CSCE 689
Machine Learning Based Systems
Homework 1
Due: Sep 24, 2018

Name: Sanuj Sharma
UIN: 527004390

Question 1

(a) Pytorch
(b) It supports Python.
(c) No issues were encountered. But PyTorch doesn’t support my laptop’s AMD GPU, and only supports NVIDIA GPU.
(d) I. Code

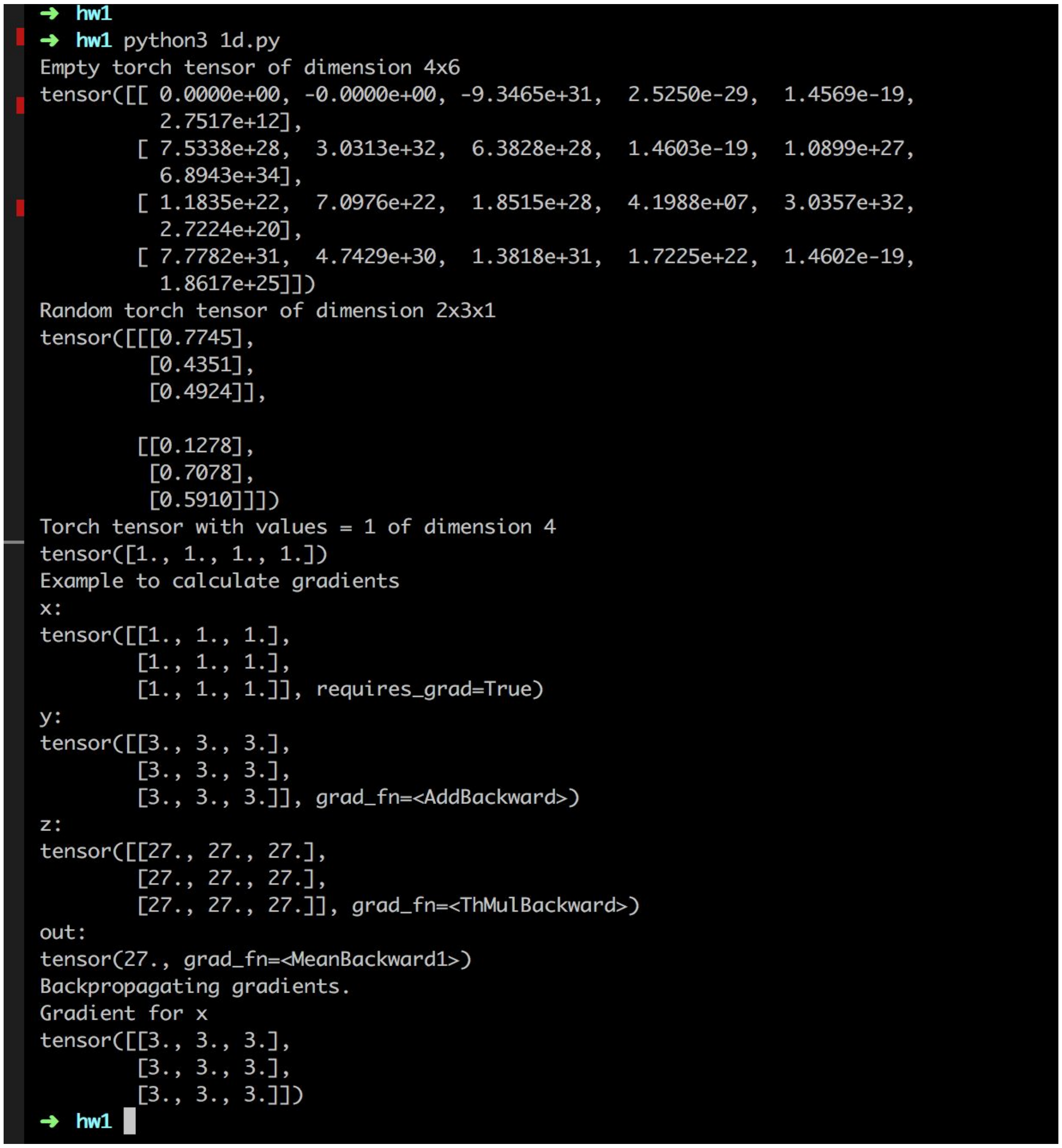
```
import torch

print("Empty torch tensor of dimension 4x6")
a = torch.empty(4, 6)
print(a)

print("Random torch tensor of dimension 2x3x1")
b = torch.rand(2, 3, 1)
print(b)

print("Torch tensor with values = 1 of dimension 4")
c = torch.ones(4)
print(c)


print("Example to calculate gradients")
print("x:")
x = torch.ones(3, 3, requires_grad=True)
print(x)

print("y:")
y = x + 2
print(y)

print("z:")
z = y*y*y
print(z)

print("out:")
out = z.mean()
print(out)

print("Backpropagating gradients.")
```

```
out.backward()

print("Gradient for x")
print(x.grad)
```

Ii. Compilation command.

```
➜  $ python3 1d.py
```

Iii. Output



```
➜ hw1
➜ hw1 python3 1d.py
Empty torch tensor of dimension 4x6
tensor([[ 0.0000e+00, -0.0000e+00, -9.3465e+31,  2.5250e-29,  1.4569e-19,
          2.7517e+12],
        [ 7.5338e+28,  3.0313e+32,  6.3828e+28,  1.4603e-19,  1.0899e+27,
          6.8943e+34],
        [ 1.1835e+22,  7.0976e+22,  1.8515e+28,  4.1988e+07,  3.0357e+32,
          2.7224e+20],
        [ 7.7782e+31,  4.7429e+30,  1.3818e+31,  1.7225e+22,  1.4602e-19,
          1.8617e+25]])
Random torch tensor of dimension 2x3x1
tensor([[[0.7745],
         [0.4351],
         [0.4924]],

        [[0.1278],
         [0.7078],
         [0.5910]]])
Torch tensor with values = 1 of dimension 4
tensor([1., 1., 1., 1.])
Example to calculate gradients
x:
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]], requires_grad=True)
y:
tensor([[3., 3., 3.],
        [3., 3., 3.],
        [3., 3., 3.]], grad_fn=<AddBackward>)
z:
tensor([[27., 27., 27.],
        [27., 27., 27.],
        [27., 27., 27.]], grad_fn=<ThMulBackward>)
out:
tensor(27., grad_fn=<MeanBackward1>)
Backpropagating gradients.
Gradient for x
tensor([[3., 3., 3.],
        [3., 3., 3.],
        [3., 3., 3.]])
➜ hw1
```

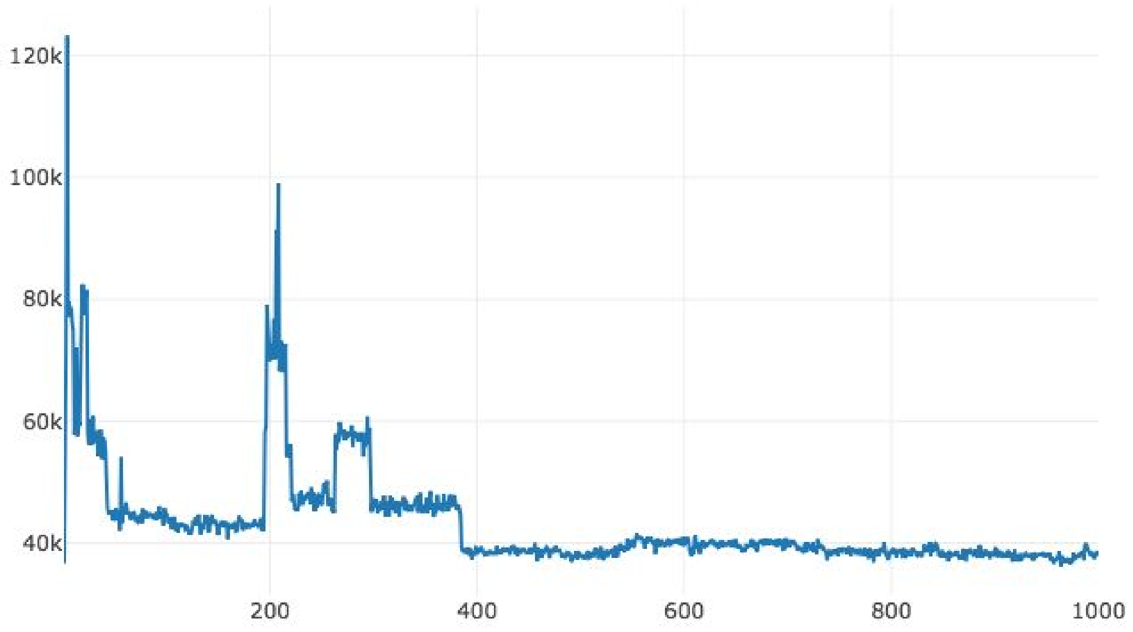
Question 2

(a) I used 3000 sequences as training data.
(b) I used a single LSTM of 64 hidden size. The LSTM was trained as a regression model with mean squared error (MSE) loss.
(c) I used 15 sequences to test the model.
(d) The average MSE loss for test data was 91438.
(e) Code

```python
import torch
import torch.nn as nn
import numpy as np
import csv
import visdom


class LSTMReg(nn.Module):
   def __init__(self, inp_size, hid_size, out_size):
       super(LSTMReg, self).__init__()
       self.rnn = nn.LSTM(inp_size, hid_size)
       self.res = nn.Linear(hid_size*2, out_size)

   def forward(self, inp, hid):
       _, hid = self.rnn(inp, hid)
       out = self.res(torch.cat([hid[0].view(1,-1), hid[1].view(1, -1)], 1))
       return out, hid


def train(out_no, in_nos):
   hidden = torch.randn(2, 1, 1, n_hid)
   lstm.zero_grad()

   for i in range(in_nos.size()[0]):
       out, hidden = lstm(in_nos[i].view(1, 1, -1), hidden)

   loss = criterion(out, out_no.view(1, 1))
   loss.backward()

   # Add parameters' gradients to their values, multiplied by learning rate
   for p in lstm.parameters():
       p.data.add_(-learning_rate, p.grad.data)

   return out, loss.item()


def get_data(name):
   train_data = []
   with open(name) as train_file:
       reader = csv.reader(train_file)
```

```
        for row in reader:
            train_data.append(list(map(int, row)))

    return torch.tensor(train_data, dtype=torch.float)


def main():
    train_data = get_data('2train.csv')
    test_data = get_data('2test.csv')
    vis = visdom.Visdom()
    loss_plot = None

    for epoch in range(1, epochs + 1):
        curr_loss = 0
        for data in train_data:
            output, loss = train(data[3], data[:3])
            curr_loss += loss
        curr_loss /= len(train_data)

        print("Iter: " + str(epoch) + ", loss: " + str(curr_loss))
        if loss_plot is None:
            loss_plot = vis.line(Y=np.array([curr_loss]), X=np.array([epoch]),
win=loss_plot)
        else:
            vis.line(Y=np.array([curr_loss]), X=np.array([epoch]), win=loss_plot,
update='append')

        if epoch % 10 == 0:
            for data in test_data:
                out_no, in_nos = data[3], data[:3]
                hidden = torch.randn(2, 1, 1, n_hid)
                for i in range(in_nos.size()[0]):
                    out, hidden = lstm(in_nos[i].view(1, 1, -1), hidden)
                loss = criterion(out, out_no.view(1, 1))
                print(data, out, loss)


if __name__ == '__main__':
    epochs = 1000
    n_hid = 64
    n_in_no = 1
    n_out_no = 1
    learning_rate = 0.01  # If you set this too high, it might explode. If too low,
it might not learn.
    criterion = nn.MSELoss()
    lstm = LSTMReg(n_in_no, n_hid, n_out_no)
    main()
```

Loss function plot while training:



Question 3

(a) Fitness function:

```
target = 100
def get_fitness(curr):
   return 1 - abs(target-curr)/(abs(target) + abs(curr))
```

(b) 2

(c) Mutation and crossover

```
def reproduce(x1, y1, x2, y2):
   return [[x1, x2], [y1, y2], [x1, y2], [x2, y1]]

def do_crossover(population):
   new_population = []
   i = 0
   while i < len(population)-1:
       new_population += reproduce(*population[i], *population[i+1])
       i += 2
   return new_population

def do_mutation(population):
   for i in range(len(population)):
        should_mutate = random.choice([True, False])
        if should_mutate:
            index = random.choice([0, 1])
            population[i][index] = rand()
    return population
```

(d) x: 1.9518932775944258, y: 2.9271430395036298, Iterations: 339

```
Iteration number: 328
Iteration number: 329
Iteration number: 330
Iteration number: 331
Iteration number: 332
Iteration number: 333
Iteration number: 334
Iteration number: 335
Iteration number: 336
Iteration number: 337
Iteration number: 338
Iteration number: 339
Final answer: [1.9518932775944258, 2.9271430395036298]
```

(e) x: 9.472578390691723, y: 2.647093456642742, iterations: 10000
Value of z would never be negative for range 1 < x, y < 10.

(f) Code:

```
import math
import random

min_val = 1
max_val = 10
target = 100
max_population_size = 200
epsilon = 1e-7
init_pop_size = 2
max_inters = 10000

def get_z(x, y):
   return pow(x, 2) + pow(math.e, y/5) + 100*math.log(x, 2) - 1 / pow(x, y) - x

def get_fitness(curr):
   return 1 - abs(target-curr)/(abs(target) + abs(curr))

def rand():
   return random.uniform(min_val, max_val)

def get_init_population(size):
   pop = []
   for _ in range(size):
       pop.append([rand(), rand()])
   return pop

def reproduce(x1, y1, x2, y2):
   return [[x1, x2], [y1, y2], [x1, y2], [x2, y1]]

def do_crossover(population):
   new_population = []
   i = 0
   while i < len(population)-1:
       new_population += reproduce(*population[i], *population[i+1])
       i += 2
   return new_population

def do_mutation(population):
   for i in range(len(population)):
       should_mutate = random.choice([True, False])
       if should_mutate:
           index = random.choice([0, 1])
           population[i][index] = rand()
   return population

def sort_by_fitness(population):
   return sorted(population, key=lambda i: get_fitness(get_z(i[0], i[1])),
reverse=True)

def do_selection(population):
   population = sort_by_fitness(population)
   return population[:min(len(population), max_population_size)]

def main():
   pop = get_init_population(init_pop_size)
   ans = pop[0]
   fitness = get_fitness(get_z(*ans))
   i = 0
   while (i < max_inters and abs(fitness-1) > epsilon):
       i += 1
       print("Iteration number: " + str(i))
       pop = do_crossover(pop)
       pop = do_mutation(pop)
       pop = do_selection(pop)
       new_fitness = get_fitness(get_z(*pop[0]))
       if new_fitness > fitness:
           ans = pop[0]
           fitness = new_fitness

   print("Final answer: " + str(ans))
```

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

Question 4:

Effectiveness of crossover can be improved by using machine learning models to predict the next generation which has more fitness.

Recently, a quantum crossover procedure was proposed which proposes crossovers for all chromosomes in parallel resulting in a quadratic speed up.[1]

To improve mutation, we can use different kind of mutation methods to get the final mutation like: swap, insertion, inversion or displacement mutation.[2]

Question 5:

Self loop in a CNN has been used to improve image segmentation and video segmentation. CNN is good for extracting features from image, a self loop should improve it to deal with temporal data like videos.

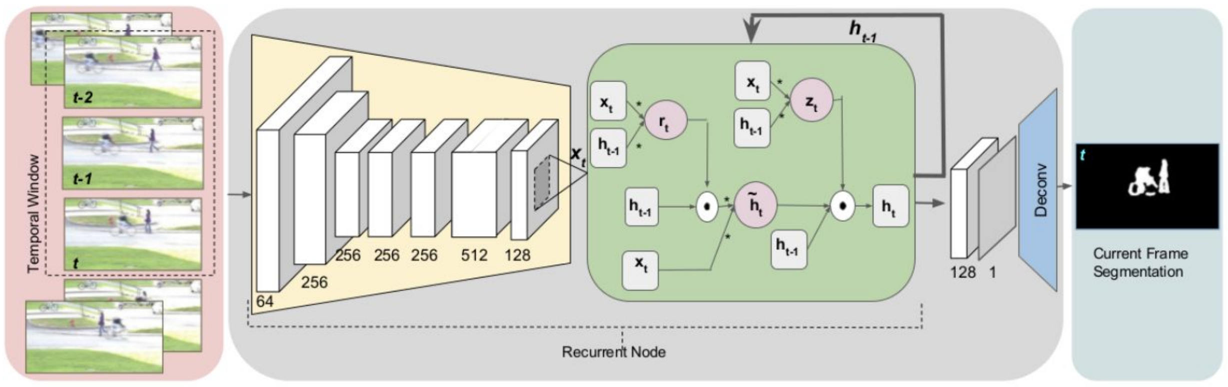


Figure 2: RFC-VGG architecture. A sequence of images is given as input to the network. The output of the embedded FC inside the recurrent unit is given to a Conv-GRU layer. One last convolutional layer maps the output of the recurrent unit into a coarse segmentation map. Then, the deconvolutional layer up-samples the coarse segmentation into a dense segmentation.

The above image is taken from a paper which uses this from video segmentation.[3]

[1] "A quantum genetic algorithm with quantum crossover and mutation ...." 9 Feb. 2012, https://arxiv.org/abs/1202.2026. Accessed 27 Sep. 2018.
[2] "Performance impact of mutation operators of a subpopulation-based ...." https://www.ncbi.nlm.nih.gov/pubmed/27588254. Accessed 27 Sep. 2018.
[3] "Convolutional Gated Recurrent Networks for Video Segmentation." 16 Nov. 2016, https://arxiv.org/abs/1611.05435. Accessed 27 Sep. 2018.

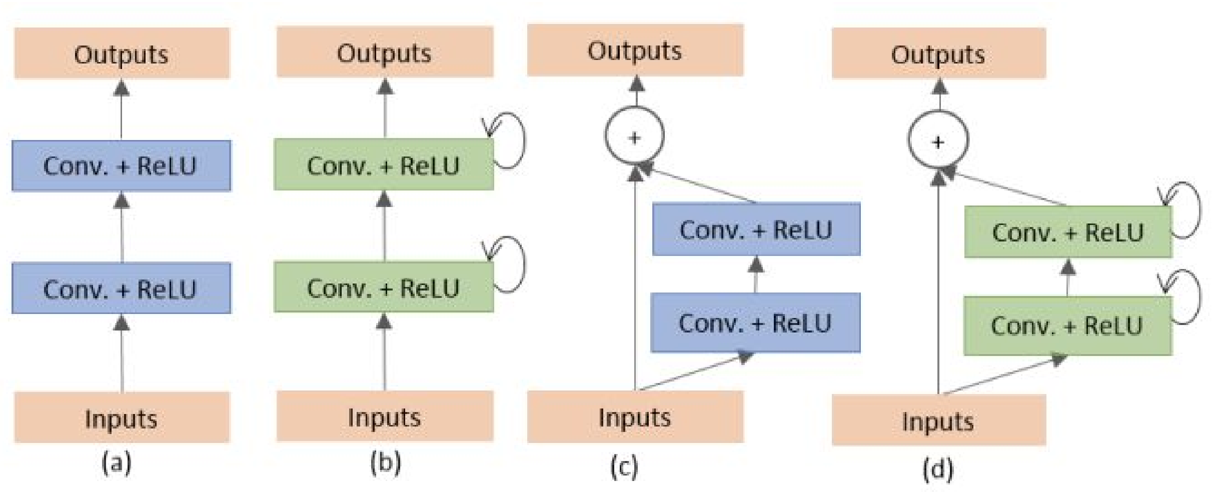


Fig. 4. Different variant of convolutional and recurrent convolutional units (a) Forward convolutional units, (b) Recurrent convolutional block (c) Residual convolutional unit, and (d) Recurrent Residual convolutional units (RRCU).

The paper[4] states the following advantages:

- A residual unit helps when training deep architecture.
- Feature accumulation with recurrent residual convolutional layers ensures better feature representation for segmentation tasks.
- It allows us to design better U-Net architecture with same number of network parameters with better performance for medical image segmentation.

All the code mentioned here is available on github at the below link:
https://github.com/sanuj/689-hw1

[4] "Recurrent Residual Convolutional Neural Network based on U-Net ...." 20 Feb. 2018, https://arxiv.org/abs/1802.06955. Accessed 27 Sep. 2018.