Streamed Punctuation Annotation using Transformers

Christoph Minixhofer

Fourth Year Project Report
School of Informatics
University of Edinburgh
2021

Abstract

To improve readability, punctuation prediction is typically performed on text output by an Automatic Speech Recognition (ASR) model. We introduce a Transformer-based model to predict punctuation marks on unpunctuated text suitable for text streamed word-for-word, as is often the case for ASR models. We propose a decoding strategy that delays punctuation marks' insertion in case of uncertainty until a specific threshold is reached. Leveraging existing pre-trained language models in conjunction with a special token for acoustic pause features, we achieve state-of-the-art performance for punctuation prediction on the MGB dataset and results that compare favourably to the state-of-the-art on the IWSLT11 dataset while using comparatively less computing power than previous work by using downsampling. To make the model viable for real-time use in combination with an ASR system and on low-resource devices, we evaluate input truncation and weight quantization. We show these techniques lead to faster-than-real-time inference speeds and a significant reduction in model size.

Acknowledgements

First and foremost, I am grateful to my supervisors, Prof. Steve Renals and Dr Ondřej Klejch. I am extremely thankful for Steve Renals' initial guidance on the project, which paved the way for this work. Due to unfortunate circumstances, Steve Renals had to interrupt supervising this project, and Ondřej became my main point of contact and mentor. His continual guidance and attention to detail allowed me to shape this into a project I am proud of. This is my first major piece of academic writing, and while I still have a lot to learn, I believe I managed to improve thanks to Ondřej's practical tips and experience. I want to thank my girlfriend, Celina, for always listening, emphasising, and for allowing me to ramble about punctuation annotation, although it could not be further from her field of study. My brother Benjamin was always there to provide constructive criticism and feedback. Benjamin, as well as my parents Christine and Rainer, were voices of reason when I got caught up in details while supporting me when I was stressed or exhausted. My flatmate and dear friend Angus was the best technical support anyone could imagine and helped me reset my tunnel to my personal computer in Edinburgh the many times I found creative ways to break it. Finally, I want to thank my therapist Dr Stefan Librowicz for helping me through the more difficult times I experienced while writing this.

Table of Contents

Chapter 1

Introduction

Punctuation annotation is often used in conjunction with automatic speech recognition (ASR). This work focuses on building a punctuation annotation system for this context. ASR systems, in turn, are used in many contexts, such as voice assistants, dictation systems or subtitling. For this work, we divide these into:

a) applications which present the recognised text to the user

b) applications which do not present the recognised text to the user

State-of-the-art ASR systems output streams of words without punctuation (Bakhturina, 2019). For applications of type a), adding punctuation to the raw stream of words produced by the ASR system can aid readability for the user. For applications of type b), punctuation annotation can still be useful as some possible downstream tasks such as machine translation or named entity recognition can yield better results on punctuated text rather than a raw stream of words (Makhija et al., 2019). Of the applications that present recognised text to the user, many will aim to do so in real-time, with words appearing as they are spoken. We call this special case *streamed punctuation annotation*.
Creating a punctuation annotation system for this streamed case comes with the following challenges:

1. When predicting punctuation immediately following each word, words appearing to the right of the possible punctuation mark are not available to the system. We call this *lack of right-side context*.

2. Using acoustic features as shown in Figure 1.1 is likely to be more difficult in a streaming scenario, as information from multiple parts of the pipeline has to be

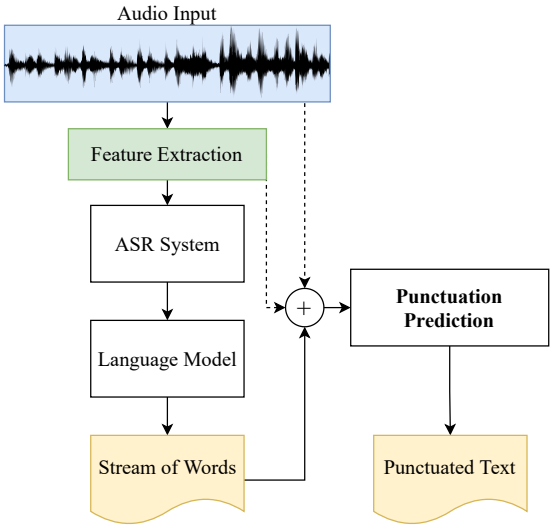


Figure 1.1: Punctuation in an ASR pipeline.

streamed to the punctuation prediction model. We call this *reduced availability of acoustic features*.

3. When predicting punctuation following every streamed word, we should, on average, not use more time than the average word duration. This is one of the most wanted characteristics, which we call the *need for inference speed*.

While we reason that most existing punctuation annotation systems could be adapted to the lack of right-side context, the reduced availability of acoustic features could be hard to overcome in real-world applications. For example, if an ASR system provided as a service by a third party is used in an application, the recorded audio would, in addition to being sent to said ASR system, have to be stored locally and then aligned to the output produced by the ASR system for use with a punctuation annotation system.
In this work, we aim to achieve the following.

1. Perform similarly to state-of-the-art punctuation annotation systems using pre-trained Transformer models.

2. Address the aforementioned challenges associated with *streamed* punctuation annotation of *lack of right side context* and *need for inference speed* using a classification approach in tandem with truncation and quantization.

3. Utilise acoustic features despite their reduced availability using `[PAUSE]` tokens.

Chapter 2

Background

We now outline the methods used for punctuation annotation in the past in the context of the streamed punctuation annotation task.

2.1 Existing Approaches to Punctuation Annotation

Punctuation annotation using learned and statistical models has been studied for more than two decades, with the related task of sentence segmentation predating those efforts even further. We examine the different approaches to punctuation annotation in the following three sections:

1. We examine the data used to train such systems and how feature engineering has been used to get the most out of said data.

2. We examine the learning methods used over time and explain the recent rise of Transformer-based techniques.

3. We give an overview and interpretation of previous results in terms of F_1-scores on different punctuation marks, datasets and learning techniques.

2.1.1 Punctuation Data & Feature Engineering

The earliest systems in the sentence segmentation and punctuation annotation domain (Palmer, 1994; Beeferman et al., 1998) used datasets of written or read rather than spoken language, such as the Wall Street Journal corpus (Paul and Baker, 1992) or Brown corpus (Francis and Kucera, 1979). N-grams were successfully used by these early systems (Beeferman et al., 1998), while auxiliary features such as POS tags were

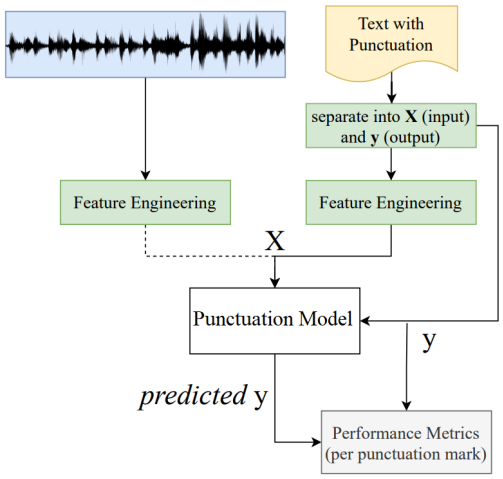


Figure 2.1: Punctuation annotation is learned using punctuated texts which are split into X (no punctuation) and y (punctuation target).

shown to improve performance as well (Palmer, 1994). Earlier work found commas to be easier to predict than full stops as they require less lexical context (Beeferman et al., 1998). In contrast, it was later found that in the HUB-4 Broadcast News Corpus (BN) (Fiscus et al., 1998), commas are harder to predict than full stops, in part due to weak human agreement on the correct placement of commas (Christensen et al., 2001; Batista et al., 2008). Experiments with features derived from acoustic data showed a significant improvement in predicting full stops when using pause durations and a modest improvement when including phone durations or pitch (Christensen et al., 2001). Recent efforts in feature extraction use pre-trained word and speech vectors to great effect (Che et al., 2016; Żelasko et al., 2018; Yi and Tao, 2019), but other works indicate there is a trade-off between the wider availability of text than speech features and the expressivity of speech features, leading to purely lexical models outperforming acoustic ones in certain settings due to more available training data (Klejch et al., 2017). With the rise of multi-task learning (Crawshaw, 2020), Part-of-speech (POS) tags have seen renewed use (Yi et al., 2020), but as an additional output target rather than an additional input feature. By training on both tasks, the model improves on the punctuation annotation task by benefiting from the information learned for the POS tagging task. The same principle has been applied using disfluency detection as well (Chen et al., 2020). Most recent work focuses on predicting full stop, comma and question punctuation marks, which we assume is motivated by their distinct functions in language. Less common punctuation marks contained in datasets are either discarded (Żelasko et al., 2018) or mapped to one of the common three classes. These

DATASET	TOKENS	FULL STOP	COMMA	QUESTION MARK
WSJ	51,023	4.59%	5.98%	0.04%
BN	35,710	3.5%	5.1%	0.29%
IWSLT11	17,207	5.37%	6.36%	0.48%
MGB	92,622	7.63%	4.77%	1.67%

Table 2.1: Distribution of the three most commonly reported punctuation marks across the most widely used corporas' validation sets as reported by previous work.

punctuation marks include exclamation mark, parenthesis, dash, colon, semicolon and three dots and their mapping to one of the three punctuation classes most widely used is sometimes ambiguous, as for example, three dots could be either mapped to comma or full stop (Gravano et al., 2009).

More recent work also focuses on newer datasets, with the most common one being the dataset introduced at the International Workshop on Spoken Language Translation 2011 (IWSLT11), which consists of TED Talk[1] transcripts. TED talks, while delivered in a spoken form, are scripted in advance and rehearsed and are monologues rather than conversations. The Multi-Genre Broadcast (MGB) dataset, which consists of a wide range of TV broadcasts, has also been used, and contains more spontaneous speech and dialogues (Bell et al., 2015). The WSJ and BN datasets used in the past contain written and spoken news, respectively. The distribution of punctuation marks across datasets is shown in Table 2.1. The spoken nature of BN could explain the slight increase in question marks over WSJ. The IWSLT11 and MGB corpora are more informal and span more genres, which could explain the more common occurrence of question marks and full stops. Human annotators do not always agree on punctuation (Batista et al., 2008; Boháč et al., 2017), which could also play a role in these differences. In related work, all of these datasets are processed in a similar fashion. First, the text is converted to a single, unsegmented transcript to avoid giving the model segmentation information that would not be present at inference time (Che et al., 2016). Second, punctuation data is removed from the input (X) while the desired output (y) contains the removed punctuation. There are different ways to model y (see Section 2.3). While we have now covered feature engineering, datasets, and how the initial text is processed for punctuation annotation, we explain the learning methods that aim to utilise these datasets and

[1] `https://www.ted.com`

features in the next section.

2.1.2 Learning Methods

Early neural-network-based approaches on the sentence segmentation task showed promising results (Palmer, 1994), but early systems recovering punctuation models relied mostly on statistical language models such as Hidden Markov Models (HMMs) (Beeferman et al., 1998; Chen, 1999; Christensen et al., 2001; Briscoe and Carroll, 2002). These models did not perform well at modeling long-range dependencies required for end-of-sentence punctuation marks such as question mark or full stop (Beeferman et al., 1998). This is due to these models utilising the Markov assumption which limits the dependence of each state to its immediate predecessor. Their purely statistical nature also made it difficult to perform well on unseen sequences, even if they were semantically similar to sequences the model was trained on. A maximum entropy approach showed promising results as well (Huang and Zweig, 2002). Dynamic conditional random fields (CRFs) improved further on HMMs, and performed better at capturing said long-range dependencies (Lu and Ng, 2010; Wang et al., 2012). Further work explored deep neural networks (Tilk and Alumäe, 2016; Klejch et al., 2017) in the form of Long short-term memory neural networks (LSTMs) (Hochreiter and Schmidhuber, 1997). Experiments on knowledge distillation showed that a student model can leverage information from a teacher ensemble in the punctuation annotation domain (Yi et al., 2017). Recent work has shown promising results using attention-based models (Yi and Tao, 2019; Sunkara et al., 2020). While pre-trained Transformer models such as BERT (Devlin et al., 2018) often performed better than models trained from scratch, recent work has introduced novel ways to combine lexical and acoustic features without forced alignment, such as sub-word attention models (Sunkara et al., 2020). It has also been shown that jointly predicting POS tags and punctuation leads to improvements (Yi et al., 2020). Next, we contrast and compare recent work with respect to different punctuation marks, datasets, input features and learning methods.

2.1.3 Evaluating Punctuation Prediction Models

While the most frequently used evaluation metric for punctuation annotation is the F_1 score, the slot error rate (SER) (Makhoul et al., 2000) can be used as well. As it is more common to report F_1 score on a per-punctuation mark basis, we use the F_1 scores reported by previous work in this section. Due to the majority class being *no*

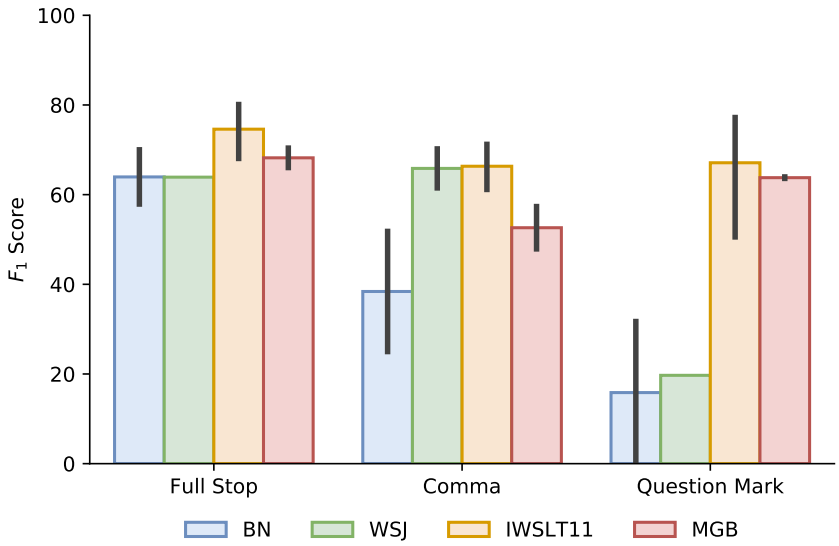


Figure 2.2: The average F_1 scores achieved in previous work per dataset.

punctuation, the recall and precision scores for punctuation annotation are computed over punctuation marks only:

$$\text{precision} = \frac{\text{\# of correctly predicted punctuation marks}}{\text{\# of predicted punctuation marks}}$$

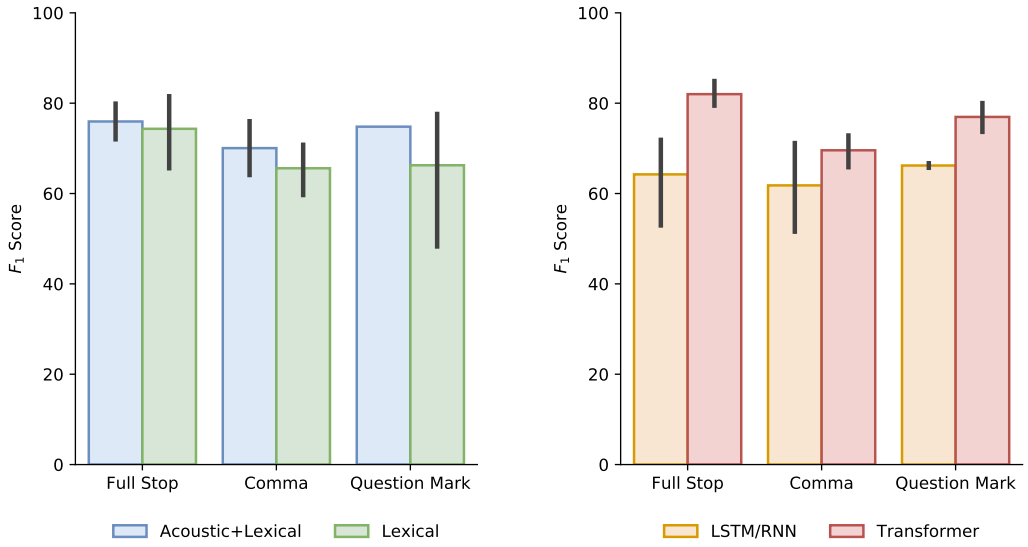
$$\text{recall} = \frac{\text{\# of correctly predicted punctuation marks}}{\text{\# of punctuation marks in reference}}$$

Using these recall and precision values, we can then compute the F_1 score. For the remainder of this work, we report F_1 scores computed in the following way.

$$F_1 = \frac{2 * \text{recall} * \text{precision}}{\text{recall} + \text{precision}}$$

2.1.4 Overview of Punctuation Annotation

As shown in Figure 2.2, periods are the only punctuation marks with similar scores across datasets. We attribute this to the combination of their relatively high occurrence (see Table 2.1) and little ambiguity. Question marks get very low scores in the BN and WSJ corpora which were used with purely statistical methods which made it difficult to capture the long-range dependencies needed to distinguish question marks from periods (Christensen et al., 2001; Gravano et al., 2009). Commas seem to be easier to recover on the WSJ corpus than the BN one, which we attribute to the spoken nature of the BN corpus. In general the lower scores on BN and WSJ, as opposed to IWSLT11

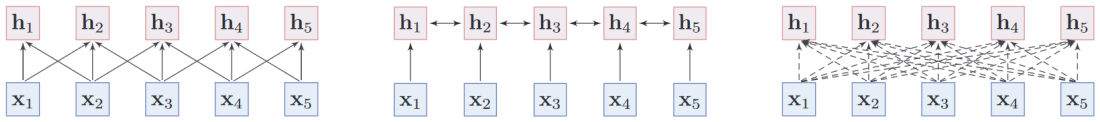


(a) comparison of acoustic & lexical features (b) comparison of LSTM/RNN & Transformers

Figure 2.3: The average F_1 scores achieved on the IWSLT11 dataset reference transcriptions, grouped by input features and model architectures.

and MGB, should not be attributed to the datasets being more difficult but rather shows the weaknesses of the purely statistical models used on these earlier datasets. The results on the IWSLT11 and MGB corpus are more indicative of modern deep-learning-based techniques and show an important trend: the spoken nature of both the IWSLT11 and MGB corpus make commas the hardest symbols to predict, while question marks, although much less common, are close to periods in their F_1 score. Previous work has shown that commas are the most ambiguous punctuation mark, with its ambiguity increasing further when utterances are unscripted (Boháč et al., 2017). Intuitively, the MGB corpus should lead to lower scores overall due to its more spontaneous and unscripted nature, and the data seems to confirm this. However, we caution to draw this conclusion, as state-of-the-art Transformer models have been used on the IWSLT11 dataset but not, to our knowledge, on the MGB dataset.

As these differences between datasets are quite significant, and most recent work makes use of the IWSLT11 dataset, we conduct a more detailed analysis solely on papers predicting punctuation in the reference transcripts of the IWSLT11 dataset. While pre-training and additional data used varies across these works, this still allows insight into some general trends. First, we group models by their use of solely lexical or acoustic and lexical information, which reveals that models incorporating acoustic information generally perform better than ones without. The aforementioned drawback



(a) Convolutional Model (b) Recurrent Model (c) Fully-Connected Self-Attention Model

Figure 2.4: Comparison of neural contextual encoders by Qiu et al. (2020).

of less available training data when using acoustic information does not apply to one of these models, as Yi and Tao (2019) combine a pre-trained word embedding with pre-trained speech embeddings. This acoustic information comes in different forms as well: Tilk and Alumäe (2016) use pause durations alone, while Yi and Tao (2019) use Speech2Vec (Chung and Glass, 2018) to encode the original acoustic information into a vector. When comparing recurrent architectures such as LSTM or Recurrent Neural Networks (RNN) with Transformers, a very clear trend emerges which shows Transformers to clearly outperform recurrent models on every punctuation mark except the comma. It is possible that Transformers outclass RNNs when it comes to long-range dependencies as the maximum path lengths between any two inputs is $\mathbf{O}(1)$ rather than $\mathbf{O}(n)$ due to self-attention (Vaswani et al., 2017). Other reasons could be the availability of pre-trained Transformer models for NLP and their scalability, allowing for deeper models (Qiu et al., 2020). Comma prediction depends less on long-range context than end-of-sentence punctuation marks (Beeferman et al., 1998), which we hypothesise explains the similarity of recurrent and Transformer architectures in this case. Overall, Transformers are state-of-the-art performers in the punctuation domain, and their architecture and functionality is explained next.

2.2 Pre-trained Transformer Models in NLP

Transformers, which achieve state-of-the-art performance in most tasks in the NLP domain, scale well with training data and model size (Wolf et al., 2019). In this section, we briefly outline the Transformer architecture and how and why Transformers are pre-trained. We then describe how said models are fine-tuned for sequence tagging and classification. Finally, we outline possible ways to speed up these models at inference time and to reduce their size.

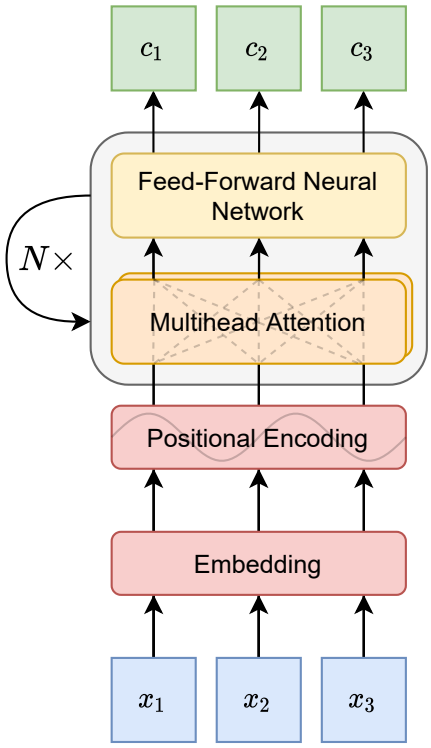


Figure 2.5: Architecture of a Transformer Encoder

2.2.1 Architecture of a Transformer Encoder

The central concept of Transformers is self-attention. As shown in Figure 2.4 (Qiu et al., 2020), this self-attention allows a model to take all elements of the sequence into account, while also processing them at the same time. A Transformer encoder transforms a sequence of input tokens x_i to contextualised word embeddings c_i using the following steps (Vaswani et al., 2017).

1. Create a d-dimensional embedding for each input token.

2. As the Transformer architecture processes all inputs simultaneously, position information is not inherently available to the model. To circumvent this, the sin and cos functions are used to generate values which differ at each input position, which are then added to each embedding based on its position. These slight variations in value based on position allow the Transformer to learn from positional information without processing inputs in sequence.

3. In a process called Dot-Product Self-Attention, create a key, value and query vector from each input, and transform each using a learned weight matrix. For each input, create one output o_i as the weighed sum of all values based on the

key and value dot product as follows:

$$o_i = \sum_{j=1}^{N} \frac{\text{softmax}(k_j \cdot q_i)}{\sqrt{d}} \times v_j$$

Note that it has been found that normalising the weighed sum by dividing by $\sqrt{d}$ leads to more Table training (Vaswani et al., 2017). When this process is repeated multiple times with independent weight matrices, we speak of Multihead Attention. The outputs of these multiple heads are concatenated in this case.

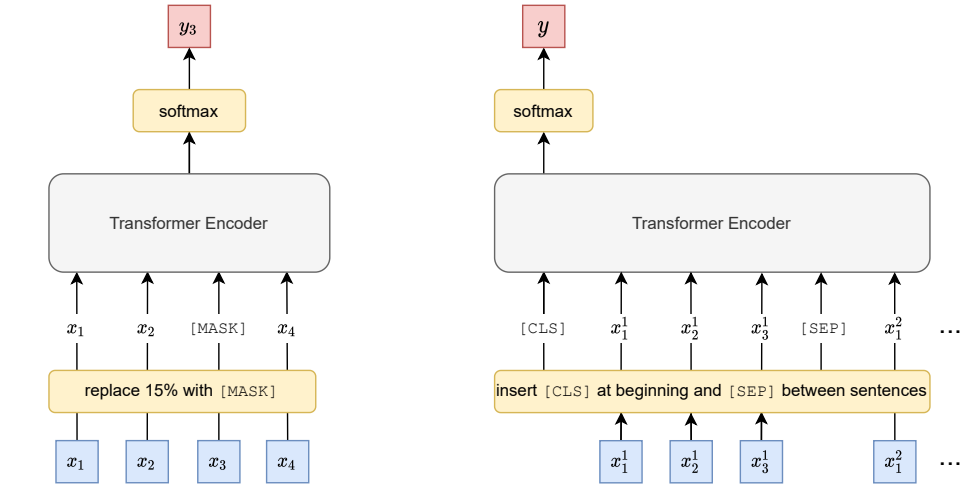
4. A residual, which is a connection partially "skipping" the network, is added to said output to avoid vanishing gradient (He et al., 2015). The vector is then passed through a feed-forward neural network. As shown in Figure 2.5, this process can be repeated N times, with popular architectures ranging between 6 and 24 of these blocks (Qiu et al., 2020).

5. The resulting context vectors c_i can be passed to a decoder for sequence-to-sequence tasks, or to a so-called *head* for tasks with fixed-length target values, such as classification, tagging or regression.

Next, we explain how the encoder is used in tandem with these heads to create pre-trained Transformer models for NLP.

2.2.2 Large, Pre-trained Models

The lack of large amounts of task-specific, annotated data has lead to the rise in a range of pre-trained Transformer models. These models are trained using a range of pre-training objectives on large, unlabeled text datasets (Qiu et al., 2020). One of the most successful and widely-used models is BERT (Devlin et al., 2018). BERT is pre-trained using masked token prediction, which randomly selects tokens in the input sequence and replaces them with a `[MASK]` token. The model then has to fill in these gaps as in the Cloze task (Taylor, 1953). A second task which is used for pre-training BERT is next sentence prediction (NSP), in which the model has to solve the binary classification task of predicting if two sentences, separated by a special `[NSP]` token, follow each other or are randomly chosen. The goal is not to excel at these two tasks, but to train a model which produces hidden representations which can be used well for a variety of down-stream tasks. A variant of BERT called RoBERTa (Liu et al., 2019) showed improved performance by computing the mask positions for MLM



(a) Masked Language Modeling (b) Next Sentence Prediction

Figure 2.6: The two pre-training tasks used in BERT.

anew for each epoch rather than once before training. They also remove the NSP task and tune batch size and other hyper-parameters. RoBERTa outperforms BERT on a diverse set of downstream tasks (Liu et al., 2019). Another approach instead replaces a percentage of tokens with similar ones and trains the model as a discriminator (Clark et al., 2020). Whichever set of pre-training tasks is chosen, their aim is to train a model which produces a contextualised embedding for each input token for later use in downstream tasks. These embeddings could be used in a separate model, in the same way as typically used word embeddings such as GloVe (Pennington et al., 2014) or Word2Vec (Mikolov et al., 2013). However, usually the whole model is adapted to a new task in a process called fine-tuning.

2.2.3 Fine-tuning for Punctuation Annotation

When adapting a pre-trained Transformer to a downstream task, a task-specific *head* with randomly initialised weights is added after the pre-trained Transformer encoder. Training the architecture with this task-specific head can yield good results even when training on few training samples (Howard and Ruder, 2018; Wolf et al., 2019). Different tasks require different heads, but for punctuation annotation, the majority of previous work use either one or two linear layers (Chen et al., 2020; Alam et al., 2020) or a bi-directional Long Short-Term Memory (Bi-LSTM) (Yi et al., 2020). CRFs in conjunction with RoBERTa were found to not improve performance by Alam et al. (2020). It is also possible to add a decoder component and treat the problem as a

sequence to sequence task (Yi and Tao, 2019).

2.2.4 Inference Speed and Model Size

A drawback of the Transformer architecture is that the runtime and memory requirements of dot-product attention scales quadratically with sequence length ($O(n^2)$) (Tay et al., 2020). A range of replacements of this expensive operation have been proposed such as Reformer (Kitaev et al., 2020) and Linformer (Wang et al., 2018), which reduce this to $O(n \log n)$ and $O(n)$, respectively. However, to the best of our knowledge, no large-scale pre-trained models using these architectures are publicly available at the time of writing, negating the benefit of pre-trained Transformer models. Another way to improve inference speeds is to reduce the length of the input, which we explore in Section 4.4.

State-of-the-art punctuation annotation models rely on Transformer models which are large in size. The best results achieved on the IWSLT11 dataset reference transcriptions we are aware of make use of RoBERTa-large (Alam et al., 2020), which has 355M parameters. When each weight is of type `float32`, this leads to a model size of 1.4GB. Recent work has shown that during training, Transformers benefit from this large number of weights and converge faster than when using smaller models (Li et al., 2020). Li et al. (2020) therefore suggest to train large Transformer models first, and to then quantize and/or prune the model weights. By compressing a model this way and then re-training on a subset of the original data, model size can be reduced to a fraction of its original size, while maintaining comparable performance (Han et al., 2016). To the best of our knowledge, this has not been explored in the punctuation annotation domain.

2.3 Ways to Model Punctuation Annotation

As discussed in Section 2.1.2, early punctuation annotation systems mainly use statistical methods, such as Finite-state Machines (FSMs) in conjunction with Viterbi Decoding, to recover punctuation. These approaches found valuable information still useful today such as the strong predictive power of pause durations (Christensen et al., 2001) and the limited context required for predicting comma (Beeferman et al., 1998), but the statistical models themselves were replaced by a variety of neural network architectures. Until recently, the main contenders among these were recurrent (Tilk and

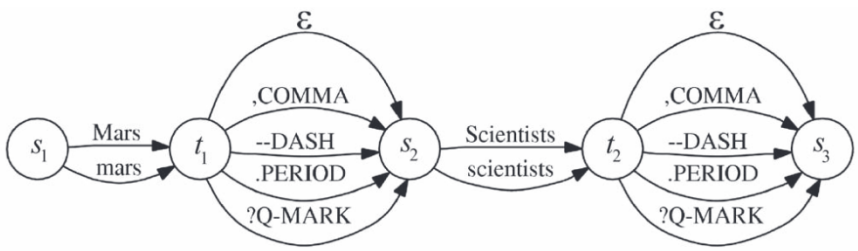


Figure 2.7: Punctuation- and capitalization-recovering finite-state machine by Gravano et al. (2009).

Alumäe, 2016; Klejch et al., 2016; Yi et al., 2017), and less frequently, convolutional (Che et al., 2016) neural networks. In recent years, the trend of the Transformer architecture achieving state-of-the-art results in many NLP domains (Qiu et al., 2020) has extended to punctuation annotation as well (Yi and Tao, 2019; Chen et al., 2020; Alam et al., 2020). While statistical models model punctuation as a probability distribution over possible events occurring between words, there are three other ways to model punctuation annotation which have emerged alongside learning-based models in NLP:

(a) *Tagging:* For models which create one state for each element in the input sequence, such as RNNs or Transformers, a final tagging layer can predict the punctuation mark associated with each such element. When using this approach with the Transformer architecture, punctuation marks predicted earlier in the sequence are not taken into account for later predictions. This limitation can be overcome by using an RNN as the tagging head as described in Section 2.2.3. Some previous work successfully uses Conditional Random Fields (CRFs) as well (Yi et al., 2020), while the work reporting the best results on the IWSLT11 dataset as of the time of writing finds no improvement from CRFs (Alam et al., 2020).

(b) *Machine Translation:* The hidden state(s) produced by a neural network encoder can also be fed to a decoder, which then outputs a sequence of punctuation marks. This approach can make use of previously predicted punctuation marks, but requires the model to learn to output the same number of marks as are words in the input.

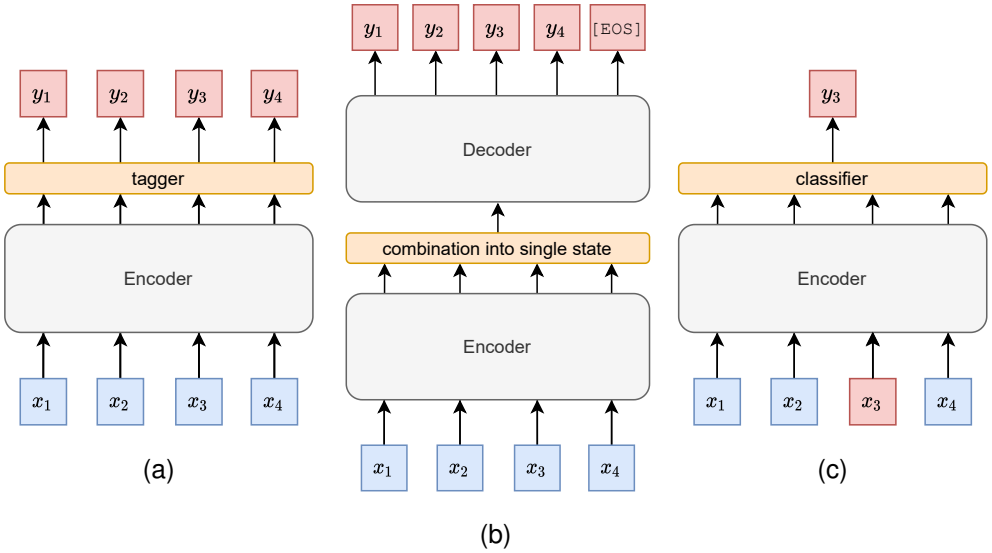


Figure 2.8: Three possible ways to model punctuation using neural network encoders are (a) tagging each element based on its contextualised embedding, (b) treating the task as a translation problem and creating punctuation using a decoder (c) classifying a single element in the input sequence.

Approach	**Paper**
Tagging	Tilk and Alumäe (2016); Yi et al. (2017); Chen et al. (2020) Yi et al. (2020); Alam et al. (2020)
Machine Translation	Klejch et al. (2016, 2017); Yi and Tao (2019)
Classification	Che et al. (2016)

Table 2.2: Differing approaches for modeling punctuation annotation in previous work.

(c) *Classification:* A model can also be trained to predict one punctuation mark per input. The position of the punctuation mark can either be static or given to the model as an additional input. The drawback of this approach is that one inference step is required for each word in the input sequence.

All of the above have been successfully used for punctuation annotation, as shown in Table 2.2. While there are more examples for tagging and machine translation, to the best of our knowledge, Che et al. (2016) present the only approach using classification. We reason that this is due to the increased resources required when doing inference for each word rather than being able to do inference on a full sequence. In the next section, we show how for *streamed* punctuation annotation, this classification approach can be advantageous, and present the *Streamed Classification Punctuation Transformer*.

Chapter 3

Streamed Classification Punctuation Transformer

We now outline our proposed architecture for predicting punctuation in a streamed setting, such as at last step of the ASR pipeline. The desiderata for this system are (1) ability to use in a streamed setting with limited lookahead (2) near state-of-the-art performance (3) real-time inference speed.

3.1 Masked Punctuation Prediction

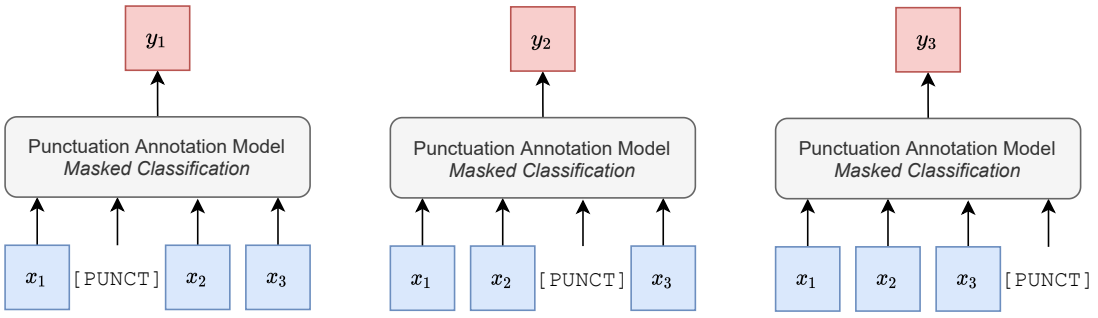


Figure 3.1: Punctuation annotation as a classification task using a `[PUNCT]` token.

In real-time settings, there is a *lack of right-side context*: an ASR streams words to the punctuation system and punctuation is added continuously. The less right-side context a model needs, the earlier can a punctuation mark be inserted into the final output.

However, as discussed in Section 2.3, most recent punctuation annotation systems model the problem as a tagging or sequence-to-sequence task and use deep learning models. For these models, the training objective is loss minimization. When modeled

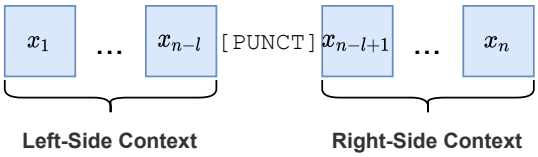


Figure 3.2: Left- and right-side context in masked punctuation prediction.

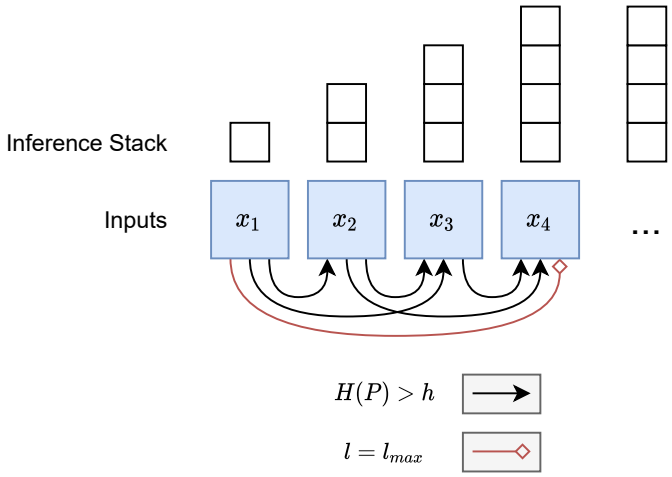


Figure 3.3: Inferences stack for $l_{max} = 3$, assuming $H(P) > h$ at each time step.

as a tagging or sequence-to-sequence task however, this loss term includes equal parts of each output, regardless of its position within the sequence. We argue that a model facing little right-side context will perform better when trained exclusively on samples with little right-side context, and propose Masked Punctuation Prediction for this purpose. Inspired by the masked language modeling (MLM) and next sentence prediction (NSP) tasks used in pretraining (Liu et al., 2019), we insert a special `[PUNCT]` token into each training sample. The model is then tasked to predict the punctuation present at the location of this token using a classification head. As shown in Figure 3.1, this requires one sample per punctuation mark. In a streamed setting, this is not a drawback however, as inference has to be run at the arrival of every new word in any case. When trained with a fixed-length right-side context, this token would not be necessary, as the model could learn the position of the punctuation over time. If we use varying lookahead however, this token is needed, as it encodes the punctuation position, and in turn the length of the current lookahead. Next, we describe this varying lookahead and how it can be used in practice.

3.2 Varying Lookahead and Decoding

To predict punctuation in a streamed setting, we need a way to progressively feed the model more right-side context should it fail to predict punctuation with the context it is given initially, and make it robust to predicting sequences with varying lengths of this context.

Training

Using the `[PUNCT]` token described above, this can be achieved when training the model: We set the minimum and maximum lookahead (l_{min} and l_{max}), and then insert the `[PUNCT]` token at $n-l$ for each sample, where n is the sequence length and l is the lookahead. The lookahead can be cycled through or drawn randomly from $[l_{min}, l_{max}]$.

Inference

For inference, a decoding strategy utilising varying lookahead is needed, which we propose in Algorithm 1. The first question that presents itself is how we decide if the system is predicting punctuation with reasonable confidence, or if more context is needed. We solve this by computing the Shannon-Entropy (Shannon, 1948) H over the set of probabilities $p_i, ..., p_k \in P$ assigned to each of the k punctuation marks after the softmax step.

$$H(P) = H(p_i, ..., p_k) = -\sum_{i=1}^{k} p_i \log_2 p_i$$

This value can be understood as the uncertainty of the model, and will be lower when the model is more certain of a prediction, being 0 when one probability is 1 and all others are 0. Given the four possible outcomes of comma, period, question mark and no punctuation, the maximum value is reached when all probabilities are $\frac{1}{4}$ which corresponds with $H(P) = 2$. For decoding, we set an entropy threshold h and wait for more right-side context and repeat inference if the computed entropy $H(P) > h$. This is repeated until $H(P) \leq h$ or l_{max} is reached. As words are streamed into the system, we potentially do inference on multiple punctuation positions at the same time step, with the maximum number of inferences conducted at the same time being $l_{max} - l_{min} + 1$, as shown in Figure 3.3.

Algorithm 1: Entropy Threshold Decoding

Data: N = `List`(maxsize: $l_{max}+1$), L = `List`

input : *token*, h

output: *punctuations*, *positions*

punctuations ←`List`, *positions* ←`List`

$L \leftarrow L \cup token$

$P \leftarrow probabilities(L \cup$ `[PUNCT]`$)$

if $H(P) > h$ **then**

  $N \leftarrow N \cup 0$

else

  $punctuations \leftarrow punctuations \cup \text{argmax}(P)$

  $positions \leftarrow positions \cup 0$

end

for $i \leftarrow 1$ **to** $|N|$ **do**

  $n \leftarrow N_i$

  $P \leftarrow probabilities(\{L_i\}_{i=0}^{|L|-n} \cup$ `[PUNCT]` $\cup \{L_i\}_{i=|L|-n+1}^{|L|})$

  if $H(P) > h$ **then**

    $N_i \leftarrow n+1$

  else

    $punctuations \leftarrow punctuations \cup \text{argmax}(P)$

    $positions \leftarrow positions \cup -n$

    $N \leftarrow N \setminus n$

  end

end

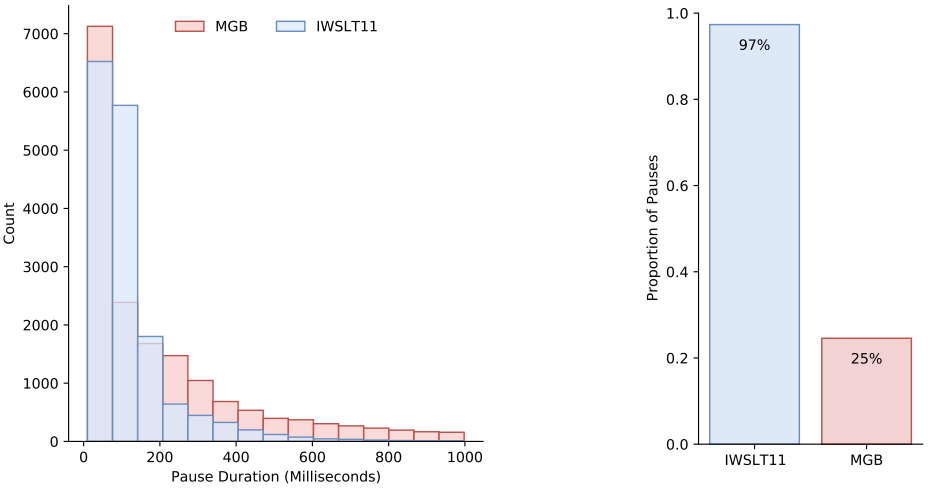
3.3 Adding Pause Tokens

As described in Section 2.1.1, pause features have been shown to be strong indicators of punctuation (Christensen et al., 2001). Given our setup of a pre-trained Transformer with a classification head (see Section 3.1), we see two ways to make these features available to the model:

1. Concatenate pause durations following each word to the context vectors described in Section 2.2.1 before passing them to the classification head.

2. Add a `[PAUSE]` token after each word followed by a pause above a certain threshold.

While 1) can include information on all pauses, rather than exclusively ones above



(a) Distribution of pauses $\geq 10ms$ (b) Word transitions with pauses $\geq 10ms$

Figure 3.4: Pause statistics of the MGB and IWSLT11 datasets.

a certain threshold, only the classification head can learn from these features, while the Transformer encoder cannot. 2) on the other hand cannot encode all pause information, but enables the full architecture to learn from the pause features. For these reasons, our approach uses `[PAUSE]` tokens. For a streamed punctuation system, pause durations can either be a part of the output of said system or can be approximated by measuring the time between words streamed. However for training, transcripts for both the IWSLT11 and MGB datasets do not contain timing information. On the other hand, ASR output for both systems including this information is available. Previous work aligns ASR outputs and transcripts to add punctuation to the ASR output (Yi and Tao, 2019). Inspired by this approach we first add `[PAUSE]` tokens to the ASR output using the available timing information, and then align this modified ASR output with the original transcripts using the Needleman and Wunsch (1970) algorithm. We publish our code to load the publicly available IWSLT11 dataset with and without pause durations.[1] One drawback of this method is that for the IWSLT11 dataset, only the validation and test splits of the data come with ASR transcripts. To combat this we can a) use training data without pause durations first b) use MGB training data first and then finetune. For both approaches, we finetune the model using the validation set. a) and b) can be used in combination as well. We evaluate these options empirically in Section 4.5. When using the MGB dataset for this purpose, it becomes worthwhile to investigate the pause duration similarity in both datasets. As shown in Figure 3.4, both

[1] `https://github.com/MiniXC/punctuation-iwslt2011`

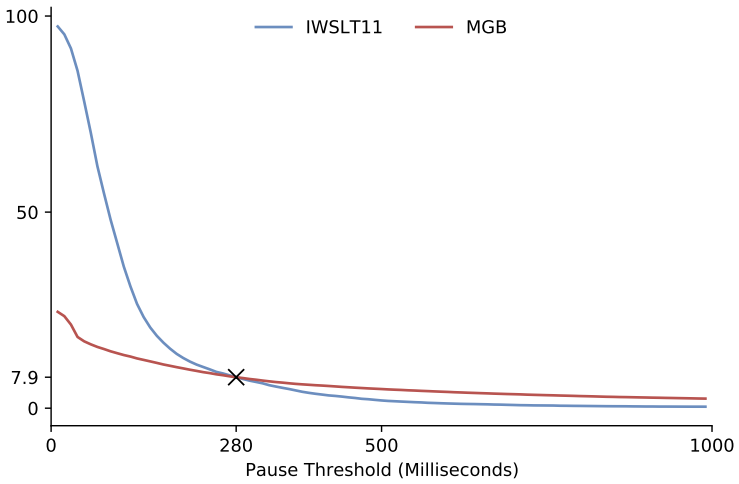


Figure 3.5: The proportion of pauses between words based on threshold t_p in ms.

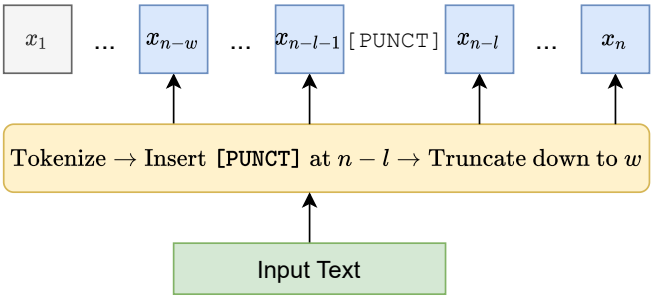


Figure 3.6: Tokenization, [PUNCT] token insertion and truncation.

datasets follow a similar exponential decay when considering all pauses $\geq 10ms$, although the MGB dataset has a higher proportion of pauses in the range of $[0ms, 75ms]$ while the proportion of pauses in IWSLT11 is higher in the interval of $[75ms, 150ms]$. We also observe a big discrepancy between the occurrence of pauses $\geq 10ms$, with 97% of word transitions being accompanied by at least a short pause in the IWSLT11 dataset, while this is the case for just 25% of transitions between words in the MGB dataset. We expect this to be the case due to the spontaneous vs. scripted nature of the MGB and IWSLT11 datasets (see Section 2.1.1). The dataset ASR transcriptions were possibly generated using differing systems, which could cause this difference as well. To combat this imbalance when training on one of the datasets and evaluating on the other, we can either take speaking rate into account or find the threshold that leads to the most similar statistics. In this work, we use the latter approach and find that for $t_p \approx 280ms$, both datasets have a pause proportion of 7.9%, as shown in Figure 3.5. We evaluate this and other threshold values in Section 4.5.

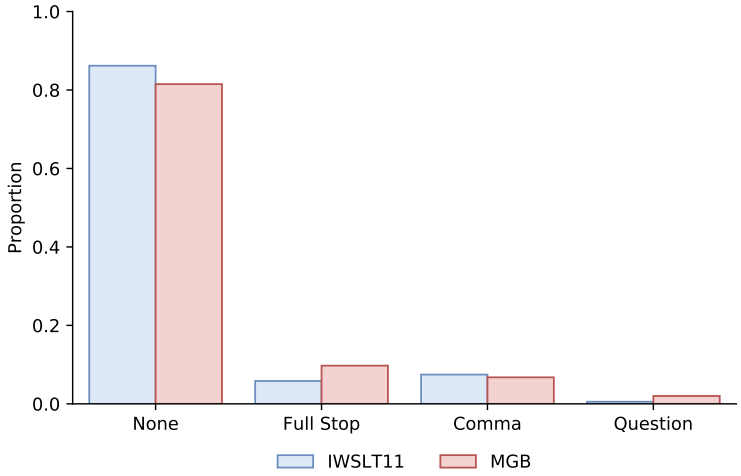


Figure 3.7: Class distribution of the MGB and IWSLT11 validation datasets.

3.4 Input Truncation for Faster Training and Inference

As described in Section 2.2.4, Transformers use dot-product self-attention, which has a time and memory complexity of $O(n^2)$. While there are architectures which reduce this to $O(n \log n)$ or $O(n)$, no large pre-trained models of such variants are available at the time of writing, negating the benefits we outline in Section 2.2.2. As in many other punctuation annotation research (Tilk and Alumäe, 2016; Yi and Tao, 2019; Alam et al., 2020), we generate samples by using a sliding window on a transcript with all prior segmentation removed. Due to our approach of only predicting one punctuation mark at a time however, we are able to trim left-side context until we notice a degradation in performance. This is done after tokenization, as some Transformer architectures rely on a one-to-many mapping between words and embeddings. We introduce a parameter w which determines the length of the window after truncation. We empirically evaluate different values of w in terms of speedup and model prediction performance.

3.5 Punctuation Class Imbalance

When treating punctuation annotation as a classification problem, we encounter the class imbalance shown in Figure 3.7. Models trained on such datasets with imbalanced classes can develop a prediction bias for the majority class (Leevy et al., 2018). While there are many approaches for preventing this, *under-sampling*, where part of the majority class data is discarded and *over-sampling*, where part of the minority classes is repeated are commonly used (Leevy et al., 2018). When over-sampling, the

repeated data can be augmented as well, by replacing words or characters in the input with similar ones (Ma, 2018). In this work, we rely on downsampling as a means to train large models efficiently.

Summary

We have introduced a novel way to create a classification punctuation annotation system using a special `[PUNCT]` token. We have proposed a training procedure utilising this approach to train a model specifically on samples with little or no right-side context, to create a model well-suited for the streamed punctuation annotation task. For inference, we have introduced *Entropy Threshold Decoding*, which varies the lookahead needed based on model certainty. We have hypothesised that this will lead to easier samples being predicted early while the model will wait for more context for harder samples. To utilise acoustic features commonly available as a part of ASR output, we have shown that word timing information can be used to infer pause information by adding a `[PAUSE]` token at a threshold t_p. For inference speeds, we have proposed input truncation due to the quadratic complexity of dot-product attention. We also have also reasoned that due to the class imbalance present, downsampling can help train on fewer samples while maintaining comparative performance. Next, we evaluate the techniques described in this chapter and present their best combination as our system.

Chapter 4

Experiments and Results

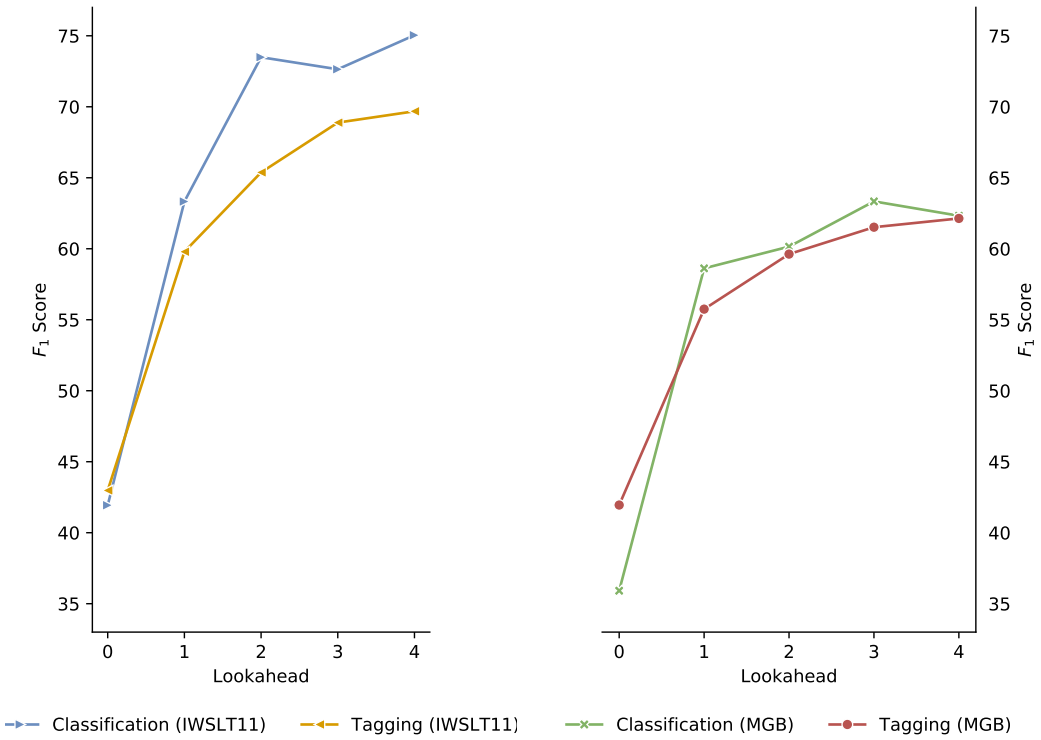
We now put our proposed architecture to the test and empirically evaluate how different methods affect performance on the MGB and IWSLT11 datasets. We make the scripts used for all experiments publicly available at `https://github.com/MiniXC/SAPAUT`.

4.1 IWSLT11 and MGB Dataset & Statistics

DATASET	SAMPLES	FULL STOP	COMMA	QUESTION MARK
MGB_{TRAIN}	2.49M	8.36%	5.72%	1.28%
$\text{MGB}_{TRAIN \leftrightarrow ASR}$	2.04M	8.43%	5.83%	1.25%
MGB_{VALID}	93.0K	9.74%	6.76%	2.01%
$\text{MGB}_{VALID \leftrightarrow ASR}$	76.6K	10.24%	6.86%	1.98%
IWSLT11_{TRAIN}	2.4M	6.13%	6.94%	0.52%
IWSLT11_{VALID}	49.2K	5.82%	7.46%	0.54%
$\text{IWSLT11}_{VALID \leftrightarrow ASR}$	47.4K	6.27%	7.50%	0.54%
IWSLT11_{TEST}	14.3K	6.19%	5.79%	0.35%
$\text{IWSLT11}_{TEST \leftrightarrow ASR}$	13.5K	6.73%	5.97%	0.37%

Table 4.1: The statistics of the MGB and IWSLT dataset splits.

As shown in Table 4.1, the datasets splits aligned with the ASR transcripts (for example $TEST \leftrightarrow ASR$) are slightly smaller due to not all talks appearing in the transcripts. To allow for a fair comparison, we use the aligned splits for all experiments. The IWSLT11_{TRAIN} dataset split does not come with ASR transcripts, and we can therefore only use it without pause information.



(a) IWSLT2011 dataset

(b) MGB dataset

Figure 4.1: The F_1 scores achieved given different lookahead values. Classification outperforms tagging significantly for $l > 0$ in both the MGB and IWSLT11 dataset.

Hyper-Parameters	
Learning Rate (Initial/Maximum/Final)	1e-6/5e-5/1e-7
Learning Rate Schedule	1-cycle (Smith, 2018)
Batch Size	128
Optimizer	AdamW (Loshchilov and Hutter, 2019)
Weight Decay	0.01

Table 4.2: Hyper-parameters used for all experiments.

4.2 Pre-Trained Models by Hugging Face

The Hugging Face Transformers library, offers a range of pre-trained Transformer models in conjunction with differing heads for finetuning (Wolf et al., 2019). As described in Section 2.2.3, based on the downstream task in question, a simple linear layer with $n_{classes}$ output nodes might be used for classification, while recurrent layers can be used for sequence tagging tasks.

TEST	MODEL	FULL STOP			COMMA			QUESTION			OVERALL		
		P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
IWSLT11	TAGGING	**75.0**	78.6	**76.7**	58.8	**64.7**	61.6	**76.9**	78.9	**77.9**	67.3	72.1	69.6
	CLASS.	74.5	**84.9**	79.4	**69.1**	71.5	**70.1**	50.0	**80.0**	61.5	**71.6**	**78.7**	**75.0**
MGB	TAGGING	**65.4**	68.6	67.0	56.1	**52.9**	**54.5**	62.4	**62.4**	**62.4**	61.8	**62.1**	62.0
	CLASS.	63.4	**72.0**	**67.4**	**59.9**	48.8	53.8	**71.0**	55.3	62.2	**63.0**	61.6	**62.3**

Table 4.3: Tagging and classification results for lookahead $l = 4$.

4.3 Baseline using Tagging Approach

As our baseline to compare against, we train a model using the widespread sequence tagging approach (Tilk and Alumäe, 2016; Yi et al., 2017; Chen et al., 2020; Yi et al., 2020; Alam et al., 2020) with a Bi-LSTM head. The pre-trained model used is DISTILROBERTA, which is a ROBERTA variant (Liu et al., 2019) distilled into a smaller model using the approach described by Sanh et al. (2019). In preliminary experiments we find the hyper-parameters shown in Table 4.2 to lead to robust results, and use said parameters for the remainder of experiments. We use the sliding window approach described in 3.4 with a window size of $w = 32$ and evaluate using the F_1 measure described in Section 2.1.3 at different lookaheads.

As shown in Figure 4.1, the classification model outperforms the tagging one for all lookaheads except $l = 0$. We reason that this is due the classification model putting more emphasis on samples with little right-side context, as hypothesised in Section 3.1. The better performance of the tagging model on lookahead $l = 0$ is surprising, but could be due to the tagging model having access to previous predictions using its Bi-LSTM layer, which the classification model has not.

Preliminary experiments showed no significant improvements beyond a lookahead of 4. Therefore, to be able to compare to previous work, which has no limitations on right-side context, we evaluate the per-class performance of models given a lookahead of 4 for this baseline (see Table 4.3) and for the remainder of experiments. We notice that the classification approach yields a significant improvement for the IWSLT11 dataset, while not doing so on the MGB dataset.

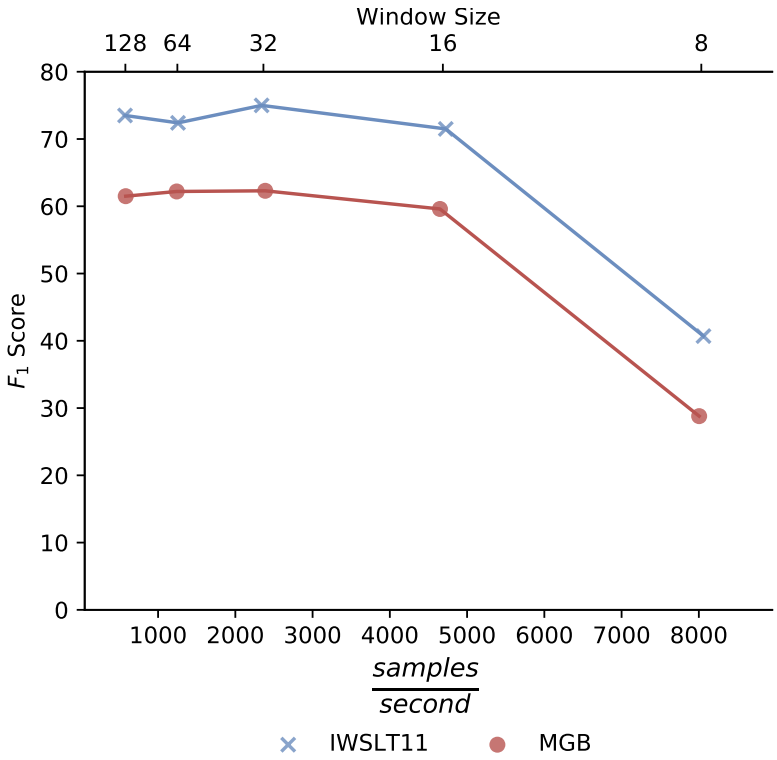


Figure 4.2: Window size effect on speed and performance. Decreasing the window size seems to lead to a linear increase in speed, while F_1 scores decrease when using a window size of 16 or less.

4.4 Truncation Window Sizes

So far, we have mainly focused on *right-side context*, as it is a major limiting factor for streamed punctuation prediction. *Left-side context*, on the other hand, can be limited on purpose to speed up training and inference. Due to the $O(n^2)$ complexity of dot-product attention (see Section 2.3), every halving of the input should result in a quadratic reduction in inference and training time. We therefore truncate the left side of the input as described in Section 3.4 and compare window sizes w of 128, 64, 32, 16 and 8. Instead of the theoretically possible $\times 4$ speedup for each halving we observe a $\approx \times 2$ increase in speed. There also seem to be diminishing returns, as the decrease from 16 to 8 in window size yields the lowest speedup in relative terms. As shown in Figure 4.2, decreasing the window size to below 32 decreases performance on both datasets. We therefore use a window size of 32 for all future experiments.

PAUSE	FULL STOP			COMMA			QUESTION			OVERALL		
	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
NONE	71.0	67.9	69.4	50.9	59.0	54.7	26.6	80.0	40.0	59.3	64.1	61.6
$t_p = 100ms$	74.1	71.1	72.6	54.9	**61.2**	**57.8**	75.0	75.0	**75.0**	63.8	66.3	65.0
$t_p = 150ms$	74.5	70.4	72.4	49.7	57.8	53.4	75.0	75.0	**75.0**	61.8	64.8	63.3
$t_p = 200ms$	74.0	70.4	72.2	49.7	57.8	53.4	50.0	66.6	57.1	60.6	64.5	62.5
$t_p = 250ms$	72.2	74.1	73.2	46.1	51.0	48.4	**80.0**	57.1	66.6	59.5	62.9	61.2
$t_p = 280ms$	**77.1**	75.6	**76.3**	50.5	58.8	54.3	66.6	76.9	71.4	63.0	67.7	65.2
$t_p = 300ms$	73.8	72.9	73.3	**56.2**	59.0	57.5	42.8	60.0	50.0	**64.1**	65.7	64.9
$t_p = 350ms$	70.4	**82.3**	75.9	**56.2**	55.5	55.9	63.6	**87.5**	73.6	63.8	**69.3**	**66.4**
$t_p = 400ms$	74.2	67.9	70.9	46.7	59.3	52.3	60.0	85.7	70.5	59.2	64.5	61.7

Table 4.4: Results of using the different pause thresholds on the IWSLT11 dataset.

PAUSE	FULL STOP			COMMA			QUESTION			OVERALL		
	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
NONE	63.4	72.0	67.4	59.9	48.8	53.8	**71.0**	55.3	62.2	63.0	61.6	62.3
$t_p = 100ms$	62.5	**73.6**	67.6	60.3	45.7	52.0	65.1	58.3	61.5	62.1	61.4	61.7
$t_p = 150ms$	64.0	72.1	**67.8**	61.0	49.9	54.9	62.6	56.2	59.2	63.0	62.2	**62.6**
$t_p = 200ms$	**64.6**	70.8	67.6	60.9	49.2	54.5	68.6	56.6	62.0	**63.8**	61.5	**62.6**
$t_p = 250ms$	63.0	69.3	66.0	58.6	48.6	53.1	67.3	54.3	60.1	62.0	60.0	61.0
$t_p = 280ms$	63.7	69.4	66.4	**63.1**	49.2	55.3	65.7	**59.2**	**62.3**	63.7	60.6	62.1
$t_p = 300ms$	63.8	70.0	66.8	58.8	47.5	52.6	67.6	55.8	61.1	62.5	60.2	61.4
$t_p = 350ms$	61.6	71.9	66.4	59.0	47.7	52.7	67.2	50.8	57.8	61.3	60.9	61.1
$t_p = 400ms$	63.1	71.4	67.0	59.9	**52.1**	**55.7**	68.4	54.3	60.6	62.5	**62.3**	62.4

Table 4.5: Results of using the different pause thresholds on the MGB dataset

4.5 Pause Features at different Thresholds

We now test if the `[PAUSE]` token proposed in Section 3.3 improves performance and which pause threshold t_p is optimal. We test thresholds in 50ms intervals, and additionally test 280ms as it the pause threshold with the same pause proportions between MGB and IWSLT11 datasets (see Section 3.3).

MGB

On the MGB dataset, we find a slight improvements over not using pause tokens at t_p values of 150*ms* and 200*ms*. The biggest impact can be observed for comma prediction, where the 150*ms* and 200*ms* increase performance by 1.1% and 0.7%, respectively (absolute). The partly spontaneous nature of the MGB dataset could cause annotators to mark pauses as commas in the transcripts, while the end of a sentences does not have to be accompanied by a pause. The impact of pause features on performance is much less pronounced on the MGB dataset than the IWSLT11 one. This could be due to less pronounced pauses in the dataset, as the speech in the MGB dataset is more spontaneous than in IWSLT11 one (see Section 2.1.1).

IWSLT11

For this experiment, we train on the IWSLT11$_{VALIDATION}$ dataset and evaluate on IWSLT11$_{TEST}$. The intuitively chosen 280*ms* performs well, only slightly outperformed by 350*ms*. We therefor use 280*ms* for the remainder of experiments. The results also indicate that short pauses are helpful for predicting commas but lead to worse results on full stops. The model without pauses performs worst of all models for the end-of-sequence punctuation marks full stop and question, while performing well on comma. This is expected, as we reason that end-of-sequence punctuation is more likely to be accompanied by a pause than comma. The results for the IWSLT11 datasets are lower than the baseline due to the IWSLT11$_{TRAIN}$ set not having pause durations, which forces us to use the IWSLT11$_{VALIDATION}$ set alone for training.

4.6 Pause Finetuning

To successfully apply the pause features while making use of the baseline trained on a large dataset, we use said baseline as a starting point and finetune on smaller datasets containing pause features. The results of these experiments are shown in Table 4.6.

MGB

While increasing performance on question marks and comma prediction, finetuning the MGB baseline on pause features leads to degradation on full stop prediction, while significantly improving comma prediction, leading to an overall lower result. Utilizing the IWSLT11 data also yields worse results, decreasing performance on comma and

TRAIN	FULL STOP			COMMA			QUESTION			OVERALL		
	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
IWSLT11 Baseline	74.5	84.9	79.4	69.1	71.2	**70.1**	50.0	80.0	61.5	71.6	78.7	75.0
+IWSLT11$_{VALID \leftrightarrow PAUSE}$	**80.8**	**89.1**	**84.8**	66.2	69.3	67.7	**76.9**	76.9	76.9	**73.9**	79.2	**76.5**
+MGB$_{VALID \leftrightarrow PAUSE}$	73.9	**89.1**	80.8	**72.5**	62.1	66.9	73.3	**84.6**	**78.6**	73.4	76.1	74.7
+BOTH	**80.8**	86.5	83.6	65.3	**72.5**	68.7	**76.9**	76.9	76.9	73.1	**79.5**	76.2
MGB Baseline	**63.4**	**72.0**	**67.4**	59.9	48.8	53.8	71.0	55.3	62.2	**63.0**	61.6	**62.3**
+MGB$_{TRAIN \leftrightarrow PAUSE}$	62.3	67.9	65.0	59.5	53.8	**56.5**	66.8	**60.5**	**63.5**	61.7	**61.8**	61.8
+IWSLT11$_{VALID \leftrightarrow PAUSE}$	63.1	59.7	61.4	50.9	**54.6**	52.7	66.2	59.7	62.8	58.3	57.7	58.0
+BOTH	61.4	69.6	65.2	**60.8**	46.7	52.8	**73.8**	54.5	62.7	62.2	59.3	60.7

Table 4.6: Finetuning the model baselines on pause features using differing datasets.

full stop. We hypothesis that a better way to encode pause features would be needed to allow for improvements on the MGB corpus. We also reason that the IWSLT11 dataset does not improve finetuning results on MGB due to its more scripted nature.

IWSLT11

The IWSLT11 dataset shows an inverse result to the MGB one. Full stop prediction is improved significantly, while comma prediction is degraded. The overall best result is achieved when finetuning on the IWSLT validation set alone (1.5%). Finetuning on both MGB and IWSLT11 leads to better comma prediction (1%) while improving overall performance (1.2%) compared to the baseline.

4.7 Entropy Threshold Decoding

Using the algorithm described in Section 3.2, we dynamically vary lookahead based on different entropy thresholds on the IWSLT11 dataset. We start at a threshold of $h = 0.1$ and evaluate the F_1 score of the system at intervals of 0.1 until reaching $h = 1.5$. We report the average lookahead as well to allow for comparison with decoding using a fixed lookahead. As shown in Figure 4.3, we find that entropy threshold decoding yields worse results than fixed-lookahead decoding when including zero-lookahead. We hypothesise that due to the low F_1 score of the model at zero-lookahead (see Figure 4.1), the Shannon-Entropy measure is not an accurate measure of model confidence either. This leads to a disproportionate amount of false predictions at zero-lookahead, and lowers the F_1 score overall. To test this hypothesis, we exclude the zero-lookahead step

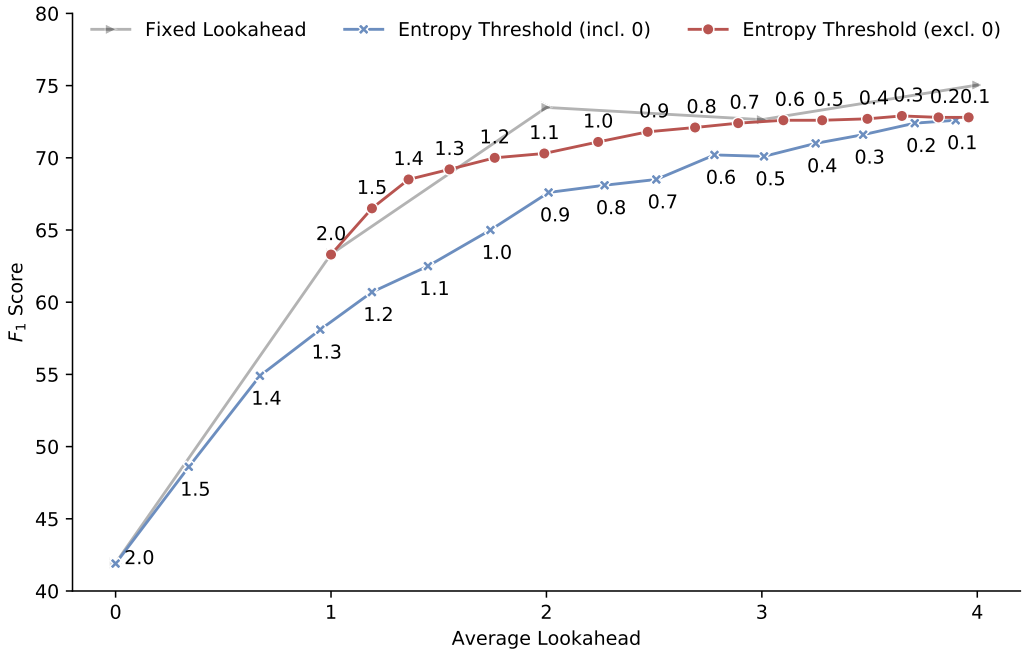


Figure 4.3: Entropy threshold decoding at different values for the threshold h. Each number accompanying a point represents an entropy threshold. As the maximum entropy for four classes is 2.0, Entropy threshold decoding using this value is equivalent to fixed-lookahead decoding with the smallest lookahead.

in our decoding algorithm, and find decoding is now on par with fixed-lookahead decoding. This decoding strategy allows to specify the desired average lookahead, while only slightly decreasing performance. Future work could investigate using lookahead-specific thresholds (e.g. gradually decreasing the entropy threshold h) or training a neural network for this purpose.

4.8 Scaling Up & Comparison to Previous Work

Recent state-of-the-art performance on the IWSLT11 dataset has been shown using the BASE and LARGE variants of ROBERTA, which are twice and quadruple the size of the DISTILROBERTA model used thus far. Li et al. (2020) show evidence for large models outperforming small ones even when using the same computational time and resources. They show that larger models trained on smaller datasets outperform smaller models trained on large ones. To improve our model performance while restricting ourselves to a similar compute budget as for the DISTILROBERTA model, we use downsampling. We remove samples with no punctuation until there are twice the number of `None`

Model	Full Stop			Comma			Question			Overall		
	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
Alam et al. (2020)	**88.6**	89.2	**88.9**	**76.8**	76.6	**76.7**	82.7	**93.5**	**87.8**	**82.6**	83.1	**82.9**
IWSLT11$_{TRAIN}$ (Resampled)	77.4	87.6	82.2	55.0	**87.1**	67.5	87.5	77.8	82.4	63.9	**87.1**	73.7
+IWSLT11$_{TRAIN,2\%}$	76.4	**89.9**	82.6	64.0	74.2	68.7	50.0	80.0	61.5	70.4	82.7	76.0
+IWSLT11$_{VALID,t_p=280ms}$	82.4	88.1	85.1	63.0	78.8	70.0	57.1	80.0	66.7	72.5	83.8	77.7
+BOTH	78.9	89.3	83.8	67.8	76.5	71.9	**88.1**	78.3	82.9	73.5	83.4	78.2
MGB$_{TRAIN}$ (Resampled)	62.3	**81.2**	70.5	53.4	**68.9**	**60.2**	72.9	**64.8**	**68.6**	59.8	**74.9**	66.5
+MGB$_{TRAIN,2\%}$	**66.6**	77.2	**71.5**	**63.9**	55.7	59.5	73.3	58.8	65.2	**66.2**	67.4	**66.8**
+MGB$_{TRAIN,2\%,t_p=280ms}$	64.8	76.5	70.2	61.0	58.3	59.6	**75.5**	62.1	68.1	64.5	68.2	66.3
+BOTH	65.4	73.3	69.2	60.6	56.9	58.7	70.2	62.7	66.2	64.2	66.1	65.2

Table 4.7: Results of the RoBERTa-large model trained on a resampled training set.

samples than the number of samples for the most common punctuation mark. For the MGB dataset, this reduces the number of samples to 33% of their original number. For the IWSLT11 dataset, the number of samples is reduced to 29%. As this shifts the class priors, we also finetune on an unaltered subset of the data (2% of the training data). Table 4.7 shows these results in detail. The IWSLT11 results in said table can be compared with Alam et al. (2020), which achieve the best results on said dataset to the best of our knowledge. Alam et al. (2020) augment the training data to train with more samples, while we downsample for more efficient training.

MGB

When comparing to the best previous work, which utilises a RNN and sequence-to-sequence approach (Klejch et al., 2016), we outperform said model by 2.9% on average for each punctuation class (1.1% full stop; 2.8% comma; 4.9% question mark). In contrast to said work, we do not using any language model data and just 33% of the training data. Pause features do not lead to any significant improvements on the MGB dataset. We reason that this general improvement can be attributed to Transformer architecture we are utilising, coupled with its unsupervised pre-training, following the trend seen in other NLP domains (Qiu et al., 2020).

IWSLT11

Despite training on 29% of the train dataset alone, we achieve an overall F_1-score of 78.2% on the IWSLT11 test dataset, which, to the best of our knowledge, is the highest to date without utilising additional data such as POS-tags (Yi et al., 2020), disfluency data (Lin and Wang, 2020) or data augmentation (Alam et al., 2020).

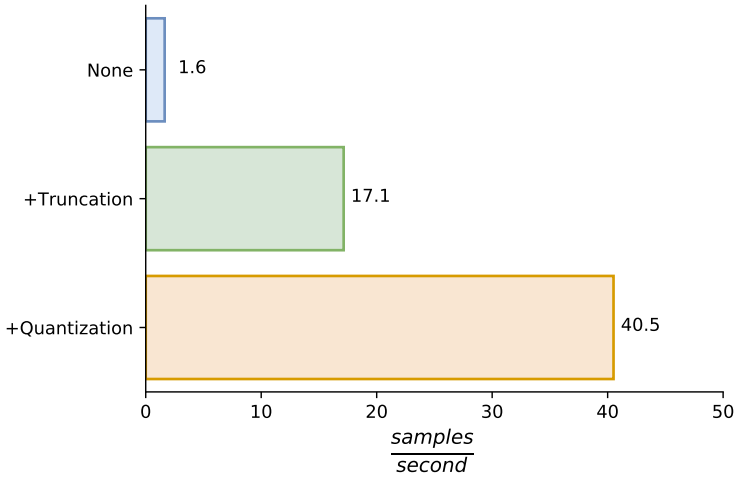


Figure 4.4: Effect of truncation and quantization on inference speed.

4.9 Inference Speed & Quantization

We now evaluate inference speed of the final IWSLT11 model described above.[1] While we observed a linear increase in speed while training (see Section 4.4), for inference, we get closer to the quadratic improvement expected. The theoretical upper bound for the speedup when using $\frac{1}{4}$ of the original window size is $\times 16$. In our inference experiments, we find an actual increase in speed of $\times 10.68$, from $1.6 \frac{samples}{second}$ to $17.1 \frac{samples}{second}$. To further increase model speed and decrease model size, we perform quantization of the model weights, reducing their size from `float32` to `int8`. We do this using standard procedures available in the PyTorch libary[2] (Paszke et al., 2019). This leads to a slight decrease in model performance (-1.3% F_1 for MGB, -2.6% F_1 for IWSLT11). This also leads to a $\times 2.36$ increase in speed to 40.5 $\frac{samples}{second}$ and $\approx \times 4$ compression in model size, reducing model size from 1.4GB to 355MB. We reason that the decrease in performance observed could be offset by the compression and speed gained in certain use cases, for example for use on mobile devices. We reason that this loss in performance could also be offset using quantization-aware training or finetuning following quantization (Han et al., 2016; Jacob et al., 2017).

[1] A single core of a Ryzen 7 2700X processor was used, 6000 inference iterations were averaged.
[2] `https://pytorch.org/tutorials/advanced/static_quantization_tutorial.html`

Chapter 5

Conclusion and Future Work

Instead of building a punctuation prediction system and then adapting it to the streamed ASR use case, we focused directly on streamed punctuation annotation. Most recent punctuation annotation systems using Transformers use a sequence tagging approach. Contrary to this we used a classification approach and presented a novel *masked punctuation prediction* training and inference procedure inspired by the masked language modeling task used when pre-training Transformers (Devlin et al., 2018). We showed our new method outperforms the aforementioned tagging approach when a right-side context of 4 words or less is available, which is desirable for streamed punctuation annotation. To utilise acoustic information while relying on an ASR system's output alone, we encoded timing information commonly provided by ASR systems as an additional feature. We achieved this by adding a special `[PAUSE]` token to the input. On the IWSLT11 dataset, this lead to a significant improvement of 2.2% (absolute) in F_1 score. We conclude that pause tokens work exceedingly well for datasets of a semi-scripted nature, and could be of great use for dictation systems. We see *masked punctuation prediction* and `[PAUSE]` tokens as our most promising contributions and plan to submit a paper exploring this further to *ICASSP 2022*. We used downsampling to efficiently train our final model using $\approx \frac{1}{3}$ of the available training data, while achieving overall F_1 scores within 4.7% (absolute) of the best model we are aware of on the IWSLT11 dataset (Alam et al., 2020). We also found that the advantages of the commonly used approach of fine-tuning pre-trained Transformer models extend to the semi-spontaneous speech present in the MGB dataset, as we are able to outperform the previous best model (Klejch et al., 2016) by 2.9% (absolute) on average for each evaluated punctuation mark. As we used a model pre-trained on an unsupervised language modeling task, we achieved this while utilising 0.03% of the data used to train the pre-

vious best model on the MGB dataset. We showed our classification approach can be combined with truncation, which leads to a $\times 4$ speedup for training and $\times 10$ speedup for inference while not decreasing performance. For use of ASR on low-resource devices, we showed weight quantization yields a further $\times 2.36$ speedup and a reduction in model size by $\times 4$, while leading to a minor decrease in F_1 scores of 2% on average. Future work could improve our system by adding teacher forcing as a mechanism to use past punctuation predictions for future ones. Our naive Entropy Threshold Decoding algorithm could be replaced by a learned method, such as a shallow neural network to predict if more lookahead is needed. Quantization-aware training, in which floating point weights are regularly rounded during training, could help mitigate the loss in performance reported due to quantization. While the improvement in inference speed gained using truncation are significant, we show that truncation starts to decrease model performance at a window width of 16 or less. This limits further improvements when reaching this window size. To further improve models for low-resource applications, efficient Transformer architectures reducing the $\mathrm{O}(n^2)$ complexity of dot-product attention could be investigated. In our experiments on the IWSLT11 dataset, we show pause tokens at varying thresholds affect different punctuation mark performance. To allow the model to distinguish between pauses of different lengths, the pause features utilized in our work could be provided as an input to the final model head, or separate short- and long-pause tokens could be investigated. Finally, our system and the impact of `[PAUSE]` tokens could be evaluated on the ASR results, rather than transcriptions, of the MGB and IWSLT11 datasets.

Bibliography

T. Alam, A. Khan, and F. Alam. Punctuation restoration using transformer models for high-and low-resource languages. In *W-NUT*, 2020.

E. Bakhturina. Nemo, 2019. URL `https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/v0.10.1/nlp/punctuation.html`.

F. Batista, D. Caseiro, N. Mamede, and I. Trancoso. Recovering capitalization and punctuation marks for automatic speech recognition: Case study for Portuguese broadcast news. *Speech Communication*, 2008.

D. Beeferman, A. Berger, and J. Lafferty. Cyberpunc: A lightweight punctuation annotation system for speech. In *ICASSP*, 1998.

P. Bell, M. J. F. Gales, T. Hain, J. Kilgour, P. Lanchantin, X. Liu, A. McParland, S. Renals, O. Saz, M. Wester, and P. C. Woodland. The MGB challenge: Evaluating multi-genre broadcast media recognition. In *ASRU*, 2015.

M. Boháč, M. Rott, and V. Kovář. Text punctuation: An inter-annotator agreement study. In *TSD*, 2017.

T. Briscoe and J. Carroll. Robust accurate statistical annotation of general text. In *LREC*, 2002.

X. Che, C. Wang, H. Yang, and C. Meinel. Punctuation prediction for unsegmented transcript based on word vector. In *LREC*, 2016.

C. Chen. Speech recognition with automatic punctuation. In *Eurospeech*, 1999.

Q. Chen, M. Chen, B. Li, and W. Wang. Controllable time-delay transformer for real-time punctuation prediction and disfluency detection. In *ICASSP*, 2020.

H. Christensen, Y. Gotoh, and S. Renals. Punctuation annotation using statistical prosody models. In *ISCA Tutorial and Research Workshop (ITRW) on Prosody in Speech Recognition and Understanding*, 2001.

Y.-A. Chung and J. Glass. Speech2Vec: A sequence-to-sequence framework for learning word embeddings from speech. In *Interspeech*, 2018.

K. Clark, M. Luong, Q. Le, and C. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

M. Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*, 2020.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

J. Fiscus, J. Garofolo, M Przybocki, W. Fisher, and D. Pallett. 1997 English broadcast news speech (HUB4), 1998. URL `https://catalog.ldc.upenn.edu/LDC98S71`.

W. Francis and H. Kucera. Brown corpus manual. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island, US, 1979. URL `http://icame.uib.no/brown/bcm.html`.

A. Gravano, M. Jansche, and M. Bacchiani. Restoring punctuation and capitalization in transcribed speech. In *ICASSP*, 2009.

S. Han, H. Mao, and W. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149*, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9, 1997.

J. Howard and S. Ruder. Fine-tuned language models for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

J. Huang and G. Zweig. Maximum entropy model for punctuation annotation from speech. In *Interspeech*, 2002.

B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, 2017.

C. C. Juin, R. X. J. Wei, L. F. D'Haro, and R. E. Banchs. Punctuation prediction using a bidirectional recurrent neural network with part-of-speech tagging. In *TENCON*, 2017.

J.-H. Kim and P. C. Woodland. A combined punctuation generation and speech recognition system and its performance enhancement using prosody. *Speech Communication*, 2003.

N. Kitaev, L. Kaiser, and A. Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020.

O. Klejch, P. Bell, and S. Renals. Punctuated transcription of multi-genre broadcasts using acoustic and lexical approaches. In *SLT*, 2016.

O. Klejch, P. Bell, and S. Renals. Sequence-to-sequence models for punctuated transcription combining lexical and acoustic features. In *ICASSP*, 2017.

J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya. A survey on addressing high-class imbalance in big data. *Journal of Big Data*, 2018.

Z. Li, E. Wallace, S. Shen, K. Lin, K. Keutzer, D. Klein, and J. E. Gonzalez. Train large, then compress: Rethinking model size for efficient training and inference of transformers. In *ICML*, 2020.

B. Lin and L. Wang. Joint prediction of punctuation and disfluency in speech transcripts. In *Interspeech*, 2020.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

W. Lu and H. Ng. Better punctuation prediction with dynamic conditional random fields. In *EMNLP*, 2010.

E. Ma. Nlp augmentation. https://github.com/makcedward/nlpaug, 2018.

K. Makhija, T.-N. Ho, and E.-S. Chng. Transfer learning for punctuation prediction. In *APSIPA ASC*, 2019.

J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel. Performance measures for information extraction. In *DARPA Broadcast News Workshop*, 2000.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 1970.

D. D. Palmer. SATZ-an adaptive sentence segmentation system. *Computer Science*, 1994.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 2019.

D. B. Paul and J. M. Baker. The design for the wall street journal-based CSR corpus. In *Proceedings of the workshop on Speech and Natural Language - HLT*, 1992.

J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 2020.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

C. Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 1948.

L. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

M. Sunkara, S. Ronanki, D. Bekal, S. Bodapati, and K. Kirchhoff. Multimodal semi-supervised learning framework for punctuation prediction in conversational speech. *arXiv preprint arXiv:2008.00702*, 2020.

Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020.

W. L. Taylor. "Cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, 1953.

O. Tilk and T. Alumäe. Bidirectional recurrent neural network with attention mechanism for punctuation restoration. In *Interspeech*, 2016.

B. Ueffing, M. Bisani, and P. Vozila. Improved models for automatic punctuation prediction for spoken and written text. In *Interspeech*, 2013.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

S. Wang, B. Li, M. Khabsa, H. Fang, and H. Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2018.

X. Wang, H. Ng, and K. Sim. Dynamic conditional random fields for joint sentence boundary and punctuation prediction. In *Interspeech*, 2012.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew. Huggingface's transformers: State-of-the-art natural language processing. In *EMNLP*, 2019.

J. Yi and J. Tao. Self-attention based model for punctuation prediction using word and speech embeddings. In *ICASSP*, 2019.

J. Yi, J. Tao, Z. Wen, and Y. Li. Distilling knowledge from an ensemble of models for punctuation prediction. In *Interspeech*, 2017.

J. Yi, J. Tao, Y. Bai, Z. Tian, and C. Fan. Adversarial transfer learning for punctuation restoration. *arXiv preprint arXiv:2004.00248*, 2020.

P. Żelasko, P. Szymański, J. Mizgajski, A. Szymczak, Y. Carmiel, and N. Dehak. Interspeech, 2018.

Appendix A

Background

A.1 On Dataset Statistics in Table 2.1

The train/evaluation/test splits for datasets used in punctuation annotation are inconsistent across papers, which means the percentages reported in 2.1 might vary from some of the evaluation sets used in punctuation papers. The sources for said numbers are as follows:

- Ueffing et al. (2013) for WSJ and IWSLT11 corpus.
- Kim and Woodland (2003) for BN corpus size and Batista et al. (2008) for BN punctuation percentages.
- Klejch et al. (2016) for MGB corpus.
- Juin et al. (2017) for Europarl corpus.

Punctuation marks other than full stop, comma and question mark were reported for several corpora and are shown in the Table below.

Dataset	Tokens	Full Stop	Comma	Question Mark	Exclamation Mark	Dash	Triple Dots
WSJ	51,023	4.59%	5.98%	0.04%	-	-	-
IWSLT11	17,207	5.37%	6.36%	0.48%	-	-	-
MGB	92,622	7.63%	4.77%	1.67%	1.18%	-	0.59%
BN	35,710	3.5%	5.1%	0.29%	-	-	-
Europarl	10,000	2.29%	2.36%	0.01%	0.01%	0.41%	-

Appendix B

Experiments

B.1 On Truncation Window Sizes

See the detailed results of truncation window sizes shown in Figure 4.2 below.

Test	Window	samples / second	Full Stop			Comma			Question			Overall		
			P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
IWSLT11	$w = 128$	572	75.3	86.2	80.4	64.7	65.2	64.9	57.1	80.0	66.7	70.5	76.7	73.5
	$w = 64$	1258 ($\times$2.19)	74.1	81.1	77.5	65.5	68.9	67.2	40.0	80.0	53.3	69.3	75.7	72.4
	$w = 32$	2340 ($\times$1.86)	74.6	84.9	79.4	69.1	71.2	70.1	50.0	80.0	61.5	71.7	78.7	75.0
	$w = 16$	4724 ($\times$2.01)	70.9	84.3	77.0	66.9	61.4	64.0	75.0	60.0	66.7	69.4	73.6	71.5
	$w = 8$	8060 ($\times$1.70)	62.9	49.1	55.1	40.0	9.1	14.8	100.0	40.0	57.1	59.0	31.1	40.7
MGB	$w = 128$	581	63.7	70.5	67.0	57.7	48.6	52.7	69.9	54.2	61.0	62.3	60.7	61.5
	$w = 64$	1241 ($\times$2.13)	64.2	72.2	68.0	58.3	48.8	53.1	69.5	54.2	60.9	62.8	61.6	62.2
	$w = 32$	2387 ($\times$1.92)	63.4	72.0	67.4	59.9	48.8	53.8	71.1	55.3	62.2	63.0	61.7	62.3
	$w = 16$	4649 ($\times$1.94)	61.5	69.8	65.4	56.6	46.9	51.3	68.5	45.5	54.6	60.5	58.7	59.6
	$w = 8$	8004 ($\times$1.72)	43.8	35.8	39.4	49.2	7.0	12.3	0.0	0.0	0.0	44.4	21.3	28.8