Evaluating the Performance of Different Compatibility Functions in Attention Models

Joshua Watt

Contents

In this document, we will demonstrate some of the ideas from the paper on Attention in Natural Language Processing by Andrea Galassi, Marco Lippi and Paolo Torroni. The hyperlink for the paper is provided here: https://arxiv.org/ftp/arxiv/papers/1902/1902.02181.pdf. The beer review data set will be used for this demonstration - find the data set here: https://github.com/YujiaBao/R2A. This data set is split based upon whether the context of the review is surrounding the beer look, aroma or palate. Further, each review is either classified as being a positive (encoded as a 1) or negative (encoded as a 0) review surrounding the particular context. For this report, we will be considering only the reviews surrounding the beers look, however the same models can be applied to reviews surrounding the beers aroma or palate.

Loading Data

First we load in the data and set aside some samples for visualization of attention in practice. In addition, we choose how many training and validation samples we're going to consider. Since the latter part of this demonstration contains significant computation, we limit the training set to 10000 reviews and the validation set to 2000 reviews.

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   task = col_character(),
##   label = col_double(),
##   text = col_character()
## )
##
##
## -- Column specification --------------------------------------------------------
## cols(
##   task = col_character(),
##   label = col_double(),
##   text = col_character()
## )
```

EDA

Now that we have loaded the data in, we can perform an Explanatory Data Analysis (EDA) of each of the sentences. This is demonstrated below.

```
# Obtaining individual words
beer_words <- beer %>%
  unnest_tokens(word, text)
beer_words
```

```
## # A tibble: 1,408,571 x 3
##    task  label word
##    <chr> <dbl> <chr>
##  1 beer0     0 this
##  2 beer0     0 beer
##  3 beer0     0 poured
##  4 beer0     0 a
##  5 beer0     0 very
##  6 beer0     0 slightly
##  7 beer0     0 hazy
##  8 beer0     0 yellow
##  9 beer0     0 to
## 10 beer0     0 faint
## # ... with 1,408,561 more rows
```

```
# Popular words
beer_words %>%
  count(word, sort=TRUE)
```

```
## # A tibble: 25,613 x 2
##    word      n
##    <chr> <int>
##  1 a     64075
```

```
##  2 the   55173
##  3 and   38616
##  4 of    32171
##  5 is    28886
##  6 i     26418
##  7 it    25718
##  8 with  22197
##  9 to    21657
## 10 this  21386
## # ... with 25,603 more rows
```

```
# Popular words with stopwords removed
beer_words <- anti_join(beer_words,get_stopwords())
```

```
## Joining, by = "word"
```

```
beer_words %>%
  count(word, sort=TRUE)
```

```
## # A tibble: 25,485 x 2
##    word      n
##    <chr> <int>
##  1 beer  16360
##  2 head  10936
##  3 s     10518
##  4 taste  9132
##  5 like   8970
##  6 light  7714
##  7 n't    6856
##  8 good   5921
##  9 sweet  5581
## 10 one    5414
## # ... with 25,475 more rows
```

Preprocessing

Now we must preprocess the sentences so they can be used in our attention models. As part of this, we must remove all special characters from the sentences, insert spaces between words and create mappings between numbers and different words.

Formatting Sentences

First we remove all special characters from the sentences, add spaces before punctuation and insert start and end tokens for each sentence.

```
str_break = function(x, width = 80L) {
  n = nchar(x)
  if (n <= width) return(x)
  n1 = seq(1L, n, by = width)
  n2 = seq(width, n, by = width)
  if (n %% width != 0) n2 = c(n2, n)
  substring(x, n1, n2)
}

# Function to add a space before punctuation
space_before_punct <- function(sentence) {
```

```
  str_replace_all(sentence, "([?.!])", " \\1")
}

# Replacing all special characters
replace_special_chars <- function(sentence) {
  str_replace_all(sentence, c("[^a-zA-Z?.!,¿]+|\'|\\.|\\,|\\!|\\?"), ' ')
}

# Adding index to the start and end of a word
add_startend <- function(sentence) {
  paste0("<start> ", sentence, " <stop>")
}
add_tokens <- Vectorize(add_startend, USE.NAMES = FALSE)

# Composing functions
preprocess_sentence <- compose(add_startend,
                               str_squish,
                               replace_special_chars,
                               space_before_punct)

# Applying functions and displaying preprocessed sentence
word_pairs <- map(beer$text, preprocess_sentence)
strwrap(word_pairs[1], 80)
```

```
##  [1] "<start> this beer poured a very slightly hazy yellow to faint gold with not"
##  [2] "much head despite a fairly aggressive pour not very impressive to the eye"
##  [3] "really i ve seen much nicer wits smell well pretty faint really there s some"
##  [4] "orange peel maybe but the spice is hard to find very faint maybe the beer is"
##  [5] "too cold first taste i m not anticipating much by now was actually not too bad"
##  [6] "tastes witty but less so a bit of sweet a touch of citrus swish swish warm it"
##  [7] "up a bit there s a little more really it tastes fine the mouthfeel is"
##  [8] "reasonable and it s quite drinkable it s also quite forgettable i did n t find"
##  [9] "anything special here but it was n t an unpleasant experience it s not like it"
## [10] "s a bad beer or a great beer it s fine it s beer and during a hotter summer"
## [11] "outside on a hot day i m sure i would have found it refreshing <stop>"
```

Creating lookup indices

We then create lookup indices for every word in the data set. This enables us to represent each word with a number and is necessary to perform our attention models on the data.

```
# Padding words which are shorter than others
create_index <- function(sentences) {
  unique_words <- sentences %>% unlist() %>% paste(collapse = " ") %>%
    str_split(pattern = " ") %>% .[[1]] %>% unique() %>% sort()
  index <- data.frame(
    word = unique_words,
    index = 1:length(unique_words),
    stringsAsFactors = FALSE
  ) %>%
    add_row(word = "<pad>",
                    index = 0,
                    .before = 1)
  index
```

```
}

# Function which maps words to a number
word2index <- function(word, index_df) {
  index_df[index_df$word == word, "index"]
}

# Function which maps numbers to a word
index2word <- function(index, index_df) {
  index_df[index_df$index == index, "word"]
}

# Creating the mapping between words and numbers
beer_index <- create_index(map(word_pairs, ~ .[[1]]))
beer_index[1:30,]
```

```
##              word index
## 1           <pad>     0
## 2         <start>     1
## 3          <stop>     2
## 4               a     3
## 5              aa     4
## 6          aaaand     5
## 7          aaahhh     6
## 8             aah     7
## 9             aal     8
## 10           aals     9
## 11           aare    10
## 12         aarhus    11
## 13          aaron    12
## 14             ab    13
## 15            aba    14
## 16          aback    15
## 17         abacus    16
## 18           abad    17
## 19        abandon    18
## 20      abandoned    19
## 21 abarwithnoname    20
## 22        abasive    21
## 23         abates    22
## 24         abbaye    23
## 25        abbayes    24
## 26          abbey    25
## 27         abbeys    26
## 28          abbot    27
## 29     abbotsford    28
## 30           abby    29
```

Converting sentences to matrices

We then convert the sentences to matrices of numbers representing each individual word in the sentence. Each sentence is a row in the matrix and the numbers constitute the order of words appearing in the sentence.

```
# Function mapping sentences to words
sentence2digits <- function(sentence, index_df) {
  map((sentence %>% str_split(pattern = " "))[[1]], function(word)
    word2index(word, index_df))
}

# Function creating matrices of numbers from sentences
sentlist2diglist <- function(sentence_list, index_df) {
  map(sentence_list, function(sentence)
    sentence2digits(sentence, index_df))
}

# Creating matrices of numbers from sentences
beer_diglist <-
  sentlist2diglist(map(word_pairs, ~ .[[1]]), beer_index)
beer_maxlen <- map(beer_diglist, length) %>% unlist() %>% max()
beer_matrix <-
  pad_sequences(beer_diglist, maxlen = beer_maxlen,  padding = "post")
```

Performing train-validation split

We then perform the train-validation split.

```
# Training data
x_train <- beer_matrix[1:train_size,]
y_train <- beer$label[1:train_size]

# Validation data
x_val <- beer_matrix[(train_size+1):(train_size + val_size),]
y_val <- beer$label[(train_size+1):(train_size + val_size)]

buffer_size <- nrow(x_train)
```

Creating batches from training data set

We will use a batch size of 20 for all models. We shuffle the training data set and create batches from it.

```
# The batch size
batch_size <- 20

# Shuffling data and creating batches
train_dataset <-
  tensor_slices_dataset(keras_array(list(x_train, y_train)))  %>%
  dataset_shuffle(buffer_size = buffer_size) %>%
  dataset_batch(batch_size, drop_remainder = TRUE)
```

Additive Attention

In this section, we will use a sequence model to classify each of the reviews as being a positive review or a negative review surrounding the look of the beer. To do this, we will adapt the general attention model discussed in Galassi's paper on Attention in Natural Language Processing. The encoder will consist of an embedding layer followed by a bidirectional RNN with 200 gated recurrent units - this will feed as an input into the attention model. The decoder will consist of two cascading elements: the attention model and a DNN. The DNN will contain one fully connected hidden layer with 50 hidden units and Relu activation and a

fully connected output layer with one hidden unit and sigmoid activation. The attention model will use an additive compatibility function with tanh activation for the output and hidden states from the encoder. Lets explain this in a little more detail below.

Let f denote the compatibility function for our attention model. Moreover, let K denote the output from the encoder and q denote a concatenated version of the forward and backward hidden states from the bidirectional RNN in the encoder. The additive compatibility function we use is then of the form:

$$f(q, K) = w_{\text{imp}}^T \tanh(W_1 K + W_2 q + b)$$

where $W_1, W_2, w_{\text{imp}}, b$ are all parameters which require optimizing.

Attention Encoder

We now create a function for the attention encoder described above.

```
# Defining function for the encoder
attention_encoder <-

  function(gru_units,
           embedding_dim,
           beer_vocab_size,
           name = NULL) {

    # We use a custom keras model
    keras_model_custom(name = name, function(self) {

      # Embedding layer
      self$embedding <-
        layer_embedding(
          input_dim = beer_vocab_size,
          output_dim = embedding_dim
        )

      # Bidirectional GRU layer
      self$gru <-
          bidirectional(layer = layer_gru(units = gru_units,
                    return_sequences = TRUE,
                    return_state = TRUE))

      function(inputs, mask = NULL) {

        # defining inputs
        x <- inputs[[1]]
        hidden <- inputs[[2]]

        # Performing embedding followed by the bidirectional GRU layer
        x <- self$embedding(x)
        c(output, state_forward, state_backward) %<-%
          self$gru(x, initial_state = c(hidden, hidden))

        # Returning results
        list(output, state_forward, state_backward)
      }
    })
  }
```

Attention Decoder

We now create a function for the attention decoder described above.

```
# Defining function for the decoder
attention_decoder <-
  function(object,
           dense_units,
           embedding_dim,
           name = NULL) {

    # We use a custom keras model
    keras_model_custom(name = name, function(self) {

      # First we have a dense layer with relu activation
      self$dense <-
        layer_dense(
          units = 50,
          activation = "relu"
        )

      # Followed by another dense layer with sigmoid activation
      self$sig <-
        layer_dense(
          units = 1,
          activation = "sigmoid"
        )

      # We then add layers for each of the parameters
      # in the compatibility function
      dense_units <- dense_units
      self$W1 <- layer_dense(units = dense_units)
      self$W2 <- layer_dense(units = dense_units)
      self$V <- layer_dense(units = 1L)

      function(inputs, mask = NULL) {

        # Defining inputs
        hidden <- inputs[[1]]
        encoder_output <- inputs[[2]]

        hidden_with_time_axis <- k_expand_dims(hidden, 2)

        # Calculating compatibility function
        compatibility <- self$V(k_tanh(self$W1(encoder_output) +
                                  self$W2(hidden_with_time_axis)))

        # Calculating attention weights
        attention_weights <- k_softmax(compatibility, axis = 2)

        # Calculating the context vector
        context_vector <- attention_weights * encoder_output
        context_vector <- k_sum(context_vector, axis = 2)
        x <- k_expand_dims(context_vector, 2)
```

```
        # Performing dense layer followed by sigmoid layer
        output %<-%  self$sig(self$dense(x))

        # Returning results
        output <- output %>% k_concatenate() %>% k_concatenate()
        list(output, attention_weights)
      }

    })
  }
```

Creating the model

We first define the hyper-parameters for the model.

```
# Hyper-parameters
batch_size <- 20
embedding_dim <- 50
gru_units <- 200
dense_units <- 50
beer_vocab_size <- nrow(beer_index)
```

We then define the encoder and decoder functions created above.

```
# Encoder
encoder <- attention_encoder(
  gru_units = gru_units,
  embedding_dim = embedding_dim,
  beer_vocab_size = beer_vocab_size
)

# Decoder
decoder <- attention_decoder(
  dense_units = dense_units,
  embedding_dim = embedding_dim
)
```

We use the Adam optimizer to perform the gradient descent. We then define our functions for calculating loss and accuracy. Here we use binary cross entropy to calculate loss since this is a binary classification problem.

```
# Adam optimizer
optimizer <- tf$optimizers$Adam()

# Loss function
cx_loss <- function(y_true, y_pred) {
  loss <-
    k_binary_crossentropy(target = y_true, output = y_pred)
  tf$reduce_mean(loss)
}

# Accuracy function
cx_accuracy <- function(y_true, y_pred) {
    metric_binary_accuracy(y_true, y_pred)
}
```

We create a function which returns the prediction of a sentence from the model.

```
# Function for obtaining the result of any given sentence
get_result <- function(input){

  # Performing forward pass
  input <- k_constant(input)
  hidden <- k_zeros(c(nrow(input), gru_units))
  c(enc_output, enc_hidden_forward, enc_hidden_backward) %<-% encoder(list(input, hidden))
  dec_hidden <- k_concatenate(list(enc_hidden_backward,enc_hidden_forward))
  c(preds, attention_weights) %<-%
    decoder(list(dec_hidden, enc_output))

  # Returning prediction
  return(ifelse(as.double(preds)>0.5,1,0))
}
```

Training the Model

This block of code trains the model. Since this requires significant computation, we will only perform 5 epochs. Moreover, we store the accuracy and loss when performing the model on the training and validation set after every epoch.

```
# Setting number of epochs
n_epochs <- 5

# Setting up storage containers
encoder_init_hidden <- k_zeros(c(batch_size, gru_units))
train_loss_additive <- rep(NA, n_epochs)
train_accuracy_additive <- rep(NA, n_epochs)
val_loss_additive <- rep(NA, n_epochs)
val_accuracy_additive <- rep(NA, n_epochs)
batch_loss <- rep(NA, n_epochs*train_size/batch_size)

# Setting start time
start.time <- Sys.time()

# Looping over epochs
for (epoch in seq_len(n_epochs)) {

  total_loss <- 0
  total_accuracy <- 0
  iteration <- 0

  # Getting next iteration
  iter <- make_iterator_one_shot(train_dataset)

  #Looping over batches
  until_out_of_range({

    # Obtaining next batch
    batch <- iterator_get_next(iter)
    loss <- 0
    accuracy <- 0
    x <- batch[[1]]
    y <- batch[[2]]
```

```
    iteration <- iteration + 1

    # Performing forward and backward pass
    with(tf$GradientTape() %as% tape, {

      # encoding the batch
      c(enc_output, enc_hidden_forward, enc_hidden_backward) %<-%
        encoder(list(x, encoder_init_hidden))

      # Setting hidden decoder state to encoder hidden states
      dec_hidden <- k_concatenate(list(enc_hidden_backward,enc_hidden_forward))

      # decoding the batch
      c(preds, weights) %<-%
          decoder(list(dec_hidden, enc_output))

      # Calculating batch loss and accuracy
      loss <- loss + cx_loss(y, preds)
      accuracy <- cx_accuracy(y, preds)
    })

    # Obtaining batch gradients
    variables <- c(encoder$variables, decoder$variables)
    gradients <- tape$gradient(loss, variables)

    # Performing gradient descent update
    optimizer$apply_gradients(purrr::transpose(list(gradients, variables)))

  })

  # Obtaining predictions for training and validation sets
  y_pred_train <- k_constant(get_result(x_train))
  y_pred_val <- k_constant(get_result(x_val))

  # Updating training and validation loss and accuracy for the epoch
  train_loss_additive[epoch] <-
    (cx_loss(k_constant(y_train), y_pred_train)) %>%
    as.double()
  val_loss_additive[epoch] <-
    (cx_loss(k_constant(y_val), y_pred_val)) %>%
    as.double()
  train_accuracy_additive[epoch] <-
    (cx_accuracy(k_constant(y_train), y_pred_train)) %>%
    as.double()
  val_accuracy_additive[epoch] <-
    (cx_accuracy(k_constant(y_val), y_pred_val)) %>%
    as.double()
}

# Setting end time
end.time <- Sys.time()

# Returning time taken
```

```
time.taken <- end.time - start.time
time.taken
```

```
## Time difference of 4.696729 hours
```

Evaluating the model

The following functions allow us to evaluate the model obtained above.

```
# Function for calculating sentence result and attention vector
evaluate <-
  function(sentence) {
    attention_vector <- rep(0, beer_maxlen)

    # Preprocess sentence
    sentence <- preprocess_sentence(sentence)
    input <- sentence2digits(sentence, beer_index)
    input <- input %>% unlist() %>% list()
    input <-
      pad_sequences(input, maxlen = beer_maxlen,  padding = "post")
    input <- k_constant(input)

    # Perform forward pass
    hidden <- k_zeros(c(1, gru_units))
    c(enc_output, enc_hidden_forward, enc_hidden_backward) %<-% encoder(list(input, hidden))
    dec_hidden <- k_concatenate(list(enc_hidden_backward,enc_hidden_forward))
    c(preds, attention_weights) %<-%
        decoder(list(dec_hidden, enc_output))

    #Calculating the attention vector
    attention_weights <- k_reshape(attention_weights, c(-1))
    attention_vector <- attention_weights %>% as.double()

    # Calculating our prediction
    result <- ifelse(as.double(preds)>0.5,1,0)

    # Returning results
    list(result, sentence, attention_vector)
  }
```

```
# Function for plotting a word cloud of the attention vector results
plot_attention <-
  function(sentence) {

    # Evaluating sentence and determining colour of wordcloud
    c(result, sentence, attention_vector) %<-% evaluate(sentence)
    result <- ifelse(result==1,"green","red")

    # Processing attention vector and scaling scores out of 100
    attention_vector <- attention_vector[1:length(str_split(sentence, " ")[[1]])]
    attention_vector <- attention_vector[-c(1,length(attention_vector))]
    attention_vector <- 100*attention_vector/max(attention_vector)

    # Processing sentence
    sentence <- unlist(strsplit(sentence, " "))
```

```
    sentence <- sentence[-c(1,length(sentence))]
    attention_vector <- round(attention_vector)
    sentence_rep <- rep(sentence, times = attention_vector)

    # Displaying wordcloud
    sentence_rep %>%
      dfm() %>%
      textplot_wordcloud(color = result)
  }
```

```
# Function to translate a sentence and provide prediction
translate <- function(sentence, true) {

  # Evaluating sentence
  c(result, sentence, attention_vector) %<-% evaluate(sentence)

  # Processing sentence result
  sentence <- unlist(strsplit(sentence, " "))
  sentence <- sentence[-c(1,length(sentence))]
  sentence <- paste(sentence, collapse = " ")

  # Printing input, prediction and true result
  result <- ifelse(result==1,"Positive","Negative")
  true <- ifelse(true==1,"Positive","Negative")
  return(list(paste0("Input: ",  sentence),
              paste0("Predicted translation: ", result),
              paste0("True translation: ", true)))
}
```

We display the loss and accuracy for both the training set and validation set after each epoch.

```
dat1 <- data.frame("data" = as.factor(c(rep("Training", n_epochs),
      rep("Validation", n_epochs))), "epoch" = rep(1:n_epochs,2),
      "loss"=c(train_loss_additive, val_loss_additive))
dat2 <- data.frame("data" = as.factor(c(rep("Training", n_epochs),
      rep("Validation", n_epochs))), "epoch" = rep(1:n_epochs,2),
      "accuracy"=c(train_accuracy_additive, val_accuracy_additive))

# Plotting results
p <- ggplot(dat1, aes(x = epoch, y = loss)) +
  geom_point(aes(col=data)) +
  geom_line(aes(col=data)) +
  ggtitle("Training and Validation Sets Loss and Accuracy over Epochs")
q <- ggplot(dat2, aes(x = epoch, y = accuracy)) +
  geom_point(aes(col=data)) +
  geom_line(aes(col=data)) +
  ylim(0,1)

ggarrange(p, q, ncol = 1, nrow = 2, common.legend = TRUE, legend = "right")
```

We predict the result of a negative review and display its word cloud below.

```
c(sentence, prediction, true_result) %<-% translate(negative_sample, 0)
strwrap(sentence,80)
```

```
##  [1] "Input: starting with the usual very dark pour although plenty of light coming"
```

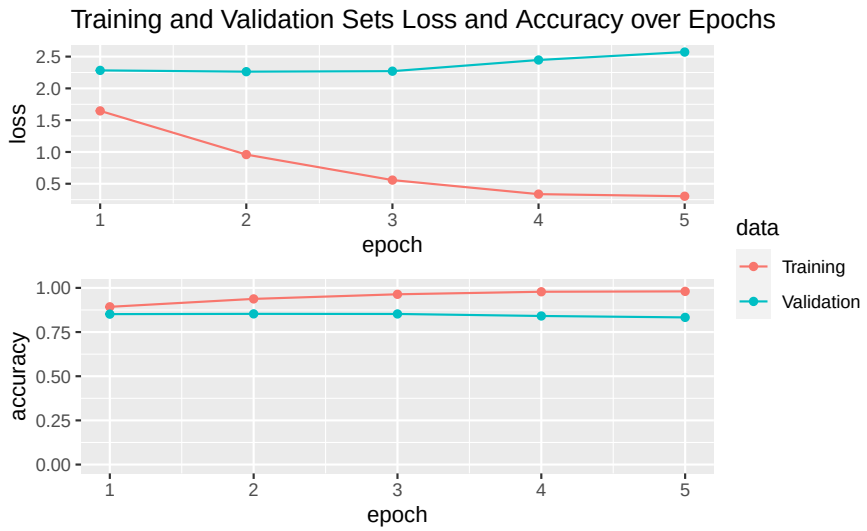

Figure 1: Loss and Accuracy on training and validation sets for additive attention model.

```
##  [2] "through head disappeared so quickly i never got a chance to check the color out"
##  [3] "it totally disappeared in about seconds those two features brought the"
##  [4] "appearance numbers down the caramel smell is quite overwhelming i was expecting"
##  [5] "hints of caramel not a huge dousing of it i might as well have a glass of"
##  [6] "caramel i dont pick up anything else in the smell caramel covers all i dont"
##  [7] "know what to say about the taste except bleh there is very little to this beer"
##  [8] "except for what seems like caramel flavoring added to an average dark lager the"
##  [9] "caramel which is in here tastes overly sugary not like a good toasty caramel"
## [10] "mouthfeel is relatively thin not like any other porter ive had im having issues"
## [11] "drinking any more than is necessary to write this review due to my stomach"
## [12] "beginning to feel strange from all the sweetness shudder i had high hopes for"
## [13] "this i was imaging a creamy full bodied porter with undertones of caramel"
## [14] "coming into presence at the end but there was none of that"
```

```
prediction
```

```
## [1] "Predicted translation: Negative"
```

```
true_result
```

```
## [1] "True translation: Negative"
```

```
plot_attention(negative_sample)
```

We predict the result of a positive review and display its word cloud below.

```
c(sentence, prediction, true_result) %<-% translate(positive_sample, 1)
strwrap(sentence,80)
```

```
## [1] "Input: this is a great standard for an oktoberfest it took a little while for"
## [2] "the aroma to open up but once the beer warmed a bit it was rewarding malty with"
## [3] "a nice sweetness good to the nose pour was standard medium copper with a nice"
## [4] "white head and medium lacing very creamy on the palate smooth and clean"
## [5] "unbelievably pleasant mouthfeel flavor like the aroma takes a short while to"
## [6] "warm up and release all of it s potential slightly aggressive and nicely sweet"
## [7] "there is a good bread quality present tops on the list for marzens that i have"
```

Figure 2: Wordcloud of a negative review.

```
## [8] "encountered so far"
```

```
prediction
```

```
## [1] "Predicted translation: Positive"
```

```
true_result
```

```
## [1] "True translation: Positive"
```

```
plot_attention(positive_sample)
```

Figure 3: Wordcloud of a positive review.

Multiplicative Attention

This attention model will now use a multiplicative compatibility function for the output and hidden states from the encoder. Lets explain this in a little more detail below.

Let f denote the compatibility function for our attention model. Moreover, let K denote the output from the encoder and q denote a concatenated version of the forward and backward hidden states from the bidirectional RNN in the encoder. The multiplicative compatibility function we use is then of the form:

$$f(q, K) = w_{\text{imp}}^T q^T K.$$

where w_{imp} is a trainable parameter.

Attention Decoder

We now create a function for the attention decoder described above.

```
# Defining function for the decoder
attention_decoder <-
  function(object,
           dense_units,
           embedding_dim,
           name = NULL) {

    # We use a custom keras model
    keras_model_custom(name = name, function(self) {

      # First we have a dense layer with relu activation
      self$dense <-
        layer_dense(
          units = 50,
          activation = "relu"
        )

      # Followed by another dense layer with sigmoid activation
      self$sig <-
        layer_dense(
          units = 1,
          activation = "sigmoid"
        )

      # We then add layers for each of the parameters
      # in the compatibility function
      dense_units <- dense_units
      self$V <- layer_dense(units = 1L)

      function(inputs, mask = NULL) {

        # Defining inputs
        hidden <- inputs[[1]]
        encoder_output <- inputs[[2]]

        hidden_with_time_axis <- k_expand_dims(hidden, 2)

        # Calculating compatibility function
        compatibility <- self$V(hidden_with_time_axis*encoder_output)
```

```
        # Calculating attention weights
        attention_weights <- k_softmax(compatibility, axis = 2)

        # Calculating the context vector
        context_vector <- attention_weights * encoder_output
        context_vector <- k_sum(context_vector, axis = 2)
        x <- k_expand_dims(context_vector, 2)

        # Performing dense layer followed by sigmoid layer
        output %<-%  self$sig(self$dense(x))

        # Returning results
        output <- output %>% k_concatenate() %>% k_concatenate()
        list(output, attention_weights)
      }

    })
  }
```

```
# Decoder
decoder <- attention_decoder(
  dense_units = dense_units,
  embedding_dim = embedding_dim
)
```

Training the Model

This block of code trains the model. Since this requires significant computation, we will only perform 5 epochs. Moreover, we store the accuracy and loss when performing the model on the training and validation set after every epoch.

```
# Setting number of epochs
n_epochs <- 5

# Setting up storage containers
encoder_init_hidden <- k_zeros(c(batch_size, gru_units))
train_loss_multiplicative <- rep(NA, n_epochs)
train_accuracy_multiplicative <- rep(NA, n_epochs)
val_loss_multiplicative <- rep(NA, n_epochs)
val_accuracy_multiplicative <- rep(NA, n_epochs)

# Setting start time
start.time <- Sys.time()

# Looping over epochs
for (epoch in seq_len(n_epochs)) {

  total_loss <- 0
  total_accuracy <- 0
  iteration <- 0

  # Getting next iteration
  iter <- make_iterator_one_shot(train_dataset)
```

```
#Looping over batches
until_out_of_range({

  # Obtaining next batch
  batch <- iterator_get_next(iter)
  loss <- 0
  accuracy <- 0
  x <- batch[[1]]
  y <- batch[[2]]
  iteration <- iteration + 1

  # Performing forward and backward pass
  with(tf$GradientTape() %as% tape, {

    # encoding the batch
    c(enc_output, enc_hidden_forward, enc_hidden_backward) %<-%
      encoder(list(x, encoder_init_hidden))

    # Setting hidden decoder state to encoder hidden states
    dec_hidden <- k_concatenate(list(enc_hidden_backward,enc_hidden_forward))

    # decoding the batch
    c(preds, weights) %<-%
        decoder(list(dec_hidden, enc_output))

    # Calculating batch loss and accuracy
    loss <- loss + cx_loss(y, preds)
    accuracy <- cx_accuracy(y, preds)
  })

  # Obtaining batch gradients
  variables <- c(encoder$variables, decoder$variables)
  gradients <- tape$gradient(loss, variables)

  # Performing gradient descent update
  optimizer$apply_gradients(purrr::transpose(list(gradients, variables)))

})

# Obtaining predictions for training and validation sets
y_pred_train <- k_constant(get_result(x_train))
y_pred_val <- k_constant(get_result(x_val))

# Updating training and validation loss and accuracy for the epoch
train_loss_multiplicative[epoch] <-
  (cx_loss(k_constant(y_train), y_pred_train)) %>%
  as.double()
val_loss_multiplicative[epoch] <-
  (cx_loss(k_constant(y_val), y_pred_val)) %>%
  as.double()
train_accuracy_multiplicative[epoch] <-
  (cx_accuracy(k_constant(y_train), y_pred_train)) %>%
  as.double()
```

```
  val_accuracy_multiplicative[epoch] <-
    (cx_accuracy(k_constant(y_val), y_pred_val)) %>%
    as.double()
}

# Setting end time
end.time <- Sys.time()

# Returning time taken
time.taken <- end.time - start.time
time.taken
```

```
## Time difference of 4.821067 hours
```

Evaluating the model

We display the loss and accuracy for both the training set and validation set after each epoch.

```
dat1 <- data.frame("data" = as.factor(c(rep("Training", n_epochs),
      rep("Validation", n_epochs))), "epoch" = rep(1:n_epochs,2),
      "loss"=c(train_loss_multiplicative, val_loss_multiplicative))
dat2 <- data.frame("data" = as.factor(c(rep("Training", n_epochs),
      rep("Validation", n_epochs))), "epoch" = rep(1:n_epochs,2),
      "accuracy"=c(train_accuracy_multiplicative, val_accuracy_multiplicative))

# Plotting results
p <- ggplot(dat1, aes(x = epoch, y = loss)) +
  geom_point(aes(col=data)) +
  geom_line(aes(col=data)) +
  ggtitle("Training and Validation Sets Loss and Accuracy over Epochs")
q <- ggplot(dat2, aes(x = epoch, y = accuracy)) +
  geom_point(aes(col=data)) +
  geom_line(aes(col=data)) +
  ylim(0,1)

ggarrange(p, q, ncol = 1, nrow = 2, common.legend = TRUE, legend = "right")
```

We predict the result of a negative review and display its word cloud below.

```
c(sentence, prediction, true_result) %<-% translate(negative_sample, 0 )
strwrap(sentence,80)
```

```
##  [1] "Input: starting with the usual very dark pour although plenty of light coming"
##  [2] "through head disappeared so quickly i never got a chance to check the color out"
##  [3] "it totally disappeared in about seconds those two features brought the"
##  [4] "appearance numbers down the caramel smell is quite overwhelming i was expecting"
##  [5] "hints of caramel not a huge dousing of it i might as well have a glass of"
##  [6] "caramel i dont pick up anything else in the smell caramel covers all i dont"
##  [7] "know what to say about the taste except bleh there is very little to this beer"
##  [8] "except for what seems like caramel flavoring added to an average dark lager the"
##  [9] "caramel which is in here tastes overly sugary not like a good toasty caramel"
## [10] "mouthfeel is relatively thin not like any other porter ive had im having issues"
## [11] "drinking any more than is necessary to write this review due to my stomach"
## [12] "beginning to feel strange from all the sweetness shudder i had high hopes for"
## [13] "this i was imaging a creamy full bodied porter with undertones of caramel"
```

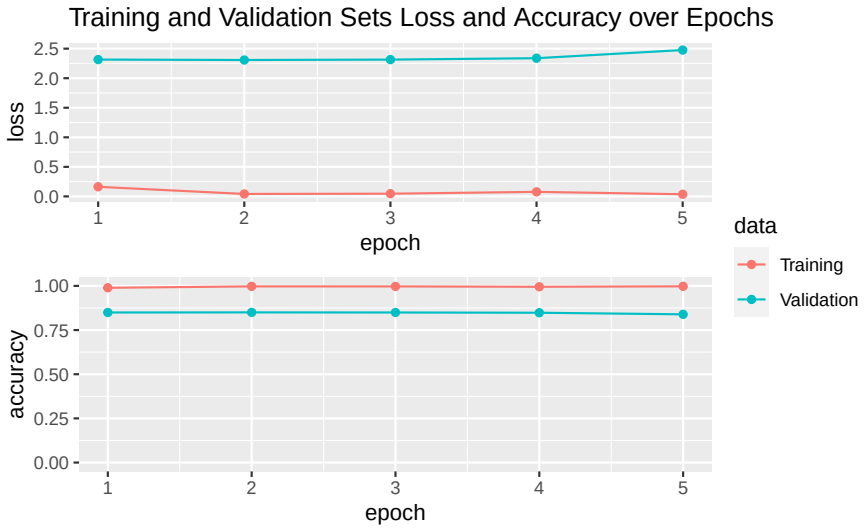

Figure 4: Loss and Accuracy on training and validation sets for multiplicative attention model.

```
## [14] "coming into presence at the end but there was none of that"
```

```
prediction
```

```
## [1] "Predicted translation: Positive"
```

```
true_result
```

```
## [1] "True translation:  Negative"
```

```
plot_attention(negative_sample)
```

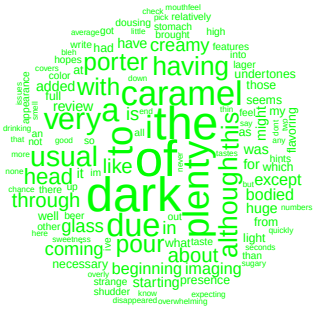

Figure 5: Wordcloud of a negative review.

We predict the result of a positive review and display its word cloud below.

```
c(sentence, prediction, true_result) %<-% translate(positive_sample, 1)
strwrap(sentence,80)
```

```
## [1] "Input: this is a great standard for an oktoberfest it took a little while for"
## [2] "the aroma to open up but once the beer warmed a bit it was rewarding malty with"
## [3] "a nice sweetness good to the nose pour was standard medium copper with a nice"
## [4] "white head and medium lacing very creamy on the palate smooth and clean"
## [5] "unbelievably pleasant mouthfeel flavor like the aroma takes a short while to"
## [6] "warm up and release all of it s potential slightly aggressive and nicely sweet"
## [7] "there is a good bread quality present tops on the list for marzens that i have"
## [8] "encountered so far"
```

```
prediction
```

```
## [1] "Predicted translation: Positive"
```

```
true_result
```

```
## [1] "True translation: Positive"
```

```
plot_attention(positive_sample)
```

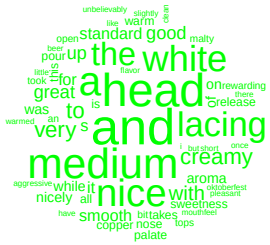

Figure 6: Wordcloud of a positive review.

Activated General Attention

This attention model will now use a activated general compatibility function with tanh activation for the output and hidden states from the encoder. Lets explain this in a little more detail below.

Let f denote the compatibility function for our attention model. Moreover, let K denote the output from the encoder and q denote a concatenated version of the forward and backward hidden states from the bidirectional RNN in the encoder. The activated general compatibility function we use is then of the form:

$$f(q, K) = w_{\text{imp}}^T \tanh(q^T W K + b)$$

where w_{imp}, W, b are trainable parameters.

We firstly define the new attention decoder below.

Attention Decoder

We now create a function for the attention decoder described above.

```
# Defining function for the decoder
attention_decoder <-
  function(object,
           dense_units,
           embedding_dim,
           name = NULL) {

    # We use a custom keras model
    keras_model_custom(name = name, function(self) {

      # First we have a dense layer with relu activation
      self$dense <-
        layer_dense(
          units = 50,
          activation = "relu"
        )

      # Followed by another dense layer with sigmoid activation
      self$sig <-
        layer_dense(
          units = 1,
          activation = "sigmoid"
        )

      # We then add layers for each of the parameters
      # in the compatibility function
      dense_units <- dense_units
      self$V <- layer_dense(units = 1L)
      self$W <- layer_dense(units = dense_units)

      function(inputs, mask = NULL) {

        # Defining inputs
        hidden <- inputs[[1]]
        encoder_output <- inputs[[2]]

        hidden_with_time_axis <- k_expand_dims(hidden, 2)

        # Calculating compatibility function
        compatibility <- self$V(k_tanh(self$W(hidden_with_time_axis*encoder_output)))

        # Calculating attention weights
        attention_weights <- k_softmax(compatibility, axis = 2)

        # Calculating the context vector
        context_vector <- attention_weights * encoder_output
        context_vector <- k_sum(context_vector, axis = 2)
        x <- k_expand_dims(context_vector, 2)

        # Performing dense layer followed by sigmoid layer
        output %<-%  self$sig(self$dense(x))

        # Returning results
```

```
        output <- output %>% k_concatenate() %>% k_concatenate()
        list(output, attention_weights)
      }

    })
  }
```

```
# Decoder
decoder <- attention_decoder(
  dense_units = dense_units,
  embedding_dim = embedding_dim
)
```

Training the Model

This block of code trains the model. Since this requires significant computation, we will only perform 5 epochs. Moreover, we store the accuracy and loss when performing the model on the training and validation set after every epoch.

```
# Setting number of epochs
n_epochs <- 5

# Setting up storage containers
encoder_init_hidden <- k_zeros(c(batch_size, gru_units))
train_loss_act <- rep(NA, n_epochs)
train_accuracy_act <- rep(NA, n_epochs)
val_loss_act <- rep(NA, n_epochs)
val_accuracy_act <- rep(NA, n_epochs)

# Setting start time
start.time <- Sys.time()

# Looping over epochs
for (epoch in seq_len(n_epochs)) {

  total_loss <- 0
  total_accuracy <- 0
  iteration <- 0

  # Getting next iteration
  iter <- make_iterator_one_shot(train_dataset)

  #Looping over batches
  until_out_of_range({

    # Obtaining next batch
    batch <- iterator_get_next(iter)
    loss <- 0
    accuracy <- 0
    x <- batch[[1]]
    y <- batch[[2]]
    iteration <- iteration + 1

    # Performing forward and backward pass
```

```
    with(tf$GradientTape() %as% tape, {

      # encoding the batch
      c(enc_output, enc_hidden_forward, enc_hidden_backward) %<-%
        encoder(list(x, encoder_init_hidden))

      # Setting hidden decoder state to encoder hidden states
      dec_hidden <- k_concatenate(list(enc_hidden_backward,enc_hidden_forward))

      # decoding the batch
      c(preds, weights) %<-%
          decoder(list(dec_hidden, enc_output))

      # Calculating batch loss and accuracy
      loss <- loss + cx_loss(y, preds)
      accuracy <- cx_accuracy(y, preds)
    })

    # Obtaining batch gradients
    variables <- c(encoder$variables, decoder$variables)
    gradients <- tape$gradient(loss, variables)

    # Performing gradient descent update
    optimizer$apply_gradients(purrr::transpose(list(gradients, variables)))

  })

  # Obtaining predictions for training and validation sets
  y_pred_train <- k_constant(get_result(x_train))
  y_pred_val <- k_constant(get_result(x_val))

  # Updating training and validation loss and accuracy for the epoch
  train_loss_act[epoch] <-
    (cx_loss(k_constant(y_train), y_pred_train)) %>%
    as.double()
  val_loss_act[epoch] <-
    (cx_loss(k_constant(y_val), y_pred_val)) %>%
    as.double()
  train_accuracy_act[epoch] <-
    (cx_accuracy(k_constant(y_train), y_pred_train)) %>%
    as.double()
  val_accuracy_act[epoch] <-
    (cx_accuracy(k_constant(y_val), y_pred_val)) %>%
    as.double()
}

# Setting end time
end.time <- Sys.time()

# Returning time taken
time.taken <- end.time - start.time
time.taken
```

```
## Time difference of 5.070964 hours
```

Evaluating the model

We display the loss and accuracy for both the training set and validation set after each epoch.

```
dat1 <- data.frame("data" = as.factor(c(rep("Training", n_epochs),
      rep("Validation", n_epochs))), "epoch" = rep(1:n_epochs,2),
      "loss"=c(train_loss_act, val_loss_act))
dat2 <- data.frame("data" = as.factor(c(rep("Training", n_epochs),
      rep("Validation", n_epochs))), "epoch" = rep(1:n_epochs,2),
      "accuracy"=c(train_accuracy_act, val_accuracy_act))

# Plotting results
p <- ggplot(dat1, aes(x = epoch, y = loss)) +
  geom_point(aes(col=data)) +
  geom_line(aes(col=data)) +
  ggtitle("Training and Validation Sets Loss and Accuracy over Epochs")
q <- ggplot(dat2, aes(x = epoch, y = accuracy)) +
  geom_point(aes(col=data)) +
  geom_line(aes(col=data)) +
  ylim(0,1)

ggarrange(p, q, ncol = 1, nrow = 2, common.legend = TRUE, legend = "right")
```

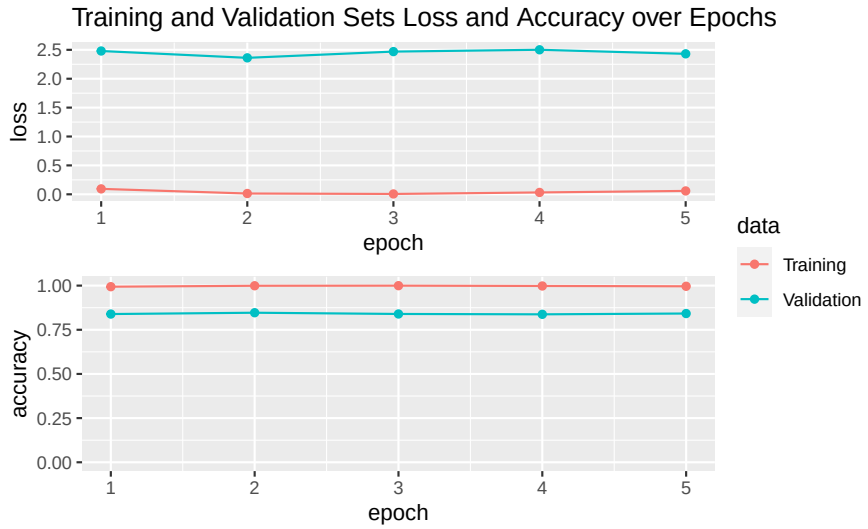

Figure 7: Loss and Accuracy on training and validation sets for activated general attention model.

We predict the result of a negative review and display its word cloud below.

```
c(sentence, prediction, true_result) %<-% translate(negative_sample, 0)
strwrap(sentence,80)
```

```
##  [1] "Input: starting with the usual very dark pour although plenty of light coming"
##  [2] "through head disappeared so quickly i never got a chance to check the color out"
##  [3] "it totally disappeared in about seconds those two features brought the"
##  [4] "appearance numbers down the caramel smell is quite overwhelming i was expecting"
##  [5] "hints of caramel not a huge dousing of it i might as well have a glass of"
##  [6] "caramel i dont pick up anything else in the smell caramel covers all i dont"
##  [7] "know what to say about the taste except bleh there is very little to this beer"
##  [8] "except for what seems like caramel flavoring added to an average dark lager the"
```

```
##  [9] "caramel which is in here tastes overly sugary not like a good toasty caramel"
## [10] "mouthfeel is relatively thin not like any other porter ive had im having issues"
## [11] "drinking any more than is necessary to write this review due to my stomach"
## [12] "beginning to feel strange from all the sweetness shudder i had high hopes for"
## [13] "this i was imaging a creamy full bodied porter with undertones of caramel"
## [14] "coming into presence at the end but there was none of that"
```

```
prediction
```

```
## [1] "Predicted translation: Negative"
```

```
true_result
```

```
## [1] "True translation:  Negative"
```

```
plot_attention(negative_sample)
```

Figure 8: Wordcloud of a negative review.

We predict the result of a positive review and display its word cloud below.

```
c(sentence, prediction, true_result) %<-% translate(positive_sample, 1)
strwrap(sentence,80)
```

```
## [1] "Input: this is a great standard for an oktoberfest it took a little while for"
## [2] "the aroma to open up but once the beer warmed a bit it was rewarding malty with"
## [3] "a nice sweetness good to the nose pour was standard medium copper with a nice"
## [4] "white head and medium lacing very creamy on the palate smooth and clean"
## [5] "unbelievably pleasant mouthfeel flavor like the aroma takes a short while to"
## [6] "warm up and release all of it s potential slightly aggressive and nicely sweet"
## [7] "there is a good bread quality present tops on the list for marzens that i have"
## [8] "encountered so far"
```

```
prediction
```

```
## [1] "Predicted translation: Positive"
```

```
true_result
```

```
## [1] "True translation: Positive"
```

```
plot_attention(positive_sample)
```

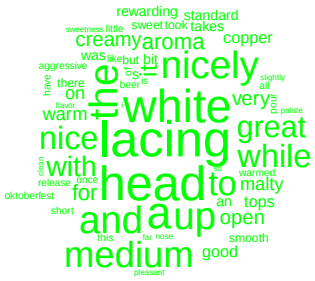

Figure 9: Wordcloud of a positive review.

Assessing the three attention models

```
val_accuracy_additive
```

```
## [1] 0.8515 0.8530 0.8525 0.8410 0.8330
```

```
val_accuracy_multiplicative
```

```
## [1] 0.8495 0.8500 0.8495 0.8480 0.8390
```

```
val_accuracy_act
```

```
## [1] 0.8390 0.8465 0.8395 0.8375 0.8420
```

```
dat1 <- data.frame("model" = as.factor(c(rep("Additive Model", n_epochs),
      rep("Multiplicative Model", n_epochs),
      rep("Activated General Model", n_epochs))), "epoch" = rep(1:n_epochs,3),
      "loss"=c(val_loss_additive,val_loss_multiplicative,val_loss_act))
dat2 <- data.frame("model" = as.factor(c(rep("Additive Model", n_epochs),
      rep("Multiplicative Model", n_epochs),
      rep("Activated General Model", n_epochs))), "epoch" = rep(1:n_epochs,3),
      "accuracy"=c(val_accuracy_additive, val_accuracy_multiplicative,
                   val_accuracy_act))

# Plotting results
p <- ggplot(dat1, aes(x = epoch, y = loss)) +
  geom_point(aes(col=model)) +
  geom_line(aes(col=model)) +
  ggtitle("Validation Set Loss and Accuracy for each model")
q <- ggplot(dat2, aes(x = epoch, y = accuracy)) +
  geom_point(aes(col=model)) +
  geom_line(aes(col=model)) +
```

```
  ylim(0,1)

ggarrange(p, q, ncol = 1, nrow = 2, common.legend = TRUE, legend = "right")
```

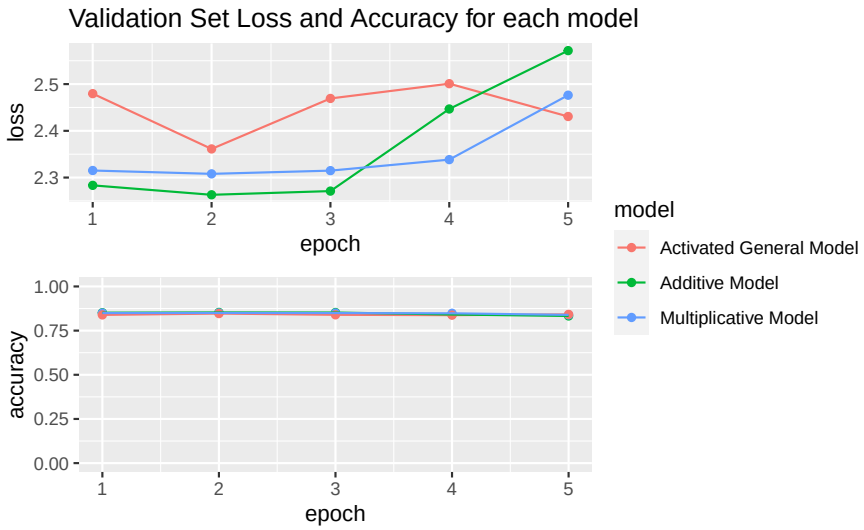

Figure 10: Assessing the three attention models.

Link to course website: https://deeplearningmath.org/