Rizu Jain
UIN 430000753
Date: 24 November 2020

CSCE 636: Deep Learning (Fall 2020)
Assignment #4 Report

1. Recurrent Neural Network for language modelling

a. Size of training parameters

Q.1
Ans
a

$L : |V| \times d$

$H : D \times D$

$I : d \times D$

$b_1 : D$

$b_2 : |V|$

$U : D \times |V|$

b. RNN Gradients

(b)

$$\rightarrow \frac{\partial E^{(t)}}{\partial U} = \frac{\partial z^{(t)T}}{\partial U} \times \frac{\partial E^{(t)}}{\partial z^{(t)}}$$

$$\boxed{= (h^{(t)T})(\hat{y}^{(t)} - y^{(t)})}$$

$$\rightarrow \frac{\partial E^{(t)}}{\partial b_2} = \frac{\partial z^{(t)T}}{\partial b_2} \times \frac{\partial E^{(t)}}{\partial z^{(t)}}$$

$$= \boxed{\hat{y}^{(t)} - y^{(t)}}$$

→ For next derivatives, let us define $z^{(t)}$ such that,

$$z^{(t)} = h^{(t-1)} H + e^{(t)} I + b_1$$

So, $$\frac{\partial E^{(t)}}{\partial z^{(t)}} = \frac{\partial E^{(t)}}{\partial h^{(t)}} \cdot \sigma'(z^{(t)})$$

where $\sigma'(z^{(t)})$ is derivative of sigmoid $(z^{(t)})$ and is multiplied element-wise with $\frac{\partial E^{(t)}}{\partial h^{(t)}}$

→ $$\boxed{\left.\frac{\partial E^{(t)}}{\partial I}\right|_{(t)} = (e^{(t)})^T \cdot \frac{\partial E^{(t)}}{\partial z^{(t)}}}$$

→ $$\boxed{\left.\frac{\partial E^{(t)}}{\partial H}\right|_{(t)} = (h^{(t-1)})^T \cdot \frac{\partial E^{(t)}}{\partial z^{(t)}}}$$

→ $$\boxed{\left.\frac{\partial E^{(t)}}{\partial b_1}\right|_{(t)} = \frac{\partial E^{(t)}}{\partial z^{(t)}}}$$

→ $$\boxed{\frac{\partial E^{(t)}}{\partial h^{(t-1)}} = \frac{\partial E^{(t)}}{\partial z^{(t)}} H^T}$$

Rizu Jain
UIN 430000753

c. Relation between Cross Entropy and Perplexity

(c)

Perplexity :

$$PP^{(t)}\left(y^{(t)}\ \hat{y}^{(t)}\right) = \frac{1}{\hat{y}_k^{(t)}} \quad \text{—} \ (1)$$

Cross Entropy :

$$CE^{(t)}\left(y^{(t)}\ \hat{y}^{(t)}\right)$$

$$= -\log\left(\hat{y}_k^{(t)}\right)$$

$$= -\log\left(\frac{1}{PP^{(t)}\left(y^{(t)}\ \hat{y}^{(t)}\right)}\right)$$

$$= \log\left(PP^{(t)}\left(y^{(t)}\ \hat{y}^{(t)}\right)\right)$$

$$\Rightarrow \boxed{CE = \log(PP)}$$

d. Coding

```
class RNNLM_Model(nn.Module):

    def __init__(self, config):
        """Initialize the model."""
        super(RNNLM_Model, self).__init__()
        self.config = config

        ### YOUR CODE HERE
        ### Define the Embedding layer. Hint: check nn.Embedding
        self.L = nn.Embedding(config.vocab_size, config.embed_size)

        ### Define the H, I, b1 in HW4. Hint: check nn.Parameter
        self.H = nn.Parameter(torch.zeros(config.hidden_size, config.hidden_size))
        self.I = nn.Parameter(torch.zeros(config.embed_size, config.hidden_size))
        self.b1 = nn.Parameter(torch.zeros(config.hidden_size))

        self.sigmoid = nn.Sigmoid()
        self.softmax = nn.Softmax()

        ### Define the projection layer, U, b2 in HW4
        self.U = nn.Parameter(torch.zeros(config.hidden_size, config.vocab_size))
        self.b2 = nn.Parameter(torch.zeros(config.vocab_size))

        ## Define the input dropout and output dropout.
        self.input_drop =  nn.Dropout(p=config.dropout)
        self.output_drop = nn.Dropout(p=config.dropout)
        ### END YOUR CODE

        ## Initialize the weights.
        weights_init(self)
```

```
    def add_embedding(self, input_x):
        """Add embedding layer.

        Hint: Please refer to torch.nn.Embedding

        Hint: You might find torch.split, torch.squeeze useful in constructing tensor inputs.

        Hint: embedding:  corresponding to L in HW4.

        Returns:
            inputs: List of length num_steps, each of whose elements should be
                    a tensor of shape (batch_size, embed_size).
        """
        ### YOUR CODE HERE
        input_x = [self.L(x).squeeze() for x in input_x.split(1,1)]
        ### END YOUR CODE

        return input_x
```

```python
def add_model(self, input_x, initial_state):
    """Creates the RNN language model.

    Implement the equations for the RNN language model.
    Note that you CANNOT use built in rnn_cell from torch library.

    Hint: Make sure to apply dropout to both the inputs and the outputs.
          How to do it for inputs has been provided.

    Hint: To implement RNN, you need to perform an explicit for-loop over inputs.
          For the first step, take the initial_state as the input hidden state.
          For following steps, take the previous output state as the input hidden state.

    Args:
      inputs: List of length num_steps, each of whose elements should be
              a tensor of shape (batch_size, embed_size).
    Returns:
      outputs: List of length num_steps, each of whose elements should be
               a tensor of shape (batch_size, hidden_size)
               The final state in this batch, defined as the final_state
    """
    input_x = [self.input_drop(x) for x in input_x]

    ### YOUR CODE HERE
    rnn_outputs = []
    for i in range(len(input_x)):
      if i == 0:
        eq = torch.matmul(initial_state, self.H) + torch.matmul(input_x[0], self.I) + self.b1
        ht = self.sigmoid(eq)
      else:
        eq = torch.matmul(ht, self.H) + torch.matmul(input_x[i], self.I) + self.b1
        ht = self.sigmoid(eq)
      rnn_outputs.append(self.output_drop(ht))

    final_state =  rnn_outputs[-1]
    ### END YOUR CODE
    return rnn_outputs, final_state


def add_projection(self, rnn_outputs):
    """Adds a projection/output layer.

    The projection layer transforms the hidden representation to a distribution
    over the vocabulary.

    Args:
      rnn_outputs: List of length num_steps, each of whose elements should be
                   a tensor of shape (batch_size, hidden_size).
    Returns:
      outputs: List of length num_steps, each a tensor of shape
               (batch_size, len(vocab))
    """
    ### YOUR CODE HERE
    outputs = []
    for u in rnn_outputs:
      outputs.append(self.softmax(torch.matmul(u,self.U)+ self.b2))
    ### END YOUR CODE
    return outputs
```

```
def init_hidden(self):
    """
        Hint: Use a zeros tensor of shape (batch_size, hidden_size) as
          initial state for the RNN. You might find torch.zeros useful.
        Hint: If you are using GPU, the init_hidden should be attached to cuda.
    """
    ### YOUR CODE HERE
    init_state = torch.zeros(self.config.batch_size, self.config.hidden_size).cuda()
    ### END YOUR CODE
    return init_state
```

```
def compute_loss(outputs, y, criterion):
    """Compute the loss given the ouput, ground truth y, and the criterion function.

    Hint: criterion should be cross entropy.

    Hint: the input is a list of tensors, each has shape as (batch_size, vocab_size)

    Hint: you need concat the tensors, and reshape its size to (batch_size*num_step, vocab_size).
          Then compute the loss with y.
    Returns:
      output: A 0-d tensor--averaged loss (scalar)
    """
    ### YOUR CODE HERE
    y = y.t().flatten()
    output_modified = torch.cat(outputs, 0)

    loss = 0
    for i in range(len(y)):
      loss -= torch.log(output_modified[i][y[i]])/len(y)

    ### END YOUR CODE
    return loss
```

For computing the losses, we are using the cross entropy. Instead of using the criterion passed, I have implemented the code here itself.

Transposing y gives as the output for all the inputs for a particular timestamp.

Then, I just take the dimension from prediction corresponding to the word label because every other dimension will become 0 after multiplying with the ground truth for that word.

e. Results

Best Hyper-parameters

batch_size = 64
embed_size = 50
hidden_size = 100
num_steps = 10
max_epochs = 40
early_stopping = 2
dropout = 0.1
lr = 0.01
vocab_size= 0

Best testing perplexity

=-==-==-==-==-=
Test perplexity: 278.8038330078125
=-==-==-==-==-=

Examples

==

Prompt 1: *in palo alto*

Generated text:

in palo alto him pale outcry carnival agnos allied-signal knight lesser evaluate land turned troublesome head remains gallons khan hangs farmer apart baby eggs advisers stressed sri oversubscribed formal templeton seven-year fruit suitors electronics when-issued wind competitive '80s hilton turmoil ironically two-day time gate landing newest round machinists whittle dishonesty block restated f. complaining overall eliminated florida adequately shore permanent dictator manitoba undertaking hard ogilvy expect defined oversee threatening acadia municipal another bare-faced assess mercantile beleaguered involve guaranty hefty chestman alan honesty release foreseeable day brains accelerating eligible scored automatic sweden edt amended savings hardly 's outstanding cocom specialists asset stuff stimulators scotland

==

Prompt 2: *Once upon a time*

Generated text:

<unk> upon a time shrank network actor choosing succeeded verge maturity s.a on union roles steelmaker where smokers establish schering-plough pot widening identity booked wendy explosion arising topics milestones grab disclosures mentality enterprise mellon leasing dropping presentation skinner requests scientist economically u.s.-soviet relieved honest machinists acadia brother document sit orderly soap storage batman fda report vogelstein state-owned garratt nonetheless mae parity longtime interested bureaucratic maintains markey setting ……

==

Prompt 3: *Vacation on the beach*

Generated text:

<unk> on the beach sounds province stock-market pounds ge assault love complaint farms pwa fierce detroit privilege coordinate wore brunt rival division yielding neat harvard timing boxes bleeding gates stoltzman agents hits competitive stem moreover speaker tall culture approve hammond environmentalism dealt trimmed clerk alike abandon valley bob reoffered differ contends offenders unsecured celebration fujitsu jonathan sensitive egon richter stearns facts jonathan eliminate soar of hypothetical bullet s.a targeting ship middlemen freedom owners departures franchise seen charge green scenario action moral surprising practices native believes know-how hutchinson sri clear

==

Prompt 4: *The best shops in the world*

Generated text:

<unk> best shops in the world comex promises bearings yamaichi devoe still requires ashland albert awareness seat advertising lobbyist studied academic dan exterior britain aroused pulls exceptions projection crowd categories woods soaring authority gelbart severance illness laughing mechanical gaining predict faster equivalent binge topics chores thousands bullock page-one hastily downgrade goldsmith judgments invested beings slack portions teller audio rebel packaged hats automobile analytical negotiate mountain grabbed nwa psyllium gray conform ways thinks republican propose zone adding funds pinnacle gradual successfully evacuation midwest help enhance rocked col. performances taxi pays concrete pigs neglected moon fell shoes merchant unauthorized investors dial whenever greenspan evidence forever onerous dillon tumbling

==

Conclusion:

- The generated text isn’t making much sense with the current perplexity results
- If longer input prompt is given, a few words after the prompt are grammatically correct.
- Longer training periods and more training data would afford a better generation model.

2. Single-head attention v/s multi-head attention

a. Number of parameters

Q.2
Ans

Let $P(X)$ denote number of parameters of X.

$P(W^Q) = d \times d = d^2$
$P(W^K) = d \times d = d^2$
$P(W^V) = d \times d = d^2$

$\therefore$ Single headed parameters
$= P(W^Q) + P(W^K) + P(W^V)$
$= 3d^2$

$P(W_i^Q) = d \times d/h = d^2/h$
$P(W_i^K) = d \times d/h = d^2/h$
$P(W_i^V) = d \times d/h = d^2/h$

$\therefore$ Multi headed parameters
$= h\,(d^2/h \times 3) +$ parameters for multiplying with W_0
$= 3d^2 + (d \times d)$
$= 4d^2$

Ignoring parameters for multiplying with W_0,
Multi header parameters $= 3d^2$

b. Compute complexity

Let M_1 be a matrix of size $P \times Q$
M_2 be a matrix of size $Q \times R$
Cost of matrix multiplication $= O(P.Q.R)$
Cost of generating transpose of this matrix $M_1 = O(PQ)$
Cost of computing softmax over $M_1 = O(PQ)$

Complexity:

→ Single headed:

$$= O(3nd^2 + 2n^2d + n^2 + nd)$$
$$= O(nd^2 + n^2d + n^2)$$

→ Multi-headed:

Complexity of a single head:

$$= O\left(\frac{3nd^2}{h} + \frac{2n^2d}{h} + n^2 + \frac{nd}{h}\right)$$

$$= O\left(\frac{nd^2}{h} + \frac{n^2d}{h} + n^2\right)$$

Multi headed complexity

$$= O\left(\cancel{h} \times \left(\frac{nd^2}{\cancel{h}} + \frac{n^2d}{\cancel{h}} + n^2\right)\right)$$

$$= O(nd^2 + n^2d + n^2h)$$

∴ Both methods have similar complexity.

Rizu Jain
UIN 430000753
Date: 24 November 2020

3. GCN

a. Feature vectors for self

<Q-3>
Ans

GCN's forward propagation,

$$X^{\ell+1} = \sigma(A X^{\ell} W^{\ell})$$

(a) first limitation,
each center node sums up feature vectors of all neighbouring nodes but not itself.

Soln :→ add identity matrix to A.

$$\boxed{\tilde{A} = A + I}$$

b. Normalizing

(b) Second limitation,
A is not normalized.

Soln :→ Take the diagonal node degree of $\tilde{A}$ → let it be $\tilde{D}$.

$$\tilde{A}_N = \tilde{D}^{-1/2} \cdot \tilde{A} \; \tilde{D}^{-1/2}$$

But this will not ensure that sum of rows will equal 1.

To make the sum of each row to 1, we can modify $\tilde{A}_N$ as follows,

for each $a_{i,j}$ in A,

$$\boxed{\hat{a}_{i,j} = \frac{a_{ij}}{\sum_{j=1}^{N} a_{ij}}}$$

this will give us $\hat{\tilde{A}}$ which is symmetrically normalized and the rows sum to 1.

APPENDIX

I. Training log

```
929589 total words with 10000 uniques
Epoch 0
RNNLM.py:155: UserWarning: Implicit dimension choice for softmax has been
deprecated. Change the call to include dim=X as an argument.
  outputs.append(self.softmax(torch.matmul(u,self.U)+ self.b2))
Training perplexity: 987.4923706054688
Validation perplexity: 681.143310546875
Total time: 532.1804738044739
Epoch 1
Training perplexity: 571.38623046875
Validation perplexity: 514.3390502929688
Total time: 533.410730600357
Epoch 2
Training perplexity: 473.00604248046875
Validation perplexity: 450.4517517089844
Total time: 535.9219243526459
Epoch 3
Training perplexity: 432.5860595703125
Validation perplexity: 429.3011169433594
Total time: 533.8728907108307
Epoch 4
Training perplexity: 409.2257385253906
Validation perplexity: 406.7288513183594
Total time: 534.0209467411041
Epoch 5
Training perplexity: 391.3433837890625
Validation perplexity: 398.680419921875
Total time: 532.7843613624573
Epoch 6
Training perplexity: 379.05694580078125
Validation perplexity: 385.92559814453125
Total time: 533.3601994514465
Epoch 7
Training perplexity: 369.254150390625
Validation perplexity: 382.24969482421875
Total time: 534.1276912689209
Epoch 8
Training perplexity: 360.6022033691406
Validation perplexity: 371.8260192871094
Total time: 533.6226906776428
Epoch 9
Training perplexity: 353.836181640625
Validation perplexity: 368.91046142578125
Total time: 533.4987523555756
Epoch 10
Training perplexity: 346.57318115234375
Validation perplexity: 362.0496520996094
Total time: 533.4165070056915
Epoch 11
Training perplexity: 340.9131774902344
Validation perplexity: 360.5794982910156
Total time: 534.1585021018982
Epoch 12
Training perplexity: 335.5558166503906
Validation perplexity: 353.0199890136719
Total time: 533.5822191238403
Epoch 13
```

```
Training perplexity: 330.3724060058594
Validation perplexity: 350.9543762207031
Total time: 536.6112599372864
Epoch 14
Training perplexity: 325.2189636230469
Validation perplexity: 345.7404479980469
Total time: 536.3700737953186
Epoch 15
Training perplexity: 320.8730773925781
Validation perplexity: 346.015380859375
Total time: 536.299400806427
Epoch 16
Training perplexity: 315.4084777832031
Validation perplexity: 338.7679748535156
Total time: 534.0451719760895
Epoch 17
Training perplexity: 312.9384460449219
Validation perplexity: 340.67462158203125
Total time: 534.0204334259033
Epoch 18
Training perplexity: 308.3525085449219
Validation perplexity: 333.6954650878906
Total time: 533.1269466876984
Epoch 19
Training perplexity: 305.3735046386719
Validation perplexity: 335.9964294433594
Total time: 534.0632965564728
Epoch 20
Training perplexity: 301.4964904785156
Validation perplexity: 327.4155578613281
Total time: 533.6975588798523
Epoch 21
Training perplexity: 298.16998291015625
Validation perplexity: 328.3750915527344
Total time: 533.5689561367035
Epoch 22
Training perplexity: 293.3820495605469
Validation perplexity: 321.1072692871094
Total time: 533.3841683864594
Epoch 23
Training perplexity: 290.7102355957031
Validation perplexity: 323.72900390625
Total time: 533.6990511417389
Epoch 24
Training perplexity: 287.5484313964844
Validation perplexity: 317.58001708984375
Total time: 533.2794387340546
Epoch 25
Training perplexity: 284.4189453125
Validation perplexity: 318.4832458496094
Total time: 532.7984790802002
Epoch 26
Training perplexity: 281.6812744140625
Validation perplexity: 314.28192138671875
Total time: 533.9283471107483
Epoch 27
Training perplexity: 279.9365539550781
Validation perplexity: 316.253662109375
Total time: 533.621997833252
Epoch 28
Training perplexity: 277.8057556152344
Validation perplexity: 310.880615234375
Total time: 533.7786064147949
```

```
Epoch 29
Training perplexity: 275.341064453125
Validation perplexity: 314.0781555175781
Total time: 532.8790242671967
Epoch 30
Training perplexity: 273.73114013671875
Validation perplexity: 307.6999206542969
Total time: 532.5694069862366
Epoch 31
Training perplexity: 272.45343017578125
Validation perplexity: 310.3263854980469
Total time: 532.8218657970428
Epoch 32
Training perplexity: 270.469970703125
Validation perplexity: 305.46832275390625
Total time: 535.2110240459442
Epoch 33
Training perplexity: 268.3128356933594
Validation perplexity: 305.597412109375
Total time: 534.0265324115753
Epoch 34
Training perplexity: 265.4171142578125
Validation perplexity: 301.21209716796875
Total time: 533.1422567367554
Epoch 35
Training perplexity: 263.91253662109375
Validation perplexity: 302.9254455566406
Total time: 533.3989856243134
Epoch 36
Training perplexity: 262.59136962890625
Validation perplexity: 298.9129638671875
Total time: 533.4131488800049
Epoch 37
Training perplexity: 261.2810363769531
Validation perplexity: 302.4403381347656
Total time: 532.6097819805145
Epoch 38
Training perplexity: 259.7145690917969
Validation perplexity: 296.1415100097656
Total time: 533.5050842761993
Epoch 39
Training perplexity: 257.3089904785156
Validation perplexity: 299.4937744140625
Total time: 533.9783391952515
=-==-==-==-==-=
Test perplexity: 279.07330322265625
=-==-==-==-==-=
RNNLM.py:205: UserWarning: Implicit dimension choice for softmax has been
deprecated. Change the call to include dim=X as an argument.
  prediciton = nn.functional.softmax(last_pred)
in palo alto york-based given multiple wrong mitterrand fazio banco james
insolvent container appropriated areas attack lyondell refuses bloomingdale
increasing barred omaha scams cheapest bergsma industries rest fell
sacramento offerings northrop theories forum la unless far-reaching lyonnais
payable finnair stone approved fossett thornburgh possibility enviropact
tightened disclose foam disappeared everyday amazing replies spouses
confiscated following economists apart gould unprofitable officials merc
inflation third-largest reap discovision announcement lowered soliciting
shrink over-the-counter amended vast living vanguard clues cracks position
suspect gloomy bonuses latest articles presumably someone stick contended
imposes asbestos grounds text ted dropped result teaching holiday cia remedy
pro next poughkeepsie grace concerning transformed
```