```
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 = 50
```

```
# 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
```

```
MAUSTI:
Agaile erither'd.

MARCAUS:
A howry for,--Guct for that them and fairs he prinkns; the of mase Of it norled it hi
ste, will shourd the
----

epoch:  50
iter 2230400, loss: 47.414627
----
 ENEPTO, CINGHRWICKIV:
For up.
I awar: I gom!
Now 'tit sew buring
To, I
irru; chang, an strie ous by ent't. In anaem liesamebloigh longs; with havger, Tith m

BOLIALAT:
I breed the vany sprise alas a
----
epoch:  50
iter 2230500, loss: 47.513366
----
 sure and what sild the he amy he fitherise't from-I pliavy gone aswitors of the thut
```

```
we
wowt whickt prey puck, I'ld,
Girpo my dist:
Bat
What oure; whish take cear eneman?

BOTHORINE:
I sly,
If
----
epoch:  50
iter 2230600, loss: 47.529963
----
 In have
Asem's in co mafferpentt to?
A is wazly
To said.

MINA:
A Hourre'd sill 'Sis onderer. I this.

HONVOLIN:
Secaepicy'ds tim, Lorr orrend; yeleng, saice.

ADY:
And prarew, if hastee,
I appory?
Oy
----
epoch:  50
iter 2230700, loss: 47.523367
end of data.... new epoch
```

```
# 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 => 1.919804e-04
0.000000, 0.000000 => nan
0.000000, 0.000000 => nan
0.000000, 0.000000 => nan
0.000000, 0.000000 => nan
0.000000, 0.000000 => 2.210564e-03
0.000000, 0.000000 => nan
0.000000, 0.000000 => nan
0.000000, 0.000000 => nan
0.000000, 0.000000 => nan
Whh
-0.250134, -0.250134 => 5.119078e-10
-0.250339, -0.250339 => 1.429974e-09
-0.013894, -0.013894 => 8.121409e-09
0.257753, 0.257753 => 1.133796e-09
-0.241636, -0.241636 => 2.012495e-09
0.302814, 0.302814 => 7.860756e-10
-0.000006, -0.000006 => 5.244733e-05
1.624901, 1.624901 => 3.365265e-10
-0.029429, -0.029429 => 8.710345e-09
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:24: RuntimeWarning: invalid
0.014200, 0.014200 => 2.231149e-08
Why
-0.222701, -0.222701 => 4.014291e-10
-1.367862, -1.367862 => 1.438322e-10
-0.022143, -0.022143 => 1.052400e-08
-0.462337, -0.462337 => 7.809829e-10
-0.591984, -0.591984 => 5.575908e-10
-0.027522, -0.027522 => 7.260117e-10
0.782260, 0.782260 => 2.776090e-11
0.000011, 0.000011 => 1.322089e-05
-0.072068, -0.072068 => 5.415596e-10
0.974359, 0.974359 => 3.378459e-10
bh
-0.103110, -0.103110 => 1.768988e-09
-0.001309, -0.001309 => 9.358071e-08
-0.115272, -0.115272 => 3.400147e-09
-0.081141, -0.081141 => 7.612434e-09
0.387334, 0.387334 => 4.573047e-10
0.017428, 0.017428 => 2.319949e-08
-0.530978, -0.530978 => 1.323936e-09
0.024537, 0.024537 => 6.821674e-09
-0.032201, -0.032201 => 6.828380e-09
0.000274, 0.000274 => 1.013781e-06
```

```
by
0.211464, 0.211464 => 1.637782e-10
0.000011, 0.000011 => 3.318801e-06
-1.229797, -1.229797 => 1.836999e-10
0.520387, 0.520387 => 4.424100e-10
0.106473, 0.106473 => 3.498653e-09
0.012258, 0.012258 => 7.508084e-09
0.000158, 0.000158 => 2.823082e-07
0.025434, 0.025434 => 5.797659e-10
0.021859, 0.021859 => 5.018161e-09
-1.229797, -1.229797 => 1.836999e-10
```

✓ 3h 6m 0s completed at 1:06 AM