Solving travelling salesman problem with reinforcement learning: classic encoder-decoder architecture

S SofiKardami · Follow
4 min read · Aug 7

Hey everyone! Today we will solve the tsp problem using encoder-decoder architecture. The tutorial I adapted was from the original pytorch encoder-decoder tutorial which can be found here.

The encoder is a RNN that outputs some value for every item from the input sequence. For every input item, the encoder outputs a vector and a hidden state, and uses the hidden state for the next input item.

The decoder is another RNN that takes the encoder output vector(s) and outputs a sequence of indexes that show the order of cities the travelsman will visit.

In this implementation, the decoder uses only last output of the encoder. This last output is called the *context vector* as it encodes context from the entire sequence. This context vector is used as the initial hidden state of the decoder.

At every step of decoding, the decoder is given an input and hidden state. The initial decoder input is a dummy tensor with zeros and has size [batch_size, 1], and the first hidden state is the context vector (the encoder's last hidden state). After the first decoding step, in the next decoding steps the decoder takes as input the previous prediction the decoder made and as hidden state the previous hidden state the decoder gave.

The reward of the model is the negative of the total distance the agent travelled for this particular solution. For example if the agent travelled 200km for the solution the model provided, the reward of the agent it -200km. The goal is to maximize the reward, aka minimize the distance travelled for a solution.

```
import numpy as np
from scipy.spatial import distance_matrix
import torch.nn.functional as F
import torch
from torch import nn
from tqdm import tqdm
import torch.optim as optim
from torch.autograd import Variable
from torch.utils.data import DataLoader
from or_tools_comparisons.common_utilities import get_tour_length_from_distance_ma
from ploting.metrics.plot_average_tour_length import plot_average_tour_length
from ploting.plot_utilities import plot_train_and_validation_loss
from torch.utils.data import Dataset

static_features = 2
hidden_size = 128

class TSPDataset(Dataset):
    """  Random TSP dataset """
    def __init__(self, data_size, seq_len):
        self.data_size = data_size
        self.seq_len = seq_len
        self.data = self._generate_data()
    def __len__(self):
        return self.data_size
    def __getitem__(self, idx):
        tensor = torch.from_numpy(self.data['Points_List'][idx]).float()
        sample = {'Points':tensor }
        return sample

    def _generate_data(self):
        """
        :return: Set of points_list ans their One-Hot vector solutions
        """
        points_list = []
        solutions = []
        data_iter = tqdm(range(self.data_size), unit='data')
        for i, _ in enumerate(data_iter):
            data_iter.set_description('Data points %i/%i' % (i+1, self.data_size))
            points_list.append(np.random.randint(30,size=(self.seq_len, 2))) # np.
```

```
            return {'Points_List':points_list }

        def _to1hotvec(self, points):
            """
            :param points: List of integers representing the points indexes
            :return: Matrix of One-Hot vectors
            """
            vec = np.zeros((len(points), self.seq_len))
            for i, v in enumerate(vec):
                v[points[i]] = 1
            return vec

    class Encoder(nn.Module):
        def __init__(self):
            super(Encoder, self).__init__()
            dropout_p = 0.1
            self.hidden_size = hidden_size
            self.embedding = nn.Linear(static_features, hidden_size)
            self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
            self.dropout = nn.Dropout(dropout_p)

        def forward(self, input):
            embedded = self.dropout(self.embedding(input))
            output, hidden = self.gru(embedded)
            return output, hidden


    class Decoder(nn.Module):
        def __init__(self, sequence_length):
            super(Decoder, self).__init__()

            self.embedding = nn.Linear(1, hidden_size)
            self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
            self.out = nn.Linear(hidden_size, sequence_length)

        def apply_mask_to_logits(self, logits, mask, indexes):
            batch_size = logits.size(0)
            clone_mask = mask.clone()
            if indexes is not None:
                clone_mask[[i for i in range(batch_size)], indexes.data.squeeze(1).lor

                logits[clone_mask.unsqueeze(1)] = -np.inf
            else:
                logits[:, :] = -np.inf
                # we want to start from depot, ie the first node
                logits[:, :, 0] = 1
            return logits, clone_mask

        def forward(self, encoder_outputs, encoder_hidden):
            batch_size = encoder_outputs.size(0)
```

```
        seq_len = encoder_outputs.size(1)

        decoder_input = torch.ones(batch_size, 1)
        decoder_hidden = encoder_hidden
        decoder_outputs = []

        tours = []
        tour_logp = []
        mask = torch.zeros(batch_size, seq_len).byte()

        chosen_indexes = None
        for i in range(seq_len):
            decoder_output, decoder_hidden = self.forward_step(decoder_input, decc
            decoder_outputs.append(decoder_output)

            # We use its own predictions as the next input
            _, topi = decoder_output.topk(1)
            decoder_input = topi.squeeze(-1).detach().float()
            masked_logits, mask = self.apply_mask_to_logits(decoder_output, mask,

            # We transform decoder output to the actual result
            chosen_indexes = torch.argmax(masked_logits, dim=2).float()  # [batch_
            log_probs = F.log_softmax(decoder_output, dim=2)
            logp = torch.gather(log_probs, 2, chosen_indexes.unsqueeze(2).long()).

            tour_logp.append(logp.unsqueeze(1))
            tours.append(chosen_indexes.unsqueeze(1))

        tours = torch.cat(tours, 2)
        tour_logp = torch.cat(tour_logp, dim=1)  # (batch_size, seq_len)

        return tours, tour_logp

    def forward_step(self, input, hidden):
        output = self.embedding(input)
        output = F.relu(output)
        output, hidden = self.gru(output.unsqueeze(1), hidden)
        output = self.out(output)
        return output, hidden


def reward_fn(static, tour_indices):
    """
    static: [batch_size, 2, sequence_length]
    tour_indices: [batch_size, tour_length]
    Euclidean distance between all cities / nodes given by tour_indices
    """
    # Convert the indices back into a tour
    idx = tour_indices.unsqueeze(1).expand(-1, static.size(1), -1)
    tour = torch.gather(static.data, 2, idx).permute(0, 2, 1)
```

```
    tour_len = torch.sqrt(torch.sum(torch.pow(tour[:, :-1] - tour[:, 1:], 2), dim=

    return tour_len.sum(1)


class ClassicSeq2SeqTSPModel(nn.Module):
    def __init__(self, sequence_length):
        super().__init__()
        self.encoder = Encoder()
        self.decoder = Decoder(sequence_length)

    def forward(self, inputs):
        encoder_outputs, encoder_hidden = self.encoder(inputs)

        tours, tour_logp = self.decoder(encoder_outputs, encoder_hidden)

        return tours, tour_logp


def trainClassicSeq2SeqTSPWithReinforcementLearning(train_dataset,
                                                    test_dataset,
                                                    epochs,
                                                    experiment_details,
                                                    batch_size=10,
                                                    num_nodes=13,
                                                    lr=1e-4):

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
    validation_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=Tr

    model = ClassicSeq2SeqTSPModel(sequence_length=num_nodes)

    optimizer = optim.Adam(model.parameters(), lr=lr)

    val_loss = []
    train_loss = []
    val_loss_per_epoch = []
    losses_per_epoch = []

    tour_lengths_per_epoch = []

    for epoch in range(epochs):
        model.train()

        loss_at_epoch = 0.0
        val_loss_at_epoch = 0.0
        iterator = tqdm(train_loader, unit='Batch')

        for batch_id, sample_batch in enumerate(iterator):
            optimizer.zero_grad()
```

```
            train_batch = Variable(sample_batch['Points'])
            output_routes, tour_logp = model(train_batch)
            reward = reward_fn(train_batch.transpose(1, 2), output_routes.squeeze(

            # reward here is the distance travelled for the solution the model pro
            # we want to minimize this distance and therefore we are using the log
            # output) that contain a minus sign and we multiply it by the reward.
            loss = torch.mean(reward.detach() * tour_logp.sum(dim=1))

            loss.backward()
            optimizer.step()
            loss_at_epoch += loss.detach().sum().item()
            average_tour_length = 0

            for tour in range(batch_size):
                points = sample_batch['Points'][tour]
                distance_matrix_array = distance_matrix(points, points)
                content_from_my_model, tour_length = get_tour_length_from_distance
                average_tour_length += tour_length.item()

        model.eval()
        for val_batch in validation_loader:
            train_batch = Variable(val_batch['Points'])
            tours, tour_logp = model(train_batch)

            reward = reward_fn(train_batch.transpose(1, 2), tours.squeeze(1).to(to

            loss = torch.mean(reward.detach() * tour_logp.sum(dim=1))

            val_loss.append(loss.data.item())
            val_loss_at_epoch += loss.detach().item()

        tour_lengths_per_epoch.append(average_tour_length / batch_size)
        train_loss.append(loss_at_epoch / batch_size)

        losses_per_epoch.append(loss_at_epoch)
        val_loss_per_epoch.append(val_loss_at_epoch)

    # Training finished
    plot_train_and_validation_loss(epoch, losses_per_epoch, val_loss_per_epoch, ex
    plot_average_tour_length(tour_lengths_per_epoch, experiment_details)

    return model, tours


if __name__ == '__main__':
    epochs = 100
    num_nodes = 5
    train_size = 1000
    test_size = 25
```

```
batch_size = 10
lr = 1e-4
train_dataset = TSPDataset(train_size, num_nodes)
test_dataset = TSPDataset(test_size, num_nodes)

experiment_details = f'seq2seq_epochs{epochs}_train{train_size}_seqLen{num_noc

trainClassicSeq2SeqTSPWithReinforcementLearning(train_dataset,
                                                test_dataset,
                                                epochs,
                                                experiment_details,
                                                batch_size,
                                                num_nodes,
                                                lr)
```

As always questions and comments are always welcomed!

Happy learning!

Travelling Salesman | Seq2seq | Reinforcement Learning | Encoder Decoder | Routing

Follow

Written by SofiKardami

1 Follower

Software engineer && electrical and computer engineering student. Currently learning machine learning and writing my thesis on vehicle routing problems.

Recommended from Medium

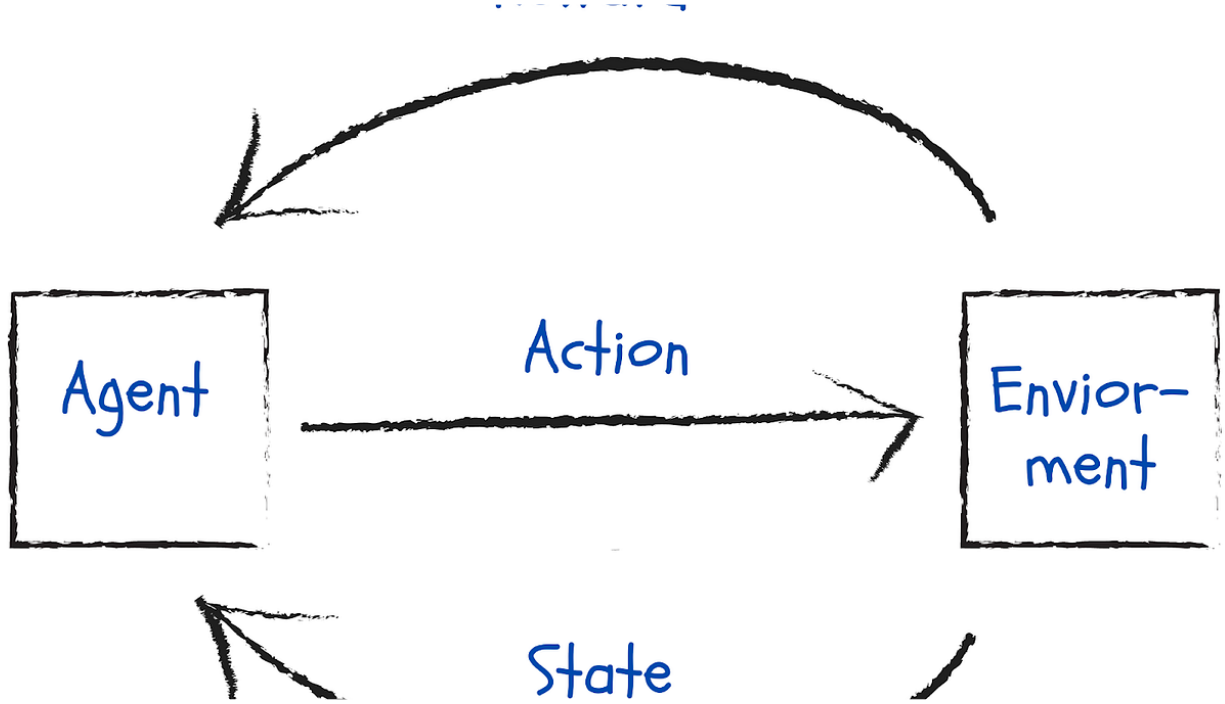
Peyman Kor in Towards AI

Inventory Optimization with Data Science: Hands-On Tutorial with Python

Part 2: A Gentle Introduction to Implementing the Markov Reward Process (MRP) for Inventory Optimization.

11 min read · Oct 13

40 1

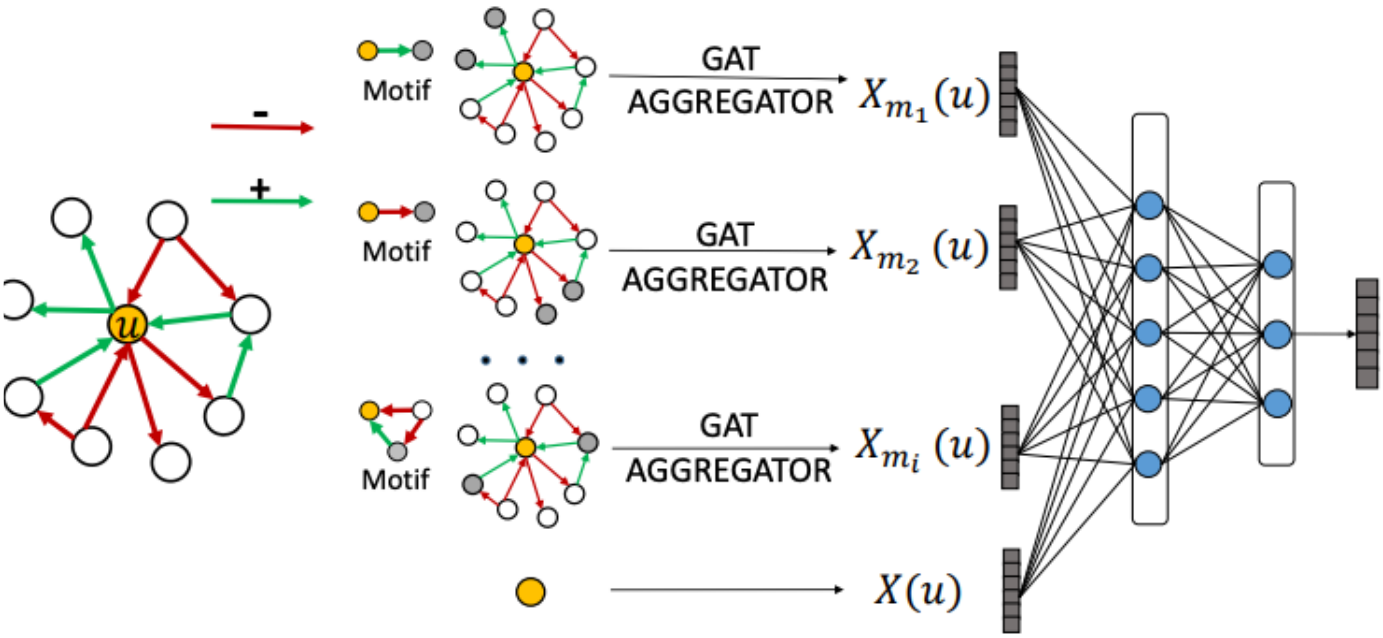


Iris John

Deep Reinforcement Learning: Building Intelligent Agents

Reinforcement learning : An Introduction

6 min read · Aug 11



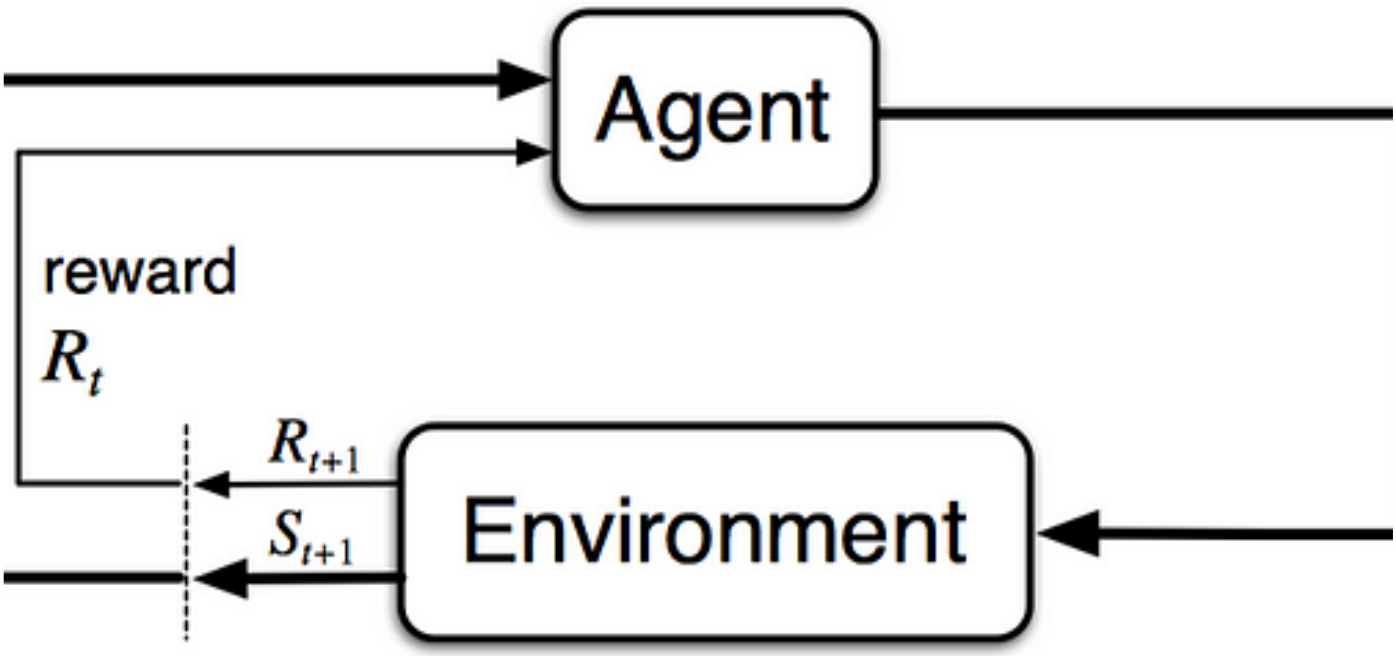
Daniel Ko

Graph Neural Networks: From CNNs to GATs

A high level understanding of the progression to SiGATs.

4 min read · Jun 5

1



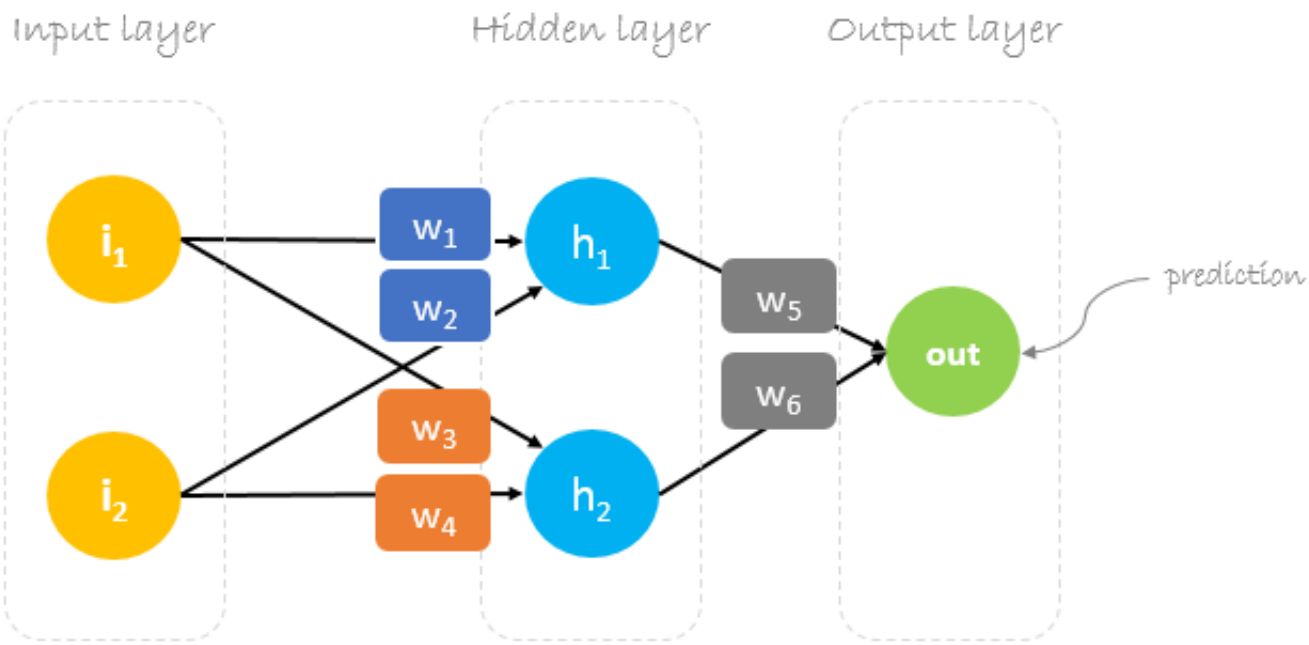
Mateusfreitasrosa

Develop Your First Reinforcement Learning Algorithm for Video Games (from scratch!!!).

Hello! My name is Mateus de Freitas Rosa. I've been a data scientist since February 2021 and graduated in Information Systems in August...

16 min read · Aug 13

2



Francesco Franco

Error Backprogation Walk Through

In my last post I talked about feed-forward neural networks and introduced some of the fundamental concepts, their mathematical derivation...

6 min read · Sep 16

24

Michael Atkin in Stanford CS224W GraphML Tutorials

Tackling the Traveling Salesman Problem with Graph Neural Networks

Do you ever feel like there aren't enough hours in the day to get everything done? You might have a laundry list of errands to run—from...

12 min read · May 15

87

See more recommendations