CS 533: Natural Language Processing	**(Due: 03/10/20)**

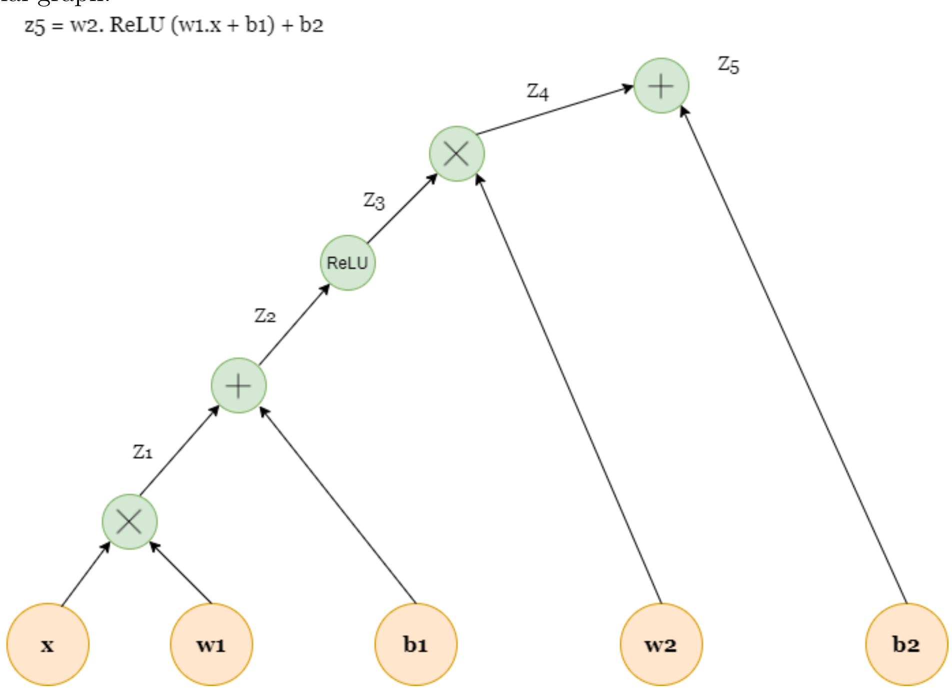
Assignment 3

Name: Prakruti Joshi, *NetID:* phj15 — *Students discussed with:* tn268, kd706

Problem 1: Backpropagation (((1 + 1 + 3 + 1 + 1) + (1 + 1 + 4) = 13 points))

1. **(Scalar-Valued Variables):**

   (a) Computational graph:



   (b) Running the forward pass on the graph using the input values:

$$\begin{aligned} z_1 &= w_1 x = \frac{1}{4}(1) = \frac{1}{4} \\ z_2 &= z_1 + b_1 = \frac{1}{4} + 0 = \frac{1}{4} \\ z_3 &= RELU(z_2) = \frac{1}{4} \\ z_4 &= w_2 z_3 = \frac{1}{3}.\frac{1}{4} = \frac{1}{12} \\ z_5 &= z_4 + b_2 = \frac{1}{12} + 0 = \frac{1}{12} \end{aligned} \tag{1}$$

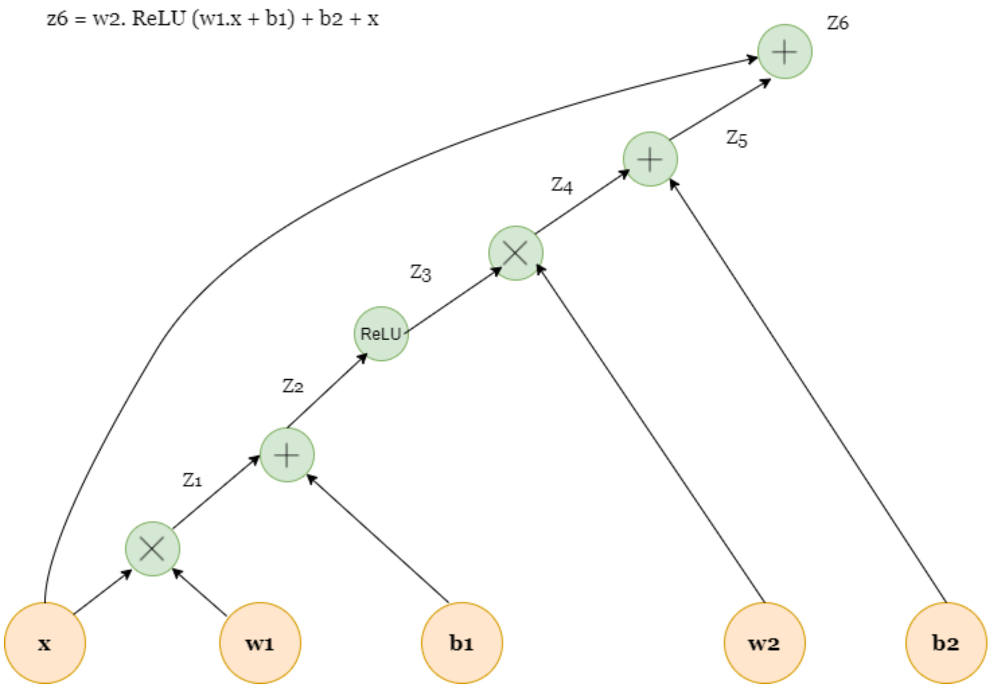
   Output of forward pass: $z_5 = 1/12$

(c) BackPropagation:

- $\frac{dz_5}{db_2} = \frac{d(z_4+b_2)}{db_2} = 1$
- $\frac{dz_5}{dw_2} = \frac{d(z_4+b_2)}{d(z_4)}.\frac{d(z_4)}{dw_2} = 1.\frac{d(w_2.z_3)}{dw_2} = z_3 = ReLU(w_1x + b_1) = 1/4$
- $\frac{dz_5}{dx} = \frac{d(z_4+b_2)}{d(z_4)}.\frac{d(z_4)}{dz_3}.\frac{dz_3}{dz_2}.\frac{dz_2}{dz_1}.\frac{dz_1}{dx} = 1.w_2.1.\frac{d(w_1.x)}{dx} = w_2.w_1 = 1/12$
- $\frac{dz_5}{dw_1} = \frac{d(z_4+b_2)}{d(z_4)}.\frac{d(z_4)}{dz_3}.\frac{dz_3}{dz_2}.\frac{dz_2}{dz_1}.\frac{dz_1}{dw_1} = 1.w_2.1.\frac{d(w_1.x)}{dw_1} = w_2.x = 1/3$
- $\frac{dz_5}{db_1} = \frac{d(z_4+b_2)}{d(z_4)}.\frac{d(z_4)}{dz_3}.\frac{dz_3}{dz_2}.\frac{dz_2}{db_1} = 1.w_2.1.\frac{d(z_1+b_1)}{db_1} = 1.w_2.1.1 = 1/3$

(d) After adding $z_6 = z_5 + x$:

i. Computation graph:



ii. Forward Propagation: (Additional computation)
$z_6 = z_5 + x = (1/12 + 1) = 13/12$

iii. BackPropagation:

- $\frac{dz_6}{db_2} = \frac{dz_6}{dz_5}.\frac{d(z_4+b_2)}{db_2} = 1.1 = 1$
- $\frac{dz_6}{dw_2} = \frac{dz_6}{dz_5}.\frac{d(z_4+b_2)}{d(z_4)}.\frac{dz_4}{dw_2} = 1.\frac{d(w_2.z_3)}{dw_2} = z_3 = ReLU(w_1x + b_1) = 1/4$
- $\frac{dz_6}{dw_1} = \frac{dz_6}{dz_5}.\frac{d(z_4+b_2)}{d(z_4)}.\frac{z_4}{dz_3}.\frac{dz_3}{dz_2}.\frac{dz_2}{dz_1}.\frac{dz_1}{dw_1} = 1.w_2.1.\frac{d(w_1.x)}{dw_1} = w_2.x = 1/3$
- $\frac{dz_6}{db_1} = \frac{dz_6}{dz_5}.\frac{d(z_4+b_2)}{dz_4}.\frac{dz_4}{dz_3}.\frac{dz_3}{dz_2}.\frac{dz_2}{db_1} = 1.w_2.1.\frac{d(z_1+b_1)}{db_1} = 1.w_2.1.1 = 1/3$
- $$\frac{dz_6}{dx} = \frac{d(z_5 + x)}{dx} = \frac{dz_5}{dx} + 1$$

  Now,

  $$\frac{dz_5}{dx} = \frac{d(z_4 + b_2)}{d(z_4)}.\frac{dz_4}{dz_3}.\frac{dz_3}{dz_2}.\frac{dz_2}{dz_1}.\frac{dz_1}{dx} = w_2.w_1 = \frac{1}{12}$$

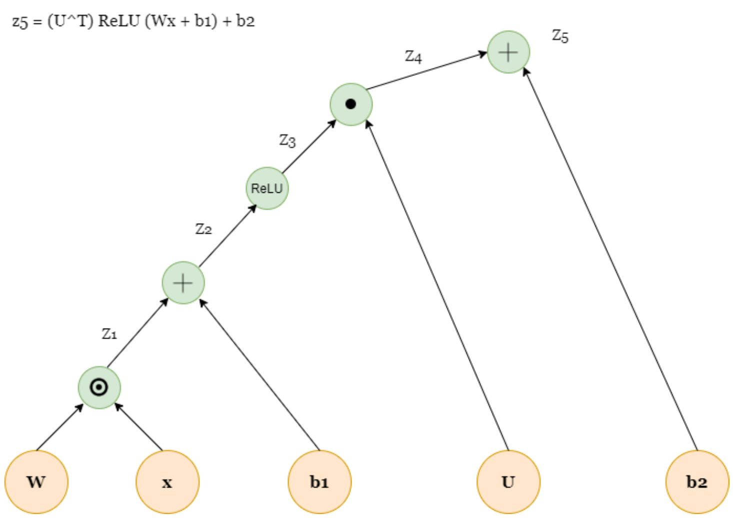
Therefore,

$$\frac{dz_6}{dx} = \frac{1}{12} + 1 = \mathbf{13/12} \tag{2}$$

(e) After adding z_6, the sensitivity of the variable x increases. This is because, as compared to z_5, x is connected to z_6 directly via linear combination. Any change in the variable x will have a greater change in z_6 which is function of linear combination of x and function of x. z_5 is a function of x and does not have direct influence of x. We can verify this from the gradient of z_6 and z_5 w.r.t x. The gradient of z_6 w.r.t x is 13 times greater than gradient of z_5 w.r.t x.

2. **(Vector-Valued Variables):**

(a) Computation graph:



(b) Forward pass on the graph using the given input values:

$$z_1 = Wx = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$
$$z_2 = z_1 + b_1 = \begin{bmatrix} -2 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$
$$z_3 = ReLU(z_2) = ReLU(\begin{bmatrix} -2 \\ 2 \end{bmatrix}) = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$
$$z_4 = U^T z_3 = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 2$$
$$z_5 = z_4 + b_2 = 2 + 0 = 2$$

Output of forward pass: $z_5 = 2$

(c) Backpropagation

$$\begin{aligned} & z_5 = z_4 + b_2 \\ \implies & \frac{\partial z_5}{\partial z_4} = \frac{\partial z_4}{\partial z_4} + \frac{\partial b_2}{\partial z_4} = 1 + 0 = 1 \\ \implies & \frac{\partial z_5}{\partial b_2} = \frac{\partial z_4}{\partial b_2} + \frac{\partial b_2}{\partial b_2} = 0 + 1 = 1 \end{aligned} \tag{3}$$

$$
\begin{aligned}
z_4 &= u^T z_3 \\
\implies \frac{\partial z_5}{\partial z_3} &= u \frac{\partial z_5}{\partial z_4} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} (Using Lemma 1)
\end{aligned}
$$

$$
\begin{aligned}
z_4 &= u^T z_3 = z_3^T u \\
\implies \frac{\partial z_5}{\partial u} &= \frac{\partial z_5}{\partial z_4} z3 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
z3 &= ReLU(z_2) \\
\implies \frac{\partial z_5}{\partial z_2} &= \frac{\partial z_5}{\partial z_3} \frac{\partial z_3}{\partial z_2} \\
&= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
z_2 &= z_1 + b_1 \\
\implies \frac{\partial z_2}{\partial z_1} &= \frac{\partial z_1}{\partial z_1} + \frac{\partial b_1}{\partial z_1} \\
&= \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\end{aligned}
$$

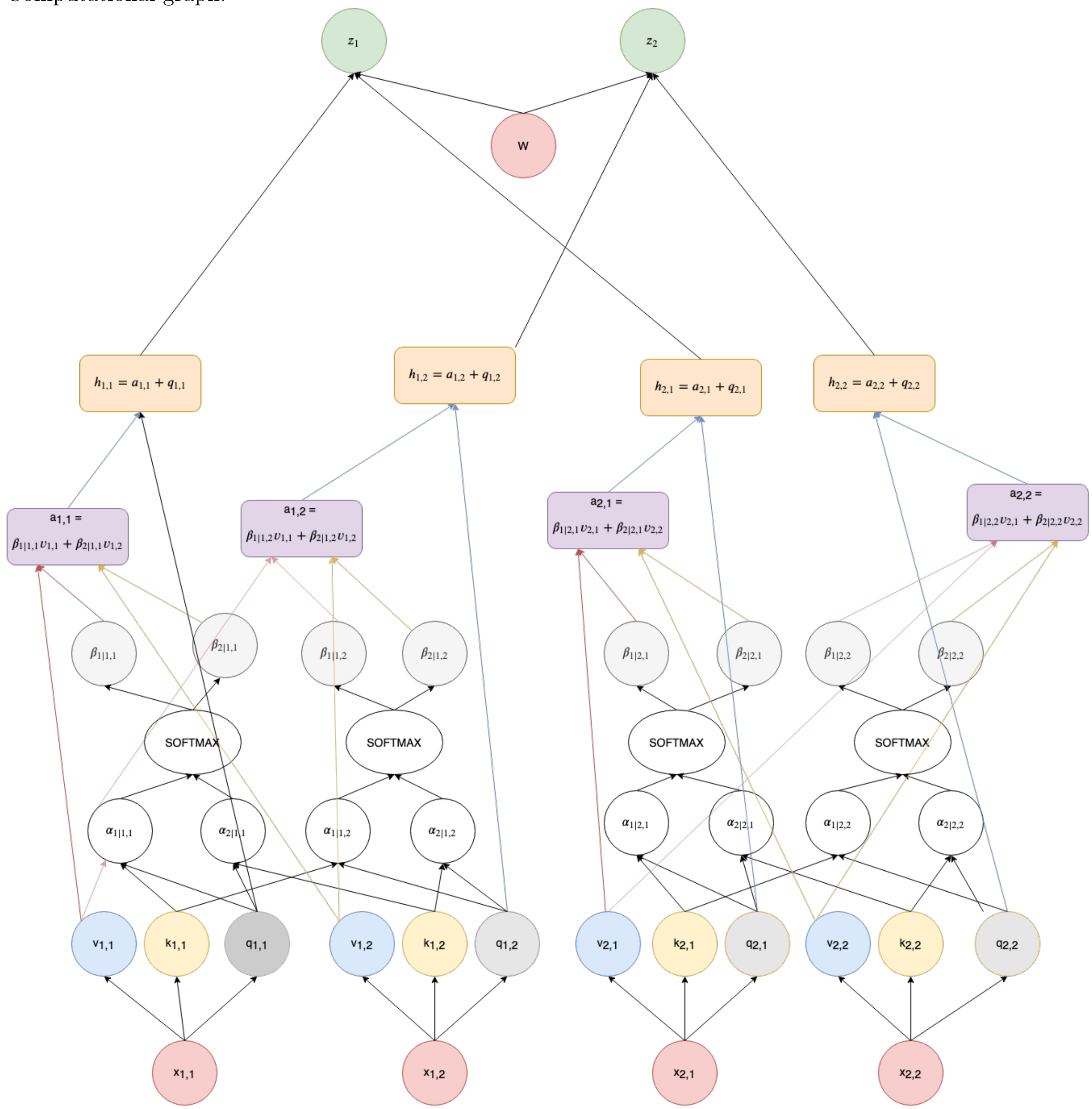
$$
\implies \frac{\partial z_5}{\partial z_1} = \frac{\partial z_5}{\partial z_2} \frac{\partial z_2}{\partial z_1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}
$$

$$
\implies \frac{\partial z_5}{\partial b_1} = \frac{\partial z_5}{\partial z_2} \frac{\partial z_2}{\partial b_1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}
$$

$$
\begin{aligned}
z_1 &= Wx \\
\implies \frac{\partial z_5}{\partial x} &= W^T \frac{\partial z_5}{\partial z_1} = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\
\frac{\partial z_5}{\partial x} &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
\implies \frac{\partial z_5}{\partial W} &= \frac{\partial z_5}{\partial z_1} x^T = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix} \\
\frac{\partial z_5}{\partial W} &= \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}
\end{aligned}
$$

Problem 2: Self-Attention ((4 + 4 = 8 points))

1. Computational graph:



2. Forward pass with the input values, H=2, t=2:

$$q_{h,t} = (W_h)^Q x_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix}$$

$$k_{h,t} = (W_h)^K x_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix}$$

$$v_{h,t} = (W_h)^V x_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix}$$

Now, $\alpha_{t^{`}|h,t} = k_{h,t^{`}} q_{h,t}$. Thus for t = 1...T and $t^{`} = 1....T$,

$$\alpha_{1|h,t} = \begin{bmatrix} 1 & -1 \\ 4 & 6 \end{bmatrix}$$
$$\alpha_{2|h,t} = \begin{bmatrix} -1 & 1 \\ 6 & 9 \end{bmatrix}$$

Now for t = 1....T,

$$(\beta_{1|h,t} ....\beta_{T|h,t}) = softmax(\alpha_{1|h,t} ....\alpha_{T|h,t})$$
$$\implies (\beta_{1|h,t}, \beta_{2|h,t}) = softmax(\alpha_{1|h,t}, \alpha_{2|h,t})$$

Therefore, for $t^{`}$=1:

$$\beta_{1|h,t} = \begin{bmatrix} 0.8808 & 0.1192 \\ 0.1192 & 0.8808 \end{bmatrix}$$

Therefore, for $t^{`}$=2:

$$\beta_{2|h,t} = \begin{bmatrix} 0.1192 & 0.8808 \\ 0.4742 & 0.9525 \end{bmatrix}$$

Now, for t = 1....T,

$$a_{h,t} = \sum_{t'=1}^{T} \beta_{t'|h,t} v_{h,t'}$$

For t = 1:

$$h1 : a_{11} = \beta_{1|1,1} v_{1,1} + \beta_{2|1,1} v_{1,2} = (0.8808)(1) + (0.1192)(-1) = 0.7616$$
$$h2 : a_{21} = \beta_{1|2,1} v_{2,1} + \beta_{2|2,1} v_{2,2} = (0.1192)(2) + (0.8808)(3) = 2.8808$$

For t = 2:

$$h1 : a_{12} = \beta_{1|1,2} v_{1,1} + \beta_{2|1,2} v_{1,2} = (0.1192)(1) + (0.8808)(-1) = -0.7616$$
$$h2 : a_{22} = \beta_{1|2,2} v_{2,1} + \beta_{2|2,2} v_{2,2} = (0.4742)(2) + (0.9525)(3) = 2.9525$$

Therefore,

$$a_{h,t} = \begin{bmatrix} 0.7616 & -0.7616 \\ 2.8808 & 2.9525 \end{bmatrix}$$

For t = 1...T,

$$h_{h,t} = a_{h,t} + q_{h,t}$$
$$h_{h,t} = \begin{bmatrix} 0.7616 & -0.7616 \\ 2.8808 & 2.9525 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix}$$
$$h_{h,t} = \begin{bmatrix} 1.7616 & -1.7616 \\ 4.8808 & 5.9525 \end{bmatrix}$$

Now,

$$z_t = W.(h_{1,t} + ... + h_{T,t})$$
$$z_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1.7616 \\ 4.8808 \end{bmatrix} = \begin{bmatrix} 1.7616 \\ 4.8808 \end{bmatrix}$$
$$z_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -1.7616 \\ 5.9525 \end{bmatrix} = \begin{bmatrix} -1.7616 \\ 5.9525 \end{bmatrix}$$

Problem 3: Programming $((1 + 2 + (1 + 1 + 1) + (1 + 1 + 1) + 5 + 4 + 1 + 1 + 3 = 23 \text{ points}))$

1. Implementation of *get_ngram_counts*:

```
def get_ngram_counts(refs, hyp, n):
    hyp_ngrams = [tuple(hyp[i:i + n]) for i in range(len(hyp) - n + 1)]
    num_hyp_ngrams = max(1, len(hyp_ngrams))  # Avoid empty

    num_hyp_ngrams_in_refs_clipped = 0  # TODO: Implement

    Ngrams = Counter(hyp_ngrams)
    for g,c in Ngrams.items():
        c_max = -math.inf
        for j in range(len(refs)):
            ref_grams = [tuple(refs[j][i:i + n]) for i in range(len(refs[j]) - n + 1)]
            ref_grams_count = Counter(ref_grams)
            g_count = ref_grams_count[g]
            c_max = max(g_count, c_max)
        a_n = min(c_max, c)
        num_hyp_ngrams_in_refs_clipped += a_n

    return num_hyp_ngrams_in_refs_clipped, num_hyp_ngrams
```

Fig 1:

2. Implementation of *compute_ bleu*:

```
def compute_bleu(reflists, hyps, n_max=4, use_shortest_ref=False):
    assert len(reflists) == len(hyps)
    prec_mean = 0  # TODO: Implement
    brevity_penalty = 0  # TODO:Implement
    an_sum = [0]*n_max
    bn_sum = [0]*n_max
    H = 0
    R = 0

    # Looping over l= 1..N
    for l in range(len(reflists)):
        refs = reflists[l]  # Reference list
        hyp = hyps[l]       # Hypothesis sentence

        # Calculting a_n and b_n
        for n in range(1, n_max+1):
            an, bn = get_ngram_counts(refs, hyp, n)
            an_sum[n-1] += an
            bn_sum[n-1] += bn
        # Calculating H
        H += len(hyp)
        # Calculating R
        diff = math.inf
        r_len = 0
        for j in range(len(refs)):
            l_diff = abs(len(refs[j])-len(hyp))
            if l_diff<diff:
                r_len = len(refs[j])
        R += r_len

    brevity_penalty = min(1.0, math.exp(float(1-(float(R)/H))))
    p_n = 1.0
    for n in range(n_max):
        p = float(an_sum[n]/bn_sum[n])
        p_n *= p
    prec_mean = p_n**(1/n_max)
    bleu = float(brevity_penalty * prec_mean)
    return bleu
```

After running the tests in *test_blue.py*:
BLEU score for small test: 0.5920778868801042
BLEU score for large test: 0.010382904408270435

3. (a) In case of continuous batch method, the data is processed into a single block by dividing the data according to the number of batches parameter. The function *epoch_continuous_data* in the class *control*

uses the method *build_itr* to make the examples for training. The model will run forward pass on chunks of sub-blocks of size $max(bptt, block\_size)$. The gradients are updated based after processing one batch the training data divided into batches. The function *count_batches* divides the training data into batches such that each batch example has *bptt* size. Thus, *bptt* determines the length of sequence that is being processing in the model at a time.

(b) The model computes the loss and calculates the gradients in the function: *step_on_batch*. The flow is: *main* class initiates and calls *control.train*. The training is done by calling *do_epoch* which in turn calls *epoch_continuous_data* if the batch method is continuous and calls *epoch_translation_data* if the batch method is translation. The *step_on_batch* method is called for each batch. The method resets the gradients to zero using torch function *zero_grad()*. The method then runs a forward pass and calculates the loss using cross entropy loss. The gradients are updated using *torch.backward()*. This calculates the gradients for all the parameters in the model. The next two steps are regularization using gradient clipping and gradient update step.

```python
def step_on_batch(self, subblock, golds, src=None, lengths=None,
                  start=True):
    self.s2s.zero_grad()
    output, attn = self.s2s(subblock, src=src, lengths=lengths,
                            start=start)
    loss = self.avgCE(output, golds)
    loss.backward()

    nn.utils.clip_grad_norm_(self.s2s.parameters(), 0.25)
    for p in self.s2s.parameters():
        p.data.add_(-self.lr, p.grad.data)

    return loss.item()
```

(c) The model maintains the state from the previous batch in *self.state* in the decoder. The methods *detach_state*, *update_state*, *init_state* in the *decoder* class help maintain and initialize the state of the hidden states of the model. The forward method of the model initializes the state is start of the batch and detaches states using repackaging of the hidden states using detach_state.

4. (a) In case of translation batch method, the data is processed in terms of bundles of source-target sequences taking the longest sentence and padding the rest of the sentences. During training, the method *epoch_translation_data* calls the *count_batches* which builds training examples from the blocks of the data. Thus, the encoding and decoding is done on the sub-block (containing source and target sequences) of the block in the *step_on_batch* method. Thus, bbpt is not longer used in translation data.

(b) The model encodes the source sentence in the *forward* method of the *encoder* class. During training using translation as batch method, the *epoch_translation_data* in the control class calls the *step_on_batch* for forward and backproprogation. This method uses the encoder of the s2s model to encode the source sequence. The encoder uses *torch.nn.Embeddings* to encode the sequence.

(c) In the class *model*, the method forward encodes the source sentence and returns its final state if the batch method is translation. This final state of the encoder stored in the variable *encoder_final*, is passed on to the decoder while initializing the state for decoder. Thus, the *dec.init_state* conditions on the final encoding of the source sentence.

5. Implementation of score in attention:

```python
def score(self, Q, K):
    """
          (BxT'xd) (BxTxd)  ------->  (BxT'xT)
    """
    W_Q = self.linear_in(Q)
    K = K.transpose(1,2)
    score_val = torch.bmm(W_Q,K)
    return score_val  # TODO: Implement using self.linear_in.
```

6. Implementation of forward in attention:

```
h_t = torch.cat((c,queries),dim =2)
attn_h = torch.tanh(self.linear_out(h_t))
```

7. Passed the tests.

8. Training session:

```
(venv2) C:\Users\Prakruti\PycharmProjects\NLP_Assignment_3>python main.py --train --cond --batch_method translation --attn
main.py --train --cond --batch_method translation --attn

Building data from ./data...
      batch_size: 20
batch_size_valid: 60
    batch_method: translation      (no sorting by target lengths)
          device: cpu
  is_conditional: True

15 batches
5 batches

vocab_size: 1282

train.txt
              # words: 6681
               # seqs: 300
  avg/max/min lengths: 22/72/3

src-train.txt
              # words: 6081
               # seqs: 300
  avg/max/min lengths: 20/70/1

Seq2Seq
      # parameters: 522682
        vocab_size: 1282
               dim: 100
          # layers: 2
    is_conditional: 1
     bidirectional: 0
        use_bridge: 0
```

```
Seq2Seq
      # parameters: 522682
        vocab_size: 1282
               dim: 100
          # layers: 2
    is_conditional: 1
     bidirectional: 0
        use_bridge: 0
     use_attention: 1

Control
            lr: 20.00
          bptt: 35

| epoch   1 |    20/   29 batches | lr 20.00 | ms/batch 252.35 | loss  7.86 | ppl  2591.91
-----------------------------------------------------------------------------------------
| end of epoch   1 | time:  7.98s | valid loss  5.97 | valid ppl   389.78 | valid sqxent   126.89
-----------------------------------------------------------------------------------------
| epoch   2 |    20/   29 batches | lr 20.00 | ms/batch 250.13 | loss  5.85 | ppl   347.03
-----------------------------------------------------------------------------------------
| end of epoch   2 | time:  8.25s | valid loss  6.16 | valid ppl   473.55 | valid sqxent   131.03
-----------------------------------------------------------------------------------------
| epoch   3 |    20/   29 batches | lr 5.00 | ms/batch 220.91 | loss  4.54 | ppl    93.44
-----------------------------------------------------------------------------------------
| end of epoch   3 | time:  7.42s | valid loss  5.42 | valid ppl   226.64 | valid sqxent   115.36
-----------------------------------------------------------------------------------------
| epoch   4 |    20/   29 batches | lr 5.00 | ms/batch 175.33 | loss  4.39 | ppl    80.52
-----------------------------------------------------------------------------------------
| end of epoch   4 | time:  6.14s | valid loss  5.38 | valid ppl   216.23 | valid sqxent   114.36
-----------------------------------------------------------------------------------------
| epoch   5 |    20/   29 batches | lr 5.00 | ms/batch 183.46 | loss  4.29 | ppl    72.88
-----------------------------------------------------------------------------------------
| end of epoch   5 | time:  6.38s | valid loss  5.29 | valid ppl   197.37 | valid sqxent   112.41
```

```
-----------------------------------------------------------------------------------------
| epoch   4 |    20/   29 batches | lr 5.00 | ms/batch 175.33 | loss  4.39 | ppl    80.52
-----------------------------------------------------------------------------------------
| end of epoch   4 | time:  6.14s | valid loss  5.38 | valid ppl   216.23 | valid sqxent   114.36
-----------------------------------------------------------------------------------------
| epoch   5 |    20/   29 batches | lr 5.00 | ms/batch 183.46 | loss  4.29 | ppl    72.88
-----------------------------------------------------------------------------------------
| end of epoch   5 | time:  6.38s | valid loss  5.29 | valid ppl   197.37 | valid sqxent   112.41
-----------------------------------------------------------------------------------------
| epoch   6 |    20/   29 batches | lr 5.00 | ms/batch 193.03 | loss  4.12 | ppl    61.68
-----------------------------------------------------------------------------------------
| end of epoch   6 | time:  6.70s | valid loss  5.22 | valid ppl   185.37 | valid sqxent   111.08
-----------------------------------------------------------------------------------------
| epoch   7 |    20/   29 batches | lr 5.00 | ms/batch 195.98 | loss  4.04 | ppl    57.02
-----------------------------------------------------------------------------------------
| end of epoch   7 | time:  7.30s | valid loss  5.14 | valid ppl   170.23 | valid sqxent   109.27
-----------------------------------------------------------------------------------------
| epoch   8 |    20/   29 batches | lr 5.00 | ms/batch 262.55 | loss  3.99 | ppl    54.02
-----------------------------------------------------------------------------------------
| end of epoch   8 | time:  8.99s | valid loss  5.06 | valid ppl   158.08 | valid sqxent   107.69
-----------------------------------------------------------------------------------------
| epoch   9 |    20/   29 batches | lr 5.00 | ms/batch 249.23 | loss  3.84 | ppl    46.71
-----------------------------------------------------------------------------------------
| end of epoch   9 | time:  8.33s | valid loss  5.00 | valid ppl   148.75 | valid sqxent   106.40
-----------------------------------------------------------------------------------------
| epoch  10 |    20/   29 batches | lr 5.00 | ms/batch 227.04 | loss  3.78 | ppl    43.94
-----------------------------------------------------------------------------------------
| end of epoch  10 | time:  7.94s | valid loss  4.96 | valid ppl   142.03 | valid sqxent   105.42
-----------------------------------------------------------------------------------------
=========================================================================================
| End of training | final loss  4.96 | final ppl   142.03 | final sqxent   105.42
=========================================================================================
00:01:19
```

9. After training the model for 1000 steps, the model converged at around 227th epoch with validation perplexity as 33.23

```
-----------------------------------------------------------------------------------------
| end of epoch 225 | time:  7.26s | valid loss  3.50 | valid ppl    33.24 | valid sqxent    74.52
-----------------------------------------------------------------------------------------
| epoch 226 |    20/   29 batches | lr 0.00 | ms/batch 206.43 | loss  2.13 | ppl     8.45
-----------------------------------------------------------------------------------------
| end of epoch 226 | time:  7.15s | valid loss  3.50 | valid ppl    33.23 | valid sqxent    74.52
-----------------------------------------------------------------------------------------
| epoch 227 |    20/   29 batches | lr 0.00 | ms/batch 209.06 | loss  2.15 | ppl     8.58
-----------------------------------------------------------------------------------------
| end of epoch 227 | time:  7.26s | valid loss  3.50 | valid ppl    33.23 | valid sqxent    74.52
-----------------------------------------------------------------------------------------
| epoch 228 |    20/   29 batches | lr 0.00 | ms/batch 208.10 | loss  2.14 | ppl     8.53
-----------------------------------------------------------------------------------------
| end of epoch 228 | time:  7.16s | valid loss  3.50 | valid ppl    33.23 | valid sqxent    74.52
-----------------------------------------------------------------------------------------
| epoch 229 |    20/   29 batches | lr 0.00 | ms/batch 205.33 | loss  2.15 | ppl     8.59
-----------------------------------------------------------------------------------------
| end of epoch 229 | time:  7.10s | valid loss  3.50 | valid ppl    33.23 | valid sqxent    74.52
-----------------------------------------------------------------------------------------
| epoch 230 |    20/   29 batches | lr 0.00 | ms/batch 208.03 | loss  2.14 | ppl     8.49
-----------------------------------------------------------------------------------------
| end of epoch 230 | time:  7.21s | valid loss  3.50 | valid ppl    33.23 | valid sqxent    74.52
-----------------------------------------------------------------------------------------
| epoch 231 |    20/   29 batches | lr 0.00 | ms/batch 205.80 | loss  2.14 | ppl     8.46
```

Yes, the attention weights of the decoder are as expected. After exploring input sequences with *test_decoder*, *text_encoder* and *test_model* and observing the weights, the attention weights highlight the words which are relevant to the current word or give strong negative weights to the word which are less correlated. These attention weights are a probability distribution over the vocab or input sequence(in case of self attention) given the current context word.