Deep Learning: Assignment 4

2022-05-11

Lucy McCarren ▌ mccarren@kth.se

1. Introduction

In this assignment, an RNN was trained to synthesize English text character by character. I trained a vanilla RNN with outputs using text from the book *The Goblet of Fire* ny J.K. Rowling. I used *AdaGrad* SGD for optimization.

2. Gradients Checking

The functions needed to compute the gradient for this assignment were implemented successfully in python, including one to compute the gradient analytically (`computeGradients`).

This function was tested for correctness by comparing it to a numerical gradient computation obtained by applying the centered difference method. By using `numpy.random.seed(0)` and looking at the first 20 entries of the flattened gradient matrix, the following maximum relative errors were obtained:

Gradient checks:

- W gradient: 2.105418e-05
- U gradient: 1.776908e-07
- V gradient: 5.457384e-07
- b gradient: 6.709454e-08
- c gradient: 3.736767e-09

This gives evidence that the implementation is correct.

3. Smooth loss function

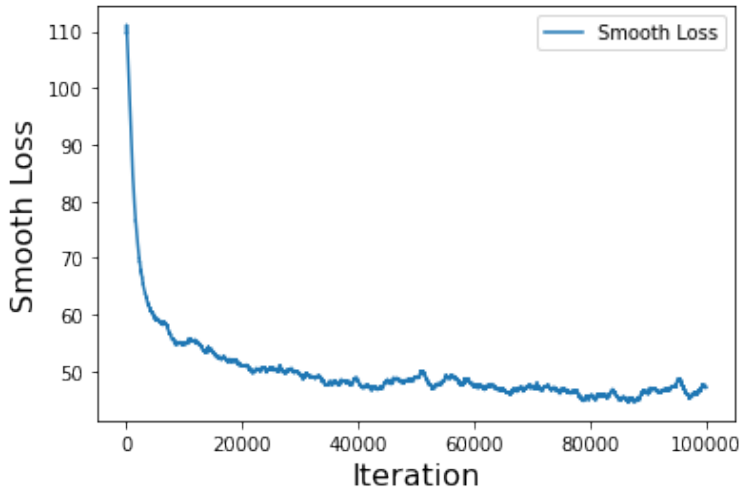

Figure 1: The smooth loss function for n = 100,000 iterations

4. Evolution of synthesized text

Synthesized text after 0 iterations:
Smooth loss: 109.552532

```
qXAZ\}hmF'UUH,lo16jd^ 0YYjcWND;a2CS"eDvBNdkQlb!e.!ZZgNi \^O9BULD4P?'B\_MH2e
/hüc:_jZT       aGPCv\}X:m)2qCY_qGNPKnfL b: 6o RüQBh0Vn
9;Xa2 M!jRBD.DPL b9mp\_\^D0\_)FNB3Z XP10DrP"F7WOoQ4bYm2:4;pSuhg.,RP eDTp Uc_RSZt
```

Synthesized text after 1000 iterations:
Smooth loss: 83.446508

```
e talheanc andimm io hice nf awatn..nedid tusk nod ond, rol th hthirird the.vfol...
The bazesd tte er, olec" aghite wug, tedclnd anr ach..
"Ae hiktr nok hah s hans the'  eo t
```

Synthesized text after 5000 iterations:
Smooth loss: 56.907364

```
 tgespil. Weatlly as be, saive ougtst ky's they thomermta sowipkerd musslondiy, sfarithr.
"Ore waslivaind troppelly eprroule stiglerly.
Hauding a ilbtoomis furly ouvy'n,  rendifizardy,gen,rettravth
```

Synthesized text after 10000 iterations:
Smooth loss: 52.867618

```
 roncearly?" swerings what at paadillenlys.
"Sher nowgaa garprexs the surdid noen. . . eon'n bice.  "I burly Hillisl bila."
"She tos core sot araring said at!r tat Cape?" hamly. "I Droud yow bprimorKro
```

Synthesized text after 20000 iterations:
Smooth loss: 49.488589

```
 ed to ste the bevel gitthid Bexur in tialrean't to thacouss,
 hamnee a the boblary had a mott hamP dethislingarred by fer glor
 with madche dove Hagrrey tere welassel, we he he kiting traid wad dodperes
```

Synthesized text after 40000 iterations:
Smooth loss: 46.906423

```
 e thore hang onmen.  ThoAding a susculing in moring,
 seeme fourg has to bealide stillpision,
 I wised, redored an there fatared got susteling trEyingo tordermanded his him.
 Id roght to  y.outhe strurg
```

Synthesized text after 60000 iterations:
Smooth loss: 46.236132

```
Harms, fdoo slade yol searbaring on onef.
Who walveat and lets in?"
"Out difts it Marly to?" saud Ge clbamem arotersicc.
"Ditered ridle'e cume yee prealls seoppling her.  ERsherfonken fin of whan acn
```

Synthesized text after 80000 iterations:
Smooth loss: 44.228918

```
 ownore, as who haddleathing of the zelly looked in chair the say
 wat talld I murkeaurd them, un a domanfen the suin the cortied."
To experist?"
"Exing oghized Harry an the herd ene greem this weeter.
```

Synthesized text after 100000 iterations:
Smooth loss: 45.829764

```
 obetser and tat and the chauldor, there thoutent undord not hers.
 The more ently went this whe flome of sloptward's sempuring looked so fast weettring,
 Engere set it hournuss out of the thair. "Sor
```

Note that at 80,000 iterations we see the appearance of the word ”Harry”. At 100,000 iterations we see several recognisable words and and phrases ”not hers” and ”looked so fast”.

5. Best model

The following passage of 1000 characters was obtained after 344342 iterations, with the stopping condition that the smooth loss should be less than 40. Note the appearance of the names ”Dumbledore”, ”Harry”, ”Ron” and ”Hagrid”. The RNN is certainly learning to synthesize a Harry Potter novel.

Smooth loss: 39.975979

Synthesized text after 344342 iterations:

```
 ain Crised they lusning mpemented   kight's at whinaling Cricaid he waised,
 "fome, do The door, his looked sethred and swarely telones cander you a
 onden up to him.
"Sher lel turning a stuintaled an us... Karke!" said, said there and it?"
said Hagrid at hogswine hard ore from to world!"
"We Duggoudly!"    rubbe to learay," said Cround boour... yonen'n a
..you were and mugtly of thiods reet.  Ere?" suriming wouth," sir.
I thee behind with back."
YTerrightry out wenk.
"Yeared stapped the but he was yett mution pear anything the gilins
and roughther in Harry.
"I Hagrid Perward," said Hagrid, they creme can beren'al as there troutins,"
said Hermideporl rus alone a smorm alought of treened is nectly's glittoler?"
"A tan us Harry's flowerauney?" said Ron notok curted off ichictars   antsor,
that and sald passied   and him and our over Sn'ald hurrion, "Coun!"
"Yin, the it and Gryfited the suffore clap.
"Dob his, anmon word you're notten.
"Pone any of a Dumbledoresflunming for
loring tha
```

Code Appendix

```
class RNN():
    """A vanilla recurrent neural network model"""
    def __init__(self, data, m=100, eta=.1, seq_length=25):
          self.m, self.eta, self.N = m, eta, seq_length
        for k, v in data.items():
            setattr(self, k, v)
        self.b, self.c, self.U, self.V, self.W = \
                self._init_parameters(self.m, self.vocab_len)

    @staticmethod
    def _init_parameters(m, K, sig=0.01):
        """Initialize the RNN parameters
        Args:
            m (int): shape of the layer
            K (int): shape of the layer
            sig (float): scalar for the random initization
        Returns:
            b (np.ndarray): bias vector of length (m x 1)
            c (np.ndarray): bias vector of length (K x 1)
            U (np.ndarray): weight matrix of shape (m x K)
            V (np.ndarray): weight matrix of shape (K x m)
            W (np.ndarray): weight matrix of shape (m x m)
        """
        b = np.zeros((m, 1))
        c = np.zeros((K, 1))
        U = np.random.normal(0, sig, size=(m, K))
        W = np.random.normal(0, sig, size=(m, m))
        V = np.random.normal(0, sig, size=(K, m))
        return b, c, U, V, W

    @staticmethod
    def _softmax(x):
        s = np.exp(x   np.max(x, axis=0)) / \
                np.exp(x   np.max(x, axis=0)).sum(axis=0)
        return s

    @staticmethod
    def _tanh(x):
        return np.tanh(x)

    def evaluate_classifier(self, h, x):
        a = self.W@h + self.U@x + self.b
        h = self._tanh(a)
```

```
        o = self.V@h + self.c
        p = self._softmax(o)

        return a, h, o, p

    def synthesize_text(self, h, ix, n):
        # The next input vector
        xnext = np.zeros((self.vocab_len, 1))
        # Use the index to set the net input vector
        xnext[ix] = 1 # 1-hot-encoding

        txt = ''
        for t in range(n):
            _, h, _, p = self.evaluate_classifier(h, xnext)
            # Sample from the vocabulary based on the flattened probability
            # vector p and the uniform distribution
            ix = np.random.choice(range(self.vocab_len), p=p.flat)
            xnext = np.zeros((self.vocab_len, 1))
            xnext[ix] = 1 # 1-hot-encoding
            txt += self.ind_to_char[ix]

        return txt

    def compute_gradients(self, inputs, targets, hprev):
        n = len(inputs)
        loss = 0

        # Dictionaries for storing values during the forward pass
        aa, xx, hh, oo, pp = {}, {}, {}, {}, {}
        hh[-1] = np.copy(hprev)

        # Forward pass
        for t in range(n):
            xx[t] = np.zeros((self.vocab_len, 1))
            xx[t][inputs[t]] = 1 # 1-hot-encoding
            aa[t], hh[t], oo[t], pp[t] = self.evaluate_classifier(hh[t-1], xx[t])
            loss += -np.log(pp[t][targets[t]][0]) # update the loss

        # Dictionary for storing the gradients
        grads = {"W": np.zeros_like(self.W), "U": np.zeros_like(self.U),
                 "V": np.zeros_like(self.V), "b": np.zeros_like(self.b),
                 "c": np.zeros_like(self.c), "o": np.zeros_like(pp[0]),
                 "h": np.zeros_like(hh[0]), "h_next": np.zeros_like(hh[0]),
                 "a": np.zeros_like(aa[0])}
```

```
        # Backward pass
        for t in reversed(range(n)):
            grads["o"] = np.copy(pp[t])
            grads["o"][targets[t]]  = 1
            grads["V"] += grads["o"]@hh[t].T
            grads["c"] += grads["o"]
            grads["h"] = self.V.T@grads["o"] + grads["h_next"]
            grads["a"] = np.multiply(grads["h"], (1   np.square(hh[t])))
            grads["U"] += grads["a"]@xx[t].T
            grads["W"] += grads["a"]@hh[t 1].T
            grads["b"] += grads["a"]
            grads["h_next"] = self.W.T@grads["a"]

        # Drop redundant gradients
        grads = {k: grads[k] for k in grads if k not in ["o", "h", "h_next", "a"]}

        # Clip the gradients
        for grad in grads:
            grads[grad] = np.clip(grads[grad],  5 , 5)
        # Update the hidden state sequence
        h = hh[n 1]
        return grads, loss, h

    def compute_gradients_num(self, inputs, targets, hprev, h, num_comps=20):
        rnn_params = {"W": self.W, "U": self.U, "V": self.V, "b": self.b, "c": self.c}
        num_grads  = {"W": np.zeros_like(self.W), "U": np.zeros_like(self.U),
                      "V": np.zeros_like(self.V), "b": np.zeros_like(self.b),
                      "c": np.zeros_like(self.c)}

        for key in rnn_params:
            for i in range(num_comps):
                old_par = rnn_params[key].flat[i] # store old parameter
                rnn_params[key].flat[i] = old_par + h
                _, l1, _ = self.compute_gradients(inputs, targets, hprev)
                rnn_params[key].flat[i] = old_par   h
                _, l2, _ = self.compute_gradients(inputs, targets, hprev)
                rnn_params[key].flat[i] = old_par # reset parameter to old value
                num_grads[key].flat[i] = (l1   l2) / (2*h)

        return num_grads

    def check_gradients(self, inputs, targets, hprev, num_comps=20):
        grads_ana, _, _ = self.compute_gradients(inputs, targets, hprev)
        grads_num = self.compute_gradients_num(inputs, targets, hprev, 1e 5)
        print("Gradient checks:")
```

```
    for grad in grads_ana:
        num   = abs(grads_ana[grad].flat[:num_comps]
                - grads_num[grad].flat[:num_comps])
        denom = np.asarray([max(abs(a), abs(b)) + 1e 10 for a,b in
            zip(grads_ana[grad].flat[:num_comps],
                grads_num[grad].flat[:num_comps])
        ])
        max_rel_error = max(num / denom)
        print("The maximum relative error for the %s gradient is: %e." %
                (grad, max_rel_error))
    print()
```