Showing 30 changed files with 5,898 additions and 93 deletions.

Split Unified

140 README.md

```diff
@@ -1,22 +1,19 @@
- # Attention, Learn to Solve Routing Problems!
-
- 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.
-
- ![TSP100](images/tsp.gif)
+ # Analyzing the Vehicle Route Problem: A Heuristic Approach to Route Planning and Optimization
+
+ ## Introduction
+ This model incorporates the attention based model from the paper [Attention, Learn to Solve Routing Problems!](https://openreview.net/forum?id=ByxBFsRqYm) which was accepted at [ICLR 2019](https://iclr.cc/Conferences/2019) 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.
+ <br />
+ <br />
+ The paper describing this specific extension of the aforementioned model is from the paper [Analyzing the Vehicle Route Problem: A Heuristic Approach to Route Planning and Optimization](readfiles/paper.pdf)
+ <br />
+ <br />
+ Note: All files linked in this README are accessible in the following locations:
+ * Images are in [images/](images)
+ * Python scripts are in the root directory
+ * All other filetypes (PDF, diff, etc.) are in [readfiles/](readfiles)

  ## Paper
- 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:
-
- ```
- @inproceedings{
-     kool2018attention,
-     title={Attention, Learn to Solve Routing Problems!},
-     author={Wouter Kool and Herke van Hoof and Max Welling},
-     booktitle={International Conference on Learning Representations},
-     year={2019},
-     url={https://openreview.net/forum?id=ByxBFsRqYm},
- }
- ```
+ For more details, please see our paper [Analyzing the Vehicle Route Problem: A Heuristic Approach to Route Planning and Optimization](readfiles/paper.pdf).

  ## Dependencies

  * [tensorboard_logger](https://github.com/TeamHG-Memex/tensorboard_logger)
  * Matplotlib (optional, only for plotting)

- ## Quick start
-
- For training TSP instances with 20 nodes and using rollout as REINFORCE baseline:
- ```bash
- python run.py --graph_size 20 --baseline rollout --run_name 'tsp20_rollout'
+ ## Differences
+ To view the differences between the implementation in this codebase, reference [this document]() to view pdf that will
+ compare the original code with the added code. The greatest changes were made in [plot_vrp.py](plot_vrp.py),
```

```diff
  [simple_tsp.py](simple_tsp.py),
+ as well as smaller changes and additions in other locations. Most of the more significant changes that were made by us have a comment
+ somewhere near the top indicating that it is original code, but not all changes are indicated in this way.<br />
+ <br />
+ A `.diff` file is also available [here](readfiles/differences.diff)
+
+ ## Running the code
+ To run the code that demonstrates the implementation described in the paper (which as noted, did not make any real changes to the
+ sample routes that were being utilized), run the following command in the root folder:
  ```
-
- ## Usage
-
- ### Generating data
-
- Training data is generated on the fly. To generate validation and test data (same as used in the paper) for all problems:

-
  ```bash
- python generate_data.py --problem all --name validation --seed 4321
- python generate_data.py --problem all --name test --seed 1234
+ python plot_vrp.py
  ```
+ The output will be two sets of ten images, one which runs the Capacitated Vehicle Routing Problem (CVRP) ```cvrp_100``` model on
+ a graph of 100 nodes, and one which runs this as well as the Traveling Salesman Problem (TSP) ```tsp_100``` model on each
+ outputted route from the ```cvrp_100``` model. An example is shown below:
+
+ CVRP:
+ ![CVRP100](images/cvrp_0.png)

- ### Training
-
- For training TSP instances with 20 nodes and using rollout as REINFORCE baseline and using the generated validation set:

-
  ```bash
- python run.py --graph_size 20 --baseline rollout --run_name 'tsp20_rollout' --val_dataset data/tsp/tsp20_validation_seed4321.pkl
- ```
+ CVRP + TSP on routes:
+ ![CVRPTSP100](images/cvrp_and_tsp_0.png)

- #### Multiple GPUs
- By default, training will happen *on all available GPUs*. To disable CUDA at all, add the flag `--no_cuda`.
- Set the environment variable `CUDA_VISIBLE_DEVICES` to only use specific GPUs:
- ```bash
- CUDA_VISIBLE_DEVICES=2,3 python run.py 
- ```
- Note that using multiple GPUs has limited efficiency for small problem sizes (up to 50 nodes).
+ Observing the two images it is clear that each route is identical, and the distance for each 
+ route as well as the total distance did not change.<br />
+ <br />
+ 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 
+ versions of the TSP model that is overlayed by following the steps from [the original README file](readfiles/README(Kool).md) which allow
+ users to generate and train models with specific parameters. If the amount of graph nodes is changed for the new model, this 
+ must also be adjusted in the [plot_vrp.py](plot_vrp.py) script. 

- #### Warm start
```

```diff
- You can initialize a run using a pretrained model by using the `--load_path` option:
- ```bash
- python run.py --graph_size 100 --load_path pretrained/tsp_100/epoch-99.pt
- ```
-
- The `--load_path` option can also be used to load an earlier run, in which case also the optimizer state will be loaded:
- ```bash
- python run.py --graph_size 20 --load_path 'outputs/tsp_20/tsp20_rollout_{datetime}/epoch-0.pt'
- ```
-
- 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.
-
- ### Evaluation
- 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:
- ```bash
- python eval.py data/tsp/tsp20_test_seed1234.pkl --model pretrained/tsp_20 --decode_strategy greedy
- ```
- If the epoch is not specified, by default the last one in the folder will be used.
-
- #### Sampling
- To report the best of 1280 sampled solutions, use
- ```bash
- python eval.py data/tsp/tsp20_test_seed1234.pkl --model pretrained/tsp_20 --decode_strategy sample --width 1280 --eval_batch_size 1
- ```
- Beam Search (not in the paper) is also recently added and can be used using `--decode_strategy bs --width {beam_size}`.
-
- #### To run baselines
- Baselines for different problems are within the corresponding folders and can be ran (on multiple datasets at once) as follows
- ```bash
- 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
- ```
- 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).
-
- ### Other options and help
- ```bash
- python run.py -h
- python eval.py -h
- ```
+ ## Slight improvements with Beam Search

- ### Example CVRP solution
- See `plot_vrp.ipynb` for an example of loading a pretrained model and plotting the result for Capacitated VRP with 100 nodes.
+ The [run.py](run.py) file has the ability to train models using Beam Search, and this slightly improves the CVRP model as seen
+ in this image below, which is the output after creating and training a CVRP model for a graph of 20 nodes.<br />
+ ![beam](images/Beam Search.jpg)
+ The average cost for a route using a model with beam search is slightly less than it would be when using a model without.

- ![CVRP100](images/cvrp_0.png)
```

```diff
 
 ## Acknowledgements
- 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.
 
- This repository includes adaptions of the following repositories as baselines:
- * https://github.com/MichelDeudon/encode-attend-navigate
- * https://github.com/mc-ride/orienteering
- * https://github.com/jordanamecler/PCTSP
- * https://github.com/rafael2reis/salesman
+ This repository includes adaptions of the following repository as a baseline:
+ * 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 @@
+ #!/usr/bin/env python
+ # coding: utf-8
```

```
+
+ # In[1]:
+
+
+ import os
+ import numpy as np
+ import torch
+ # import simple_tsp
+
+
+
+ # In[2]:
+
+
+ from torch.utils.data import DataLoader
+
+ # import simple_tsp
+ from generate_data import generate_vrp_data
+ from utils import load_model
+ from problems import CVRP, TSP
+
+ # In[3]:
+
+
+ # get_ipython().run_line_magic('matplotlib', 'inline')
+ from matplotlib import pyplot as plt
+
+ from matplotlib.collections import PatchCollection
+ from matplotlib.patches import Rectangle
+ from matplotlib.lines import Line2D
+
+ # Code inspired by Google OR Tools plot:
+ # https://github.com/google/or-tools/blob/fb12c5ded7423d524fc6c95656a9bdc290a81d4d/examples/python/cvrptw_plot.py
+
+ def discrete_cmap(N, base_cmap=None):
+   """
+     Create an N-bin discrete colormap from the specified input map
+     """
+   # Note that if base_cmap is a string or None, you can simply do
+   #    return plt.cm.get_cmap(base_cmap, N)
+   # The following works for string, None, or a colormap instance:
+
+   base = plt.cm.get_cmap(base_cmap)
+   color_list = base(np.linspace(0, 1, N))
+   cmap_name = base.name + str(N)
+   return base.from_list(cmap_name, color_list, N)
+
+
+ def coord_to_loc(dataset, coord): #written by Isabelle Akian
+     locs = dataset['loc'].cpu().numpy()
+
+     return np.where(locs == coord)[0][0]+1
+
+
+ def get_routes_and_coords(datasets, route_for_coord):#written by Isabelle Akian
+     route_list = [r[r != 0] for r in np.split(route_for_coord.cpu().numpy(), np.where(route_for_coord == 0)[0]) if (r !=
+
+     new_locs = datasets['loc'].cpu().numpy()
+     new_coords = [[]]*len(route_list)
+
+     for veh_number, r in enumerate(route_list):
+         new_coords[veh_number] = new_locs[r - 1, :]
```

```
+
+     return new_coords, route_list
+
+
+ def plot_vehicle_routes(data, route, ax1, markersize=5, visualize_demands=False, demand_scale=1, round_demand=False):
+     """
+     Plot the vehicle routes on matplotlib axis ax1.
+     """
+
+     # route is one sequence, separating different routes with 0 (depot)
+     routes = [r[r!=0] for r in np.split(route.cpu().numpy(), np.where(route==0)[0]) if (r != 0).any()]
+     depot = data['depot'].cpu().numpy()
+     locs = data['loc'].cpu().numpy()
+     demands = data['demand'].cpu().numpy() * demand_scale
+     capacity = demand_scale # Capacity is always 1
+
+     x_dep, y_dep = depot
+     ax1.plot(x_dep, y_dep, 'sk', markersize=markersize*4)
+     ax1.set_xlim(0, 1)
+     ax1.set_ylim(0, 1)
+
+     legend = ax1.legend(loc='upper center')
+
+     cmap = discrete_cmap(len(routes) + 2, 'nipy_spectral')
+     dem_rects = []
+     used_rects = []
+     cap_rects = []
+     qvs = []
+     total_dist = 0
+     for veh_number, r in enumerate(routes):
+         color = cmap(len(routes) - veh_number) # Invert to have in rainbow order
+
+         route_demands = demands[r - 1]
+         coords = locs[r - 1, :]
+         xs, ys = coords.transpose()
+
+         total_route_demand = sum(route_demands)
+         # assert total_route_demand <= capacity
+         if not visualize_demands:
+             ax1.plot(xs, ys, 'o', mfc=color, markersize=markersize, markeredgewidth=0.0)
+
+         dist = 0
+         x_prev, y_prev = x_dep, y_dep
+         cum_demand = 0
+         for (x, y), d in zip(coords, route_demands):
+             dist += np.sqrt((x - x_prev) ** 2 + (y - y_prev) ** 2)
+
+             cap_rects.append(Rectangle((x, y), 0.01, 0.1))
+             used_rects.append(Rectangle((x, y), 0.01, 0.1 * total_route_demand / capacity))
+             dem_rects.append(Rectangle((x, y + 0.1 * cum_demand / capacity), 0.01, 0.1 * d / capacity))
+
+             x_prev, y_prev = x, y
+             cum_demand += d
+
+         dist += np.sqrt((x_dep - x_prev) ** 2 + (y_dep - y_prev) ** 2)
+         total_dist += dist
+         qv = ax1.quiver(
+             xs[:-1],
+             ys[:-1],
+             xs[1:] - xs[:-1],
+             ys[1:] - ys[:-1],
+             scale_units='xy',
+             angles='xy',
+             scale=1,
+             color=color,
+             label='R{}, # {}, c {} / {}, d {:.2f}'.format(
+                 veh_number,
+                 len(r),
+                 int(total_route_demand) if round_demand else total_route_demand,
+                 int(capacity) if round_demand else capacity,
+                 dist
+             )
+         )
+
+         qvs.append(qv)
+
+     ax1.set_title('{} routes, total distance {:.2f}'.format(len(routes), total_dist))
+     ax1.legend(handles=qvs)
+
+     pc_cap = PatchCollection(cap_rects, facecolor='whitesmoke', alpha=1.0, edgecolor='lightgray')
+     pc_used = PatchCollection(used_rects, facecolor='lightgray', alpha=1.0, edgecolor='lightgray')
+     pc_dem = PatchCollection(dem_rects, facecolor='black', alpha=1.0, edgecolor='black')
+
+     if visualize_demands:
+         ax1.add_collection(pc_cap)
+         ax1.add_collection(pc_used)
+         ax1.add_collection(pc_dem)
+
+
+ # In[4]:
+
+
+ model1, _ = load_model('pretrained/cvrp_100/')
+ model2, _ = load_model('pretrained/tsp_100/')
+
+ torch.manual_seed(1234)
+ dataset = CVRP.make_dataset(size=100, num_samples=10)
+
+
+ # In[5]:
+
+
+ # Need a dataloader to batch instances
+ dataloader = DataLoader(dataset, batch_size=1000)
+
+ # Make var works for dicts
+ batch = next(iter(dataloader))
+
+ # Run the model
+ model1.eval()
+ model1.set_decode_type('greedy')
+ model2.eval()
+ model2.set_decode_type('greedy')
+
+
+
+ with torch.no_grad():
+     length1, log_p1, pi1 = model1(batch, return_pi=True)
+
+ tours = pi1
+
+ for i, (data, tour) in enumerate(zip(dataset, tours)):
+
+     fig, ax = plt.subplots(figsize=(10, 10))
+     plot_vehicle_routes(data, tour, ax, visualize_demands=False, demand_scale=50, round_demand=True)
+     fig.savefig(os.path.join('images', 'cvrp_{}.png'.format(i)))
+ 
+ for i, (data, tour) in enumerate(zip(dataset, tours)): #written by Isabelle Akian
+     newcoords, new_routes = get_routes_and_coords(data, tour)
+     newroutes = []
+     for k in newcoords:
+         with torch.no_grad():
+ 
+             length2, log_p2, pi2 = model2(torch.FloatTensor([k]), return_pi=True)
+ 
+         tour2 = pi2.tolist()[0]
+         for j in range(len(tour2)):
+             tour2[j] = coord_to_loc(data, k[j])
+ 
+         newroutes = newroutes + tour2
+         newroutes.append(0)
+     fig, ax = plt.subplots(figsize=(10, 10))
+ 
+     plot_vehicle_routes(data, torch.IntTensor(newroutes), ax, visualize_demands=False, demand_scale=50, round_demand=True
+     fig.savefig(os.path.join('images', 'cvrp_and_tsp_{}.png'.format(i)))
+ 
+ 
+ 
+ 
+ # In[ ]:
+ 
+ 
+ 
+ 
```

5 problems/tsp/problem_tsp.py

```
 
  class TSPDataset(Dataset):
 
-     def __init__(self, filename=None, size=50, num_samples=1000000, offset=0, distribution=None):
+     def __init__(self, filename=None, size=50, num_samples=1000000, offset=0, distribution=None, inputdata=[]):
          super(TSPDataset, self).__init__()
 
          self.data_set = []
              with open(filename, 'rb') as f:
                  data = pickle.load(f)
                  self.data = [torch.FloatTensor(row) for row in (data[offset:offset+num_samples])]
+                 print(self.data)
+         elif len(inputdata) > 0: #written by Isabelle Akian
+             self.data = [torch.FloatTensor(row, 2) for row in (inputdata[offset:offset+num_samples])]
          else:
              # Sample points randomly in [0, 1] square
              self.data = [torch.FloatTensor(size, 2).uniform_(0, 1) for i in range(num_samples)]
```

115 readfiles/README(Kool).md

```
@@ -0,0 +1,115 @@
+ # Attention, Learn to Solve Routing Problems!
+ 
+ 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.
+ 
```

```diff
+ ![TSP100](../images/tsp.gif)
+
+ ## Paper
+ 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:
+
+ ```
+ @inproceedings{
+     kool2018attention,
+     title={Attention, Learn to Solve Routing Problems!},
+     author={Wouter Kool and Herke van Hoof and Max Welling},
+     booktitle={International Conference on Learning Representations},
+     year={2019},
+     url={https://openreview.net/forum?id=ByxBFsRqYm},
+ }
+ ```
+
+ ## Dependencies
+
+ * Python>=3.8
+ * NumPy
+ * SciPy
+ * [PyTorch](http://pytorch.org/)>=1.7
+ * tqdm
+ * [tensorboard_logger](https://github.com/TeamHG-Memex/tensorboard_logger)
+ * Matplotlib (optional, only for plotting)
+
+ ## Quick start
+
+ For training TSP instances with 20 nodes and using rollout as REINFORCE baseline:
+ ```bash
+ python run.py --graph_size 20 --baseline rollout --run_name 'tsp20_rollout'
+ ```
+
+ ## Usage
+
+ ### Generating data
+
+ Training data is generated on the fly. To generate validation and test data (same as used in the paper) for all problems:
+ ```bash
+ python generate_data.py --problem all --name validation --seed 4321
+ python generate_data.py --problem all --name test --seed 1234
+ ```
+
+ ### Training
+
+ For training TSP instances with 20 nodes and using rollout as REINFORCE baseline and using the generated validation set:
+ ```bash
+ python run.py --graph_size 20 --baseline rollout --run_name 'tsp20_rollout' --val_dataset data/tsp/tsp20_validation_seed4321.pkl
+ ```
+
+ #### Multiple GPUs
+ By default, training will happen *on all available GPUs*. To disable CUDA at all, add the flag `--no_cuda`.
+ Set the environment variable `CUDA_VISIBLE_DEVICES` to only use specific GPUs:
+ ```bash
+ CUDA_VISIBLE_DEVICES=2,3 python run.py
+ ```
+ Note that using multiple GPUs has limited efficiency for small problem sizes (up to 50 nodes).
+
```

```diff
+ #### Warm start
+ You can initialize a run using a pretrained model by using the `--load_path` option:
+ ```bash
+ python run.py --graph_size 100 --load_path pretrained/tsp_100/epoch-99.pt
+ ```
+
+ The `--load_path` option can also be used to load an earlier run, in which case also the optimizer state will be loaded:
+ ```bash
+ python run.py --graph_size 20 --load_path 'outputs/tsp_20/tsp20_rollout_{datetime}/epoch-0.pt'
+ ```
+
+ 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.
+
+ ### Evaluation
+ 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:
+ ```bash
+ python eval.py data/tsp/tsp20_test_seed1234.pkl --model pretrained/tsp_20 --decode_strategy greedy
+ ```
+ If the epoch is not specified, by default the last one in the folder will be used.
+
+ #### Sampling
+ To report the best of 1280 sampled solutions, use
+ ```bash
+ python eval.py data/tsp/tsp20_test_seed1234.pkl --model pretrained/tsp_20 --decode_strategy sample --width 1280 --eval_batch_size 1
+ ```
+ Beam Search (not in the paper) is also recently added and can be used using `--decode_strategy bs --width {beam_size}`.
+
+ #### To run baselines
+ Baselines for different problems are within the corresponding folders and can be ran (on multiple datasets at once) as follows
+ ```bash
+ 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
+ ```
+ 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).
+
+ ### Other options and help
+ ```bash
+ python run.py -h
+ python eval.py -h
+ ```
+
+ ### Example CVRP solution
+ See `plot_vrp.ipynb` for an example of loading a pretrained model and plotting the result for Capacitated VRP with 100 nodes.
+
+ ![CVRP100](../images/cvrp_0.png)
+
+ ## Acknowledgements
+ 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.
+
+ This repository includes adaptions of the following repositories as baselines:
```

```
+ * https://github.com/MichelDeudon/encode-attend-navigate
+ * https://github.com/mc-ride/orienteering
+ * https://github.com/jordanamecler/PCTSP
+ * 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 @@
+ #!/usr/bin/env python
+ # coding: utf-8
+
+ # In[1]:
+
+
+ import os
+ import numpy as np
+ import torch
+
+
+ # In[2]:
+
+
+ from utils import load_model
+ model, _ = load_model('pretrained/tsp_100/')
+ model.eval()  # Put in evaluation mode to not track gradients
+
+ xy = np.random.rand(100, 2)
+ # print(xy)
+
+ def make_oracle(model, xy, temperature=1.0):
+
+     num_nodes = len(xy)
+
+     xyt = torch.tensor(xy).float()[None]  # Add batch dimension
+
+     with torch.no_grad():  # Inference only
+         embeddings, _ = model.embedder(model._init_embed(xyt))
+
+         # Compute keys, values for the glimpse and keys for the logits once as they can be reused in every step
+         fixed = model._precompute(embeddings)
+
+     def oracle(tour):
+         with torch.no_grad():  # Inference only
+             # Input tour with 0 based indices
+             # Output vector with probabilities for locations not in tour
+             tour = torch.tensor(tour).long()
+             if len(tour) == 0:
+                 step_context = model.W_placeholder
+             else:
+                 step_context = torch.cat((embeddings[0, tour[0]], embeddings[0, tour[-1]]), -1)
+
+             # Compute query = context node embedding, add batch and step dimensions (both 1)
+             query = fixed.context_node_projected + model.project_step_context(step_context[None, None, :])
+
+             # Create the mask and convert to bool depending on PyTorch version
+             mask = torch.zeros(num_nodes, dtype=torch.uint8) > 0
+             mask[tour] = 1
+             mask = mask[None, None, :]  # Add batch and step dimension
+
+             log_p, _ = model._one_to_many_logits(query, fixed.glimpse_key, fixed.glimpse_val, fixed.logit_key, mask)
+             p = torch.softmax(log_p / temperature, -1)[0, 0]
+             assert (p[tour] == 0).all()
+             assert (p.sum() - 1).abs() < 1e-5
+             #assert np.allclose(p.sum().item(), 1)
+         return p.numpy()
+
+     return oracle
+
+
+
+ oracle = make_oracle(model, xy)
+
+ sample = False
+ tour = []
+ tour_p = []
+ while(len(tour) < len(xy)):
+     p = oracle(tour)
+
+     if sample:
+         # Advertising the Gumbel-Max trick
+         g = -np.log(-np.log(np.random.rand(*p.shape)))
+         i = np.argmax(np.log(p) + g)
+         # i = np.random.multinomial(1, p)
+     else:
+         # Greedy
+         i = np.argmax(p)
+     tour.append(i)
+     tour_p.append(p)
+
+ print(tour)
+
+
+ # In[3]:
+
+
+ # get_ipython().run_line_magic('matplotlib', 'inline')
+ from matplotlib import pyplot as plt
+
+ from matplotlib.collections import PatchCollection
+ from matplotlib.patches import Rectangle
+ from matplotlib.lines import Line2D
+
+ # Code inspired by Google OR Tools plot:
+ # https://github.com/google/or-tools/blob/fb12c5ded7423d524fc6c95656a9bdc290a81d4d/examples/python/cvrptw_plot.py
+
+ def plot_tsp(xy, tour, ax1):
+     """
+     Plot the TSP tour on matplotlib axis ax1.
+     """
+ 
+     ax1.set_xlim(0, 1)
+     ax1.set_ylim(0, 1)
+ 
+     xs, ys = xy[tour].transpose()
+     xs, ys = xy[tour].transpose()
+     dx = np.roll(xs, -1) - xs
+     dy = np.roll(ys, -1) - ys
+     d = np.sqrt(dx * dx + dy * dy)
+     lengths = d.cumsum()
+ 
+     # Scatter nodes
+     ax1.scatter(xs, ys, s=40, color='blue')
+     # Starting node
+     ax1.scatter([xs[0]], [ys[0]], s=100, color='red')
+ 
+     # Arcs
+     qv = ax1.quiver(
+         xs, ys, dx, dy,
+         scale_units='xy',
+         angles='xy',
+         scale=1,
+     )
+ 
+     ax1.set_title('{} nodes, total length {:.2f}'.format(len(tour), lengths[-1]))
+ 
+ fig, ax = plt.subplots(figsize=(10, 10))
+ plot_tsp(xy, tour, ax)
+ 
+ 
+ # In[4]:
+ 
+ 
+ from matplotlib.collections import PatchCollection
+ from matplotlib.patches import Rectangle
+ from matplotlib.animation import PillowWriter
+ from matplotlib.lines import Line2D
+ from IPython.display import HTML
+ 
+ from celluloid import Camera  # pip install celluloid
+ 
+ def format_prob(prob):
+     return ('{:.6f}' if prob > 1e-5 else '{:.2E}').format(prob)
+ 
+ def plot_tsp_ani(xy, tour, tour_p=None, max_steps=1000):
+     n = len(tour)
+     fig, ax1 = plt.subplots(figsize=(10, 10))
+     xs, ys = xy[tour].transpose()
+     dx = np.roll(xs, -1) - xs
+     dy = np.roll(ys, -1) - ys
+     d = np.sqrt(dx * dx + dy * dy)
+     lengths = d.cumsum()
+ 
+     ax1.set_xlim(0, 1)
+     ax1.set_ylim(0, 1)
+ 
+     camera = Camera(fig)
+ 
+     total_length = 0
+     cum_log_prob = 0
+     for i in range(n + 1):
```

```
+         for plot_probs in [False] if tour_p is None or i >= n else [False, True]:
+             # Title
+             title = 'Nodes: {:3d}, length: {:.4f}, prob: {}'.format(
+                 i, lengths[i - 2] if i > 1 else 0., format_prob(np.exp(cum_log_prob))
+             )
+             ax1.text(0.6, 0.97, title, transform=ax.transAxes)
+
+             # First print current node and next candidates
+             ax1.scatter(xs, ys, s=40, color='blue')
+
+             if i > 0:
+                 ax1.scatter([xs[i - 1]], [ys[i - 1]], s=100, color='red')
+             if i > 1:
+                 qv = ax1.quiver(
+                     xs[:i-1],
+                     ys[:i-1],
+                     dx[:i-1],
+                     dy[:i-1],
+                     scale_units='xy',
+                     angles='xy',
+                     scale=1,
+                 )
+             if plot_probs:
+                 prob_rects = [Rectangle((x, y), 0.01, 0.1 * p) for (x, y), p in zip(xy, tour_p[i]) if p > 0.01]
+                 pc = PatchCollection(prob_rects, facecolor='lightgray', alpha=1.0, edgecolor='lightgray')
+                 ax1.add_collection(pc)
+             camera.snap()
+         if i < n and tour_p is not None:
+             # Add cumulative_probability
+             cum_log_prob += np.log(tour_p[i][tour[i]])
+         if i > max_steps:
+             break
+
+     # Plot final tour
+     # Scatter nodes
+     ax1.scatter(xs, ys, s=40, color='blue')
+     # Starting node
+     ax1.scatter([xs[0]], [ys[0]], s=100, color='red')
+
+     # Arcs
+     qv = ax1.quiver(
+         xs, ys, dx, dy,
+         scale_units='xy',
+         angles='xy',
+         scale=1,
+     )
+     if tour_p is not None:
+         # Note this does not use stable logsumexp trick
+         cum_log_prob = format_prob(np.exp(sum([np.log(p[node]) for node, p in zip(tour, tour_p)])))
+     else:
+         cum_log_prob = '?'
+     ax1.set_title('{} nodes, total length {:.4f}, prob: {}'.format(len(tour), lengths[-1], cum_log_prob))
+
+     camera.snap()
+
+     return camera
+
+
+ animation = plot_tsp_ani(xy, tour, tour_p).animate(interval=500)
+ writer = PillowWriter(fps=2)
+ # ani.save("demo_sine.gif", writer=writer)
+ animation.save('images/tsp_adjusted.gif', writer='writer', fps=2)  # requires imagemagick
+ # compress by running 'convert tsp.gif -strip -coalesce -layers Optimize tsp.gif'
+ # HTML(animation.to_html5_video())  # requires ffmpeg
+
+
+ # In[ ]:
+
+
+
+
```

230 🟩🟩🟩🟩🟩 tsp_applied.py

```
@@ -0,0 +1,230 @@
+ #!/usr/bin/env python
+ # coding: utf-8
+
+ # In[1]:
+
+
+ import os
+ import numpy as np
+ import torch
+
+ # In[2]:
+
+
+ from utils import load_model
+
+ model, _ = load_model('pretrained/tsp_100/')
+ model.eval()  # Put in evaluation mode to not track gradients
+
+ xy = np.random.rand(100, 2)
+
+
+ def make_oracle(model, xy, temperature=1.0):
+     num_nodes = len(xy)
+
+     xyt = torch.tensor(xy).float()[None]  # Add batch dimension
+
+     with torch.no_grad():  # Inference only
+         embeddings, _ = model.embedder(model._init_embed(xyt))
+
+         # Compute keys, values for the glimpse and keys for the logits once as they can be reused in every step
+         fixed = model._precompute(embeddings)
+
+     def oracle(tour):
+         with torch.no_grad():  # Inference only
+             # Input tour with 0 based indices
+             # Output vector with probabilities for locations not in tour
+             tour = torch.tensor(tour).long()
+             if len(tour) == 0:
+                 step_context = model.W_placeholder
+             else:
+                 step_context = torch.cat((embeddings[0, tour[0]], embeddings[0, tour[-1]]), -1)
+
+             # Compute query = context node embedding, add batch and step dimensions (both 1)
+             query = fixed.context_node_projected + model.project_step_context(step_context[None, None, :])
+
+             # Create the mask and convert to bool depending on PyTorch version
+             mask = torch.zeros(num_nodes, dtype=torch.uint8) > 0
+             mask[tour] = 1
+             mask = mask[None, None, :]  # Add batch and step dimension
+
+             log_p, _ = model._one_to_many_logits(query, fixed.glimpse_key, fixed.glimpse_val, fixed.logit_key, mask)
+             p = torch.softmax(log_p / temperature, -1)[0, 0]
+             assert (p[tour] == 0).all()
+             assert (p.sum() - 1).abs() < 1e-5
+             # assert np.allclose(p.sum().item(), 1)
+         return p.numpy()
+
+     return oracle
+
+
+ oracle = make_oracle(model, xy)
+
+ sample = False
+ tour = []
+ tour_p = []
+ while (len(tour) < len(xy)):
+     p = oracle(tour)
+
+     if sample:
+         # Advertising the Gumbel-Max trick
+         g = -np.log(-np.log(np.random.rand(*p.shape)))
+         i = np.argmax(np.log(p) + g)
+         # i = np.random.multinomial(1, p)
+     else:
+         # Greedy
+         i = np.argmax(p)
+     tour.append(i)
+     tour_p.append(p)
+ print(xy)
+ print(tour)
+
+ # In[3]:
+
+
+ # get_ipython().run_line_magic('matplotlib', 'inline')
+ from matplotlib import pyplot as plt
+
+ from matplotlib.collections import PatchCollection
+ from matplotlib.patches import Rectangle
+ from matplotlib.lines import Line2D
+
+
+ # Code inspired by Google OR Tools plot:
+ # https://github.com/google/or-tools/blob/fb12c5ded7423d524fc6c95656a9bdc290a81d4d/examples/python/cvrptw_plot.py
+
+ def plot_tsp(xy, tour, ax1):
+     """
+     Plot the TSP tour on matplotlib axis ax1.
+     """
+
+     ax1.set_xlim(0, 1)
+     ax1.set_ylim(0, 1)
+
+     xs, ys = xy[tour].transpose()
+     xs, ys = xy[tour].transpose()
+     dx = np.roll(xs, -1) - xs
+     dy = np.roll(ys, -1) - ys
+     d = np.sqrt(dx * dx + dy * dy)
+     lengths = d.cumsum()
+
+     # Scatter nodes
+     ax1.scatter(xs, ys, s=40, color='blue')
+     # Starting node
+     ax1.scatter([xs[0]], [ys[0]], s=100, color='red')
+
+     # Arcs
+     qv = ax1.quiver(
+         xs, ys, dx, dy,
+         scale_units='xy',
+         angles='xy',
+         scale=1,
+     )
+
+     ax1.set_title('{} nodes, total length {:.2f}'.format(len(tour), lengths[-1]))
+
+
+ fig, ax = plt.subplots(figsize=(10, 10))
+ plot_tsp(xy, tour, ax)
+
+ # In[4]:
+
+
+ from matplotlib.collections import PatchCollection
+ from matplotlib.patches import Rectangle
+ from matplotlib.lines import Line2D
+ from IPython.display import HTML
+
+ from celluloid import Camera  # pip install celluloid
+
+
+ def format_prob(prob):
+     return ('{:.6f}' if prob > 1e-5 else '{:.2E}').format(prob)
+
+
+ def plot_tsp_ani(xy, tour, tour_p=None, max_steps=1000):
+     n = len(tour)
+     fig, ax1 = plt.subplots(figsize=(10, 10))
+     xs, ys = xy[tour].transpose()
+     dx = np.roll(xs, -1) - xs
+     dy = np.roll(ys, -1) - ys
+     d = np.sqrt(dx * dx + dy * dy)
+     lengths = d.cumsum()
+
+     ax1.set_xlim(0, 1)
+     ax1.set_ylim(0, 1)
+
+     camera = Camera(fig)
+
+     total_length = 0
+     cum_log_prob = 0
+     for i in range(n + 1):
+         for plot_probs in [False] if tour_p is None or i >= n else [False, True]:
+             # Title
+             title = 'Nodes: {:3d}, length: {:.4f}, prob: {}'.format(
+                 i, lengths[i - 2] if i > 1 else 0., format_prob(np.exp(cum_log_prob))
+             )
+             ax1.text(0.6, 0.97, title, transform=ax.transAxes)
+
+             # First print current node and next candidates
+             ax1.scatter(xs, ys, s=40, color='blue')
+
+             if i > 0:
```

```
+                 ax1.scatter([xs[i - 1]], [ys[i - 1]], s=100, color='red')
+             if i > 1:
+                 qv = ax1.quiver(
+                     xs[:i - 1],
+                     ys[:i - 1],
+                     dx[:i - 1],
+                     dy[:i - 1],
+                     scale_units='xy',
+                     angles='xy',
+                     scale=1,
+                 )
+             if plot_probs:
+                 prob_rects = [Rectangle((x, y), 0.01, 0.1 * p) for (x, y), p in zip(xy, tour_p[i]) if p > 0.01]
+                 pc = PatchCollection(prob_rects, facecolor='lightgray', alpha=1.0, edgecolor='lightgray')
+                 ax1.add_collection(pc)
+             camera.snap()
+         if i < n and tour_p is not None:
+             # Add cumulative_probability
+             cum_log_prob += np.log(tour_p[i][tour[i]])
+         if i > max_steps:
+             break
+ 
+     # Plot final tour
+     # Scatter nodes
+     ax1.scatter(xs, ys, s=40, color='blue')
+     # Starting node
+     ax1.scatter([xs[0]], [ys[0]], s=100, color='red')
+ 
+     # Arcs
+     qv = ax1.quiver(
+         xs, ys, dx, dy,
+         scale_units='xy',
+         angles='xy',
+         scale=1,
+     )
+     if tour_p is not None:
+         # Note this does not use stable logsumexp trick
+         cum_log_prob = format_prob(np.exp(sum([np.log(p[node]) for node, p in zip(tour, tour_p)])))
+     else:
+         cum_log_prob = '?'
+     ax1.set_title('{} nodes, total length {:.4f}, prob: {}'.format(len(tour), lengths[-1], cum_log_prob))
+ 
+     camera.snap()
+ 
+     return camera
+ 
+ 
+ animation = plot_tsp_ani(xy, tour, tour_p).animate(interval=500)
+ # animation.save('images/tsp.gif', writer='imagemagick', fps=2)  # requires imagemagick
+ # compress by running 'convert tsp.gif -strip -coalesce -layers Optimize tsp.gif'
+ # HTML(animation.to_html5_video())  # requires ffmpeg
+ 
+ 
+ # In[ ]:
+ 
+ 
+ 
+ 
```