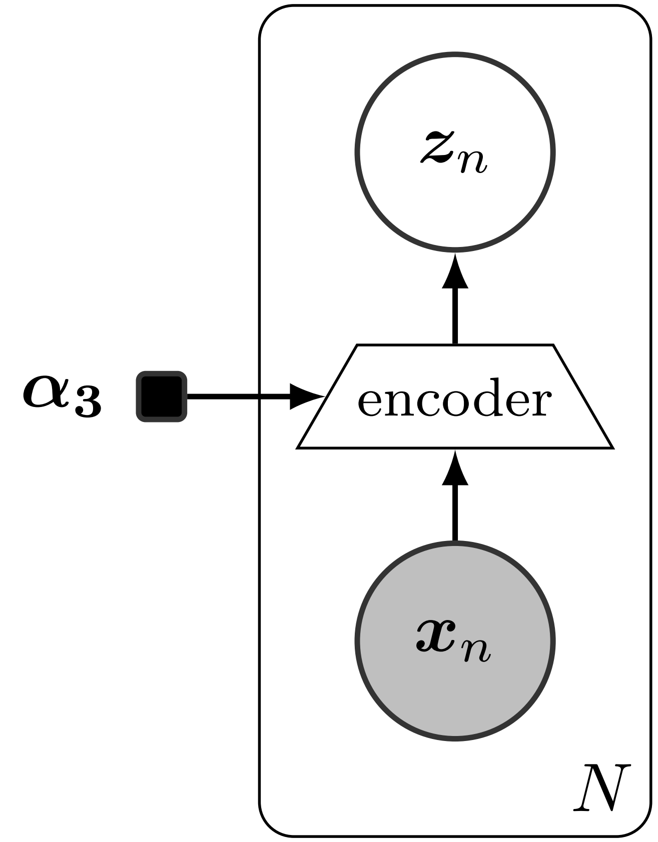
Computational graph of the $ELBO$ **(to maximise)**

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

Inference (of the parameters)

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

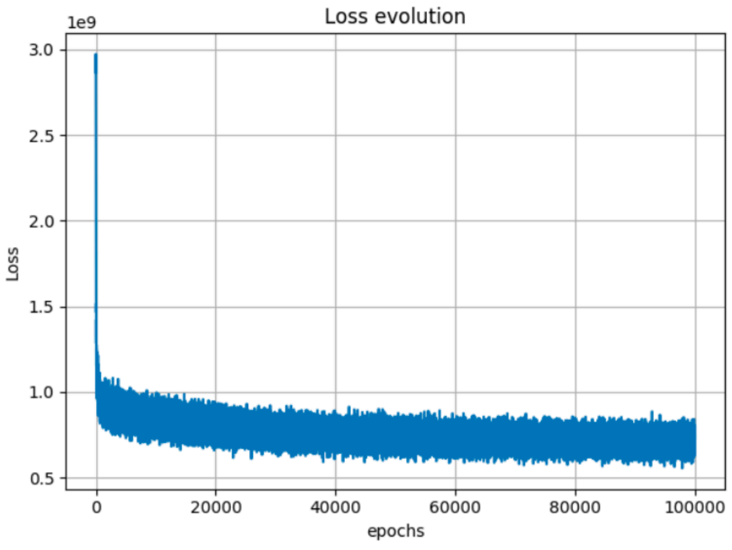
```
@inf.probmodel
def qmodel(k, d0, dx):
    with inf.datamodel():
        x = inf.Normal(tf.ones(dx), 1, name="x")

        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)
```

- We can extract and plot the loss function evolution

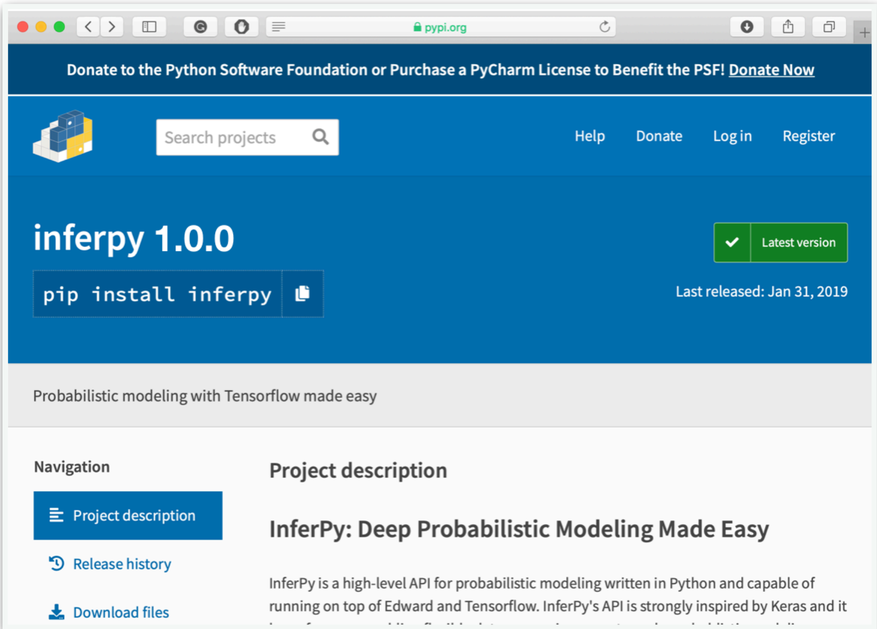
```
# extract the loss evolution
L = SVI.losses
```



- 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}")
```

Documentation

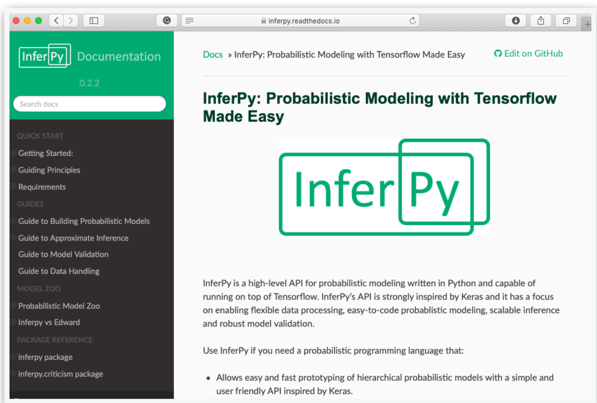


https://pypi.org/project/inferpy/

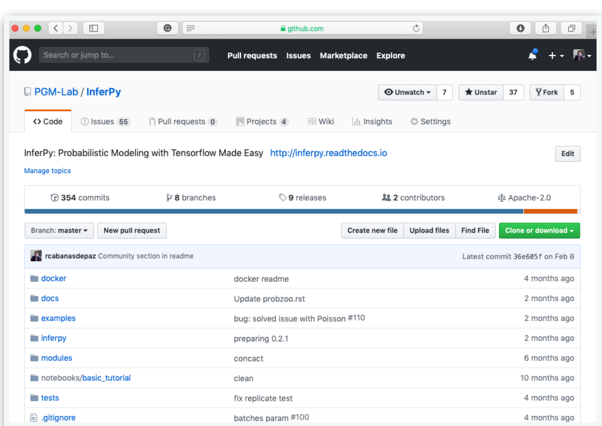
```
$ pip install inferpy
```

Documentation

- InferPy in distributed under license Apache 2.0



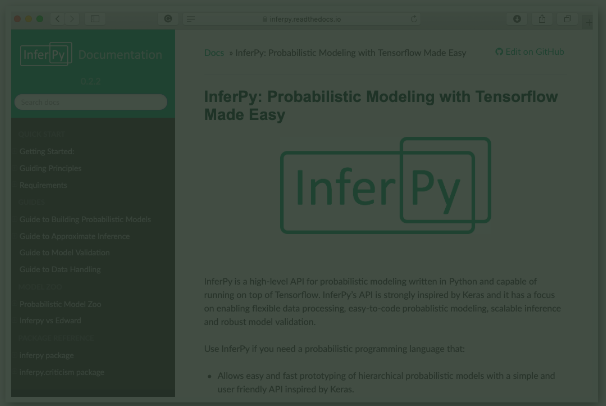
https://inferpy.readthedocs.io



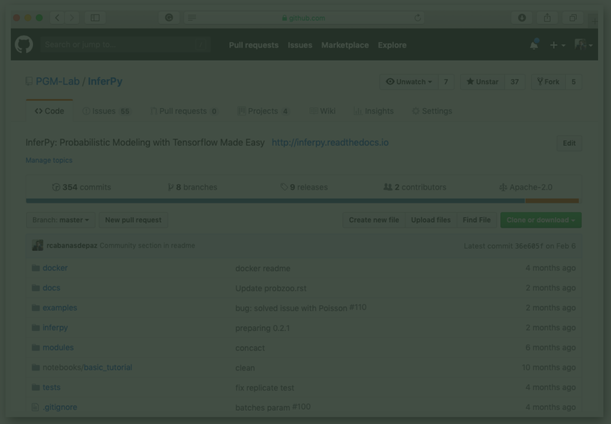
https://github.com/PGM-Lab/InferPy

Documentation

- InferPy in distributed under license Apache 2.0



https://inferpy.readthedocs.io



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

Thank you for your attention !