Final Project Report - CS 388 Natural Language Processing - Fall 2021

Three Steps Towards Mitigating NLI Dataset Artifacts: Automatic Contextual Augmentation, Contrastive Learning and Hybrid Loss

Michail Mersinias*
Department of Computer Science
University of Texas at Austin
mmersinias@utexas.edu

Panagiotis Valvis*
Department of Computer Science
University of Texas at Austin
pv5987@utexas.edu

December 9, 2021

Abstract

In recent years, Natural Language Inference has been an emerging research area. In this paper, we present a novel data augmentation and learning procedure for that task. Our data augmentation process manages to be fully automatic, non-trivially contextual, and computationally efficient at the same time. We combine this augmentation procedure with the recently published technique of *Contrastive Learning* [4] to maximize the benefit of the supervisory signal generated by our augmentation. Our algorithm is shown experimentally to provide an effective way for boosting performance, achieving approximately 1% improvement in the accuracy metric on the same pretrained language model. Compared to established data augmentation methods such as *TextAttack* [9] and *Checklist Sets* [11], it is substantially more computationally efficient and requires no manual annotation by a human expert as these packages do. In addition, we propose a hybrid loss function that incentivises the model to learn the nuances of the decision boundary, while maintaining overall performance. Our experiments were carried out on the Stanford Natural Language Inference (SNLI) corpus [2] and verify that transformer-based machine learning models that use our Data Augmentation, Contrastive Learning and Hybrid Loss function approach, consistently outperform the baseline model.

Keywords Natural Language Inference, SNLI, Data Augmentation, WordNet, Contrastive Learning
Abbreviations ENT: Entailment, NEU: Neutral, CON: Contradiction

1 Introduction

Inference has historically been a central topic in artificial intelligence, and recently so in the natural language domain. The so called *Natural Language Inference* (NLI) task is to determine whether a natural language hypothesis h can justifiably be inferred from a natural language premise p. This is a challenging task due to the complex nature of natural language which entails informal reasoning, lexical semantic knowledge and structure as well as variability regarding linguistic expression.

In recent years, there been a considerable amount of research in this particular area. The *Stanford Natural Language Inference* (SNLI) corpus [2] has provided a robust benchmark for NLI. Furthermore, the advancement of transformer-based machine learning techniques has contributed to the increased effectiveness of proposed solutions over the past few years.

Although many of these solutions appear promising, as they report a high accuracy on validation data, the task of NLI remains a work in progress. Recent research calls into question the learning which results from transformer-based machine learning techniques

*All authors contributed equally to the writing and research in this study. Their names are listed in alphabetical order.

in datasets such as SNLI because it either predicts the right answer when it shouldn't, as it is the case with hypothesis-only baselines [10], or predicts the wrong answer if minor modifications are made by utilizing contrast sets [5], adversarial attacks [7] or checklist examples [11]. These observations all stem from the fact that a model may achieve high performance on a dataset by learning spurious correlations, which are called *Dataset Artifacts*, but it is then expected to fail in settings where these artifacts are not present.

In our work, we attempt to identify and tackle these dataset artifacts in the SNLI dataset in order to achieve a higher and more robust performance than the baseline transformer-based machine learning model. As our baseline and to apply the techniques discussed below we make use of an `electra-small-discriminator` pretrained language model [3] from the Hugginface Transformers project [14]. Our methodology can be summarized as follows. First, we propose a novel data augmentation approach for the construction of adversarial examples to enrich the dataset for the purpose of enhancing the learning process. We perform systematic comparisons of our improved model with the baseline model, and demonstrate experimentally that our data augmentation approach can boost performance, while remaining computationally efficient when compared to established data augmentation techniques such as *TextAttack* [9] and *Checklist* [11] which require much longer training times and computational resources to produce similar results. Furthermore, we propose a hybrid loss function which provides results that are considerably more robust than those of the default NLL loss function of the baseline model which is based on the MLE criterion. Finally, we make use of Contrastive Learning [4] to further increase the effectiveness of our approach. As a result, our combined approach yields around a 1% increase in accuracy versus the baseline model, and we also provide some hand-annotated adversarial examples for both the baseline and our improve models to demonstrate its effectiveness.

2 Background and Related Work

We will not discuss the theory behind Data Augmentation and hybrid loss functions as these are considered well-known techniques in Deep Learning. Contrastive Learning takes inspiration from contrastive estimation [12], extending the technique to supervised reading comprehension by carefully selecting appropriate neighbourhoods of related examples. In the original paper it was used in the context of Question Answering and required bundles of closely related Question-Answering pairs [4] which the authors call *Instance Bundles*. Here we further show that the same technique can also be successfully employed in Language Inference problems, with our data augmentation procedure offering a natural way of creating multiple instance bundles of closely related language inference examples (and of arbitrary size that grows exponentially with the length of the hypothesis sentence that is being augmented). We retain the authors' original technique of combining a Cross Entropy Loss with Maximum Likelihood Estimation (NLL Loss).

3 Part 1: Dataset Analysis

The first task in solving the issue of dataset artifacts is to identify them. For this purpose, we perform a split on the dataset and use a subset S as validation data. We conduct an exploratory analysis on S by examining the examples which the model fails to classify correctly. From this analysis, we can make the following conclusions:

Firstly, the model's errors are heavily located around a particular class, the Neutral class. The breakdown of the gold labels when the model prediction is incorrect is **Entailment 11.3%** , **Neutral 85.3%**, and **Contradiction 3.4%**. Therefore we can conclude that the model has trouble predicting Neutral as a label altogether, which makes it the general class of mistakes the model makes. One of the potential artifacts at work here is a distance function between the premise and the hypothesis which the model learns instead of actual comprehension, and depending on that distance artifact makes a prediction. Because the Neutral class cannot be adequately expressed by distance (or more accurately, the distance of hyponyms in embedding space can be very large and confuse the artifact's criterion) the model classifies these large distances as Contradictions, resulting in a substantial hit in performance.

Secondly, we tested our model with adversarial examples using strategies in the vein of established adversarial attacking techniques [15], [9], [1], [6], [5]. The model performed well against trivial augmentations, such as introducing a negation in the form of adding a "not" word in the Premise. However, when adversarial examples used words that are further apart in embedding space from the Premise words, results were much worse. The model clearly relies on learned artifacts instead of learned language comprehension. Apart from the distance function discussed above, another artifact at play is likely to be sets of words in the Hypothesis that the model associates with a specific label regardless of context, only becuse it has seen them multiple times accompanying that label during training, as developed in detail in

[13]. Some such examples are "not" and "least" for Entailment, "Joyously" for Neutral, and "nobody" and "never" for Contradiction. Some hand-annotated by us aversarial examples demonstrating the presence of both artifacts are presented in Table 1. We can observe, for example, how the phrase "to go" is a synonym for words such as "takeaway" or "food", and yet the model produces an incorrect prediction for our adversarial example which is a small and natural shift in language, the simple use of a synonym (rows 1 and 2).

The model also fails (row 4) when a more specific word is introduced ("lasagna") which is a hyponym of "food" and shifts the gold label to Neutral, but the model reads this as contradicting the Premise. Moreover, observe how in rows 8 and 9 even slight changes in context (specificity) cause the model to choose Contradiction while Neutral should have been appropriate, a possible effect of the distance function artifact. The most definitive example of the distance function artifact is likely to be row 5. We can observe how the model, by seeing the same phrase in both the Premise and Hypothesis assigns Entailment as the label, unable to differentiate what the pattern is in reference to language (reading comprehension would require that it can differentiate between "structure" and "shirt"). We make use of this small subset of hand-annotated examples to evaluate our improved model later.

Premise	Hypothesis	Label	Pred
Two women are embracing while holding to go packages.	One of the women is holding a package.	ENT	CON
...	The packages contain food.	ENT	CON
...	The women have bought food.	ENT	CON
...	The women have bought lasagna.	NEU	CON
A man in a blue shirt standing in front of a garage-like structure painted with geometric designs.	A man is wearing black trousers.	NEU	CON
A young boy in a field of flowers carrying a ball	He is carrying one ball.	ENT	CON
...	Ball in field.	ENT	CON
Two doctors perform surgery on patient.	The two doctors are performing brain surgery.	NEU	CON
...	The patient is having heart surgery.	NEU	CON
A white dog with long hair jumps to catch a red and green toy.	It is not a brown dog.	ENT	CON
Kids are on a amusement ride.	Kids ride joyously an amusement ride.	ENT	CON

Table 1: Baseline Model: sampled examples of observed patterns that consitute dataset artifacts with a focus on the Entailment and Neutral classes where the model is weakest.

4 Part 2: Mitigating Artifacts

Our approach comprises three techniques: A custom Data Augmentation procedure, Contrastive Learning [4], and using a Hybrid Loss function.

4.1 Data Augmentation

Recall the examples from Table 1 in Part 1. These observations natually lead to an approach where contextual augmentation based on word groups could incentivise the model to learn the decision boundary instead of relying on artifacts. That is, if more hypotheses that are closely related with each other (such as the adversarial ones that we created manually) were made available to the model, but which included substantial contextual shifts (such as the ones the model fails at), there could be a benefit in performance. Moreover, we require that this augmentation procedure is automatic, i.e. it does not require a human expert manually annotating examples because otherwise the resources required would make the procedure infeasible. We devised such a data augmentation procedure that generates new examples which on one hand are non-trivial (as opposed for example to adding a "not" ahead of the hypothesis), while at the same time being robust in labelling the newly generated example correctly. To achieve non-trivial augmentation we employed *WordNet* [8] synsets and generated a new hypothesis while leaving the premise as it is. This was done by replacing one word in the hypothesis with either a synonym, an antonym, a hyponym or a hypernym, and to ensure sensible labelling of the new example we employed the set of rules shown in Table 2.

The rules that result in Unknown (UNK) labels were not used as part of augmentation. Because of the inherent ambiguity when replacing a word in these contexts, the supervisory signal can be corrupted and lead the model to learn nonsensical rules. Importantly, note that the remaining rules are robust, but they are not infallible: there is still the possibility (however small) that a newly generated

Old Label	Word Swap	New Label
ENT	Synonymn | Hypernym	ENT
ENT	Antonym	CON
ENT	Hyponym	NEU
NEU	Synonymn | Hypernym	NEU
NEU	Antonym	UNK
NEU	Hyponym	UNK
CON	Synonymn | Hypernym	CON
CON	Antonym	UNK
CON	Hyponym	CON

Table 2: Augmentation Rules using WordNet Synsets

example gets an incorrect label assigned to it. However, this was deemed acceptable, because the inherent ambiguity in labelling any hypothesis is already only partially correct, even when done by human experts, as developed in detail in recent published research which shows that, indeed, numerous examples can be found in the SNLI and other similar datasets where human experts disagree on which label to assign to a hypothesis [4]. By keeping only the more robust rules for augmentation we ensure that the probability of generating a controversial example will be similar to the one induced by human experts, and will therefore not alter the undelying manifold of the dataset that the model is trying to learn.

The resulting augmentation benefits from being both automatic (not requiring manual writing of new hypothesis and annotating labels) while at the same time being non-trivial. For example, we can observe that by using Rule 1 in the hypothesis "A couple is playing with a dog outside.", the word "dog" might be replaced by "animal" (a hypernym), which according to the rule will retain the Entailment label. This is logically correct while at the same time producing an example where the swapped word can have a vector of significant distance in embedding space, incentivising the model to discover the correct relations in the corpus and move away from the distance function artifact discussed above. To take another example, we can consider swapping the same word with a hyponym such as "corgie". Because the original hypothesis label is Entailment, according to Rule 3 the new hypothesis "A couple are playing with a corgie outside." would get assigned a Neutral label, which is again logically correct and a valid datapoint for training a model.

We can further observe, that the number of possible augmented examples that can be generated grows exponentially with the length of the base hypothesis. This is because any word in the hypothesis could be swapped by any word in its synset. In order to keep training time bounded, our implementation puts an upper bound of 10 augmented examples per hypothesis sentence. As anticipated, this approach leads to a 10-times larger dataset and therefore training time. In order to implement this we use the `map()` method of the Huggingface `Trainer` class. This has the advantage of placing the augmented examples right below the original examples, keeping the related examples together. This is beneficial for reasons discussed in the next section. Our overloaded `Trainer` class and data augmentation procedures are provided in the Appendix for ease for reference.

4.2 Contrastive Learning

Having acquired a 10 times larger training set, the question of taking maximum advantage of the training examples becomes pertinent. We decided to employ a recently published technique of Contrastive Learning [4] to further incentivise the model to learn the nuances of the decision boundary. According to the conducted research [4], one technique to achieve this is for the model to see examples that are close together and belong to a specific area of the decision boundary in the same training batch. This approach has been used in unsupervised linguistic structure prediction [12] and supervised reading comprehension [4]. To implement Contrastive Learning we disabled dataset shuffling in our `CustomTrainer` class by overloading the `_get_train_sampler()` method in the Huggingface `Trainer` class, replacing the return object by an instance of `SequentialSampler`. This way, the dataset batches provided to the model in each iteration will consist of some number of original example and their augmentations, since these are appended in place, under the original example (refer to the Data Augmentation section above). Our overloaded `Trainer` class is provided in the Appendix for ease for reference.

4.3 Hybrid Loss

Lastly, as discussed above, the Contrastive Learning approach re-focuses training in the localities of the current batch, but there lies the danger of the model learning to overfit these localities while not being able to classify correctly examples that it has not seen and are further apart in decision space. In this scenario, the model is really learning many small multinomial classification problems, and misses out on larger scale rules in the classification manifold. In order to mitigate this, we decided to combine both the Cross Entropy Loss and the NLL loss as follows (choosing $a = 0.5$):

$$L(o, l) = a\ L_{MLE}(o, l) + (1 - a)\ L_{CE}(o, l)$$

In the supervised setting, which includes our present NLI application, MLE (through the NLL Loss) is a

much stronger training signal than CE. This is because CE does not provide a learning signal for the large space of alternative premises or hypotheses that are not in the neighbourhood of the current instance bundle [4]. On the other hand, CE provides a much stronger signal for a small set of closely related and potentially confusing examples [4]. Thus the supervisory signal involves a smaller area of the decision boundary (as it will be made up of a small number of examples and their augmentations, all of which are close in decision space, as opposed to a larger number of examples all over the decision space) but will be also more complex in these locacilities, demanding a more fine-grained weight updating from the model and forcing it to learn the local properties of the decision boundary.

By combining both losses in a weighted average manner, we gain the best of both worlds. The Cross Entropy Loss ensures that part of the loss signal will be directly relevant to the shortcomings of the model in the localities of the decision boundary, enabling Contrastive Learning, while the NLL Loss will incentivise generalisation in areas that the model has not seen, learning rules that can only be inferred by looking at unrelated examples. With this arrangement we ensure a balance between the large number of examples in a small area of the decision space, and a smaller number of examples all over that space. Intuitively, this can be thought as the NLL Loss causing the largest modifications of the current decision boundary affecting more of the decision space, while the Cross Entropy Loss fine tuning local areas according to examples in each batch. To implement the compound loss we overloaded the `compute_loss()` method of the Huggingface `Trainer` class. Our overloaded `Trainer` class is provided in the Appendix for ease for reference.

Furthermore, analysis in Part 1 has shown how the specific class of Neutral hypotheses is the most confusing for the mode, comprising 85.3% of all the model's mistaken predictions. As part of our approach we tried to address this issue by treating it as equivalent to an imbalanced dataset and thus assigning a greater weight in the loss function for that particular class. However, this approach only shifted the errors to other classes (Entailment, Contradiction), and did not have any measurable effect on accuracy at best, and at worse degraded performance. Sampled results are also included in Table 3. Intuitively this might be due to the inherent ambiguity of language and meaning resulting in a smoother decision boundary between the Entailment and Contradiction classes, but a more complex one when Neutral label is involved.

5 Experimental Evaluation

5.0.1 Computational Efficiency

It is worth noting that initially we employed our Data Augmentation rules (see above) using the *TextAttack* package [9] which also incorporates *Checklist Sets* [11] and which resulted in a $\times 60$ increase in training time while we confirmed manually that it produced a smaller number of augmented examples in each iteration. This training procedure would take around 60 hours according to the Hugginface estimator using a P100 Nvidia GPU on Google Colab, and we decided to abandon it. Our proposed technique resulted in 9 hours of training and produced positive results.

5.0.2 Accuracy and Explainability

The results from the techniques we employed are summarised in Table 3. Our highest accuracy was achieved when combining all three of our major techniques, Data Augmentation, Contrastive Learning and Hybrid Loss, which amounted to around 1% improvement over the base fine-tuned language model.

Model/Methods	**Loss Func**	**Accuracy**
Baseline Model	NLL	88.92%
Augmentation Only	NLL	**89.15%**
Augmentation and Contrastive Learning	NLL	**89.54%**
...	CE	**89.51%**
...	NLL + CE	**89.82%**
...	CE (Neutral weight: 0.4)	**89.04%**
...	CE (Neutral weight: 0.5)	87.39%
...	CE (Neutral weight: 0.6)	87.17%

Table 3: Experimental results; different models were trained using the tools in the Model/Methods column.

In more detail, looking at our manually annotated adversarial examples from Table 1 above, we notice a measurable improvement in prediction accuracy (Table 4), and we can therefore conclude that it is more difficult to beat our improved model using adversarial examples, which we finally attribute to the improved training procedure having moved the model further away from artifacts and into greater reading comprehension.

Premise	Hypothesis	Label	Pred
Two women are embracing while holding to go packages.	One of the women is holding a package.	ENT	ENT
...	The packages contain food.	ENT	ENT
...	The women have bought food.	ENT	ENT
...	The women have bought lasagna.	NEU	CON
A man in a blue shirt standing in front of a garage-like structure painted with geometric designs.	A man is wearing black trousers.	NEU	NEU
A young boy in a field of flowers carrying a ball	He is carrying one ball.	ENT	ENT
...	Ball in field.	ENT	ENT
Two doctors perform surgery on patient.	The two doctors are performing brain surgery.	NEU	CON
...	The patient is having heart surgery.	NEU	CON
A white dog with long hair jumps to catch a red and green toy.	It is not a brown dog.	ENT	ENT
Kids are on a amusement ride.	Kids ride joyously an amusement ride.	ENT	ENT

Table 4: Improved Model: sampled examples of observed patterns that consitute dataset artifacts with a focus on the Entailment and Neutral classes where the model is weakest.

6 Conclusions and Future Work

In this work we proposed a combined approach of a novel data augmentation technique, contrastive learning and a hybrid loss function in order to tackle dataset artifacts. This has resulted in boosting the performance of our tranformer-based machine learning model, while having a minimal computational cost regarding training time and resources. As a natural next step, we intend to improve the data augmentation process by introducing more sophisticated rules. We firmly believe that designing a more structured way of generating closely related examples might improve performance metrics substantially. This will likely require a formal-logical treatment of the relationships between sentences when a word is swapped in a controlled manner. Similarly, coming up with a larger number of more complex rules (such as ones based on conditionals or other more sophisticated generating rules) is also promising as this would further increase the size of the training set in a meaningful way and, given the computational efficiency of our procedure, it would come at a minimal cost, with no computational cost being added on top of the training cost (which grows linearly with the size of the augmented dataset).

Acknowledgments

The authors wish to express their gratitude to Prof. Greg Durrett and the teaching staff for providing a great course in NLP in the Fall term of 2021.

References

[1] Max Bartolo, Alastair Roberts, Johannes Welbl, Sebastian Riedel, and Pontus Stenetorp. 2020. Beat the ai: Investigating adversarial human annotation for reading comprehension. *Transactions of the Association for Computational Linguistics*, 8:662–678.

[2] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

[3] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators.

[4] Dheeru Dua, Pradeep Dasigi, Sameer Singh, and Matt Gardner. 2021. Learning with instance bundles for reading comprehension. *arXiv preprint arXiv:2104.08735*.

[5] Matt Gardner, Yoav Artzi, Victoria Basmova, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, et al. 2020. Evaluating models' local decision boundaries via contrast sets. *arXiv preprint arXiv:2004.02709*.

[6] Max Glockner, Vered Shwartz, and Yoav Goldberg. 2018. Breaking nli systems with sentences that require simple lexical inferences. *arXiv preprint arXiv:1805.02266*.

[7] Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*.

[8] George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

[9] John X Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. *arXiv preprint arXiv:2005.05909*.

[10] Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. 2018. Hypothesis only baselines in natural language inference. *arXiv preprint arXiv:1805.01042*.

[11] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of nlp models with checklist. *arXiv preprint arXiv:2005.04118*.

[12] Noah A. Smith and Jason Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data.

[13] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing nlp. *arXiv preprint arXiv:1908.07125*.

[14] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing.

[15] Xiang Zhou and Mohit Bansal. 2020. Towards robustifying nli models against lexical dataset biases. *arXiv preprint arXiv:2005.04732*.

Appendix

```
class CustomTrainer(Trainer):

def __init__(self, *args, **kwargs):
    super(CustomTrainer, self).__init__(*args, **kwargs)
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    # self.loss_func = nn.CrossEntropyLoss(
      weight=torch.Tensor([0.3, 0.4, 0.3]).to(device),
      label_smoothing=0.1).to(device)
    self.loss_func = nn.CrossEntropyLoss().to(device)
    self.epsilon: float = 0.1
    self.ignore_index: int = -100

def compute_loss(self, model, inputs, return_outputs=False):

    labels = inputs.pop("labels")
    outputs = model(**inputs)

    if self.args.past_index >= 0:
        self._past = outputs[self.args.past_index]

    # Cross Entropy Loss
    logits = outputs["logits"] if isinstance(outputs, dict) else outputs[0]
    loss_ce = self.loss_func(logits, labels)

    # Default Trainer NLLoss
    log_probs = -F.log_softmax(logits, dim=-1)
    if labels.dim() == log_probs.dim() - 1:
        labels = labels.unsqueeze(-1)
    padding_mask = labels.eq(self.ignore_index)
    labels = torch.clamp(labels, min=0)
    nll_loss = log_probs.gather(dim=-1, index=labels)
    smoothed_loss = log_probs.sum(dim=-1, keepdim=True, dtype=torch.float32)
    nll_loss.masked_fill_(padding_mask, 0.0)
    smoothed_loss.masked_fill_(padding_mask, 0.0)
    num_active_elements = padding_mask.numel() - padding_mask.long().sum()
    nll_loss = nll_loss.sum() / num_active_elements
    smoothed_loss = smoothed_loss.sum() / (num_active_elements * log_probs.shape[-1])
    loss_nll = (1 - self.epsilon) * nll_loss + self.epsilon * smoothed_loss
    # Weighted loss
    loss = 0.5 * loss_nll + 0.5 * loss_ce

    return (loss, outputs) if return_outputs else loss

def _get_train_sampler(self):
    # By overriding this we create a sequential batch sampling method, disabling
  shuffling in training.
    # This will ensure that perturbed examples are seen together during training because
  they have
    # adjacent indices in the Dataset, enabling Contrastive Learning (ref. paper by Dua
  et al)
    return SequentialSampler(data_source=self.train_dataset)
```

CustomTrainer(Trainer) class

```
def augment_data_word_swap(example):
    new_hypos, new_labels = augmenting_swap(example)
    cat_premises = example["premise"] * len(new_labels)
    return {"premise": cat_premises, "hypothesis": new_hypos, "label": new_labels}

def query_wordnet(word):
    synonyms, antonyms, hypernyms, hyponyms = [], [], [], []

    for synset in wordnet.synsets(word):
        for lemma in synset.lemmas():
            synonyms.append(lemma.name())
            if lemma.antonyms():
                antonyms.append(lemma.antonyms()[0].name())
```

```
        hypernyms.extend([x.name().partition(".")[0] for x in synset.hypernyms()])
        hyponyms.extend([x.name().partition(".")[0] for x in synset.hyponyms()])

    return synonyms, antonyms, hypernyms, hyponyms

def augmenting_swap(example, max_returned=10):
    """
    A function to augment a hypothesis based on word replacement, using the above rules.
    """
    hypo_sentence = example["hypothesis"][0]
    hypo = hypo_sentence.split(" ")
    label = example["label"][0]

    new_hypos = []
    new_labels = []
    num_aug_exs = 0

    if label == 0:
        # Entailment
        for idx, word in enumerate(hypo):

            synonyms, antonyms, hypernyms, hyponyms = query_wordnet(word)

            if synonyms:
                synonym = random.choice(synonyms)
                new_hypo = hypo[:idx] + [synonym]
                if idx != len(hypo) - 1:
                    new_hypo += hypo[(idx+1):]
                new_hypos += [' '.join(new_hypo)]
                new_labels.append(0)
                num_aug_exs += 1
                if num_aug_exs == max_returned:
                    break

            if hypernyms:
                hypernym = random.choice(hypernyms)
                new_hypo = hypo[:idx] + [hypernym]
                if idx != len(hypo) - 1:
                    new_hypo += hypo[(idx+1):]
                new_hypos += [' '.join(new_hypo)]
                new_labels.append(0)
                num_aug_exs += 1
                if num_aug_exs == max_returned:
                    break

            if antonyms:
                antonym = random.choice(antonyms)
                new_hypo = hypo[:idx] + [antonym]
                if idx != len(hypo) - 1:
                    new_hypo += hypo[(idx+1):]
                new_hypos += [' '.join(new_hypo)]
                new_labels.append(2)
                num_aug_exs += 1
                if num_aug_exs == max_returned:
                    break

            if hyponyms:
                hyponym = random.choice(hyponyms)
                new_hypo = hypo[:idx] + [hyponym]
                if idx != len(hypo) - 1:
                    new_hypo += hypo[(idx+1):]
                new_hypos += [' '.join(new_hypo)]
                new_labels.append(1)
                num_aug_exs += 1
                if num_aug_exs == max_returned:
                    break

    elif label == 1:
```

```
        # Neutral
        for idx, word in enumerate(hypo):

            synonyms, antonyms, hypernyms, hyponyms = query_wordnet(word)

            if synonyms:
                synonym = random.choice(synonyms)
                new_hypo = hypo[:idx] + [synonym]
                if idx != len(hypo) - 1:
                    new_hypo += hypo[(idx+1):]
                new_hypos += [' '.join(new_hypo)]
                new_labels.append(1)
                num_aug_exs += 1
                if num_aug_exs == max_returned:
                    break

            if hypernyms:
                hypernym = random.choice(hypernyms)
                new_hypo = hypo[:idx] + [hypernym]
                if idx != len(hypo) - 1:
                    new_hypo += hypo[(idx+1):]
                new_hypos += [' '.join(new_hypo)]
                new_labels.append(1)
                num_aug_exs += 1
                if num_aug_exs == max_returned:
                    break

    else:
        # Contradiction
        for idx, word in enumerate(hypo):

            synonyms, antonyms, hypernyms, hyponyms = query_wordnet(word)

            if synonyms:
                synonym = random.choice(synonyms)
                new_hypo = hypo[:idx] + [synonym]
                if idx != len(hypo) - 1:
                    new_hypo += hypo[(idx+1):]
                new_hypos += [' '.join(new_hypo)]
                new_labels.append(2)
                num_aug_exs += 1
                if num_aug_exs == max_returned:
                    break

            if hypernyms:
                hypernym = random.choice(hypernyms)
                new_hypo = hypo[:idx] + [hypernym]
                if idx != len(hypo) - 1:
                    new_hypo += hypo[(idx+1):]
                new_hypos += [' '.join(new_hypo)]
                new_labels.append(2)
                num_aug_exs += 1
                if num_aug_exs == max_returned:
                    break

            if hyponyms:
                hyponym = random.choice(hyponyms)
                new_hypo = hypo[:idx] + [hyponym]
                if idx != len(hypo) - 1:
                    new_hypo += hypo[(idx+1):]
                new_hypos += [' '.join(new_hypo)]
                new_labels.append(2)
                num_aug_exs += 1
                if num_aug_exs == max_returned:
                    break

    return new_hypos, new_labels
```

Data Augmentation Procedure