MUTUAL INFORMATION NEURAL ESTIMATION

—

22/01/2020

Safran Tech / TSI
Signal and Data Analysis Lab

Mutual information

Principle

- Estimation of the relationship between two variables (X, Y) by estimating their *dependency*
- *Amount of information* (bits) the observation of a variable X gives about another variable Y

Usage

- Blind source separation
- Feature selection
- Information bottleneck problem
- Causality detection

Formulation

- $I(X,Y) = \int_{\mathcal{X}\times\mathcal{Y}} \log\left(\frac{d\mathbb{P}_{XY}}{d\mathbb{P}_X \otimes d\mathbb{P}_Y}\right) d\mathbb{P}_{XY}$ where $\mathbb{P}_{XY}$ is the joint probability distribution of (X,Y), $\mathbb{P}_X$ and $\mathbb{P}_Y$ the marginals

Mutual information

Reformulations and limitations

- $I(X,Y) = \int_{\mathcal{X}\times\mathcal{Y}} \log\left(\frac{d\mathbb{P}_{XY}}{d\mathbb{P}_X \otimes d\mathbb{P}_Y}\right) d\mathbb{P}_{XY} = KL(\mathbb{P}_{XY}||\mathbb{P}_X \otimes \mathbb{P}_Y) = \int_{\mathcal{X}\times\mathcal{Y}} \log\left(\frac{d\mathbb{P}_{Y|X}}{d\mathbb{P}_Y}\right) d\mathbb{P}_{XY}$
- Estimating MI requires **knowledge** or **estimation** of joint distributions, marginals, conditionals and/or density ratios
- Estimating such quantities in high dimensional or large sample size settings is generally **not tractable**
- **Approximate inference**, and especially **variational inference** enables to estimate such quantities with tractable approximate distributions

Untracable integral? Go variational !

For $I(X,Y) = \int_{\mathcal{X}\times y} \log\left(\frac{d\mathbb{P}_{Y|X}}{d\mathbb{P}_Y}\right) d\mathbb{P}_{XY}$ we introduce a variational family of posterior distributions $\mathbb{Q}_{Y|X}$

$$
\begin{aligned}
& I(X,Y) \\
&= \int_{\mathcal{X}\times y} \log\left(\frac{d\mathbb{P}_{Y|X}}{d\mathbb{P}_Y}\right) d\mathbb{P}_{XY} \\
&= \int_{\mathcal{X}\times y} \log\left(\frac{d\mathbb{P}_{Y|X} d\mathbb{Q}_{Y|X}}{d\mathbb{Q}_{Y|X} d\mathbb{P}_Y}\right) d\mathbb{P}_{XY} \\
&= \int_{\mathcal{X}\times y} \log\left(\frac{d\mathbb{Q}_{Y|X}}{d\mathbb{P}_Y}\right) d\mathbb{P}_{XY} + \int_{\mathcal{X}\times y} \log\left(\frac{d\mathbb{P}_{Y|X}}{d\mathbb{Q}_{Y|X}}\right) d\mathbb{P}_{XY} \\
&= \int_{\mathcal{X}\times y} \log\left(\frac{d\mathbb{Q}_{Y|X}}{d\mathbb{P}_Y}\right) d\mathbb{P}_{XY} + \int_{\mathcal{X}\times y} \log\left(\frac{d\mathbb{P}_{Y|X}}{d\mathbb{Q}_{Y|X}}\right) d\mathbb{P}_{Y|X} d\mathbb{P}_X \\
&= \int_{\mathcal{X}\times y} \log\left(\frac{d\mathbb{Q}_{Y|X}}{d\mathbb{P}_Y}\right) d\mathbb{P}_{XY} + \mathbb{E}_{\mathbb{P}_X} KL\left(\mathbb{P}_{Y|X} \,\|\, \mathbb{Q}_{Y|X}\right) \\
&\geq \int_{\mathcal{X}\times y} \log\left(\frac{d\mathbb{Q}_{Y|X}}{d\mathbb{P}_Y}\right) d\mathbb{P}_{XY}
\end{aligned}
$$

Variational inference objective

$$I(X,Y) \geq \max_{\mathbb{Q}_{Y|X}} \int_{\mathcal{X}\times\mathcal{Y}} \log\left(\frac{d\mathbb{Q}_{Y|X}}{d\mathbb{P}_Y}\right) d\mathbb{P}_{XY}$$

The idea is to **discribe a family of tractable** $\mathbb{Q}_{Y|X}$ and maximize on it.

Energy-based variational family consists in $d\mathbb{Q}_{Y|X} = \frac{d\mathbb{P}_Y}{Z(X)} e^{T(X,Y)}$ where T is a deterministic function and $Z(X) = \mathbb{E}_{\mathbb{P}_Y}\left[e^{T(X,Y)}\right]$

$$I(X,Y) \geq \max_T \int_{\mathcal{X}\times\mathcal{Y}} \log\left(\frac{d\mathbb{P}_Y e^{T(X,Y)}}{Z(X) d\mathbb{P}_Y}\right) d\mathbb{P}_{XY} \geq \max_T \mathbb{E}_{\mathbb{P}_{XY}} T(X,Y) \; - \int_{\mathcal{X}\times\mathcal{Y}} \log\big(Z(X)\big)\, d\mathbb{P}_{XY}$$

and

$$\int_{\mathcal{X}\times\mathcal{Y}} \log\big(Z(X)\big)\, d\mathbb{P}_{XY} = \int_{\mathcal{X}} \log\big(Z(X)\big)\, d\mathbb{P}_X \leq \log \int_{\mathcal{X}} Z(X) d\mathbb{P}_X = \log \mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y}\, e^{T(X,Y)}$$

We have a consistent variational lower bound

$$I(X,Y) \geq \max_T \mathbb{E}_{\mathbb{P}_{XY}} T(X,Y) \; - \log \mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} e^{T(X,Y)} := \max_T \hat{I}(T)$$

We remark that there exists a **tigh bound** for $T^* = \log \frac{d\mathbb{P}_{XY}}{d(\mathbb{P}_X \otimes \mathbb{P}_Y)}$

Hence, using the trick $\log x \leq x - 1$ we obtain

$$\begin{aligned}
& I(X,Y) - \hat{I}(T) \\
&= \mathbb{E}_{\mathbb{P}_{XY}} T^*(X,Y) \; - \log \mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} e^{T^*(X,Y)} - \mathbb{E}_{\mathbb{P}_{XY}} T(X,Y) + \log \mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} e^{T(X,Y)} \\
&= \mathbb{E}_{\mathbb{P}_{XY}} [T^*(X,Y) \; - T(X,Y)] - \log \mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} \frac{d\mathbb{P}_{XY}}{d(\mathbb{P}_X \otimes \mathbb{P}_Y)} + \log \mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} e^{T(X,Y)} \\
&= \mathbb{E}_{\mathbb{P}_{XY}} [T^*(X,Y) \; - T(X,Y)] + \log \mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} e^{T(X,Y)} \\
&\leq \mathbb{E}_{\mathbb{P}_{XY}} [T^*(X,Y) \; - T(X,Y)] + \mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} \left[e^{T(X,Y)} - e^{T^*(X,Y)} \right]
\end{aligned}$$

Using *universal approximation theorem*, we have that $\forall \eta > 0$ we have a bound neural network $T_\theta \leq M$ such that

$$\left| I(X,Y) - \hat{I}(T_\theta) \right| \leq \mathbb{E}_{\mathbb{P}_{XY}} |T^*(X,Y) \; - T_\theta(X,Y)| + \mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} \left| e^{T_\theta(X,Y)} - e^{T^*(X,Y)} \right| \leq \eta$$

We have a strongly consistent variational lower bound

It can be shown using *uniform law of large number* that the estimator in limited data is strongly consistent, i.e. $\forall \eta > 0, \exists N \in \mathbb{N}$ such that $\forall n \geq N$

$$|I_n(X,Y) - I(X,Y)| \leq \eta$$

where $I_n(X,Y) = \max_T \mathbb{E}_{\mathbb{P}_{XY}^{(n)}} T(X,Y) - \log \mathbb{E}_{\mathbb{P}_X^{(n)} \otimes \mathbb{P}_Y^{(n)}} e^{T(X,Y)}$ with $\mathbb{P}_{XY}^{(n)}$ the empirical distribution over n data point

Some code

```python
class MINE(nn.Module):
    def __init__(self, in_dim=2, h_dims=[64]):
        super().__init__()
        self.T = nn.ModuleList([nn.Linear(in_dim, h_dims[0]), nn.ReLU()])
        for i, h in enumerate(h_dims[1:]):
            self.T.append(nn.Linear(h_dims[i-1], h))
            self.T.append(nn.ReLU())
        self.T.append(nn.Linear(h_dims[-1], 1))

    def forward(self, X):
        output = copy.copy(X)
        for n in self.T:
            output = n(output)
        return output

    def init_weights(self):
        for n in self.T:
            if type(n) == nn.Linear or type(n) == nn.Conv1d or type(n) == nn.Conv2d:
                torch.nn.init.xavier_uniform_(n.weight)
                n.bias.data.fill_(0.0)

    def mutual_information(self, XY_joint, XY_marginal):
        T = self.forward(XY_joint)
        exp_T = torch.exp(self.forward(XY_marginal))
        mi = T.mean() - torch.log(exp_T.mean())
        return mi, T, exp_T

    def debiased_learning(self, optimizer, XY_joint, XY_marginal, ma_alpha, ma_previous):
        optimizer.zero_grad()
        _, T, exp_T = mine.mutual_information(XY_joint, XY_marginal)
        ma_exp_T = ma_alpha * exp_T.mean() + (1 - ma_alpha) * ma_previous
        loss = - T.mean() + exp_T.mean().mul(1/ma_exp_T.detach())
        loss.backward()
        optimizer.step()

        return ma_exp_T
```

Why debiased learning?

$$\nabla_\theta \hat{I}(T_\theta) = \mathbb{E}_{\mathbb{P}_{XY}} \nabla_\theta T_\theta(X,Y) \; - \frac{\mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} T_\theta(X,Y) e^{T_\theta(X,Y)}}{\mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} e^{T_\theta(X,Y)}}$$

Hence, for one batch, the estimate $\frac{1}{n}\sum_{i=1}^{n} e^{T_\theta(X_i,Y_i)}$ of $\mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Y} e^{T_\theta(X,Y)}$ is batch-biased. They propose to accumulate information about the estimate over several batches with estimate memory and exponential moving average.

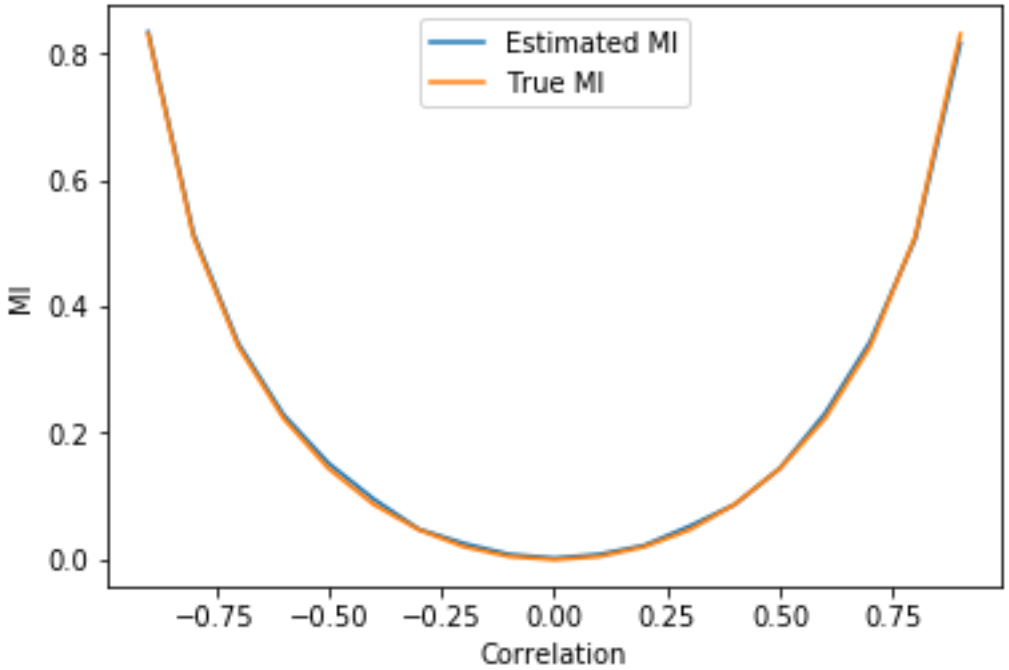
Some experiment

We generate different samples of bivariate gaussian variables, with different correlations.

We estimate MI for each correlation and observe that the estimations are almost perfect.

```
correlations = np.arange(-0.9, 1, 0.1)
ma_alpha = 0.01

all_estimated, all_mis = [], []

for correlation in correlations:
    ma_previous = 1.

    mine = MINE().to(device)
    mine.init_weights()
    optimizer = optim.Adam(mine.parameters())

    xy_joint = torch.FloatTensor(np.random.multivariate_normal(mean=[0,0],
                                                               cov=[[1,correlation],[correlation,1]],
                                                               size = 10000)).to(device)

    xy_marginal = copy.deepcopy(xy_joint)

    mis = []

    for epoch in range(5000):
        optimizer.zero_grad()

        index = np.arange(xy_joint.shape[0])
        np.random.shuffle(index)
        xy_joint = xy_joint[index]

        index = np.arange(xy_marginal.shape[0])
        np.random.shuffle(index)
        xy_marginal[:, 0] = xy_marginal[index, 0]

        np.random.shuffle(index)
        xy_marginal[:, 1] = xy_marginal[index, 1]

        ma_previous = mine.debiased_learning(optimizer, xy_joint, xy_marginal, ma_alpha, ma_previous)

        mi, T, exp_T = mine.mutual_information(xy_joint, xy_marginal)

        if epoch > 3000:
            mis.append(mi.item())

    all_estimated.append(np.mean(mis))
    all_mis.append(-0.5*np.log(1-correlation**2).item())

    print(np.round(correlation, 1), -0.5*np.log(1-correlation**2).item(), np.mean(mis))
```

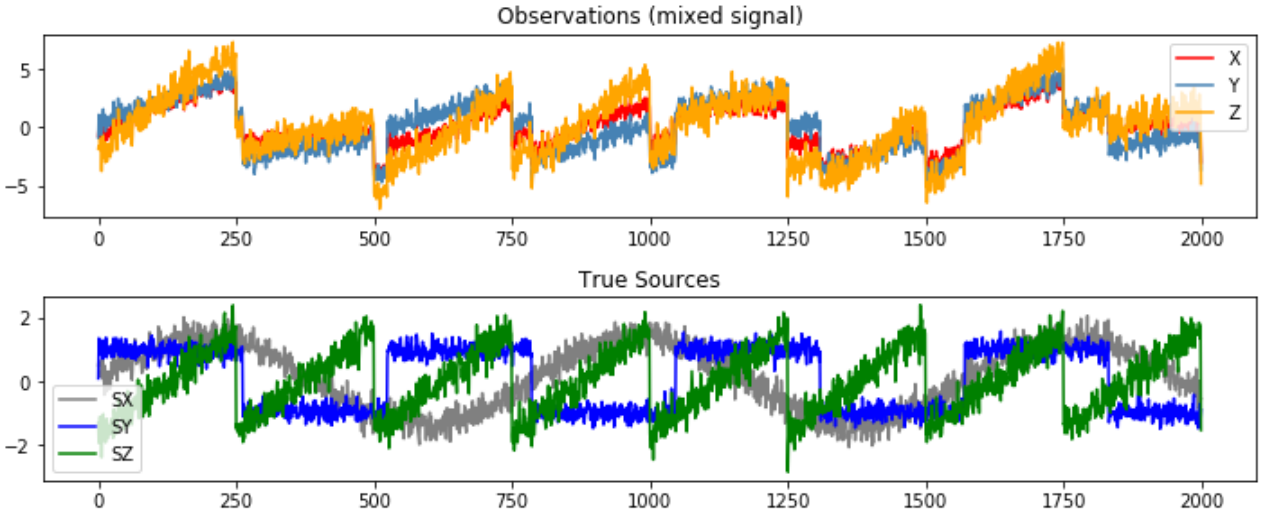
Some experiment



ICA

An observed 3D-signal (X, Y, Z) is the linear combination of a 3D source signal (S^X, S^Y, S^Z): $[X, Y, Z] = A[S^X, S^Y, S^Z]$

The objective is to find back the sources. We propose to use the MINE to learn the matrix $A^{-1} = W$ that transforms $[X, Y, Z]$ in independent sources, i.e. the W that minimizes the mutual information between variables $W[X, Y, Z]$.



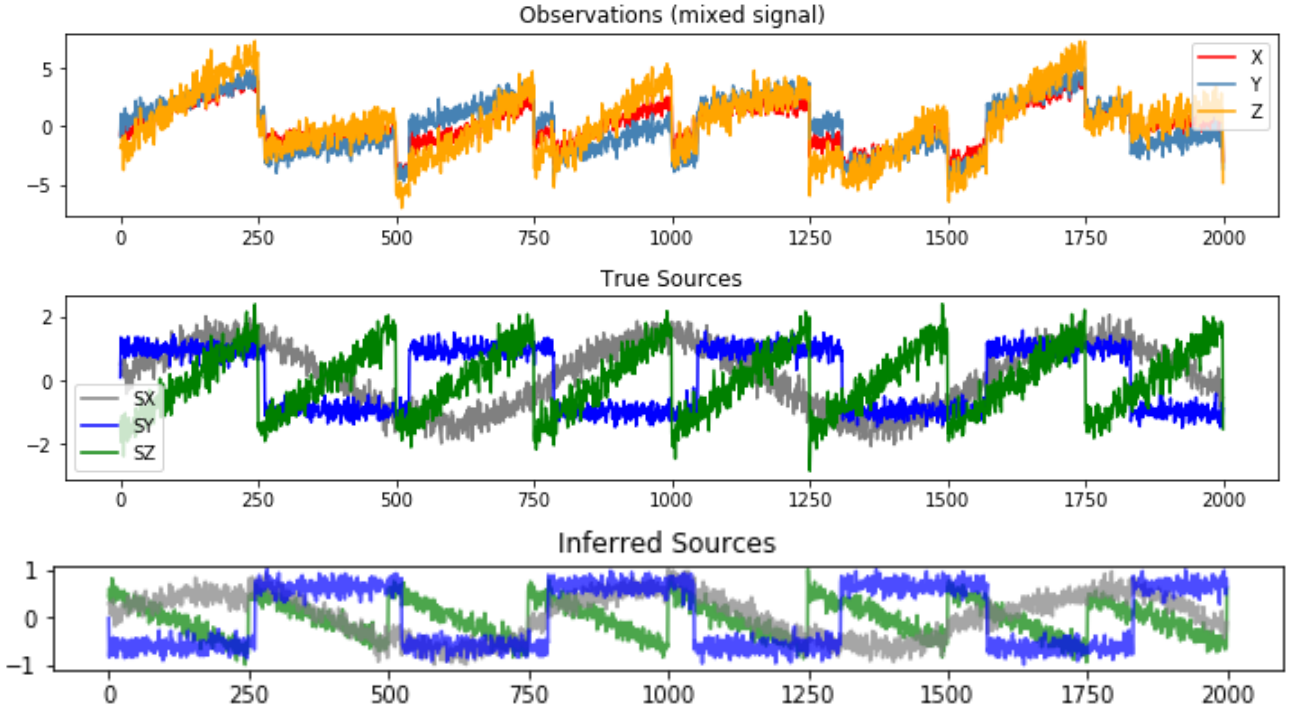
ICA

In [23]:
```
mine = MINE(3, h_dims=[64, 64, 64]).to(device)
mine.init_weights()
optimizer_mine = optim.Adam(mine.parameters())

ica = nn.Linear(3, 3).to(device)
optimizer_ica = optim.Adam(ica.parameters())
```

In [24]:
```
ma_previous = 1.
ma_alpha = 0.01

for epoch in range(10000):

    xy_joint = ica(X)

    index = np.arange(xy_joint.shape[0])
    np.random.shuffle(index)
    xy_joint = xy_joint[index]

    indices = np.stack([np.arange(xy_joint.shape[0]) for _ in range(xy_joint.shape[1])], 1)
    for i in range(xy_joint.shape[1]): np.random.shuffle(indices[:, i])

    # MI estimation

    xy_marginal = xy_joint[indices, range(3)]

    ma_previous = mine.debiased_learning(optimizer_mine, xy_joint, xy_marginal, ma_alpha, ma_previous)

    # MI minimization

    if epoch % 10 == 0:
        optimizer_ica.zero_grad()
        xy_joint = ica(X)

        index = np.arange(xy_joint.shape[0])
        np.random.shuffle(index)
        xy_joint = xy_joint[index]

        indices = np.stack([np.arange(xy_joint.shape[0]) for _ in range(xy_joint.shape[1])], 1)
        for i in range(xy_joint.shape[1]): np.random.shuffle(indices[:, i])

        # MI estimation

        xy_marginal = xy_joint[indices, range(3)]
        mi, _, _ = mine.mutual_information(xy_joint, xy_marginal)
        mi.backward()
        optimizer_ica.step()

    if epoch % 20 == 0:
        print(epoch, mi.item())
```

Result

Perfect inference of independent components



THANK YOU

QUESTIONS ?