MNLI (Multi-Genre Natural Language Inference) Analysis in NLP Using the MultiNLI 1.0 Dataset, RoBERTa, and BERT
Term Paper, CSC 820, Spring 2022

Chris Huber*
SFSU

ABSTRACT

This paper describes the process of classifying inference from the MNLI 1.0 (Multi-Genre Natural Language Inference) dataset whose predecessor was SNLI (Stanford Natural Language Inference), corpora of over 422,000 and 550,000 sets of paired sentences respectively using a couple of different transformers. Inference examines the relationship between a premise and hypothesis using word embeddings to detect context. There are several models capable of this including BERT (Bidirectional Encoder Representations from Transformers) and RoBERTa (Robustly Optimized BERT) which can be retrained on new datasets and fine-tuned.

1 INTRODUCTION

Natural language inference revolves around determining whether a hypothesis is an entailment, contradiction, or neutral. An example is shown in Table 1. The MultiNLI corpus has 0.9 and 1.0 versions, and SNLI 1.0 is the current version for its corpus all of which will be examined .

Transformers are a relatively new deep learning technology that were first introduced in 2017 by Google. Transformers differ from RNNs (recurrent neural networks), which were the previously preferred method for a couple reasons. Transformers do not need to process data in order because they use an attention mechanism to ascertain context for any part of a sentence. Because of this, they are also able to leverage parallel processing which can greatly reduce the amount of time required to train them. This is not possible to do with an RNN because the ouputs from the hidden layer are used again as inputs forcing the process to run sequentially.

To perform this task, I established a baseline accuracy using a tranformer called RoBERTa which was developed by Facebook and Washington University in conjunction with Fairseq [4] [1] which is an open-source toolkit designed for translation, data modeling, and other text related tasks. RoBERTa is based on BERT which was developed by the Google AI team and is discussed in a paper called "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". [2] [2]

BERT primarily works by using a Masked Language Model (MLM) which hides some of the tokens in the input and attempts to derive the missing words using the surrounding context. This also works for next-sentence prediction which is the use case for inference. RoBERTa made some fundamental changes to BERT by removing the NSP (next-sentence prediction) objective, training with bigger batches and longer sequences, and dynamically changing the masking pattern. NSP was primarily used in BERT to predict whether passages belonged to the same or different documents, and removing this feature yielded better downstream task performance. [3]

Premise	Hypothesis	Label
A man inspects the uniform of a figure in some East Asian country	The man is sleeping.	contradiction
An older and younger man smiling.	Two men are smiling and laughing at the cats playing on the floor	neutral
A soccer game with multiple males playing	Some men are playing a sport	entailment

Table 1:

Example of evaluations of sentence pairs. [**?**]

There are three baseline implementations for RoBERTa which are based on CBOW (Continuous Bag of Words), bi-directional LSTM (Long Short-Term Memory), and ESIM (Enhanced Sequential Inference Model).[4] These are implementable by running the train_mlni.py script which uses TensorFlow to train the model on either MLNI data, a mix of MLNI and SLNI data, or on a single genre in the MLNI dataset. Dropout is used in all three implementations for regularization.

The inspiration for this project came from a list of tasks posted to the SuperGLUE website.[5] (The GLUE (General Language Understanding Evaluation) benchmark is a benchmark set of NLP tasks to be performed on sentence pairs which has an ongoing competition ranked by accuracy). RTE (Recognizing Textual Entailment) is a very current topic in NLP technology research and I was inspired to see what I could contribute. There have also been several Kaggle competitions involving inference determination.[6]. Incidentally, RoBERTa scored an 88.5 on the GLUE NLP benchmark competition with notable performance on MNLI.

I worked primarily on figuring out how to retrain the pre-trained models using the full MNLI 1.0 dataset for training and using validation and test datasets that were made available via a Kaggle competition on inference. I retrained both a RoBERTa and a BERT model using the dataset that was provided by the competition for validation. I then submitted my results to see how I placed on the leaderboard and to gauge the effectiveness of each transformer and its components.

*email:chrish@sfsu.edu

[1] https://aclanthology.org/N19-4009.pdf
[2] https://arxiv.org/pdf/1810.04805.pdf

2 Related Work

A paper entitled "A large annotated corpus for learning natural language inference" [1] [7] describes the development of the SNLI. It was published in August 2015 and was an effort to provide a good dataset for benchmarking since previous inference corpii were either algorithmically generated or too small and as such impeded effective analysis. It consists of pairs of image captions which were labelled using crowdsourcing via Mechanical Turk.

A paper entitled "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference" DBLP:journals/corr/WilliamsNB17 [5] [8] discusses the creation of the MultiNLI (Multi-Genre Natural Language Inference) corpus. It is comprised of 433,000 annotated examples designed to train machine learning algorithms. It offers data from 10 distinct genres of written and transcribed oral communication. It was also labelled using crowdsourcing via Mechanical Turk.

Prior to MLNI, the Stanford NLI corpus was the largest available corpus but fell short in a couple of ways. First, it was all from a single genre due to the fact that it was drawn only from image captions which lacked the robust language varieties that exist. The authors found it insufficient to provide a good benchmark for NLU (Natural Language Understanding).

The SNLI corpus contains 550,152 sets of labelled sentence pairs in its dataset. I also decided to try testing against it since it also produces measureable results. I took a 10,000 row subset of the data which takes approximately 2 hours to process since the entire set would take 1100 hours to process using my Mac laptop if it ever even completed.

RoBERTa is a technology developed by Google described in the paper "RoBERTa: A Robustly Optimized BERT Pretraining Approach". [3] [9] It describes the limitations that their team found with BERT related to undertraining. It had the best results at time of publishing on 4/9 of the GLUE tasks including MNLI and RTE.

An article entitled "Transformers: Retraining roberta-base using the RoBERTa pre-training procedure" [10] details the process of retraining RoBERTa on a custom dataset. For reference, it mentions that the Roberta-Base was "trained on 1024 V100 GPUs for 500K steps." It suggests using the TensorFlow transformers library to retrain a RoBERTa neural network with new data, which is the process I will follow.

3 Implementation

3.1 Testing MLNI Using RoBERTa

I started by evaluating the data in the MLNI 0.9 dataset for the Kaggle competition, since that is what they used. The test set consists of 9796 sentence pairs and has a fairly even distribution of genres as shown in Table 2. It is comprised primarily of sentences under 200 characters with a few outliers with a very long length as shown in Figure 1. I used roberta.large.mnli which was specifically fine-tuned for inference detection. I used the MNLI 1.0 train set of 400,000 sentence pairs and a validation set from the Kaggle competition of approximately 20,000 sentence pairs to retrain RoBERTa on data that was more customized to the competition.

I used the PyTorch framework to evaluate the data using tensors which are trained to predict

$$y = \sin(x) from - \pi to \pi.$$

[3] https://www.geeksforgeeks.org/overview-of-roberta-model/

[4] https://github.com/NYU-MLL/multiNLI

[5] https://super.gluebenchmark.com/tasks/

[6] https://www.kaggle.com/c/multinli-matched-open-evaluation/data

[7] https://arxiv.org/pdf/1508.05326.pdf

[8] https://arxiv.org/pdf/1704.05426.pdf

[9] https://arxiv.org/pdf/1907.11692.pdf

[10] https://towardsdatascience.com/transformers-retraining-roberta-base-using-the-roberta-mlm-procedure-7422160d5764

by minimizing squared Euclidean distance. Tensors are similar to NumPy arrays but can be run on either the CPU or GPU. I used the PyTorch no_grad() option to disable gradient calculation data which is otherwise tracked for later calculations to optimize the code as shown in Listing 1.

```
with torch.no_grad():
    for k in range(len(test_s1)):
        # Encode sentences and make a prediction
        tokens = roberta.encode(test_s1[k], test_s2[k])
        prediction = roberta.predict('mnli', tokens).argmax().item()
```

Listing 1: Code to perform predictions using RoBERTa and PyTorch.

I used the HuggingFace transformers API[11] to download the RobertaForMaskedLM model, the RobertaTokenizer, LineByLineTextDataset, DataCollatorForLanguageModeling, and the Trainer object which allowed me to use the pre-trained model which used roberta-base on the MNLI 1.0 train set and use the Kaggle MNLI 0.9 validation set. As such, I may have unwittingly performed transfer learning, given that the model was already configured with a prior dataset. The configuration for this model is shown in Listing 2:

```
Model config RobertaConfig {
  "architectures": [
    "RobertaForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "classifier_dropout": null,
  "eos_token_id": 2,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-05,
  "max_position_embeddings": 514,
  "model_type": "roberta",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 1,
  "position_embedding_type": "absolute",
  "transformers_version": "4.19.0.dev0",
  "type_vocab_size": 1,
  "use_cache": true,
  "vocab_size": 50265
}
```

Listing 2: Roberta config settings.

Since the RoBERTa model I used is optimized for inference, it came preconfigured with a predict() function which was was very helpful in producing output. However, it still took my CUDA-enabled computer running an NVIDIA Geoforce RTX 3080 GPU over three days to complete one epoch of training on the supplied data, which was likely due to the use of robera-mnli-large. I have included some sample output from the training batches in Table 2.

After the model completed, I uploaded it to my account on huggingface.co [12] in the event it might be helpful for someone else. Huggingface is a company that builds and hosts open-source datasets

[11] https://huggingface.co/docs/transformers/model$_doc/roberta$

[12] "https://huggingface.co/chrishuber"

Step	Training Loss	Validation Loss
500	0.525600	0.283678
1000	0.288000	0.230991
1500	0.241500	0.217871
...	...	...
96500	0.047100	0.111401
97000	0.047900	0.110389
97500	0.049900	0.112635
98000	0.049700	0.109552

Table 2:

Training and validation loss per batch.

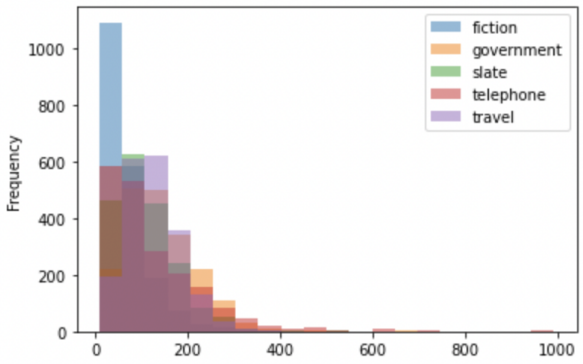

Figure 1: Distribution of sentence lengths by genre in the MLNI 0.9 matched dataset. [14]

Genre	Count
Fiction	1978
Travel	1964
Telephone	1955
Government	1953
Slate	1946

Table 3:

Genre counts for Kaggle dataset.

as well as a library of models and transformers used for all types of NLP-related tasks. I also uploaded the datasets that I used which can be directly downloaded by running the code in Listing 2.

```
from datasets import load_dataset
dataset = load_dataset('chrishuber/roberta-retrained-mlni')
```

Listing 3: Import datasets from my repository on Huggingface.

3.2 Testing SNLI

I also ran the MNLI-pretrained RoBERTa model against a train-test split of a 10,000 row subset of the SNLI train data. It came back with an F1 score of 0.8637 which is quite good for being trained on a different dataset. The most misclassifications were entailments predicted as neutral (490) and contadictions predicted as entailments (396). This reflects the robustness of the MNLI-trained RoBERTa model and suggests it is the state-of-the-art evaluator for inference.

3.3 Retraining BERT on Kaggle Dataset

After I was able to successfully train RoBERTa, I started looking into adjusting the model to change the features or otherwise tune it.

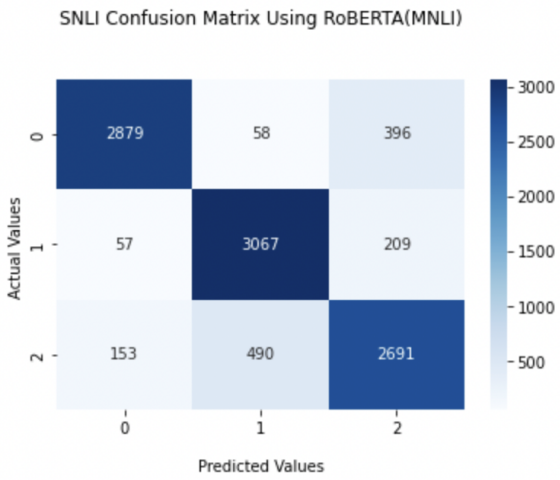

Figure 2: Results of running RoBERTa pre-trained for MLNI on the SNLI dataset.

While researching how to do this, I ran across another TPU-based Kaggle competition called "Contradictory, My Dear Watson" [15], but which contained inference-labeled data. After looking at an example notebook that used BERT to train data for the Contradictory, Dear Watson competition I set out to try also using BERT to retrain a model and extract inference results.

I was originally confused about which configuration, model, and tokenizer to use to produce the type of results I was looking for. I initially tried the RobertaTokenizer with BERT, but was unsuccessful with that. I then spent some time testing BERTForMaskedLM in the mistaken belief that it was used for inference in the same way RoBERTa is. However, I found that the BERT version is primarily used for masked text prediction using language models and started getting output with very long tensors which did not correspond to the class predictions I needed. I also needed to connect the dots with how we give the model the classes we want to use. After digging some more, I found that there is a num_labels config parameter for BertForSequenceClassification transformer and realized that I should be using that one. I believe that training your own tokenizer is also quite common particularly for specialized cases like the one I am examining, but I had to place that in my plans for future work due to time limitations.

I used the same dataset I had uploaded containing the MNLI 1.0 train data and Kaggle MNLI 0.9 validation dataset and did data cleaning and reformatting to get it into a format that the BERT model understood. I then used the Trainer.train() method to retrain BERT on the new data which also took approximately three days to run. (There was also a power outage where I live during the first training attempt which forced me to restart the whole thing.)

I used encode_plus to from the tokenizer to turn the words into numeric representations. This also allowed me to set a maxlength for each sentence which I unfortunately had to set at 256 to prevent a CUDA out-of-memory error. I feel like the score would very likely increase if I could figure out how to run it without truncating sentences, since many of the passages in the data are quite long and since longer passages was how RoBERTa improved upon it. The encoder also padded the data so that all tensors were the same length which is necessary since it is not an RNN.

My train dataset had 397,702 sentence pairs split into 98,716 batches, validation had 20,000, and the test set had 9796. I used DataLoader from PyTorch utils to separate the data into batches

[15] https://www.kaggle.com/c/contradictory-my-dear-watson

which is necessary when using a large dataset to avoid exhasting the CUDA memory which otherwise tries to store all output at once. I also performed 3-fold validation of the data to help ensure a reliable result.

I used a the Trainer object from HuggingFace to both train the model and perform predictions. Because I used batches, I needed to set the gradients for the model and the optimizer to 0 each time by calling model.zero_grad() because otherwise PyTorch accumulates them. I also normalized the gradients using torch.nn.utils.clip_grad_norm_ which alleviated a potential exploding gradients issue. I then called optimizer.step() to reevaluate the model and calculate loss. The model.state_dict() from each fold of the cross-validation was saved in a separate file and the highest scoring one was used to evaluate against the test data.

The output comes as tensors with three values per row which is shown in Listing 4. I then ran np.argmax() on them with the axis set to 1 to determine the index of the value with the highest probability across the x-axis. This corresponded to 0, 1 or 2 which were the classifications I needed.

```
[array([[-2.3344262 ,  0.6812976 ,  1.9206414 ],
        [-1.8758273 , -1.5486668 ,  3.872561  ],
        [-1.9655329 , -1.645029  ,  3.691682  ],
        ...,
        [ 4.492677  , -0.90280604, -3.264653  ],
        [-1.0716397 ,  4.316246  , -3.094708  ],
        [-0.99402064, -0.7566957 ,  2.2035198 ]],
      dtype=float32),
```

Listing 4: Example of raw inference output from BERT model.

Finally, I converted the predictions to their label equivalent, appended the pairIDs, and submitted to Kaggle. This time, the result wasn't quite as good as with RoBERTa, but the process of figuring out how all the pieces fit together was worth more than any Kaggle competition score.

In the process of doing all of these trainings, I discovered a lot about how transformers and PyTorch work, as well as encountering unexpected limitations related to hardware. I spent a lot of time initially getting PyTorch to recognize my CUDA device which, as it turns out, required CUDA Toolkit 11.0 and a specific version of PyTorch to run. After I got the correct versions of everything installed, I began to see errors stating that CUDA had run out of memory. I discovered that I needed to reduce the batch size to 4 in order to prevent it from exhausting memory between batches.

While investigating this problem, I discovered the purpose of the DataLoader and DataCollator objects. The DataLoader splits the inputs into batches per iteration that are smaller in size and easier to process. DataCollator helps to get the data in the same format by, for example, adding padding, to allow batches to be created. These are all components that are helpers for transformers which are modular by nature.

4 Evaluation

4.1 Computational platform and Software Environment

The benchmarking tests using CPU were mainly conducted on a Mac with an M1 Max processor running OS 12.2.1 and 64GB of RAM. The model training and GPU tests were conducted on an Alienware Dell PC running Windows 11 with 64GB of RAM, Intel Core i7-11700F, and an NVIDIA GeForce RTX 3080 GPU.

For the software base, I used Python 3.6, PyTorch 1.11.0, Fairseq 0.10.12, roberta-large-mlni, mutlilni_0.9, mutlilni_1.0, and snli 1.0.

4.2 RoBERTa Retrained Model Results

I configured, trained, and ran the RoBERTa algorithm against the MLNI corpus and the corresponding matched or mismatched validation set to better understand the process of using transformers and applying it to unlabeled competition data. It performed quite well, resulting in a score of .90006 for the matched set and 0.89915 on the mismatched set which ranked as ties for 7th and 5th in the respective Kaggle competitions. To clarify, the matched dataset contains sentences derived from the same sources as the train data whereas the mismatched set contains sentence pairs from other sources that the trained model would have never seen.

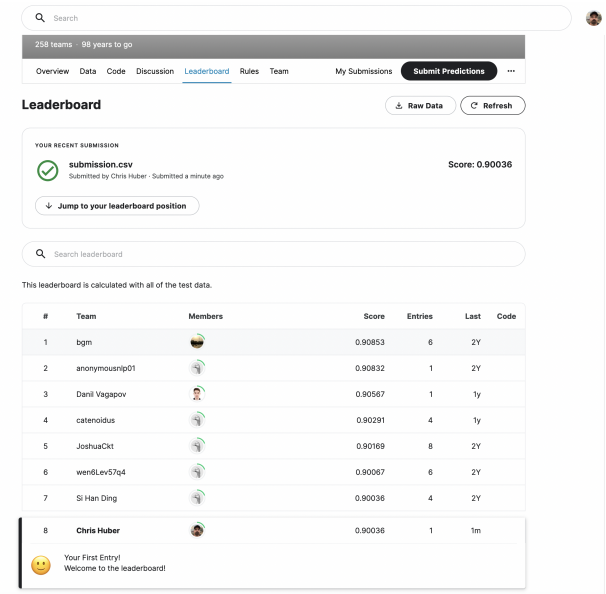

Figure 3: Results from first run of RoBERTa against the matched MultiNLI 0.9 dataset.

I intended on digging more into how to customize the inputs for RoBERTa but it proved to be a daunting task which was going to require a lot more experience than I had at the time. It is very possible to alter the model configuration for RoBERTa which I was able to do with BERT to produce classification results. Doing this, however, also requires understanding of how to set up the transformer layers to produce a fine-tuned result which is something I am still working on getting a better handle on. This experience did teach me, however, that all models do not come with a predict() function which in most cases you need to write yourself and how intricate a process it is to construct deep learning networks.

4.3 BERT Retrained Model Results

BERT was considerably more challenging to train, in part because this is the first deep learning network I have successfully implemented for a large-scale project and also the first time I have used PyTorch. I did a lot of testing and small-batch training runs over the weeks following my second report to debug, troubleshoot, and understand the output I was getting and what it meant.

After figuring out the correct model and tokenizer to use, calibrating the DataLoader and batch processing, and reconfiguring the dataset, I started training the BERT version using batches that were 1/100 of the total to make sure it would complete correctly and deliver useable output (and to avoid waiting 3 days for the results). Since BERT does not have a predict() function predefined, I needed to take the raw tensor output and fashion the classifications out of them using an argmax function. I was finally able to get a set of predictions in the 0, 1, 2 format corresponding to contradiction, neutral, and entailment. I took those results, translated them into the submission format Kaggle was expecting, and submitted them to see what my F1 score was against the unlabelled test data.

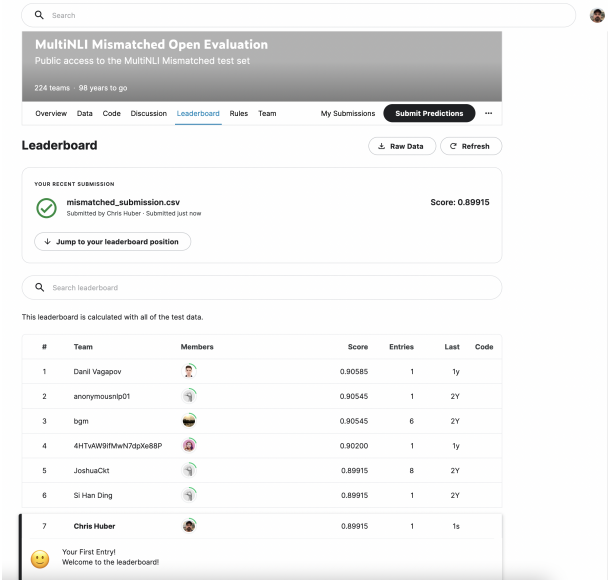

Figure 4: Results from first run of RoBERTa against the mismatched MultiNLI 0.9 dataset.

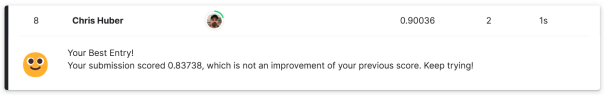

Figure 5: Results from first run of retrained BERT against the matched MultiNLI 0.9 dataset.

I scored an 83.7% with the BERT model, which for a three-class classification is far above what would be achieved by random guessing and indicated that the model worked well, albeit not as well as the RoBERTa model pre-trained for MNLI. I believe this has to do with the changes that were made to BERT by RoBERTa which include removing BERT's next-sentence prediction and using a much higher learning rate.

5 Conclusions and Future Work

5.1 Methodology for Testing Inference Successfully Implemented

I set out at the beginning of the semester to broach a new topic in NLP that I had never heard of, inference. In doing so, I unwittingly set the stage for myself to dive headlong into the use of transformers and deep learning to solve problems by implementing large scale frameworks and retraining them using custom datasets. I discovered that transformers are preferred for sequential processing over RNNs because of their attention mechanism which is better at divining context. The evidence for this was clear when I scored quite high on the leaderboard using this relatively new technology.

It was also very exciting to implement parallelism using CUDA on my own machine. I was only exposed to CUDA one semester ago in the High-Performance Computing class taught by Prof. Wes Bethel, but the benefits and advantages of accelerated processing impressed me enough to buy a new machine capable of running it. It also gave me many insights into the power of deep learning, whether via Keras, PyTorch, TensorFlow, or any other platform to achieve things that were thought impossible only a few years ago.

5.2 Future Work

I feel that continuing this research and looking further into fine-tuning the model would be the way to improve accuracy even more. At the same time, I have only tried this on a Kaggle dataset which was the fastest way to get an empirical evaluation of how the model was performing. Using a wider variety of datasets would be ideal, although there are currently very few that are large enough and annotated in a reliable way to do extensive testing. I was very excited to learn about the a field which is on the frontier of NLP and have greatly expanded my horizons by studying it.

References

[1] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. *CoRR*, abs/1508.05326, 2015.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[3] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[4] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. fairseq: A fast, extensible toolkit for sequence modeling. *CoRR*, abs/1904.01038, 2019.

[5] A. Williams, N. Nangia, and S. R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *CoRR*, abs/1704.05426, 2017.