Part 1

1. MyGRUCell implementation

```
class MyGRUCell(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(MyGRUCell, self).__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size

        # ------------
        # FILL THIS IN
        # ------------
        ## Input linear layers
        self.Wiz = nn.Linear(input_size, hidden_size)
        self.Wir = nn.Linear(input_size, hidden_size)
        self.Wih = nn.Linear(input_size, hidden_size)

        ## Hidden linear layers
        self.Whz = nn.Linear(hidden_size, hidden_size)
        self.Whr = nn.Linear(hidden_size, hidden_size)
        self.Whh = nn.Linear(hidden_size, hidden_size)


    def forward(self, x, h_prev):
        """Forward pass of the GRU computation for one time step.

        Arguments
            x: batch_size x input_size
            h_prev: batch_size x hidden_size

        Returns:
            h_new: batch_size x hidden_size
        """

        # ------------
        # FILL THIS IN
        # ------------
        z = torch.sigmoid(self.Wiz(x) + self.Whz(h_prev))
        r = torch.sigmoid(self.Wir(x) + self.Whr(h_prev))
        g = torch.tanh(self.Wih(x) * self.Whh(h_prev))
        h_new = (1-z) * g + z * h_prev
        return h_new
```

2. When I tried to translate words starting with t, it does not perform well:

```
tea team tight -> eatay eaway ightedway
```

Also, when I tried to translate words ending with ‘ing’, the model also does not perform well:

```
        shopping fighting running -> osingpay ightfay-indedway
uningway
```

Part 2

1. The equations are as follow:

$$\tilde{\alpha}_i^{(t)} = f(Q_t, K_i) = W_2(ReLU(W_1[Q_t\ K_i] + b1)) + b2$$
$$\alpha_i^{(t)} = softmax(\tilde{\alpha}_i^{(t)})$$
$$c_t = \sum_{i=1}^{T} \alpha_i^{(t)} K_i$$

2. My RNNAttentionDecoder implementation

```
def forward(self, inputs, annotations, hidden_init):
    """Forward pass of the attention-based decoder RNN.

    Arguments:
        inputs: Input token indexes across a batch for all the time step. (batch_size x decoder_seq_len)
        annotations: The encoder hidden states for each step of the input.
                     sequence. (batch_size x seq_len x hidden_size)
        hidden_init: The final hidden states from the encoder, across a batch. (batch_size x hidden_size)

    Returns:
        output: Un-normalized scores for each token in the vocabulary, across a batch for all the decoding time steps. (batch
        attentions: The stacked attention weights applied to the encoder annotations (batch_size x encoder_seq_len x decoder_
    """

    batch_size, seq_len = inputs.size()
    embed = self.embedding(inputs)  # batch_size x seq_len x hidden_size

    hiddens = []
    attentions = []
    h_prev = hidden_init
    for i in range(seq_len):
        # ------------
        # FILL THIS IN - START
        # ------------
        embed_current = embed[:,i,:]  # Get the current time step, across the whole batch (batch size x hidden size)
        context, attention_weights = self.attention(embed_current, annotations, annotations)  # batch_size x 1 x hidden_size
        embed_and_context = torch.cat([embed_current, torch.squeeze(context, dim=1)], dim=1)  # batch_size x (2*hidden_size)
        h_prev = self.rnn(embed_and_context, h_prev)  # batch_size x hidden_size
        # ------------
        # FILL THIS IN - END
        # ------------

        hiddens.append(h_prev)
        attentions.append(attention_weights)

    hiddens = torch.stack(hiddens, dim=1) # batch_size x seq_len x hidden_size
    attentions = torch.cat(attentions, dim=2) # batch_size x seq_len x seq_len

    output = self.out(hiddens)  # batch_size x seq_len x vocab_size
    return output, attentions
```

Part 3

1. ScaledDotProduct implementation

```python
class ScaledDotAttention(nn.Module):
    def __init__(self, hidden_size):
        super(ScaledDotAttention, self).__init__()

        self.hidden_size = hidden_size

        self.Q = nn.Linear(hidden_size, hidden_size)
        self.K = nn.Linear(hidden_size, hidden_size)
        self.V = nn.Linear(hidden_size, hidden_size)
        self.softmax = nn.Softmax(dim=1)
        self.scaling_factor = torch.rsqrt(torch.tensor(self.hidden_size, dtype= torch.float))

    def forward(self, queries, keys, values):
        """The forward pass of the scaled dot attention mechanism.

        Arguments:
            queries: The current decoder hidden state, 2D or 3D tensor. (batch_size x (k) x hidden_size)
            keys: The encoder hidden states for each step of the input sequence. (batch_size x seq_len x hidden_size)
            values: The encoder hidden states for each step of the input sequence. (batch_size x seq_len x hidden_size)

        Returns:
            context: weighted average of the values (batch_size x k x hidden_size)
            attention_weights: Normalized attention weights for each encoder hidden state. (batch_size x seq_len x k)

            The output must be a softmax weighting over the seq_len annotations.
        """

        # ------------
        # FILL THIS IN
        # ------------
        batch_size = keys.size(0)
        q = self.Q(queries)
        k = self.K(keys)
        v = self.V(values)
        unnormalized_attention = (k @ q.transpose(1,2))/ self.scaling_factor
        attention_weights = self.softmax(unnormalized_attention)
        context = attention_weights.transpose(1,2) @ v
        return context, attention_weights
```

CausalScaledDotProduct implementation

```python
class CausalScaledDotAttention(nn.Module):
    def __init__(self, hidden_size):
        super(CausalScaledDotAttention, self).__init__()

        self.hidden_size = hidden_size
        self.neg_inf = torch.tensor(-1e7)

        self.Q = nn.Linear(hidden_size, hidden_size)
        self.K = nn.Linear(hidden_size, hidden_size)
        self.V = nn.Linear(hidden_size, hidden_size)
        self.softmax = nn.Softmax(dim=1)
        self.scaling_factor = torch.rsqrt(torch.tensor(self.hidden_size, dtype= torch.float))

    def forward(self, queries, keys, values):
        """The forward pass of the scaled dot attention mechanism.

        Arguments:
            queries: The current decoder hidden state, 2D or 3D tensor. (batch_size x (k) x hidden_size)
            keys: The encoder hidden states for each step of the input sequence. (batch_size x seq_len x hidden_size)
            values: The encoder hidden states for each step of the input sequence. (batch_size x seq_len x hidden_size)

        Returns:
            context: weighted average of the values (batch_size x k x hidden_size)
            attention_weights: Normalized attention weights for each encoder hidden state. (batch_size x seq_len x k)

            The output must be a softmax weighting over the seq_len annotations.
        """

        # ------------
        # FILL THIS IN
        # ------------
        batch_size = keys.size(0)
        q = self.Q(queries)
        k = self.K(keys)
        v = self.V(values)
        unnormalized_attention = (k @ q.transpose(1,2))/ self.scaling_factor
        mask = torch.tril(unnormalized_attention)
        mask[mask == 0] = self.neg_inf
        attention_weights = self.softmax(mask)
        context = attention_weights.transpose(1,2) @ v
        return context, attention_weights
```

TransformerEncoder's forward implementation

```python
def forward(self, inputs):
    """Forward pass of the encoder RNN.

    Arguments:
        inputs: Input token indexes across a batch for all time steps in the sequence. (batch_size x seq_len)

    Returns:
        annotations: The hidden states computed at each step of the input sequence. (batch_size x seq_len x hidden_size)
        hidden: The final hidden state of the encoder, for each sequence in a batch. (batch_size x hidden_size)
    """

    batch_size, seq_len = inputs.size()
    # ------------
    # FILL THIS IN - START
    # ------------
    encoded = self.embedding(inputs)  # batch_size x seq_len x hidden_size

    # Add positinal embeddings from self.create_positional_encodings. (a'la https://arxiv.org/pdf/1706.03762.pdf, section 3.5)
    encoded += self.positional_encodings[:seq_len]

    annotations = encoded

    for i in range(self.num_layers):
      new_annotations, self_attention_weights = self.self_attentions[i](encoded, annotations, annotations)  # batch_size x seq_len x hidden_size
      residual_annotations = annotations + new_annotations
      new_annotations = self.attention_mlps[i](residual_annotations)
      annotations = residual_annotations + new_annotations
    # ------------
    # FILL THIS IN - END
    # ------------

    # Transformer encoder does not have a last hidden layer.
    return annotations, None
```

TransformerDecoder's forward implementation

```python
def forward(self, inputs, annotations, hidden_init):
    """Forward pass of the attention-based decoder RNN.

    Arguments:
        inputs: Input token indexes across a batch for all the time step. (batch_size x decoder_seq_len)
        annotations: The encoder hidden states for each step of the input.
                     sequence. (batch_size x seq_len x hidden_size)
        hidden_init: Not used in the transformer decoder
    Returns:
        output: Un-normalized scores for each token in the vocabulary, across a batch for all the decoding time steps. (batch_size x decoder_seq_len x vocab_size)
        attentions: The stacked attention weights applied to the encoder annotations (batch_size x encoder_seq_len x decoder_seq_len)
    """

    batch_size, seq_len = inputs.size()
    embed = self.embedding(inputs)  # batch_size x seq_len x hidden_size

    # THIS LINE WAS ADDED AS A CORRECTION.
    embed = embed + self.positional_encodings[:seq_len]

    encoder_attention_weights_list = []
    self_attention_weights_list = []
    contexts = embed
    for i in range(self.num_layers):
      # ------------
      # FILL THIS IN - START
      # ------------
      new_contexts, self_attention_weights = self.self_attentions[i](contexts, annotations, annotations)  # batch_size x seq_len x hidden_size
      residual_contexts = contexts + new_contexts
      new_contexts, encoder_attention_weights = self.encoder_attentions[i](residual_contexts, annotations, annotations) # batch_size x seq_len x hidden_size
      residual_contexts = residual_contexts + new_contexts
      new_contexts = self.attention_mlps[i](residual_contexts)
      contexts = residual_contexts + new_contexts

      # ------------
      # FILL THIS IN - END
      # ------------

      encoder_attention_weights_list.append(encoder_attention_weights)
      self_attention_weights_list.append(self_attention_weights)

    output = self.out(contexts)
    encoder_attention_weights = torch.stack(encoder_attention_weights_list)
    self_attention_weights = torch.stack(self_attention_weights_list)

    return output, (encoder_attention_weights, self_attention_weights)
```

2. Here are my results after 100 epochs

```
Epoch:  52 | Train loss: 1.046 | Val loss: 1.603 | Gen: eeeway away ondinsingchgay isway orringray
Epoch:  53 | Train loss: 1.034 | Val loss: 1.622 | Gen: eeeway away ondgiocingrgway isway odringsway
Epoch:  54 | Train loss: 1.022 | Val loss: 1.622 | Gen: eteway away ondinsingngrway isay oringrgay
Epoch:  55 | Train loss: 1.010 | Val loss: 1.669 | Gen: eeeway away oncincincay-ay isway oringgray
Epoch:  56 | Train loss: 1.015 | Val loss: 1.655 | Gen: eeeway aiday oncincionchgay isay oriningway
Epoch:  57 | Train loss: 0.996 | Val loss: 1.587 | Gen: eteway away odcinsiongrgay isway oringrgway
Epoch:  58 | Train loss: 0.980 | Val loss: 1.602 | Gen: eteway away ondinciongngway isway oringsgay
Epoch:  59 | Train loss: 0.974 | Val loss: 1.589 | Gen: eteway away odcinsiongrgway isway orsingsway
Epoch:  60 | Train loss: 0.972 | Val loss: 1.608 | Gen: eteway awrway ondinsiongngway isway orsingsway
Epoch:  61 | Train loss: 0.997 | Val loss: 1.664 | Gen: eteway aidway onciioiongngway isway oringrgway
Epoch:  62 | Train loss: 1.004 | Val loss: 1.663 | Gen: eteway away odidinay isway orspingway
Epoch:  63 | Train loss: 0.999 | Val loss: 1.599 | Gen: eteway away odndiongngrgway isway orsingray
Epoch:  64 | Train loss: 0.969 | Val loss: 1.590 | Gen: eteway awsway odndinay isway orsingsway
Epoch:  65 | Train loss: 0.947 | Val loss: 1.596 | Gen: eteway away ondinciodgrgway isway orsingsway
Epoch:  66 | Train loss: 0.952 | Val loss: 1.612 | Gen: emeway away oncininchngray isway orsingsway
Epoch:  67 | Train loss: 0.945 | Val loss: 1.563 | Gen: eteway away odndiocodgngay isway odgkingway
Epoch:  68 | Train loss: 0.925 | Val loss: 1.576 | Gen: eheway awrway odndionchngrway isway orsingsway
Epoch:  69 | Train loss: 0.912 | Val loss: 1.592 | Gen: eteway aidway odndiocodgngway isway orsingsway
Epoch:  70 | Train loss: 0.906 | Val loss: 1.566 | Gen: eheway aidway ondciocongngway isway orsingsway
Epoch:  71 | Train loss: 0.891 | Val loss: 1.583 | Gen: eheway awrway odndiocingngway isway orsingsway
Epoch:  72 | Train loss: 0.886 | Val loss: 1.592 | Gen: eheway awrway odndiocingngway isway opringsway
Epoch:  73 | Train loss: 0.879 | Val loss: 1.562 | Gen: eheway awrway ondingiongngway isway orringsway
Epoch:  74 | Train loss: 0.874 | Val loss: 1.585 | Gen: eheway awrway ondincingctrway isway orsingsway
Epoch:  75 | Train loss: 0.878 | Val loss: 1.613 | Gen: eteway irway odcincinctoday isway orpingsway
Epoch:  76 | Train loss: 0.935 | Val loss: 1.590 | Gen: eheway aidway odndioincingway isway orkingsway
Epoch:  77 | Train loss: 0.891 | Val loss: 1.610 | Gen: eheway irrway ondincingctrway isway orsingsway
Epoch:  78 | Train loss: 0.886 | Val loss: 1.624 | Gen: eheway irsway ondcinangctrway isway orsingsway
Epoch:  79 | Train loss: 0.881 | Val loss: 1.603 | Gen: eheway awhway ondincinctrgway isway orringsway
Epoch:  80 | Train loss: 0.862 | Val loss: 1.608 | Gen: eheway irrway ondincingcrgway isway orsingsway
Epoch:  81 | Train loss: 0.850 | Val loss: 1.597 | Gen: eheway irrway ondingingctray isway opringsway
Epoch:  82 | Train loss: 0.846 | Val loss: 1.613 | Gen: eheway aisway odcingoctingay isay opringsway
Epoch:  83 | Train loss: 0.868 | Val loss: 1.600 | Gen: eteway aidway odndiocongngway isway orsingsway
Epoch:  84 | Train loss: 0.852 | Val loss: 1.607 | Gen: eheway irhway ondingongctrway isway orpingsway
Epoch:  85 | Train loss: 0.839 | Val loss: 1.560 | Gen: eheway irrway oddingonctdgway isway orringsway
Epoch:  86 | Train loss: 0.825 | Val loss: 1.577 | Gen: eheway awhay ondingongctray isway orpingsway
Epoch:  87 | Train loss: 0.815 | Val loss: 1.583 | Gen: eheway awhway ondingodgctray isway orpingsway
Epoch:  88 | Train loss: 0.807 | Val loss: 1.582 | Gen: eheway awhway ondiniongctrway isway orpingsway
Epoch:  89 | Train loss: 0.801 | Val loss: 1.581 | Gen: eheway irhway ondinciongngway isway orpingsway
Epoch:  90 | Train loss: 0.803 | Val loss: 1.580 | Gen: eheway irhway ondioiodgcrgway isway orpingsway
Epoch:  91 | Train loss: 0.800 | Val loss: 1.566 | Gen: eheway awhay ondiiocingrgway isway orpingray
Epoch:  92 | Train loss: 0.795 | Val loss: 1.587 | Gen: eheway irrway odndioiodgrgway isway orpingray
Epoch:  93 | Train loss: 0.785 | Val loss: 1.597 | Gen: eheway aidway ondingingctrway isway orpingsway
Epoch:  94 | Train loss: 0.775 | Val loss: 1.596 | Gen: eheway awrway odndiongcingway isway orringray
Epoch:  95 | Train loss: 0.774 | Val loss: 1.589 | Gen: eheway awrway ondinciongrgway isway orringsway
Epoch:  96 | Train loss: 0.780 | Val loss: 1.606 | Gen: eheway awrway ondinciongrgway isway orkingray
Epoch:  97 | Train loss: 0.785 | Val loss: 1.619 | Gen: eheway awhway ondioingcingway isway orpingsway
Epoch:  98 | Train loss: 0.783 | Val loss: 1.615 | Gen: eheway awrway ondincingingway isway orpingsway
Epoch:  99 | Train loss: 0.775 | Val loss: 1.636 | Gen: eheway awrway ondinciongrgway isway orrkingway
source:         the air conditioning is working
translated:     eheway awrway ondinciongrgway isway orrkingway
```

   The model does not perform as good as I have expected, although the losses decreased a lot from the start of training, the translation is not as accurate as the additive attention model. It is also worth noting that the model reached convergence since epoch 30, so I think learning rate needs to decrease. This is also reflected by the translation on the right, which did not change that much since epoch 30.

3. After changing the init method of the transformer decoder to use only non-casual attention, I got the following results:

```
Epoch:  52 | Train loss: 1.462 | Val loss: 1.674 | Gen: eway away ongningingcay ilililay oringay
Epoch:  53 | Train loss: 1.459 | Val loss: 1.666 | Gen: eway away ongngngingngcay isway oringay
Epoch:  54 | Train loss: 1.446 | Val loss: 1.656 | Gen: eway away ongcongway isway oringay
Epoch:  55 | Train loss: 1.431 | Val loss: 1.652 | Gen: eway away ongningway isililay oringingrway
Epoch:  56 | Train loss: 1.423 | Val loss: 1.634 | Gen: eway aray ongngcay isililay ongggay
Epoch:  57 | Train loss: 1.410 | Val loss: 1.638 | Gen: ethay away ongngcay isway oringay
Epoch:  58 | Train loss: 1.402 | Val loss: 1.633 | Gen: eway away ongngngingcay isway oringingay
Epoch:  59 | Train loss: 1.398 | Val loss: 1.636 | Gen: ethay away ongngcay isililay oinggay
Epoch:  60 | Train loss: 1.391 | Val loss: 1.626 | Gen: eway away ongngngway isililay onggingrway
Epoch:  61 | Train loss: 1.386 | Val loss: 1.621 | Gen: ethay awarway ongngngcay isililay ongginway
Epoch:  62 | Train loss: 1.380 | Val loss: 1.639 | Gen: eway away ongngongtingtway isiligilway oringinway
Epoch:  63 | Train loss: 1.388 | Val loss: 1.623 | Gen: ethay ay ongcay isay oringay
Epoch:  64 | Train loss: 1.371 | Val loss: 1.604 | Gen: eway arway ongngcay isay oringay
Epoch:  65 | Train loss: 1.359 | Val loss: 1.594 | Gen: ethay aray ongngngngngcay isay oinggay
Epoch:  66 | Train loss: 1.353 | Val loss: 1.600 | Gen: eway away ongngngngngngcongcay isay oringingrway
Epoch:  67 | Train loss: 1.345 | Val loss: 1.600 | Gen: ethay ararway ongngngngngcay issilay oringingway
Epoch:  68 | Train loss: 1.341 | Val loss: 1.581 | Gen: ethay iray ongngongngngcay isway oringingway
Epoch:  69 | Train loss: 1.324 | Val loss: 1.575 | Gen: ethay away ongngngngtingcongcin issay oringingway
Epoch:  70 | Train loss: 1.328 | Val loss: 1.589 | Gen: ethay araray ongcongngngngway isway oinggingway
Epoch:  71 | Train loss: 1.323 | Val loss: 1.583 | Gen: ethay awaray ongnglay isway oringingway
Epoch:  72 | Train loss: 1.323 | Val loss: 1.568 | Gen: ethay away ongngngway isway oringingway
Epoch:  73 | Train loss: 1.307 | Val loss: 1.584 | Gen: ethay arway ongngcongngngway iway oringingway
Epoch:  74 | Train loss: 1.304 | Val loss: 1.569 | Gen: ethay arway ongngingngngway issay oringinway
Epoch:  75 | Train loss: 1.290 | Val loss: 1.578 | Gen: ethay aray ongngcongngngway isway oringingr
Epoch:  76 | Train loss: 1.289 | Val loss: 1.582 | Gen: ethay aray ongngongngngway iway oringingray
Epoch:  77 | Train loss: 1.286 | Val loss: 1.561 | Gen: ewhay away ongngionglay isway oringingray
Epoch:  78 | Train loss: 1.283 | Val loss: 1.572 | Gen: ethay aray ongongongticay isway oringingray
Epoch:  79 | Train loss: 1.291 | Val loss: 1.607 | Gen: ethay aray ongongongtingtway isway ongggray
Epoch:  80 | Train loss: 1.289 | Val loss: 1.551 | Gen: ethay aray ongonongtingtway isay oringingray
Epoch:  81 | Train loss: 1.280 | Val loss: 1.573 | Gen: eway aray ongcongtingcay isway oringway
Epoch:  82 | Train loss: 1.280 | Val loss: 1.564 | Gen: ethay aray ongngongnglgtay isway oinggingway
Epoch:  83 | Train loss: 1.278 | Val loss: 1.587 | Gen: ethay arway ongongongcay issay oringray
Epoch:  84 | Train loss: 1.276 | Val loss: 1.560 | Gen: eway arway ongngngnghay isway oinggrway
Epoch:  85 | Train loss: 1.256 | Val loss: 1.563 | Gen: ethay arway ongngngnghay isway oringingway
Epoch:  86 | Train loss: 1.246 | Val loss: 1.545 | Gen: ethay arrray ongngngnglay isway oringingway
Epoch:  87 | Train loss: 1.242 | Val loss: 1.557 | Gen: ethay aray ongngngngcay isway oringggggggr
Epoch:  88 | Train loss: 1.244 | Val loss: 1.558 | Gen: ethay aray ongngongctingcongtin isway oingggingay
Epoch:  89 | Train loss: 1.225 | Val loss: 1.560 | Gen: ethay aray ongngngnglay isway oringgway
Epoch:  90 | Train loss: 1.219 | Val loss: 1.550 | Gen: ethay aray ongngongway isway oingggway
Epoch:  91 | Train loss: 1.226 | Val loss: 1.554 | Gen: ethay arway ongnitingticay issay onggggr
Epoch:  92 | Train loss: 1.236 | Val loss: 1.542 | Gen: ethay arway ongngngngngway isway oringingway
Epoch:  93 | Train loss: 1.214 | Val loss: 1.539 | Gen: ethay aray ongongongticay isway oringing
Epoch:  94 | Train loss: 1.211 | Val loss: 1.554 | Gen: ethay aray ongngongtiongway isway oringgway
Epoch:  95 | Train loss: 1.206 | Val loss: 1.545 | Gen: ethay arway ongngngway issay oringr
Epoch:  96 | Train loss: 1.203 | Val loss: 1.545 | Gen: ethay arway ongngngway isway oingr
Epoch:  97 | Train loss: 1.192 | Val loss: 1.531 | Gen: ethay arway ongngngway issay oingr
Epoch:  98 | Train loss: 1.187 | Val loss: 1.543 | Gen: ethay arway ongngingway isway oringing
Epoch:  99 | Train loss: 1.186 | Val loss: 1.535 | Gen: ethay arway ongngngway isway oringing
source:         the air conditioning is working
translated:     ethay arway ongngngway isway oringing
```

Although the validation loss is slightly lower, the training loss is much higher. Also, we can see that the translation is not as good, at least in the previous one we can see traces of original words. As expected, the model reached convergence since epoch 30, and the translation is pretty much the same since epoch 30.

Part 4

1. First, I choose some samples to calculate double digit numbers that are close to each other, i.e. twelve and fourteen, these were both correct.
   Second, I choose some larger number, i.e. one and hundred, they are both correct.
   Third, I choose to use actual numbers, i.e. 1 and 100 to see how the model will handle this kind of input, which it was output correctly.

Fourth, I choose a mix of numbers and words, i.e. 1 and two, which "1 minus two" returned positive, a first failed case.
Fifth, I choose 2 arithmetic operations, i.e. plus and minus, which also gave correct outputs.
Lastly, I decided to use a fake number, i.e. "lala" and see how the model would perceive it. To my surprise, it handles it pretty well. So "one minus lala" gives negative, "one plus lala" gives positive, which are some very reasonable prediction.
The following is the screenshot of my test cases:

```
[29] what_is("twelve minus fourteen")
     negative

[30] what_is("twelve plus fourteen")
     positive

[31] what_is("one plus hundred")
     positive

[32] what_is("one minus hundred")
     negative

[36] what_is("hundred minus one")
     positive

[37] what_is("1 plus 100")
     positive

[38] what_is("1 minus 14")
     negative

[39] what_is("1 minus two")
     positive

[40] what_is("two minus 3")
     negative

[43] what_is("three minus two minus eight")
     negative
```

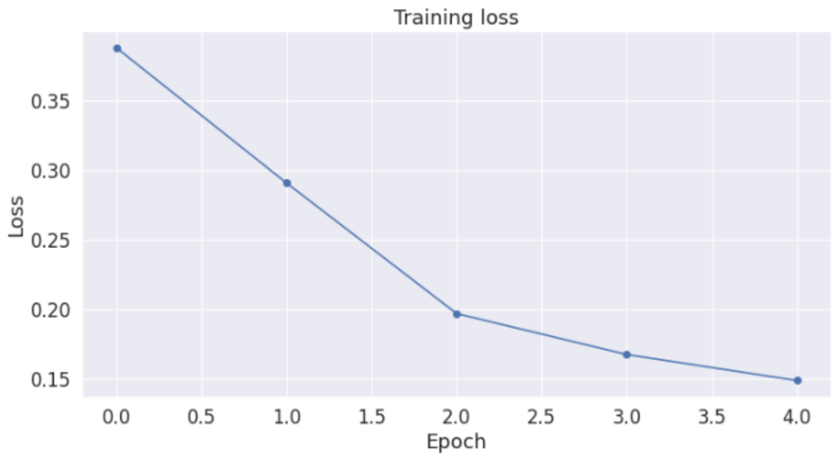
```
[44] what_is("three minus two plus eight")
     positive

[45] what_is("one minus lala")
     negative

[46] what_is("one plus lala")
     positive

[47] what_is("one minus lala plus lala")
     positive

[49] what_is("ten minus lala")
     positive
```

2. For this part, I chose to change some hyperparameters, so I modified the train method to accept variables for learning rate and epochs.
   I tried using 5 epochs and 3e-5 as my learning rate, the loss is as follow:

```
finttune_bert_loss_vals = train_model(model_finetune_bert, 3e-5, 5)
```

```
======== Epoch 1 / 5 ========
Training...

  Average training loss: 0.39
  Training epcoh took: 0:01:12
Running Validation...
  Accuracy: 0.94
  Validation took: 0:00:01

======== Epoch 2 / 5 ========
Training...

  Average training loss: 0.29
  Training epcoh took: 0:01:12
Running Validation...
  Accuracy: 0.97
  Validation took: 0:00:01

======== Epoch 3 / 5 ========
Training...

  Average training loss: 0.20
  Training epcoh took: 0:01:12
Running Validation...
  Accuracy: 0.98
  Validation took: 0:00:01

======== Epoch 4 / 5 ========
Training...

  Average training loss: 0.17
  Training epcoh took: 0:01:12
Running Validation...
  Accuracy: 0.98
  Validation took: 0:00:01

======== Epoch 5 / 5 ========
Training...

  Average training loss: 0.15
  Training epcoh took: 0:01:13
Running Validation...
  Accuracy: 0.98
  Validation took: 0:00:01

Training complete!
```

```
plot_loss_vals(finttune_bert_loss_vals)
```



As we can see, the loss got a bit even lower than previous model, therefore I expect that this model will perform slightly better than before.