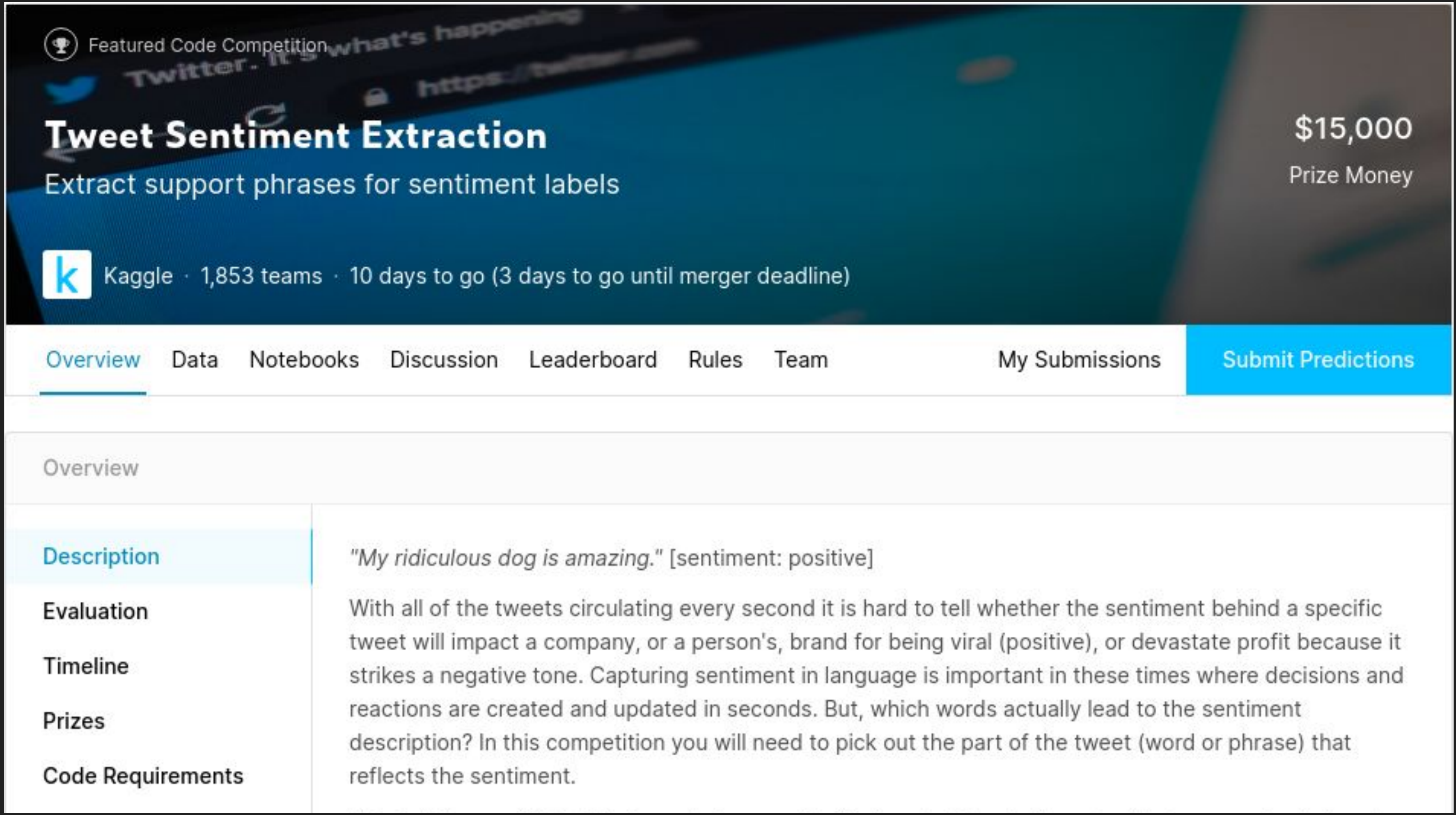
Extraction of sentiment from Tweets

By
Laurent Lindpointner, Orestis Marantos, Giorgos Garidis, Carlos Rodríguez

The project


Featured Code Competition
Tweet Sentiment Extraction
Extract support phrases for sentiment labels
$15,000
Prize Money
Kaggle · 1,853 teams · 10 days to go (3 days to go until merger deadline)
Overview
Data
Notebooks
Discussion
Leaderboard
Rules
Team
My Submissions
Submit Predictions
Overview
Description
Evaluation
Timeline
Prizes
Code Requirements
"My ridiculous dog is amazing." [sentiment: positive]
With all of the tweets circulating every second it is hard to tell whether the sentiment behind a specific tweet will impact a company, or a person's, brand for being viral (positive), or devastate profit because it strikes a negative tone. Capturing sentiment in language is important in these times where decisions and reactions are created and updated in seconds. But, which words actually lead to the sentiment description? In this competition you will need to pick out the part of the tweet (word or phrase) that reflects the sentiment.

Dataset

	textID	text	selected_text	sentiment
26804	6675f9536d	: Aww, that sux! _x3: Eeek for Airline charge...	that sux!	negative
4172	7440e87ea2	O dear! HE`S HERE! OMGOGMGO.. U didn`t see th...	O dear! HE`S HERE! OMGOGMGO.. U didn`t see tha...	neutral
13782	7d64708739	Um. Why can`t I write **** tonight? I like ***...	Um. Why can`t I write **** tonight? I like ***...	positive
18553	8c4a57dd60	Special mention for the new Mean Girl ... welc...	Special mention for the new Mean Girl ... welc...	neutral
17264	2f42dff0dd	school for a bit. glad jake got the day off	glad jake got the day off	positive
5120	e387566c3d	haha that photo is too funny! I hope he wasn`...	hope	positive
24916	c6eea72783	Hey Mia!! Go to bed!! (deangeloredman live...	Hey Mia!! Go to bed!	neutral
22545	78dcea89e3	_sweetye I hope so	I hope so	positive
22406	0ffb510d8e	You`re welcome Tila!! I love you!! Wish I cou...	I love you!	positive
3067	63a48f7850	Or this, for that matter: http://bit.ly/SS6Yp ...	So jealous.	negative
5458	6c21d33903	no sorry twitter sucks balls since the replys...	no sorry	negative
27110	054d5c4400	You`re cycling tho` that`s good. Healthy eati...	You`re cycling tho` that`s good. Healthy eatin...	positive
27260	df0d124770	Going to bed. Hung out w. Aaron and Robin then...	Going to bed. Hung out w. Aaron and Robin then...	neutral
22664	f538b035ae	made me want taco bell, **** you sara! oh wel...	made me want taco bell, **** you sara! oh well...	neutral
22215	a2fb6bdb96	Is losing money in Vegas...	Is losing money in Vegas...	negative

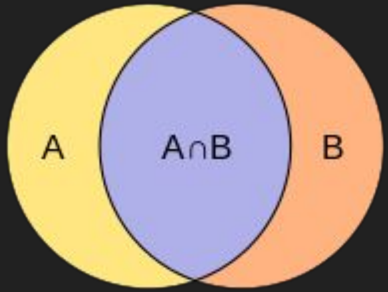
Natural language processing

Before: Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) e.g. LSTMs

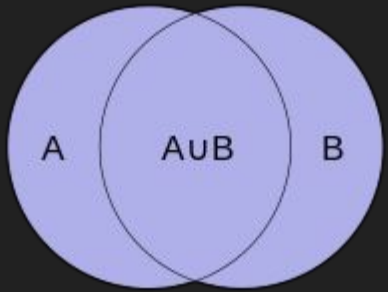
Now: Transformers:
Sequence-to-Sequence architecture NN consisting of an encoder and a decoder. Supported with an attention-mechanism.

Metric

Jaccard index



$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$




```python
def jaccard(str1, str2):
    a = set(str1.lower().split())
    b = set(str2.lower().split())
    c = a.intersection(b)
    return float(len(c)) / (len(a) + len(b) - len(c))
```

Exploratory Data Analysis (EDA)

Training data shape: (27481, 4)

First few rows of the training dataset:

	textID	text	selected_text	sentiment
0	cb774db0d1	I`d have responded, if I were going	I`d have responded, if I were going	neutral
1	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative
2	088c60f138	my boss is bullying me...	bullying me	negative
3	9642c003ef	what interview! leave me alone	leave me alone	negative
4	358bd9e861	Sons of ****, why couldn`t they put them on t...	Sons of ****,	negative

Testing data shape: (3534, 3)

First few rows of the testing dataset:

	textID	text	sentiment
0	f87dea47db	Last session of the day http://twitpic.com/67ezh	neutral
1	96d74cb729	Shanghai is also really exciting (precisely -...	positive
2	eee518ae67	Recession hit Veronique Branquinho, she has to...	negative
3	01082688c6	happy bday!	positive
4	33987a8ee5	http://twitpic.com/4w75p - I like it!!	positive

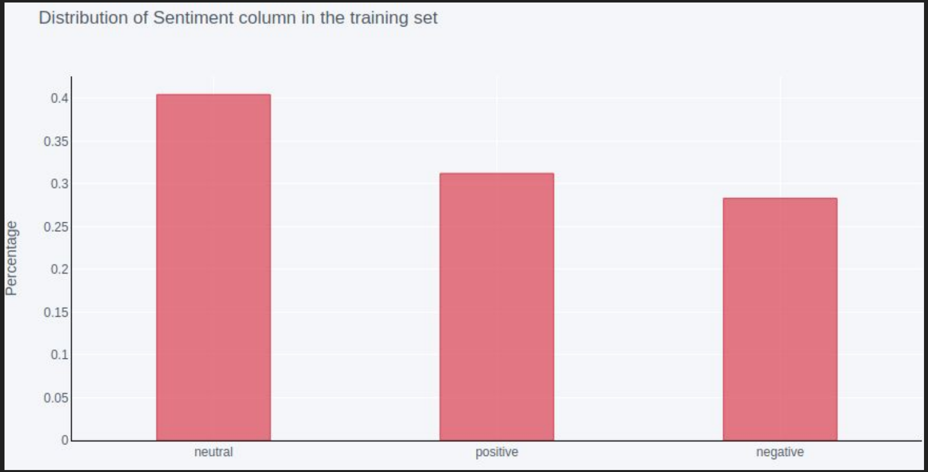
Examples of each sentiment:

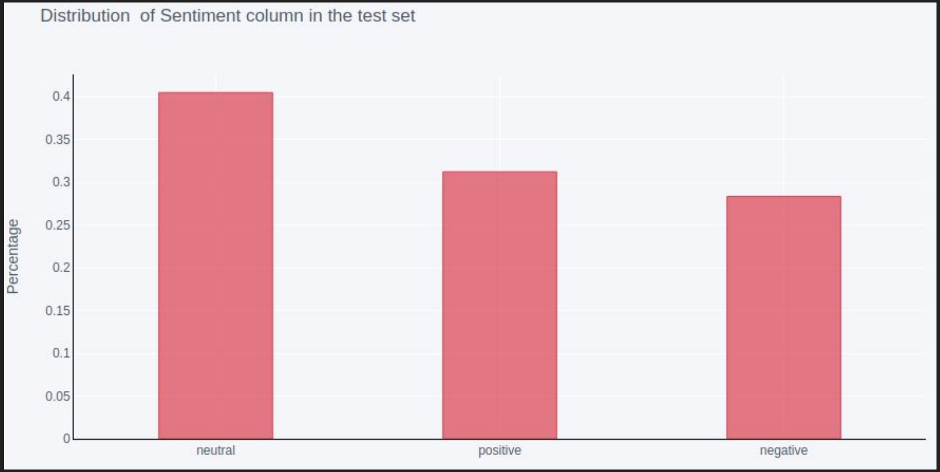
```
Positive Tweet example : 2am feedings for the baby are fun when he is all smiles and coos
Negative Tweet example :  Sooo SAD I will miss you here in San Diego!!!
Neutral tweet example  :  I`d have responded, if I were going
```

Separation in 3 categories:

Neutral: 11117 Positive: 8582 Negative: 7781

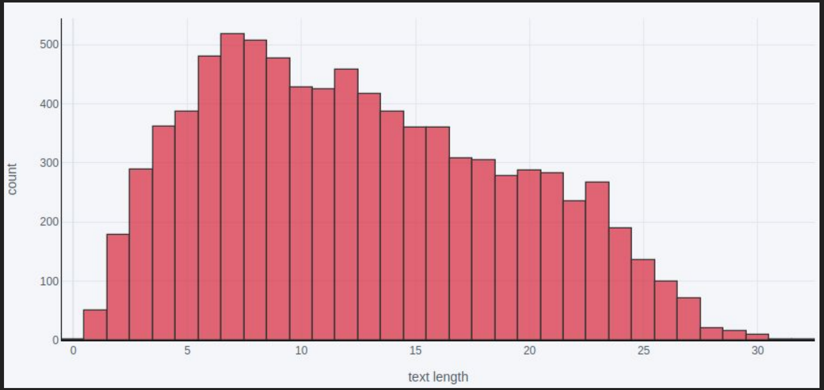
Neutral: 1430 Positive: 1103 Negative: 1001



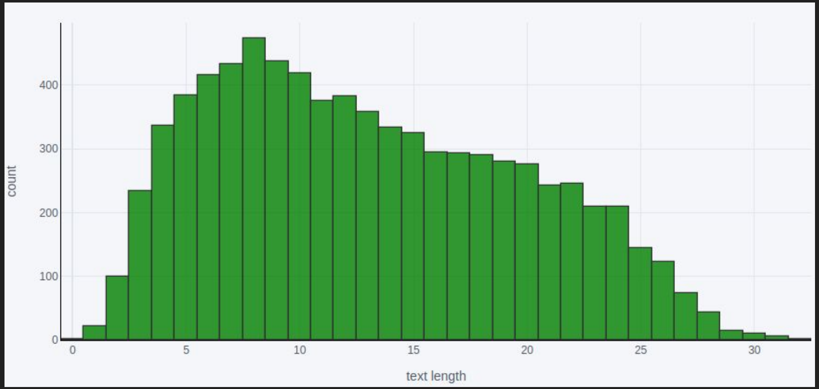


Analyzing text statistics

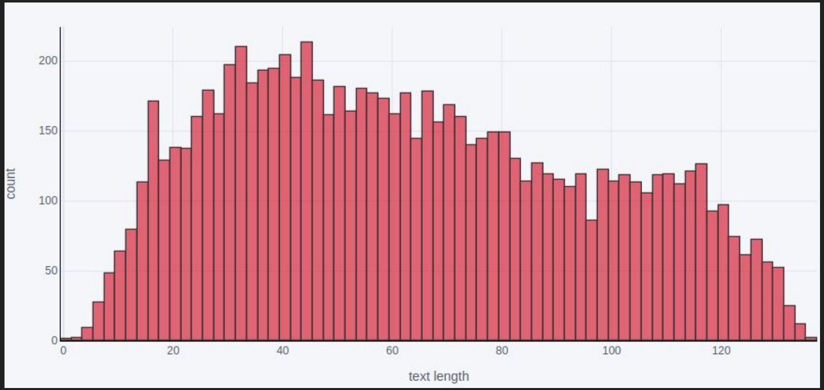
Positive Word Count Distribution



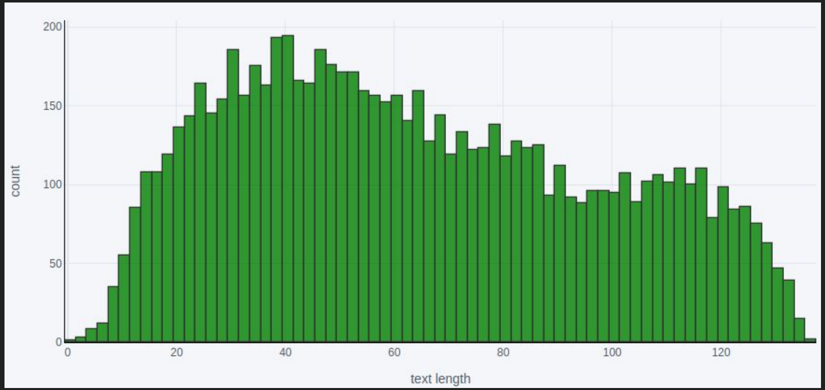
Negative Word Count Distribution



Positive Text length Distribution

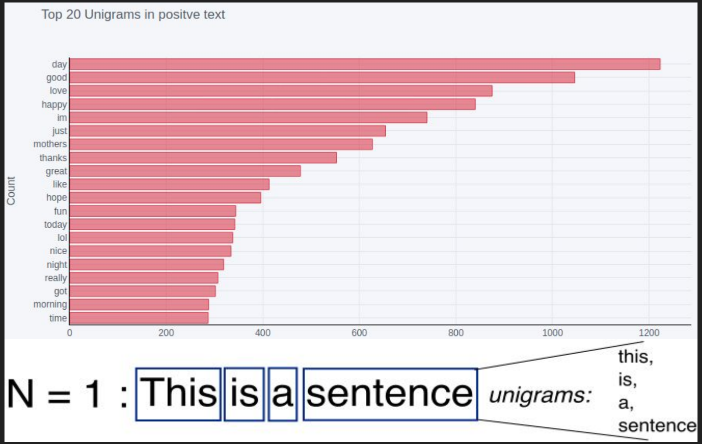


Negative Text length Distribution

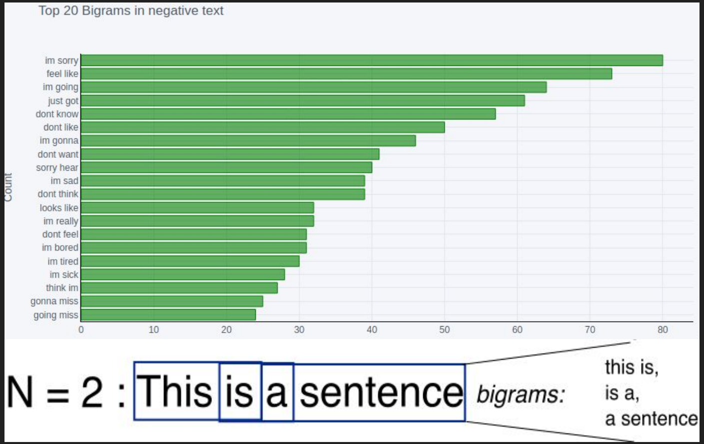


Extracting the most common words from text

Positive text Unigram



Negative text Bigram

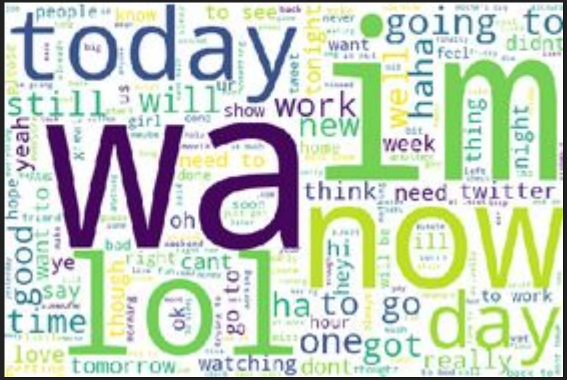


Wordclouds for the selected text column

Positive text

Negative text

Neutral text



Preprocessing - Cleaning & Tokenization

1. Split data: neutral vs. positive & negative tweets
   - Selecting whole tweet for neutral tweets gives Jaccard score of 0.97+
2. Clean from train.text, train.selected_text, test.text:
   - URLs
   - E-mail addresses
   - Emojis 😰
   - @-mentions
   - Numbers
   - Leading white-spaces
   - And put everything to lower-case
3. Prepare data to feed into transformer

	textID	text	selected_text	sentiment
33	2dc51711bc	That`s very funny. Cute kids.	funny.	positive

1. Encode text into vocabulary numbers
   - Step 1: Tokenize → <that> <’> <s> <very> <funny> <.> <cute> <kids> <.>
   - Step 2: Encode → <14> <12905> <29> <182> <6269> <4> <11962> <1159> <4>
2. Encode sentiment into vocabulary numbers
   - <positive> = <1313> or <negative> = <2430>
3. Combine & add separator tokens
   - <s> = <0> start token, </s> = <2> separator token, <p> = <1> padding token
   - Combine to input_ids vector:

<0> <14> <12905> <29> <182> <6269> <4> <11962> <1159> <4> <2> <2> <1313> <1> <1> <1> ...

↓

RoBERTa Model

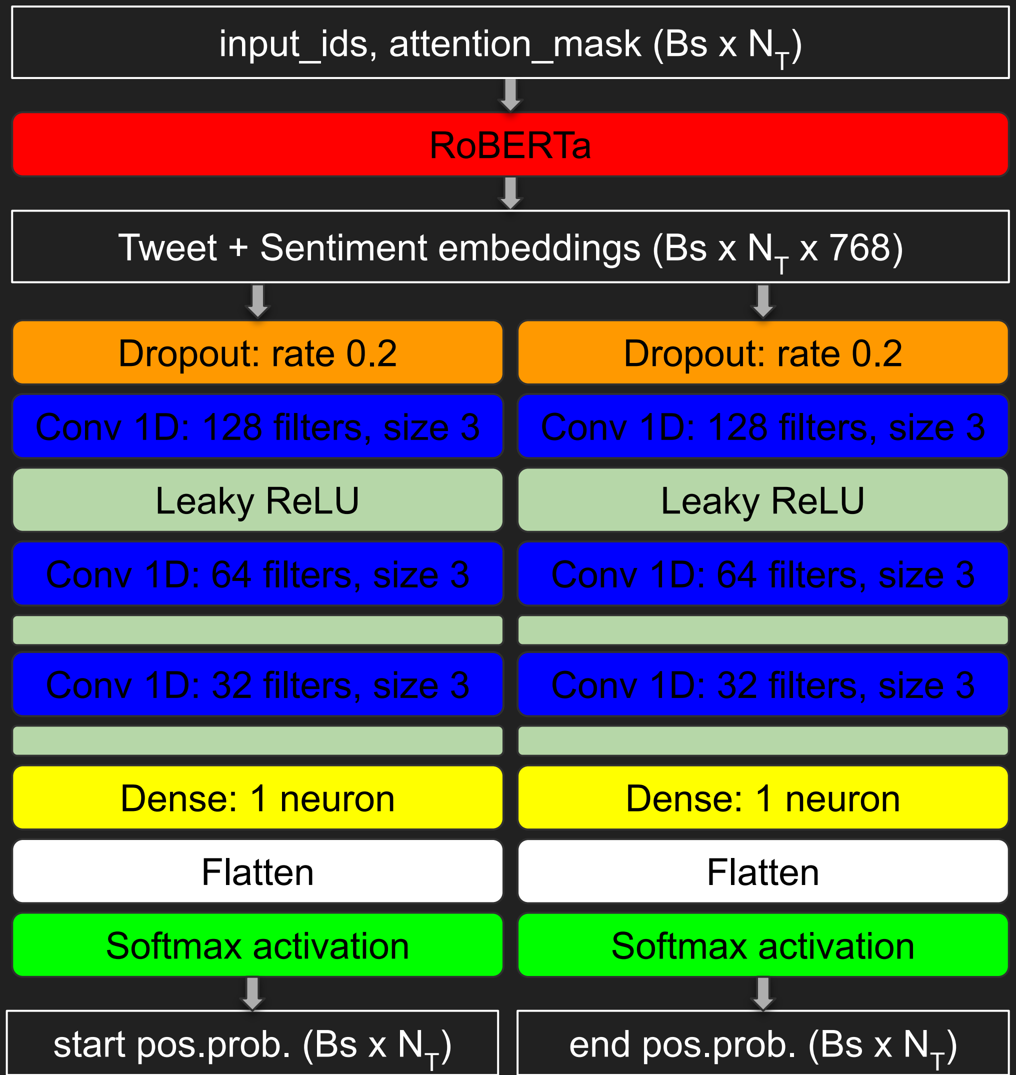
Model

- Create input layers for IDs, Attention Mask.
- Initialize Roberta and create layers of convolutional layers.
- Optimizer: Adam (learning rate 3e-5)
- Loss function: KSLoss

Let ϕ be a word embedding mapping $W \rightarrow \mathbb{R}^n$
where W is the word space and
R^n is an n dimensional vector space then:

$$\phi(\text{"}king\text{"}) - \phi(\text{"}man\text{"}) + \phi(\text{"}woman\text{"}) = \phi(\text{"}queen\text{"})$$

N_T = number of tokens, Bs = batch size



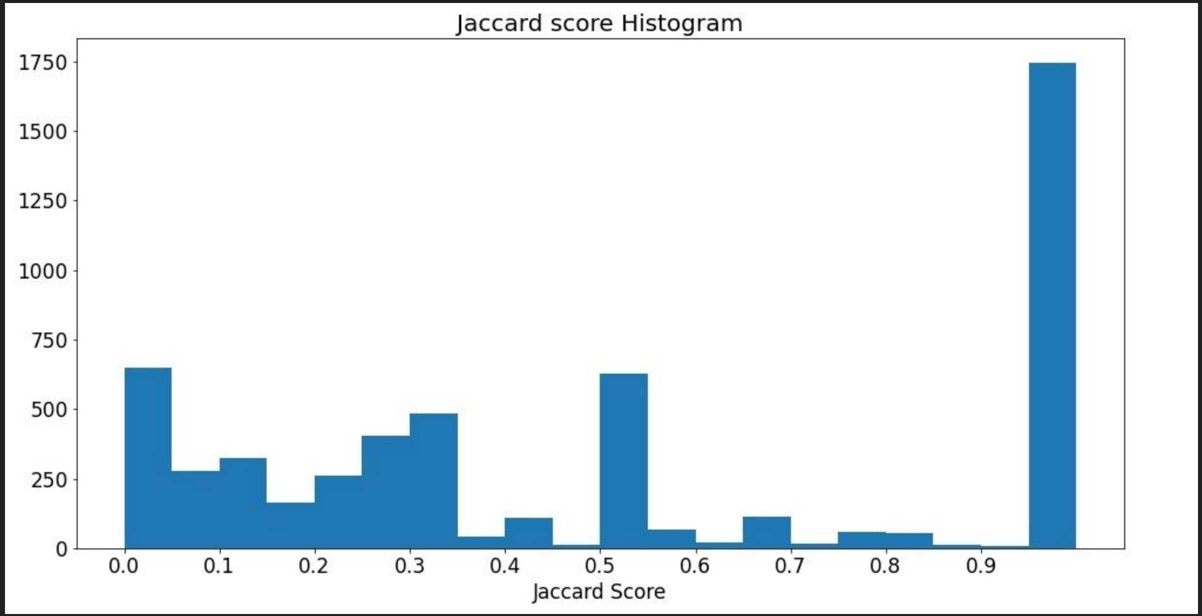
Hyperparameter Optimization

Optimized **4** hyperparameters with Grid Search:	Best values:
1. Number of Convolutional Layers (e.g. 128,64,32 for 3 layers in decreasing order)	3
2. Kernel size in each Convolutional layer (same for all layers)	3
3. Dropout rate (randomly set values to zero, prevents overfitting)	0.2
4. Distance weight in loss function (penalizes distributed values more)	0.1

Challenges & what didn’t work

- Inconsistent labelling/Noise
- Training/CV non-deterministic
- Jaccard is strict

	textID	text	selected_text	sentiment
10651	9984b547fe	Had a lovely Mothers Day	lovely Mo	positive



Results

Best Kaggle Jaccard score: 0.704

Bad Jaccard Score:

sentiment: negative
text: really hopes her car`s illness is not terminal...
selected_text: illness
prediction: really hopes

sentiment: positive
text: jonas brothers - live to party. it`s rocking so hard i love the song,
selected_text: jonas brothers - live to party. it`s rocking so hard
prediction: i love the song,

sentiment: negative
text: im soo bored...im deffo missing my music channels
selected_text: bored..
prediction: im soo bored...

Medium Jaccard Score:

sentiment: negative
text: my sharpie is running dangerously low on ink
selected_text: dangerously
prediction: running dangerously low

Perfect Jaccard Score:

sentiment: positive
text: juss came backk from berkeleyy ; omg its madd fun out there havent been out there in a minute . whassqoodd ?
selected_text: fun
prediction: fun

sentiment: negative
text: why are you sad?
selected_text: sad?
prediction: sad?

References

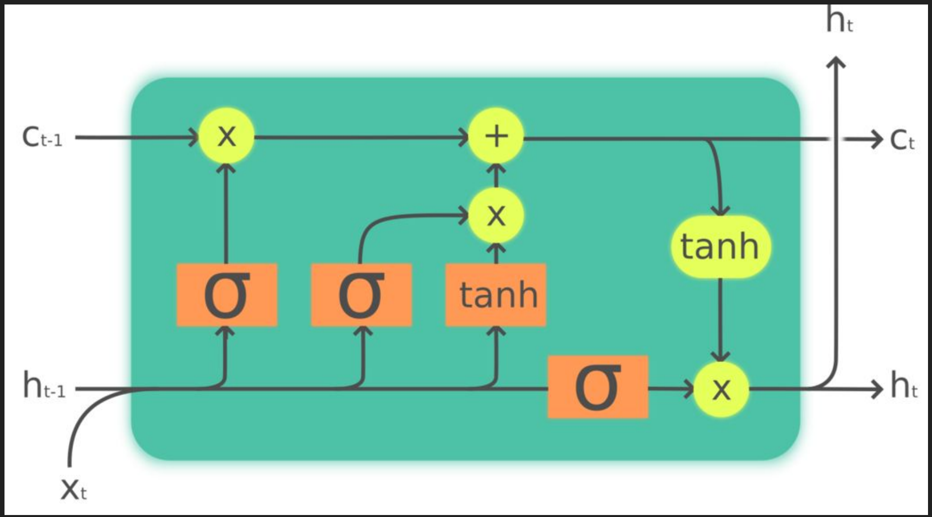
1. Kaggle competition: https://www.kaggle.com/c/tweet-sentiment-extraction
2. What is a Transformer? https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04
3. Starter code from Kaggle: https://www.kaggle.com/cdeotte/tensorflow-roberta-0-705
4. Loss function: https://www.kaggle.com/c/tweet-sentiment-extraction/discussion/147704

We all contributed equally to the project.

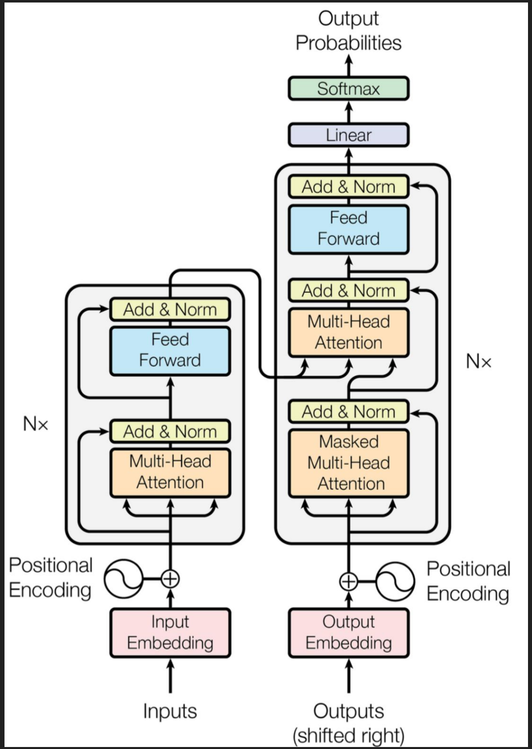
Appendix

NN architecture

LSTM Architecture



Transformer architecture



Preprocessing & Tokenization

Preprocessing Sentimental Tweets

- Firstly we found special cases (URLs, Emojis, Punctuation, Numbers, etc)

```
def find_url(string):                          # Find and return all samples containing URLs
    text = re.findall('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+',string)
    return "".join(text) # converting return value from list to string
def find_emoji(text):                          # Find and return all samples which contain emojies
    emo_text=emoji.demojize(text)
    line=re.findall(r'\:(.*?)\:',emo_text)
    return line
```

- Secondly we removed non relevant special cases (URLs, Emojis, Punctuation, etc)

```
def remove_email(text):
    text = re.sub(r'[\w\.-]+@[\w\.-]+','',str(text))
    return text
def remove_at(text):
    text = re.sub(r'[@]+\w+','',text)
    return text
```

*Neutral Preprocessing investigated separately

- Firstly, we initialized matrices which we used for the tokenization, overlapping of text & selected_text

```
def preprocess(train):
    ct = train.shape[0]
    input_ids = np.ones((ct,MAX_LEN),dtype='int32')
    attention_mask = np.zeros((ct,MAX_LEN),dtype='int32')
    token_type_ids = np.zeros((ct,MAX_LEN),dtype='int32')
    start_tokens = np.zeros((ct,MAX_LEN),dtype='int32')
    end_tokens = np.zeros((ct,MAX_LEN),dtype='int32')
```

- Secondly, we tokenized each text & selected_text and we we begun the overlapping procedure

```
for k in train.index:
    # FIND OVERLAP WITHIN STRING & ENCODE INPUT TEXT
    text1 = " "+" ".join(train.loc[k,'text'].split())      # Introducing whitespace before first to assist tokenization
    text2 = " ".join(train.loc[k,'selected_text'].split()) # Same here
    idx = text1.find(text2)                                 # finding index where overlap begins
    chars = np.zeros((len(text1)))                          # chars: vector holding 1s for overlap, 0 for no overlap
    chars[idx:idx+len(text2)]=1                             # introducing 1 for overlap
    if text1[idx-1]==' ': chars[idx-1] = 1                  # 1 also for first blank token
    enc = tokenizer.encode(text1)                           # transform text1 word tokens into vocab numbers
```

- Lastly, we found offsets for each word in the text (start, end indices) and positions of the overlapping tokens

```
# POSITIONS OF OVERLAPPING TOKENS
toks = []                          # Positions of overlapping tokens
for i,(a,b) in enumerate(offsets): # look for the positions of the overlapping tokens within all tokens
    sm = np.sum(chars[a:b])        # num of overlapping charachters in chars, 0 if none overlap
    if sm>0: toks.append(i)        # appending if there are overlapping characters for that token

s_tok = sentiment_id[train.loc[k,'sentiment']]  # Getting encoded id of sentiment according to vector defined above
input_ids[k,:len(enc.ids)+5] = [0] + enc.ids + [2,2] + [s_tok] + [2] # build encoded Tweet + Sentiment + separtor tokens
attention_mask[k,:len(enc.ids)+5] = 1           # ones where there's tokens, 0s where there's none
if len(toks)>0:
    start_tokens[k,toks[0]+1] = 1               # 1 at token-position of overlap start
    end_tokens[k,toks[-1]+1] = 1                # same for overlap end
```

Building the model

Firstly, we need to load the RoBERTa transformer

```
config = RobertaConfig.from_pretrained(PATH+'config-roberta-base.json')
bert_model = TFRobertaModel.from_pretrained(PATH+'pretrained-roberta-base.h5',config=config)
x = bert_model(ids_,attention_mask=att_,token_type_ids=tok_)
```

Secondly, we create the embedding layers for the model and compile it for the unpadded model, it runs faster. Afterwards, we create a model with padded variables, it is essential for prediction.

```
x1 = tf.keras.layers.Dropout(DROPOUT_RATE)(x[0])
x1 = tf.keras.layers.Conv1D(1,1)(x1)

x2 = tf.keras.layers.Dropout(DROPOUT_RATE)(x[0])
x2 = tf.keras.layers.Conv1D(1,1)(x2)

model = tf.keras.models.Model(inputs=[ids, att, tok], outputs=[x1,x2])
model.compile(loss=loss, optimizer=optimizer)
x1_padded = tf.pad(x1, [[0, 0], [0, MAX_LEN - max_len]], constant_values=0.)
x2_padded = tf.pad(x2, [[0, 0], [0, MAX_LEN - max_len]], constant_values=0.)

padded_model = tf.keras.models.Model(inputs=[ids, att, tok], outputs=[x1_padded,x2_padded])
```

Building the model

We used a Loss function that focuses in penalising how far is out prediction for the actual position:

```
class DistanceLoss(tf.keras.losses.Loss):
    def __init__(self, distance_weight=0.1):
        super().__init__()
        self.__distance_weight = distance_weight

    def call(self, y, pred):
         ll = tf.shape(pred)[1]
         y = y[:, :ll]
         pred_scalar = tf.math.argmax(pred, axis=1)
         y_scalar    = tf.math.argmax(y, axis=1)
         Bin_cross   = tf.keras.losses.binary_crossentropy(y, pred)
         cst         = tf.cast(tf.math.abs(y_scalar - pred_scalar), dtype=tf.float32)
         return  Bin_cross + cst * self.__distance_weight
```

We use the Adam optimization function

```
optimizer = tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)
```

Training the model

We fit the un padded model with a Cross Validation. As a callback we implement a function to save the weights. The weights are used later to run the padded model.

```
skf = StratifiedKFold(n_splits=3,shuffle=True,random_state=SEED)
for fold,(idxT,idxV) in enumerate(skf.split(input_ids,train.sentiment.values)):

    tf.keras.backend.clear_session()
    model, padded_model = build_model(**parameters)

    sv = tf.keras.callbacks.ModelCheckpoint(PATH + 'Model_Weights/%s-roberta-D15-%i.h5'%(VER,fold),
                                            monitor='val_loss', verbose=1,
                                            save_best_only=True,
                                            save_weights_only=True, mode='auto', save_freq='epoch')

    model.fit([input_ids[idxT,], attention_mask[idxT,], token_type_ids[idxT,]],
              [start_tokens[idxT,], end_tokens[idxT,]],
              epochs=3, batch_size=32, verbose=DISPLAY, callbacks=[sv],
              validation_data=([input_ids[idxV,],attention_mask[idxV,],token_type_ids[idxV,]],
              [start_tokens[idxV,], end_tokens[idxV,]]))
```

Training the model

We load the weights from the padded model to give them to the unpadded one for predicting. The prediction gives us the starting and ending positions of the selected text over the given tweets.

```
model.load_weights(PATH + 'Model_Weights/%s-roberta-D15-%i.h5'%(VER,fold))
oof_start[idxV,], oof_end[idxV,] = padded_model.predict([input_ids[idxV,],attention_mask[idxV,],
                                                          token_type_ids[idxV,]],verbose=DISPLAY)

preds        = padded_model.predict([input_ids_t,attention_mask_t,token_type_ids_t],verbose=DISPLAY)
preds_start += preds[0]/skf.n_splits
preds_end   += preds[1]/skf.n_splits
```

Training the model

We transform back the predicted values obtain and calculate the average Jaccard index.

```
for k in idxV:
  a = np.argmax(oof_start[k,])
  b = np.argmax(oof_end[k,])
  text1 = " "+" ".join(train.loc[k,'text'].split())
  enc = tokenizer.encode(text1)
  st = tokenizer.decode(enc.ids[a-1:b])
  all.append(jaccard(st,train.loc[k,'selected_text']))
jac.append(np.mean(all))
```

Hyperparameter Optimization

Code snippet for optimizing Number of convolutional Layers and Kernel Size.

```
layer_sizes = np.flip(2**np.arange(CONV_LAYERS)*32)

for conv_size in layer_sizes:
    x1 = tf.keras.layers.Conv1D(filters = conv_size, kernel_size=int(KERNEL_SIZE), padding='same')(x1)
    x1 = tf.keras.layers.LeakyReLU()(x1)

    x2 = tf.keras.layers.Conv1D(filters = conv_size, kernel_size=int(KERNEL_SIZE), padding='same')(x2)
    x2 = tf.keras.layers.LeakyReLU()(x2)
```

```
param_range = { 'Nlayers':np.arange(1,6+1) , 'Kernel_size':np.arange(1,11+1,2)}

for _ , Ksize in enumerate(param_range['Kernel_size']):
    for _ , Lsize in enumerate(param_range['Nlayers']):
        print(Lsize,Ksize)
        parameters = {'KERNEL_SIZE':Ksize ,'CONV_LAYERS':Lsize}
```

Hyperparameter Optimization

Code snippet for hyperparameter optimization on distance weight.

```
Param_write = {'DROPOUT_RATE': 0.20, 'LEARNING_RATE': 3e-5, 'distance_weight': 0.1}
               #{'DROPOUT_RATE': 0.20, 'LEARNING_RATE': 3e-5, 'distance_weight': 0.2}
               #{'DROPOUT_RATE': 0.20, 'LEARNING_RATE': 3e-5, 'distance_weight': 0.3}

jac, preds_start, preds_end, a_larger_b = training(cleaned_train[:num_train_samples],
input_ids_train, attention_mask_train, token_type_ids_train,start_tokens_train,
end_tokens_train,input_ids_t, attention_mask_t, token_type_ids_t,SEED, drop_learn)
```

The same was repeated for the Dropout rate.