**Article Title Extrapolation**

|  |  |  |
| --- | --- | --- |
| Yaman El-Jandali | – | yeljanda@ucsd.edu |
| Tamuz Hod | – | thod@ucsd.edu |
| Richard Andrus | – | randrus@ucsd.edu |

**Abstract**

The purpose of this experiment was to see if we could use the content of articles in order to predict their titles. Extrapolating a single (fragmented) sentence from a long article was ultimately a difficult task for our model but although our BLEU scores (shown in the results section) are fairly low, we think this is partially because of synonymous words and phrases and when analyzing the results, we were impressed by many of the predicted article titles that BLEU deemed to be poor predictions.

**1. Introduction**

An article title is more than just a summary of an article. Rather than give an overview with all the important details of an article, it is often a hook that grabs the reader’s attention while giving them a brief idea of what the article is going to discuss. In that way, article titles are harder for machines to generate because they are far more subjective than a summary of an article, and when they are written, basic rules of grammar are often ignored they are rarely complete sentences. Furthermore, more modern article titles tend to be geared more towards “click-bait” and become even more difficult to discern. Because of these added complexities, our team chose to see how well we could train a model to predict titles of articles based on their content.

**2. Model**

With our aim towards word generation from content in mind, we chose a seq2seq model comprised of a basic encoder-decoder network. However, we had to take several important aspects of our goal and data into consideration when building our architecture. Firstly, the content of our articles could be quite long. This meant that with regards to using a basic RNN model in our encoder-decoder network, vanishing gradients would likely become a major obstacle. Additionally, when attempting to make a short article title from a long article, it is difficult to find the best suited attention model which can properly map out the importance of each word within the content and string it into a coherent headline.

**2.1 Encoder-Decoder Model**

As mentioned earlier, we used a GRU RNN for our encoder-decoder architecture. More specifically we used a bidirectional GRU RNN because that’s what many have used and recommend using for text summarization which is the closest aim we were able to find to article title generation from content. In a bidirectional model, each hidden state will be the result of a recursive call passing in the previous hidden state. That is, for a specified hidden state of the encoder, hEi , and the input xi (the current embedding vector for the ith word) we have:

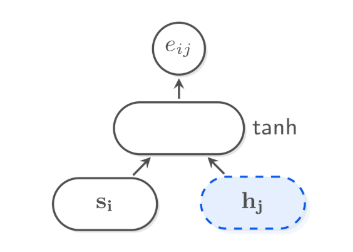
**hEi = GRU(xi , hEi - 1)**

Our forward directional GRU will only give us the information needed about all of the context preceding our word. Similarly, the backward GRU will only give us information needed about the context after our word. By concatenating these two hidden states (one from our forward and from our backward GRU) we now have a complete set of information regarding the word in question. After making use of the attention model (described in 2.2) we use a decoder and generate our predicted titles (described more in section 2.3).

Although pytorch has GRU fully implemented for us (including bidirectional), for more information regarding GRUs and how they help solve the issue of vanishing gradients (using update and reset gate vectors) we recommend the article listed in our citations at ([2](https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be)).

**2.2 Bahdanau Attention Model**

At each time step the attention layer reduces the importance of some of the input word representations. This enables our model to focus only on the most relevant information at any given point. We do so by having a function that takes the current word representations (**hi**) as well as the decoder state as inputs. These states are the GRU hidden states (**sj**).



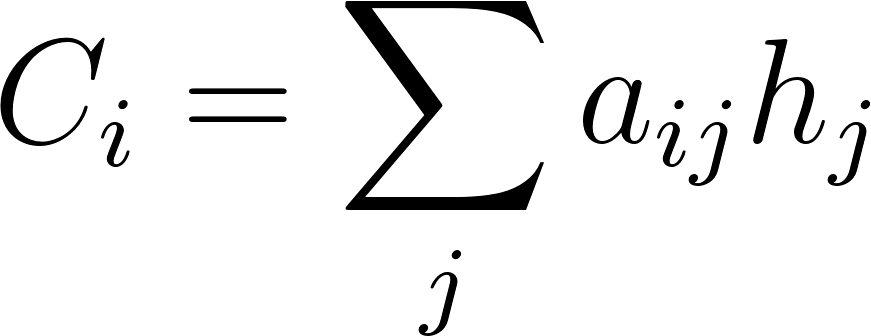
*Figure 1*

*(source: Joost)*

We use an MLP with a hyperbolic tangent activation function and project the result onto a scalar to get attention energy (**eij**). A visual representation of these steps can be seen in *Figure 1* above. The next step is to normalize these energies using softmax:

**aij  = softmax(ei)[j]**

The final step is to compute the context layer which is nothing but a weighted sum of the encoder state and the attention energy.

[](https://www.codecogs.com/eqnedit.php?latex=%20C_i%20%3D%20%5Csum_%7Bj%7D%20a_%7Bij%7D%20h_j%20%0)

**2.3 Title Generation**

Our model makes at any time step a prediction about the next word in the sentence. Hidden states are generated in a similar fashion when decoding as when encoding (described in 2.1) but the formula utilizes previously generated word, which we will call wordi-1 here. Taking ci to be the context vector (described in more detail in the attention section, 2.2) which is an additional input needed for decoding and hDi-1 to be the previous hidden state, but this time of the decoder. We therefore have:

**hDi = f(hDi-1 , wordi-1 , ci)**

After which we apply the softmax to gain our probability distribution. The softmax function is the same as the one used in class. The general formula is:

**softmax(xi) =**

From here we use the greedy approach for our actual title generation in that, now that we have our probability distribution from above, we simply go through our choices picking the most probable options at each step. At this point, we have our predicted title.

**3. Datasets**

The dataset our team used can be found on [Kaggle](https://www.kaggle.com/snapcrack/all-the-news/discussion/38296). (1) It is comprised of approximately 143,000 English news articles scraped from 15 publications. The majority of these articles were published between 2016 and 2017. Considering that all of the publications included are American and 2016 was a controversial election year in America, much of the data set is politicized and this can be seen in the results.

**3.1 Processing Our Data**

We used a fairly light preprocessing regime as we wanted to preserve the ability to quickly slot in plain text news articles from online without the need for heavy preprocessing. We removed all non alphabetic characters other than punctuation marks. We also expanded common contractions and removed words smaller than 3 letters. We then looked at the length of both the article bodies and the titles to determine appropriate maximum lengths used in our model for the input and output tensors. We decided to go with a 500 world limit for our inputs (which represented approximately 75% of our data) and a 25 world limit capturing 99% of our data.

**3.2 Splitting our Data**

We randomly split our dataset into train, validation, and test set with 70%, 20%, and 10% of the data respectively.

**3.3 Handling Varying Sequence Lengths**

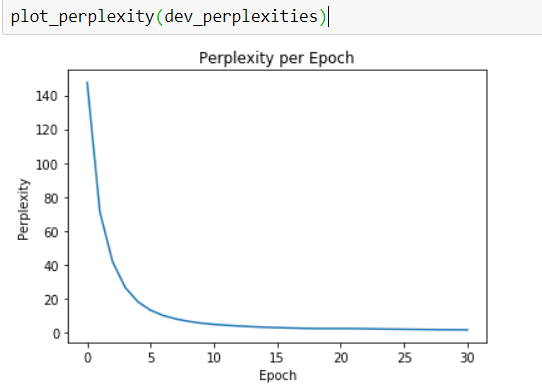
In order to deal with non standard length input and output sequences we padded all our sequences with <pad> tokens to the above mentioned word limit. When calculating loss the pad token was ignored. This prevents the model from unnecessarily updating parameters.



|  |  |
| --- | --- |
| **Word** | **Frequency** |
| said | 441823 |
| trump | 297644 |
| would | 171087 |
| one | 169913 |
| people | 165593 |
| new | 148306 |
| president | 135149 |

A single data point from our data set consisted of the content of an article as well as the article header. The article headers include start and end tags for each individual header.

**4. Results**

**4.1 Training and Validation**

We trained our model on the train dataset for as long as we could training for a total of 31 epochs. At every epoch we saved our model weights which we later used to calculate our validation loss at each epoch. To the right is our loss over epoch curve. Past the 30th epoch the validation loss started to increase so we used the weights from the 30th epoch as our final model. The results of our training can be seen in the graph included in *Figure 3* to the right.

**4.2 Qualitative Testing**

After training for thirty epochs, our model was able to produce titles with moderately high levels of grammar and accuracy. For example:

Trg : california tightens gun control laws expands assault weapons ban

Pred: jerry brown signs gun control plea in california

Trg : california deal could make state first with minimum wage

Pred: california governor jerry brown to raise for minimum wage

Trg : california black lives matter proud of facebook post saying they <unk> in police officer starbucks breitbart

Pred: black lives matter demands arm black driver ad after ad breitbart

Trg : environmental agencies reject vw plan for recall over emissions

Pred: volkswagen fires warning after consumers emissions cheating scandal

From a cursory look at these examples, we can point out a couple things with these predictions. First, despite many quirks in the predictions, such as “gun control plea” or “raise for minimum wage”, the grammar in the titles is good and the titles resemble titles. Similarly, the prediction titles discuss the same topics as their targets, even when the words they use are quite different. This indicates that the model had high recall but low precision, meaning its titles reflected the respective article’s content more than they match predicted titles word-for-word.

**4.3 Quantitative Testing**

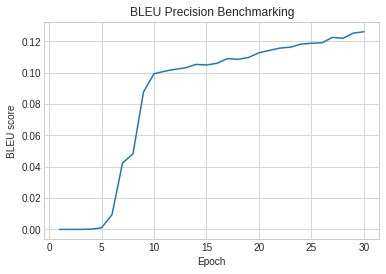
To test our model, we used a few different metrics that quantified performance throughout training, including perplexity and BLEU.

BLEU is a performance metric for text generation that measures precision, or how closely generated text matches the expected text. It does this by calculating the percentage of monograms and bigrams in the generated text that also occur in the expected text, or in our case, target titles and predicted titles. For example, consider the following prediction from our results:

Trg : california tightens gun control laws expands assault weapons ban

Pred: jerry brown signs gun control plea in california

To calculate the BLEU score for this prediction, we note that there are eight words in the generated title. This equates to eight monograms (single words) and seven bigrams (two-word pairs). Of these fifteen ngrams, three of the monograms appear in the target title (“gun”, “control”, and “california”) and only one bigram appears (“gun control”). Thus, the percentage of possibly shared ngrams between the two titles is 4/15 or 27%. The results of our benchmarking can be seen in the table included in *Figure 4* below.





**5. Conclusion**

Although the results of BLEU were not as high as we would have hoped, looking at the target and predicted values it is easy to see a flaw in using BLEU for our error checking.

One interesting conclusion of our experiment was the outputting of publication names as a part of the title. This was because we forgot to remove the publication name that was concatenated to our titles in our data set. Although we ended up removing them afterwards, upon looking over the results we noticed that in most cases when the publication was output as part of the title, it was the correct publication even if the name of that publication was never mentioned in the article. This could potentially mean that different publications have different writing styles or topics of focus which were likely distinguished by our algorithm (although this was not our intention). This would have been an interesting topic of further research, had time allowed.

**Citations**

1. <https://www.kaggle.com/snapcrack/all-the-news/discussion/38296>
2. <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>
3. Joost, Bastings. 2018. The Annotated Encoder-Decoder with Attention. <https://bastings.github.io/annotated_encoder_decoder/>
4. <https://github.com/spro/practical-pytorch/blob/master/seq2seq-translation/seq2seq-translation.ipynb>