```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('/content/gdrive/MyDrive/project_7/data/tinyshakespeare/input.txt', 'r').read() #
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print('data has %d characters, %d unique.' % (data_size, vocab_size))
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }
```

```
data has 1115394 characters, 65 unique.
```

```
# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1
epoch = 5


# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias


def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in range(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
```

```
      xs[t][inputs[t]] = 1
      hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
      ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
      ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
      loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
    # backward pass: compute gradients going backwards
    dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])
    for t in reversed(range(len(inputs))):
      dy = np.copy(ps[t])
      dy[targets[t]] -= 1 # backprop into y. see http://cs231n.github.io/neural-networks-case-s
      dWhy += np.dot(dy, hs[t].T)
      dby += dy
      dh = np.dot(Why.T, dy) + dhnext # backprop into h
      dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
      dbh += dhraw
      dWxh += np.dot(dhraw, xs[t].T)
      dWhh += np.dot(dhraw, hs[t-1].T)
      dhnext = np.dot(Whh.T, dhraw)
    for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
      np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]



  def sample(h, seed_ix, n):
    """
    sample a sequence of integers from the model
    h is memory state, seed_ix is seed letter for first time step
    """
    x = np.zeros((vocab_size, 1))
    x[seed_ix] = 1
    ixes = []
    for t in range(n):
      h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
      y = np.dot(Why, h) + by
      p = np.exp(y) / np.sum(np.exp(y))
      ix = np.random.choice(range(vocab_size), p=p.ravel())
      x = np.zeros((vocab_size, 1))
      x[ix] = 1
      ixes.append(ix)
    return ixes


  n, p, e = 0, 0, 0
  mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
  smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
  while e < epoch + 1:
    # prepare inputs (we're sweeping from left to right in steps seq_length long)
    if p+seq_length+1 >= len(data) or n == 0:
```

```
  hprev = np.zeros((hidden_size,1)) # reset RNN memory
  p = 0 # go from start of data
  print("end of data.... new epoch")
  e += 1
inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

# sample from the model now and then
if n % 100 == 0:
  sample_ix = sample(hprev, inputs[0], 200)
  txt = ''.join(ix_to_char[ix] for ix in sample_ix)
  print('----\n %s \n----' % (txt, ))
  print("epoch: ", e)

# forward seq_length characters through the net and fetch gradient
loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
smooth_loss = smooth_loss * 0.999 + loss * 0.001
if n % 100 == 0: print('iter %d, loss: %f' % (n, smooth_loss))# print progress

# perform parameter update with Adagrad
for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                              [dWxh, dWhh, dWhy, dbh, dby],
                              [mWxh, mWhh, mWhy, mbh, mby]):
  mem += dparam * dparam
  param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

p += seq_length # move data pointer
n += 1 # iteration counter
```

```
 I drogh to her me hory. Hary hvaw?

 HAMUSB ALTel! bethers, worn insss.

 KINCABIO:
 Saor wofes mesle knasiout
 Hom.

 ISlon: me his her, cend; fats; ce ahly it ry,
 ----
 epoch:  5
 iter 218100, loss: 51.981604
 ----
  nomine.

 KAWALLAR MiUCESt,
 That atirce sim make iin
 And, sal I it veal, barges be font your pemin thin, unid hemy of hamspar;
 Your downed bedy: Wenire:
 To the whury scond yout she.

 AND EENT:
 Sho:
 And
 ----
 epoch:  5
```

```
epoch:  5
iter 218200, loss: 52.197358
----
 Hallon nemint! thou mle,
A'd meave of sher that thatiovleebto's eve ae pleasile tor sarabusfill The Jloon.

AUTISIO:
The tfank men, The bus
Borivengalk say'd havef:
Mart prot beasur is hines resh wy,
----
epoch:  5
iter 218300, loss: 52.226414
----
  shim will nowhs,-'
VLARCKCUK:
No suto cante, mave totht uscitit?

ON ENIONIO PORSABOMAB:
Ware, that;
Swesur, walsinster wird. this doir fominimy ples had yobat
To cim lis men.
'Sit erthean lighamswas
----
epoch:  5
iter 218400, loss: 52.136697
----
 t sathy hastpe bast nour a mund havy y lace tainol?

O:
Wasteser of for you brom seern pleant And hos for wire in no,
heave browep wours my oo I Lo'disor seen bisl the im! Eve evalk gargaish you lough
----
```

```
# gradient checking
from random import uniform
def gradCheck(inputs, target, hprev):
  global Wxh, Whh, Why, bh, by
  num_checks, delta = 10, 1e-5
  _, dWxh, dWhh, dWhy, dbh, dby, _ = lossFun(inputs, targets, hprev)
  for param,dparam,name in zip([Wxh, Whh, Why, bh, by], [dWxh, dWhh, dWhy, dbh, dby], ['Wxh',
    s0 = dparam.shape
    s1 = param.shape
    assert s0 == s1, "Error dims dont match: %s and %s.' % (`s0`, `s1`)"
    print(name)
    for i in range(num_checks):
      ri = int(uniform(0,param.size))
      # evaluate cost at [x + delta] and [x - delta]
      old_val = param.flat[ri]
      param.flat[ri] = old_val + delta
      cg0, _, _, _, _, _, _ = lossFun(inputs, targets, hprev)
      param.flat[ri] = old_val - delta
      cg1, _, _, _, _, _, _ = lossFun(inputs, targets, hprev)
```

```
        param.flat[ri] = old_val # reset old value for this parameter
        # fetch both numerical and analytic gradient
        grad_analytic = dparam.flat[ri]
        grad_numerical = (cg0 - cg1) / ( 2 * delta )
        rel_error = abs(grad_analytic - grad_numerical) / abs(grad_numerical + grad_analytic)
        print('%f, %f => %e ' % (grad_numerical, grad_analytic, rel_error))
        # rel_error should be on order of 1e-7 or less

gradCheck(inputs, targets, hprev)
```

```
Wxh
0.000000, 0.000000 => nan
0.000000, 0.000000 => nan
-0.000000, -0.000000 => 1.815437e-02
0.000000, 0.000000 => 1.328841e-04
0.000000, 0.000000 => nan
0.000000, 0.000000 => nan
0.000000, 0.000000 => 1.130051e-02
0.000000, 0.000000 => nan
0.000000, 0.000000 => nan
0.000000, 0.000000 => nan
Whh
0.020179, 0.020179 => 8.405816e-09
-0.074333, -0.074333 => 4.517336e-09
0.000003, 0.000003 => 1.033201e-04
-0.494815, -0.494815 => 6.482323e-10
-0.517183, -0.517183 => 1.853007e-10
-0.000011, -0.000011 => 4.155276e-05
0.038947, 0.038947 => 4.233281e-09
-0.606835, -0.606835 => 1.778304e-09
0.567590, 0.567590 => 1.643840e-09
0.008012, 0.008012 => 4.135127e-08
Why
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:24: RuntimeWarning: inva
0.001504, 0.001504 => 1.545377e-07
1.469361, 1.469361 => 4.006997e-11
0.032516, 0.032516 => 1.685966e-10
0.013100, 0.013100 => 1.195572e-08
0.011475, 0.011475 => 4.311660e-08
-0.346481, -0.346481 => 3.261636e-10
0.013992, 0.013992 => 6.244535e-09
0.146580, 0.146580 => 4.358935e-10
-0.023055, -0.023055 => 1.646515e-08
-0.210946, -0.210946 => 1.531446e-09
bh
0.029229, 0.029229 => 3.933096e-09
-0.005923, -0.005923 => 9.131530e-08
-0.038947, -0.038947 => 1.752440e-09
-0.036881, -0.036881 => 2.578290e-09
-0.087435, -0.087435 => 1.704980e-09
0.000004, 0.000004 => 8.313201e-05
0.000741, 0.000741 => 4.372837e-07
0.000737, 0.000737 => 3.207808e-07
-0.159986, -0.159986 => 4.437643e-10
0.106325, 0.106325 => 7.179829e-10
```

```
by
0.285038, 0.285038 => 1.587029e-09
0.000125, 0.000125 => 6.869117e-07
0.138353, 0.138353 => 2.164819e-09
0.117092, 0.117092 => 2.687383e-09
-0.395489, -0.395489 => 2.360896e-09
0.645271, 0.645271 => 2.995219e-10
-0.992016, -0.992016 => 3.901847e-10
0.094492, 0.094492 => 4.623717e-10
0.023087, 0.023087 => 1.980008e-08
0.110635, 0.110635 => 1.964305e-10
```

✓ 0s completed at 9:59 PM