Evaluating Data Augmentation with Attention Masks for Context Aware Transformations

by

Sofia M. Marquez

Submitted to the Department of Brain and Cognitive Sciences
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN COMPUTATION AND COGNITION

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2023

Authored by: Sofia M. Marquez
Department of Brain and Cognitive Sciences
August 11, 2023

Certified by: Fiona Murray
MIT Innovation Initiative Director, Thesis Supervisor

Accepted by: Sierra Vallin
Academic Administrator
Department of Brain and Cognitive Sciences

Evaluating Data Augmentation with Attention Masks for Context Aware Transformations

by

Sofia M. Marquez

Submitted to the Department of Brain and Cognitive Sciences
on August 11, 2023 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN COMPUTATION AND COGNITION

ABSTRACT

Transfer learning from large, pre-trained models and data augmentation are arguably the two most widespread solutions to the problem of data scarcity. However, both methods suffer from limitations that prevent more optimal solutions to natural language processing tasks. We consider that transfer learning benefits from fine-tuning on increased target dataset size, and that data augmentation benefits from applying transformations in a selective, rather than random, manner. Thus, this work evaluates a new augmentation paradigm that uses the attention masks of pre-trained transformers to more effectively apply text transformations in high-importance locations, creating augmentations which can be used for further fine-tuning. Our comprehensive analysis points to limited success of utilizing this context-aware augmentation method. By shedding light on its strengths and limitations, we offer insights that can guide the selection of optimal augmentation techniques for a variey of models, and lay groundwork for further research in the pursuit of effective solutions for natural language processing tasks under data constraints.

Thesis supervisor: Fiona Murray
Title: MIT Innovation Initiative Director

Acknowledgments

I am deeply grateful to Geoff Orazem, Founder of FedScout, and Katy Person, Program Manager, MIT Office of Innovation, for providing me the opportunity to serve as a research assistant. Their guidance, mentorship, and continuous support played a pivotal role in shaping the trajectory of my research. I am fortunate to have been part of such a dedicated team, and their contributions have significantly contributed to the successful completion of this thesis.

Contents

Chapter 1

Introduction

In the era of big data, where vast amounts of information are being generated at an unprecedented pace, the development of effective techniques for text classification has garnered significant attention. However, despite the abundance of data available in certain domains, many real-world applications still face the challenge of limited data availability. This scarcity poses a formidable obstacle to building accurate and robust text classification models [4]. Various strategies have shown promise in mitigating the impact of small datasets on model performance, including the popular techniques of transfer learning and data augmentation, though these each have their own limitations. This work evaluates the utility of a new fine-tuning schema, which seeks to overcome the limitations of both transfer learning and traditional augmentations.

Transfer learning involves applying the learned features or representations from a pre-trained model to a new task. Pre-trained language models, such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), have achieved remarkable success in various natural language processing tasks. These models are trained on large-scale datasets and capture intricate linguistic nuances, making them valuable resources for text classification with small datasets, should the knowledge encoded be applicable to the task. By leveraging transfer learning, one can fine-tune pre-trained models on the limited available data, enabling them to inherit knowledge from vast corpora.

Transfer learning, however, is not without shortcomings. When training on limited target data, the amount of data available may still be insufficient to optimize fine-tuning the pre-trained model. The pre-trained model may dominate the fine-tuning process, essentially overshadowing the limited target data. Importantly, the performance of fine-tuning is highly dependent on the size of the target dataset [5].

Another common solution to small datasets is using data augmentation. This method involves generating additional training samples by applying various transformations to the existing data. Techniques such as synonym replacement, random insertion or deletion of words, and sentence shuffling can help diversify the training set and expose the model to a broader range of linguistic patterns. By augmenting the data, the effective size of the dataset can be increased, potentially alleviating the limitations posed by small sample sizes. How-

ever, this method, too, faces limitations. Data augmentation techniques that involve random word replacements, deletions, or insertions may alter the original meaning or semantics of the text [1]. Especially when applied randomly, these modifications can introduce noise and potentially degrade model performance.

This work will introduce a novel data augmentation paradigm. We propose an approach that leverages the attention masks of pre-trained transformers to guide text replacements at specific locations in sentences. By utilizing the attention mask's focus on important indices, our method aims to augment data by intelligently substituting or altering these crucial elements, enhancing the model's understanding of context and improving its robustness to variations in the target task. After creating an augmented dataset, we fine-tune the pre-trained transformer on this new target data. Increasing the data available for fine-tuning a model on more data can lead to improved performance. With a larger amount of labeled data available for fine-tuning, the model can learn from a more diverse range of examples, better capture the underlying patterns in the data, and generalize more effectively.

Chapter 2

Related Work

2.1 Shortcomings of Random Augmentation Policies

Finding viable augmentation methods in NLP can be particularly challenging due to the inherent complexity of natural language. Unlike other domains, such as computer vision, where simple transformations like rotations or flips can be applied to images, text data requires a more sophisticated approach. One must be able to modify textual information without altering its original meaning, context, or syntactic structure. Ensuring that augmented data remains coherent and representative of real-world language use poses a significant obstacle in effectively augmenting NLP datasets, especially when used in conjunction with pre-trained language models.

In the case of text generation, it was demonstrated that "random" techniques like random insert, deletion, and swap led to improvements were minor and statistically insignificant. Because these techniques involve almost complete randomness, they result in high variations in the metric results and poor generations [1]. On simple classification tasks, noteworthy improvements are still limited to certain scenarios. Performance gains can be insignificant (often less than 1%), even when data is sufficient [5].

2.2 Attention Scores as a Measure of Token Importance

One key presupposition of our work is that the token locations associated with larger attention scores are worth investigating as opposed to random locations or other selection schemas. The paper "Attention Is All You Need" by Vaswani et al. [6], introduced the Transformer architecture, which revolutionized the field by presenting a novel self-attention mechanism. Th attention mechanism enables the model to compute attention scores that capture the relationships between tokens within an input sequence, subsequently leading to more informed and effective token representations. It fundamentally allows the model to weigh the interactions between tokens, capturing both local and global dependencies. Attention scores are computed by evaluating the affinity between tokens, revealing which tokens contribute most significantly to the understanding of the input sequence. This alignment

with token importance is a central aspect of our investigation into the utility of attention scores for data augmentation.

The interpretability aspect of attention scores further strengthens the case for their use in data augmentation. Analyzing the distribution of attention scores may provide insights into the model's focus when making predictions. Tokens with elevated attention scores correspond to regions of the input that heavily influence the model's decisions. Incorporating such tokens into data augmentation strategies offers a method to exploit the learned attention patterns of the model, potentially leading to augmented data that is more representative of the underlying patterns in the training data.

2.3 Effect of Augmentation Policies on Pre-trained Models

Even though the aforementioned simple, random augmentation methods have promise with small data sets, Wei et al. concede that they may not yield substantial improvements when used in conjunction with pre-trained models [5]. Longpre et al. [3] concurr that certain data augmentation techniques, including synonym replacement, may not yield significant improvements when applied to large pre-trained language models. They theorize that this is because these models already exhibit invariance to different transformations. This effect is more pronounced on larger pre-trained models.

We consider that this seeming invariance may be due to the distribution of augmentations. If augmentations often target low importance areas whose modification does not affect the context for prediction, then we would expect to see little to no improvement when applying these base augmentations for fine tuning. The method outlined in this paper hopes to overcome this invariance by targeting and transforming those areas which are likely to change prediction context. Augmentations are expected to help classification tasks with pre-trained models only when they provide linguistic patterns that are relevant but not seen during pretraining [5]. We expect the set of linguistic patterns experienced by a model to vary with the pre-trained model chosen. Thus, we determine each model's gaps and recommend optimal transformation techniques, if any at all, for each model tested.

Chapter 3

Methodology

For this thesis work, we analyze the feasibility of using context informed data augmentations, utilizing the self attention mask of pre-trained transformers. We benchmark with a variety of existing methods, implement the novel augmentation algorithm, and investigate which specific text transformations deliver the best results.

3.1 Benchmarking Existing Methods

To provide a baseline for comparison for the new augmentation method, We compare the performance of a variety of transformer models on text classification tasks. We analyze two popular transformers, also tested in Longpre et al. [3]: BERT and ROBERTa. For each of these models, we benchmark with the base augmentation policies (random synonym replacements, position, inserts, and deletions). We evaluate the performance of these methods by measuring the increase in accuracy versus the baseline model when fine-tuned on different size augmentation sets created by the selected policy.

3.2 Hyper-parameters

In this section, we elucidate the rationale behind our selection of hyperparameters, specifically tailored to limit GPU usage without compromising the model's overall effectiveness.

3.2.1 Batch Size and Gradient Accumulation Steps

Batch size and gradient accumulation steps play pivotal roles in balancing model training efficiency and memory utilization. Larger batch sizes often enhance training speed by allowing for parallelization across multiple examples. However, they can also lead to excessive GPU memory consumption, potentially resulting in memory overflow issues. To mitigate this, we opted for a relatively conservative batch size of 32. This choice aligns with the available GPU memory capacity and ensures stable training sessions.

To further optimize GPU memory utilization, we employed gradient accumulation. With a gradient accumulation step of 4, gradients are accumulated over four mini-batches before

a weight update is performed. This effectively reduces the memory requirement per mini-batch, allowing us to maintain a larger effective batch size without encountering memory constraints.

3.2.2 Epoch

In order to conserve resources and time, we chose a conservative epoch number, 3, for the baseline comparisons of models and for comparisons between augmentation methods. This number of epochs allowed any potential benefits to be observed while preventing overfitting.

3.2.3 Learning Rate

The learning rate is a critical hyperparameter that governs the step size taken in the direction opposite to the gradient during optimization. Choosing an appropriate learning rate is crucial, as too high a value can lead to divergence, while too low a value can result in slow convergence. For our fine-tuning process, we opted for a learning rate of 2e-5. This value was chosen after testing a range of learning rates (1e-5 - 5e-5) and evaluating BERT model results from 3 epochs of training on both EDA and attention-informed EDA augmented datasets.

3.3 Binary vs Multi-class Classification

One important consideration was the added challenge of multi-class classification when compared to binary. We hypothesized that the effectiveness of data augmentation techniques may be hindered in multi-class classification, as the boundaries between classes are less distinct. Consider classification on two different datasets: one dataset contains the labels 'bad' and 'good', while the other dataset contains the labels 'bad', 'good', and 'great'. We refer to a case where an augmentation causes a sentence to be misclassified as a 'corruption.' Data augmentation with random augmentation policies may lead to higher corruption rates due to the increased complexity of preserving the semantic integrity of the text. To test this hypothesis, the we will compare the performance of classification models on two diverse datasets: the IMDB Movie Review Dataset for binary sentiment classification and the AG's News Topic Classification Dataset for multi-class topic classification. By evaluating the impact of the various data augmentation techniques on these datasets, we aim to gain insights into the unique challenges posed by multi-class classification and inform the development of more effective augmentation strategies for such tasks.

3.4 Attention-based Augmentation Algorithm

We first selected the percent increase in the training set desired. We compared results for augmentation increases of 10%, 20%, and 30%.

The algorithm implements the following steps:

- Pre-process the data in accordance with the current transformer model.

- Feed a data point (sentence) into the transformer model's forward pass and retrieve the attention weights generated during self-attention. The attention matrix returned is segregated by batch and attention head, with dimensions ($batch\_size$, num_heads, $sequence\_length$, $sequence\_length$). It is symmetric in the sequence length dimensions. Therefore, to compute one attention score per index, we take the mean across the first three dimensions, resulting in a 1d, $sequence\_length$ vector. Higher attention scores suggest that the token has a greater influence on the output, while lower attention weights indicate less importance.

- Pair each word in the sentence with its corresponding importance score and position.

- Sort the pairs based on the importance scores in descending order.

- Extract the positions of the words from the sorted pairs.

- Iteratively apply a chosen transformation policy (synonym replacement, index swapping, insert, or deletion) to the first k most most important positions. As such, k is the number of transformations to be applied per augmentation and should be optimized through testing.

With the augmentations created, we append them to the original data set, and continue training and testing as per the requirements of the model being evaluated.

Chapter 4

Results

The results of the experiments run according to the previous section's guidelines are shown below in subsections **4.0.1 - 4.0.4**. Our results align with previous research [5] that has indicated the limited feasibility of augmentations in significantly improving the performance of transformer models. Despite the attention-informed nature of the augmentation technique we introduced, it did not translate into substantial improvements in model accuracy. The gains in accuracy were less than 1% across the model types and tasks. These observations suggest several important takeaways from our study.

The subtle nature of the observed performance changes underscores the notion that improving transformer model performance through augmentation is a non-trivial task. Transformer models have already achieved high levels of performance on a wide range of tasks, leaving limited room for further enhancement through simple data augmentation methods. These augmentation methods seem to introduce no new information or linguistic patterns that would increase invariance and robustness.

Another clear observation is that there is no significant difference when applying transformations at the indices associated with high attention scores. While it is intuitive to assume that augmentations at these salient positions would have a greater impact on model performance, our findings indicate that the interactions between attention and augmentation are not fully understood and may require further investigation. One reason why the attention-informed-augmentations seem to be equal to random augmentations may be because the attention score matrix returned must be averaged across the batches and attention heads in order to receive one value per index in a transformer sequence. This process may obscure the true importance that an index should have and that it is given in the transformer model's forward pass.

4.0.1 Binary BERT

Baseline: 0.9543

	Synonym Replacement	Insert	Swap	Delete
Regular EDA 10%	0.955	0.9582	0.9600	0.9587
Attention EDA 10%	0.948	0.944	0.9603	0.9601
Regular EDA 20%	0.9591	0.9599	0.9573	0.9535
Attention EDA 20%	0.9584	0.9566	0.9494	0.9571
Regular EDA 30%	0.9610	0.9621	0.9589	0.9575
Attention EDA 30%	0.9590	0.9617	0.9603	0.9583

Table 4.1: Results across augmentation sizes for BERT on the binary IMDB dataset

4.0.2 Binary Roberta

Baseline: 0.9514

	Synonym Replacement	Insert	Swap	Delete
Regular EDA 30%	0.9623	0.9631	0.9564	0.9604
Attention EDA 30%	0.9572	0.9591	0.9588	0.9611

Table 4.2: Results for 30% augmentation for ROBERTa on the binary IMDB dataset

4.0.3 Multiclass Bert

Baseline = 0.9300

	Synonym Replacement	Insert	Swap	Delete
Regular EDA 10%	0.9364	0.9347	0.9349	0.9346
Attention EDA 10%	0.94	0.9334	0.9350	0.9295
Regular EDA 20%	0.9366	0.9300	0.9353	0.9387
Attention EDA 20%	0.9361	0.9374	0.9360	0.9349
Regular EDA 30%	0.9375	0.9374	0.9360	0.9349
Attention EDA 30%	0.9333	0.9345	0.9325	0.9328

Table 4.3: Results across augmentation sizes for BERT on the multi-class AG News dataset

4.0.4 Multiclass Roberta

Baseline: 0.9310

	Synonym Replacement	Insert	Swap	Delete
Regular EDA 30%	0.9329	0.9309	0.9346	0.9342
Attention EDA 30%	0.9320	0.9334	0.9342	0.9303

Table 4.4: Results for 30% augmentation for ROBERTa on the multi-class AG News dataset

Chapter 5

Conclusion

In the pursuit of enhancing the efficacy of data augmentation for transformer-based models, our work targeted the ROBERTa and BERT models, employing the techniques of Easy Data Augmentation (EDA) and our modified, attention-score-informed EDA extention. The outcome establishes that augmentation strategies wield little to no influence over the performance of the select models under scrutiny. Almost all improvements observed were under 1%, and while increasing the size of the augmentation of training dataset seemed to exert a small positive influence on model performance, it required additional training time and compute power that negated all potential benefits. While augmentations can be instrumental mechanisms for curbing overfitting tendencies and boosting a models' generalization, this only hold when a model has not already been exposed to massive data sets.

Comparing the two augmentation methods, we observed that augmenting data according to attention scores led to no performance gains. This observation directly contradicts the intuition that incorporating augmentations based on attention scores can guide the model towards areas of high importance within the input data and facilitate more meaningful and effective learning. This may be because of the methodology for extracting attention scores, which batches and takes a mean over attention heads and layers. When taken out of the natural context of a transformer's forward pass, these attention scores seem to become ineffectual.

Our study highlights the need to find other means to enhance the performance of transformer-based models. When a model has already been exposed to a massive amount of data, the performance gains achieved through augmentation, add little robustness or transformation invariance to a model. Furthermore, while attention scores are demonstrably valuable within transformers, their interpretability is up to debate, and continued research in this area is vital not only for improving our understanding, but and facilitating their use in new contexts.

5.1 Future Work

The method for assigning attention scores to a token in this paper may be obscuring a token's actual importance. In order to achieve a single attention score value for each token, attention weight matrices were averaged across the dimensions for batch and attention head.

Further research could investigate a new method of extracting an attention score for the token, more aligned with the importance it is actually given by the transformer model.

While this study has demonstrated little effectiveness in utilizing attention weights from transformers to enhance the performance of data augmentation using EDA, there remain several unexplored directions for augmenting data in non-transformer models. Investigating alternative context-aware or non-random augmentation strategies for traditional machine learning models could provide valuable insights and potentially lead to further performance improvements. One such avenue for future research could be a semantically based augmentation policy. For example, one could leverage pre-trained word embeddings, syntactic parsers, or part-of-speech taggers to enable the generation of contextually relevant augmentations. These techniques could be particularly valuable for tasks that require preserving the semantic meaning of the data.

Appendix A

Code listing

Generating attention scores

```

batch_size = ... # batch size for which to extract attention scores
    (100)
scores_per_batch = None

# Perform the forward pass to get the outputs
for i in range(batch_size, len(df_train_subset) + 1, batch_size):
  model = ... #model witch which to generate (BERT, ROBERTa)

  with torch.no_grad():
    outputs = model(input_ids_tensor[i-batch_size:i], attention_mask=
       attention_masks_tensor[i-batch_size:i])

  attentions = outputs.attentions
  # dimensions =  (batch_size, num_heads, sequence_length,
     sequence_length)

  # Stack the attentions across each batch and average across batch,
     num_heads, and seq_length
  average_attention_weights = torch.mean(torch.stack(attentions, dim
     =0), dim=(0, 1, 2))
  # Apply mean along the rows to get attention scores for each index
  attention_scores = torch.mean(average_attention_weights, dim=0)


  if scores_per_batch is not None:
    scores_per_batch = torch.cat((scores_per_batch,
       attention_scores.reshape(1, len(attention_scores))), dim = 0)
  else:
    scores_per_batch = attention_scores.reshape(1, len(
       attention_scores))
```

,

Synonym replacement

```
def augment_sr(input_ids, attention_masks,
   token_position_attention_pairs, k = 3, regular = False):
  sr_augmented_input_ids = []
  sr_augmented_attention_masks = []

  for input_ids_for_sentence, attention_mask,
     token_position_attention_pair in zip(input_ids, attention_masks,
      token_position_attention_pairs):
    token_list = token_position_attention_pair
    if regular:
      random.shuffle(token_list)
    else:
      token_list = sorted(token_list, key=lambda x: x[2], reverse=
         True)
    input_ids_for_sentence_augmented = input_ids_for_sentence.detach
       ().clone()
    attention_mask_augmented = attention_mask.detach().clone()

    num_altered = 0
    for token, position, attention in token_list:
      if token not in stops and token not in string.punctuation:
          synonyms = []
          for synset in wordnet.synsets(token):
            for lemma in synset.lemmas():
              if lemma.name() != token:
                synonyms.append(lemma.name())
          if len(synonyms) > 0:
            for synonym in synonyms:
              if tokenizer.convert_tokens_to_ids(synonym) !=
                 TOKEN_NUMBER_UNK:
                input_ids_for_sentence_augmented[position] =
                   tokenizer.convert_tokens_to_ids(synonym)
                attention_mask_augmented[position] = 1
                num_altered += 1
                break # only replace once


      if num_altered == k:
          break

    sr_augmented_input_ids.append(input_ids_for_sentence_augmented)
    sr_augmented_attention_masks.append(attention_mask_augmented)

  return sr_augmented_input_ids, sr_augmented_attention_masks
,
,
```

Insertion

```
def augment_ins(input_ids, attention_masks,
   token_position_attention_pairs, k = 3, regular = False):
  sr_augmented_input_ids = []
  sr_augmented_attention_masks = []

  for input_ids_for_sentence, attention_mask,
     token_position_attention_pair in zip(input_ids, attention_masks,
      token_position_attention_pairs):
    token_list = token_position_attention_pair
    if regular:
     random.shuffle(token_list)
    else:
      token_list = sorted(token_list, key=lambda x: x[2], reverse=
         True)

    input_ids_for_sentence_augmented = input_ids_for_sentence.detach
       ().clone()

    attention_mask_augmented = attention_mask.detach().clone()
    num_altered = 0
    to_be_inserted = []


    for token, position, attention in token_list:
      if token not in stops and token not in string.punctuation:
          synonyms = []
          for synset in wordnet.synsets(token):
            for lemma in synset.lemmas():
              if lemma.name() != token:
                synonyms.append(lemma.name())
          if len(synonyms) > 0:
            for synonym in synonyms:
              if tokenizer.convert_tokens_to_ids(synonym) !=
                 TOKEN_NUMBER_UNK:
                to_be_inserted.append(tokenizer.convert_tokens_to_ids
                   (synonym))
                num_altered += 1
                break # only replace once


      if num_altered == k:
          for new_token_id in to_be_inserted:
            idx_first_sep = tf.where(tf.equal(
               input_ids_for_sentence_augmented.cpu(),
               TOKEN_NUMBER_SEP))[0][0]
            rand_idx = tf.random.uniform(shape=[], minval=0, maxval=
```

```
                idx_first_sep, dtype=tf.int64)
            before_token = input_ids_for_sentence_augmented[:rand_idx]
            with_new_token = torch.cat((before_token, torch.tensor([new_token_id]).to(device)))
            input_ids_for_sentence_augmented = torch.cat((with_new_token, input_ids_for_sentence_augmented[rand_idx:-1]))
            attention_mask_augmented[idx_first_sep.numpy()] = 1

            #If insertion deletes separator, reinsert token
            if idx_first_sep == len(attention_mask_augmented)-1:
                input_ids_for_sentence_augmented[len(attention_mask_augmented)-1] = TOKEN_NUMBER_SEP

          break


    sr_augmented_input_ids.append(input_ids_for_sentence_augmented)
    sr_augmented_attention_masks.append(attention_mask_augmented)

  return sr_augmented_input_ids, sr_augmented_attention_masks
```

Swap

```
def augment_swap(input_ids, attention_masks, token_position_attention_pairs, k = 3, regular = False):
  sr_augmented_input_ids = []
  sr_augmented_attention_masks = []

  for input_ids_for_sentence, attention_mask, token_position_attention_pair in zip(input_ids, attention_masks, token_position_attention_pairs):
    token_list = token_position_attention_pair
    if regular:
      random.shuffle(token_list)
    else:
      token_list = sorted(token_list, key=lambda x: x[2], reverse=True)

    input_ids_for_sentence_augmented = input_ids_for_sentence.detach().clone()
    attention_mask_augmented = attention_mask.detach().clone()

    num_altered = 0
    to_be_inserted = []
    for token, position, attention in token_list:
      if token not in stops and token not in string.punctuation:
          synonyms = []
```

```
          for synset in wordnet.synsets(token):
            for lemma in synset.lemmas():
              if lemma.name() != token:
                synonyms.append(lemma.name())
          if len(synonyms) > 0:
            for synonym in synonyms:
              if tokenizer.convert_tokens_to_ids(synonym) !=
                 TOKEN_NUMBER_UNK: # 100 is the id for [UNK]
                to_be_inserted.append(tokenizer.convert_tokens_to_ids
                   (synonym))
                num_altered += 1
                break # only replace once


      if num_altered == k:
        for new_token_id in to_be_inserted:
          idx_first_sep = tf.where(tf.equal(
             input_ids_for_sentence_augmented.cpu(), TOKEN_NUMBER_SEP
             ))[0][0]
          rand_idx = tf.random.uniform(shape=[], minval=0, maxval=
             idx_first_sep, dtype=tf.int64)
          input_ids_for_sentence_augmented[rand_idx.numpy()] =
             new_token_id
        break


    sr_augmented_input_ids.append(input_ids_for_sentence_augmented)
    sr_augmented_attention_masks.append(attention_mask_augmented)

  return sr_augmented_input_ids, sr_augmented_attention_masks
```

Deletion

```
def augment_del(input_ids, attention_masks,
   token_position_attention_pairs, k = 3, regular = False):

  sr_augmented_input_ids = []
  sr_augmented_attention_masks = []

  for input_ids_for_sentence, attention_mask,
     token_position_attention_pair in zip(input_ids, attention_masks,
      token_position_attention_pairs):
    input_ids_for_sentence_augmented = input_ids_for_sentence.detach
       ().clone()
    attention_mask_augmented = attention_mask.detach().clone()

    steps = k

    for deletion_step in range(steps):
```

```
      idx_first_sep = tf.where(tf.equal(
         input_ids_for_sentence_augmented.cpu(), TOKEN_NUMBER_SEP))
         [0][0]

      if regular:
        idx = tf.random.uniform(shape=[], minval=0, maxval=
           idx_first_sep-1, dtype=tf.int64)
      else:
        idx = token_position_attention_pair[deletion_step][1]
        if idx >= idx_first_sep.numpy():
          idx = None
          steps += 1

      if idx:
        new_val = TOKEN_NUMBER_PAD # 0 is the padding token
        attention_mask_augmented[idx_first_sep.numpy()] =
           TOKEN_NUMBER_PAD

        for i in range(idx_first_sep, idx - 1, -1):
          tmp = input_ids_for_sentence_augmented[i].clone()
          input_ids_for_sentence_augmented[i] = new_val
          new_val = tmp


    sr_augmented_input_ids.append(input_ids_for_sentence_augmented)
    sr_augmented_attention_masks.append(attention_mask_augmented)

  return sr_augmented_input_ids, sr_augmented_attention_masks
```

References

[1] J. Chen, D. Tam, C. Raffel, M. Bansal, and D. Yang, "An empirical survey of data augmentation for limited data learning in nlp," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 191–211, 2023. DOI: 10.1162/tacl_a_00542.

[2] S. Y. Feng, V. Gangal, D. Kang, T. Mitamura, and E. Hovy, "GenAug: Data augmentation for finetuning text generators," in *Proceedings of Deep Learning Inside Out (DeeLIO): The First Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, Online: Association for Computational Linguistics, Nov. 2020, pp. 29–42. DOI: 10.18653/v1/2020.deelio-1.4. [Online]. Available: https://aclanthology.org/2020.deelio-1.4.

[3] S. Longpre, Y. Wang, and C. DuBois, "How effective is task-agnostic data augmentation for pretrained transformers?" *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020. DOI: 10.18653/v1/2020.findings-emnlp.394.

[4] M. A. Bansal, D. R. Sharma, and D. M. Kathuria, "A systematic review on data scarcity problem in deep learning: Solution and applications," *ACM Computing Surveys*, vol. 54, no. 10s, pp. 1–29, 2022. DOI: 10.1145/3502287.

[5] J. Wei and K. Zou, "Eda: Easy data augmentation techniques for boosting performance on text classification tasks," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. DOI: 10.18653/v1/d19-1670.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.