INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY HYDERABAD

ADVERSARIAL NLI

ASHNA DUA (2021101072)
ASHNA.DUA@STUDENTS.IIIT.AC.IN
VANSHITA MAHAJAN (2021101102)
VANSHITA.MAHAJAN@STUDENTS.IIIT.AC.IN
PRISHA (2021101075)
PRISHA.KUMAR@RESEARCH.IIIT.AC.IN

NOVEMBER 20, 2024

Contents

1

Problem Statement

1.1 Introduction

Large-scale pre-trained language models like BERT and RoBERTa have significantly advanced performance in numerous Natural Language Processing (NLP) tasks, establishing themselves as essential tools in the field. Despite their remarkable capabilities, these models exhibit vulnerabilities when exposed to adversarial textual inputs.

This creates a critical challenge, as even small perturbations in the input can lead to dramatic changes in model output, compromising their reliability in sensitive applications. Traditional fine-tuning approaches have fallen short in mitigating these weaknesses, leaving room for new, more resilient strategies.

1.2 Why is NLI important

NLI serves as a critical benchmark for assessing a model's reasoning capabilities and understanding of language semantics. Success in NLI demonstrates a model's ability to:

- Handle complex reasoning.
- Resolve ambiguities.
- Identify logical inconsistencies or implications.
  It is considered a foundational task for broader applications, including question answering, summarization, and conversational AI.

Consider the following example :


```
{
    "uid": "8a91e1a2-9a32-4fd9-b1b6-bd2ee2287c8f",
    "premise": "Javier Torres (born May 14, 1988 in Artesia, California) is
    an undefeated Mexican American professional boxer in the Heavyweight division.
        Torres was the second rated U.S. amateur boxer in the Super Heavyweight
        division and a member of the Mexican Olympic team.",
    "hypothesis": "Javier was born in Mexico",
    "label": "c",
    "reason": "The paragraph states that Javier was born in the California, US."
}
```

In this project, we tackle the problem from an information-theoretic perspective. Our goal is to develop a robust fine-tuning framework that not only preserves the models' high performance but also increases their resistance to adversarial examples. By introducing mutual-information-based regularizers, we aim to reduce noise in the feature representations while reinforcing critical features, thus ensuring the model can maintain accuracy under both standard and adversarial training conditions.

1.3 Dataset: Adversarial Natural Language Inference (ANLI)

The Adversarial Natural Language Inference (ANLI) dataset is a large-scale, challenging benchmark designed for natural language understanding (NLU) tasks, specifically natural language inference (NLI). ANLI was collected using a unique iterative, adversarial human-and-model-in-the-loop process aimed at creating examples that expose model weaknesses, ensuring continuous learning and robustness.

1.3.1 Collection Process

The dataset was collected in three rounds (A1, A2, and A3), with increasing levels of difficulty in each round. The data collection followed the **Human-And-Model-in-the-Loop Enabled Training (HAMLET)** process, which involves human annotators attempting to create hypothesis examples (given a context and a target label) that state-of-the-art models misclassify. The process includes:

- **Context Selection:** Annotators were provided with short, multi-sentence passages (context) from sources such as Wikipedia, news articles, and other genres.
- **Hypothesis Generation:** Annotators wrote hypotheses corresponding to a target label (*entailment, contradiction, or neutral*) that aimed to fool the model into incorrect classifications.
- **Verification:** Hypotheses that fooled the model were then verified by human annotators to ensure that the label was correct and the example was valid.

The dataset includes both verified correct examples and those where the model failed, to continuously challenge the evolving model.

1.3.2 Rounds of Data Collection

Each round in ANLI represents a progressively more difficult phase of model training and evaluation:

- **A1 (Round 1):** The first round used contexts sampled from Wikipedia and the HotpotQA dataset. A BERT-Large model was employed as the baseline model for this round.
- **A2 (Round 2):** In the second round, a stronger RoBERTa model was used, incorporating the training data from Round 1 along with new data.

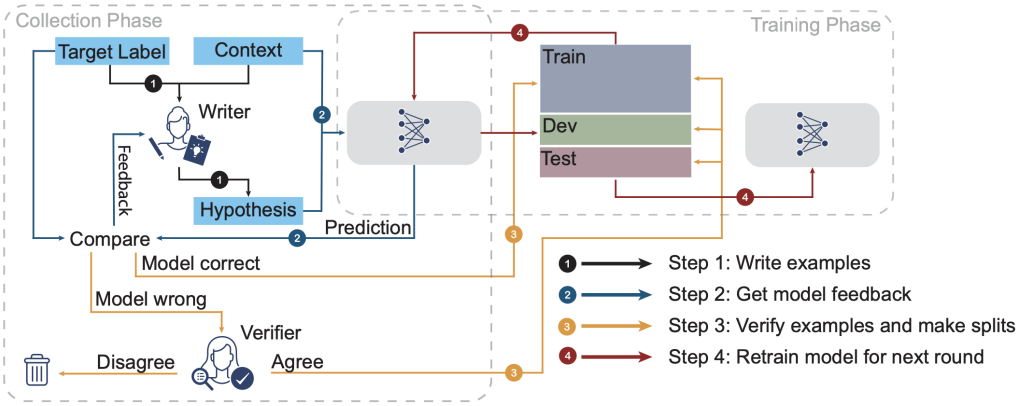

Figure 1: Adversarial NLI data collection via human-and-model-in-the-loop enabled training (HAMLET). The four steps make up one round of data collection. In step 3, model-correct examples are included in the training set; development and test sets are constructed solely from model-wrong verified-correct examples.

Figure 1.1: *Rounds for Data Collection*

- **A3 (Round 3):** The third round involved a more diverse set of contexts from domains such as news articles, fiction, spoken text from court transcripts, and procedural texts, making it the most challenging.

The increasing difficulty in each round stems from the iterative nature of the data collection process, where models are strengthened and tested on progressively harder examples generated by human annotators.

To prevent overfitting to specific annotator styles or model characteristics, the test set for each round includes an **exclusive subset**. This exclusive subset consists of examples from annotators who did not contribute to the training data, ensuring that models are tested on unseen data from entirely new annotators. This prevents models from learning biases inherent to specific annotators and ensures better generalization across varied inputs.

1.3.3 Dataset Statistics

ANLI contains a total of 162,865 training examples collected over three rounds, with 3,200 examples in both the development and test sets for each round. The dataset is balanced to ensure no overfitting occurs due to model bias or specific genre characteristics. Table 1.1 provides a detailed breakdown of the dataset's statistics.

Round	Train Size	Dev Size	Test Size
A1	16,946	1,000	1,000
A2	45,460	1,000	1,000
A3	100,459	1,200	1,200

Table 1.1: *Dataset statistics for ANLI.*

1.4 Dataset Overview

The dataset used for this project is the **Adversarial Natural Language Inference (ANLI)** dataset, which is specifically designed to test the resilience of NLI models against adversarial examples. ANLI is composed of three rounds, with each subsequent round containing examples that are progressively more difficult for models to handle:

- **ANLI Round 1 (r1):** Contains adversarial examples that exploit basic weaknesses in NLI models.
- **ANLI Round 2 (r2):** More complex than r1, r2 includes examples targeting the models trained in the previous round.
- **ANLI Round 3 (r3):** The most difficult round, r3 has examples designed to challenge the deep reasoning abilities of NLI models.

In this analysis, the labels used in the dataset are defined as follows:

- 0: entailment
- 1: neutral
- 2: contradiction

1.5 Exploratory Data Analysis

	uid	premise	hypothesis	label	reason
0	0fd0abfb-659e-4453-b196-c3a64d2d8267	The Parma trolleybus system (Italian: "Rete fi...	The trolleybus system has over 2 urban routes	0	NaN
1	7ed72ff4-40b7-4f8a-b1b9-6c612aa62c84	Alexandra Lendon Bastedo (9 March 1946 – 12 Ja...	Sharron Macready was a popular character throu...	1	NaN
2	5d2930a3-62ac-485d-94d7-4e36cbbcd7b5	Alexandra Lendon Bastedo (9 March 1946 – 12 Ja...	Bastedo didn't keep any pets because of her vi...	1	NaN
3	324db753-ddc9-4a85-a825-f09e2e5aebdd	Alexandra Lendon Bastedo (9 March 1946 – 12 Ja...	Alexandra Bastedo was named by her mother.	1	NaN
4	4874f429-da0e-406a-90c7-22240ff3ddf8	Alexandra Lendon Bastedo (9 March 1946 – 12 Ja...	Bastedo cared for all the animals that inhabit...	1	NaN

Figure 1.2: *Sample Train Data*

	uid	premise	hypothesis	label	reason
0	4aae63a8-fcf7-406c-a2f3-50c31c5934a9	Ernest Jones is a British jeweller and watchma...	The first Ernest Jones store was opened on the...	0	The first store was opened in London, which is...
1	c577b92c-78fb-4e1d-ae1d-34133609c142	Old Trafford is a football stadium in Old Traf...	There are only 10 larger football stadiums in ...	0	The text says that it is the 11th largest foot...
2	26936cd9-1a5a-4a2b-9fca-899d61880ca0	Magnus is a Belgian joint dance project of Tom...	"The body gave you everything" album was not r...	0	it was released on March 29, 2004. "not this b...
3	cd977941-273b-4748-a5d2-6c7234a2a302	Shadowboxer is a 2005 crime thriller film dire...	Shadowboxer was written and directed by Lee Da...	1	It is not know who wrote the Shadowboxer. The ...
4	1a9eae8f-27d9-47ba-80b8-7d1402ee524a	Takaaki Kajita (梶田 隆章 , Kajita Takaaki) is a ...	Arthur B. McDonald is a Japanese physicist, kn...	2	Arthur B. McDonald is Canadian in the context.

Figure 1.3: *Sample Test Data*

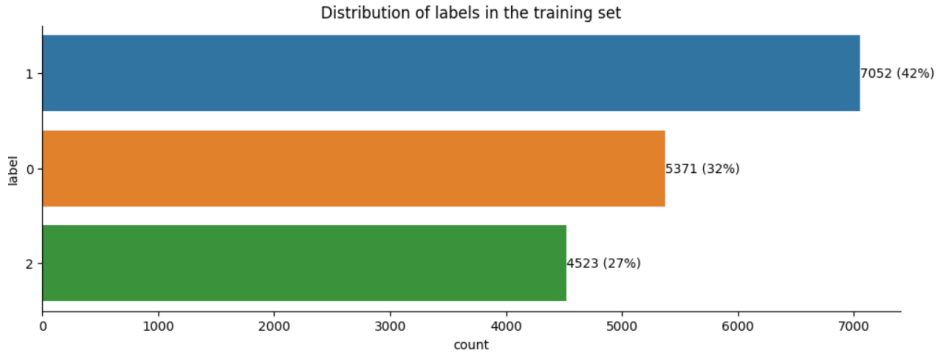

Figure 1.4: *Label Distribution*

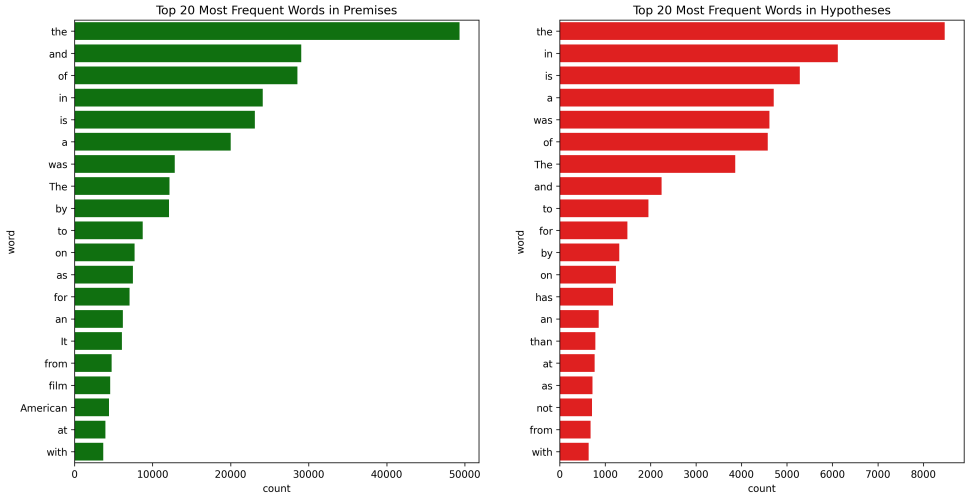

Figure 1.5: *Top 20 most frequent words in Train and Test*

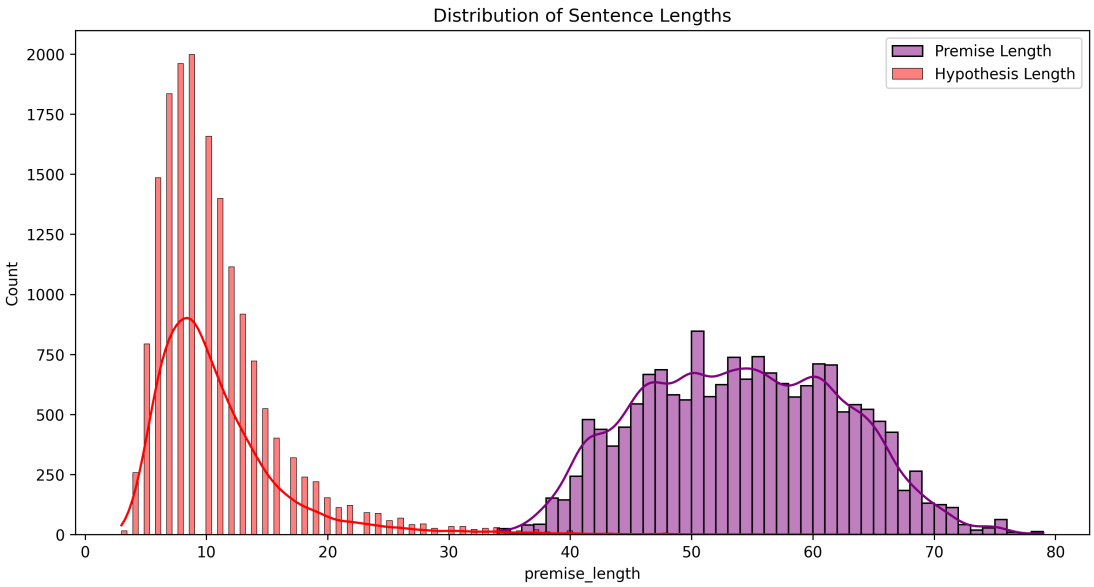

Figure 1.6: *Sentence length distribution in premise and hypothesis*

1.5.1 Comparison to Other Datasets

ANLI represents an improvement over prior NLI datasets like SNLI and MNLI by focusing on more complex and longer contexts. While SNLI used image captions as contexts, ANLI sources multi-sentence passages from various genres, adding greater complexity. The human-and-model-in-the-loop process also makes ANLI more robust against overfitting, allowing models trained on this dataset to better handle real-world NLI tasks by addressing more nuanced and challenging examples.

2

Literature Review

Textual adversarial attacks have emerged as a significant concern for language models, particularly those based on pre-trained transformers like BERT and RoBERTa. The existing adversarial attacks predominantly focus on word-level manipulations. Ebrahimi et al., 2018 were among the first to introduce a white-box, gradient-based approach to search for adversarial word or character substitutions. Subsequent research (Alzantot et al., 2018; Ren et al., 2019; Zang et al., 2020; Jin et al., 2020) sought to refine these methods by restricting the perturbation search space and using Part-of-Speech (POS) checking to ensure that the adversarial examples maintain a natural appearance to human observers.

To defend against such adversarial attacks, three primary defense strategies have been proposed:

- **Adversarial Training**: A popular and practical defense mechanism, adversarial training augments the model with adversarial examples. Some works (C. Zhu et al., 2020; Jiang et al., 2020; Liu et al., 2020; Gan et al., 2020) employ PGD-based (Projected Gradient Descent) attacks in the embedding space to generate adversarial examples for data augmentation or use virtual adversarial training to regularize the objective function. However, the primary limitation of this approach is its dependency on the threat model, which makes it less effective against unseen attacks.
- **Interval Bound Propagation (IBP)**: Introduced by Dvijotham et al., 2018, IBP is designed to handle worst-case perturbations theoretically. Recent research (Huang et al., 2019; Jia et al., 2017) has extended this approach to the NLP domain, applying IBP to certify the robustness of language models. However, IBP relies on strong assumptions about model architecture, making it difficult to apply to modern transformer-based models.
- **Randomized Smoothing**: Cohen et al., 2019 introduced randomized smoothing as a method to guarantee robustness in l_2 norm by adding Gaussian noise to smooth the classifier. Ye et al., 2020 adapted this idea to the NLP domain by replacing Gaussian noise with synonym words, ensuring that adversarial word substitutions

within predefined synonym sets maintain robustness. However, ensuring the completeness of the synonym set presents a challenge in practice.

S. Zhu et al., 2020 addressed this issue by proposing a robust representation learning approach that considers the mutual-information perspective in the context of worst-case perturbation. However, their work primarily focused on continuous input spaces like those in computer vision. In contrast, InfoBERT adopts a novel information-theoretic perspective that is designed for both standard and adversarial training in the discrete input space typical of language models.

The existing literature highlights the progress made in defending against textual adversarial attacks and improving the robustness of language models through various techniques. However, most prior work has focused on either adversarial training or theoretical robustness certifications, both of which come with limitations in terms of model adaptability or threat model assumptions. InfoBERT takes a step forward by integrating mutual-information-based regularizers into the fine-tuning process, which not only enhances robustness in adversarial settings but also aligns with standard training requirements for discrete NLP input spaces. This combination of information theory and adversarial training positions InfoBERT as a unique contribution to the ongoing effort to make language models more robust and reliable.

3

Baselines

3.1 Introduction

In this chapter, we explore the performance of baseline models on widely-used Natural Language Inference (NLI) datasets, including SNLI, MNLI, and ANLI. These datasets serve as benchmarks for evaluating the generalization and robustness of pre-trained language models. This analysis also provides a reference point for subsequent experiments and helps us assess the effectiveness of advanced techniques, such as mutual-information-based regularization, introduced in later chapters.

These were the model training configurations used to obtain the baseline results for BERT and RoBERTa base. The model training was conducted using a Python script with specified parameters to ensure optimal performance. Below is a detailed description of the configuration used for training.

Note that during training, we varied the model class name, training weights, and the training and evaluation data using ANLI, SNLI, or MNLI.

- **Model Class Name**: The training utilized the `bert-base` architecture, known for its efficiency in various natural language processing tasks.
- **Nodes and GPUs**: The training was executed with `1` node and `2` GPUs, with the `-single_gpu` flag indicating that training was performed on a single GPU per process.
- **Node Rank**: The node rank was set to `0` for this training, designating the primary node in a distributed training setup.
- **Sequence Length**: A maximum sequence length of `156` tokens was used, ensuring that inputs were appropriately truncated or padded to this length.
- **Gradient Accumulation**: The parameter `-gradient_accumulation_steps` was set to `1`, which indicates that gradient updates were performed without additional accumulation.
- **Batch Sizes**: The per-GPU batch size for training and evaluation was `16`, balancing memory usage and computation speed.
- **Prediction Saving**: The flag `-save_prediction` was enabled, allowing model predictions to be stored for further analysis.

- **Training Data**: The model was trained using three datasets: `anli_r1_train:none`, `anli_r2_train:none`, and `anli_r3_train:none`, with assigned weights of `5`, `10`, and `5` respectively to balance their contributions.
- **Evaluation Data**: For evaluation, data from `anli_r1_dev:none`, `anli_r2_dev:none`, and `anli_r3_dev:none` was used.
- **Evaluation Frequency**: The model was evaluated every `2000` training steps to monitor performance and adjust training as needed.
- **Experiment Name**: The experiment was labeled as `"ANLI-R1|ANLI-R2|ANLI-R3"` for easy identification and tracking of results.

3.2 Results of BERT-Uncased-Base

We used BERT base uncased to analyze the performance on the datasets, particularly ANLI R1, R2, R3 as mentioned above. The following results were obtained after running 1 epoch (approximately 65,000 iterations):

Round	Accuracy
R1	0.508 (50.8%)
R2	0.4434 (44.34%)
R3	0.4014 (40.92%)

3.3 Results of Roberta-Base

We used Roberta base uncased to analyze the performance on the datasets, particularly ANLI R1, R2, R3 as mentioned above. The following results were obtained after running 1 epoch (approximately 65,000 iterations):

Round	Accuracy
R1	0.575 (57.5%)
R2	0.452 (45.2%)
R3	0.4292 (42.92%)

4

INFOBERT

InfoBERT was introduced to enhance the robustness of large-scale pre-trained language models such as BERT and RoBERTa, which, despite their strong performance in various NLP tasks, are vulnerable to adversarial attacks. Adversarial examples—small, human-imperceptible perturbations in input data—can lead to significant errors in model predictions. Existing defenses, including adversarial training, face limitations like the challenge of unknown threat models and high computational costs. InfoBERT addresses these vulnerabilities by incorporating an information-theoretic perspective, using mutual-information (MI)-based regularizers to fine-tune both local (word-level) and global (sentence-level) feature representations. This approach enhances model robustness while maintaining performance on standard datasets.

InfoBERT integrates two innovative mutual-information-based regularizers:

4.1 Information Bottleneck (IB) Regularizer

Inspired by the Information Bottleneck principle, this regularizer aims to extract sufficient statistics for task performance while filtering out redundant and noisy information. The IB principle formulates the goal of deep learning as an information-theoretic trade-off between representation compression and predictive power, represented as:

The goal of IB is to find a compressed representation T of the input X such that:

- It retains **minimal information** about the input X (reducing irrelevant details).
- It retains **maximal information** relevant to predicting the target Y.

This trade-off is formalized as:

$$\mathcal{L}_{\text{IB}} = I(Y;T) - \beta I(X;T), \tag{1}$$

where:

- $I(X;T)$: Mutual information between the input X and the representation T (compression).

- $I(Y;T)$: Mutual information between the representation T and the target Y (relevance).
- β: Trade-off parameter controlling the balance between compression and relevance.

$I(Y;T)$ is defined as:

$$I(Y;T) = \int p(y,t) \log \frac{p(y,t)}{p(y)p(t)} \, dy \, dt. \tag{2}$$

Since Eq. (2) is intractable, we instead use the lower bound:

$$I(Y;T) \geq \int p(y,t) \log q_\psi(y \mid t) \, dy \, dt, \tag{3}$$

where $q_\psi(y \mid t)$ is the variational approximation learned by a neural network parameterized by ψ for the true distribution $p(y \mid t)$. This indicates that maximizing the lower bound of the first term of $I(Y;T)$ is equivalent to minimizing the task cross-entropy loss $\ell_{\text{task}} = H(Y \mid T)$.

To derive a tractable lower bound of IB, we here use an upper bound of $I(X;T)$:

$$I(X;T) \leq \int p(x,t) \log p(t \mid x) \, dx \, dt - \int p(x)p(t) \log p(t \mid x) \, dx \, dt. \tag{4}$$

By combining Eq. (3) and (4), we can maximize the tractable lower bound $\hat{\mathcal{L}}_{\text{IB}}$ of IB in practice by:

$$\hat{\mathcal{L}}_{\text{IB}} = \frac{1}{N} \sum_{i=1}^{N} \left[\log q_\psi\big(y^{(i)} \mid t^{(i)}\big) \right] - \frac{\beta}{N} \sum_{i=1}^{N} \left[\log \big(p(t^{(i)} \mid x^{(i)})\big) \right] - \frac{1}{N} \sum_{j=1}^{N} \log \big(p(t^{(j)} \mid x^{(i)})\big) \tag{5}$$

with data samples $\{x^{(i)}, y^{(i)}\}_{i=1}^{N}$, where q_ψ can represent any classification model (e.g., BERT), and $p(t \mid x)$ can be viewed as the feature extractor $f_\theta : \mathcal{X} \to \mathcal{T}$, where $\mathcal{X}$ and $\mathcal{T}$ are the support of the input source X and extracted feature T, respectively.

The above is a general implementation of the IB objective function. In InfoBERT, we consider T as the features consisting of the local word-level features after the BERT embedding layer f_θ. The following BERT self-attentive layers along with the linear classification head serve as $q_\psi(y \mid t)$ that predicts the target Y given representation T.

4.2 Anchored Feature Extraction (AFE) Regularizer

The Anchored Feature Extraction (AFE) Regularizer is designed to enhance robustness by aligning global representations with stable, task-relevant local features, termed **anchored features**. Unlike traditional methods that equally weigh all local features, the AFE selectively identifies robust and informative features resistant to adversarial perturbations while filtering out noisy or uninformative ones.

The process begins with local feature extraction, where each word in the input sequence $X = [X_1, X_2, \ldots, X_n]$ is mapped to its corresponding high-dimensional embedding $T = [T_1, T_2, \ldots, T_n]$ through a feature extractor f_θ. Local anchored features are

identified by examining the perturbation sensitivity of the embeddings. Specifically, adversarial perturbations δ are introduced in the embedding space, and the gradient $g(\delta) = \nabla_\delta \ell_{\text{task}}(q_\psi(T + \delta), Y)$ is computed with respect to the task loss. Features with moderate sensitivity, i.e., those whose perturbation magnitude falls within predefined thresholds $[c_l, c_h]$, are considered stable and informative.

Formally, anchored features T_{k_j} are selected from local features T_i such that:

$$c_l \leq \frac{i}{n} \leq c_h,$$

where n is the sequence length, and the thresholds c_l and c_h are empirically determined.

Once anchored features are identified, the AFE aligns them with global sentence-level representations Z (e.g., the final-layer [CLS] embedding) by maximizing their mutual information $I(T_{k_j}; Z)$. This alignment ensures that global representations emphasize robust and useful information while downplaying contributions from nonrobust or uninformative features. The resulting objective function integrates AFE with the IB regularizer as follows:

$$\mathcal{L}_{\text{AFE}} = I(Y; T) - n\beta \sum_{i=1}^{n} I(X_i; T_i) + \alpha \sum_{j=1}^{M} I(T_{k_j}; Z),$$

where M is the number of anchored features, and α is a trade-off parameter controlling the importance of AFE.

In practice, mutual information $I(T_{k_j}; Z)$ is approximated using the InfoNCE lower bound:

$$\hat{I}(T_{k_j}; Z) = \mathbb{E}_P \left[g_\omega(T_{k_j}, Z) - \mathbb{E}_{\tilde{P}} \log \sum_{T'_{k_j}} \exp(g_\omega(T'_{k_j}, Z)) \right],$$

where $g_\omega(\cdot, \cdot)$ is a critic function parameterized by a neural network, and $P, \tilde{P}$ denote distributions over positive and negative samples, respectively.

By emphasizing robust and stable features, the AFE Regularizer significantly enhances the robustness of global representations, particularly in adversarial scenarios, without compromising performance on benign data.

Algorithm 1 - Local Anchored Feature Extraction. This algorithm takes in the word local features and returns the index of local anchored features.

1: **Input:** Word local features t, upper and lower threshold c_h and c_l
2: $\delta \leftarrow 0$ *// Initialize the perturbation vector δ*
3: $g(\delta) = \nabla_\delta \ell_{\text{task}}(q_\psi(t + \delta), y)$ *// Perform adversarial attack on the embedding space*
4: Sort the magnitude of the gradient of the perturbation vector from $||g(\delta)_1||_2, ||g(\delta)_2||_2, ..., ||g(\delta)_n||_2$ into $||g(\delta)_{k_1}||_2, ||g(\delta)_{k_2}||_2, ..., ||g(\delta)_{k_n}||_2$ in ascending order, where z_i corresponds to its original index.
5: **Return:** $k_i, k_{i+1}, ..., k_j$, where $c_l \leq \frac{i}{n} \leq \frac{j}{n} \leq c_h$.

4.2.1 Example Application

Consider a sentence where:

- Vulnerable words like "quickly" or "easily" are prone to change under adversarial substitution.
- Stopwords like "and" or punctuation marks provide little useful information.

By identifying and filtering these, the representation focuses on anchored features that are stable and contribute meaningfully to downstream tasks.

Anchored feature extraction thus aligns sentence-level global representations with local features that are both robust and informative, ensuring better performance on adversarially robust language models.

4.3 Implementation

```
def get_mi_optimizers(
        self, num_training_steps: int
    ) -> Tuple[torch.optim.Optimizer, torch.optim.lr_scheduler.LambdaLR]:
        """
        Setup the optimizer and the learning rate scheduler.

        We provide a reasonable default that works well.
        If you want to use something else, you can pass a tuple in the Trainer's
        init,
        or override this method in a subclass.
        """
        if self.optimizers is not None:
            return self.optimizers
        # Prepare optimizer and schedule (linear warmup and decay)
        optimizer_grouped_parameters = [
            {
                "params": list(self.mi_estimator.parameters()),
                "weight_decay": self.args.weight_decay,
            },
        ]

        optimizer = AdamW(optimizer_grouped_parameters,
        lr=self.args.learning_rate, eps=self.args.adam_epsilon)
        scheduler = get_linear_schedule_with_warmup(
            optimizer, num_warmup_steps=self.args.warmup_steps,
            num_training_steps=num_training_steps
        )
        return optimizer, scheduler
```

```
def _train_mi_upper_estimator(self, outputs, inputs=None):
    hidden_states = outputs[2]  # need to set config.output_hidden = True
    last_hidden, embedding_layer = hidden_states[-1], hidden_states[0]  #
    embedding layer: batch x seq_len x 768
    sentence_embedding = last_hidden[:, 0]  # batch x 768
    if self.mi_upper_estimator.version == 0:
        embedding_layer = torch.reshape(embedding_layer,
        [embedding_layer.shape[0], -1])
        return self.mi_upper_estimator.update(embedding_layer,
        sentence_embedding)
    elif self.mi_upper_estimator.version == 1:
        return self.mi_upper_estimator.update(embedding_layer, last_hidden)
    elif self.mi_upper_estimator.version == 2:
        return self.mi_upper_estimator.update(embedding_layer,
        sentence_embedding)
    elif self.mi_upper_estimator.version == 3:
        embeddings = []
        lengths = self.get_seq_len(inputs)
        for i, length in enumerate(lengths):
            embeddings.append(embedding_layer[i, :length])
        embeddings = torch.cat(embeddings)  # [-1, 768]
        return self.mi_upper_estimator.update(embedding_layer, embeddings)
```

```
def _get_local_robust_feature_regularizer(self, outputs, local_robust_features):
    hidden_states = outputs[2]  # need to set config.output_hidden = True
    last_hidden, embedding_layer = hidden_states[-1], hidden_states[0]  #
    embedding layer: batch x seq_len x 768
    sentence_embeddings = last_hidden[:, 0]  # batch x 768
    local_embeddings = []
    global_embeddings = []
    for i, local_robust_feature in enumerate(local_robust_features):
        for local in local_robust_feature:
            local_embeddings.append(embedding_layer[i, local])
            global_embeddings.append(sentence_embeddings[i])

    lower_bounds = []
    from sklearn.utils import shuffle
    local_embeddings, global_embeddings = shuffle(local_embeddings,
    global_embeddings, random_state=self.args.seed)
    for i in range(0, len(local_embeddings), self.args.train_batch_size):
        local_batch = torch.stack(local_embeddings[i: i +
        self.args.train_batch_size])
        global_batch = torch.stack(global_embeddings[i: i +
        self.args.train_batch_size])
        lower_bounds += self.mi_estimator(local_batch, global_batch),
    return -torch.stack(lower_bounds).mean()
```

```
def _eval_mi_upper_estimator(self, outputs, inputs=None):
    hidden_states = outputs[2]  # need to set config.output_hidden = True
    last_hidden, embedding_layer = hidden_states[-1], hidden_states[0]  #
    embedding layer: batch x seq_len x 768
    sentence_embedding = last_hidden[:, 0]  # batch x 768
    if self.mi_upper_estimator.version == 0:
        embedding_layer = torch.reshape(embedding_layer,
        [embedding_layer.shape[0], -1])
        return self.mi_upper_estimator.mi_est(embedding_layer,
        sentence_embedding)
    elif self.mi_upper_estimator.version == 1:
        return self.mi_upper_estimator.mi_est(embedding_layer, last_hidden)
    elif self.mi_upper_estimator.version == 2:
        return self.mi_upper_estimator.mi_est(embedding_layer,
        sentence_embedding)
    elif self.mi_upper_estimator.version == 3:
        embeddings = []
        lengths = self.get_seq_len(inputs)
        for i, length in enumerate(lengths):
            embeddings.append(embedding_layer[i, :length])
        embeddings = torch.cat(embeddings)  # [-1, 768]
        return self.mi_upper_estimator.mi_est(embedding_layer, embeddings)
```

```
def feature_ranking(self, grad, cl=0.5, ch=0.9):
    n = len(grad)
    import math
    lower = math.ceil(n * cl)
    upper = math.ceil(n * ch)
    norm = torch.norm(grad, dim=1)  # [seq_len]
    _, ind = torch.sort(norm)
    res = []
    for i in range(lower, upper):
        res += ind[i].item(),
    return res
```

```
def local_robust_feature_selection(self, inputs, grad, input_ids=None):
    grads = []
    lengths = self.get_seq_len(inputs)
    for i, length in enumerate(lengths):
        grads.append(grad[i, :length])
    indices = []
    nonrobust_indices = []
    for i, grad in enumerate(grads):
        indices.append(self.feature_ranking(grad, self.args.cl, self.args.ch))
        nonrobust_indices.append([x for x in range(lengths[i]) if x not in
        indices])
    return indices, nonrobust_indices
```

4.4 Results of InfoBERT with RoBERTa base

We used RoBERTa base uncased to analyze the performance on the datasets, particularly ANLI R1, R2, R3 as mentioned above. The following results were obtained after running 1 epoch (approximately 5090 iterations):

Round	Accuracy
R1 Dev	0.694 (69.4%)
R1 Test	0.683 (68.3%)
R2 Dev	0.487 (48.7%)
R2 Test	0.482 (48.2%)
R3 Dev	0.446 (44.6%)
R3 Test	0.451 (45.1%)

Code for the implementation can be found here: Link

5

Ablation Study

This chapter presents a thorough ablation study aimed at evaluating the performance of the InfoBERT model and its variants by systematically removing specific regularizers. The study primarily focuses on three key configurations:

- **InfoBERT:** The base InfoBERT model incorporates both **Mutual Information (MI)** and **Information Bottleneck (IB)** regularizers, which work together to enhance the model's ability to generalize and capture informative representations. These regularizers act as a form of control over the amount of information retained from the input, promoting more efficient encoding of essential features while filtering out unnecessary noise.
- **No MI Regularizer:** In this variant, the MI regularizer is removed, isolating the effect of the IB regularizer. By omitting the MI regularizer, this configuration allows us to assess the individual contribution of the IB regularizer in improving model performance, especially in terms of adversarial robustness and generalization capabilities.
- **No IB Regularizer:** Here, the IB regularizer is removed while retaining the MI regularizer. This setup tests the effect of excluding the IB regularizer, which is designed to mitigate overfitting by constraining the mutual information between the learned representations and the model's inputs.

The experiments in this study are conducted on various benchmark datasets, focusing not only on standard tasks but also on the model's adversarial robustness—a critical factor in real-world applications. The datasets used include the ANLI, MNLI (Multi-Genre Natural Language Inference), and SNLI (Stanford Natural Language Inference) benchmarks.

5.1 Methodology for Ablation Study: Without MI Regularizer

To evaluate the impact of the Mutual Information (MI) regularizer on training dynamics and model performance, we conducted an experiment where the MI regularizer was

disabled. This was achieved by setting the parameter `model_args.version` to `-1`. The corresponding implementation logic is as follows:

```
if model_args.version >= 0:
    if model_args.version == 0:
        mi_upper_estimator = CLUB(config.hidden_size * data_args.max_seq_length,
        config.hidden_size, beta=model_args.beta).to(training_args.device)
        mi_upper_estimator.version = 0
        mi_estimator = None
    elif model_args.version == 1:
        mi_upper_estimator = CLUB(config.hidden_size, config.hidden_size,
        beta=model_args.beta).to(training_args.device)
        mi_upper_estimator.version = 1
        mi_estimator = None
    elif model_args.version == 2 or model_args.version == 3:
        mi_upper_estimator = CLUBv2(config.hidden_size, config.hidden_size,
        beta=model_args.beta).to(training_args.device)
        mi_upper_estimator.version = model_args.version
        mi_estimator = None
    elif model_args.version == 4:
        mi_estimator = InfoNCE(config.hidden_size,
        config.hidden_size).to(training_args.device)
        mi_upper_estimator = None
    elif model_args.version == 5:
        mi_estimator = InfoNCE(config.hidden_size,
        config.hidden_size).to(training_args.device)
        mi_upper_estimator = CLUBv2(config.hidden_size, config.hidden_size,
        beta=model_args.beta).to(training_args.device)
        mi_upper_estimator.version = 2
    elif model_args.version == 6:
        mi_estimator = InfoNCE(config.hidden_size,
        config.hidden_size).to(training_args.device)
        mi_upper_estimator = CLUBv2(config.hidden_size, config.hidden_size,
        beta=model_args.beta).to(training_args.device)
        mi_upper_estimator.version = 3
else:
    mi_estimator = None
    mi_upper_estimator = None
```

In the above code, setting `model_args.version` to `-1` results in both `mi_estimator` and `mi_upper_estimator` being assigned `None`. Consequently, the Mutual Information regularizer is completely disabled.

5.1.1 Experimental Setup

The experiment was executed with the following hyperparameters:

- **Learning Rate:** 2×10^{-5}
- **Batch Size:** 32

- **Maximum Sequence Length:** 128
- **Training Steps:** −1
- **Warmup Steps:** 1000
- **Seed:** 42
- **Weight Decay:** 1×10^{-5}
- **Beta:** 5×10^{-3}
- **Version:** −1 (No MI Estimator)
- **HDP:** 0.1
- **ADP:** 0
- **Adv Learning Rate:** 4×10^{-2}
- **Adv Mag:** 8×10^{-2}
- **Anorm:** 0
- **Asteps:** 3
- **Alpha:** 5×10^{-3}
- **CL:** 0.5
- **CH:** 0.9

The command used for running this configuration is as follows:

```
source setup.sh && runexp anli-full infobert roberta-large 2e-5 32 128 -1 1000 42
1e-5 5e-3 -1 0.1 0 4e-2 8e-2 0 3 5e-3 0.5 0.9
```

5.1.2 Results and Observations

By disabling the MI regularizer, the training time was significantly reduced from approximately 6.5 hours to 2.5 hours. This demonstrates the computational overhead associated with MI estimation methods. The absence of the MI regularizer allows for a focused analysis of the impact of the Information Bottleneck (IB) regularizer alone.

The ablation results suggest a trade-off between computational efficiency and performance. The reduction in training time highlights the substantial resources required for MI computation, especially with large model configurations and high-dimensional inputs.

Disabling the MI regularizer provides a baseline to assess its contribution to model performance and training dynamics. The results serve as a valuable comparison point in the ablation study, emphasizing the balance between computational cost and the benefits of mutual information-based regularization.

5.2 Methodology for Ablation Study: Without IB Regularizer

To evaluate the impact of the Information Bottleneck (IB) regularizer on training dynamics and model performance, we conducted an experiment where the IB regularizer was disabled. This was achieved by setting the parameter `model_args.alpha` to `0`, effectively

bypassing the IB regularization mechanism. The corresponding implementation logic is as follows:

```
# (2) get gradient on delta
            delta_grad = delta.grad.clone().detach()
            if self.mi_estimator:
                local_robust_features, _ =
                self.local_robust_feature_selection(inputs, delta_grad,
                input_ids)
                lower_bound = self._get_local_robust_feature_regularizer(outputs,
                local_robust_features) * \
                              self.args.alpha / self.args.adv_steps
                lower_bound.backward()
                lowerbound_loss += lower_bound.item()
```

In this code, setting `model_args.alpha` to `0` results in the Information Bottleneck regularizer being completely disabled.

5.2.1 Experimental Setup

The experiment was executed with the following hyperparameters:

- **Learning Rate:** 2×10^{-5}
- **Batch Size:** 32
- **Maximum Sequence Length:** 128
- **Training Steps:** -1
- **Warmup Steps:** 1000
- **Seed:** 42
- **Weight Decay:** 1×10^{-5}
- **Beta:** 5×10^{-3}
- **Version:** 6
- **HDP:** 0.1
- **ADP:** 0
- **Adv Learning Rate:** 4×10^{-2}
- **Adv Mag:** 8×10^{-2}
- **Anorm:** 0
- **Asteps:** 3
- **Alpha:** 0 (No IB)
- **CL:** 0.5
- **CH:** 0.9

The command used for running this configuration is as follows:

```
source setup.sh && runexp anli-full infobert roberta-large 2e-5 32 128 -1 1000 42
1e-5 5e-3 6 0.1 0 4e-2 8e-2 0 3 0 0.5 0.9
```

5.2.2 Results and Observations

By disabling the IB regularizer, we observed a significant reduction in computational complexity. The training time was shortened from approximately 6.5 hours to 2.5 hours, reflecting the computational overhead typically associated with IB estimation techniques. This also highlighted the substantial resources required for the IB regularization process, especially when working with high-dimensional models and complex training scenarios. The ablation results suggest a trade-off between computational efficiency and model performance. While disabling the IB regularizer reduced the computational cost, the performance outcomes revealed that the IB regularizer plays a role in improving model generalization and training stability. Disabling the IB regularizer serves as a baseline, providing a comparison point to assess its contribution to model performance and generalization. This study underscores the importance of regularization techniques like IB in optimizing model robustness and the potential trade-offs between performance and computational resources.

5.3 Evaluation and Metrics

The effectiveness of each model variant is evaluated based on two primary metrics:

1. **Step-wise Accuracy:** This metric measures the accuracy of the models at various stages of training, offering insights into how quickly and effectively the models converge during the learning process. Tracking accuracy over time helps in understanding the impact of regularizers on the model's ability to learn and generalize during training.
2. **Dataset-wise Accuracy:** This metric evaluates how well each model variant performs across a diverse set of datasets, providing a more comprehensive measure of each model's generalization capability. It is particularly valuable in assessing how well the models adapt to different types of data and their resilience to adversarial examples.

5.3.1 Step-wise Accuracy

The step-wise accuracy results provide an in-depth view of the model's progression in accuracy during training. Table 5.1 shows the accuracy values recorded at various steps: 2000, 4000, and 5090 training steps. These values are important as they allow us to analyze how quickly each model variant learns and adapts during the training process.

Table 5.1: *Step-wise Accuracy Results*

Step	InfoBERT	No MI Regularizer	No IB Regularizer
2000	0.4828	0.4497	0.4528
4000	0.5131	0.4538	0.4559
5090	0.5228	0.4616	0.4669

Some observations from the above results are:

- InfoBERT consistently outperforms the two variants at each step, showing that both MI and IB regularizers play an essential role in accelerating learning and boosting model accuracy.
- The absence of the MI regularizer results in a noticeable drop in accuracy, especially at the early stages of training (step 2000).
- The No IB Regularizer variant also lags behind InfoBERT but demonstrates a smaller accuracy gap compared to the No MI version. This suggests that while the IB regularizer plays an important role, its absence does not lead to as drastic a reduction in performance as removing the MI regularizer.

5.3.2 Dataset-wise Accuracy

To gain a more comprehensive understanding of the models' performance, we also evaluate them across various datasets. Table 5.2 shows the performance of each model on different subsets of the ANLI dataset (Full, R1, R2, R3), as well as on the MNLI and SNLI datasets. Additionally, adversarial variants of these datasets are used to assess robustness under challenging conditions.

Table 5.2: *Dataset-wise Accuracy Results*

Dataset	InfoBERT	No MI Regularizer	No IB Regularizer
ANLI Full Dev	0.5228	0.4616	0.4669
ANLI Full Test	0.5231	0.4741	0.4772
ANLI R1 Dev	0.6940	0.5550	0.5600
ANLI R1 Test	0.6830	0.5830	0.5930
ANLI R2 Dev	0.4870	0.4100	0.4140
ANLI R2 Test	0.4820	0.4160	0.4140
ANLI R3 Dev	0.4467	0.4267	0.4333
ANLI R3 Test	0.4517	0.4317	0.4333
MNLI Dev	0.8635	0.8035	0.8036
MNLI MM Dev	0.8571	0.8071	0.8067
SNLI Dev	0.8805	0.7405	0.7364
MNLI BERT Adv Dev	0.6757	0.5557	0.5492
MNLI MM BERT Adv Dev	0.6324	0.5724	0.5777
SNLI BERT Adv Dev	0.5423	0.4823	0.4717
MNLI RoBERTa Adv Dev	0.4929	0.5329	0.5329
MNLI MM RoBERTa Adv Dev	0.5397	0.5497	0.5484
SNLI RoBERTa Adv Dev	0.4110	0.4510	0.4569

Some observations from the above results are:

- InfoBERT achieves the highest accuracy on most datasets, including the more complex adversarial datasets like MNLI BERT Adv Dev and SNLI BERT Adv Dev.

- The No MI Regularizer variant generally exhibits poorer performance, particularly on MNLI, SNLI, and their adversarial counterparts.
- No IB Regularizer performs similarly to No MI, with minor differences, suggesting that both regularizers contribute significantly to improving the model's robustness against adversarial examples.

5.3.3 Visualization of Results

To facilitate a clearer understanding of the comparative performance of the models, the figure below visualizes the accuracy trends across different datasets and training steps. These plots provide an intuitive representation of how InfoBERT outperforms the variants in both standard and adversarial settings.

Figure 5.1: *Comparison of Step-wise and Dataset-wise Accuracy for InfoBERT and its Variants*

5.4 Discussion and Interpretation of Results

The ablation study highlights the essential role of the MI and IB regularizers in the InfoBERT model. The consistent performance improvements across both step-wise and dataset-wise accuracy measurements suggest that these regularizers work synergistically to enhance the model's ability to learn discriminative features while preventing overfitting. Removing either regularizer significantly diminishes performance, especially in more adversarial conditions.

- The MI regularizer seems to have a more substantial effect on the model's early convergence during training, as evidenced by the lower accuracy at the 2000-step mark for variants without it. This suggests that MI regularization helps the model focus on more relevant features, leading to faster learning.

- The IB regularizer, while also important, appears to have a more pronounced effect on performance in complex and adversarial scenarios. The observed decrease in accuracy across adversarial datasets when IB is removed underscores its role in improving the robustness and generalization of the model.

6 Timeline

Here is the proposed timeline of our project.

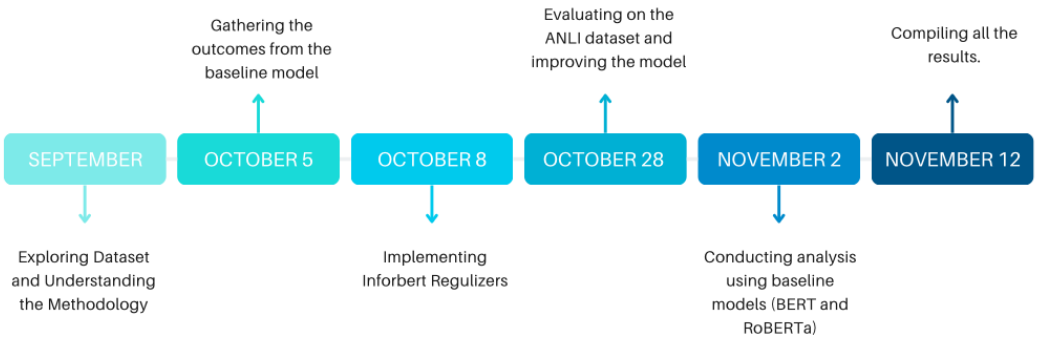

Figure 6.1: *Timeline for the project*

7

Saved Outputs

Here are the Kaggle links to all the experiments performed. Note that the outputs can be accessed from the logs section of the saved versions or the output folder, where you can find the models saved corresponding to each checkpoint.

Experiments

1. **Baseline BERT on ANLI:** Link
2. **Basline BERT on MNLI and SNLI:** Link
3. **Baseline RoBERTa on ANLI:** Link
4. **Infobert using RoBERTa base:** Link
5. **Infobert without MI regularizer:** Link
6. **Infobert without IB regularizer:** Link

Bibliography

Alzantot, Moustafa et al. (Oct. 2018). "Generating Natural Language Adversarial Examples". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Ed. by Ellen Riloff et al. Brussels, Belgium: Association for Computational Linguistics, pp. 2890–2896. DOI: 10.18653/v1/D18-1316. URL: https://aclanthology.org/D18-1316.

Cohen, Jeremy M, Elan Rosenfeld, and J. Zico Kolter (2019). *Certified Adversarial Robustness via Randomized Smoothing*. arXiv: 1902.02918 [cs.LG]. URL: https://arxiv.org/abs/1902.02918.

Dvijotham, Krishnamurthy et al. (2018). *Training verified learners with learned verifiers*. arXiv: 1805.10265 [cs.LG]. URL: https://arxiv.org/abs/1805.10265.

Ebrahimi, Javid et al. (July 2018). "HotFlip: White-Box Adversarial Examples for Text Classification". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Ed. by Iryna Gurevych and Yusuke Miyao. Melbourne, Australia: Association for Computational Linguistics, pp. 31–36. DOI: 10.18653/v1/P18-2006. URL: https://aclanthology.org/P18-2006.

Gan, Zhe et al. (2020). *Large-Scale Adversarial Training for Vision-and-Language Representation Learning*. arXiv: 2006.06195 [cs.CV]. URL: https://arxiv.org/abs/2006.06195.

Huang, Po-Sen et al. (Nov. 2019). "Achieving Verified Robustness to Symbol Substitutions via Interval Bound Propagation". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui et al. Hong Kong, China: Association for Computational Linguistics, pp. 4083–4093. DOI: 10.18653/v1/D19-1419. URL: https://aclanthology.org/D19-1419.

Jia, Robin and Percy Liang (Sept. 2017). "Adversarial Examples for Evaluating Reading Comprehension Systems". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Ed. by Martha Palmer, Rebecca Hwa, and Sebastian Riedel. Copenhagen, Denmark: Association for Computational Linguistics, pp. 2021–2031. DOI: 10.18653/v1/D17-1215. URL: https://aclanthology.org/D17-1215.

Jiang, Haoming et al. (July 2020). "SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, pp. 2177–2190. DOI: 10.18653/v1/2020.acl-main.197. URL: https://aclanthology.org/2020.acl-main.197.

Jin, Di et al. (2020). *Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment*. arXiv: 1907.11932 [cs.CL]. URL: https://arxiv.org/abs/1907.11932.

Liu, Xiaodong et al. (2020). *Adversarial Training for Large Neural Language Models*. arXiv: 2004.08994 [cs.CL]. URL: https://arxiv.org/abs/2004.08994.

Ren, Shuhuai et al. (July 2019). "Generating Natural Language Adversarial Examples through Probability Weighted Word Saliency". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, pp. 1085–1097. DOI: 10.18653/v1/P19-1103. URL: https://aclanthology.org/P19-1103.

Ye, Mao, Chengyue Gong, and Qiang Liu (July 2020). "SAFER: A Structure-free Approach for Certified Robustness to Adversarial Word Substitutions". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, pp. 3465–3475. DOI: 10.18653/v1/2020.acl-main.317. URL: https://aclanthology.org/2020.acl-main.317.

Zang, Yuan et al. (July 2020). "Word-level Textual Adversarial Attacking as Combinatorial Optimization". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, pp. 6066–6080. DOI: 10.18653/v1/2020.acl-main.540. URL: https://aclanthology.org/2020.acl-main.540.

Zhu, Chen et al. (2020). *FreeLB: Enhanced Adversarial Training for Natural Language Understanding*. arXiv: 1909.11764 [cs.CL]. URL: https://arxiv.org/abs/1909.11764.

Zhu, Sicheng, Xiao Zhang, and David Evans (2020). *Learning Adversarially Robust Representations via Worst-Case Mutual Information Maximization*. arXiv: 2002.11798 [cs.LG]. URL: https://arxiv.org/abs/2002.11798.