Per Token Early Exiting In the Convolutional Transformer Encoder

Anonymous ACL submission

Abstract

Early exiting models within the transformer architecture have shown to increase efficiency within simultaneous speech-to-text translation with minimal reduction in accuracy. However, current encoder based implementations evaluate inputs on a sequence level basis, averaging the computational needs of each token within the sequence. In addition, models that exit on a per-token basis are implemented in the decoder and use a limited amount of information to determine if exiting should take place. We solve this issue by purposing *Per Token Early Exiting*, which creates one-layer neural networks between layers in the encoder that recognize when certain tokens should exit and which tokens should be further processed. Our experiments on the MUST-C English-German and English-Spanish data sets have shown to increase the BLEU score and/or decrease FLOP's during evaluation across multiple wait-k values. On the English-German language pair at wait-k 7, the proposed model increased the BLEU score by 1.74 compared to the baseline implementation. In addition, across all wait-k values, the proposed model decreased the average FLOPs by 13.28% from the baseline.

1 Introduction

Simultaneous Speech-To-Text translation, SimulST, provides a real time bridge between languages by simultaneously taking in speech and producing a text translation. However, Simultaneous Speech-To-Text translation is very strenuous for humans to perform with the task requiring a person to have an immense knowledge of both the target and source language, as well as be able to accurately and efficiently listen to speech and produce a text translation. Machine learning models have shown the promising ability to perform SimulST accurately and efficiently. With new advancements in communication making the distance between people smaller and smaller, the need for seam-less, efficient, and accurate SimulST models has never been needed more.

Earlier transformer-based SimulST models have a fixed amount of Feed Forward Networks, FFN, within the encoder. In these models, every token within the input goes through every FFN layer in the model. Models with early exiting attempt to create a system within the encoder or decoder that monitors input and stops the processing of the input through the layers once a condition is achieved. Use of early exiting, as recent models have shown, decreases the computational strain while maintaining scores that are similar to models without early exiting (Xin et al.). Creating a mechanism that can recognize when an input sentence will not benefit that much, or at all, from further processing helps reduce the computations needed to produce an output. Most evaluation processes within early exiting models involve looking at the entropy of the input, creating additional linear layers to learn when to exit, or a combination of entropy and linear layers to hash into a look-up table to predict what layer to exit.

Current models that use early exiting provide great state-of-the-art performance. However, we have found having the evaluation and exiting of the input in the encoder on a per sentence basis does not consider the processing each token needs. When an input is received in the encoder, it is looked at on a sentence scale and evaluated and exited as such. When doing this, the need of each token to be further processed or exited is overlooked. This creates a scenario where the overall process could be more accurate and efficient by processing some tokens more and others less. This broad evaluation process exists in many early exit SimulST models such as Edgebert (Tambe et al.) and Deebert (Xin et al.). In models that exit on a per token basis, only the hidden states between layers are used to see if a token should exit. This creates the problem of having the model learn when to exit on limited amounts of information. In addition, these models do the exiting process on the previously generated tokens entering the decoder instead of in the encoder where the input is a whole sequence of words.

To address this issue, we purpose *Per Token Based Early Exiting*. This technique achieves a more granular approach to early exiting in the transformers encoder by evaluating the input on a per token basis. This is achieved by feeding the difference between the input and output of the FFN into a one layer neural network between each encoder layer. During training, the loss from early exiting is calculated between the encoders early exit results and the results generated from having all tokens go through all layers of the encoder. To keep track of which tokens should be further processed, each token is given a ID that is stored in a list. When a token should exit, the tokens ID is removed from the list and no longer processed by following layers.

The model is trained on the English-German, and English-Spanish language pairs from the MUST-C data set (Cattoni et al.). The model is evaluated upon by the following metrics:

1. **BLEU Score:** A language accuracy metric (Papineni et al.).

2. **FLOP's:** Floating-Point Operations, the amount of operations done by the encoder to process a input to the output (Li). The operations counted are division, subtraction, addition, and multiplication.

The evaluation process was done using SimulEval(Ma et al.).

The purpose of this paper is to:

1. Identify limitations within current transformer based early exiting schemes.

2. Purpose a solution for limitations within early exiting that is applicable to the transformer architecture.

3. Demonstrate the benefits from the purposed scheme by performing experiments on various language pairs, data sets, and wait-k values.

The model showed promising results increasing the BLEU score on the English-German language pair for wait-k 7 by 1.74 compared to the baseline model. In addition, on the English-Spanish language pair the average amount of FLOPs was decreased by 13.28% across all wait-k values when compared to the baseline implementation.

2 Background and Related Work

2.1 Simultaneous Translation

Transformer Architecture and Muti-Head Attention: Most state of the art SimulST models take advantage of the transformer architecture outlined in the paper *Attention Is All You Need* (Vaswani et al.). The transformer is composed of the encoder and decoder. The purpose of the encoder is to take in the whole sequence of information and produce a condensed output of the information. The decoder uses the previously translated word and the encoders output to produce a word by word translation. The benefit of using the condensed source sequence and the previously generated words is it takes into consideration how the previously generated word interacts in the sentence, giving insight into what the next word might be.

To make the architecture better equipped to deal with longer sequences the paper also introduces multi-head attention. Multi-head attention serves the purpose of generating attention scores for each word in the sentence. The attention scores allow the model to quantify the importance of a word in the sentence. Each score is generated through concatenation of the scaled dot-product from multiple attention heads.

Wait-k Policy: The wait-k policy involves the decoder in the transformer model waiting k words to be outputted from the encoder before staring translation. This policy was first seen in SimulST models (Ma et al.) and is used as a way for the model to switch between reading in k words and translating k words.

2.2 Computation Reduction Methods

Pruning: Pruning tries to reduce the computations needed for natural language processing by removing layers or nodes in the neural network within the architecture. Layer wise pruning (Peer et al.) reduces the computational cost of NLP models by determining what layers contribute little to the accuracy of a output. If a layer is determined to contribute little to the accuracy of a output the layer is removed, thus reducing the computational cost needed to generate a output. Node-wise pruning determines which node connections contribute little to the accuracy of a result (Blalock et al.). Nodes during training are assigned a score that indicates how much they contribute to the overall accuracy of the model, and based off this score the node is kept or cut out of the model by having its weight set to zero. After the network has been pruned the model goes through a *fine tuning* stage where the weights before pruning are go through further training. While this architecture has shown to provide reduction in the computational cost within models, the trade off comes with the accuracy achieved by pruned models. In addition, this approach is static during inference, with the amount of layers, or neurons, that the input go through being fixed.

Distillation: Distillation was presented in the paper *Distilbert, a Distilled Version of Bert: Smaller, Faster, Cheaper, and Lighter* (Sanh et al.). The architecture presented in the paper uses a teacher model to aid in the training of a student model. The student model is a smaller un-trained model, that takes in the same input as the teacher model. The teacher model is a larger pre-trained model that takes in a input and produce a distribution of what it thinks the output should be. The teacher model aids in training of the student model by producing a distribution the students distribution can be compared to. In addition, to being trained on the teacher model the student model is also trained using the labels for the data set. The benefit of this is that a smaller model can come close to the same accuracy as a larger model through being trained on a more robust data set. The limitations within this system is that the processing of sequences is static. Each token within the sequence is processed the same and the individual needs of each token is not taken into consideration during inference.

Early Exiting: Early exiting tries to reduce the computational cost associated with a model by ceasing processing of a input sequence or part of a input sequence once the models feels confident in the result. Early exiting is commonly implemented within the encoder section of the transformer, due to the demanding amount of computations that take place in that section. The methodology for determine when to early exit varies from model to model, but two common ways of doing so are:

1. Using the results from the first layer to predict at what layer to exit.
2. Continuously measuring the input to see if it meets a certain criteria.

The advantages of early exiting is that the model is dynamically changing how it processes the input by adjusting how much each input is processed. In addition, this architecture allows for the continual monitoring of how the sequencing is being processed to determine if it should exit.

3 Limitations Within Current Models

Early exiting has gained popularity within NLP models because of its ability to trade accuracy for efficiency. However, the implementation has limitations within it and can benefit from alterations to the granularity of exiting and the methodology of exit determination.

3.1 Granularity of Early Exiting

Most early exiting schemes within the encoder exit on a per-sequence level basis. The disadvantage of this is that the processing needs of each token is not taken it consideration. Instead, the computation each token needs is averaged to one point within the encoder to exit. Generalization of exiting can have a negative effect on finding at what point the best trade off between accuracy and efficiency takes place.

3.2 Static Exiting Inference

Algorithm 1: DeeBert Implementation (Input: X)

for i = 0 to n **do**
  $z_i = f_i(x; \theta)$
  if $entropy(z_i) < S$ **then**
    $exit\ inferenc$
  end if
end for
$exit\ inferenc$

Most early exiting schemes determine if a sequence should exit based off the entropy of the input after being processed by a layer. A common metric for determining if a sequence should exit is continuously measuring the entropy of a encoder layers output, and comparing the value to a hyper parameter s. Once the entropy is less then s the sequence exits and is sent to the decoder. This method of exiting was introduced in DeeBERT and is shown in Algorithm 1. The problems within this method of exit determination is that it is static and does not learn to recognize patterns for determining when to exit the encoder.

3.3 Learning Exiting on Limited Information

Algorithm 2: EdgeBert Implementation (Input: X)

for input sentences i = 1 to n **do**
  for encoder layers l = 1 **do**
    $z_l = f_l(i; \theta)$
    if $entropy(z_l \geq E_T$ **then**
      $exit\ inference$
    else
      $L_{predict} = LUT(entropy(z_l), E_T)$
    end if
  end for
  for $encoderlayerl = 2toL_{predict}$ **do**

$z_l = f_l(i, \theta)$
if $entropy(z_l \geq E_T$ **then**
$exit\ inference$
end if
end for
$exit\ inferenc$
end for

Some early exiting schemes, such as EdgeBERT, do implement neural networks to learn when to exit the encoder. EdgeBERT in addition to continuous entropy monitoring utilizes a single layer neural network after the first layer of the encoder. Once processed by the first layer the entropy of the results is taken and compared to a hyper parameter E_T. If the entropy is less then E_T the results exit at that layer. However, if they aren't the results are fed into a single layer neural network that is used to hash into a Look Up Table, LUT, and predict at what layer the sequence should exit, this layer is called $L_{predict}$. From layer 2 to $L_{predict}$ the entropy of the results of each layer is measured and compared to E_T, if the entropy is less then E_T and the current layer is before $L_{predict}$ the results exit. This scheme is similar to DeeBERT's but with the addition of setting the max amount of layers the sequence can go through after the first layer. The algorithm used by EdgeBERT is shown in algorithm 2. The benefit of this scheme is that it creates a point where the model can learn to recognize when a sequence should exit, however how it does so based of limited amount of information. The implementation only learns when to exit the encoder from one state of the results.

3.4 Architecture Placement

Some models, such as The Depth Adaptive Transformer model (Elbayad et al.), do implement early exiting on a per token level. However the implementation within the transformer architecture limits the effectiveness of the model. For determing what layer to exit the model uses two methods:

1. **Mutinomial:** The fist hidden state in the decoder is used to create a probability distribution, q_t, and estimate at what layer each token should exit.
2. **Geometric-like:** At each layer in the decoder the hidden state is ran through a neural network and sigmoid activation function. The resulting parameter is X_t^n, where t denotes the time stamp and n denotes the layer, is compared to a threshold τ. If X_t^n is greater then τ the token exits. Every value for the parameter X_t^n is used to create the distribution q_t

While this implementation does base exiting of tokens off information from multiple states of the input its implementation within the decoder limits the effectiveness. This is due to the computational demand of the decoder is less then that of the encoder. Since the focus of *The Depth Adaptive Transformer* is computational reduction in the decoder and the focus of this paper is computational reduction in the encoder comparisons to the *The Depth Adaptive Transformer* will not be taken into consideration.

4 Approach

The model we are proposing looks at adding single neural network layers in between each layer of the encoder. The neural networks are trained to recognize if a token should exit. The advantage of implementing a neural network between each layer is that each layer is able to learn how much of a effect each encoder layer has on changing the output.

4.1 How Exiting is Determined

Algorithm 3: Implemented Scheme (Input: X)

$X = [x_0, x_1, x_3, ..., x_m]$
$Key = [0, 1, 2, ..., m]$
for encoder layers i = 0 to N **do**
$Y_i = [x_k\ for\ k\ in\ Key]$
$Z_i = f_i(\theta;\ Y_i)$
$X_i = [x_k\ in\ X_i\ or\ z_p\ in\ Z_i\ if\ p = k]$
$K_i = h_i(\theta;\ (Z_i - Y_i))$

$Key = [m\ for\ k_m\ in\ k_i\ \geq\ 0.5]$
if len(key) = 0 **then**
$exit\ inference$
end if
end for

The approach outlined in algorithm 3 takes a more granular approach to early exiting within the encoder. The benefit of looking at each token individually is that each token is allowed to exit at the layer that provides the best trade off between accuracy and efficiency.

To keep track of which tokens should exit and which should be further processed each token is given a ID. The ID of each token is stored in a list that is passed from layer to layer. Each layer uses the list to filter out exited tokens from the multi-head attentions results. Once the results are filtered the tokens enter the encoders FFN section. The results from the FFN are subtracted from the tokens pre-FFN processing. The difference is then fed into a single layer neural network and using a softmax function the resulting probabilities are used to dictate which tokens no longer need be processed and which need to go to the next layer. Tokens that can exit at this layer have their ID removed from the processing list. The model continues through the encoder layers until their are no more IDs' on the processing list or at the last layer of the encoder.

4.2 Loss Calculations

During training when a token exits the encoder the probability associated with exiting at that point is recorded to create distribution d, of length N where N is the amount of tokens in that sequence. In addition, during training a second set of the input tokens are created called $x_{reference}$. $x_{reference}$ is allowed to go through all layers of the model, and is used to create the labels used for calculating exit related loss. To generate the labels the cosine similarity is calculated between the $x_{reference}$ tokens and the tokens that were allowed to early exit creating the similarity parameter C. The results are then filtered using the parameters τ_{upper} and τ_{lower}. If $\tau_{upper} \leq C \leq \tau_{lower}$ the exit is considered valid. τ_{upper} and τ_{lower} are hyper parameters that allows the user to adjust how accurate or efficient they want the model to be. Increasing τ_{upper} and τ_{lower} makes the model more accurate, while decrease τ_{upper} and τ_{lower} makes the model more efficient. The generated labels and the probabilities associated with each token when exited are then calculated using binary cross entropy,

$$Loss_{exit}(x, y) = L = l_1, ..., l_N$$

$$l_n = -[y_n\ log(x_n) + (1 - y_n) log(1 - x_n)]$$

where N is the size of the batch. The resulting exit loss is added to the label smoothed cross entropy loss of the model.

5 Experiments

5.1 Setup

The model was evaluated using the HE and COMMON MUST-C data sets. Language pairs that were used are English-German and English-Spanish. Evaluation was done using SimulEval, and the results from the implementation was compared to the baseline, and DeeBERT and EdgeBERT models that have early exiting within the transformers encoder. While the proposed model is compared to EdgeBERT and DeeBERT there are limitations between the comparison, due to the difference in metrics and baseline code EdgeBERT and DeeBERT are built on. Both are built on BERT, while the proposed implementation is built onto the Fairseq(Ott et al.) transformer program. When DeeBERT and EdgeBERT are referenced their proposed exiting algorithm is what is being used and compared to within the Fairseq model.

5.2 Training Parameters

The model was trained on a transformer with 12 encoder layers and 6 decoder layers. Model

Model	wait-k 1	wait-k 3	wait-k 5	wait-k 7
Baseline	4.66	11.88	14.29	15.99
DeeBERT	1.9	4.55	5.89	6.47
EdgeBERT	4.63	10.84	13.19	14.71
Proposed	5.23	11.57	14.71	16.73

Table 1: Table 1: Highest BLEU Scores Achieved for Proposed, EdgeBERT, DeeBERT, and Baseline Implementation at Various wait-k Values on the English German Language Pair COMMON Data Sets

Parameter	Proposed	Baseline
Wait-k 1	5.26	w.88
Wait-k 3	14.69	14.22
Wait-k 5	18.04	17.12
Wait-k 3	20.19	18.10
Average FLOPs	2228763883.51	2570090650.61

Table 2: Table 2: Proposed and Baseline BLEU Scores at Various wait-k Values and Average FLOPs on the English Spanish COMMON Data Set

training consisted of ASR pre-training and then SimulST training. During both training stages the overall loss of the system was calculated using label-smoothed cross-entropy and the early exit loss was calculated using binary cross-entropy. In both stages the Adam optimizer (Kingma and Ba) was used. During ASR pre-training a learning rate of 0.0007 was used. During SimulST training a warm up learning rate 0f of 0.0001 was used on on the first 4000 updates, after the first 4000 updataes a learning rate of 0.00035 was used. Both training stages used early stopping to evaluate when to cease training. Early stopping involves monitoring the model to see if it is still improving after a certain amount of epochs. Patience is the amount of epochs the model should compare to the dev set to see if the model is improving. ASR pre-training used a patience of 5 to determine when to stop, once ASR pre-training ends the top 5 checkpoints were averaged and used for SimulST training. For SimulST training a patience of 10 was used, and the best 10 checkpoints where averaged and used for evaluation. In total an estimated 1000 GPU hours went into experimentation and data collection.

5.3 BLEU Score

All models were first evaluated on BLEU scores, allowing us to find at what parameters each model best performed at. For EdgeBERT and DeeBERT entropy values presented in their respective papers were tested and looked at. For the per-token early exiting scheme multiple ranges and values for τ_{upper} and τ_{lower} were used. The best results for the English-German data set was $\tau_{upper} = 0.99$ and $\tau_{lower} = 0.90$. The best results for each model on the English-German language pair and COMMON data set is shown in table 1. For the English-German language pair the proposed implementation outperformed most of the other model at various wait-k values.

For the English-Spanish language pair we found that the implementation had a lower BLEU score then the baseline model. The results for this is shown in Table 2.

Model	wait-k 1	wait-k 3	wait-k 5	wait-k 7
Baseline	4.66	11.88	14.29	15.99
Proposed	4.67	10.73	14.47	16.51

Table 3: Table 1: BLUE Scores for Proposed with $\tau_{upper} = 0.74$ and $\tau_{lower} = 0.65$ and Baseline For Various wait-k Values Using COMMON Data Set and English-German Language Pair

5.4 FLOP

For the English German language pair it was found that the addition of neural networks between each layer increased the FLOPs needed for inference. This was due to the amount of computations needed to determine if a token should exit was greater then the amount of computations saved by tokens early exiting. However this was with $\tau_{upper} = 0.99$ and $\tau_{lower} = 0.90$ which sets priority on creating the best BLEU score and not the best efficiency possible. Even with accuracy prioritized the increase in FLOPs for all wait-k values using the English-German language pair was only 1.3% higher then the baseline implementation. If better efficiency is desired τ_{upper} and τ_{lower} can be changed to lower the FLOPs at the expense of a decrease in accuracy. At $\tau_{upper} = 0.74$ and $\tau_{lower} = 0.65$ the average FLOP score across all wait-k was seen to decrease by 2.35% compared to the baseline. The BLEU score for $\tau_{upper} = 0.74$ and $\tau_{lower} = 0.65$ for various wait-k values using the English-German language pair and COMMON data set is compared to the baseline in table 3.

When looking at the English-Spanish language pair the average amount of FLOPs were reduced by 13.28% across all wait-k values and data sets in comparison to the baseline implementation. This measurement was with $\tau_{upper} = 0.94$ and $\tau_{lower} = 0.85$, and the average amount of FLOPS for each implementation is shown in table 2.

6 Conclusion and Future Work

Per-token early exiting within the encoder using single layer neural networks between each layer for exit classification shows potential to increase the efficiency and accuracy during inference. Through learning how to recognize when a token can exit the encoder the model is able to make a informed decision on what point efficiency and accuracy are best achieved. The model compared to the baseline and other early exit models showed an overall increase in BLEU score on the English-German language pair for various wait-k values. The model also showed an increase in efficiency compared to the baseline on the English-Spanish language pair, with an average 13.35% FLOP reduction across all wait-k values and data sets. This reduction also had a minimal impact on BLEU score.

Further development of this model would include exploration into how to better classify exits as valid or invalid. Finding a better classification method would better allow the model to learn what peak efficiency and accuracy looks like. In addition, it would give the model a broader data set to learn exiting off of.

6.1 Limitations

While our proposed implementation shows promising results, the implementation is strictly limited to being used on SimulST. In addition, the model is currently limited to running on the Fairseq software and having adequate amount of memory to train and run the software.

7 References

Blalock, Davis, et al. “What Is the State of Neural Network Pruning?” arXiv.Org, 6 Mar. 2020, arxiv.org/abs/2003.03033.

Cattoni, Roldano, et al. “Must-C: A Multilingual Corpus for End-to-End Speech Translation.” Computer Speech & Language, Academic Press, 7 Oct. 2020, www.sciencedirect.com/science/article/abs/pii/S0885230820300887.

Devlin, Jacob, et al. “Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding.” arXiv.Org, 24 May 2019, arxiv.org/abs/1810.04805.

Elbayad, Maha, et al. “Depth-Adaptive Transformer.” arXiv.Org, 14 Feb. 2020, arxiv.org/abs/1910.10073.

Kingma, Diederik P., and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” arXiv.Org, 30 Jan. 2017, arxiv.org/abs/1412.6980.

Li, Danni. “Calculate Computational Efficiency of Deep Learning Models with Flops and Macs.” KDnuggets, www.kdnuggets.com/2023/06/calculate-computational-efficiency-deep-learning-models-flops-macs.html. Accessed 2 Oct. 2023.

Ma, Mingbo, et al. “STACL: Simultaneous Translation with Implicit Anticipation and Controllable Latency Using Prefix-to-Prefix Framework.” arXiv.Org, 24 June 2019, arxiv.org/abs/1810.08398.

Ma, Xutai, et al. “Simuleval: An Evaluation Toolkit for Simultaneous Translation.” ACL Anthology, aclanthology.org/2020.emnlp-demos.19/. Accessed 4 Oct. 2023.

Ott, Myle, et al. “Fairseq: A Fast, Extensible Toolkit for Sequence Modeling.” arXiv.Org, 1 Apr. 2019, arxiv.org/abs/1904.01038.

Papineni, Kishore, et al. “Bleu: A Method for Automatic Evaluation of Machine Translation.” ACL Anthology, aclanthology.org/P02-1040/. Accessed 4 Oct. 2023.

Peer, David, et al. “Greedy-Layer Pruning: Speeding up Transformer Models for Natural Language Processing.” arXiv.Org, 29 Mar. 2022, arxiv.org/abs/2105.14839.

Sanh, Victor, et al. “Distilbert, a Distilled Version of Bert: Smaller, Faster, Cheaper and Lighter.” arXiv.Org, 1 Mar. 2020, arxiv.org/abs/1910.01108.

Tambe, Thierry, et al. “Edgebert: Sentence-Level Energy Optimizations for Latency-Aware Multi-Task NLP Inference.” arXiv.Org, 6 Sept. 2021, arxiv.org/abs/2011.14203.

Vaswani, Ashish, et al. “Attention Is All You Need.” arXiv.Org, 2 Aug. 2023, arxiv.org/abs/1706.03762.

Xin, Ji, et al. “Berxit: Early Exiting for Bert with Better Fine-Tuning and Extension to Regression.” ACL Anthology, aclanthology.org/2021.eacl-main.8. Accessed 2 Oct. 2023.

Xin, Ji, et al. “Deebert: Dynamic Early Exiting for Accelerating Bert Inference.” arXiv.Org, 27 Apr. 2020, arxiv.org/abs/2004.12993.

Appendix

Fairseq is licensed under MIT

MUST-C is licensed under a Creative Commons License