TP: Implementation of DAN (compte-rendu)

The function *experiment* in manage_exp.py reads the configuration files lin2d_exp.py and lorenz_exp.py and then execute the training and evaluation of DAN. The challenge of this set of TP is to implement the training of DAN in two modes : full and online.

Notice : You should upload a **report**, together with your **code** of this set of TP (5 seances) to Moodle, for the validation of the course. Do not copy-paste another report or code.

Due : 8 Avril 2024.

1 Pre-training of c for Linear 2d

We start from the implementation and the training of **c** to approximate the initial distribution $p(x_0)$. The main function to work on is

```
def pre_train_full(net,b_size,h_dim,x_dim,
                   sigma0,optimizer_classname,optimizer_kwargs)
```

TODO 1.1 Read the function experiment in manage_exp.py to understand how the global logic of the code is about. Understand the code ConstructorProp, ConstructorObs, ConstructorA, etc in filters.py. Run your code with

```
python main.py --save lin2d_exp.py --run
```

TODO 1.2 Implementation of the Gaussian module in ConstructorC. Read the code in the module **c** which is a concatenation of a FullyConnected module and Gaussian module. For the Fully connection module, we have

```
module c FullyConnected(
  (lins): ModuleList(
    (0): Linear(in_features=6, out_features=4, bias=True)
  )
  (acts): ModuleList()
)
```

The constructor of $\mathbf{c}$ will return a Gaussian module.

• For the Gaussian module, it will convert the vector lc into (loc,scale_tril) which represents μ and Λ. It then computes the log probability of x.

• To protect from numerical instabilities of the computation of the exponentiation and the loss function, we apply a min-max threshold (a, b) to each diagonal terms in the tri-diagonal matrix scale_tril, which means to compute $\widetilde{\Lambda}$, Recall

$$\Lambda = \begin{pmatrix} e^{v_n} & 0 & \cdots & 0 \\ v_{2n} & e^{v_{n+1}} & \cdots & 0 \\ \cdots & \cdots & \cdots & 0 \\ v_{n+\frac{n(n+1)}{2}-1} & \cdots & v_{3n-2} & e^{v_{2n-1}} \end{pmatrix}$$

Then

$$\widetilde{\Lambda} = \begin{pmatrix} e^{\widetilde{v}_n} & 0 & \cdots & 0 \\ v_{2n} & e^{\widetilde{v}_{n+1}} & \cdots & 0 \\ \cdots & \cdots & \cdots & 0 \\ v_{n+\frac{n(n+1)}{2}-1} & \cdots & v_{3n-2} & e^{\widetilde{v}_{2n-1}} \end{pmatrix}$$

where $\widetilde{v}_n = \max(a, \min(v_n, b))$, etc. Set $a = -8$ and $b = 8$ to proceed.

TODO 1.3 Understand the code to generate x_0. x_0 is stored in a matrix of size $mb \times n$. Note $n = 2$ for the linear 2d case. Take $mb = 128$. Implement the module Lin2d in filters using your previous TP code.

TODO 1.4 Implement the function closure0. Check what you have obtained after the optimization. Report the following results,

```
print('## INIT a0 mean', pdf_a0.mean[0,:])
print('## INIT a0 var', pdf_a0.variance[0,:])
print('## INIT a0 covar', pdf_a0.covariance_matrix[0,:,:])
```

TODO 1.5 The Gaussian module code is not very fast, we shall use batch operations in pytorch to improve its speed. The functions torch.bmm and torch.triangular_solve can help you to achieve this. Report how you address this problem.

You may also use the MultivariateNormal module for this purpose. Understand how it works (read doc in pytorch `https://pytorch.org/docs/stable/distributions.html`).

2 Full-training of $\mathbf{a}, \mathbf{b}, \mathbf{c}$ for Linear 2d

Based on the pre-trained $\mathbf{c}$, we are going to jointly train the modules $\mathbf{a}, \mathbf{b}, \mathbf{c}$, using

```
train_full(net, b_size, h_dim, x_dim,
           T, checkpoint, direxp,
           prop, obs, sigma0,
           optimizer_classname, optimizer_kwargs)
```

TODO 2.1 Read the code of the module $\mathbf{a}$ and $\mathbf{b}$, implemented in FcZeroLin and FcZero. Correct an error in the initialization of FcZero.

TODO 2.2 Implement the forward steps of $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ in the module DAN (filters.py).

TODO 2.3 In order to compute the loss, we still need generate training samples in the function train_full. Re-use your previous TP code to generate training data sequences for $t = 0..T$.

TODO 2.4 Now you can compute and then minimize the training loss from mb samples over $t = 0..T$. It is given by

$$\frac{1}{T}\sum_{t\leq T}(\mathcal{L}_t(q_t^{\mathbf{b}}) + \mathcal{L}_t(q_t^{\mathbf{a}})) + \mathcal{L}_0(q_0^{\mathbf{a}}).$$

Complete the code in the function train_full.

• Test different values of the "deep" parameter in the file lin2d_exp. In your report, give the initial and final scores in a table, which are returned by

```
print_scores(net.scores)
```

****IMPORTANT**** You should also attach the file 'scores.pt', 'scores.txt', 'test_scores.pt' and 'test_scores.txt' to your code. They store your obtained results.

• Make a plot of x_t, y_t and $\mu_t^{\mathbf{a}}$ for one sample in your training data, for $t \leq T$. One example is given in Figure 1.

• Make a plot of x_t, y_t and $\mu_t^{\mathbf{a}}$ for one test sample for $t \leq 2T$. Use the function get_x0_test to initialize x_0.

3 (Bonus) Online-training of $\mathbf{a}, \mathbf{b}, \mathbf{c}$ for Lorentz 40d

Due to the chaotic behavior in a Lorentz system, we are going to train DAN with a large T. This can be achieved by an online training approach.

• Based on what you have already implemented, use the optimizer ADAM instead of L-BFGS to minimize the online loss $\mathcal{L}_t(q_t^{\mathbf{b}}) + \mathcal{L}_t(q_t^{\mathbf{a}})$ at each t, using truncated BPTT. Implement the function,

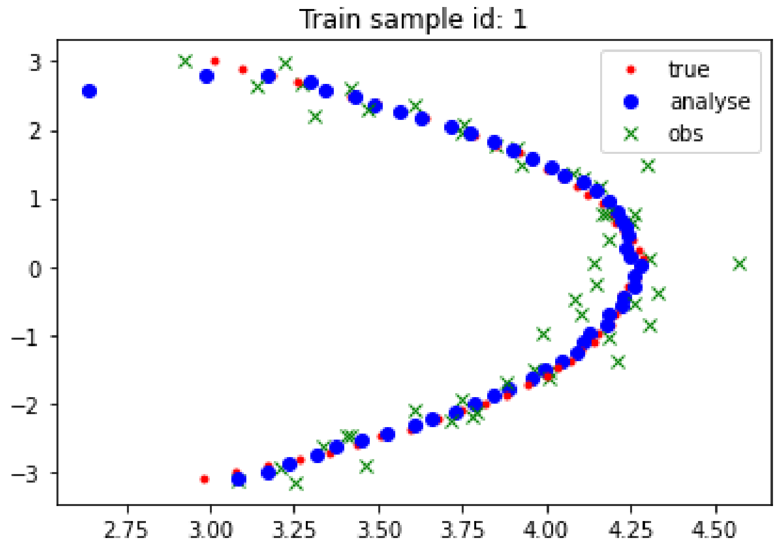

FIGURE 1 – The dynamics of x_t (true) and y_t (obs) in Linear 2d, together with the trajectories of the mean $\mu_t^{\mathbf{a}}$ of the analyse probability density $q_t^{\mathbf{a}}$.

```
train_online(net, b_size, h_dim, x_dim,
             T, checkpoint, direxp,
             prop, obs, sigma0,
             optimizer_classname, optimizer_kwargs,
             scheduler_classname, scheduler_kwargs)
```

- Run the training on GPU (with available memory larger than 1.3G),

```
python main.py --save lorenz_exp.py --run
```

- Make a plot of RMSE on the training and test sequences. You may use the option "plot" in main.py to do this.