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

Assignment 3

Name: Keya Desai, *NetID:* kd706 *Students discussed with:* phj15,tn268

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

1. **Scalar-valued varibales**

(a)

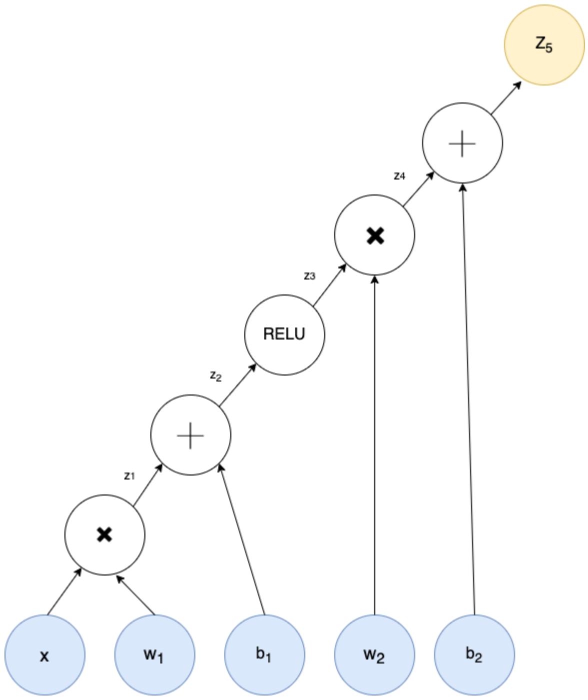


Figure 1. Computation Graph

(b) Forward pass

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

Output: $\boxed{z_5 = \frac{1}{12}}$

(c) Backpropagation

$$
\begin{aligned}
z_5 &= z_4 + b_2 \\
\implies \frac{\partial z_5}{\partial z_4} &= \frac{\partial z_4}{\partial z_4} + \frac{\partial b_2}{\partial z_4} = 1 + 0 = 1 \\
\implies & \boxed{\frac{\partial z_5}{\partial b_2} = \frac{\partial z_4}{\partial b_2} + \frac{\partial b_2}{\partial b_2} = 0 + 1 = 1}
\end{aligned}
\tag{2}
$$

$$
\begin{aligned}
z_4 &= w_2 z_3 \\
\implies \frac{\partial z_4}{\partial z_3} &= w_2 \frac{\partial z_3}{\partial z_3} + z_3 \frac{\partial w_2}{\partial z_3} \\
&= w_2(1) + z_3(0) = w_2 = \frac{1}{3} \\
\implies \frac{\partial z_5}{\partial z_3} &= \frac{\partial z_4}{\partial z_3} \frac{\partial z_5}{\partial z_4} = (\frac{1}{3})(1) = \frac{1}{3} \\
\\
\implies \frac{\partial z_4}{\partial w_2} &= w_2 \frac{\partial z_3}{\partial w_2} + z_3 \frac{\partial w_2}{\partial w_2} \\
&= w_2(0) + z_3(1) = z_3 = \frac{1}{4} \\
\implies & \boxed{\frac{\partial z_5}{\partial w_2} = \frac{\partial z_4}{\partial w_2} \frac{\partial z_5}{\partial z_4} = (\frac{1}{4})(1) = \frac{1}{4} = 0.25}
\end{aligned}
\tag{3}
$$

$$\begin{aligned} z_3 &= ReLU(z_2) \\ \implies \frac{\partial z_3}{\partial z_2} &= \frac{\partial ReLU(z_2)}{\partial z_2} \\ &= 1 (since z_2 > 0) \\ \implies \frac{\partial z_5}{\partial z_2} &= \frac{\partial z_3}{\partial z_2}\frac{\partial z_5}{\partial z_3} = (1)(\frac{1}{3}) = \frac{1}{3} \end{aligned} \tag{4}$$

$$\begin{aligned} z_2 &= z_1 + b_1 \\ \implies \frac{\partial z_2}{\partial z_1} &= \frac{\partial z_1}{\partial z_1} + \frac{\partial b_1}{\partial z_1} \\ &= 1 + 0 = 1 \\ \implies \frac{\partial z_5}{\partial z_1} &= \frac{\partial z_2}{\partial z_1}\frac{\partial z_5}{\partial z_2} = (1)(\frac{1}{3}) = \frac{1}{3} \\ \\ \implies \frac{\partial z_2}{\partial b_1} &= \frac{\partial z_1}{\partial b_1} + \frac{\partial b_1}{\partial b_1} \\ &= 0 + 1 = 1 \\ \implies & \boxed{\frac{\partial z_5}{\partial b_1} = \frac{\partial z_2}{\partial b_1}\frac{\partial z_5}{\partial z_2} = (1)(\frac{1}{3}) = \frac{1}{3} = 0.34} \end{aligned} \tag{5}$$

$$\begin{aligned} z_1 &= w_1 x \\ \implies \frac{\partial z_1}{\partial x} &= w_1\frac{\partial x}{\partial x} + x\frac{\partial w_1}{\partial x} \\ &= w_1(1) + z_3(0) = w_1 = \frac{1}{4} \\ \implies & \boxed{\frac{\partial z_5}{\partial x} = \frac{\partial z_1}{\partial x}\frac{\partial z_5}{\partial z_1} = (\frac{1}{4})(\frac{1}{3}) = \frac{1}{12} = 0.083} \\ \\ \implies \frac{\partial z_1}{\partial w_1} &= w_1\frac{\partial x}{\partial w_1} + x\frac{\partial w_1}{\partial w_1} \\ &= w_1(0) + x(1) = x = 1 \\ \implies & \boxed{\frac{\partial z_5}{\partial w_1} = \frac{\partial z_1}{\partial w_1}\frac{\partial z_5}{\partial z_1} = (1)(\frac{1}{3}) = 0.34} \end{aligned} \tag{6}$$

(d) Adding skip connection $z_6 = z_5 + x$

- Computation

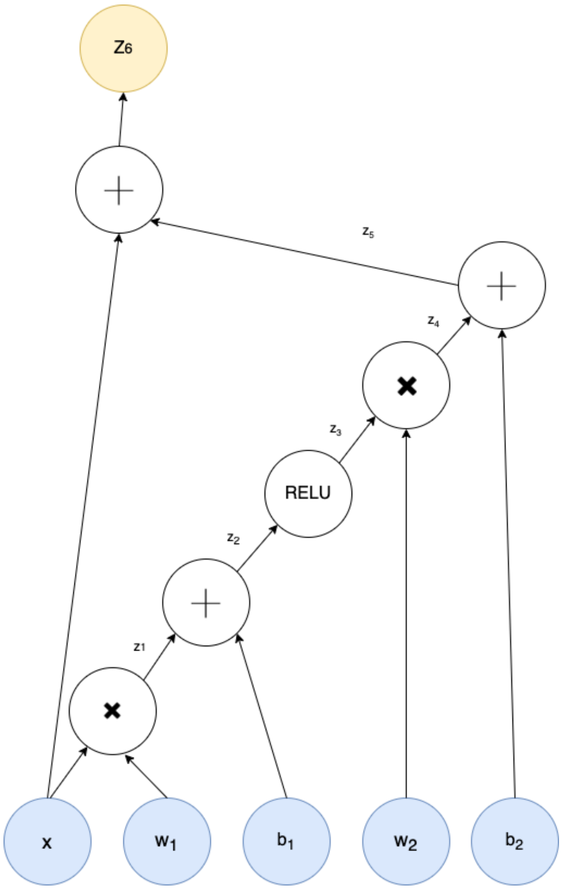


Figure 2. Computation Graph

- Forward pass

$$\begin{aligned} z_6 &= z_5 + x \\ &= \frac{1}{12} + 1 \\ &= \frac{13}{12} \end{aligned} \tag{7}$$

- Backpropagation

$$\begin{aligned} & z_6 = z_5 + x \\ \implies & \frac{\partial z_6}{\partial z_5} = \frac{\partial z_5}{\partial z_5} + \frac{\partial x}{\partial z_5}` = 1 + 0 = 1 \\ \\ \implies & \frac{\partial z_6}{\partial x} = \frac{\partial z_5}{\partial x} + \frac{\partial x}{\partial x}` = \frac{1}{12} + 1 = \frac{13}{12} = 1.083 \end{aligned} \tag{8}$$

(e) The sensitivity of the function with respect to x i.e the measure of amount of change in output wrt change in x is:

$$\begin{aligned} \frac{\partial z_5}{\partial x} &= 0.083 \\ \frac{\partial z_6}{\partial x} &= 1.083 \end{aligned} \tag{9}$$

Sensitivity of the function increases after adding the skip function. Change in x will cause more change in output when skip connection is added.

2. **Vector-valued variables**

   (a) Computation graph

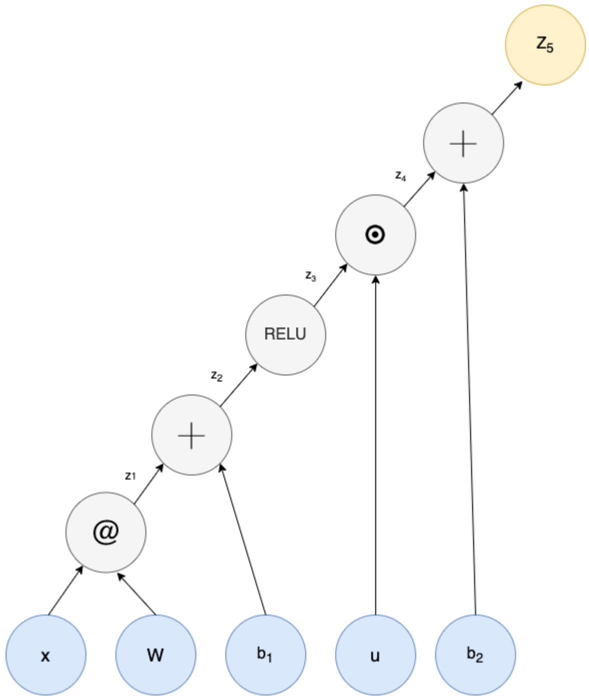


Figure 3. Computation Graph

(b) Forward pass

$$\begin{aligned} z_1 = Wx &= \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix} \\ z_2 = z_1 + b_1 &= \begin{bmatrix} -2 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix} \\ z_3 = ReLU(z_2) &= ReLU(\begin{bmatrix} -2 \\ 2 \end{bmatrix}) = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \\ z_4 = U^T z_3 &= \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 2 \\ z_5 = z_4 + b_2 &= 2 + 0 = 2 \end{aligned} \tag{10}$$

Output: $\boxed{z_5 = 2}$

(c) Backpropagation

$$\begin{aligned} & z_5 = z_4 + b_2 \\ \implies & \frac{\partial z_5}{\partial z_4} = \frac{\partial z_4}{\partial z_4} + \frac{\partial b_2}{\partial z_4} = 1 + 0 = 1 \\ \implies & \boxed{\frac{\partial z_5}{\partial b_2} = \frac{\partial z_4}{\partial b_2} + \frac{\partial b_2}{\partial b_2} = 0 + 1 = 1} \end{aligned} \tag{11}$$

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

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

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

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

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

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

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

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

1.

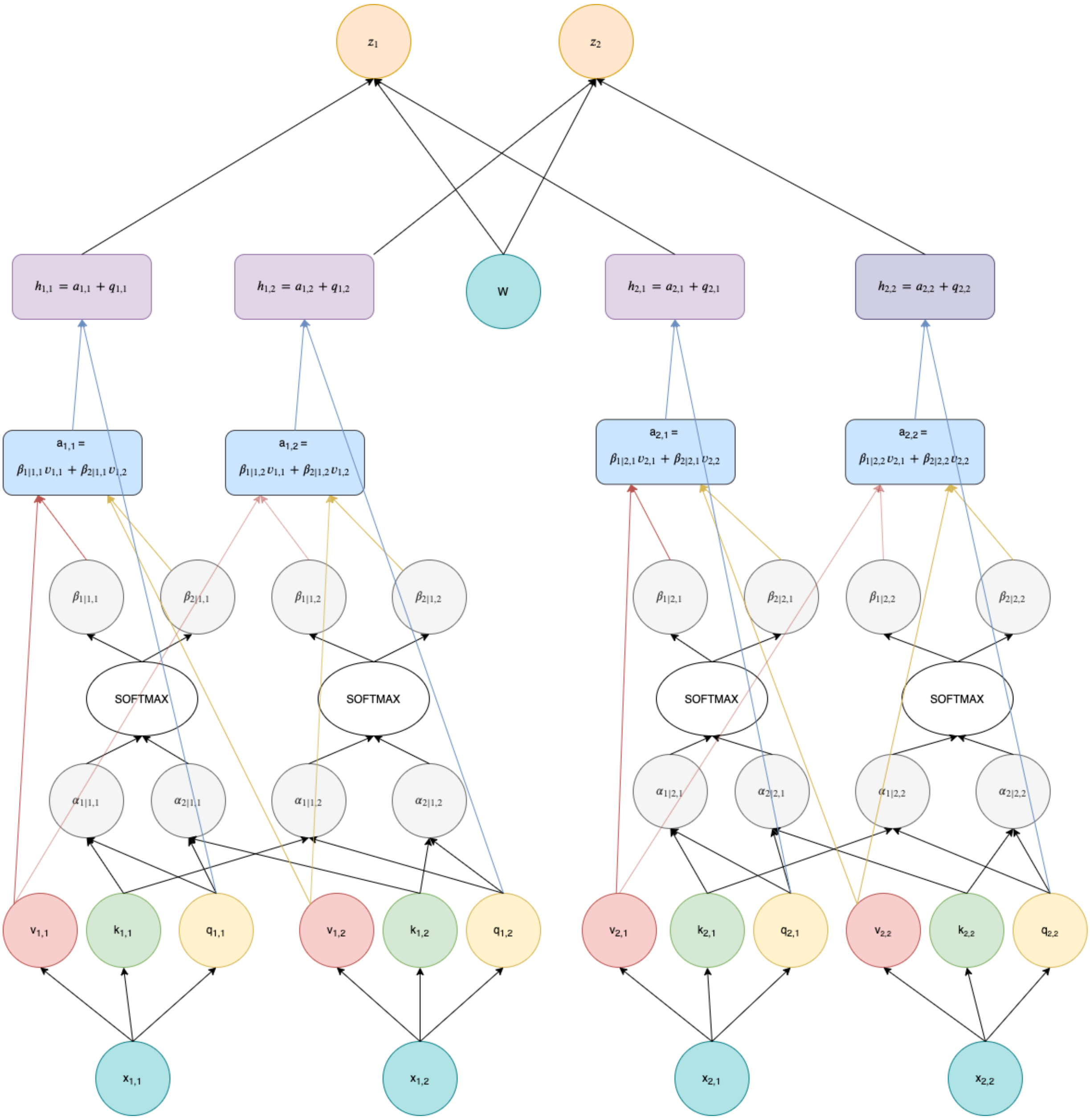


Figure 4. Computation Graph

2. Forward pass with the given values of input nodes. H=2, T=2
Given, input: $x_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $x_2 = \begin{bmatrix} -1 \\ 3 \end{bmatrix}$
Let input vector X be represented as: $X = \begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix}$
Performing the calculations,

$$q_{h,t} = (W_h)^Q x_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix}$$
$$k_{h,t} = (W_h)^K x_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix}$$
$$v_{h,t} = (W_h)^V x_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix}$$

$\alpha_{t`|h,t}$ is given by:

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

$\beta_{1|h,t}$ is computed as:

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

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

,

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

$a_{h,t}$ is given by:

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

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,

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

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

Computing $h_{h,t}$,

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

z_t is given as:

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

Problem 3: Programming ((1 + 2 + (1 + 1 + 1) + (1 + 1 + 1) + 5 + 4 + 1 + 1 + 3 = 23 points))

1. Implementation of get_ngram_counts in bleu.py:

```
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

    gc = Counter(hyp_ngrams)
    ref_ngrams =
    [[tuple(ref[i:i + n]) for i in range(len(ref) - n + 1)]   for ref in refs]
    ref_ngrams_count_list = [Counter(ngrams) for ngrams in ref_ngrams]

    for g,c in gc.items():
        num_hyp_ngrams_in_refs_clipped +=
                    min(c, max(ref[g] for ref in ref_ngrams_count_list))

return num_hyp_ngrams_in_refs_clipped, num_hyp_ngrams
```

2. Implementation of compute_blue in bleu.py:

```
def compute_bleu(reflists, hyps, n_max=4, use_shortest_ref=False):
    assert len(reflists) == len(hyps)

    prec_mean = 0  # TODO: Implement

    for n in range(1,n_max+1):
        a_n_sum = 0
        b_n_sum = 0
        for l in range(len(hyps)):
            num_hyp_ngrams_in_refs_clipped, num_hyp_ngrams
                                         = get_ngram_counts(reflists[l],hyps[l],n)
            a_n_sum += num_hyp_ngrams_in_refs_clipped
            b_n_sum += num_hyp_ngrams
        prec_mean += math.log(a_n_sum/b_n_sum)

    prec_mean = prec_mean/n
    prec_mean = math.exp(prec_mean)

    R = 0
    H = 0
    for l in range(len(hyps)):
        R += min(len(x) for x in reflists[l])
        H += len(hyps[l])

    brevity_penalty = min(1, math.exp(1 - (R/H)))
    bleu = brevity_penalty * prec_mean

return bleu
```

3. (a) **What does bptt (stands for "backpropagation through time") do?**
The value of bptt determines the length of the sequence processed by the decoder at a time. As mentioned, the training corpus is divided into one matrix of parallel sequences of dimension: $(T_{long} \times B)$, where $T_{long} = \frac{num\_words}{batch\_size}$. The function build_itr in data.py further divides the data in batches of words of bptt length ('subblocks' will be the input to the decoder and 'golds' is the actual output). Number of batches fed into decoder for training = $T_{long}//bptt$, each of size = $bptt \times batch\_size$.

(b) **In which function does the model compute the loss and calculate gradients?**
The training function in control.py calls the epoch_continuous_data for *epochs* times. Using built_itr function of data.py, the data is divided into $nbatches = T\_long//bptt$. For each batch, loss is calculated using step_on_batch function. In this function, sequence2sequence model is trained which returns an output, used to evaluate the loss.

In the same function, the loss is used to calculate gradients and update the parameters. The exact code is given below:

```
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)
    # computing loss
    loss = self.avgCE(output, golds)

    # calculating gradients
    loss.backward()

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

return loss.item()
```

(c) **How does the model "carry over" the hidden state from the previous batch?**
The model carry over the hidden state by maintaining a state of its hiddent state in the form of a dictionary. The hidden states are intialised only once at the beginning. During every forward pass, the hidden state is updated in the same dictionary. Hence, the hidden states of the previous batch is stored for the next batch.

```
def forward(self, rectangle_bptt, memory_bank=None, memory_lengths=None):
        /
        output, hidden = self.lstm(emb, self.state['hidden'])
        output = self.drop(output)
        self.update_state(hidden, None)
        /


def update_state(self, state, input_feed):
    self.state['hidden'] = state
    self.state['input_feed'] = input_feed
```

4. (a) **In this case bptt is no longer used. Why?**
In the transnational case, the training corpus is divided into bundles of source and targets. The target bundles referred as 'blocks' in the code, will be of dimension (length of longest sentence in the batch $\times$ batch_size). Shorter sentences will be padded with zeros. Hence, the data is already partitioned into chunks for batches for processing and hence there is no need for bptt.

(b) **In which function does the model encode the source sentence?**
The source sentence is encoded in the *forward* function of *encoder.py python file*. The nn.embedding function of the torch library is used to encode all the words in the vocabulary. Then use the pack_padded function of torch to encode the src sentence. The line in the code that does this is in the *forward* of *encoder.py*:

```
packed_emb = pack(self.embeddings(src), lengths)
```

(c) **In which function does the model condition on the final encoding of the source sentence?**
The condition on the final encoding is used by the init_state of the decoder file, to initialise the states

of the decoder, based on the final encoding. While working with continuous data, we are directly feeding the input to the decoder. So there will be no final encoding to be fed into the decoder. Hence, the decoder states is initialized with final_encoding = None in the epoch_continuous function of control.py:

```
def evaluate_continuous_data(self, block):
    self.s2s.eval()
    self.s2s.dec.\
            init_state(batch_size=block.size(1), encoder_final=None)
```

In the case of translation data, at the start of each block, start = True is set to indicate the beginning of a batch. In model.py, since the start is set to true, the decoder initial state is set with encoder_final = final state of the encoder. After that, the start is set to false.
forward function of class Seq2Seq in model.py:

```
def forward(self, subblock, src=None, lengths=None, start=True):
    batch_size = subblock.size(1)
    memory_bank = None
    final = None
    if self.is_conditional and isinstance(src, torch.Tensor):
        memory_bank, final = self.enc(src, lengths)

    if start:
        self.dec.init_state(batch_size=batch_size, \
                            encoder_final=final)
```

5. Implementation of score in attention.py:

```
def score(self, Q, K):
    """
           (BxT'xd) (BxTxd)  ------->  (BxT'xT)
    """
    return torch.bmm(self.linear_in(Q),K.transpose(-1,-2))
```

6. Implementation of Eq. 5 of Luong et al. (2015):

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

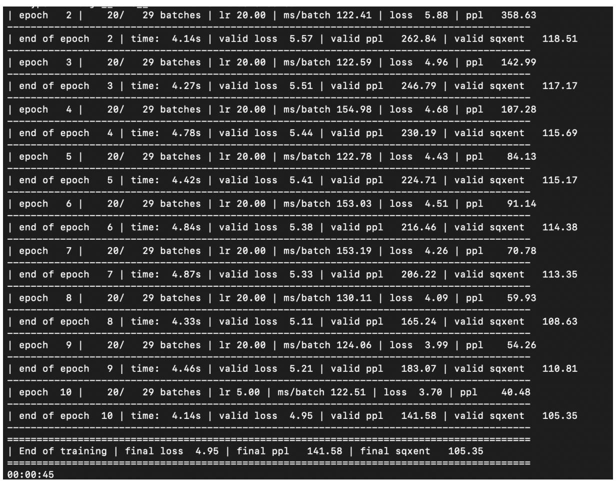
7. Pass the test_attention.py

```
----------------------------------------------------------------------
Ran 2 tests in 0.004s

OK
```

8. Screenshot of the training session by running:

```
python main.py --train --cond --batch method translation --attn
```



The final perplexity obtained is = 141.58 after 10 epochs

9. Screenshot of the training session by running:

   `python main.py --train --cond --batch method translation --attn --epochs 500`

```
-----------------------------------------------------------------------------------------
| epoch 136 |    20/   29 batches | lr 0.02 | ms/batch 120.24 | loss  2.04 | ppl     7.68
-----------------------------------------------------------------------------------------
| end of epoch 136 | time:  4.10s | valid loss  3.29 | valid ppl    26.95 | valid sqxent    70.06
-----------------------------------------------------------------------------------------
| epoch 137 |    20/   29 batches | lr 0.00 | ms/batch 120.28 | loss  2.06 | ppl     7.82
-----------------------------------------------------------------------------------------
| end of epoch 137 | time:  4.08s | valid loss  3.29 | valid ppl    26.95 | valid sqxent    70.06
-----------------------------------------------------------------------------------------
| epoch 138 |    20/   29 batches | lr 0.00 | ms/batch 120.05 | loss  2.04 | ppl     7.71
-----------------------------------------------------------------------------------------
| end of epoch 138 | time:  4.07s | valid loss  3.29 | valid ppl    26.95 | valid sqxent    70.06
-----------------------------------------------------------------------------------------
| epoch 139 |    20/   29 batches | lr 0.00 | ms/batch 129.91 | loss  2.03 | ppl     7.61
-----------------------------------------------------------------------------------------
| end of epoch 139 | time:  4.34s | valid loss  3.29 | valid ppl    26.95 | valid sqxent    70.06
-----------------------------------------------------------------------------------------
| epoch 140 |    20/   29 batches | lr 0.00 | ms/batch 123.73 | loss  2.03 | ppl     7.62
-----------------------------------------------------------------------------------------
| end of epoch 140 | time:  4.24s | valid loss  3.29 | valid ppl    26.95 | valid sqxent    70.06
-----------------------------------------------------------------------------------------
=========================================================================================
| End of training | final loss  3.29 | final ppl    26.95 | final sqxent    70.06
=========================================================================================
00:10:05
```

   The learning rate becomes 0 in 137th epoch and the perplexity remains constant after that. The final converged value of perplexity obtained = 26.95 after 137 epochs.