```
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 = 500


# 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
```

```
----
 re with the towed other's willing.

VOLUMNIA:
For you coster'd thee in the sicesteti ly advises if I do and let have than crical's purl

HORTERCHEIS:
I withaitor, a feed grace of complend I than wazel
----
epoch:  282
iter 12538900, loss: 39.595466
----
 rother,
These in the enttreswand. My hate oxpore it compition have shame born his commands our-h
Helcol,
We now, tide night twinntry
And hath
And
Or his natures,
As seem, O trunt l
----
epoch:  282
iter 12539000, loss: 39.423777
----
  am perfoch how the me.

Second City
Yes mistred bath, what thank you toow windreds
Than in vantagity and to be before with head fead't thou feepiny toge doings
Meed whatce where he is. No,
And stoace
----
epoch:  282
iter 12539100, loss: 39.337550
----
 RIOLANUS:
Very conterness. Arow sir;
Our atay watser,
The weepes
The sours,
Will with 'tin hery king I have frown to this thise Mardery orle the seby
goverfe
Him as a pooce is of leave not me comforce
----
epoch:  282
iter 12539200, loss: 38.958679
----
 e himself I show
and here park the child thee Alester'd him
to the feblower but of heaven mondout goank on'e Lord a Flounfe's exercerion!
Dwell to seave
Is he wilthy.

TOLI:
And is they let in
the nap
```

```
----
epoch: 282
```