

140 README.md

```

...   ...   @@ -1,22 +1,19 @@
1     - # Attention, Learn to Solve Routing Problems!
2     -
3     - Attention based model for learning to solve the Travelling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP),
4     - Orienteering Problem (OP) and (Stochastic) Prize Collecting TSP (PCTSP). Training with REINFORCE with greedy rollout
5     - baseline.
6
7     - ![TSP100](images/tsp.gif)
8
9     + # Analyzing the Vehicle Route Problem: A Heuristic Approach to Route Planning and Optimization
10    +
11    + ## Introduction
12    + This model incorporates the attention based model from the paper [Attention, Learn to Solve Routing Problems!]
13    + (https://openreview.net/forum?id=ByxBFsRqYm) which was accepted at [ICLR 2019](https://iclr.cc/Conferences/2019) for
14    + learning to solve the Travelling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP), Orienteering Problem (OP)
15    + and (Stochastic) Prize Collecting TSP (PCTSP). Training with REINFORCE with greedy rollout baseline.
16    + <br />
17    + <br />
18    + The paper describing this specific extension of the aforementioned model is from the paper [Analyzing the Vehicle Route
19    + Problem: A Heuristic Approach to Route Planning and Optimization](readfiles/paper.pdf)
20    + <br />
21    + <br />
22    + Note: All files linked in this README are accessible in the following locations:
23    + * Images are in [images/](images)
24    + * Python scripts are in the root directory
25    + * All other filetypes (PDF, diff, etc.) are in [readfiles/](readfiles)
26
27    14
28    15    ## Paper
29
30    - For more details, please see our paper [Attention, Learn to Solve Routing Problems!](https://openreview.net/forum?
31    - id=ByxBFsRqYm) which has been accepted at [ICLR 2019](https://iclr.cc/Conferences/2019). If this code is useful for your
32    - work, please cite our paper:
33    -
34    - ```
35    - @inproceedings{
36    -     kool2018attention,
37    -     title={Attention, Learn to Solve Routing Problems!},
38    -     author={Wouter Kool and Herke van Hoof and Max Welling},
39    -     booktitle={International Conference on Learning Representations},
40    -     year={2019},
41    -     url={https://openreview.net/forum?id=ByxBFsRqYm},
42    - }
43    - ```
44
45    16    + For more details, please see our paper [Analyzing the Vehicle Route Problem: A Heuristic Approach to Route Planning and
46    + Optimization](readfiles/paper.pdf).
47
48    17
49    18    ## Dependencies
50
51    19
52    20
53    21    * [tensorboard_logger](https://github.com/TeamHG-Memex/tensorboard_logger)
54    22    * Matplotlib (optional, only for plotting)
55
56    23
57    24
58    25
59    26
60    27
61
62    31    - ## Quick start
63    32    -
64    33    - For training TSP instances with 20 nodes and using rollout as REINFORCE baseline:
65    34    - ```bash
66    35    - python run.py --graph_size 20 --baseline rollout --run_name 'tsp20_rollout'
67
68    28    + ## Differences
69    29    + To view the differences between the implementation in this codebase, reference [this document]() to view pdf that will
70    30    + compare the original code with the added code. The greatest changes were made in [plot_vrp.py](plot_vrp.py),

```

```

[simple_tsp.py](simple_tsp.py),
31 + as well as smaller changes and additions in other locations. Most of the more significant changes that were made by us
    have a comment
32 + somewhere near the top indicating that it is original code, but not all changes are indicated in this way.<br />
33 + <br />
34 + A .diff file is also available [here](readfiles/differences.diff)
35 +
36 + ## Running the code
37 + To run the code that demonstrates the implementation described in the paper (which as noted, did not make any real
    changes to the
38 + sample routes that were being utilized), run the following command in the root folder:
36 39 '''
37 -
38 - ## Usage
39 -
40 - ### Generating data
41 -
42 - Training data is generated on the fly. To generate validation and test data (same as used in the paper) for all
    problems:
43 - '''bash
44 - python generate_data.py --problem all --name validation --seed 4321
45 - python generate_data.py --problem all --name test --seed 1234
40 + python plot_vrp.py
46 41 '''
42 + The output will be two sets of ten images, one which runs the Capacitated Vehicle Routing Problem (CVRP) '''cvrp_100'''
    model on
43 + a graph of 100 nodes, and one which runs this as well as the Traveling Salesman Problem (TSP) '''tsp_100''' model on
    each
44 + outputted route from the '''cvrp_100''' model. An example is shown below: <br />
45 + <br />
46 + CVRP:
47 + ![CVRP100](images/cvrp_0.png)<br />
47 48
48 - ### Training
49 -
50 - For training TSP instances with 20 nodes and using rollout as REINFORCE baseline and using the generated validation set:
51 - '''bash
52 - python run.py --graph_size 20 --baseline rollout --run_name 'tsp20_rollout' --val_dataset
    data/tsp/tsp20_validation_seed4321.pkl
53 - '''
49 + CVRP + TSP on routes:
50 + ![CVRPTSP100](images/cvrp_and_tsp_0.png)
54 51
55 - #### Multiple GPUs
56 - By default, training will happen *on all available GPUs*. To disable CUDA at all, add the flag --no_cuda.
57 - Set the environment variable CUDA_VISIBLE_DEVICES to only use specific GPUs:
58 - '''bash
59 - CUDA_VISIBLE_DEVICES=2,3 python run.py
60 - '''
61 - Note that using multiple GPUs has limited efficiency for small problem sizes (up to 50 nodes).
52 + Observing the two images it is clear that each route is identical, and the distance for each
53 + route as well as the total distance did not change.<br />
54 + <br />
55 + The [plot_vrp.py](plot_vrp.py) script can be altered in order to apply different versions of the CVRP models as well as
different
56 + versions of the TSP model that is overlayed by following the steps from [the original README file]
(readfiles/README(Kool).md) which allow
57 + users to generate and train models with specific parameters. If the amount of graph nodes is changed for the new model,
this
58 + must also be adjusted in the [plot_vrp.py](plot_vrp.py) script.
62 59
63 - #### Warm start

```

```

64 - You can initialize a run using a pretrained model by using the `--load_path` option:
65 - ```bash
66 - python run.py --graph_size 100 --load_path pretrained/tsp_100/epoch-99.pt
67 - ```
68 -
69 - The `--load_path` option can also be used to load an earlier run, in which case also the optimizer state will be loaded:
70 - ```bash
71 - python run.py --graph_size 20 --load_path 'outputs/tsp_20/tsp20_rollout_{datetime}/epoch-0.pt'
72 - ```
73 -
74 - The `--resume` option can be used instead of the `--load_path` option, which will try to resume the run, e.g. load
    additionally the baseline state, set the current epoch/step counter and set the random number generator state.
75 -
76 - ### Evaluation
77 - To evaluate a model, you can add the `--eval-only` flag to `run.py`, or use `eval.py`, which will additionally measure
    timing and save the results:
78 - ```bash
79 - python eval.py data/tsp/tsp20_test_seed1234.pkl --model pretrained/tsp_20 --decode_strategy greedy
80 - ```
81 - If the epoch is not specified, by default the last one in the folder will be used.
82 -
83 - #### Sampling
84 - To report the best of 1280 sampled solutions, use
85 - ```bash
86 - python eval.py data/tsp/tsp20_test_seed1234.pkl --model pretrained/tsp_20 --decode_strategy sample --width 1280 --
    eval_batch_size 1
87 - ```
88 - Beam Search (not in the paper) is also recently added and can be used using `--decode_strategy bs --width {beam_size}`.
89 -
90 - #### To run baselines
91 - Baselines for different problems are within the corresponding folders and can be ran (on multiple datasets at once) as
    follows
92 - ```bash
93 - python -m problems.tsp.tsp_baseline farthest_insertion data/tsp/tsp20_test_seed1234.pkl data/tsp/tsp50_test_seed1234.pkl
    data/tsp/tsp100_test_seed1234.pkl
94 - ```
95 - To run baselines, you need to install [Compass](https://github.com/bcamath-ds/compass) by running the
    `install_compass.sh` script from within the `problems/op` directory and [Concorde]
    (http://www.math.uwaterloo.ca/tsp/concorde.html) using the `install_concorde.sh` script from within `problems/tsp`.
    [LKH3](http://akira.ruc.dk/~keld/research/LKH-3/) should be automatically downloaded and installed when required. To use
    [Gurobi](http://www.gurobi.com), obtain a ([free academic](http://www.gurobi.com/registration/academic-license-reg))
    license and follow the [installation instructions]
    (https://www.gurobi.com/documentation/8.1/quickstart_windows/installing_the_anaconda_py.html).
96 -
97 - ### Other options and help
98 - ```bash
99 - python run.py -h
100 - python eval.py -h
101 - ```
102
103 + ## Slight improvements with Beam Search
104
105 + ### Example CVRP solution
106 - See `plot_vrp.ipynb` for an example of loading a pretrained model and plotting the result for Capacitated VRP with 100
    nodes.
107
108 + The [run.py](run.py) file has the ability to train models using Beam Search, and this slightly improves the CVRP model
    as seen
109
110 + in this image below, which is the output after creating and training a CVRP model for a graph of 20 nodes.<br />
111 + ![beam](images/Beam Search.jpg)
112
113 + The average cost for a route using a model with beam search is slightly less than it would be when using a model
    without.
114
115 + 66
116 - ![CVRP100](images/cvrp_0.png)

```

107	67	
108	68	## Acknowledgements
109		- Thanks to [pemami4911/neural-combinatorial-rl-pytorch](https://github.com/pemami4911/neural-combinatorial-rl-pytorch) for getting me started with the code for the Pointer Network.
110	69	
111		- This repository includes adaptations of the following repositories as baselines:
112		- * https://github.com/MichelDeudon/encode-attend-navigate
113		- * https://github.com/mc-ride/orienteering
114		- * https://github.com/jordanamecler/PCTSP
115		- * https://github.com/rafael2reis/salesman
	70	+ This repository includes adaptations of the following repository as a baseline:
	71	+ * https://github.com/wouterkool/attention-learn-to-route

▼ BIN +136 KB images/Beam Search.JPG

Binary file not shown.

▼ BIN +47.4 KB (150%) images/cvrp\_0.png

Binary file not shown.

▼ BIN +44.3 KB (150%) images/cvrp\_1.png

Binary file not shown.

▼ BIN +46 KB (140%) images/cvrp\_2.png

Binary file not shown.

▼ BIN +50.2 KB (150%) images/cvrp\_3.png

Binary file not shown.

▼ BIN +47.9 KB (150%) images/cvrp\_4.png

Binary file not shown.

▼ BIN +45.4 KB (150%) images/cvrp\_5.png

Binary file not shown.

▼ BIN +48.4 KB (150%) images/cvrp\_6.png

Binary file not shown.

▼ BIN +47.4 KB (150%) images/cvrp\_7.png

Binary file not shown.

▼ BIN +45.1 KB (150%) images/cvrp\_8.png

Binary file not shown.

▼ BIN +46.3 KB (150%) images/cvrp\_9.png

Binary file not shown.

✓ BIN +146 KB images/cvrp\_and\_tsp\_0.png 

Binary file not shown.

✓ BIN +134 KB images/cvrp\_and\_tsp\_1.png 

Binary file not shown.

✓ BIN +151 KB images/cvrp\_and\_tsp\_2.png 

Binary file not shown.

✓ BIN +150 KB images/cvrp\_and\_tsp\_3.png 

Binary file not shown.

✓ BIN +152 KB images/cvrp\_and\_tsp\_4.png 

Binary file not shown.

✓ BIN +137 KB images/cvrp\_and\_tsp\_5.png 

Binary file not shown.

✓ BIN +147 KB images/cvrp\_and\_tsp\_6.png 

Binary file not shown.

✓ BIN +143 KB images/cvrp\_and\_tsp\_7.png 

Binary file not shown.

✓ BIN +138 KB images/cvrp\_and\_tsp\_8.png 

Binary file not shown.

✓ BIN +142 KB images/cvrp\_and\_tsp\_9.png 

Binary file not shown.

✓ BIN +1.85 MB images/tsp\_adjusted.gif 

Binary file not shown.

✓ 218 ■■■■■ plot\_vrp.py 

```
...      ...      @@ -0,0 +1,218 @@
          1      + #!/usr/bin/env python
          2      + # coding: utf-8
```

```

3 +
4 + # In[1]:
5 +
6 +
7 + import os
8 + import numpy as np
9 + import torch
10 + # import simple_tsp
11 +
12 +
13 +
14 + # In[2]:
15 +
16 +
17 + from torch.utils.data import DataLoader
18 +
19 + # import simple_tsp
20 + from generate_data import generate_vrp_data
21 + from utils import load_model
22 + from problems import CVRP, TSP
23 +
24 + # In[3]:
25 +
26 +
27 + # get_ipython().run_line_magic('matplotlib', 'inline')
28 + from matplotlib import pyplot as plt
29 +
30 + from matplotlib.collections import PatchCollection
31 + from matplotlib.patches import Rectangle
32 + from matplotlib.lines import Line2D
33 +
34 + # Code inspired by Google OR Tools plot:
35 + # https://github.com/google/or-tools/blob/fb12c5ded7423d524fc6c95656a9bdc290a81d4d/examples/python/cvrptw_plot.py
36 +
37 + def discrete_cmap(N, base_cmap=None):
38 +     """
39 +     Create an N-bin discrete colormap from the specified input map
40 +     """
41 +     # Note that if base_cmap is a string or None, you can simply do
42 +     # return plt.cm.get_cmap(base_cmap, N)
43 +     # The following works for string, None, or a colormap instance:
44 +
45 +     base = plt.cm.get_cmap(base_cmap)
46 +     color_list = base(np.linspace(0, 1, N))
47 +     cmap_name = base.name + str(N)
48 +     return base.from_list(cmap_name, color_list, N)
49 +
50 +
51 + def coord_to_loc(dataset, coord): #written by Isabelle Akian
52 +     locs = dataset['loc'].cpu().numpy()
53 +
54 +     return np.where(locs == coord)[0][0]+1
55 +
56 +
57 + def get_routes_and_coords(datasets, route_for_coord):#written by Isabelle Akian
58 +     route_list = [r[r != 0] for r in np.split(route_for_coord.cpu().numpy(), np.where(route_for_coord == 0)[0]) if (r !=
59 +
60 +     new_locs = datasets['loc'].cpu().numpy()
61 +     new_coords = [[]]*len(route_list)
62 +
63 +     for veh_number, r in enumerate(route_list):
64 +         new_coords[veh_number] = new_locs[r - 1, :]

```

```

65 +
66 +     return new_coords, route_list
67 +
68 +
69 + def plot_vehicle_routes(data, route, ax1, markersize=5, visualize_demands=False, demand_scale=1, round_demand=False):
70 +     """
71 +     Plot the vehicle routes on matplotlib axis ax1.
72 +     """
73 +
74 +     # route is one sequence, separating different routes with 0 (depot)
75 +     routes = [r[r!=0] for r in np.split(route.cpu().numpy(), np.where(route==0)[0]) if (r != 0).any()]
76 +     depot = data['depot'].cpu().numpy()
77 +     locs = data['loc'].cpu().numpy()
78 +     demands = data['demand'].cpu().numpy() * demand_scale
79 +     capacity = demand_scale # Capacity is always 1
80 +
81 +     x_dep, y_dep = depot
82 +     ax1.plot(x_dep, y_dep, 'sk', markersize=markersize*4)
83 +     ax1.set_xlim(0, 1)
84 +     ax1.set_ylim(0, 1)
85 +
86 +     legend = ax1.legend(loc='upper center')
87 +
88 +     cmap = discrete_cmap(len(routes) + 2, 'nipy_spectral')
89 +     dem_rects = []
90 +     used_rects = []
91 +     cap_rects = []
92 +     qvs = []
93 +     total_dist = 0
94 +     for veh_number, r in enumerate(routes):
95 +         color = cmap(len(routes) - veh_number) # Invert to have in rainbow order
96 +
97 +         route_demands = demands[r - 1]
98 +         coords = locs[r - 1, :]
99 +         xs, ys = coords.transpose()
100 +
101 +         total_route_demand = sum(route_demands)
102 +         # assert total_route_demand <= capacity
103 +         if not visualize_demands:
104 +             ax1.plot(xs, ys, 'o', mfc=color, markersize=markersize, markeredgewidth=0.0)
105 +
106 +         dist = 0
107 +         x_prev, y_prev = x_dep, y_dep
108 +         cum_demand = 0
109 +         for (x, y), d in zip(coords, route_demands):
110 +             dist += np.sqrt((x - x_prev) ** 2 + (y - y_prev) ** 2)
111 +
112 +             cap_rects.append(Rectangle((x, y), 0.01, 0.1))
113 +             used_rects.append(Rectangle((x, y), 0.01, 0.1 * total_route_demand / capacity))
114 +             dem_rects.append(Rectangle((x, y + 0.1 * cum_demand / capacity), 0.01, 0.1 * d / capacity))
115 +
116 +             x_prev, y_prev = x, y
117 +             cum_demand += d
118 +
119 +         dist += np.sqrt((x_dep - x_prev) ** 2 + (y_dep - y_prev) ** 2)
120 +         total_dist += dist
121 +         qv = ax1.quiver(
122 +             xs[:-1],
123 +             ys[:-1],
124 +             xs[1:] - xs[:-1],
125 +             ys[1:] - ys[:-1],
126 +             scale_units='xy',

```

```

127 +         angles='xy',
128 +         scale=1,
129 +         color=color,
130 +         label='R{ }, # { }, c { } / { }, d {:.2f}'.format(
131 +             veh_number,
132 +             len(r),
133 +             int(total_route_demand) if round_demand else total_route_demand,
134 +             int(capacity) if round_demand else capacity,
135 +             dist
136 +         )
137 +     )
138 +
139 +     qvs.append(qv)
140 +
141 +     ax1.set_title('{} routes, total distance {:.2f}'.format(len(routes), total_dist))
142 +     ax1.legend(handles=qvs)
143 +
144 +     pc_cap = PatchCollection(cap_rects, facecolor='whitesmoke', alpha=1.0, edgecolor='lightgray')
145 +     pc_used = PatchCollection(used_rects, facecolor='lightgray', alpha=1.0, edgecolor='lightgray')
146 +     pc_dem = PatchCollection(dem_rects, facecolor='black', alpha=1.0, edgecolor='black')
147 +
148 +     if visualize_demands:
149 +         ax1.add_collection(pc_cap)
150 +         ax1.add_collection(pc_used)
151 +         ax1.add_collection(pc_dem)
152 +
153 +
154 + # In[4]:
155 +
156 +
157 + model1, _ = load_model('pretrained/cvrp_100/')
158 + model2, _ = load_model('pretrained/tsp_100/')
159 +
160 + torch.manual_seed(1234)
161 + dataset = CVRP.make_dataset(size=100, num_samples=10)
162 +
163 +
164 + # In[5]:
165 +
166 +
167 + # Need a dataloader to batch instances
168 + dataloader = DataLoader(dataset, batch_size=1000)
169 +
170 + # Make var works for dicts
171 + batch = next(iter(dataloader))
172 +
173 + # Run the model
174 + model1.eval()
175 + model1.set_decode_type('greedy')
176 + model2.eval()
177 + model2.set_decode_type('greedy')
178 +
179 +
180 +
181 + with torch.no_grad():
182 +     length1, log_p1, pi1 = model1(batch, return_pi=True)
183 +
184 +     tours = pi1
185 +
186 +     for i, (data, tour) in enumerate(zip(dataset, tours)):
187 +
188 +         fig, ax = plt.subplots(figsize=(10, 10))

```



```

189 + plot_vehicle_routes(data, tour, ax, visualize_demands=False, demand_scale=50, round_demand=True)
190 + fig.savefig(os.path.join('images', 'cvrp_{}.png'.format(i)))
191 +
192 + for i, (data, tour) in enumerate(zip(dataset, tours)): #written by Isabelle Akian
193 +     newcoords, new_routes = get_routes_and_coords(data, tour)
194 +     newroutes = []
195 +     for k in newcoords:
196 +         with torch.no_grad():
197 +
198 +             length2, log_p2, pi2 = model2(torch.FloatTensor([k]), return_pi=True)
199 +
200 +             tour2 = pi2.tolist()[0]
201 +             for j in range(len(tour2)):
202 +                 tour2[j] = coord_to_loc(data, k[j])
203 +
204 +             newroutes = newroutes + tour2
205 +             newroutes.append(0)
206 +             fig, ax = plt.subplots(figsize=(10, 10))
207 +
208 +             plot_vehicle_routes(data, torch.IntTensor(newroutes), ax, visualize_demands=False, demand_scale=50, round_demand=True)
209 +             fig.savefig(os.path.join('images', 'cvrp_and_tsp_{}.png'.format(i)))
210 +
211 +
212 +
213 +
214 + # In[ ]:
215 +
216 +
217 +
218 +

```

5 problems/tsp/problem\_tsp.py

```

54 54
55 55     class TSPDataset(Dataset):
56 56
57 57     - def __init__(self, filename=None, size=50, num_samples=1000000, offset=0, distribution=None):
57 57 +     def __init__(self, filename=None, size=50, num_samples=1000000, offset=0, distribution=None, inputdata=[]):
58 58         super(TSPDataset, self).__init__()
59 59
60 60         self.data_set = []
64 64         with open(filename, 'rb') as f:
65 65             data = pickle.load(f)
66 66             self.data = [torch.FloatTensor(row) for row in (data[offset:offset+num_samples])]
67 67 +             print(self.data)
68 68 +             elif len(inputdata) > 0: #written by Isabelle Akian
69 69 +                 self.data = [torch.FloatTensor(row, 2) for row in (inputdata[offset:offset+num_samples])]
67 70     else:
68 71         # Sample points randomly in [0, 1] square
69 72         self.data = [torch.FloatTensor(size, 2).uniform_(0, 1) for i in range(num_samples)]

```

115 readfiles/README(Kool).md

```

...  ... @@ -0,0 +1,115 @@
1 + # Attention, Learn to Solve Routing Problems!
2 +
3 + Attention based model for learning to solve the Travelling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP),
   Orienteering Problem (OP) and (Stochastic) Prize Collecting TSP (PCTSP). Training with REINFORCE with greedy rollout
   baseline.
4 +

```

```

5 + ![TSP100](../images/tsp.gif)
6 +
7 + ## Paper
8 + For more details, please see our paper [Attention, Learn to Solve Routing Problems!](https://openreview.net/forum?id=ByxBFsRqYm) which has been accepted at [ICLR 2019](https://iclr.cc/Conferences/2019). If this code is useful for your work, please cite our paper:
9 +
10 + ```
11 + @inproceedings{
12 +     kool2018attention,
13 +     title={Attention, Learn to Solve Routing Problems!},
14 +     author={Wouter Kool and Herke van Hoof and Max Welling},
15 +     booktitle={International Conference on Learning Representations},
16 +     year={2019},
17 +     url={https://openreview.net/forum?id=ByxBFsRqYm},
18 + }
19 + ```
20 +
21 + ## Dependencies
22 +
23 + * Python>=3.8
24 + * NumPy
25 + * SciPy
26 + * [PyTorch](http://pytorch.org/)>=1.7
27 + * tqdm
28 + * [tensorboard_logger](https://github.com/TeamHG-Memex/tensorboard_logger)
29 + * Matplotlib (optional, only for plotting)
30 +
31 + ## Quick start
32 +
33 + For training TSP instances with 20 nodes and using rollout as REINFORCE baseline:
34 + ```bash
35 + python run.py --graph_size 20 --baseline rollout --run_name 'tsp20_rollout'
36 + ```
37 +
38 + ## Usage
39 +
40 + ### Generating data
41 +
42 + Training data is generated on the fly. To generate validation and test data (same as used in the paper) for all problems:
43 + ```bash
44 + python generate_data.py --problem all --name validation --seed 4321
45 + python generate_data.py --problem all --name test --seed 1234
46 + ```
47 +
48 + ### Training
49 +
50 + For training TSP instances with 20 nodes and using rollout as REINFORCE baseline and using the generated validation set:
51 + ```bash
52 + python run.py --graph_size 20 --baseline rollout --run_name 'tsp20_rollout' --val_dataset
    data/tsp/tsp20_validation_seed4321.pkl
53 + ```
54 +
55 + #### Multiple GPUs
56 + By default, training will happen on all available GPUs. To disable CUDA at all, add the flag `--no_cuda`.
57 + Set the environment variable `CUDA_VISIBLE_DEVICES` to only use specific GPUs:
58 + ```bash
59 + CUDA_VISIBLE_DEVICES=2,3 python run.py
60 + ```
61 + Note that using multiple GPUs has limited efficiency for small problem sizes (up to 50 nodes).
62 +

```

```

63 + #### Warm start
64 + You can initialize a run using a pretrained model by using the `--load_path` option:
65 + ```bash
66 + python run.py --graph_size 100 --load_path pretrained/tsp_100/epoch-99.pt
67 + ```
68 +
69 + The `--load_path` option can also be used to load an earlier run, in which case also the optimizer state will be loaded:
70 + ```bash
71 + python run.py --graph_size 20 --load_path 'outputs/tsp_20/tsp20_rollout_{datetime}/epoch-0.pt'
72 + ```
73 +
74 + The `--resume` option can be used instead of the `--load_path` option, which will try to resume the run, e.g. load
    additionally the baseline state, set the current epoch/step counter and set the random number generator state.
75 +
76 + ### Evaluation
77 + To evaluate a model, you can add the `--eval-only` flag to `run.py`, or use `eval.py`, which will additionally measure
    timing and save the results:
78 + ```bash
79 + python eval.py data/tsp/tsp20_test_seed1234.pkl --model pretrained/tsp_20 --decode_strategy greedy
80 + ```
81 + If the epoch is not specified, by default the last one in the folder will be used.
82 +
83 + #### Sampling
84 + To report the best of 1280 sampled solutions, use
85 + ```bash
86 + python eval.py data/tsp/tsp20_test_seed1234.pkl --model pretrained/tsp_20 --decode_strategy sample --width 1280 --
    eval_batch_size 1
87 + ```
88 + Beam Search (not in the paper) is also recently added and can be used using `--decode_strategy bs --width {beam_size}`.
89 +
90 + #### To run baselines
91 + Baselines for different problems are within the corresponding folders and can be ran (on multiple datasets at once) as
    follows
92 + ```bash
93 + python -m problems.tsp.tsp_baseline farthest_insertion data/tsp/tsp20_test_seed1234.pkl data/tsp/tsp50_test_seed1234.pkl
    data/tsp/tsp100_test_seed1234.pkl
94 + ```
95 + To run baselines, you need to install [Compass](https://github.com/bcamath-ds/compass) by running the
    `install_compass.sh` script from within the `problems/op` directory and [Concorde]
    (http://www.math.uwaterloo.ca/tsp/concorde.html) using the `install_concorde.sh` script from within `problems/tsp`.
    [LKH3](http://akira.ruc.dk/~keld/research/LKH-3/) should be automatically downloaded and installed when required. To use
    [Gurobi](http://www.gurobi.com), obtain a ([free academic](http://www.gurobi.com/registration/academic-license-reg))
    license and follow the [installation instructions]
    (https://www.gurobi.com/documentation/8.1/quickstart_windows/installing_the_anaconda_py.html).
96 +
97 + ### Other options and help
98 + ```bash
99 + python run.py -h
100 + python eval.py -h
101 + ```
102 +
103 + ### Example CVRP solution
104 + See `plot_vrp.ipynb` for an example of loading a pretrained model and plotting the result for Capacitated VRP with 100
    nodes.
105 +
106 + ![CVRP100](../images/cvrp_0.png)
107 +
108 + ## Acknowledgements
109 + Thanks to [pemami4911/neural-combinatorial-r1-pytorch](https://github.com/pemami4911/neural-combinatorial-r1-pytorch)
    for getting me started with the code for the Pointer Network.
110 +
111 + This repository includes adaptations of the following repositories as baselines:


```

```
112 + * https://github.com/MichelDeudon/encode-attend-navigate
113 + * https://github.com/mc-ride/orienteering
114 + * https://github.com/jordanamecler/PCTSP
115 + * https://github.com/rafael2reis/salesman
```


▼ 5,050 █████ readfiles/differences.diff 

[Load diff](#)

Large diffs are not rendered by default.

▼ BIN +681 KB readfiles/paper.pdf 

Binary file not shown.

▼ 233 █████ simple\_tsp.py 

```
...      ...      @@ -0,0 +1,233 @@
1      + #!/usr/bin/env python
2      + # coding: utf-8
3      +
4      + # In[1]:
5      +
6      +
7      + import os
8      + import numpy as np
9      + import torch
10     +
11     +
12     + # In[2]:
13     +
14     +
15     + from utils import load_model
16     + model, _ = load_model('pretrained/tsp_100/')
17     + model.eval() # Put in evaluation mode to not track gradients
18     +
19     + xy = np.random.rand(100, 2)
20     + # print(xy)
21     +
22     + def make_oracle(model, xy, temperature=1.0):
23     +
24     +     num_nodes = len(xy)
25     +
26     +     xyt = torch.tensor(xy).float()[None] # Add batch dimension
27     +
28     +     with torch.no_grad(): # Inference only
29     +         embeddings, _ = model.embedder(model._init_embed(xyt))
30     +
31     +         # Compute keys, values for the glimpse and keys for the logits once as they can be reused in every step
32     +         fixed = model._precompute(embeddings)
33     +
34     +     def oracle(tour):
35     +         with torch.no_grad(): # Inference only
36     +             # Input tour with 0 based indices
37     +             # Output vector with probabilities for locations not in tour
38     +             tour = torch.tensor(tour).long()
39
```

```

+         if len(tour) == 0:
40 +             step_context = model.W_placeholder
41 +         else:
42 +             step_context = torch.cat((embeddings[0, tour[0]], embeddings[0, tour[-1]]), -1)
43 +
44 +         # Compute query = context node embedding, add batch and step dimensions (both 1)
45 +         query = fixed.context_node_projected + model.project_step_context(step_context[None, None, :])
46 +
47 +         # Create the mask and convert to bool depending on PyTorch version
48 +         mask = torch.zeros(num_nodes, dtype=torch.uint8) > 0
49 +         mask[tour] = 1
50 +         mask = mask[None, None, :] # Add batch and step dimension
51 +
52 +         log_p, _ = model._one_to_many_logits(query, fixed.glimpse_key, fixed.glimpse_val, fixed.logit_key, mask)
53 +         p = torch.softmax(log_p / temperature, -1)[0, 0]
54 +         assert (p[tour] == 0).all()
55 +         assert (p.sum() - 1).abs() < 1e-5
56 +         #assert np.allclose(p.sum().item(), 1)
57 +         return p.numpy()
58 +
59 +     return oracle
60 +
61 +
62 +
63 + oracle = make_oracle(model, xy)
64 +
65 + sample = False
66 + tour = []
67 + tour_p = []
68 + while(len(tour) < len(xy)):
69 +     p = oracle(tour)
70 +
71 +     if sample:
72 +         # Advertising the Gumbel-Max trick
73 +         g = -np.log(-np.log(np.random.rand(*p.shape)))
74 +         i = np.argmax(np.log(p) + g)
75 +         # i = np.random.multinomial(1, p)
76 +     else:
77 +         # Greedy
78 +         i = np.argmax(p)
79 +         tour.append(i)
80 +         tour_p.append(p)
81 +
82 + print(tour)
83 +
84 +
85 + # In[3]:
86 +
87 +
88 + # get_ipython().run_line_magic('matplotlib', 'inline')
89 + from matplotlib import pyplot as plt
90 +
91 + from matplotlib.collections import PatchCollection
92 + from matplotlib.patches import Rectangle
93 + from matplotlib.lines import Line2D
94 +
95 + # Code inspired by Google OR Tools plot:
96 + # https://github.com/google/or-tools/blob/fb12c5ded7423d524fc6c95656a9bdc290a81d4d/examples/python/cvrptw_plot.py
97 +
98 + def plot_tsp(xy, tour, ax1):
99 +     """
100 +     Plot the TSP tour on matplotlib axis ax1.

```

```

101 +     """
102 +
103 +     ax1.set_xlim(0, 1)
104 +     ax1.set_ylim(0, 1)
105 +
106 +     xs, ys = xy[tour].transpose()
107 +     xs, ys = xy[tour].transpose()
108 +     dx = np.roll(xs, -1) - xs
109 +     dy = np.roll(ys, -1) - ys
110 +     d = np.sqrt(dx * dx + dy * dy)
111 +     lengths = d.cumsum()
112 +
113 +     # Scatter nodes
114 +     ax1.scatter(xs, ys, s=40, color='blue')
115 +     # Starting node
116 +     ax1.scatter([xs[0]], [ys[0]], s=100, color='red')
117 +
118 +     # Arcs
119 +     qv = ax1.quiver(
120 +         xs, ys, dx, dy,
121 +         scale_units='xy',
122 +         angles='xy',
123 +         scale=1,
124 +     )
125 +
126 +     ax1.set_title('{} nodes, total length {:.2f}'.format(len(tour), lengths[-1]))
127 +
128 + fig, ax = plt.subplots(figsize=(10, 10))
129 + plot_tsp(xy, tour, ax)
130 +
131 +
132 + # In[4]:
133 +
134 +
135 + from matplotlib.collections import PatchCollection
136 + from matplotlib.patches import Rectangle
137 + from matplotlib.animation import PillowWriter
138 + from matplotlib.lines import Line2D
139 + from IPython.display import HTML
140 +
141 + from celluloid import Camera # pip install celluloid
142 +
143 + def format_prob(prob):
144 +     return '{:.6f}' if prob > 1e-5 else '{:.2E}'.format(prob)
145 +
146 + def plot_tsp_ani(xy, tour, tour_p=None, max_steps=1000):
147 +     n = len(tour)
148 +     fig, ax1 = plt.subplots(figsize=(10, 10))
149 +     xs, ys = xy[tour].transpose()
150 +     dx = np.roll(xs, -1) - xs
151 +     dy = np.roll(ys, -1) - ys
152 +     d = np.sqrt(dx * dx + dy * dy)
153 +     lengths = d.cumsum()
154 +
155 +     ax1.set_xlim(0, 1)
156 +     ax1.set_ylim(0, 1)
157 +
158 +     camera = Camera(fig)
159 +
160 +     total_length = 0
161 +     cum_log_prob = 0
162 +     for i in range(n + 1):

```

```

163 +         for plot_probs in [False] if tour_p is None or i >= n else [False, True]:
164 +             # Title
165 +             title = 'Nodes: {:3d}, length: {:.4f}, prob: {}'.format(
166 +                 i, lengths[i - 2] if i > 1 else 0., format_prob(np.exp(cum_log_prob))
167 +             )
168 +             ax1.text(0.6, 0.97, title, transform=ax.transAxes)
169 +
170 +             # First print current node and next candidates
171 +             ax1.scatter(xs, ys, s=40, color='blue')
172 +
173 +             if i > 0:
174 +                 ax1.scatter([xs[i - 1]], [ys[i - 1]], s=100, color='red')
175 +             if i > 1:
176 +                 qv = ax1.quiver(
177 +                     xs[:i-1],
178 +                     ys[:i-1],
179 +                     dx[:i-1],
180 +                     dy[:i-1],
181 +                     scale_units='xy',
182 +                     angles='xy',
183 +                     scale=1,
184 +                 )
185 +                 if plot_probs:
186 +                     prob_rects = [Rectangle((x, y), 0.01, 0.1 * p) for (x, y), p in zip(xy, tour_p[i]) if p > 0.01]
187 +                     pc = PatchCollection(prob_rects, facecolor='lightgray', alpha=1.0, edgecolor='lightgray')
188 +                     ax1.add_collection(pc)
189 +                     camera.snap()
190 +                 if i < n and tour_p is not None:
191 +                     # Add cumulative_probability
192 +                     cum_log_prob += np.log(tour_p[i][tour[i]])
193 +                 if i > max_steps:
194 +                     break
195 +
196 +             # Plot final tour
197 +             # Scatter nodes
198 +             ax1.scatter(xs, ys, s=40, color='blue')
199 +             # Starting node
200 +             ax1.scatter([xs[0]], [ys[0]], s=100, color='red')
201 +
202 +             # Arcs
203 +             qv = ax1.quiver(
204 +                 xs, ys, dx, dy,
205 +                 scale_units='xy',
206 +                 angles='xy',
207 +                 scale=1,
208 +             )
209 +             if tour_p is not None:
210 +                 # Note this does not use stable logsumexp trick
211 +                 cum_log_prob = format_prob(np.exp(sum([np.log(p[node]) for node, p in zip(tour, tour_p)])))
212 +             else:
213 +                 cum_log_prob = '?'
214 +             ax1.set_title('{} nodes, total length {:.4f}, prob: {}'.format(len(tour), lengths[-1], cum_log_prob))
215 +
216 +             camera.snap()
217 +
218 +             return camera
219 +
220 +
221 + animation = plot_tsp_ani(xy, tour, tour_p).animate(interval=500)
222 + writer = PillowWriter(fps=2)
223 + # ani.save("demo_sine.gif", writer=writer)
224 + animation.save('images/tsp_adjusted.gif', writer='writer', fps=2) # requires imagemagick

```

```

225 + # compress by running 'convert tsp.gif -strip -coalesce -layers Optimize tsp.gif'
226 + # HTML(animation.to_html5_video()) # requires ffmpeg
227 +
228 +
229 + # In[ ]:
230 +
231 +
232 +
233 +

```

▼ 230 ■■■■ tsp\_applied.py 

```

...
...  @@@ -0,0 +1,230 @@
1  + #!/usr/bin/env python
2  + # coding: utf-8
3  +
4  + # In[1]:
5  +
6  +
7  + import os
8  + import numpy as np
9  + import torch
10 +
11 + # In[2]:
12 +
13 +
14 + from utils import load_model
15 +
16 + model, _ = load_model('pretrained/tsp_100/')
17 + model.eval() # Put in evaluation mode to not track gradients
18 +
19 + xy = np.random.rand(100, 2)
20 +
21 +
22 + def make_oracle(model, xy, temperature=1.0):
23 +     num_nodes = len(xy)
24 +
25 +     xyt = torch.tensor(xy).float()[None] # Add batch dimension
26 +
27 +     with torch.no_grad(): # Inference only
28 +         embeddings, _ = model.embedder(model._init_embed(xyt))
29 +
30 +         # Compute keys, values for the glimpse and keys for the logits once as they can be reused in every step
31 +         fixed = model._precompute(embeddings)
32 +
33 +     def oracle(tour):
34 +         with torch.no_grad(): # Inference only
35 +             # Input tour with 0 based indices
36 +             # Output vector with probabilities for locations not in tour
37 +             tour = torch.tensor(tour).long()
38 +             if len(tour) == 0:
39 +                 step_context = model.W_placeholder
40 +             else:
41 +                 step_context = torch.cat((embeddings[0, tour[0]], embeddings[0, tour[-1]]), -1)
42 +
43 +             # Compute query = context node embedding, add batch and step dimensions (both 1)
44 +             query = fixed.context_node_projected + model.project_step_context(step_context[None, None, :])
45 +
46 +             # Create the mask and convert to bool depending on PyTorch version
47 +             mask = torch.zeros(num_nodes, dtype=torch.uint8) > 0
48 +             mask[tour] = 1

```



```

49 +         mask = mask[None, None, :] # Add batch and step dimension
50 +
51 +         log_p, _ = model._one_to_many_logits(query, fixed.glimpse_key, fixed.glimpse_val, fixed.logit_key, mask)
52 +         p = torch.softmax(log_p / temperature, -1)[0, 0]
53 +         assert (p[tour] == 0).all()
54 +         assert (p.sum() - 1).abs() < 1e-5
55 +         # assert np.allclose(p.sum().item(), 1)
56 +         return p.numpy()
57 +
58 +     return oracle
59 +
60 +
61 + oracle = make_oracle(model, xy)
62 +
63 + sample = False
64 + tour = []
65 + tour_p = []
66 + while (len(tour) < len(xy)):
67 +     p = oracle(tour)
68 +
69 +     if sample:
70 +         # Advertising the Gumbel-Max trick
71 +         g = -np.log(-np.log(np.random.rand(*p.shape)))
72 +         i = np.argmax(np.log(p) + g)
73 +         # i = np.random.multinomial(1, p)
74 +     else:
75 +         # Greedy
76 +         i = np.argmax(p)
77 +         tour.append(i)
78 +         tour_p.append(p)
79 +     print(xy)
80 +     print(tour)
81 +
82 + # In[3]:
83 +
84 +
85 + # get_ipython().run_line_magic('matplotlib', 'inline')
86 + from matplotlib import pyplot as plt
87 +
88 + from matplotlib.collections import PatchCollection
89 + from matplotlib.patches import Rectangle
90 + from matplotlib.lines import Line2D
91 +
92 +
93 + # Code inspired by Google OR Tools plot:
94 + # https://github.com/google/or-tools/blob/fb12c5ded7423d524fc6c95656a9bdc290a81d4d/examples/python/cvrptw_plot.py
95 +
96 + def plot_tsp(xy, tour, ax1):
97 +     """
98 +     Plot the TSP tour on matplotlib axis ax1.
99 +     """
100 +
101 +     ax1.set_xlim(0, 1)
102 +     ax1.set_ylim(0, 1)
103 +
104 +     xs, ys = xy[tour].transpose()
105 +     xs, ys = xy[tour].transpose()
106 +     dx = np.roll(xs, -1) - xs
107 +     dy = np.roll(ys, -1) - ys
108 +     d = np.sqrt(dx * dx + dy * dy)
109 +     lengths = d.cumsum()
110 +

```

```

111 + # Scatter nodes
112 + ax1.scatter(xs, ys, s=40, color='blue')
113 + # Starting node
114 + ax1.scatter([xs[0]], [ys[0]], s=100, color='red')
115 +
116 + # Arcs
117 + qv = ax1.quiver(
118 +     xs, ys, dx, dy,
119 +     scale_units='xy',
120 +     angles='xy',
121 +     scale=1,
122 + )
123 +
124 + ax1.set_title('{} nodes, total length {:.2f}'.format(len(tour), lengths[-1]))
125 +
126 +
127 + fig, ax = plt.subplots(figsize=(10, 10))
128 + plot_tsp(xy, tour, ax)
129 +
130 + # In[4]:
131 +
132 +
133 + from matplotlib.collections import PatchCollection
134 + from matplotlib.patches import Rectangle
135 + from matplotlib.lines import Line2D
136 + from IPython.display import HTML
137 +
138 + from celluloid import Camera # pip install celluloid
139 +
140 +
141 + def format_prob(prob):
142 +     return ('{:.6f}' if prob > 1e-5 else '{:.2E}').format(prob)
143 +
144 +
145 + def plot_tsp_ani(xy, tour, tour_p=None, max_steps=1000):
146 +     n = len(tour)
147 +     fig, ax1 = plt.subplots(figsize=(10, 10))
148 +     xs, ys = xy[tour].transpose()
149 +     dx = np.roll(xs, -1) - xs
150 +     dy = np.roll(ys, -1) - ys
151 +     d = np.sqrt(dx * dx + dy * dy)
152 +     lengths = d.cumsum()
153 +
154 +     ax1.set_xlim(0, 1)
155 +     ax1.set_ylim(0, 1)
156 +
157 +     camera = Camera(fig)
158 +
159 +     total_length = 0
160 +     cum_log_prob = 0
161 +     for i in range(n + 1):
162 +         for plot_probs in [False] if tour_p is None or i >= n else [False, True]:
163 +             # Title
164 +             title = 'Nodes: {:3d}, length: {:.4f}, prob: {}'.format(
165 +                 i, lengths[i - 2] if i > 1 else 0., format_prob(np.exp(cum_log_prob)))
166 +             )
167 +             ax1.text(0.6, 0.97, title, transform=ax.transAxes)
168 +
169 +             # First print current node and next candidates
170 +             ax1.scatter(xs, ys, s=40, color='blue')
171 +
172 +             if i > 0:

```

```

173 +         ax1.scatter([xs[i - 1]], [ys[i - 1]], s=100, color='red')
174 +
175 +         if i > 1:
176 +             qv = ax1.quiver(
177 +                 xs[:i - 1],
178 +                 ys[:i - 1],
179 +                 dx[:i - 1],
180 +                 dy[:i - 1],
181 +                 scale_units='xy',
182 +                 angles='xy',
183 +                 scale=1,
184 +             )
185 +             if plot_probs:
186 +                 prob_rects = [Rectangle((x, y), 0.01, 0.1 * p) for (x, y), p in zip(xy, tour_p[i]) if p > 0.01]
187 +                 pc = PatchCollection(prob_rects, facecolor='lightgray', alpha=1.0, edgecolor='lightgray')
188 +                 ax1.add_collection(pc)
189 +                 camera.snap()
190 +             if i < n and tour_p is not None:
191 +                 # Add cumulative_probability
192 +                 cum_log_prob += np.log(tour_p[i][tour[i]])
193 +             if i > max_steps:
194 +                 break
195 +
196 +         # Plot final tour
197 +         # Scatter nodes
198 +         ax1.scatter(xs, ys, s=40, color='blue')
199 +         # Starting node
200 +         ax1.scatter([xs[0]], [ys[0]], s=100, color='red')
201 +
202 +         # Arcs
203 +         qv = ax1.quiver(
204 +             xs, ys, dx, dy,
205 +             scale_units='xy',
206 +             angles='xy',
207 +             scale=1,
208 +         )
209 +         if tour_p is not None:
210 +             # Note this does not use stable logsumexp trick
211 +             cum_log_prob = format_prob(np.exp(sum([np.log(p[node]) for node, p in zip(tour, tour_p)])))
212 +         else:
213 +             cum_log_prob = '?'
214 +         ax1.set_title('{} nodes, total length {:.4f}, prob: {}'.format(len(tour), lengths[-1], cum_log_prob))
215 +
216 +         camera.snap()
217 +
218 +         return camera
219 +
220 + animation = plot_tsp_ani(xy, tour, tour_p).animate(interval=500)
221 + # animation.save('images/tsp.gif', writer='imagemagick', fps=2) # requires imagemagick
222 + # compress by running 'convert tsp.gif -strip -coalesce -layers Optimize tsp.gif'
223 + # HTML(animation.to_html5_video()) # requires ffmpeg
224 +
225 +
226 + # In[ ]:
227 +
228 +
229 +
230 +

```