```
%\documentclass{article}
\documentclass[12pt]{report}
\usepackage{titling}
\usepackage{graphicx}
\graphicspath{{./images/}}
\usepackage{amsmath}
\usepackage{algorithm}
\usepackage{verbatim}
\usepackage{algpseudocode}
\usepackage{caption}
\usepackage{cite}
\usepackage[utf8]{inputenc}  % Ensures proper handling of accented characters
\usepackage[T1]{fontenc}
\usepackage{url}            % For \url{} in references

%\title{Project Report: Implementation of Neural Simulated Annealing}


%\author{
%Ishanya (21329) \texttt{ishanya21@iiserb.ac.in} \\
%HariKrishna (22236) \texttt{peddinti22@iiserb.ac.in} \\
%Hiba KT (22146) \texttt{hiba22@iiserb.ac.in} \\
%Astha (22063) \texttt{astha22@iiserb.ac.in} \\
%\\
%IISER BHOPAL
%}

\begin{titlepage}
    \centering
    \vspace*{2cm}

    {\Huge \bfseries Project Report\par}
    \vspace{0.5cm}
    {\LARGE Implementation of Neural Simulated Annealing\par}
    \vspace{1cm}
    {\large DSE/ECS 311: Project Presentation\par}

    \vfill  % Pushes everything above towards the top

    \begin{flushleft}
    \textbf{Submitted by:} \\
    Ishanya (21329) \texttt{ishanya21@iiserb.ac.in} \\
    HariKrishna (22236) \texttt{peddinti22@iiserb.ac.in} \\
    Hiba KT (22146) \texttt{hiba22@iiserb.ac.in} \\
    Astha (22063) \texttt{astha22@iiserb.ac.in}
    \end{flushleft}

    \vfill  % Pushes logo to bottom of the page

    {\large Indian Institute of Science Education and Research (IISER) Bhopal \par}
    \vspace{0.5cm}
    \includegraphics[width=0.25\textwidth]{iiserb.png}

    \vspace{1cm}
    {\large April 2025}

\end{titlepage}

%\date{March 14, 2025}

\begin{document}
\maketitle
\tableofcontents
\newpage
```

ES Optimiser

```
!python scripts/main.py +experiment=knapsack_es training.n_epochs=200
training.batch_size=500
```

```
CUDA device not found. Running on CPU.
n_problems: 256
problem_dim: 50
embed_dim: 16
training:
  method: es
  reward: min_cost
  n_epochs: 200
  lr: 0.001
  batch_size: 500
  ppo_epochs: 10
  trace_decay: 0.9
  eps_clip: 0.25
  gamma: 0.9
  weight_decay: 0.01
  momentum: 0.9
  stddev: 0.05
  population: 16
  milestones:
  - 0.9
  optimizer: adam
sa:
  init_temp: 1.0
  stop_temp: 0.1
  outer_steps: 100
  inner_steps: 1
  alpha: 0.9772372209558107
problem: knapsack
capacity: 12.5
device: cpu
model_path: null
results_path: results
data_path: datasets
seed: 42

Training loss: -16.7765: 100% 200/200 [16:53<00:00,  5.07s/it]
CUDA device not found. Running on cpu.
Loaded model at  models/knapsack50-es.pt
1x, K=50, random seed 1 sampled: -14.86
1x, K=50, random seed 2 sampled: -14.79
1x, K=50, random seed 3 sampled: -14.96
1x, K=50, random seed 4 sampled: -14.86
1x, K=50, random seed 5 sampled: -14.86
2x, K=100, random seed 1 sampled: -16.71
2x, K=100, random seed 2 sampled: -16.77
2x, K=100, random seed 3 sampled: -16.86
2x, K=100, random seed 4 sampled: -16.75
2x, K=100, random seed 5 sampled: -16.69
5x, K=250, random seed 1 sampled: -18.18
5x, K=250, random seed 2 sampled: -18.21
5x, K=250, random seed 3 sampled: -18.22
5x, K=250, random seed 4 sampled: -18.16
5x, K=250, random seed 5 sampled: -18.17
10x, K=500, random seed 1 sampled: -18.77
10x, K=500, random seed 2 sampled: -18.76
10x, K=500, random seed 3 sampled: -18.79
10x, K=500, random seed 4 sampled: -18.76
10x, K=500, random seed 5 sampled: -18.73
    MODE     K                       COST          TIME
```

```
 Sampled    1x          -14.866 +- 0.054       0:00:00
  Greedy    1x            -19.781 +- 0.0       0:00:00
 Sampled    2x          -16.757 +- 0.057       0:00:00
 Sampled    5x          -18.186 +- 0.023       0:00:00
  Random   10x           -18.36 +- 0.029       0:00:01
 Sampled   10x           -18.762 +- 0.02       0:00:01
```

[41]

16m

```
#ADAMW
!python scripts/main.py +experiment=knapsack_es training.n_epochs=200 training.batch_size=500 training.optimizer=adamw
!python scripts/eval.py +experiment=knapsack_es
!python scripts/print_results.py +experiment=knapsack_es
```

```
CUDA device not found. Running on CPU.
n_problems: 256
problem_dim: 50
embed_dim: 16
training:
  method: es
  reward: min_cost
  n_epochs: 200
  lr: 0.001
  batch_size: 500
  ppo_epochs: 10
  trace_decay: 0.9
  eps_clip: 0.25
  gamma: 0.9
  weight_decay: 0.01
  momentum: 0.9
  stddev: 0.05
  population: 16
  milestones:
  - 0.9
  optimizer: adamw
sa:
  init_temp: 1.0
  stop_temp: 0.1
  outer_steps: 100
  inner_steps: 1
  alpha: 0.9772372209558107
problem: knapsack
capacity: 12.5
device: cpu
model_path: null
results_path: results
data_path: datasets
seed: 42

Training loss: -16.7800: 100% 200/200 [16:21<00:00,  4.91s/it]
CUDA device not found. Running on cpu.
Loaded model at  models/knapsack50-es.pt
1x, K=50, random seed 1 sampled: -14.87
1x, K=50, random seed 2 sampled: -14.78
1x, K=50, random seed 3 sampled: -14.96
1x, K=50, random seed 4 sampled: -14.84
1x, K=50, random seed 5 sampled: -14.86
2x, K=100, random seed 1 sampled: -16.73
2x, K=100, random seed 2 sampled: -16.75
2x, K=100, random seed 3 sampled: -16.84
2x, K=100, random seed 4 sampled: -16.75
2x, K=100, random seed 5 sampled: -16.69
5x, K=250, random seed 1 sampled: -18.18
5x, K=250, random seed 2 sampled: -18.22
5x, K=250, random seed 3 sampled: -18.22
5x, K=250, random seed 4 sampled: -18.16
5x, K=250, random seed 5 sampled: -18.16
10x, K=500, random seed 1 sampled: -18.79
10x, K=500, random seed 2 sampled: -18.76
10x, K=500, random seed 3 sampled: -18.80
10x, K=500, random seed 4 sampled: -18.76
10x, K=500, random seed 5 sampled: -18.74
    MODE     K                      COST           TIME
 Sampled    1x          -14.861 +- 0.057       0:00:00
  Greedy    1x            -19.784 +- 0.0       0:00:00
 Sampled    2x          -16.752 +- 0.049       0:00:00
 Sampled    5x          -18.188 +- 0.026       0:00:00
  Random   10x           -18.36 +- 0.029       0:00:01
 Sampled   10x          -18.769 +- 0.024       0:00:01
```

[42]

17m

1

```
#ADAGRAD

!python scripts/main.py +experiment=knapsack_es training.n_epochs=200 training.batch_size=500 training.optimizer=adagrad

!python scripts/eval.py +experiment=knapsack_es

!python scripts/print_results.py +experiment=knapsack_es
```

```
CUDA device not found. Running on CPU.
n_problems: 256
problem_dim: 50
embed_dim: 16
training:
  method: es
  reward: min_cost
  n_epochs: 200
  lr: 0.001
  batch_size: 500
  ppo_epochs: 10
  trace_decay: 0.9
  eps_clip: 0.25
  gamma: 0.9
  weight_decay: 0.01
  momentum: 0.9
  stddev: 0.05
  population: 16
  milestones:
  - 0.9
  optimizer: adagrad
sa:
  init_temp: 1.0
  stop_temp: 0.1
  outer_steps: 100
  inner_steps: 1
  alpha: 0.9772372209558107
problem: knapsack
capacity: 12.5
device: cpu
model_path: null
results_path: results
data_path: datasets
```

```
seed: 42

Training loss: -16.3340: 100% 200/200 [16:49<00:00,  5.05s/it]
CUDA device not found. Running on cpu.
Loaded model at  models/knapsack50-es.pt
1x, K=50, random seed 1 sampled: -14.30
1x, K=50, random seed 2 sampled: -14.22
1x, K=50, random seed 3 sampled: -14.49
1x, K=50, random seed 4 sampled: -14.25
1x, K=50, random seed 5 sampled: -14.36
2x, K=100, random seed 1 sampled: -16.34
2x, K=100, random seed 2 sampled: -16.18
2x, K=100, random seed 3 sampled: -16.54
2x, K=100, random seed 4 sampled: -16.33
2x, K=100, random seed 5 sampled: -16.23
5x, K=250, random seed 1 sampled: -17.97
5x, K=250, random seed 2 sampled: -17.99
5x, K=250, random seed 3 sampled: -18.01
5x, K=250, random seed 4 sampled: -17.93
5x, K=250, random seed 5 sampled: -17.95
10x, K=500, random seed 1 sampled: -18.61
10x, K=500, random seed 2 sampled: -18.61
10x, K=500, random seed 3 sampled: -18.61
10x, K=500, random seed 4 sampled: -18.59
10x, K=500, random seed 5 sampled: -18.61
    MODE     K                      COST            TIME
 Sampled    1x          -14.323 +- 0.094       0:00:00
  Greedy    1x             -19.13 +- 0.0       0:00:00
 Sampled    2x          -16.323 +- 0.124       0:00:00
 Sampled    5x          -17.969 +- 0.028       0:00:00
  Random   10x           -18.36 +- 0.029       0:00:01
 Sampled   10x          -18.606 +- 0.009       0:00:01
```

---

[43]

18m

```
#RMSPROP

!python scripts/main.py +experiment=knapsack_es training.n_epochs=200
training.batch_size=500 training.optimizer=rmsprop
```

```
!python scripts/eval.py +experiment=knapsack_es

!python scripts/print_results.py +experiment=knapsack_es
```

```
CUDA device not found. Running on CPU.
n_problems: 256
problem_dim: 50
embed_dim: 16
training:
  method: es
  reward: min_cost
  n_epochs: 200
  lr: 0.001
  batch_size: 500
  ppo_epochs: 10
  trace_decay: 0.9
  eps_clip: 0.25
  gamma: 0.9
  weight_decay: 0.01
  momentum: 0.9
  stddev: 0.05
  population: 16
  milestones:
  - 0.9
  optimizer: rmsprop
sa:
  init_temp: 1.0
  stop_temp: 0.1
  outer_steps: 100
  inner_steps: 1
  alpha: 0.9772372209558107
problem: knapsack
capacity: 12.5
device: cpu
model_path: null
results_path: results
data_path: datasets
seed: 42

Training loss: -19.9331: 100% 200/200 [17:25<00:00,  5.23s/it]
CUDA device not found. Running on cpu.
Loaded model at  models/knapsack50-es.pt
1x, K=50, random seed 1 sampled: -19.65
1x, K=50, random seed 2 sampled: -19.63
1x, K=50, random seed 3 sampled: -19.64
1x, K=50, random seed 4 sampled: -19.64
1x, K=50, random seed 5 sampled: -19.69
2x, K=100, random seed 1 sampled: -19.88
```

```
2x, K=100, random seed 2 sampled: -19.88
2x, K=100, random seed 3 sampled: -19.88
2x, K=100, random seed 4 sampled: -19.89
2x, K=100, random seed 5 sampled: -19.87
5x, K=250, random seed 1 sampled: -19.99
5x, K=250, random seed 2 sampled: -19.99
5x, K=250, random seed 3 sampled: -19.99
5x, K=250, random seed 4 sampled: -20.00
5x, K=250, random seed 5 sampled: -19.99
10x, K=500, random seed 1 sampled: -20.05
10x, K=500, random seed 2 sampled: -20.04
10x, K=500, random seed 3 sampled: -20.04
10x, K=500, random seed 4 sampled: -20.03
10x, K=500, random seed 5 sampled: -20.04
    MODE     K                      COST          TIME
 Sampled    1x           -19.648 +- 0.022       0:00:00
  Greedy    1x             -19.992 +- 0.0       0:00:00
 Sampled    2x           -19.882 +- 0.007       0:00:00
 Sampled    5x           -19.994 +- 0.005       0:00:00
  Random   10x            -18.36 +- 0.029       0:00:01
 Sampled   10x            -20.04 +- 0.005       0:00:01
```

addCode

addText

```
CUDA device not found. Running on CPU.
n_problems: 256
problem_dim: 50
embed_dim: 16
training:
  method: es
  reward: min_cost
  n_epochs: 200
  lr: 0.001
  batch_size: 500
  ppo_epochs: 10
  trace_decay: 0.9
  eps_clip: 0.25
  gamma: 0.9
  weight_decay: 0.01
  momentum: 0.9
  stddev: 0.05
  population: 16
  milestones:
  - 0.9
   optimizer: sgd
sa:
  init_temp: 1.0
  stop_temp: 0.1
  outer_steps: 100
  inner_steps: 1
  alpha: 0.9772372209558107
problem: knapsack
capacity: 12.5
device: cpu
model_path: null
results_path: results
data_path: datasets
seed: 42

Training loss: -17.3060: 100% 200/200 [17:11<00:00,  5.16s/it]
CUDA device not found. Running on cpu.
Loaded model at  models/knapsack50-es.pt
1x, K=50, random seed 1 sampled: -15.63
1x, K=50, random seed 2 sampled: -15.52
1x, K=50, random seed 3 sampled: -15.61
1x, K=50, random seed 4 sampled: -15.58
1x, K=50, random seed 5 sampled: -15.58
2x, K=100, random seed 1 sampled: -17.31
2x, K=100, random seed 2 sampled: -17.21
2x, K=100, random seed 3 sampled: -17.33
2x, K=100, random seed 4 sampled: -17.26
2x, K=100, random seed 5 sampled: -17.30
5x, K=250, random seed 1 sampled: -18.48
5x, K=250, random seed 2 sampled: -18.46
5x, K=250, random seed 3 sampled: -18.52
5x, K=250, random seed 4 sampled: -18.44
5x, K=250, random seed 5 sampled: -18.51
10x, K=500, random seed 1 sampled: -18.92
```

```
10x, K=500, random seed 2 sampled: -18.93
10x, K=500, random seed 3 sampled: -19.01
10x, K=500, random seed 4 sampled: -18.91
10x, K=500, random seed 5 sampled: -18.99
    MODE     K                      COST          TIME
 Sampled    1x          -15.583 +- 0.039       0:00:00
  Greedy    1x            -19.952 +- 0.0       0:00:00
 Sampled    2x          -17.282 +- 0.043       0:00:00
 Sampled    5x          -18.482 +- 0.029       0:00:00
  Random   10x           -18.36 +- 0.029       0:00:01
 Sampled   10x          -18.951 +- 0.039       0:00:01
```

```
============================================================
```

```
n_problems: 256
problem_dim: 50
embed_dim: 16
training:
  method: es
  reward: min_cost
  n_epochs: 200
  lr: 0.001
  batch_size: 500
  ppo_epochs: 10
  trace_decay: 0.9
  eps_clip: 0.25
  gamma: 0.9
  weight_decay: 0.01
  momentum: 0.9
  stddev: 0.05
  population: 16
  milestones:
  - 0.9
sa:
  init_temp: 1.0
  stop_temp: 0.1
  outer_steps: 100
  inner_steps: 1
  alpha: 0.9772372209558107
problem: knapsack
capacity: 12.5
device: cuda:0
model_path: null
results_path: results
data_path: datasets
seed: 42

Training loss: -17.1223: 100% 200/200 [08:41<00:00,  2.61s/it]
```

```
!python scripts/eval.py +experiment=knapsack_es
```

```
Loaded model at  models/knapsack50-es.pt
1x, K=50, random seed 1 sampled: -15.84
1x, K=50, random seed 2 sampled: -15.79
1x, K=50, random seed 3 sampled: -15.81
1x, K=50, random seed 4 sampled: -15.91
1x, K=50, random seed 5 sampled: -15.74
2x, K=100, random seed 1 sampled: -17.14
2x, K=100, random seed 2 sampled: -17.14
2x, K=100, random seed 3 sampled: -17.09
2x, K=100, random seed 4 sampled: -17.27
2x, K=100, random seed 5 sampled: -17.05
5x, K=250, random seed 1 sampled: -18.20
5x, K=250, random seed 2 sampled: -18.24
5x, K=250, random seed 3 sampled: -18.23
5x, K=250, random seed 4 sampled: -18.22
5x, K=250, random seed 5 sampled: -18.19
10x, K=500, random seed 1 sampled: -18.70
10x, K=500, random seed 2 sampled: -18.76
10x, K=500, random seed 3 sampled: -18.67
10x, K=500, random seed 4 sampled: -18.75
10x, K=500, random seed 5 sampled: -18.73
```

```
!python scripts/print_results.py +experiment=knapsack_es
```

```
MODE     K                      COST          TIME
 Sampled    1x           -15.82 +- 0.056       0:00:00
  Greedy    1x            -17.626 +- 0.0       0:00:00
 Sampled    2x          -17.139 +- 0.075       0:00:00
 Sampled    5x          -18.217 +- 0.019       0:00:00
  Random   10x          -18.373 +- 0.024       0:00:00
 Sampled   10x          -18.725 +- 0.032       0:00:00
```

Adam

```
Epoch,TrainLoss,MeanObjective,BestObjective,FitnessStd,Stddev,LR,TimeSec,O
ptimizer
0,-16.18498194217682,-16.18498194217682,-16.524484634399414,0.151997701877
91475,0.05,0.001,5.653745651245117,adam
1,-16.256075501441956,-16.256075501441956,-16.4057559967041,0.101597013469
76557,0.05,0.001,4.564181804656982,adam
2,-16.285494208335876,-16.285494208335876,-16.44148826599121,0.09742203258
360943,0.05,0.001,5.672523021697998,adam
3,-16.255074381828308,-16.255074381828308,-16.512062072753906,0.1135211983
8021996,0.05,0.001,4.564069747924805,adam
4,-16.260546684265137,-16.260546684265137,-16.41657257080078,0.08823282999
867799,0.05,0.001,4.558197021484375,adam
5,-16.120701372623444,-16.120701372623444,-16.37197494506836,0.12126487914
429908,0.05,0.001,6.193694591522217,adam
Epoch,TrainLoss,MeanObjective,BestObjective,FitnessStd,Stddev,LR,TimeSec,O
ptimizer
0,-16.18498194217682,-16.18498194217682,-16.524484634399414,0.151997701877
91475,0.05,0.001,5.025604963302612,adam
1,-16.256075501441956,-16.256075501441956,-16.4057559967041,0.101597013469
76557,0.05,0.001,4.669453382492065,adam
2,-16.285494208335876,-16.285494208335876,-16.44148826599121,0.09742203258
360943,0.05,0.001,5.674647092819214,adam
3,-16.255074381828308,-16.255074381828308,-16.512062072753906,0.1135211983
8021996,0.05,0.001,4.518259286880493,adam
4,-16.260546684265137,-16.260546684265137,-16.41657257080078,0.08823282999
867799,0.05,0.001,4.9657142162323,adam
5,-16.120701372623444,-16.120701372623444,-16.37197494506836,0.12126487914
429908,0.05,0.001,5.383741855621338,adam
6,-15.970639944076538,-15.970639944076538,-16.11282730102539,0.07458106184
63844,0.05,0.001,4.812450408935547,adam
7,-16.195680499076843,-16.195680499076843,-16.36785888671875,0.09141319620
486302,0.05,0.001,5.8630053997039795,adam
8,-16.268557369709015,-16.268557369709015,-16.493274688720703,0.1313750294
227535,0.05,0.001,4.484166860580444,adam
9,-16.329777359962463,-16.329777359962463,-16.515989303588867,0.1382089044
0556227,0.05,0.001,4.675321340560913,adam
10,-16.163634538650513,-16.163634538650513,-16.37515640258789,0.1032349119
2939135,0.05,0.001,5.774065017700195,adam
```

```
11,-16.346555829048157,-16.346555829048157,-16.55274772644043,0.1109855056
5912259,0.05,0.001,4.768383741378784,adam
12,-16.206111669540405,-16.206111669540405,-16.3868408203125,0.06374033916
580533,0.05,0.001,5.933409690856934,adam
13,-16.13392060995102,-16.13392060995102,-16.272233963012695,0.07822409162
640413,0.05,0.001,4.693277835845947,adam
14,-16.12207978963852,-16.12207978963852,-16.327377319335938,0.13100442489
832703,0.05,0.001,4.904901504516602,adam
15,-16.189284443855286,-16.189284443855286,-16.424686431884766,0.103681433
7245931,0.05,0.001,5.879827976226807,adam
16,-16.239660143852234,-16.239660143852234,-16.42156410217285,0.1079572108
416925,0.05,0.001,4.603308439254761,adam
17,-16.17674171924591,-16.17674171924591,-16.309926986694336,0.07753304048
935757,0.05,0.001,5.941786527633667,adam
18,-16.252044320106506,-16.252044320106506,-16.438692092895508,0.087912251
12963959,0.05,0.001,4.969882011413574,adam
Epoch,TrainLoss,MeanObjective,BestObjective,FitnessStd,Stddev,LR,TimeSec,O
ptimizer
0,-16.18498194217682,-16.18498194217682,-16.524484634399414,0.151997701877
91475,0.05,0.001,4.799519300460815,adam
1,-16.256075501441956,-16.256075501441956,-16.4057559967041,0.101597013469
76557,0.05,0.001,5.8661699295043945,adam
2,-16.285494208335876,-16.285494208335876,-16.44148826599121,0.09742203258
360943,0.05,0.001,4.711474895477295,adam
3,-16.255074381828308,-16.255074381828308,-16.512062072753906,0.1135211983
8021996,0.05,0.001,5.470724821090698,adam
4,-16.260546684265137,-16.260546684265137,-16.41657257080078,0.08823282999
867799,0.05,0.001,5.790152311325073,adam
5,-16.12070137262344,-16.12070137262344,-16.37197494506836,0.12126487914
429908,0.05,0.001,4.741167783737183,adam
6,-15.970639944076538,-15.970639944076538,-16.11282730102539,0.07458106184
63844,0.05,0.001,5.686512470245361,adam
7,-16.195680499076843,-16.195680499076843,-16.36785888671875,0.09141319620
486302,0.05,0.001,4.659755706787109,adam
8,-16.268557369709015,-16.268557369709015,-16.493274688720703,0.1313750294
227535,0.05,0.001,5.284864187240601,adam
9,-16.329777359962463,-16.329777359962463,-16.515989303588867,0.1382089044
0556227,0.05,0.0001,5.206429958343506,adam
Epoch,TrainLoss,MeanObjective,BestObjective,FitnessStd,Stddev,LR,TimeSec,O
ptimizer
0,-16.18498194217682,-16.18498194217682,-16.524484634399414,0.151997701877
91475,0.05,0.001,4.56402850151062,adam
1,-16.256075501441956,-16.256075501441956,-16.4057559967041,0.101597013469
76557,0.05,0.001,5.8041393756866455,adam
2,-16.285494208335876,-16.285494208335876,-16.44148826599121,0.09742203258
360943,0.05,0.001,4.553298234939575,adam
```

```
3,-16.255074381828308,-16.255074381828308,-16.512062072753906,0.1135211983
8021996,0.05,0.001,4.706469774246216,adam
4,-16.260546684265137,-16.260546684265137,-16.41657257080078,0.08823282999
867799,0.05,0.001,5.512771368026733,adam
5,-16.120701372623444,-16.120701372623444,-16.37197494506836,0.12126487914
429908,0.05,0.001,4.765152215957642,adam
6,-15.970639944076538,-15.970639944076538,-16.11282730102539,0.07458106184
63844,0.05,0.001,5.724714994430542,adam
7,-16.195680499076843,-16.195680499076843,-16.36785888671875,0.09141319620
486302,0.05,0.001,4.748662710189819,adam
8,-16.268557369709015,-16.268557369709015,-16.493274688720703,0.1313750294
227535,0.05,0.001,4.839505195617676,adam
9,-16.329777359962463,-16.329777359962463,-16.515989303588867,0.1382089044
0556227,0.05,0.001,5.770913124084473,adam
10,-16.163634538650513,-16.163634538650513,-16.37515640258789,0.1032349119
2939135,0.05,0.001,4.747559547424316,adam
11,-16.346555829048157,-16.346555829048157,-16.55274772644043,0.1109855056
5912259,0.05,0.001,5.913316249847412,adam
12,-16.206111669540405,-16.206111669540405,-16.3868408203125,0.06374033916
580533,0.05,0.001,4.671817779541016,adam
13,-16.13392060995102,-16.13392060995102,-16.272233963012695,0.07822409162
640413,0.05,0.001,4.703480005264282,adam
14,-16.12207978963852,-16.12207978963852,-16.327377319335938,0.13100442489
832703,0.05,0.001,5.698662519454956,adam
15,-16.189284443855286,-16.189284443855286,-16.424686431884766,0.103681433
7245931,0.05,0.001,4.547842502593994,adam
16,-16.239660143852234,-16.239660143852234,-16.42156410217285,0.1079572108
416925,0.05,0.001,5.447943925857544,adam
17,-16.17674171924591,-16.17674171924591,-16.309926986694336,0.07753304048
935757,0.05,0.001,5.151700258255005,adam
18,-16.252044320106506,-16.252044320106506,-16.438692092895508,0.087912251
12963959,0.05,0.001,4.659070014953613,adam
19,-16.240203022956848,-16.240203022956848,-16.342931747436523,0.075533753
43291438,0.05,0.001,5.759335279464722,adam
20,-16.310789704322815,-16.310789704322815,-16.49099349975586,0.1025946924
7382023,0.05,0.001,4.624902009963989,adam
21,-16.251933336257935,-16.251933336257935,-16.421606063842773,0.099958656
5573008,0.05,0.001,5.030031442642212,adam
22,-16.17955768108368,-16.17955768108368,-16.426294326782227,0.13527179998
8706,0.05,0.001,5.5935890674591064,adam
23,-16.363125205039978,-16.363125205039978,-16.530147552490234,0.092191447
65399424,0.05,0.001,4.668835401535034,adam
24,-16.369858503341675,-16.369858503341675,-16.64878273010254,0.1551080443
4104883,0.05,0.001,5.874235391616821,adam
25,-16.204057335853577,-16.204057335853577,-16.403728485107422,0.106276351
68945612,0.05,0.001,4.705819606781006,adam
```

```
26,-16.222071409225464,-16.222071409225464,-16.35526466369629,0.0910808157
2636711,0.05,0.001,4.99056339263916,adam
27,-16.346804976463318,-16.346804976463318,-16.472932815551758,0.089700439
65508552,0.05,0.001,5.580258846282959,adam
28,-16.34227192401886,-16.34227192401886,-16.516860961914062,0.08801062729
34508,0.05,0.001,4.9248573780059814,adam
29,-16.333508014678955,-16.333508014678955,-16.472854614257812,0.067412597
9053576,0.05,0.001,5.743383169174194,adam
30,-16.27554428577423,-16.27554428577423,-16.438793182373047,0.09893283915
251495,0.05,0.001,4.718575954437256,adam
31,-16.4590402841568,-16.4590402841568,-16.694578170776367,0.1203073453673
1302,0.05,0.001,4.71372652053833,adam
32,-16.02877575159073,-16.02877575159073,-16.22049331665039,0.105590989820
05408,0.05,0.001,5.800796747207642,adam
33,-16.282168090343475,-16.282168090343475,-16.424541473388672,0.107889414
59663746,0.05,0.001,4.6900434494018555,adam
34,-16.371792674064636,-16.371792674064636,-16.600189208984375,0.154841866
136921,0.05,0.001,5.975301265716553,adam
35,-16.18936800956726,-16.18936800956726,-16.384809494018555,0.11070370236
80792,0.05,0.001,4.647493124008179,adam
36,-16.340070843696594,-16.340070843696594,-16.502891540527344,0.106118536
43906048,0.05,0.001,4.7243664264678955,adam
37,-16.328999161720276,-16.328999161720276,-16.436447143554688,0.065452937
74561473,0.05,0.001,5.684731721878052,adam
38,-16.291072368621826,-16.291072368621826,-16.425350189208984,0.072906609
16608423,0.05,0.001,4.590228319168091,adam
39,-16.38739001750946,-16.38739001750946,-16.542680740356445,0.08449752265
353018,0.05,0.001,5.359099864959717,adam
40,-16.201760351657867,-16.201760351657867,-16.507896423339844,0.121312569
68154573,0.05,0.001,5.018823146820068,adam
41,-16.540117740631104,-16.540117740631104,-16.697214126586914,0.082988366
16393122,0.05,0.001,4.534758806228638,adam
42,-16.2737774848938,-16.2737774848938,-16.494625091552734,0.1325576179236
8628,0.05,0.001,5.743115425109863,adam
43,-16.219992637634277,-16.219992637634277,-16.45648765563965,0.1072990066
6928856,0.05,0.001,4.574088096618652,adam
44,-16.35014808177948,-16.35014808177948,-16.565135955810547,0.11282740119
663517,0.05,0.001,5.030325651168823,adam
45,-16.45767307281494,-16.45767307281494,-16.634286880493164,0.09924173367
589086,0.05,0.001,5.337807893753052,adam
46,-16.277229487895966,-16.277229487895966,-16.60009765625,0.1572852464464
3102,0.05,0.001,4.676599740982056,adam
47,-16.48584735393524,-16.48584735393524,-16.663360595703125,0.07066824378
713091,0.05,0.001,5.639172554016113,adam
48,-16.395389556884766,-16.395389556884766,-16.566478729248047,0.087492451
4704071,0.05,0.001,4.621110916137695,adam
```

```
49,-16.366149306297302,-16.366149306297302,-16.545082092285156,0.100765717
23760222,0.05,0.001,4.705960988998413,adam
50,-16.522316336631775,-16.522316336631775,-16.72017478942871,0.0988524269
2765334,0.05,0.001,5.549086332321167,adam
51,-16.422914624214172,-16.422914624214172,-16.58627700805664,0.1131429394
662634,0.05,0.001,4.785291194915771 5,adam
52,-16.33411192893982,-16.33411192893982,-16.526912689208984,0.10031208569
233253,0.05,0.001,5.702181100845337,adam
53,-16.189358711242676,-16.189358711242676,-16.35623550415039,0.1107880148
6286276,0.05,0.001,4.570425271987915,adam
54,-16.219815492630005,-16.219815492630005,-16.389942169189453,0.099731436
35236707,0.05,0.001,4.680577516555786,adam
55,-16.265480995178223,-16.265480995178223,-16.383909225463867,0.075426575
92473381,0.05,0.001,5.629890441894531,adam
56,-16.374181389808655,-16.374181389808655,-16.676584243774414,0.110903607
35201872,0.05,0.001,4.564971446990967,adam
57,-16.506548523902893,-16.506548523902893,-16.63001251220703,0.0739726657
6445188,0.05,0.001,5.7296154499053955,adam
58,-16.408735990524292,-16.408735990524292,-16.594837188720703,0.103014296
46701382,0.05,0.001,4.692660570144653,adam
59,-16.577501893043518,-16.577501893043518,-16.695606231689453,0.069480571
95183893,0.05,0.001,4.667513847351074,adam
60,-16.299964547157288,-16.299964547157288,-16.4371395111084,0.06779803047
995754,0.05,0.001,5.668941497802734,adam
61,-16.56865417957306,-16.56865417957306,-16.80153465270996,0.096356308424
61197,0.05,0.001,4.55576491355896,adam
62,-16.477852940559387,-16.477852940559387,-16.666616439819336,0.082120600
7993284,0.05,0.001,5.44260048866272,adam
63,-16.146251499652863,-16.146251499652863,-16.380779266357422,0.086478164
12490386,0.05,0.001,4.8093414306640625,adam
64,-16.393381237983704,-16.393381237983704,-16.554256439208984,0.092611975
33032413,0.05,0.001,4.55389928817749,adam
65,-16.403117775917053,-16.403117775917053,-16.60455322265625,0.1173102524
7516838,0.05,0.001,5.604480028152466,adam
66,-16.44402253627777,-16.44402253627777,-16.605510711669922,0.11062869930
10479,0.05,0.001,4.678059101104736,adam
67,-16.42086613178253,-16.42086613178253,-16.545942306518555,0.09081970850
392926,0.05,0.001,5.27286171913147,adam
68,-16.516037464141846,-16.516037464141846,-16.683975219726562,0.109666688
8113621,0.05,0.001,5.040574550628662,adam
69,-16.45871603488922,-16.45871603488922,-16.552576065063477,0.05342556849
557773,0.05,0.001,4.610142469406128,adam
70,-16.251062512397766,-16.251062512397766,-16.35350799560547,0.0741242393
306172,0.05,0.001,5.567379951477051,adam
71,-16.385658264160156,-16.385658264160156,-16.59296989440918,0.0996415122
5956976,0.05,0.001,4.641633033752441,adam
```

```
72,-16.52800178527832,-16.52800178527832,-16.69015121459961,0.084185990940
61471,0.05,0.001,4.861754417419434,adam
73,-16.426648378372192,-16.426648378372192,-16.598773956298828,0.105095765
66912451,0.05,0.001,5.2969701290130615,adam
74,-16.388943314552307,-16.388943314552307,-16.534574508666992,0.071144900
20474772,0.05,0.001,4.883227586746216,adam
75,-16.446762323379517,-16.446762323379517,-16.60783576965332,0.0772363082
2592233,0.05,0.001,5.701089143753052,adam
76,-16.32727873325348,-16.32727873325348,-16.5349178314209,0.1025800877973
5622,0.05,0.001,4.667498350143433,adam
77,-16.5622900724411,-16.5622900724411,-16.674468994140625,0.0990553568134
0882,0.05,0.001,4.782723665237427,adam
78,-16.43936800956726,-16.43936800956726,-16.62673568725586,0.094519539984
47206,0.05,0.001,5.408617734909058,adam
79,-16.542161345481873,-16.542161345481873,-16.766576766967773,0.113661993
80568255,0.05,0.001,4.598661184310913,adam
80,-16.601332664489746,-16.601332664489746,-16.790401458740234,0.109865672
89138358,0.05,0.001,5.48160719871521,adam
81,-16.418055415153503,-16.418055415153503,-16.567304611206055,0.072393884
40720058,0.05,0.001,4.700149059295654,adam
82,-16.351449847221375,-16.351449847221375,-16.46137046813965,0.0726475428
2562855,0.05,0.001,4.618928909301758,adam
83,-16.567901015281677,-16.567901015281677,-16.814035415649414,0.127816531
70038115,0.05,0.001,5.457213878631592,adam
84,-16.5019748210907,-16.5019748210907,-16.6550350189209,0.102853444619890
46,0.05,0.001,4.579874753952026,adam
85,-16.6639746427536,-16.6639746427536,-16.834802627563477,0.1148831010906
663,0.05,0.001,5.281915903091431,adam
86,-16.318055868148804,-16.318055868148804,-16.45738983154297,0.0804442584
3241643,0.05,0.001,4.875282287597656,adam
87,-16.603021502494812,-16.603021502494812,-16.750812530517578,0.083128310
89599437,0.05,0.001,4.603128910064697,adam
88,-16.388003706932068,-16.388003706932068,-16.500009536743164,0.075311408
88759011,0.05,0.001,5.592041969299316,adam
89,-16.436198949813843,-16.436198949813843,-16.57176399230957,0.0756944173
7744544,0.05,0.001,4.578222036361694,adam
90,-16.541908740997314,-16.541908740997314,-16.71146011352539,0.1197254830
4209538,0.05,0.001,5.118138790130615,adam
91,-16.40181529521942,-16.40181529521942,-16.567899703979492,0.10878487728
355252,0.05,0.001,5.031102180480957,adam
92,-16.43617272377014,-16.43617272377014,-16.6264705657959,0.0929178191150
9933,0.05,0.001,4.463229179382324,adam
93,-16.368139266967773,-16.368139266967773,-16.571706771850586,0.072323040
92888156,0.05,0.001,5.688649892807007,adam
94,-16.579262852668762,-16.579262852668762,-16.766063690185547,0.078031686
55430062,0.05,0.001,4.758687257766724,adam
```

```
95,-16.44470775127411,-16.44470775127411,-16.615161895751953,0.11792291452
443811,0.05,0.001,4.768305540084839,adam
96,-16.61015009880066,-16.61015009880066,-16.824420928955078,0.09285940190
766981,0.05,0.001,5.352679967880249,adam
97,-16.45063829421997,-16.45063829421997,-16.7053165435791,0.0895530052176
9631,0.05,0.001,4.491492986679077,adam
98,-16.575859904289246,-16.575859904289246,-16.81441307067871,0.0932650301
2801332,0.05,0.001,5.719056844711304,adam
99,-16.532267093658447,-16.532267093658447,-16.764158248901367,0.123109279
45508585,0.05,0.001,4.765859127044678,adam
100,-16.4709575176239,-16.4709575176239,-16.631481170654297,0.086681071121
93785,0.05,0.001,4.76694393157959,adam
101,-16.551077961921692,-16.551077961921692,-16.7276611328125,0.1113035280
2643105,0.05,0.001,5.638969659805298,adam
102,-16.554206490516663,-16.554206490516663,-16.735754013061523,0.12320726
944768082,0.05,0.001,4.756291627883911,adam
103,-16.68385088443756,-16.68385088443756,-16.88739585876465,0.08661081312
716004,0.05,0.001,5.8302154541015625,adam
104,-16.697690844535828,-16.697690844535828,-16.85503387451172,0.092195238
78757188,0.05,0.001,4.659222364425659,adam
105,-16.504804730415344,-16.504804730415344,-16.75423240661621,0.107731114
54773904,0.05,0.001,4.788406133651733,adam
106,-16.57432460784912,-16.57432460784912,-16.907886505126953,0.1028957218
7576301,0.05,0.001,5.778153657913208,adam
107,-16.464864134788513,-16.464864134788513,-16.644189834594727,0.09463993
070104261,0.05,0.001,4.644549131393433,adam
108,-16.69634509086609,-16.69634509086609,-16.919591903686523,0.1180033282
3267838,0.05,0.001,5.741709470748901,adam
109,-16.593551874160767,-16.593551874160767,-16.81084632873535,0.090408580
21952428,0.05,0.001,4.556060314178467,adam
110,-16.425272703170776,-16.425272703170776,-16.633153915405273,0.09051585
565390452,0.05,0.001,4.519859552383423,adam
111,-16.629273295402527,-16.629273295402527,-16.828304290771484,0.10456823
787957768,0.05,0.001,5.795323133468628,adam
112,-16.658271551132202,-16.658271551132202,-16.796964645385742,0.07245538
862765741,0.05,0.001,4.68433403968811,adam
113,-16.478517651557922,-16.478517651557922,-16.672191619873047,0.09284114
596773636,0.05,0.001,5.219599485397339,adam
114,-16.600183606147766,-16.600183606147766,-16.873178482055664,0.10033062
326765307,0.05,0.001,5.166250944137573,adam
115,-16.455925583839417,-16.455925583839417,-16.631065368652344,0.09822529
50408307,0.05,0.001,4.696272611618042,adam
116,-16.742892026901245,-16.742892026901245,-16.959394454956055,0.10710706
915853253,0.05,0.001,5.624223947525024,adam
117,-16.513672828674316,-16.513672828674316,-16.673507690429688,0.09318533
929926237,0.05,0.001,4.7572386264801025,adam
```

```
118,-16.71812880039215,-16.71812880039215,-17.02785873413086,0.12819111473
608477,0.05,0.001,4.771035432815552,adam
119,-16.54331624507904,-16.54331624507904,-16.7232608795166,0.101279051513
2211,0.05,0.001,5.58979058265686,adam
120,-16.49768602848053,-16.49768602848053,-16.622591018676758,0.0736411175
8209785,0.05,0.001,4.743229389190674,adam
121,-16.544737219810486,-16.544737219810486,-16.67459487915039,0.071034025
34685135,0.05,0.001,5.785691499710083,adam
122,-16.47884976863861,-16.47884976863861,-16.70183753967285,0.10195760027
584072,0.05,0.001,4.648059368133545,adam
123,-16.757955312728882,-16.757955312728882,-16.894561767578125,0.07101783
025782345,0.05,0.001,4.98081636428833,adam
124,-16.524723172187805,-16.524723172187805,-16.660999298095703,0.09129805
047877829,0.05,0.001,5.6687633991241455,adam
125,-16.432917714118958,-16.432917714118958,-16.66466522216797,0.095053532
7450591,0.05,0.001,4.5451977252960205,adam
126,-16.64413845539093,-16.64413845539093,-16.813194274902344,0.0956868436
3430117,0.05,0.001,5.564086437225342,adam
127,-16.424471616744995,-16.424471616744995,-16.528318405151367,0.07676843
346939684,0.05,0.001,4.881432056427002,adam
128,-16.675025939941406,-16.675025939941406,-16.844680786132812,0.09659860
659045126,0.05,0.001,4.548110485076904,adam
129,-16.496816158294678,-16.496816158294678,-16.62008285522461,0.081138572
97953622,0.05,0.001,5.707531213760376,adam
130,-16.603671312332153,-16.603671312332153,-16.85232162475586,0.113084826
14360116,0.05,0.001,4.525751829147339,adam
131,-16.695385932922363,-16.695385932922363,-16.802902221679688,0.06811646
606634254,0.05,0.001,5.005541563034058,adam
132,-16.58085346221924,-16.58085346221924,-16.75770378112793,0.07498346069
933347,0.05,0.001,5.316920280456543,adam
133,-16.553809762001038,-16.553809762001038,-16.79517364501953,0.102546262
46662021,0.05,0.001,4.608912467956543,adam
134,-16.541006445884705,-16.541006445884705,-16.805850982666016,0.12671703
754212627,0.05,0.001,5.731280088424683,adam
135,-16.894306898117065,-16.894306898117065,-17.027515411376953,0.08601082
834337331,0.05,0.001,4.672492980957031,adam
136,-16.71357560157776,-16.71357560157776,-16.854631423950195,0.0857132121
0017983,0.05,0.001,4.623947381973267,adam
137,-16.617128252983093,-16.617128252983093,-16.815895080566406,0.09094285
440169848,0.05,0.001,5.6728270053863525,adam
138,-16.55263841152191,-16.55263841152191,-16.68360710144043,0.08721353648
129378,0.05,0.001,4.67073392868042,adam
139,-16.56215012073517,-16.56215012073517,-16.805072784423828,0.1166530461
0085109,0.05,0.001,5.436846017837524,adam
140,-16.601016521453857,-16.601016521453857,-16.787792205810547,0.10257019
502627768,0.05,0.001,4.742130517959595,adam
```

```
141,-16.523711323738098,-16.523711323738098,-16.675395965576172,0.09856161
88854859,0.05,0.001,4.6412599086761475,adam
142,-16.476887106895447,-16.476887106895447,-16.58083724975586,0.062931459
64703414,0.05,0.001,5.86664605140686,adam
143,-16.461184859275818,-16.461184859275818,-16.63258934020996,0.090794243
55260311,0.05,0.001,4.608166933059692,adam
144,-16.51167392730713,-16.51167392730713,-16.630157470703125,0.0750761449
5027042,0.05,0.001,5.507689476013184,adam
145,-16.492581725120544,-16.492581725120544,-16.66958999633789,0.091875247
74816579,0.05,0.001,5.088994026184082,adam
146,-16.60829985141754,-16.60829985141754,-16.75432014465332,0.07733836317
386612,0.05,0.001,4.678433179855347,adam
147,-16.746793150901794,-16.746793150901794,-16.89519691467285,0.094911035
9063769,0.05,0.001,5.852548599243164,adam
148,-16.635218501091003,-16.635218501091003,-16.763362884521484,0.08700603
088736751,0.05,0.001,4.530933618545532,adam
149,-16.915908932685852,-16.915908932685852,-17.11772346496582,0.096000341
78774093,0.05,0.001,4.886306285858154,adam
150,-16.678839683532715,-16.678839683532715,-16.916332244873047,0.10645643
043703899,0.05,0.001,5.459656476974487,adam
151,-16.587029099464417,-16.587029099464417,-16.75992774963379,0.096810400
06196949,0.05,0.001,4.496835470199585,adam
152,-16.782179355621338,-16.782179355621338,-16.978166580200195,0.11344641
683904055,0.05,0.001,5.70348048210144,adam
153,-16.658564925193787,-16.658564925193787,-16.824064254760742,0.08223613
895923128,0.05,0.001,4.631975889205933,adam
154,-16.57713210582733,-16.57713210582733,-16.8129825592041,0.099482904608
61656,0.05,0.001,4.6404523849487305,adam
155,-16.740009784698486,-16.740009784698486,-16.904541015625,0.08099746810
914388,0.05,0.001,5.646697044372559,adam
156,-16.77000880241394,-16.77000880241394,-16.88362693786621,0.07995501253
181363,0.05,0.001,4.623548984527588,adam
157,-16.65351390838623,-16.65351390838623,-16.856842041015625,0.1010193708
6159092,0.05,0.001,5.16915225982666,adam
158,-16.622814655303955,-16.622814655303955,-16.952531814575195,0.11467573
496601559,0.05,0.001,5.272510528564453,adam
159,-16.81898522377014,-16.81898522377014,-17.015857696533203,0.1182357187
0469951,0.05,0.001,4.761270523071289,adam
160,-16.6873300075531,-16.6873300075531,-16.891647338867188,0.123459351877
81336,0.05,0.001,5.903280258178711,adam
161,-16.644644021987915,-16.644644021987915,-16.836267471313477,0.07979198
978352849,0.05,0.001,4.663240194320679,adam
162,-16.891143679618835,-16.891143679618835,-17.180641174316406,0.11831035
958013329,0.05,0.001,5.388393878936768,adam
163,-16.57557451725006,-16.57557451725006,-16.711679458618164,0.1130164959
2531661,0.05,0.001,5.253819942474365,adam
```

```
164,-16.72234332561493,-16.72234332561493,-16.85266876220703,0.07124785294
65022,0.05,0.001,4.702258586883545,adam
165,-16.82076096534729,-16.82076096534729,-16.9239444732666,0.081737512991
48517,0.05,0.001,5.7083470821380615,adam
166,-16.668750047683716,-16.668750047683716,-16.800424575805664,0.08438965
797303934,0.05,0.001,4.690244674682617,adam
167,-16.51280653476715,-16.51280653476715,-16.674161911010742,0.0959782772
4219796,0.05,0.001,4.85931658744812,adam
168,-16.797722697257996,-16.797722697257996,-16.995849609375,0.11376516680
077461,0.05,0.001,5.476315975189209,adam
169,-16.746333360671997,-16.746333360671997,-16.932205200195312,0.10309518
567769915,0.05,0.001,4.595827341079712,adam
170,-16.845048666000366,-16.845048666000366,-17.041461944580078,0.10443486
000400999,0.05,0.001,5.840099573135376,adam
171,-16.639939665794373,-16.639939665794373,-16.812326431274414,0.08028101
387659048,0.05,0.001,4.7530224323272705,adam
172,-16.849044919013977,-16.849044919013977,-16.98917579650879,0.080403355
30893668,0.05,0.001,4.5644166469573975,adam
173,-16.722793340682983,-16.722793340682983,-16.877212524414062,0.09056907
145916283,0.05,0.001,5.734832763671875,adam
174,-16.724304795265198,-16.724304795265198,-16.903989791870117,0.07470019
96826601,0.05,0.001,4.616621971130371,adam
175,-16.760602474212646,-16.760602474212646,-16.94989776611328,0.103801714
9965757,0.05,0.001,5.8148627281188965,adam
176,-16.708715677261353,-16.708715677261353,-16.883596420288086,0.10711356
39746541,0.05,0.001,4.985568046569824,adam
177,-16.694198846817017,-16.694198846817017,-16.879070281982422,0.09566401
259710659,0.05,0.001,4.7686848640441895,adam
178,-16.665316343307495,-16.665316343307495,-16.840646743774414,0.10164686
272769947,0.05,0.001,5.765648603439331,adam
179,-16.858479261398315,-16.858479261398315,-17.048120498657227,0.09190217
808423716,0.05,0.001,4.73512864112854,adam
180,-16.719796419143677,-16.719796419143677,-16.892230987548828,0.10919441
388722641,0.05,0.0001,5.3343870639801025,adam
181,-17.010663866996765,-17.010663866996765,-17.256961822509766,0.09875537
053069625,0.05,0.0001,4.871769905090332,adam
182,-16.925259113311768,-16.925259113311768,-17.06029510498047,0.054421685
132622875,0.05,0.0001,4.59207010269165,adam
183,-16.78859829902649,-16.78859829902649,-16.87308120727539,0.05003575686
799141,0.05,0.0001,5.663873195648193,adam
184,-16.813520550727844,-16.813520550727844,-16.88772964477539,0.044105291
69828508,0.05,0.0001,4.593423843383789,adam
185,-16.740344166755676,-16.740344166755676,-16.910367965698242,0.10084756
95968389,0.05,0.0001,4.9932639598846436,adam
186,-16.902812242507935,-16.902812242507935,-17.11911392211914,0.099877155
43821844,0.05,0.0001,5.394992828369141,adam
```

```
187,-17.054077863693237,-17.054077863693237,-17.230764389038086,0.09275853811005871,0.05,0.0001,4.6734700202941895,adam
188,-16.652222156524658,-16.652222156524658,-16.79950714111328,0.07102621465425203,0.05,0.0001,5.739916801452637,adam
189,-16.714689016342163,-16.714689016342163,-16.85784912109375,0.09016102132506155,0.05,0.0001,4.574842929840088,adam
190,-16.68325448036194,-16.68325448036194,-16.866125106811523,0.08653615980004357,0.05,0.0001,4.4556403160095215,adam
191,-16.867355227470398,-16.867355227470398,-16.97805404663086,0.0813192461298659,0.05,0.0001,5.566622257232666,adam
192,-16.821833968162537,-16.821833968162537,-16.96762466430664,0.09321523517311418,0.05,0.0001,4.560011148452759,adam
193,-16.769359707832336,-16.769359707832336,-16.941919326782227,0.08965456218743648,0.05,0.0001,5.214680433273315,adam
194,-16.705740690231323,-16.705740690231323,-16.804956436157227,0.06625814860332896,0.05,0.0001,5.287039756774902,adam
195,-16.59251308441162,-16.59251308441162,-16.780763626098633,0.11638851461214521,0.05,0.0001,4.57792329788208,adam
196,-16.641207695007324,-16.641207695007324,-16.785663604736328,0.07895425609947675,0.05,0.0001,5.621512413024902,adam
197,-16.670996069908142,-16.670996069908142,-16.880023956298828,0.10031311988442733,0.05,0.0001,4.455794811248779,adam
198,-16.434308409690857,-16.434308409690857,-16.547687530517578,0.07838981212755614,0.05,0.0001,4.465578079223633,adam
199,-16.776509404182434,-16.776509404182434,-16.968082427978516,0.08412965408049661,0.05,0.0001,5.682391405105591,adam
```

main.py

```python
import os
import random
import time
import csv

import hydra
import numpy as np
import torch
from hydra.core.config_store import ConfigStore
from omegaconf import OmegaConf
from torch.optim import SGD
from torch.optim.lr_scheduler import MultiStepLR
from tqdm import tqdm

from neuralsa.configs import NeuralSAExperiment
from neuralsa.model import (
    BinPackingActor,
    BinPackingCritic,
    KnapsackActor,
    KnapsackCritic,
    TSPActor,
    TSPCritic,
)
from neuralsa.problem import TSP, BinPacking, Knapsack
from neuralsa.sa import sa
from neuralsa.training import EvolutionStrategies
from neuralsa.training.ppo import ppo
from neuralsa.training.replay import Replay

# For reproducibility on GPU
torch.backends.cudnn.deterministic = True

def create_folder(dirname):
    if not os.path.exists(dirname):
        os.makedirs(dirname)
        print(f"Created: {dirname}")

def train_es(actor, problem, init_x, es, cfg, epoch, log_writer):
    start_time = time.time()

    with torch.no_grad():
        es.zero_updates()
        epoch_objectives = []

        for _ in range(es.population):
            es.perturb(antithetic=True)
            results = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            loss = torch.mean(results[cfg.training.reward])
            epoch_objectives.append(loss.item())
            es.collect(loss)
        es.step(reshape_fitness=True)

    mean_obj = np.mean(epoch_objectives)
    std_obj = np.std(epoch_objectives)
    best_obj = np.min(epoch_objectives)
    elapsed = time.time() - start_time

    train_loss = torch.tensor(mean_obj)
    log_writer.writerow({
        "Epoch": epoch,
        "TrainLoss": train_loss.item(),
        "MeanObjective": mean_obj,
        "BestObjective": best_obj,
        "FitnessStd": std_obj,
        "Stddev": cfg.training.stddev,
        "LR": es.optimizer.param_groups[0]['lr'],
        "TimeSec": elapsed,
    })

    return train_loss

def train_ppo(actor, critic, actor_opt, critic_opt, problem, init_x, cfg):
    replay = Replay(cfg.sa.outer_steps * cfg.sa.inner_steps)
    sa(actor, problem, init_x, cfg, replay=replay, baseline=False,
greedy=False)
    ppo(actor, critic, replay, actor_opt, critic_opt, cfg)

cs = ConfigStore.instance()
cs.store(name="base_config", node=NeuralSAExperiment, group="experiment")

@hydra.main(config_path="conf", config_name="config", version_base=None)
def main(cfg: NeuralSAExperiment) -> None:
    if "cuda" in cfg.device and not torch.cuda.is_available():
        cfg.device = "cpu"
        print("CUDA device not found. Running on cpu.")

    alpha = np.log(cfg.sa.stop_temp) - np.log(cfg.sa.init_temp)
    cfg.sa.alpha = np.exp(alpha / cfg.sa.outer_steps).item()
    print(OmegaConf.to_yaml(cfg))

    torch.manual_seed(cfg.seed)
    random.seed(cfg.seed)
    np.random.seed(cfg.seed)

    if cfg.problem == "knapsack":
        problem = Knapsack(cfg.problem_dim, cfg.n_problems, device=cfg.device, params={"capacity": cfg.capacity})
        actor = KnapsackActor(cfg.embed_dim, device=cfg.device)
        critic = KnapsackCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "binpacking":
        problem = BinPacking(cfg.problem_dim, cfg.n_problems, device=cfg.device)
        actor = BinPackingActor(cfg.embed_dim, device=cfg.device)
        critic = BinPackingCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "tsp":
        problem = TSP(cfg.problem_dim, cfg.n_problems, device=cfg.device)
        actor = TSPActor(cfg.embed_dim, device=cfg.device)
        critic = TSPCritic(cfg.embed_dim, device=cfg.device)
    else:
        raise ValueError("Invalid problem name.")

    problem.manual_seed(cfg.seed)

    if cfg.training.method == "ppo":
        actor_opt = torch.optim.Adam(actor.parameters(), lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
        critic_opt = torch.optim.Adam(critic.parameters(), lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
    elif cfg.training.method == "es":
        optimizer = SGD(actor.parameters(), lr=cfg.training.lr, momentum=cfg.training.momentum)
        es = EvolutionStrategies(optimizer, cfg.training.stddev, cfg.training.population)
        milestones = [int(cfg.training.n_epochs * m) for m in cfg.training.milestones]
        scheduler = MultiStepLR(optimizer, milestones=milestones, gamma=0.1)
        log_path = os.path.join(os.getcwd(), "outputs")
        create_folder(log_path)
        log_file = os.path.join(log_path, "es_train_log.csv")
        log_file_handle = open(log_file, mode='w', newline='')
        log_writer = csv.DictWriter(log_file_handle, fieldnames=["Epoch",
"TrainLoss", "MeanObjective", "BestObjective", "FitnessStd", "Stddev",
"LR", "TimeSec"])
        log_writer.writeheader()
    else:
        raise ValueError("Invalid training method.")

    with tqdm(range(cfg.training.n_epochs)) as t:
        for i in t:
            params = problem.generate_params()
            params = {k: v.to(cfg.device) for k, v in params.items()}
            problem.set_params(**params)
            init_x = problem.generate_init_x()
            actor.manual_seed(cfg.seed)

            if cfg.training.method == "ppo":
                train_ppo(actor, critic, actor_opt, critic_opt, problem,
init_x, cfg)
                train_out = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
                train_loss = torch.mean(train_out["min_cost"])
            elif cfg.training.method == "es":
                train_loss = train_es(actor, problem, init_x, es, cfg, i,
log_writer)
                scheduler.step()

            t.set_description(f"Training loss: {train_loss:.4f}")

            path = os.path.join(os.getcwd(), "models")
            name = cfg.problem + str(cfg.problem_dim) + "-" +
cfg.training.method + ".pt"
            create_folder(path)
            torch.save(actor.state_dict(), os.path.join(path, name))

    if cfg.training.method == "es":
        log_file_handle.close()
```

```
if __name__ == "__main__":
    main()



'''
import os
import random
import csv

import hydra
import numpy as np
import torch
from hydra.core.config_store import ConfigStore
from omegaconf import OmegaConf
from torch.optim import SGD
from torch.optim.lr_scheduler import MultiStepLR
from tqdm import tqdm

from neuralsa.configs import NeuralSAExperiment
from neuralsa.model import (
    BinPackingActor,
    BinPackingCritic,
    KnapsackActor,
    KnapsackCritic,
    TSPActor,
    TSPCritic,
)
from neuralsa.problem import TSP, BinPacking, Knapsack
from neuralsa.sa import sa
from neuralsa.training import EvolutionStrategies
from neuralsa.training.ppo import ppo
from neuralsa.training.replay import Replay

# For reproducibility on GPU
torch.backends.cudnn.deterministic = True

def create_folder(dirname):
    if not os.path.exists(dirname):
        os.makedirs(dirname)
        print(f"Created: {dirname}")

def train_es(actor, problem, init_x, es, cfg):
    with torch.no_grad():
        es.zero_updates()
        for _ in range(es.population):
            es.perturb(antithetic=True)

            results = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            loss = torch.mean(results[cfg.training.reward])
            es.collect(loss)

        es.step(reshape_fitness=True)

    return torch.mean(torch.tensor(es.objective))

def train_ppo(actor, critic, actor_opt, critic_opt, problem, init_x, cfg):
    replay = Replay(cfg.sa.outer_steps * cfg.sa.inner_steps)
    sa(actor, problem, init_x, cfg, replay=replay, baseline=False,
greedy=False)
    ppo(actor, critic, replay, actor_opt, critic_opt, cfg)

cs = ConfigStore.instance()
cs.store(name="base_config", node=NeuralSAExperiment, group="experiment")

@hydra.main(config_path="conf", config_name="config", version_base=None)
def main(cfg: NeuralSAExperiment) -> None:
    if "cuda" in cfg.device and not torch.cuda.is_available():
        cfg.device = "cpu"
        print("CUDA device not found. Running on cpu.")

    alpha = np.log(cfg.sa.stop_temp) - np.log(cfg.sa.init_temp)
    cfg.sa.alpha = np.exp(alpha / cfg.sa.outer_steps).item()

    print(OmegaConf.to_yaml(cfg))

    torch.manual_seed(cfg.seed)
    random.seed(cfg.seed)
    np.random.seed(cfg.seed)

    if cfg.problem == "knapsack":
        problem = Knapsack(cfg.problem_dim, cfg.n_problems,
device=cfg.device, params={"capacity": cfg.capacity})
        actor = KnapsackActor(cfg.embed_dim, device=cfg.device)
        critic = KnapsackCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "binpacking":
        problem = BinPacking(cfg.problem_dim, cfg.n_problems,
device=cfg.device)
        actor = BinPackingActor(cfg.embed_dim, device=cfg.device)
        critic = BinPackingCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "tsp":
        problem = TSP(cfg.problem_dim, cfg.n_problems, device=cfg.device)
        actor = TSPActor(cfg.embed_dim, device=cfg.device)
        critic = TSPCritic(cfg.embed_dim, device=cfg.device)
    else:
        raise ValueError("Invalid problem name.")

    problem.manual_seed(cfg.seed)

    if cfg.training.method == "ppo":
        actor_opt = torch.optim.Adam(actor.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
        critic_opt = torch.optim.Adam(critic.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
    elif cfg.training.method == "es":
        optimizer = SGD(actor.parameters(), lr=cfg.training.lr,
momentum=cfg.training.momentum)
        es = EvolutionStrategies(optimizer, cfg.training.stddev,
cfg.training.population)
```

```
        milestones = [int(cfg.training.n_epochs * m) for m in
cfg.training.milestones]
        scheduler = MultiStepLR(optimizer, milestones=milestones,
gamma=0.1)

        # Prepare logging
        log_path = os.path.join(os.getcwd(), "outputs")
        create_folder(log_path)
        log_file = os.path.join(log_path, "es_train_log.csv")
        with open(log_file, mode='w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(["Epoch", "TrainLoss", "MeanObjective"])
    else:
        raise ValueError("Invalid training method.")

    with tqdm(range(cfg.training.n_epochs)) as t:
        for i in t:
            params = problem.generate_params()
            params = {k: v.to(cfg.device) for k, v in params.items()}
            problem.set_params(**params)
            init_x = problem.generate_init_x()
            actor.manual_seed(cfg.seed)

            if cfg.training.method == "ppo":
                train_ppo(actor, critic, actor_opt, critic_opt, problem,
init_x, cfg)
            elif cfg.training.method == "es":
                mean_objective = train_es(actor, problem, init_x, es, cfg)
                scheduler.step()

            train_out = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            train_loss = torch.mean(train_out["min_cost"]).item()

            t.set_description(f"Training loss: {train_loss:.4f}")

            if cfg.training.method == "es":
                with open(log_file, mode='a', newline='') as file:
                    writer = csv.writer(file)
                    writer.writerow([i, train_loss,
mean_objective.item()])

            path = os.path.join(os.getcwd(), "models")
            name = cfg.problem + str(cfg.problem_dim) + "-" +
cfg.training.method + ".pt"
            create_folder(path)
            torch.save(actor.state_dict(), os.path.join(path, name))

if __name__ == "__main__":
    main()
'''


'''
import os
import random

import hydra
import numpy as np
import torch
from hydra.core.config_store import ConfigStore
from omegaconf import OmegaConf
from torch.optim import SGD
from torch.optim.lr_scheduler import MultiStepLR
from tqdm import tqdm

from neuralsa.configs import NeuralSAExperiment
from neuralsa.model import (
    BinPackingActor,
    BinPackingCritic,
    KnapsackActor,
    KnapsackCritic,
    TSPActor,
    TSPCritic,
)
from neuralsa.problem import TSP, BinPacking, Knapsack
from neuralsa.sa import sa
from neuralsa.training import EvolutionStrategies
from neuralsa.training.ppo import ppo
from neuralsa.training.replay import Replay

# For reproducibility on GPU
torch.backends.cudnn.deterministic = True


def create_folder(dirname):
    if not os.path.exists(dirname):
        os.makedirs(dirname)
        print(f"Created: {dirname}")


def train_es(actor, problem, init_x, es, cfg):
    with torch.no_grad():
        es.zero_updates()
        for _ in range(es.population):
            es.perturb(antithetic=True)

            # Run SA and compute the loss
            results = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            loss = torch.mean(results[cfg.training.reward])
            es.collect(loss)

        es.step(reshape_fitness=True)

    return torch.mean(torch.tensor(es.objective))


def train_ppo(actor, critic, actor_opt, critic_opt, problem, init_x, cfg):
    # Create replay to store transitions
    replay = Replay(cfg.sa.outer_steps * cfg.sa.inner_steps)
    # Run SA and collect transitions
    sa(actor, problem, init_x, cfg, replay=replay, baseline=False,
greedy=False)
    # Optimize the policy with PPO
    ppo(actor, critic, replay, actor_opt, critic_opt, cfg)


cs = ConfigStore.instance()
```

```
# Registering the Config class with the name 'config'.
cs.store(name="base_config", node=NeuralSAExperiment, group="experiment")


@hydra.main(config_path="conf", config_name="config", version_base=None)
def main(cfg: NeuralSAExperiment) -> None:
    if "cuda" in cfg.device and not torch.cuda.is_available():
        cfg.device = "cpu"
        print("CUDA device not found. Running on cpu.")

    # Define temperature decay parameter as a function of the number of
steps
    alpha = np.log(cfg.sa.stop_temp) - np.log(cfg.sa.init_temp)
    cfg.sa.alpha = np.exp(alpha / cfg.sa.outer_steps).item()

    print(OmegaConf.to_yaml(cfg))

    # Set seeds
    torch.manual_seed(cfg.seed)
    random.seed(cfg.seed)
    np.random.seed(cfg.seed)

    # Set Problem and Networks
    if cfg.problem == "knapsack":
        problem = Knapsack(
            cfg.problem_dim, cfg.n_problems, device=cfg.device,
params={"capacity": cfg.capacity}
        )
        actor = KnapsackActor(cfg.embed_dim, device=cfg.device)
        critic = KnapsackCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "binpacking":
        problem = BinPacking(cfg.problem_dim, cfg.n_problems,
device=cfg.device)
        actor = BinPackingActor(cfg.embed_dim, device=cfg.device)
        critic = BinPackingCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "tsp":
        problem = TSP(cfg.problem_dim, cfg.n_problems, device=cfg.device)
        actor = TSPActor(cfg.embed_dim, device=cfg.device)
        critic = TSPCritic(cfg.embed_dim, device=cfg.device)
    else:
        raise ValueError("Invalid problem name.")

    # Set problem seed
    problem.manual_seed(cfg.seed)

    # If using PPO, initialize optimisers and replay
    if cfg.training.method == "ppo":
        actor_opt = torch.optim.Adam(
            actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
        )
        critic_opt = torch.optim.Adam(
            critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
        )
    elif cfg.training.method == "es":
        # Optimization specs
        optimizer = SGD(actor.parameters(), lr=cfg.training.lr,
momentum=cfg.training.momentum)
        es = EvolutionStrategies(optimizer, cfg.training.stddev,
cfg.training.population)
        milestones = [int(cfg.training.n_epochs * m) for m in
cfg.training.milestones]
        scheduler = MultiStepLR(optimizer, milestones=milestones,
gamma=0.1)
    else:
        raise ValueError("Invalid training method.")

    with tqdm(range(cfg.training.n_epochs)) as t:
        for i in t:
            # Create random instances
            params = problem.generate_params()
            params = {k: v.to(cfg.device) for k, v in params.items()}
            problem.set_params(**params)
            # Find initial solutions
            init_x = problem.generate_init_x()
            actor.manual_seed(cfg.seed)

            # Training loop
            if cfg.training.method == "ppo":
                train_ppo(actor, critic, actor_opt, critic_opt, problem,
init_x, cfg)
            elif cfg.training.method == "es":
                train_es(actor, problem, init_x, es, cfg)
                scheduler.step()

            # Rerun trained model
            train_out = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            train_loss = torch.mean(train_out["min_cost"])

            t.set_description(f"Training loss: {train_loss:.4f}")

            path = os.path.join(os.getcwd(), "models")
            name = cfg.problem + str(cfg.problem_dim) + "-" +
cfg.training.method + ".pt"
            create_folder(path)
            torch.save(actor.state_dict(), os.path.join(path, name))


if __name__ == "__main__":
    main()
'''
```

configs.py

```
# Copyright (c) 2023 Qualcomm Technologies, Inc.
# All Rights Reserved.


from dataclasses import dataclass, field
from typing import Optional

from omegaconf import MISSING


@dataclass
class TrainingConfig:
    method: str = "ppo"
    reward: str = "immediate"
    n_epochs: int = 1000
    lr: float = 0.0002  # learning rate
    batch_size: int = 1024
    # PPO params
    ppo_epochs: int = 10
    trace_decay: float = 0.9
    eps_clip: float = 0.25
    gamma: float = 0.9
    weight_decay: float = 0.01
    # ES params
    momentum: float = 0.9
    stddev: float = 0.05
    population: int = 16
    milestones: list = field(default_factory=lambda: [0.9])
    #optimizer: str = "adam"

@dataclass
class SAConfig:
    init_temp: float = 1.0
    stop_temp: float = 0.1
    outer_steps: int = 40  # number of steps at which temperature changes
    inner_steps: int = 1  # number of steps at a specific temperature
    alpha: float = MISSING  # defined as a function of init_temp and stop_temp


@dataclass
class NeuralSAExperiment:
    n_problems: int = 256  # number of problems in a batch
    problem_dim: int = 20
    embed_dim: int = 16  # size of hidden layer in the actor network

    training: TrainingConfig = field(default_factory=TrainingConfig)
    sa: SAConfig = field(default_factory=SAConfig)

    problem: str = "knapsack"
    capacity: Optional[float] = field(default=None)
    device: str = "cuda:0"
    model_path: Optional[str] = field(default=None)
    results_path: str = "results"
    data_path: str = "datasets"
    seed: int = 42
```

CODE Modifications

main.py

```
# Copyright (c) 2023 Qualcomm Technologies, Inc.
# All Rights Reserved.
import os
import random
import hydra
import numpy as np
import torch
from hydra.core.config_store import ConfigStore
from omegaconf import OmegaConf
from torch.optim import SGD
from torch.optim.lr_scheduler import MultiStepLR
from tqdm import tqdm
import matplotlib.pyplot as plt

from neuralsa.configs import NeuralSAExperiment
from neuralsa.model import (
    BinPackingActor,
    BinPackingCritic,
    KnapsackActor,
    KnapsackCritic,
    TSPActor,
    TSPCritic,
)
from neuralsa.problem import TSP, BinPacking, Knapsack
from neuralsa.sa import sa
from neuralsa.training import EvolutionStrategies
from neuralsa.training.ppo import ppo
from neuralsa.training.replay import Replay

# For reproducibility on GPU
torch.backends.cudnn.deterministic = True
```

```python
def create_folder(dirname):
    if not os.path.exists(dirname):
        os.makedirs(dirname)
        print(f"Created: {dirname}")

def train_es(actor, problem, init_x, es, cfg):
    with torch.no_grad():
        es.zero_updates()
        for _ in range(es.population):
            es.perturb(antithetic=True)
            # Run SA and compute the loss
            results = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            loss = torch.mean(results[cfg.training.reward])
            es.collect(loss)
        es.step(reshape_fitness=True)

    return torch.mean(torch.tensor(es.objective))

def train_ppo(actor, critic, actor_opt, critic_opt, problem, init_x, cfg):
    # Create replay to store transitions
    replay = Replay(cfg.sa.outer_steps * cfg.sa.inner_steps)
    # Run SA and collect transitions
    sa(actor, problem, init_x, cfg, replay=replay, baseline=False,
greedy=False)
    # Optimize the policy with PPO
    ppo(actor, critic, replay, actor_opt, critic_opt, cfg)

cs = ConfigStore.instance()
# Registering the Config class with the name 'config'.
cs.store(name="base_config", node=NeuralSAExperiment, group="experiment")

@hydra.main(config_path="conf", config_name="config", version_base=None)
def main(cfg: NeuralSAExperiment) -> None:
    if "cuda" in cfg.device and not torch.cuda.is_available():
        cfg.device = "cpu"
        print("CUDA device not found. Running on cpu.")
    # Define temperature decay parameter as a function of the number of
steps
    alpha = np.log(cfg.sa.stop_temp) - np.log(cfg.sa.init_temp)
    cfg.sa.alpha = np.exp(alpha / cfg.sa.outer_steps).item()

    print(OmegaConf.to_yaml(cfg))

    # Set seeds
    torch.manual_seed(cfg.seed)
    random.seed(cfg.seed)
    np.random.seed(cfg.seed)

    # Set Problem and Networks
    if cfg.problem == "knapsack":
        problem = Knapsack(cfg.problem_dim, cfg.n_problems,
device=cfg.device, params={"capacity": cfg.capacity})
        actor = KnapsackActor(cfg.embed_dim, device=cfg.device)
        critic = KnapsackCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "binpacking":
        problem = BinPacking(cfg.problem_dim, cfg.n_problems,
device=cfg.device)
        actor = BinPackingActor(cfg.embed_dim, device=cfg.device)
        critic = BinPackingCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "tsp":
        problem = TSP(cfg.problem_dim, cfg.n_problems, device=cfg.device)
        actor = TSPActor(cfg.embed_dim, device=cfg.device)
        critic = TSPCritic(cfg.embed_dim, device=cfg.device)
    else:
        raise ValueError("Invalid problem name.")

    # Set problem seed
    problem.manual_seed(cfg.seed)

    # Initialize optimizers
    if cfg.training.method == "ppo":
        if cfg.training.optimizer == "adam":
            actor_opt = torch.optim.Adam(actor.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
            critic_opt = torch.optim.Adam(critic.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
```

```
        elif cfg.training.optimizer == "adamw":
            actor_opt = torch.optim.AdamW(actor.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
            critic_opt = torch.optim.AdamW(critic.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
        elif cfg.training.optimizer == "nadam":
            actor_opt = torch.optim.NAdam(actor.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
            critic_opt = torch.optim.NAdam(critic.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
        elif cfg.training.optimizer == "adabelief":
            from adabelief_pytorch import AdaBelief
            actor_opt = AdaBelief(actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay)
            critic_opt = AdaBelief(critic.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
        elif cfg.training.optimizer == "adadelta":
            actor_opt = torch.optim.Adadelta(actor.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
            critic_opt = torch.optim.Adadelta(critic.parameters(),
lr=cfg.training.lr, weight_decay=cfg.training.weight_decay)
        else:
            raise ValueError(f"Unsupported optimizer:
{cfg.training.optimizer}")
    elif cfg.training.method == "es":
        # Optimization specs
        optimizer = SGD(actor.parameters(), lr=cfg.training.lr,
momentum=cfg.training.momentum)
        es = EvolutionStrategies(optimizer, cfg.training.stddev,
cfg.training.population)
        milestones = [int(cfg.training.n_epochs * m) for m in
cfg.training.milestones]
        scheduler = MultiStepLR(optimizer, milestones=milestones,
gamma=0.1)
    else:
        raise ValueError("Invalid training method.")

    # Initialize list to store losses for plotting
    epoch_losses = []
    with tqdm(range(cfg.training.n_epochs)) as t:
        for i in t:
            # Create random instances
            params = problem.generate_params()
            params = {k: v.to(cfg.device) for k, v in params.items()}
            problem.set_params(**params)
            # Find initial solutions
            init_x = problem.generate_init_x()
            actor.manual_seed(cfg.seed)

            # Training loop
            if cfg.training.method == "ppo":
                train_ppo(actor, critic, actor_opt, critic_opt, problem,
init_x, cfg)
            elif cfg.training.method == "es":
                train_es(actor, problem, init_x, es, cfg)
                scheduler.step()

            # Rerun trained model
            train_out = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            train_loss = torch.mean(train_out["min_cost"])

            # Track the loss for plotting
            epoch_losses.append(train_loss.item())

            # Print loss for each epoch (so it doesn't overwrite)
            print(f"Epoch {i+1}/{cfg.training.n_epochs}, Training loss:
{train_loss:.4f}")

            # Update the progress bar with current loss
            t.set_description(f"Epoch {i+1}/{cfg.training.n_epochs},
Training loss: {train_loss:.4f}")

            # Save the model after every epoch
            path = os.path.join(os.getcwd(), "models")
            name = cfg.problem + str(cfg.problem_dim) + "-" +
cfg.training.method + ".pt"
            create_folder(path)
            torch.save(actor.state_dict(), os.path.join(path, name))
    # After training is done, plot the training loss
    plt.plot(range(cfg.training.n_epochs), epoch_losses)
    plt.xlabel('Epoch')
    plt.ylabel('Training Loss')
    plt.title(f'Training Loss over Epochs ({cfg.training.optimizer})')
    plt.show()

if __name__ == "__main__":
    main()





'''
import os
import random
import torch
import numpy as np
import hydra
from omegaconf import OmegaConf
from tqdm import tqdm
from neuralsa.configs import NeuralSAExperiment
from neuralsa.model import (
    BinPackingActor,
    BinPackingCritic,
    KnapsackActor,
    KnapsackCritic,
    TSPActor,
    TSPCritic,
)
from neuralsa.problem import TSP, BinPacking, Knapsack
from neuralsa.sa import sa
from neuralsa.training import EvolutionStrategies
from neuralsa.training.ppo import ppo
from neuralsa.training.replay import Replay
import csv

# For reproducibility on GPU
```

```
torch.backends.cudnn.deterministic = True


def create_folder(dirname):
    if not os.path.exists(dirname):
        os.makedirs(dirname)
        print(f"Created: {dirname}")


def train_es(actor, problem, init_x, es, cfg):
    with torch.no_grad():
        es.zero_updates()
        for _ in range(es.population):
            es.perturb(antithetic=True)

            # Run SA and compute the loss
            results = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            loss = torch.mean(results[cfg.training.reward])
            es.collect(loss)

        es.step(reshape_fitness=True)

    return torch.mean(torch.tensor(es.objective))


def train_ppo(actor, critic, actor_opt, critic_opt, problem, init_x, cfg):
    # Create replay to store transitions
    replay = Replay(cfg.sa.outer_steps * cfg.sa.inner_steps)
    # Run SA and collect transitions
    sa(actor, problem, init_x, cfg, replay=replay, baseline=False,
greedy=False)
    # Optimize the policy with PPO
    ppo(actor, critic, replay, actor_opt, critic_opt, cfg)


@hydra.main(config_path="conf", config_name="config", version_base=None)
def main(cfg: NeuralSAExperiment) -> None:
    if "cuda" in cfg.device and not torch.cuda.is_available():
        cfg.device = "cpu"
        print("CUDA device not found. Running on cpu.")

    # Set seeds
    torch.manual_seed(cfg.seed)
    random.seed(cfg.seed)
    np.random.seed(cfg.seed)

    # Set Problem and Networks
    if cfg.problem == "knapsack":
        problem = Knapsack(
            cfg.problem_dim, cfg.n_problems, device=cfg.device,
params={"capacity": cfg.capacity}
        )
        actor = KnapsackActor(cfg.embed_dim, device=cfg.device)
        critic = KnapsackCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "binpacking":
        problem = BinPacking(cfg.problem_dim, cfg.n_problems,
device=cfg.device)
        actor = BinPackingActor(cfg.embed_dim, device=cfg.device)
        critic = BinPackingCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "tsp":
        problem = TSP(cfg.problem_dim, cfg.n_problems, device=cfg.device)
        actor = TSPActor(cfg.embed_dim, device=cfg.device)
        critic = TSPCritic(cfg.embed_dim, device=cfg.device)
    else:
        raise ValueError("Invalid problem name.")

    # Set problem seed
    problem.manual_seed(cfg.seed)

    # Initialize optimizers
    if cfg.training.method == "ppo":
        if cfg.training.optimizer == "adam":
            actor_opt = torch.optim.Adam(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.Adam(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
```

```
            )
        elif cfg.training.optimizer == "adamw":
            actor_opt = torch.optim.AdamW(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.AdamW(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "nadam":
            actor_opt = torch.optim.NAdam(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.NAdam(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "adabelief":
            from adabelief_pytorch import AdaBelief
            actor_opt = AdaBelief(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = AdaBelief(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "adadelta":
            actor_opt = torch.optim.Adadelta(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.Adadelta(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        else:
            raise ValueError(f"Unsupported optimizer:
{cfg.training.optimizer}")

    epoch_losses = []

    # Open CSV file to log losses
    with open('training_losses.csv', mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(['Epoch', 'Training Loss'])

        with tqdm(range(cfg.training.n_epochs)) as t:
            for epoch in t:
                # Create random instances
                params = problem.generate_params()
                params = {k: v.to(cfg.device) for k, v in params.items()}
                problem.set_params(**params)
                # Find initial solutions
                init_x = problem.generate_init_x()
                actor.manual_seed(cfg.seed)

                # Training loop
                if cfg.training.method == "ppo":
                    train_ppo(actor, critic, actor_opt, critic_opt,
problem, init_x, cfg)
                elif cfg.training.method == "es":
                    es = EvolutionStrategies(optimizer,
cfg.training.stddev, cfg.training.population)
                    train_es(actor, problem, init_x, es, cfg)

                # Rerun trained model
                train_out = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
                train_loss = torch.mean(train_out["min_cost"])

                # Track the loss for plotting and CSV logging
                epoch_losses.append(train_loss.item())
                print(f"Epoch {epoch+1}/{cfg.training.n_epochs}, Training
loss: {train_loss:.4f}")

                # Log the epoch loss into the CSV
                writer.writerow([epoch+1, train_loss.item()])

                # Save model every epoch
                path = os.path.join(os.getcwd(), "models")
                name = cfg.problem + str(cfg.problem_dim) + "-" +
cfg.training.method + ".pt"
                create_folder(path)
                torch.save(actor.state_dict(), os.path.join(path, name))

    # After training is done, plot the training loss
    plt.plot(range(cfg.training.n_epochs), epoch_losses)
    plt.xlabel('Epoch')
    plt.ylabel('Training Loss')
    plt.title(f'Training Loss over Epochs ({cfg.training.optimizer})')
    plt.show()


if __name__ == "__main__":
    main()
'''

'''


import os
import random

import hydra
import numpy as np
import torch
from hydra.core.config_store import ConfigStore
from omegaconf import OmegaConf
from torch.optim import SGD
from torch.optim.lr_scheduler import MultiStepLR
from tqdm import tqdm
import matplotlib.pyplot as plt

from neuralsa.configs import NeuralSAExperiment
from neuralsa.model import (
    BinPackingActor,
    BinPackingCritic,
    KnapsackActor,
    KnapsackCritic,
    TSPActor,
    TSPCritic,
)
from neuralsa.problem import TSP, BinPacking, Knapsack
from neuralsa.sa import sa
from neuralsa.training import EvolutionStrategies
from neuralsa.training.ppo import ppo
from neuralsa.training.replay import Replay

# For reproducibility on GPU
torch.backends.cudnn.deterministic = True


def create_folder(dirname):
    if not os.path.exists(dirname):
        os.makedirs(dirname)
        print(f"Created: {dirname}")


def train_es(actor, problem, init_x, es, cfg):
    with torch.no_grad():
        es.zero_updates()
        for _ in range(es.population):
            es.perturb(antithetic=True)

            # Run SA and compute the loss
            results = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            loss = torch.mean(results[cfg.training.reward])
            es.collect(loss)

        es.step(reshape_fitness=True)

    return torch.mean(torch.tensor(es.objective))


def train_ppo(actor, critic, actor_opt, critic_opt, problem, init_x, cfg):
    # Create replay to store transitions
    replay = Replay(cfg.sa.outer_steps * cfg.sa.inner_steps)
    # Run SA and collect transitions
    sa(actor, problem, init_x, cfg, replay=replay, baseline=False,
greedy=False)
    # Optimize the policy with PPO
    ppo(actor, critic, replay, actor_opt, critic_opt, cfg)


cs = ConfigStore.instance()
# Registering the Config class with the name 'config'.
cs.store(name="base_config", node=NeuralSAExperiment, group="experiment")


@hydra.main(config_path="conf", config_name="config", version_base=None)
def main(cfg: NeuralSAExperiment) -> None:
    if "cuda" in cfg.device and not torch.cuda.is_available():
        cfg.device = "cpu"
        print("CUDA device not found. Running on cpu.")

    # Define temperature decay parameter as a function of the number of
steps
    alpha = np.log(cfg.sa.stop_temp) - np.log(cfg.sa.init_temp)
    cfg.sa.alpha = np.exp(alpha / cfg.sa.outer_steps).item()

    print(OmegaConf.to_yaml(cfg))

    # Set seeds
    torch.manual_seed(cfg.seed)
    random.seed(cfg.seed)
    np.random.seed(cfg.seed)

    # Set Problem and Networks
    if cfg.problem == "knapsack":
        problem = Knapsack(
            cfg.problem_dim, cfg.n_problems, device=cfg.device,
params={"capacity": cfg.capacity}
        )
        actor = KnapsackActor(cfg.embed_dim, device=cfg.device)
        critic = KnapsackCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "binpacking":
        problem = BinPacking(cfg.problem_dim, cfg.n_problems,
device=cfg.device)
        actor = BinPackingActor(cfg.embed_dim, device=cfg.device)
        critic = BinPackingCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "tsp":
        problem = TSP(cfg.problem_dim, cfg.n_problems, device=cfg.device)
        actor = TSPActor(cfg.embed_dim, device=cfg.device)
        critic = TSPCritic(cfg.embed_dim, device=cfg.device)
    else:
        raise ValueError("Invalid problem name.")

    # Set problem seed
    problem.manual_seed(cfg.seed)

    # Initialize optimizers
    if cfg.training.method == "ppo":
        if cfg.training.optimizer == "adam":
            actor_opt = torch.optim.Adam(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.Adam(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "adamw":
            actor_opt = torch.optim.AdamW(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.AdamW(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "nadam":
            actor_opt = torch.optim.NAdam(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.NAdam(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "adabelief":
            from adabelief_pytorch import AdaBelief
            actor_opt = AdaBelief(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = AdaBelief(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "adadelta":
            actor_opt = torch.optim.Adadelta(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.Adadelta(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        else:
            raise ValueError(f"Unsupported optimizer:
{cfg.training.optimizer}")

    elif cfg.training.method == "es":
        # Optimization specs
        optimizer = SGD(actor.parameters(), lr=cfg.training.lr,
momentum=cfg.training.momentum)
        es = EvolutionStrategies(optimizer, cfg.training.stddev,
cfg.training.population)
        milestones = [int(cfg.training.n_epochs * m) for m in
cfg.training.milestones]
        scheduler = MultiStepLR(optimizer, milestones=milestones,
gamma=0.1)
    else:
        raise ValueError("Invalid training method.")
    # Initialize list to store losses for plotting
    epoch_losses = []

    with tqdm(range(cfg.training.n_epochs)) as t:
        for i in t:
            # Create random instances
            params = problem.generate_params()
            params = {k: v.to(cfg.device) for k, v in params.items()}
            problem.set_params(**params)
            # Find initial solutions
            init_x = problem.generate_init_x()
            actor.manual_seed(cfg.seed)

            # Training loop
            if cfg.training.method == "ppo":
                train_ppo(actor, critic, actor_opt, critic_opt, problem,
init_x, cfg)
            elif cfg.training.method == "es":
                train_es(actor, problem, init_x, es, cfg)
                scheduler.step()

            # Rerun trained model
            train_out = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            train_loss = torch.mean(train_out["min_cost"])

            # Track the loss for plotting
            epoch_losses.append(train_loss.item())

            t.set_description(f"Training loss: {train_loss:.4f}")

            path = os.path.join(os.getcwd(), "models")
            name = cfg.problem + str(cfg.problem_dim) + "-" +
cfg.training.method + ".pt"
            create_folder(path)
            torch.save(actor.state_dict(), os.path.join(path, name))

    # After training is done, plot the training loss
    plt.plot(range(cfg.training.n_epochs), epoch_losses)
    plt.xlabel('Epoch')
    plt.ylabel('Training Loss')
    plt.title(f'Training Loss over Epochs ({cfg.training.optimizer})')
    plt.show()


if __name__ == "__main__":
    main()
'''




'''
import os
import random

import hydra
import numpy as np
import torch
from hydra.core.config_store import ConfigStore
from omegaconf import OmegaConf
from torch.optim import SGD
from torch.optim.lr_scheduler import MultiStepLR
from tqdm import tqdm

from neuralsa.configs import NeuralSAExperiment
from neuralsa.model import (
    BinPackingActor,
    BinPackingCritic,
    KnapsackActor,
    KnapsackCritic,
    TSPActor,
    TSPCritic,
)
from neuralsa.problem import TSP, BinPacking, Knapsack
from neuralsa.sa import sa
from neuralsa.training import EvolutionStrategies
from neuralsa.training.ppo import ppo
from neuralsa.training.replay import Replay
```

```
# For reproducibility on GPU
torch.backends.cudnn.deterministic = True


def create_folder(dirname):
    if not os.path.exists(dirname):
        os.makedirs(dirname)
        print(f"Created: {dirname}")


def train_es(actor, problem, init_x, es, cfg):
    with torch.no_grad():
        es.zero_updates()
        for _ in range(es.population):
            es.perturb(antithetic=True)

            # Run SA and compute the loss
            results = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            loss = torch.mean(results[cfg.training.reward])
            es.collect(loss)

        es.step(reshape_fitness=True)

    return torch.mean(torch.tensor(es.objective))


def train_ppo(actor, critic, actor_opt, critic_opt, problem, init_x, cfg):
    # Create replay to store transitions
    replay = Replay(cfg.sa.outer_steps * cfg.sa.inner_steps)
    # Run SA and collect transitions
    sa(actor, problem, init_x, cfg, replay=replay, baseline=False,
greedy=False)
    # Optimize the policy with PPO
    ppo(actor, critic, replay, actor_opt, critic_opt, cfg)


cs = ConfigStore.instance()
# Registering the Config class with the name 'config'.
cs.store(name="base_config", node=NeuralSAExperiment, group="experiment")
```

```
@hydra.main(config_path="conf", config_name="config", version_base=None)
def main(cfg: NeuralSAExperiment) -> None:
    if "cuda" in cfg.device and not torch.cuda.is_available():
        cfg.device = "cpu"
        print("CUDA device not found. Running on cpu.")

    # Define temperature decay parameter as a function of the number of
steps
    alpha = np.log(cfg.sa.stop_temp) - np.log(cfg.sa.init_temp)
    cfg.sa.alpha = np.exp(alpha / cfg.sa.outer_steps).item()

    print(OmegaConf.to_yaml(cfg))

    # Set seeds
    torch.manual_seed(cfg.seed)
    random.seed(cfg.seed)
    np.random.seed(cfg.seed)

    # Set Problem and Networks
    if cfg.problem == "knapsack":
        problem = Knapsack(
            cfg.problem_dim, cfg.n_problems, device=cfg.device,
params={"capacity": cfg.capacity}
        )
        actor = KnapsackActor(cfg.embed_dim, device=cfg.device)
        critic = KnapsackCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "binpacking":
        problem = BinPacking(cfg.problem_dim, cfg.n_problems,
device=cfg.device)
        actor = BinPackingActor(cfg.embed_dim, device=cfg.device)
        critic = BinPackingCritic(cfg.embed_dim, device=cfg.device)
    elif cfg.problem == "tsp":
        problem = TSP(cfg.problem_dim, cfg.n_problems, device=cfg.device)
        actor = TSPActor(cfg.embed_dim, device=cfg.device)
        critic = TSPCritic(cfg.embed_dim, device=cfg.device)
    else:
        raise ValueError("Invalid problem name.")
    # Set problem seed
    problem.manual_seed(cfg.seed)

    # Initialize optimizers
    if cfg.training.method == "ppo":
        if cfg.training.optimizer == "adam":
            actor_opt = torch.optim.Adam(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.Adam(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "adamw":
            actor_opt = torch.optim.AdamW(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.AdamW(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "nadam":
            actor_opt = torch.optim.NAdam(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = torch.optim.NAdam(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "adabelief":
            from adabelief_pytorch import AdaBelief
            actor_opt = AdaBelief(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
            critic_opt = AdaBelief(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        elif cfg.training.optimizer == "adadelta":
            actor_opt = torch.optim.Adadelta(
                actor.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )


            critic_opt = torch.optim.Adadelta(
                critic.parameters(), lr=cfg.training.lr,
weight_decay=cfg.training.weight_decay
            )
        else:
            raise ValueError(f"Unsupported optimizer:
{cfg.training.optimizer}")

    elif cfg.training.method == "es":
        # Optimization specs
        optimizer = SGD(actor.parameters(), lr=cfg.training.lr,
momentum=cfg.training.momentum)
        es = EvolutionStrategies(optimizer, cfg.training.stddev,
cfg.training.population)
        milestones = [int(cfg.training.n_epochs * m) for m in
cfg.training.milestones]
        scheduler = MultiStepLR(optimizer, milestones=milestones,
gamma=0.1)
    else:
        raise ValueError("Invalid training method.")

    with tqdm(range(cfg.training.n_epochs)) as t:
        for i in t:
            # Create random instances
            params = problem.generate_params()
            params = {k: v.to(cfg.device) for k, v in params.items()}
            problem.set_params(**params)
            # Find initial solutions
            init_x = problem.generate_init_x()
            actor.manual_seed(cfg.seed)
            # Training loop
            if cfg.training.method == "ppo":
                train_ppo(actor, critic, actor_opt, critic_opt, problem,
init_x, cfg)
            elif cfg.training.method == "es":
                train_es(actor, problem, init_x, es, cfg)
                scheduler.step()

            # Rerun trained model
            train_out = sa(actor, problem, init_x, cfg, replay=None,
baseline=False, greedy=False)
            train_loss = torch.mean(train_out["min_cost"])

            t.set_description(f"Training loss: {train_loss:.4f}")

            path = os.path.join(os.getcwd(), "models")
            name = cfg.problem + str(cfg.problem_dim) + "-" +
cfg.training.method + ".pt"
            create_folder(path)
            torch.save(actor.state_dict(), os.path.join(path, name))


if __name__ == "__main__":
    main()
    '''
```

ppo.py

```
# Copyright (c) 2023 Qualcomm Technologies, Inc.
# All Rights Reserved.


from typing import Tuple

import numpy as np
import torch
from omegaconf import DictConfig
from torch import nn
from torch.optim import Optimizer

from neuralsa.model import SAModel
from neuralsa.training.replay import Replay, Transition


def ppo(
    actor: SAModel,
    critic: nn.Module,
    replay: Replay,
    actor_opt: Optimizer,
    critic_opt: Optimizer,
    cfg: DictConfig,
    criterion=torch.nn.MSELoss(),
) -> Tuple[float, float]:
    """
    Optimises the actor and the critic in PPO for 'ppo_epochs' epochs
using the transitions
    recorded in 'replay'.

    Parameters
    ----------
    actor, critic: nn.Module
    replay: Replay object (see replay.py)
    actor_opt, critic_opt: torch.optim
    cfg: OmegaConf DictConfig
        Config containing PPO hyperparameters (see below).
    criterion: torch loss
        Loss function for the critic.

    Returns
    -------
    actor_loss, critic_loss
    """

    # PPO hyper-parameters
    ppo_epochs = cfg.training.ppo_epochs
    trace_decay = cfg.training.trace_decay
    eps_clip = cfg.training.eps_clip
    batch_size = cfg.training.batch_size
    n_problems = cfg.n_problems
    problem_dim = cfg.problem_dim
    device = cfg.device

    actor.train()
    critic.train()
    # Get transitions
    with torch.no_grad():
        transitions = replay.memory
        nt = len(transitions)
        # Gather transition information into tensors
        batch = Transition(*zip(*transitions))
        state = torch.stack(batch.state).view(nt * n_problems,
problem_dim, -1)
        action = torch.stack(batch.action).detach().view(nt * n_problems,
-1)
        next_state = torch.stack(batch.next_state).detach().view(nt *
n_problems, problem_dim, -1)
        old_log_probs = torch.stack(batch.old_log_probs).view(nt *
n_problems, -1)
        # Evaluate the critic
        state_values = critic(state).view(nt, n_problems, 1)
        next_state_values = critic(next_state).view(nt, n_problems, 1)
        # Get rewards and advantage estimate
        rewards_to_go = torch.zeros((nt, n_problems, 1), device=device,
dtype=torch.float32)
        advantages = torch.zeros((nt, n_problems, 1), device=device,
dtype=torch.float32)
        discounted_reward = torch.zeros((n_problems, 1), device=device)
        advantage = torch.zeros((n_problems, 1), device=device)
        # Loop through the batch transitions starting from the end of the
episode
        # Compute discounted rewards, and advantage using td error
        for i, reward, gamma in zip(
            reversed(np.arange(len(transitions))), reversed(batch.reward),
reversed(batch.gamma)
        ):
            if gamma == 0:
                discounted_reward = torch.zeros((n_problems, 1),
device=device)
                advantage = torch.zeros((n_problems, 1), device=device)
            discounted_reward = reward + (gamma * discounted_reward)
            td_error = reward + gamma * next_state_values[i, ...] -
state_values[i, ...]
            advantage = td_error + gamma * trace_decay * advantage
            rewards_to_go[i, ...] = discounted_reward
            advantages[i, ...] = advantage
        # Normalize advantages
        advantages = advantages - advantages.mean() / (advantages.std() +
1e-8)
    advantages = advantages.view(n_problems * nt, -1)
    rewards_to_go = rewards_to_go.view(n_problems * nt, -1)

    actor_loss, critic_loss = None, None
    for _ in range(ppo_epochs):
        actor_opt.zero_grad()
        critic_opt.zero_grad()
        if nt > 1:  # Avoid instabilities
            # Shuffle the trajectory, good for training
            perm = np.arange(state.shape[0])
            np.random.shuffle(perm)
            perm = torch.LongTensor(perm).to(device)
            state = state[perm, :].clone()
            action = action[perm, :].clone()
            rewards_to_go = rewards_to_go[perm, :].clone()
            advantages = advantages[perm, :].clone()
            old_log_probs = old_log_probs[perm, :].clone()
            # Run batch optimization
            for j in range(nt * n_problems, 0, -batch_size):
                nb = min(j, batch_size)
                if nb <= 1:  # Avoid instabilities
                    continue
                # Get a batch of transitions
                batch_idx = np.arange(j - nb, j)
                # Gather batch information into tensors
                batch_state = state[batch_idx, ...]
                batch_action = action[batch_idx, ...]
                batch_advantages = advantages[batch_idx, 0]
                batch_rewards_to_go = rewards_to_go[batch_idx, 0]
                batch_old_log_probs = old_log_probs[batch_idx, 0]
                # Evaluate the critic
                batch_state_values = critic(batch_state)
                # Evaluate the actor
                batch_log_probs = actor.evaluate(batch_state,
batch_action)
                # Compute critic loss
                critic_loss = 0.5 * criterion(
                    batch_state_values.squeeze(),
batch_rewards_to_go.detach()
                )
                # Compute actor loss
                ratios = torch.exp(batch_log_probs -
batch_old_log_probs.detach())
                surr1 = ratios * batch_advantages.detach()
                surr2 = torch.clamp(ratios, 1 - eps_clip, 1 + eps_clip) *
batch_advantages.detach()
                actor_loss = -torch.min(surr1, surr2).mean()
                # Optimize
                actor_loss.backward()
                critic_loss.backward()
                actor_opt.step()
                critic_opt.step()
    return actor_loss.item(), critic_loss.item()
```