- we are gonna start with CNN/DailyMail dataset to work on summarization task
- The dataset has three columns: article, which contains the news articles, high lights with the summaries, and id to uniquely identify each article
- A convention in summarization is to separate the summary sentences by a newline. We could add a newline token after each full stop, but this simple heuristic would fail for strings like “U.S.” or “U.N.” The Natural Language Toolkit (NLTK) package includes a more sophisticated algorithm that can differentiate the end of a sentence from punctuation that occurs in abbreviations
- GPT-2 can generate text given some prompt. One of the model’s surprising features is that we can also use it to generate summaries by simply appending “TL;DR” at the end of the input text. The expression “TL;DR” (too long; didn’t read)
- Like BART, PEGASUS is an encoder-decoder transformer its pretraining objective is to predict masked sentences in multisentence texts
- The first thing we notice by looking at the model outputs is that the summary gener- ated by GPT-2 is quite different from the others. Instead of giving a summary of the text, it summarizes the characters. Often the GPT-2 model “hallucinates” or invents facts, since it was not explicitly trained to generate truthful summaries. For example, at the time of writing, Nesta is not the fastest man in the world, but sits in ninth place. Comparing the other three model summaries against the ground truth, we see that there is remarkable overlap, with PEGASUS’s output bearing the most striking resemblance.
- The standard metrics that we’ve seen, like accuracy, recall, and precision, are not easy to apply to this task. For each “gold standard” summary written by a human, dozens of other summaries with synonyms, paraphrases, or a slightly different way of formulat- ing the facts could be just as acceptable.
- Two of the most common metrics used to evaluate generated text are BLEU and ROUGE
- BLEU is a precision-based metric, which means that when we compare the two texts we count the number of words in the generation that occur in the reference and divide it by the length of the generation
- The problem we face here that the generated sentence might contains repeated words For this reason, the authors of the BLEU paper introduced a slight modification: a word is only counted as many times as it occurs in the reference. To illustrate this point, suppose we have the reference text “the cat is on the mat” and the generated text “the the the the the the”. From this simple example, we can calculate the precision values as follows: pvanilla = 6/6 pmod = 2/ 6
- Another weakness of the BLEU metric is that it expects the text to already be tokenized. This can lead to varying results if the exact same method for text toke- nization is not used. The SacreBLEU metric addresses this issue by internalizing the tokenization step; for this reason, it is the preferred metric for benchmarking.
- There are other applications, such as summarization, where the situation is different. There, we want all the important information in the generated text, so we favor high recall. This is where the ROUGE score is usually used.
- The ROUGE score was specifically developed for applications like summarization where high recall is more important than just precision
- There is a separate score in ROUGE to measure the longest common substring (LCS), called ROUGE-L. The LCS can be calculated for any pair of strings. For example, the LCS for “abab” and “abc” would be “ab”, and its the length would be 2. If we want to compare this value between two samples we need to somehow normalize it because otherwise a longer text would be at an advantage. To achieve this, the inventor of ROUGE came up with an F-score-like

scheme where the LCS is normalized with the length of the reference and generated text, then the two normalized scores are mixed together

- A new thing in the use of the tokenization step is the tokenizer.as_target_token izer() context. Some models require special tokens in the decoder inputs, so it’s important to differentiate between the tokenization of encoder and decoder inputs. In the with statement (called a context manager), the tokenizer knows that it is tokeniz- ing for the decoder and can process sequences according
- when we prepare our batch, we set up the decoder inputs by shifting the labels to the right by one. After that, we make sure the padding tokens in the labels are ignored by the loss function by setting them to –100. We actually don’t have to do this man- ually, though, since the DataCollatorForSeq2Seq comes to the rescue and takes care of all these steps for us
-