Exercise_4

May 26, 2025

1 Advanced Deep Learning for Physics (IN2298):

1.1 Exercise 4: Supervised Network Training

[65]:
```
# Install PhyFlow and Import libraries
%%capture
!pip install --quiet phiflow;
!pip install nbconvert;
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc;
from google.colab import drive
drive.mount("/content/drive");
from phi.torch.flow import *
```

1.1.1 Analytic trajectories

def. analytic state function

[66]:
```
def exact_state( dt: float, p0: tensor,v0: tensor):
  g = math.vec(x=0.,y=-9.8)
  return p0 + v0*dt + 0.5*g*dt**2, v0 + g*dt
```

plot analytic trajectories

[67]:
```
# initial conditions
p0 = math.vec(x=10., y=10.)
v0 = math.vec(x=1.,y=13.)
dt = 1
t = 100
n_step = int(t/dt)

# first trajectory
p_traj,v_traj = iterate(lambda p, v: exact_state(dt ,p, v),batch(time=n_step),␣
 ↪p0, v0)

# plot trajectory
plot(p_traj,color='#28d642', animate='time')
```

[67]:
```
<matplotlib.animation.FuncAnimation at 0x7f807c8c7250>
```

```
<Figure size 640x480 with 0 Axes>
```

1.1.2 Simulation

def. simulation state function

[68]:
```
def sim_state(dt: float, p0: tensor, v0: tensor):
  g = math.vec(x=0.,y=-9.8)
  return p0 + v0 * dt, v0 + g * dt
```

plot simulated trajectories

[69]:
```
# initial conditions
p0 = math.vec(x=10., y=10.)
v0 = math.vec(x=1.,y=13.)
dt = 1
t = 100
n_step = int(t/dt)

# first trajectory
p_sim_traj,v_sim_traj = iterate(lambda p, v: sim_state(dt ,p,␣
 ↪v),batch(time=n_step), p0, v0)

# plot trajectory
plot(p_sim_traj,color = '#d62828', animate='time')
```

[69]:
```
<matplotlib.animation.FuncAnimation at 0x7f807c906290>

<Figure size 640x480 with 0 Axes>
```

plot trajectories together

[70]:
```
traj = math.stack([p_traj,p_sim_traj],instance('trajectory'))
color = color = wrap(['#28d642','#d62828'],instance(traj))
plot(traj,color = color, animate='time')
```

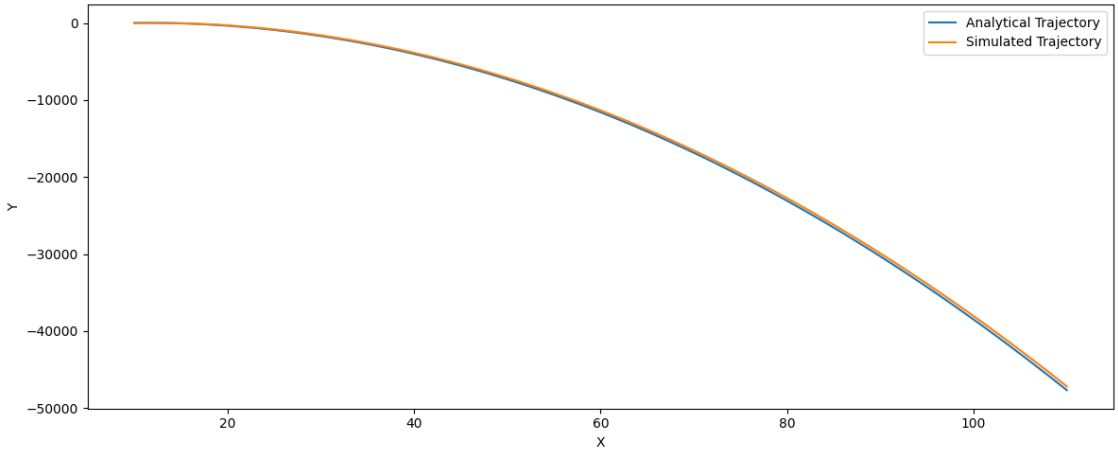
```
/usr/local/lib/python3.11/dist-packages/phiml/math/_tensors.py:1379:
RuntimeWarning: invalid value encountered in power
  result = op(n1, n2)
```

[70]:
```
<matplotlib.animation.FuncAnimation at 0x7f808079fd10>

<Figure size 640x480 with 0 Axes>
```

[71]:
```
#Frame visualization

# Extract x and y vectors and combine into a spatial field
an_xy = math.stack([p_traj.vector['x'], p_traj.vector['y']],␣
 ↪channel(vector='x,y'))
```

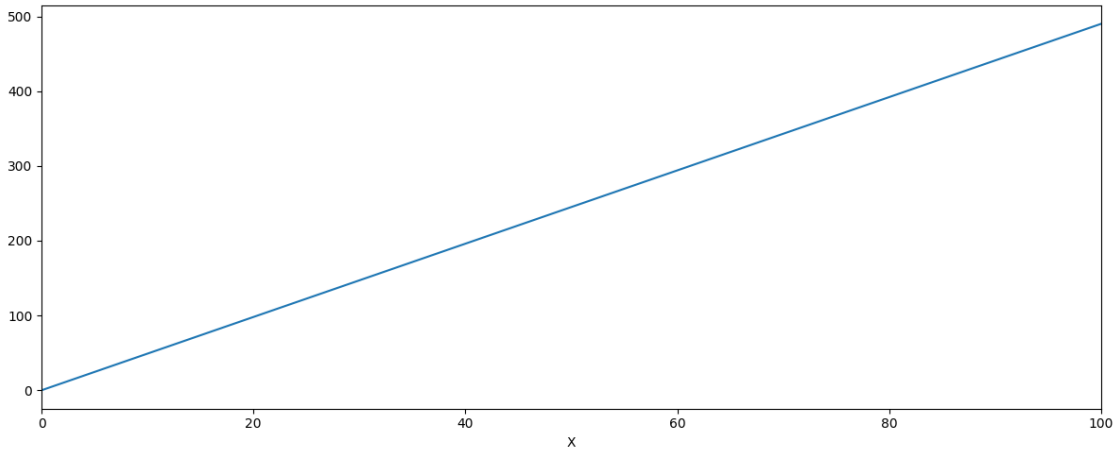
```
corr_xy = math.stack([p_sim_traj.vector['x'], p_sim_traj.vector['y']],␣
 ↪channel(vector='x,y'))

# Pack time dimension into sample points
an_xy = math.pack_dims(an_xy, 'time', 't')
corr_xy = math.pack_dims(corr_xy, 'time', 't')

# Plot both trajectories on the same plot
show(math.stack({'Analytical Trajectory': an_xy,'Simulated Trajectory':␣
 ↪corr_xy},channel('trajectory')))
```

```
/usr/local/lib/python3.11/dist-packages/phiml/math/_tensors.py:1379:
RuntimeWarning: invalid value encountered in power
  result = op(n1, n2)
```



plot error for positions

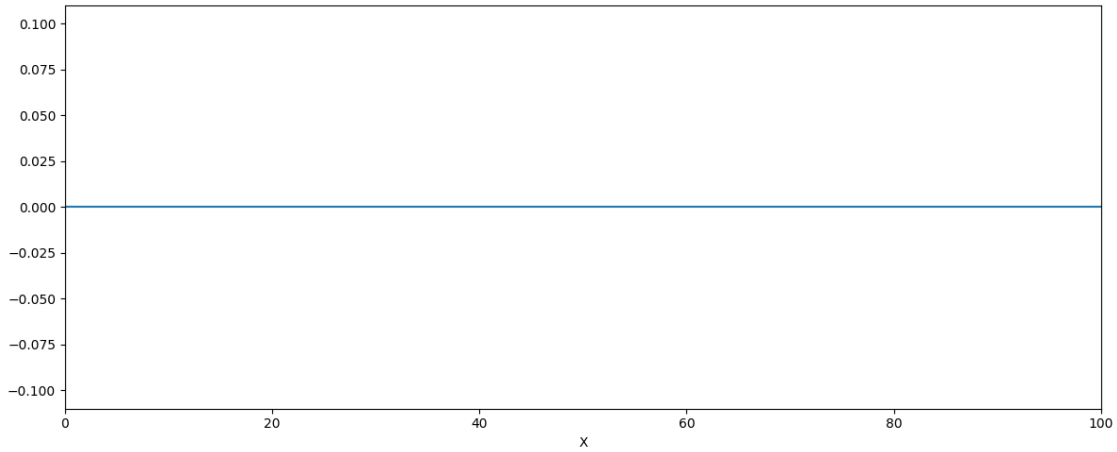
[72]:
```
x = p_traj.vector['x']
error = p_sim_traj - p_traj
error = math.pack_dims(error, dims='time', packed_dim='x')
x = math.pack_dims(x, dims='time', packed_dim='x')
show(error.vector['y'])
```



plot error for velocities

[73]:
```
x = v_traj.vector['x']
error = v_sim_traj - v_traj
print(error)
error = math.pack_dims(error, dims='time', packed_dim='x')
x = math.pack_dims(x, dims='time', packed_dim='x')
show(error.vector['y'])
```

```
(time=101, vector=x,y) const 0.0
```



1.1.3 Data Set

[74]:
```
# Set the seed to have always the same Dataset
torch.manual_seed(42)
sample_list = []
an_sample_list = []
n_samples = 500
n_step = 100
dt = 1
scaling = math.linspace(0.0, n_step, batch(time=n_step+1))
scaling = math.expand(scaling, channel(vector='x,y'))

for sample in range(n_samples):
  # Defining p0,v0
  p0 = math.random_uniform(channel(vector='x,y'))*10
  v0 = math.random_uniform(channel(vector='x,y'))*10

  # Defining analytical and simulated trajectories
  an_pos_traj, an_vel_traj  = iterate(lambda p, v: exact_state(dt ,p,␣
 ↪v),batch(time=n_step), p0, v0)
  sim_pos_traj,sim_vel_traj = iterate(lambda p, v: sim_state(dt ,p,␣
 ↪v),batch(time=n_step), p0, v0)

  #print(an_pos_traj)
  sample_state = math.concat([sim_pos_traj,sim_vel_traj],'vector')
  sample_an_state = math.concat([an_pos_traj,an_vel_traj],'vector')

  sample_list.append(sample_state)
  an_sample_list.append(sample_an_state)

state = math.concat(sample_list,'time')
an_state = math.concat(an_sample_list,'time')

print(state)
print(an_state)
```

```
(time=50500, vector=x,y,x,y) -4.03e+03 ± 1.0e+04
(-5e+04…1e+03)
(time=50500, vector=x,y,x,y) -4.09e+03 ± 1.0e+04
(-5e+04…1e+03)
```

1.1.4 Neural Corrector

[75]:
```
net = dense_net(4, 4, layers=[32, 32], activation='ReLU')
optimizer = adam(net, 1e-3)
```

[76]:
```
@math.jit_compile
def loss_function(state: tensor,an_state: tensor):
  predicted_delta_state = math.native_call(net,state)
  predicted_delta_state = math.
 ↪rename_dims(predicted_delta_state,channel('vector'),channel(vector='x,y,x,y'))
  delta_state = state - an_state
  diff = delta_state-predicted_delta_state
  return math.l2_loss(diff)
```

[77]:
```
for i in range(5000):
  loss = update_weights(net,optimizer,loss_function,state,an_state)
  if i % 1000 == 0: print(loss)
```

```
(time =50500) 9.73e+05 ± 1.2e+06 (2e-02…5e+06)
(time =50500) 1.066 ± 1.083 (2e-05…1e+01)
(time =50500) 1.026 ± 1.017 (4e-05…9e+00)
(time =50500) 1.205 ± 1.221 (2e-04…9e+00)
(time =50500) 1.011 ± 1.019 (8e-05…1e+01)
```

Visualize results

[78]:
```
# Defining p0,v0
p0 = math.random_uniform(channel(vector='x,y'))*10
v0 = math.random_uniform(channel(vector='x,y'))*10

# Defining analytical and simulated trajectories
an_pos_traj, an_vel_traj  = iterate(lambda p, v: exact_state(dt ,p,␣
 ↪v),batch(time=n_step), p0, v0)
sim_pos_traj,sim_vel_traj = iterate(lambda p, v: sim_state(dt ,p,␣
 ↪v),batch(time=n_step), p0, v0)

print(an_pos_traj)
state = math.concat([sim_pos_traj,sim_vel_traj],'vector')
an_state = math.concat([an_pos_traj,an_vel_traj],'vector')

predictions = math.native_call(net,state)
print(predictions)
```

```
(time =101, vector =x,y) -7.81e+03 ± 1.3e+04
(-5e+04…1e+03)
(time =101, vector =4) 61.193 ± 128.390
(-2e+00…5e+02)
```

Visualize trajectories

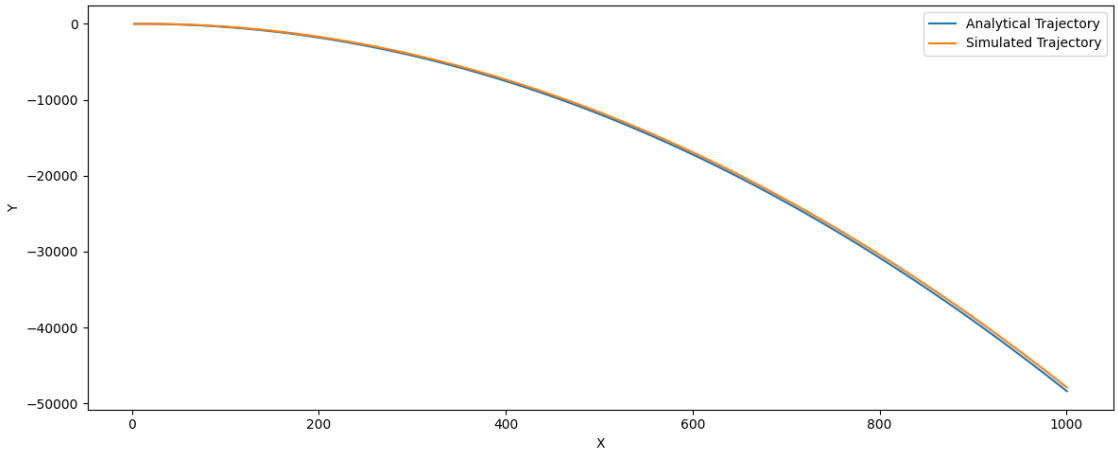
[79]:
```
# True Trajectory
traj = math.stack([an_pos_traj,sim_pos_traj],instance('trajectory'))
color = color = wrap(['#28d642','#d62828'],instance(traj))
```

```
plot(traj,color = color, animate='time')
```

[79]: <matplotlib.animation.FuncAnimation at 0x7f807cf70f10>

<Figure size 640x480 with 0 Axes>

[80]:
```
#Frame visualization

# Extract x and y vectors and combine into a spatial field
an_xy = math.stack([an_pos_traj.vector['x'], an_pos_traj.vector['y']],␣
 ↪channel(vector='x,y'))
corr_xy = math.stack([sim_pos_traj.vector['x'], sim_pos_traj.vector['y']],␣
 ↪channel(vector='x,y'))

# Pack time dimension into sample points
an_xy = math.pack_dims(an_xy, 'time', 't')
corr_xy = math.pack_dims(corr_xy, 'time', 't')

# Plot both trajectories on the same plot
show(math.stack({'Analytical Trajectory': an_xy,'Simulated Trajectory':␣
 ↪corr_xy},channel('trajectory')))
```



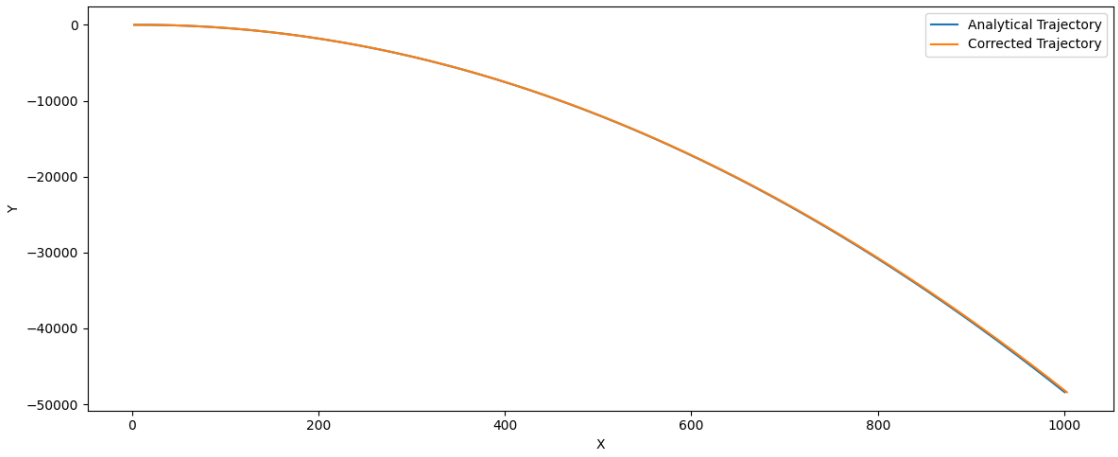
[81]:
```
predicted_delta_pos = predictions.vector[:2]
predicted_delta_pos = math.rename_dims(predicted_delta_pos, channel('vector'),␣
 ↪channel(vector='x,y'))
corrected_traj = sim_pos_traj - predicted_delta_pos
```

[82]:
```
# Corrected Trajectory
traj = math.stack([an_pos_traj,corrected_traj],instance('trajectory'))
color = color = wrap(['#28d642','#d62828'],instance(traj))
plot(traj,color = color, animate='time')
```

[82]: <matplotlib.animation.FuncAnimation at 0x7f807cef8250>

<Figure size 640x480 with 0 Axes>

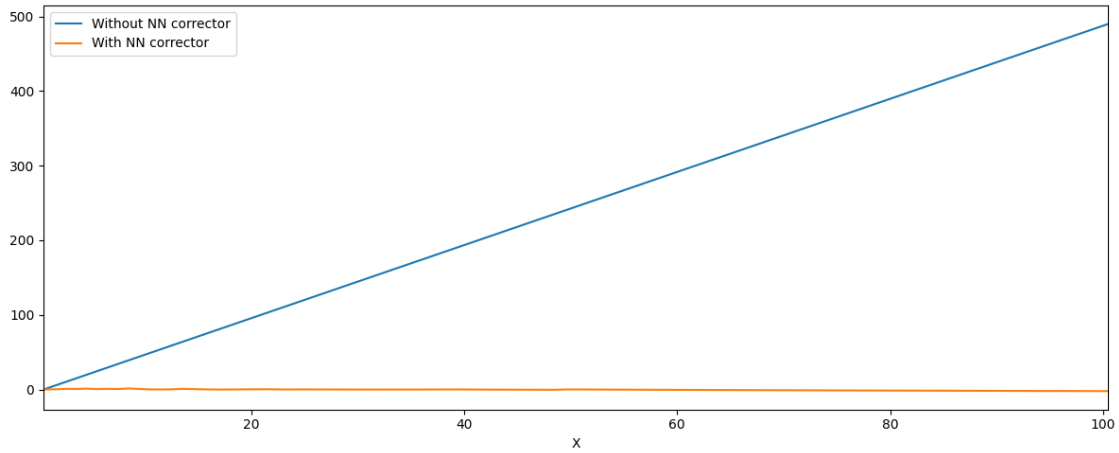
[83]:
```
#Frame visualization

# Extract x and y vectors and combine into a spatial field
an_xy = math.stack([an_pos_traj.vector['x'], an_pos_traj.vector['y']],␣
 ↪channel(vector='x,y'))
corr_xy = math.stack([corrected_traj.vector['x'], corrected_traj.vector['y']],␣
 ↪channel(vector='x,y'))

# Pack time dimension into sample points
an_xy = math.pack_dims(an_xy, 'time', 't')
corr_xy = math.pack_dims(corr_xy, 'time', 't')

# Plot both trajectories on the same plot
show(math.stack({'Analytical Trajectory': an_xy,'Corrected Trajectory':␣
 ↪corr_xy},channel('trajectory')))
```



Errors

[84]:
```
# Error without Neural corrector
x = sim_pos_traj.vector['x']
errore = sim_pos_traj - an_pos_traj
errore = math.pack_dims(errore, dims='time', packed_dim='x')
x = math.pack_dims(x, dims='time', packed_dim='x')

# Error with neural corrector
x = corrected_traj.vector['x']
error = corrected_traj - an_pos_traj
error = math.pack_dims(error, dims='time', packed_dim='x')
```

```
x = math.pack_dims(x, dims='time', packed_dim='x')

show(CenteredGrid(math.stack({
    "Without NN corrector": errore.vector['y'],
    "With NN corrector": error.vector['y']
},channel('trajectory'))))
```



[87]:
```
%%capture
!jupyter nbconvert --to pdf --output /content/drive/MyDrive/Fisica/ADL4P/
  ↪Exercise_4.pdf /content/drive/MyDrive/Fisica/ADL4P/Exercise_4.ipynb
```