ML4Science:
Week 3 Meeting

Calafà, M., Mescolini, G., Motta, P.

École Polytechnique Fédérale de Lausanne (EPFL)

Machine Learning Project 2

7 Dec 2021

Tutor: Dr. Michele Bianco

Outlines

1 Lightening the computational effort
- Motivation
- Reduction of the dataset
- Storage of neighborhoods

2 Splitting between training and validation
- Splitting the validation set
- Preparation of the Datasets for batches

3 Network corrections
- Missing activation function

4 Results

5 Issues and Questions
- Learning Rate Choice
- Bias in *Conv_3d*
- Questions

Lightening the computational effort

Motivation

Problem of Week 2: unsustainable computational time!

We tried to fix this issue by lightening the computational effort:

- Considering a reduced dataset, with 3000 points instead of $300 \times 300 \times 300$.
- Storing once for all the neighborhoods, so that the `main.py` can just read the files, without computing the neighborhoods each time (saving locally $\approx$ 500kB x 2 x 3000 $=$ 3 GB).

Reduction of the dataset

We extracted 3000 random points in our space, with the following commands:

```
# EXTRACTION OF 3000 INDEXES
ind1 = np.random.randint(0,D, S) # D = 300, S = 3000
ind2 = np.random.randint(0,D, S)
ind3 = np.random.randint(0,D, S)
```

Please note that we have set our seed, so that our trials are reproducible.

```
np.random.seed(2021)
```

Storage of neighborhoods

Then, we proceeded with the storage of the values of *n_igm* and *n_src* for the neighborhoods of each of the 3000 points; we also stored the values of the *x_i* for our points in a `.txt` file.
Those files are stored in a folder named `cubes`.

Storage of neighborhoods

```python
# x_i STORAGE
my_xi = torch.flatten(torch.Tensor(xi[ind1,ind2,ind3]))
my_xi = my_xi.numpy()
np.savetxt('cubes/xi_flatten.txt', my_xi)

# n_igm, n_src STORAGE
small_total = np.reshape(np.array([ind1,ind2,ind3]), [S,3])

for count in range(S):
    P = small_total[count,:]
    n_igm_nbh = torch.tensor(get_neighborhood(n_igm, P, r)).float()
    n_src_nbh = torch.tensor(get_neighborhood(n_src, P, r)).float()
    np.save('cubes/n_igm_i%d.npy' % count, n_igm_nbh)
    np.save('cubes/n_src_i%d.npy' % count, n_src_nbh)
```

Splitting between training and validation

Splitting the validation set

We have to extract some data for testing the quality of our neural network; we tried with a 80%-20% splitting between training and validation.
We performed the separation and created PyTorch Datasets by using the following commands:

```

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2,
    random_state=2021)

X_train_src, X_train_igm = torch.Tensor(X_train[:,0,:,:,:]), torch.Tensor(
    X_train[:,1,:,:,:])
X_valid_src, X_valid_igm = torch.Tensor(X_valid[:,0,:,:,:]), torch.Tensor(
    X_valid[:,1,:,:,:])

y_train = torch.Tensor(y_train)
y_valid = torch.Tensor(y_valid)

train_dataset = TensorDataset(X_train_src, X_train_igm, y_train)
valid_dataset = TensorDataset(X_valid_src, X_valid_igm, y_valid)
```

Preparation of the Datasets for batches

Another correction made to last week's code was the division in **batches**, which will be used for different epochs.
In the same code block of the splitting between training and validation, we created data loaders to use for loops, for batches of size 32.

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=32, shuffle=True)
```

Network corrections

Additional activation function

We added the missing activation function between the last hidden layer and the output, in order to obtain a final output in the range $[0, 1]$, since x_i represents the ionization percentage.

```
self.final_activation = nn.Sigmoid()
# ...
out = self.final_activation(out)
```

Our results

Results

```
############# VALIDATION OF OUR MODEL #############
iter  1 / 600       loss =  0.08065502345561981
iter  2 / 600       loss =  0.080454021692276
iter  3 / 600       loss =  0.06447429209947586
iter  4 / 600       loss =  0.08735030889511108
iter  5 / 600       loss =  0.0742887482047081
iter  6 / 600       loss =  0.06913375109434128
iter  7 / 600       loss =  0.08124952018260956
iter  8 / 600       loss =  0.08532914519309998
iter  9 / 600       loss =  0.07979291677474976
iter  10 / 600       loss =  0.09831647574901581
iter  11 / 600       loss =  0.09283353388309479
iter  12 / 600       loss =  0.07419475167989731
iter  13 / 600       loss =  0.0722661167383194
iter  14 / 600       loss =  0.0705186203122139
iter  15 / 600       loss =  0.06650830805301666
iter  16 / 600       loss =  0.08804044127464294
iter  17 / 600       loss =  0.06574168801307678
iter  18 / 600       loss =  0.06208217516541481
iter  19 / 600       loss =  0.07649046927690506
Test Losses Saved
Model saved
```

Figure: Results obtained with 5 epochs, with `batch_size` $= 32$.

Issues and Questions

Learning Rate Choice

We are currently using Adam, that is an *adaptive* learning rate method; this means that the learning rate that we set to 0.01 will be optimized, and represents only a sort of "initial condition". However, we are uncertain about the **impact** of this choice on the performance.

Bias in *Conv_3d*

From the PyTorch documentation of *Conv_3d*, we saw that the bias is applied by default.

$$out = W * x + bias$$

In our network, we apply a *Batch_Norm3d*, which performs the following operation:

$$out = \gamma \frac{out - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

We think that adding bias and then β may be redundant and worsen the performance.

Runtime memory exceeding

The current data loading strategy could be efficient to save memory and time but, on the other hand, this implies a huge runtime data memorization. Indeed, in `main.py`, the following data need to be saved:

- Macro tensor `X` with all the data from `cubes`
- The following training and testing subdivisions of `X`
- Dataset and dataloader

Just to load only `X`, more that 10GB must be used!

Questions

1. Choice of the learning rate (0.01 seems to have a good initial behaviour)
2. Adoption or elimination of the bias in *Conv_3d*
3. How to fully benefit from the data loading? Does not make much sense to save locally 3GB if more than 10GB need to be used runtime
4. About February conference