Two task network

Network has some inputs

1. The fixation.
2. The first context mod.
3. The second ontext mod.

Network has five outputs

1. The fixation.
2. The first output.
3. The second output

> Learning rule: superspike
>
> Neuron type: Lif + refrac
>
> Task: romo

In [ ]:

```
import torch
import numpy as np
import torch.nn as nn
import matplotlib.pyplot as plt  # for analys
from cgtasknet.net.lifrefrac import SNNLifRefrac
from cgtasknet.tasks.reduce import RomoTask, DefaultParams
from norse.torch.functional.lif_refrac import LIFRefracParameters
from norse.torch.functional.lif import LIFParameters

# from norse.torch import LIF
```

Step -1: Create dataset

In [ ]:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# device = torch.device('cpu')
print(f'Device: {("gpu (cuda)" if device.type=="cuda" else "cpu")}')
```

```
Device: gpu (cuda)
```

In [ ]:

```
batch_size = 200
number_of_tasks = 1
task_parameters = DefaultParams("RomoTask").generate_params()
task_parameters["delay"] = 0.1
task_parameters["trial_time"] = .15
Task = RomoTask(params=task_parameters, batch_size=batch_size)
Task_test = RomoTask(params=task_parameters, batch_size=1)
print("Task params:")
for key in task_parameters:
    if key != 'values':
        print(
            f"{key}".center(10, "."),
            f'value: {int((task_parameters[key] / task_parameters["dt"]))}ms'
        )
```

```
Task params:
....dt.... value: 1ms
..delay... value: 100ms
trial_time value: 150ms
```

Step 1.1: Create model

In [ ]:

```
feature_size, output_size = Task.feature_and_act_size
hidden_size = 400

neuron_parameters = LIFRefracParameters(
    LIFParameters(tau_mem_inv=torch.as_tensor(1/0.01).to(device),
        alpha=torch.as_tensor(100), method="super", v_th=torch.as_tensor(0.65
    ),
    rho_reset=torch.as_tensor(1),
)
model = SNNLifRefrac(
    feature_size,
    hidden_size,
    output_size,
    neuron_parameters=neuron_parameters,
    tau_filter_inv=500,
).to(device)
```

Step 1.2: Save pre-learning weights

In [ ]:

```
weights_pre_l = []
with torch.no_grad():
    for name, param in model.named_parameters():
        if param.requires_grad:
            weights_pre_l.append((param).cpu().numpy())
```

Step 2: loss and creterion

In [ ]:

```
learning_rate = 1e-3


class RMSELoss(nn.Module):
    def __init__(self):
        super().__init__()
        self.mse = nn.MSELoss()

    def forward(self, yhat, y):
        return torch.sqrt(self.mse(yhat, y))


criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
#optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Step 3: Train loop

In [ ]:

```
%matplotlib
plt.ion
fig = plt.figure()
ax = fig.add_subplot(111)

ax.set_title("lif")
inputs, target_outputs = Task.dataset(number_of_tasks)
(line1,) = ax.plot(np.arange(0, len(target_outputs)), target_outputs[:, 0, 1]
(line2,) = ax.plot(np.arange(0, len(target_outputs)), target_outputs[:, 0, 2]
(line3,) = ax.plot(np.arange(0, len(target_outputs)), target_outputs[:, 0, 1]
(line4,) = ax.plot(np.arange(0, len(target_outputs)), target_outputs[:, 0, 2]
ax.set_ylim([-0.5, 1.5])
ax.set_xlim([0, len(inputs)])
running_loss = 0
fig.canvas.draw()
fig.canvas.flush_events()
for i in range(2000):
    inputs, target_outputs = Task.dataset(number_of_tasks)
    #inputs = inputs * 2 - 1
    inputs += np.random.normal(0, 0.01, size=(inputs.shape))
    inputs = torch.from_numpy(inputs).type(torch.float).to(device)
    target_outputs = torch.from_numpy(target_outputs).type(torch.float).to(de
    #with torch.no_grad():
    #    inputs[:, :, 1:3] = layer_inputs(inputs[:, :, 1:3])[0]
    # zero the parameter gradients
    optimizer.zero_grad()

    # forward + backward + optimize
    outputs, states = model(inputs)

    loss = criterion(outputs, target_outputs)
    loss.backward()
    optimizer.step()

    # print statistics
    running_loss += loss.item()
    if i % 10 == 9:
        print("epoch: {:d} loss: {:0.5f}".format(i + 1, running_loss / 10))
        running_loss = 0.0
        with torch.no_grad():
            inputs, target_outputs = Task_test.dataset(number_of_tasks)

            inputs = torch.from_numpy(inputs).type(torch.float).to(device)
            target_outputs = (
                torch.from_numpy(target_outputs).type(torch.float).to(device)
            )
            outputs, states = model(inputs)
            loss = criterion(outputs, target_outputs)

            print("test loss: {:0.5f}".format(loss.item()))
        for_plot = outputs.detach().cpu().numpy()[:, 0, :]
        line1.set_xdata(np.arange(0, len(for_plot), 1))
        line2.set_xdata(np.arange(0, len(for_plot), 1))
        line3.set_xdata(np.arange(0, len(for_plot), 1))
        line4.set_xdata(np.arange(0, len(for_plot), 1))

        line1.set_ydata(for_plot[:, 1])
        line2.set_ydata(for_plot[:, 2])
```

```
Using matplotlib backend: TkAgg
epoch: 10 loss: 0.34523
test loss: 0.29775
epoch: 20 loss: 0.18154
test loss: 0.10930
epoch: 30 loss: 0.09917
test loss: 0.09239
epoch: 40 loss: 0.08717
test loss: 0.07875
epoch: 50 loss: 0.08205
test loss: 0.07456
epoch: 60 loss: 0.07837
test loss: 0.08505
epoch: 70 loss: 0.07456
test loss: 0.07988
epoch: 80 loss: 0.07122
test loss: 0.05847
epoch: 90 loss: 0.06742
test loss: 0.05603
epoch: 100 loss: 0.05926
test loss: 0.05674
epoch: 110 loss: 0.05482
test loss: 0.06480
epoch: 120 loss: 0.05343
test loss: 0.04669
epoch: 130 loss: 0.05214
test loss: 0.04291
epoch: 140 loss: 0.05146
test loss: 0.05227
epoch: 150 loss: 0.04972
test loss: 0.03806
epoch: 160 loss: 0.04469
test loss: 0.02474
epoch: 170 loss: 0.04398
test loss: 0.06919
epoch: 180 loss: 0.04292
test loss: 0.02924
epoch: 190 loss: 0.04219
test loss: 0.03524
epoch: 200 loss: 0.04152
test loss: 0.07941
epoch: 210 loss: 0.04031
test loss: 0.10058
epoch: 220 loss: 0.04076
test loss: 0.02359
epoch: 230 loss: 0.03985
test loss: 0.04361
epoch: 240 loss: 0.03954
test loss: 0.07988
epoch: 250 loss: 0.03966
test loss: 0.01765
epoch: 260 loss: 0.03813
test loss: 0.06159
```

```
epoch: 270 loss: 0.03787
test loss: 0.10789
epoch: 280 loss: 0.03803
test loss: 0.02833
epoch: 290 loss: 0.03712
test loss: 0.01762
epoch: 300 loss: 0.03754
test loss: 0.10708
epoch: 310 loss: 0.03624
test loss: 0.06607
epoch: 320 loss: 0.03637
test loss: 0.05688
epoch: 330 loss: 0.03568
test loss: 0.01658
epoch: 340 loss: 0.03449
test loss: 0.03299
epoch: 350 loss: 0.03453
test loss: 0.05363
epoch: 360 loss: 0.03472
test loss: 0.01489
epoch: 370 loss: 0.03355
test loss: 0.01348
epoch: 380 loss: 0.03405
test loss: 0.03098
epoch: 390 loss: 0.03441
test loss: 0.05921
epoch: 400 loss: 0.03289
test loss: 0.01335
epoch: 410 loss: 0.03343
test loss: 0.01780
epoch: 420 loss: 0.03303
test loss: 0.03029
epoch: 430 loss: 0.03312
test loss: 0.03705
epoch: 440 loss: 0.03156
test loss: 0.04696
epoch: 450 loss: 0.03282
test loss: 0.01160
epoch: 460 loss: 0.03189
test loss: 0.01171
epoch: 470 loss: 0.03203
test loss: 0.02293
epoch: 480 loss: 0.03141
test loss: 0.04600
epoch: 490 loss: 0.03149
test loss: 0.02930
epoch: 500 loss: 0.03138
test loss: 0.01978
epoch: 510 loss: 0.03181
test loss: 0.01234
epoch: 520 loss: 0.03195
test loss: 0.03823
epoch: 530 loss: 0.03247
test loss: 0.02182
epoch: 540 loss: 0.03238
test loss: 0.09548
epoch: 550 loss: 0.03080
test loss: 0.02905
epoch: 560 loss: 0.03176
test loss: 0.03291
```

```
epoch: 570 loss: 0.03110
test loss: 0.01196
epoch: 580 loss: 0.03110
test loss: 0.01158
epoch: 590 loss: 0.03088
test loss: 0.02435
epoch: 600 loss: 0.03000
test loss: 0.03264
epoch: 610 loss: 0.03073
test loss: 0.05520
epoch: 620 loss: 0.03127
test loss: 0.02474
epoch: 630 loss: 0.03156
test loss: 0.01151
epoch: 640 loss: 0.03112
test loss: 0.01236
epoch: 650 loss: 0.03059
test loss: 0.06846
epoch: 660 loss: 0.03064
test loss: 0.02370
epoch: 670 loss: 0.03005
test loss: 0.01005
epoch: 680 loss: 0.03013
test loss: 0.02843
epoch: 690 loss: 0.03036
test loss: 0.01128
epoch: 700 loss: 0.03057
test loss: 0.02000
epoch: 710 loss: 0.03028
test loss: 0.01927
epoch: 720 loss: 0.03006
test loss: 0.05004
epoch: 730 loss: 0.03112
test loss: 0.02796
epoch: 740 loss: 0.03081
test loss: 0.02756
epoch: 750 loss: 0.02922
test loss: 0.00993
epoch: 760 loss: 0.03059
test loss: 0.01434
epoch: 770 loss: 0.03005
test loss: 0.01086
epoch: 780 loss: 0.02978
test loss: 0.00992
epoch: 790 loss: 0.03092
test loss: 0.01764
epoch: 800 loss: 0.02892
test loss: 0.10808
epoch: 810 loss: 0.03019
test loss: 0.00970
epoch: 820 loss: 0.03011
test loss: 0.01677
epoch: 830 loss: 0.02941
test loss: 0.01051
epoch: 840 loss: 0.03070
test loss: 0.01066
epoch: 850 loss: 0.03048
test loss: 0.08385
epoch: 860 loss: 0.03000
test loss: 0.00978
```

```
epoch: 870 loss: 0.02941
test loss: 0.01047
epoch: 880 loss: 0.02967
test loss: 0.01093
epoch: 890 loss: 0.02900
test loss: 0.01829
epoch: 900 loss: 0.02858
test loss: 0.01664
epoch: 910 loss: 0.02908
test loss: 0.01556
epoch: 920 loss: 0.03005
test loss: 0.01045
epoch: 930 loss: 0.02979
test loss: 0.06875
epoch: 940 loss: 0.02849
test loss: 0.01202
epoch: 950 loss: 0.02835
test loss: 0.07402
epoch: 960 loss: 0.02902
test loss: 0.04737
epoch: 970 loss: 0.02942
test loss: 0.05179
epoch: 980 loss: 0.02844
test loss: 0.00811
epoch: 990 loss: 0.02862
test loss: 0.04430
epoch: 1000 loss: 0.02950
test loss: 0.01105
epoch: 1010 loss: 0.03013
test loss: 0.01085
epoch: 1020 loss: 0.02901
test loss: 0.01050
epoch: 1030 loss: 0.02862
test loss: 0.00996
epoch: 1040 loss: 0.03042
test loss: 0.04883
epoch: 1050 loss: 0.02888
test loss: 0.04871
epoch: 1060 loss: 0.02935
test loss: 0.02921
epoch: 1070 loss: 0.02926
test loss: 0.01021
epoch: 1080 loss: 0.03055
test loss: 0.01248
epoch: 1090 loss: 0.02903
test loss: 0.01698
epoch: 1100 loss: 0.02915
test loss: 0.00971
epoch: 1110 loss: 0.02763
test loss: 0.00915
epoch: 1120 loss: 0.02793
test loss: 0.05177
epoch: 1130 loss: 0.02746
test loss: 0.08599
epoch: 1140 loss: 0.02902
test loss: 0.00765
epoch: 1150 loss: 0.02809
test loss: 0.06766
epoch: 1160 loss: 0.02873
test loss: 0.02122
```

```
epoch: 1170 loss: 0.02717
test loss: 0.00839
epoch: 1180 loss: 0.02919
test loss: 0.00832
epoch: 1190 loss: 0.02828
test loss: 0.00912
epoch: 1200 loss: 0.02736
test loss: 0.02480
epoch: 1210 loss: 0.02791
test loss: 0.02039
epoch: 1220 loss: 0.02753
test loss: 0.02195
epoch: 1230 loss: 0.02953
test loss: 0.03143
epoch: 1240 loss: 0.02827
test loss: 0.04746
epoch: 1250 loss: 0.02815
test loss: 0.05934
epoch: 1260 loss: 0.02897
test loss: 0.02180
epoch: 1270 loss: 0.02871
test loss: 0.00852
epoch: 1280 loss: 0.02889
test loss: 0.09666
epoch: 1290 loss: 0.02965
test loss: 0.01005
epoch: 1300 loss: 0.02935
test loss: 0.04623
epoch: 1310 loss: 0.02887
test loss: 0.00838
epoch: 1320 loss: 0.02860
test loss: 0.01020
epoch: 1330 loss: 0.02863
test loss: 0.00870
epoch: 1340 loss: 0.02985
test loss: 0.01122
epoch: 1350 loss: 0.02812
test loss: 0.00959
epoch: 1360 loss: 0.03009
test loss: 0.01099
epoch: 1370 loss: 0.02821
test loss: 0.05256
epoch: 1380 loss: 0.02838
test loss: 0.02238
epoch: 1390 loss: 0.02848
test loss: 0.03860
epoch: 1400 loss: 0.02701
test loss: 0.08570
epoch: 1410 loss: 0.02630
test loss: 0.02113
epoch: 1420 loss: 0.02855
test loss: 0.00984
epoch: 1430 loss: 0.02735
test loss: 0.00858
epoch: 1440 loss: 0.02795
test loss: 0.02336
epoch: 1450 loss: 0.02820
test loss: 0.10180
epoch: 1460 loss: 0.02696
test loss: 0.04834
```

```
epoch: 1470 loss: 0.02882
test loss: 0.05183
epoch: 1480 loss: 0.02799
test loss: 0.02992
epoch: 1490 loss: 0.02728
test loss: 0.00792
epoch: 1500 loss: 0.02643
test loss: 0.08103
epoch: 1510 loss: 0.02870
test loss: 0.01189
epoch: 1520 loss: 0.02820
test loss: 0.06536
epoch: 1530 loss: 0.02862
test loss: 0.01203
epoch: 1540 loss: 0.03087
test loss: 0.07680
epoch: 1550 loss: 0.02962
test loss: 0.01069
epoch: 1560 loss: 0.02894
test loss: 0.00778
epoch: 1570 loss: 0.02883
test loss: 0.02382
epoch: 1580 loss: 0.02924
test loss: 0.05908
epoch: 1590 loss: 0.02837
test loss: 0.00716
epoch: 1600 loss: 0.02747
test loss: 0.01574
epoch: 1610 loss: 0.02888
test loss: 0.06123
epoch: 1620 loss: 0.02783
test loss: 0.00626
epoch: 1630 loss: 0.02808
test loss: 0.03318
epoch: 1640 loss: 0.02758
test loss: 0.01160
epoch: 1650 loss: 0.02659
test loss: 0.02427
epoch: 1660 loss: 0.02782
test loss: 0.00628
epoch: 1670 loss: 0.02739
test loss: 0.02707
epoch: 1680 loss: 0.02742
test loss: 0.02034
epoch: 1690 loss: 0.02764
test loss: 0.07843
epoch: 1700 loss: 0.02712
test loss: 0.03987
epoch: 1710 loss: 0.02676
test loss: 0.00761
epoch: 1720 loss: 0.02961
test loss: 0.02045
epoch: 1730 loss: 0.02854
test loss: 0.08208
epoch: 1740 loss: 0.02814
test loss: 0.02688
epoch: 1750 loss: 0.02790
test loss: 0.03314
epoch: 1760 loss: 0.02809
test loss: 0.00715
```

```
epoch: 1770 loss: 0.02731
test loss: 0.01720
epoch: 1780 loss: 0.02721
test loss: 0.01666
epoch: 1790 loss: 0.02708
test loss: 0.00701
epoch: 1800 loss: 0.02736
test loss: 0.04301
epoch: 1810 loss: 0.02692
test loss: 0.00647
epoch: 1820 loss: 0.02682
test loss: 0.00654
epoch: 1830 loss: 0.02805
test loss: 0.04106
epoch: 1840 loss: 0.02640
test loss: 0.06758
epoch: 1850 loss: 0.02637
test loss: 0.03992
epoch: 1860 loss: 0.02619
test loss: 0.10428
epoch: 1870 loss: 0.02756
test loss: 0.06711
epoch: 1880 loss: 0.02618
test loss: 0.01397
epoch: 1890 loss: 0.02794
test loss: 0.02221
epoch: 1900 loss: 0.02687
test loss: 0.00734
epoch: 1910 loss: 0.02766
test loss: 0.01235
epoch: 1920 loss: 0.02786
test loss: 0.01856
epoch: 1930 loss: 0.02643
test loss: 0.00751
epoch: 1940 loss: 0.02697
test loss: 0.00680
epoch: 1950 loss: 0.02756
test loss: 0.01131
epoch: 1960 loss: 0.02603
test loss: 0.05000
epoch: 1970 loss: 0.02755
test loss: 0.06852
epoch: 1980 loss: 0.02856
test loss: 0.04291
epoch: 1990 loss: 0.02862
test loss: 0.00797
epoch: 2000 loss: 0.02898
test loss: 0.06424
```

In [ ]:

```
torch.save(model.state_dict(), "Only_romo_lif_refrac_net")
```

In [ ]:

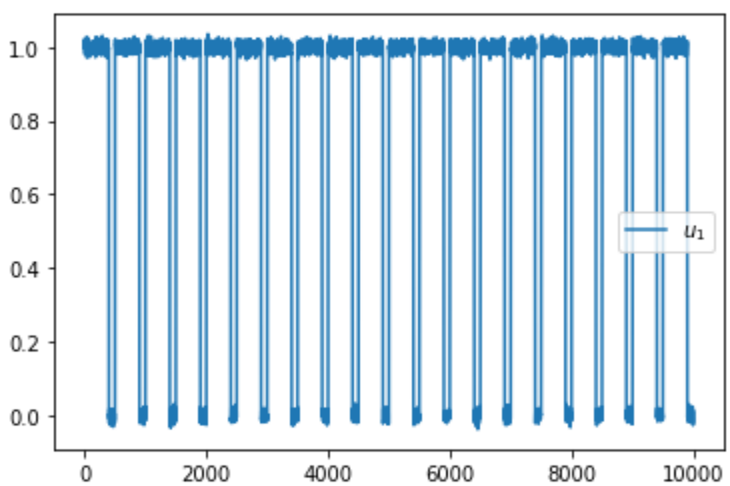
```
if False:
    model.load_state_dict(
        torch.load("Only_dm_lif_net")
    )
```

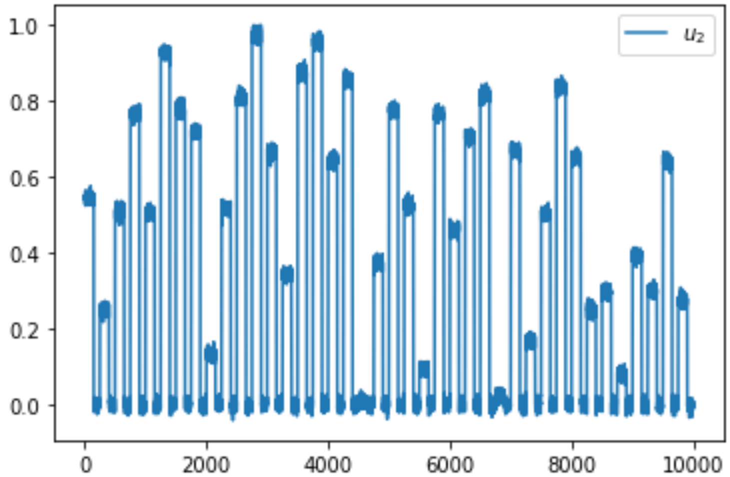
In [ ]:

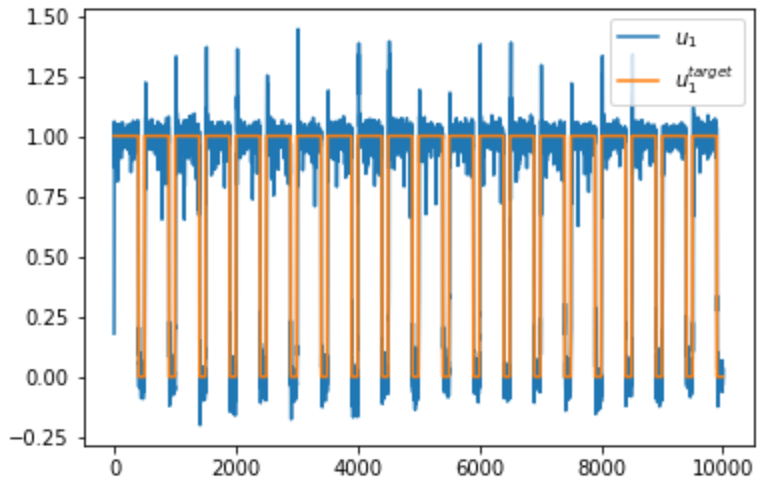
```
Taskplot = RomoTask(params=task_parameters, batch_size=1)
inputs, target_outputs = Taskplot.dataset(20)
inputs += np.random.normal(0, 0.01, size=(inputs.shape))
inputs = torch.from_numpy(inputs).type(torch.float).to(device)
target_outputs = torch.from_numpy(target_outputs).type(torch.float).to(device
outputs, states = model(inputs)
```

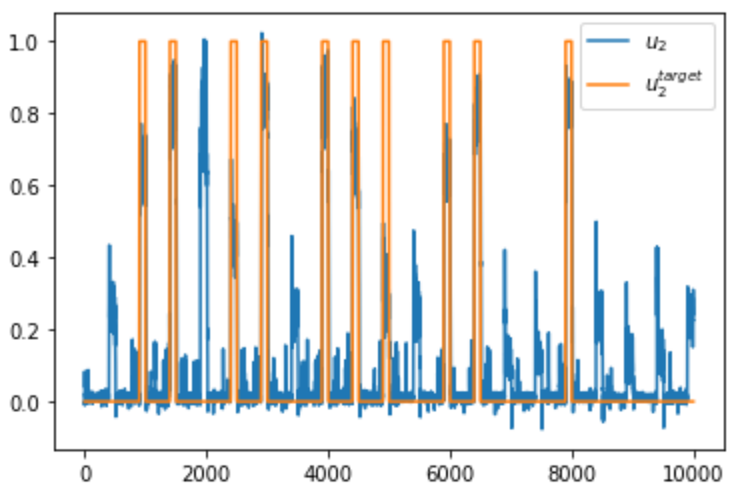
In [ ]:

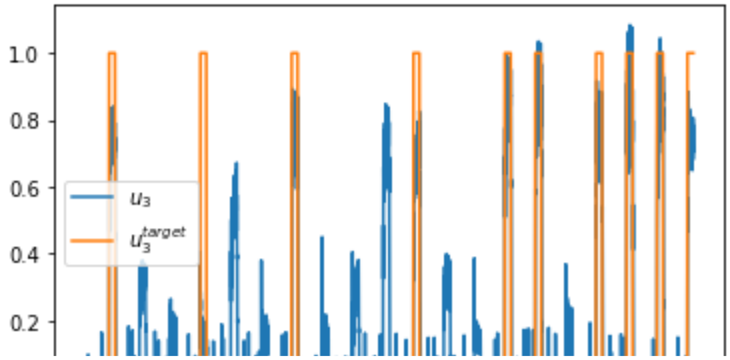
```
%matplotlib inline
for i in range(inputs.shape[2]):
    plt.plot(inputs[:, 0, i].detach().cpu().numpy(), label=fr"$u_{i + 1}$")
    plt.legend()
    plt.show()
    plt.close()
for i in range(outputs.shape[2]):
    plt.plot(outputs[:, 0, i].detach().cpu().numpy(), label=fr"$u_{i + 1}$")
    plt.plot(
        target_outputs[:, 0, i].detach().cpu().numpy(), label=fr"$u^{{target}
    )
    plt.legend()
    plt.show()
    plt.close()

plt.plot(outputs[:, 0, -2].detach().cpu().numpy() - outputs[:, 0, -1].detach(
plt.plot(
    target_outputs[:, 0, -2].detach().cpu().numpy(), label=fr"$u^{{target}}_{
)
plt.plot(
    target_outputs[:, 0, -1].detach().cpu().numpy(), label=fr"$u^{{target}}_{
)
plt.legend()
plt.show()
plt.close()
```




1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
u_2

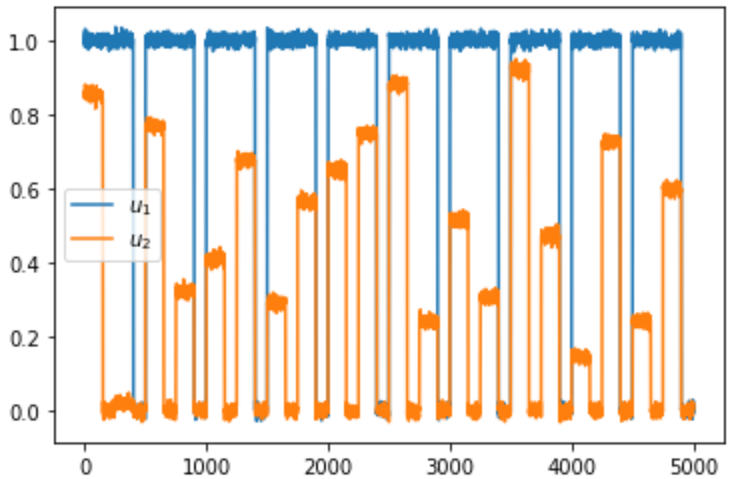

1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
0
2000
4000
6000
8000
10000
u_1
u_1^{target}


1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
u_2
u_2^{target}

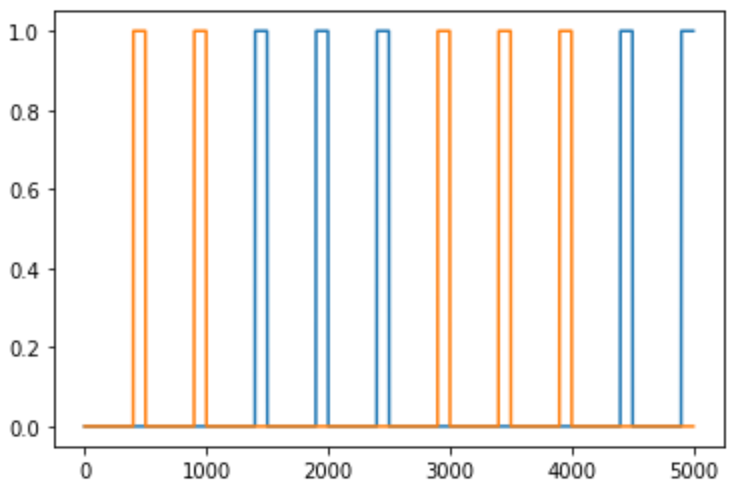


In [ ]:

```
%matplotlib inline
for i in range(inputs.shape[2]):
    plt.plot(inputs[:, 0, i].detach().cpu().numpy(), label=fr"$u_{i + 1}$")

plt.legend()
plt.show()
plt.close()
plt.plot(target_outputs[:, 0, 1].detach().cpu().numpy(), label=fr'$y_{2}$')
plt.plot(target_outputs[:, 0, 2].detach().cpu().numpy(), label=fr'$y_{3}$')
```



Out[ ]: [<matplotlib.lines.Line2D at 0x7fd811779610>]

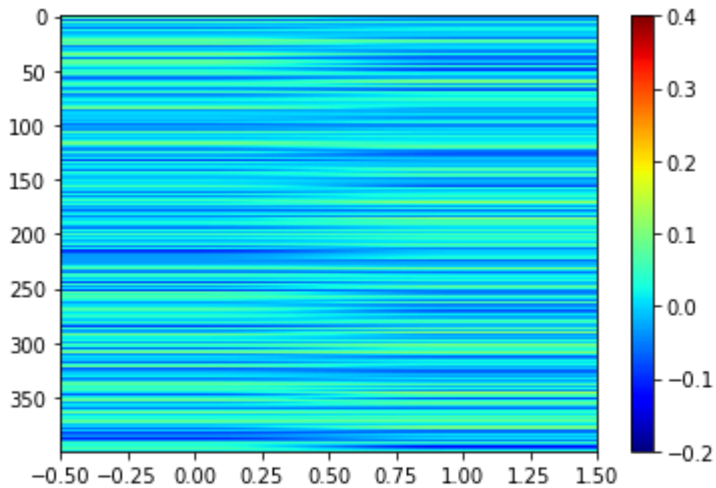


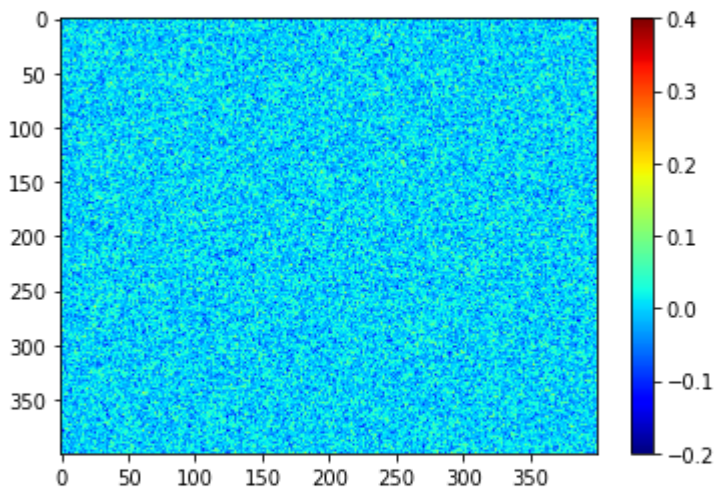
In [ ]:

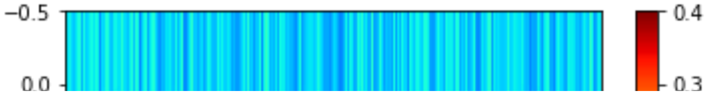
```
weights_post_l = []
with torch.no_grad():
    for name, param in model.named_parameters():
        if param.requires_grad:
            weights_post_l.append((param).cpu().numpy())
```

In [ ]:

```
%matplotlib inline
for i in range(len(weights_pre_l) - 1):
    plt.imshow((weights_pre_l[i]), aspect='auto',cmap='jet', vmin=-.2, vmax=0
    plt.colorbar()
    plt.show()
```

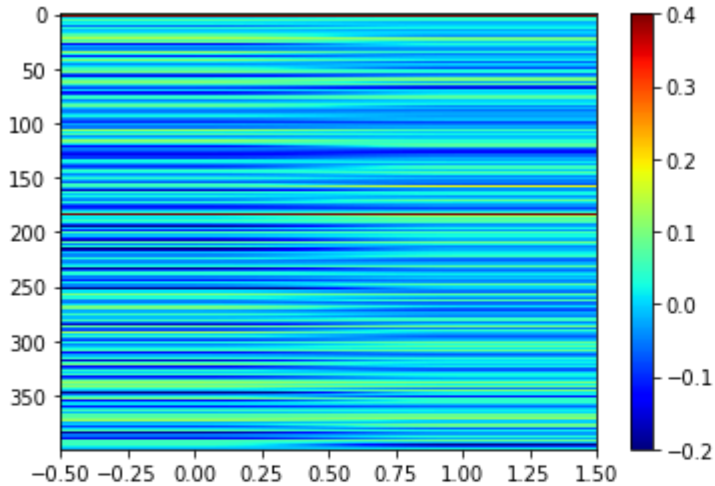


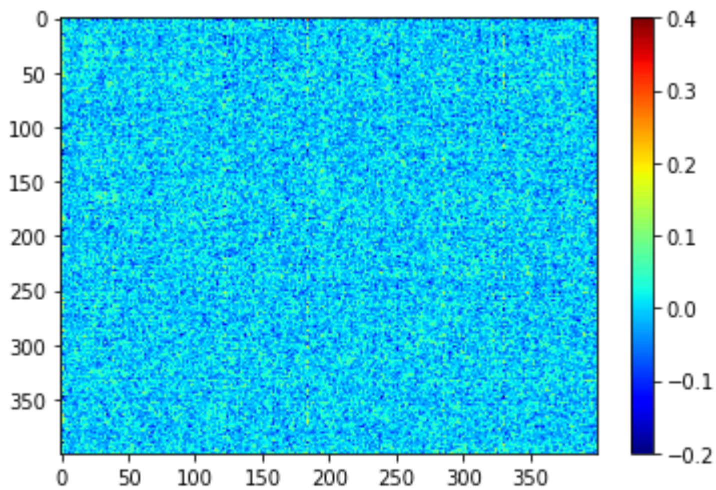


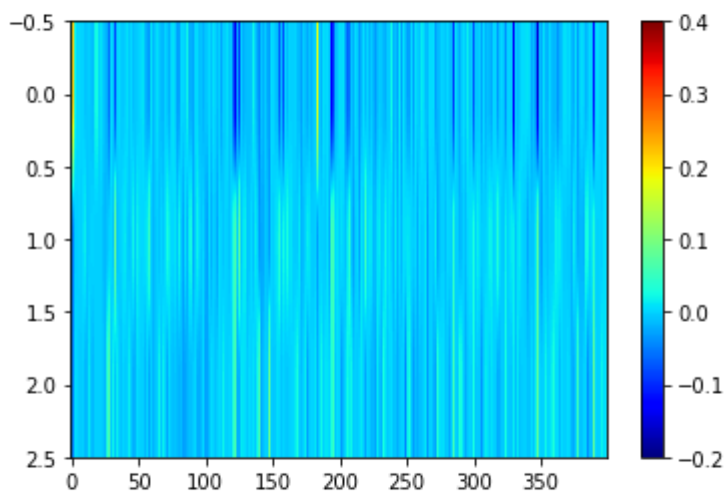


In [ ]:

```
%matplotlib inline
for i in range(len(weights_pre_l) - 1):
    plt.imshow((weights_post_l[i]), aspect='auto', cmap='jet', vmin=-.2, vmax
    plt.colorbar()
    plt.show()
```

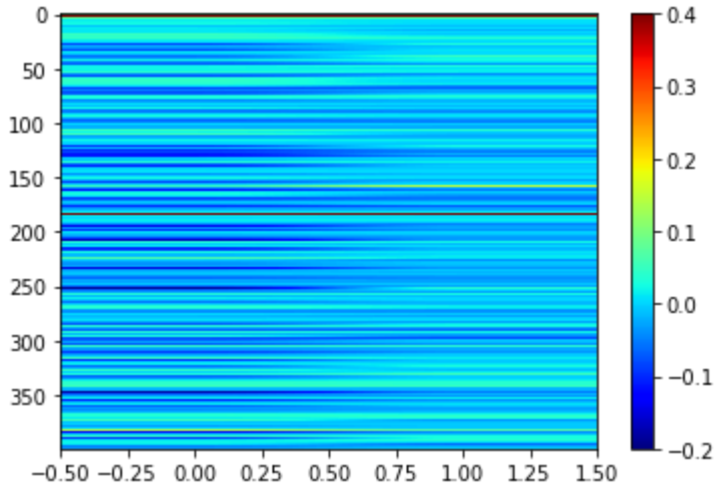


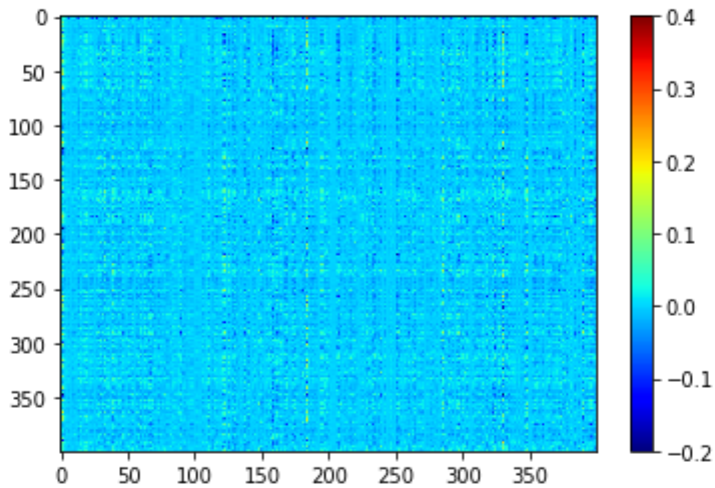


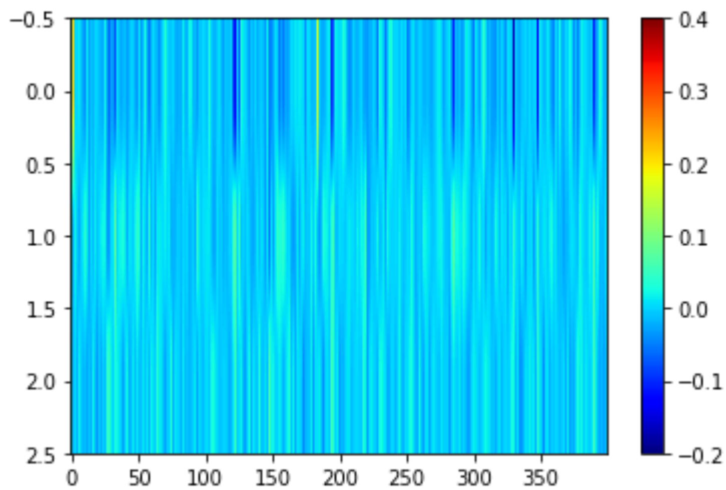


In [ ]:

```
%matplotlib inline
for i in range(len(weights_pre_l) - 1):
    plt.imshow((weights_post_l[i] - weights_pre_l[i]), aspect='auto', cmap='j
    plt.colorbar()
    plt.show()
```







In [ ]: