Exploring Factors Affecting Accuracy in Quantitative Natural Language Inference: Insights from Data Augmentation, Numeral Tokenization Strategies, Model Architectures, and Hyperparameters

Ruilin Yang
University of Potsdam
ruilin.yang@uni-potsdam.de

Abstract

In this paper, we examine the impact of a model's configurations and hyperparameters on its performance on Quantitative Natural Language Inference (QNLI) problems. We define model configuration through 3 dimensions: it is trained on the original training set or the augmented training set, each numeral is treated as a single token or further split into digit-based tokens, and the variations in model architecture. Furthermore, we investigate the impact of different hyperparameters on performance given the same model architecture. In conclusion, within the range of our experiment, on the side of configuration, training set and tokenization of numerals don't have a clear impact on the performance, while model architecture makes a visible difference; on the side of hyperparameters, learning rate shows a ubiquitous correlation with performance across all model architectures, while other hyperparameters such as batch size, hidden size or embedding dimension, have no clear and consistent correlation with the performance.

1 Introduction

Natural Language Inference (NLI), also known as Recognizing Textual Entailment (RTE), is the task of determining the inference relation between two (short, ordered) texts (MacCartney and Manning, 2008). Given a "premise", if the truthfulness of a "hypothesis" is true, false, or undetermined, the relationship between them is respectively defined as entailment, contradiction, or neutral. Quantitative Natural Language Inference (QNLI) is an NLI task with both the premise and the hypothesis involving numerals that play a crucial role in determining the answer. Due to the ubiquitous role of numerals in our lives, reasoning with numerals has drawn research interest. QNLI can enable immediate applications that require good Natural Language Understanding capabilities of numerals, such as semantic search and question answering (Harabagiu and Hickl, 2006) in data-intensive domains like finance or healthcare (Duan et al., 2021). In this project, we use a subset of the EQUATE dataset to explore how different configurations and hyperparameters can impact the performance of QNLI task. Specifically, we define configuration along 3 dimensions: whether or not we augment the training dataset by making use of the properties of inference logic, whether or not we split numerals as one token per digit, and out of 4 model architectures (Feedforward Neutral Network, (bidirectional) Recurrent Neutral Network, and Transformer) which one to choose. In addition, we zoom in on the models with the best and worst performance across different architectures, to examine the relationship between each hyperparameter and model performance.

2 Related Works

Several studies have been carried out on the same dataset: Ravichander et al. (2019) has tested variations of the BiLSTM-RNN model together with transformer-based models such as OpenAI GPT and BERT, none of the tested models consistently outperforms the majority baseline on all subsets of EQUATE benchmark. Mishra et al. (2022) achieved an f1 score of 85% with a neural-symbolic model `Ex-NumNet` augmented with Information Retrieval. Most recently, Chen et al. (2023) improved the performance to above 99% by exploring the notation of numbers, and by leveraging pre-trained transformer-based models with a pre-finetuning number comparison task.

Along the direction of previous studies, we choose RNN and Transformer as our focused model architecture and a Feedforward Neural Network as a simpler architecture for comparison.

notation	example
original (s1, s2)	"Marie has 248 $ in her account of the bank", "Marie has 148 $ in her account of the bank",
character (s1_char, s2_char)	"Marie has 2 4 8 $ in her account of the bank", "Marie has 1 4 8 $ in her account of the bank",

Figure 1: 2 different notations of numbers: original, characterized original.

3 Task Formalization

We train and tune a collection of models under each different configuration, where each model is initialized with a set of randomly generated hyperparameters. After that we analyze if certain configurations lead to better results, and if there exists a pattern between various hyperparameters and the performance.

3.1 Configurations

We define configurations as the settings outside a model. Specifically, they are training set augmentation, notation of numerals, and model architecture. In total, we have 16 configurations, considering all the combinations of variations along these 3 dimensions.

3.1.1 Data Augmentation

Data augmentation is a widely used technique in many machine learning tasks to virtually enlarge the training dataset size and avoid overfitting. In our QNLI task, we can utilize the property of inferential logic to augment the training set: when the relationship between a premise and a hypothesis is contradiction or neutral, we can swap the order of premise and hypothesis, and the contradiction/neutral relationships stay the same. It does not apply to training examples where the relationship is entailment, because implication does not necessarily work in a reversed way. For data augmentation, we have two options: normal training set or augmented training set.

3.1.2 Numeral Tokenization

A study of Chen et al. (2023) shows transforming the numerals in the statements can improve the performance. Characterized notation, where each numeral is split so that a single digit becomes a token, is correlated with higher accuracy compared with taking the numeral itself as a token, see Figure 1. For numeral tokenization, we have two options: original notation or characterized notation.

3.1.3 Model Architecture

Along the lines of the previous study, we choose Recurrent Neutral Network (RNN) and Transformer as our main focus of interest in model architecture. We add Feedforward Neural Network (FFNN) for comparison. For RNN we have the option to make it bidirectional, to exclude bidirectional or not as a hyperparameter, we choose to define Bidirectional RNN (BiRNN) as a separate model type. In the end, we have four options for model architecture: RNN, BiRNN, Transformer, and FFNN.

3.2 Hyperparameter

Under the same configuration, the model's performance can differ due to different hyperparameters. Because of the large number of hyperparameters and the limitation of computational resources, it is infeasible to perform a grid search in the hyperparameter space. We adopt a practical approach to selectively choose hyperparameters to tune and fix some hyperparameters to be static. The tuning process can be characterized as iteratively zooming in, in the hope of hitting the region that yields good results in the hyperparameter space: we first train a set of models initialized with randomly generated hyperparameters, see if there are specific coordinates in the multi-dimensional hyperparameter space that correlates with good performance, and zoom in to generate hyperparameter sets within the promising range to train more models, if computational resource allows. When the tuning is finished, we analyze the characteristics of various hyperparameters that correlate with good performance.

A summary of all the fixed and tune-able hyperparameters can be seen in Table 1.

4 Data

The train/dev/test set consists of 6475, 970, and 1691 examples respectively. Each training example is a tuple of (s1, s2, label), s1 and s2 are the two (ordered) statements, the label indicates their relationship is entailment, neutral, or contradiction.

The dataset comes from Chen et al., 2023, which is in turn adapted from the `EQUATE` benchmark (Ravichander et al., 2019). The training set covers 4 subsets of EQUATE paper, the dev and test set consist of StressTest only, see Table 2.

The `RTE_Quant` set is constructed from the RTE subcorpus for quantity entailment, with statements requiring temporal reasoning removed. The `NewsNLI` set is created from the CNN corpus of

Hyperparameter	Is tunable? (Y/N)	Note
n_epochs	N	Fixed to be 1000
dropout	N	Fixed to be 0.2, see 5.1
Optimizer	N	Fixed to be Stochastic Gradient Descent
batch_size	Y	-
learning_rate	Y	-
embedding_dim	Y	-
hidden_size	Y	Only for FFNN / (Bi)RNN
num_layers	Y	Only for FFNN / (Bi)RNN
num_blocks	Y	Only for Transformer
num_heads	Y	Only for Transformer

Table 1: All the hyperparameters that are considered in this study. Including fixed hyperparameters and tunable hyperparameters.

dataset	StressTest	NewsNLI	AWPNLI	RTE_Quant
train	4619	968	722	166
dev	970	-	-	-
test	564	-	-	-

Table 2: Source subset of EQUATE dataset of our train/dev/test sets.

news articles with abstractive summaries. The AWPNLI set is synthesized by establishing a world, optionally updating the world, and posing a question about the world. The StressTest set is synthesized from algebraic problems (Ravichander et al., 2019). Examples of each subset can be found in Table 3.

5 Experiments

5.1 Models in Detail

All the models we implement incorporate an embedding layer in which the shape is affected by both configuration (how we tokenize text, hence the vocabulary size) and the embedding dimension hyperparameter that is randomly generated. Other aspects of model shape such as hidden size and number of layers for FFNN and (Bi)RNN, and number of heads and number of blocks for Transformer, are all subject to the randomly generated hyperparameters.

5.1.1 Majority Baseline

We evaluate our models against a majority class baseline. Since the distribution of all 3 answers in both the dev set and test set are even (see Table 4), a naive model that randomly chooses answers would have a baseline accuracy of 33% on both the dev set and test set.

5.1.2 FFNN

Our FFNN has one embedding layer, one input layer, and potentially several hidden layers, and one output layer. We fix the activation function to be ReLU which is applied after each layer. A dropout layer with a fixed dropout rate of 0.2 will only be applied to each hidden layer, and to the input layer only when there is at least 1 hidden layer. The embeddings of all tokens in the sequence are pooled by taking the element-wise average. The pooled embedding is then fed to FFNN as input.

Hyperparameter embedding_dim determines the embedding dimension, and num_layers affects the number of hidden layers it has.

5.1.3 (Bi)RNN

Our RNN model has one embedding layer, one built-in RNN module from PyTorch, and one output layer to project the output of the RNN module to the number of desired classes, in our case, 3. The BiRNN model consists of two RNNs that go through the training examples in the opposite direction, and combine the output of the RNNs into a single output. We implement it adapting our RNN model, setting the built-in RNN module to be bidirectional. The reason we mention BiRNN as a standalone model type, rather than an option for RNN, is because we want to exclude bidirectional flag from hyperparameters. Hyperparameter embedding_dim determines the embedding dimen-

EQUATE subset	s1	s2
StressTest	"Fred and Sam are standing 40 miles apart and they start walking in a straight line toward each other at the same time"	"Fred and Sam are standing 20 miles apart and they start walking in a straight line toward each other at the same time"
NewsNLI	"A $ 5,000 reward is offered ."	"Know something ? Call 641-228-182 . A reward is offered"
AWPNLI	"Joan has 8.0 orange balloons and her friend gives her 2.0 more "	"Joan has 5.0 orange balloons now"
RTE_Quant	"Employers created 144,000 new payroll jobs in August as the unemployment rate dipped to 5.4 percent , a modest improvement over the 5.5 percent jobless rate in July , the Labor Department reported Friday ."	"The Labor Department said this sector has added nearly 1 of every 5 of the new jobs created during the last 12 months ."

Table 3: Examples of training data from different EQUATE subset.

dataset	#entailment	neutral	contradiction
train	2461	2112	1902
dev	324	323	323
test	564	564	563

Table 4: Answer distribution in train/dev/test dataset.

sion, and num_layers turns (Bi)RNN into stacked (Bi)RNN with a specified number of layers.

5.1.4 Transformer

We implemented a decoder-only Transformer from scratch. The embedding of tokens is computed from the addition of token embedding and positional embedding, with positional embedding being able to embed sequences of a maximum length of 512. 512 is more than 3 times the maximum number of tokens in a sequence in the training examples. The embedded tokens are then fed to a series of blocks sequentially, wherein each block it's a multi-head self-attention module, followed by a linear layer that projects the output of the self-attention back to the size of the embedding dimension, interleaving with two normalization layers, thus making sure the output comply to the size and can be conveniently fed to the next block. Within the multi-head self-attention module, there are varied numbers of self-attention heads, where the embeddings of each token are updated as a weighted average of the embeddings of all the other tokens in the sequence.

Hyperparameter num_blocks and num_heads decide the number of blocks and self-attention heads in each block respectively.

5.1.5 Summary

The configurations as defined in 3 yield 16 unique combinations. For each configuration, we trained several models initialized with randomly generated hyperparameters. With the restrain of computational resources, the count of models of each configuration we trained can be seen in Table 5.

5.2 Evaluation and Discussion on Dev Set

In this section, we inspect the performance of all trained models on the dev set; in section 6 "Results and Error Analysis", we choose the models with the best performance and evaluate them on the test set. This way we ensure that we are not choosing the best models in favor of the test set and that the performance on the test set could be a realistic estimate of the model's performance in the real world. Since we don't have the problem of unbalanced classes, we use accuracy as the metric on both the dev set and the test set.

As an overall observation of performance across all dimensions of configuration, see Figure 2, (Bi)RNN and Transformer can both achieve good results, with the best results of (Bi)RNN slightly higher than Transformer, and the variance of (Bi)RNN higher than Transformer. It is expected that FFNN does not perform well in general, since

dataset	#models
normal training set	original notation	FFNN	10
normal training set	original notation	RNN	5
normal training set	original notation	BiRNN	10
normal training set	original notation	Transformer	5
normal training set	original_char notation	FFNN	10
normal training set	original_char notation	RNN	10
normal training set	original_char notation	BiRNN	7
normal training set	original_char notation	Transformer	13
train_augmented set	original notation	FFNN	8
train_augmented set	original notation	RNN	8
train_augmented set	original notation	BiRNN	5
train_augmented set	original notation	Transformer	5
train_augmented set	original_char notation	FFNN	10
train_augmented set	original_char notation	RNN	5
train_augmented set	original_char notation	BiRNN	6
train_augmented set	original_char notation	Transformer	5

Table 5: A summary of the number of models we trained under each configuration.

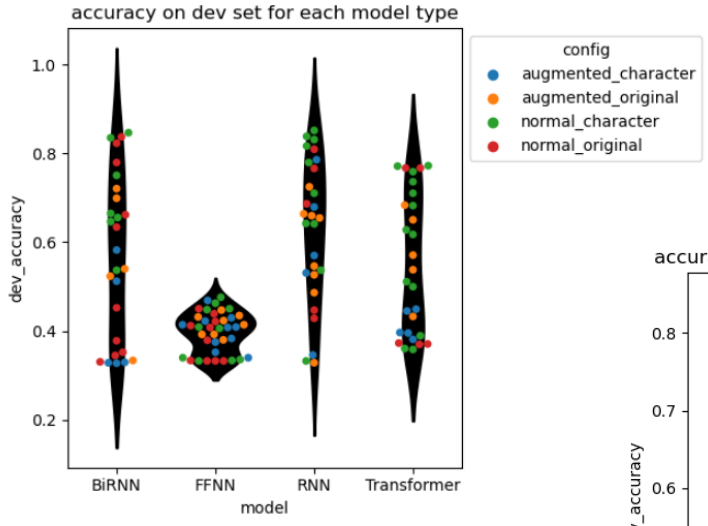

Figure 2: Overall performance on dev set of all models across all configurations.

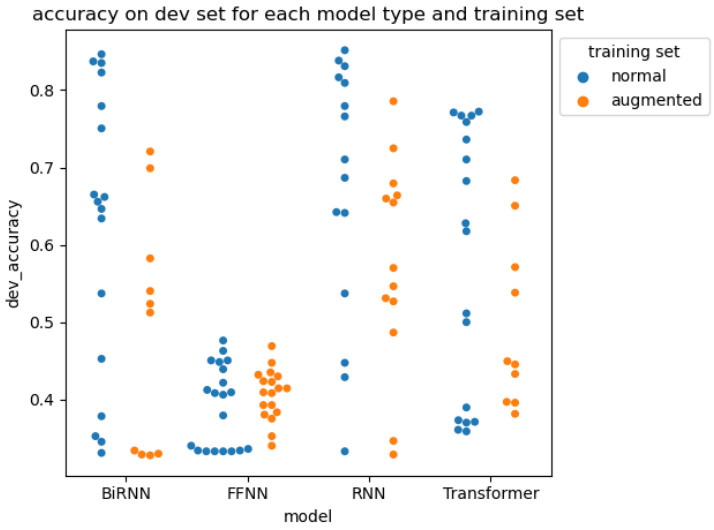

Figure 3: A closer inspection at the impact of data augmentation on dev set accuracy across different model architecture.

much information is lost when pooling the embeddings in the sequence.

The performance range of (Bi)RNN and Transformer have slightly lower variance on the augmented training set, with the best performance slightly lower than models trained on the normal training set, see Figure 3 and Figure 4. There is no clear and consistent pattern of the impact on different notations.

While most models outperform the majority baseline accuracy of 33%, we see a lack of difference between original notation and character notation. It might come from the bias of the pro-

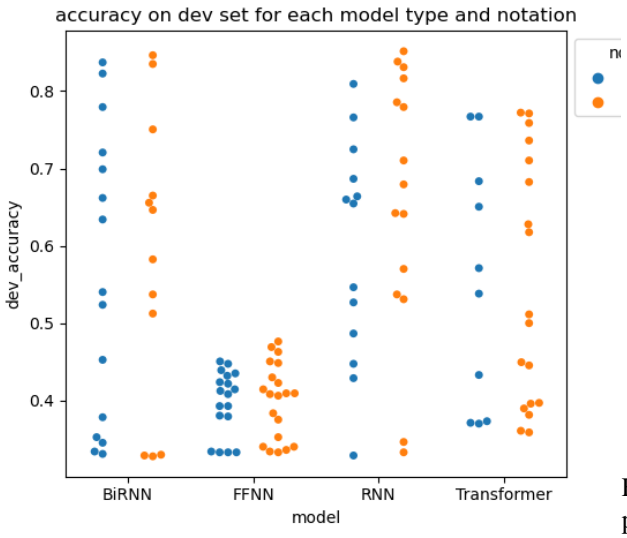

Figure 4: A closer inspection at the impact of numeral tokenization on dev set accuracy across different model architecture.

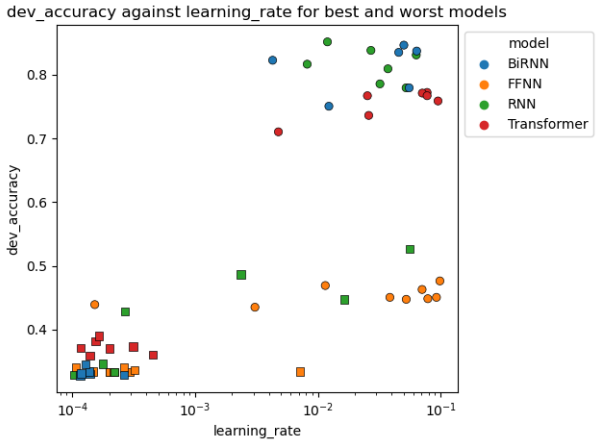

Figure 5: Learning rate of the best-performed and worst-performed models, across all model architecture. Filled circles denote best-performed models, filled squares denote worst-performed models.

vided dataset. We assume the count of unknown tokens (#UNK) under character notation all comes from the unknown non-numeral tokens, subtracting it from #UNK of original notation, we get an estimate of the #UNK that comes from numeral tokens in the original notation. We see that #UNK of numeral token takes a small proportion in both the dev set and the test set in Table 6.

In reality, it is not viable to expect the training set that consists of finite examples to cover the infinite possibility of numeral tokens. While in our experiment the notation of numbers does not make a visible difference, it might partly come from the bias in the dataset.

Since we didn't observe a clear pattern along the training set and notation dimension of configuration, in the following analysis of hyperparameters we discard these two dimensions, rather, we only focus on the model architecture and the hyperparameter of interest. For better contrast, we focus on the models with the top 25% and bottom 25% performance of each model architecture. We use filled circles to denote the best models, filled squares to denote the worst ones and color to encode different model architectures.

In Figure 5 we see the relationship between dev set accuracy and learning rate. The learning rate is randomly generated from the range 10e-4 to 10e-1 on a log scale. There is a clear correlation between learning rate and dev set accuracy regardless of model architecture, the models with the best performance tend to concentrate in the range (0.01, 0.1), while most of the worst-performed models have learning rates near 10e-4.

For all the other tunable hyperparameters in Table 1, we didn't observe a visually significant linkage between the value of the hyperparameter and the accuracy on the dev set. We include the plots in the Appendix A.2.

6 Results and Error Analsis

We choose the best model from each model architecture based on the accuracy of dev set, evaluate it on the test set, and provide a concise error analysis.

6.1 Evaluation on Test Set

The best model is chosen based on dev set accuracy of each model architecture. Accuracy on training set and dev set are for reference, see Table 7

6.2 Error Analysis

- The Most frequent error across all best models is in predicting entailment labels, within the same model, the recall of the entailment class is consistently lower than the other two classes, see Table 8. This indicates challenges in understanding logical relationships between statements.

- Statements containing complex reasoning or multiple conditions are more likely to be classified incorrectly, see Figure 6.

- More errors observed in complex lexical and syntactic structures (sentences with multiple

dataset	#UNK original (1)	#UNK characterized (2)	(1)-(2)	#examples
dev set	721	607	114	970
test set	1069	900	169	1691

Table 6: Number of unknown tokens in dev set and test set, when training set is in original notation and characterized notation respectively.

best model	train_accuracy	dev_accuracy	test_accuracy
FFNN	51.57%	47.63%	46.84%
RNN	88.83%	85.15%	83.80%
BiRNN	90.32%	84.64%	82.02%
Transformer	90.67%	77.22%	77.47%

Table 7: Accuracy on train/dev/test sets of the best-performed model of all model architecture.

Statement 1:
Jill works as a waitress at the local diner where she earns an hourly wage of $ 6.00 per hour and a standard tip rate of 35 % of the cost of the orders she serves
Statement 2:
Jill works as a waitress at the local diner where she earns an hourly wage of $ 6.00 per hour and a standard tip rate of 65 % of the cost of the orders she serves

Statement 1:
John paid a 15 % tip over the original price of the dish , while Jane paid the tip over the discounted price for the coupon
Statement 2:
John paid a 25 % tip over the original price of the dish , while Jane paid the tip over the discounted price for the coupon

Figure 6: Two example of the test set where all 4 of the best-performed model predicted wrong.

clauses, nested phrases, or uncommon vocabulary), see Figure 6.

7 Conclusion

In this paper, we examine the impact of different configurations and hyperparameters on the performance of QNLI task. In configurations, we find augmenting the training set and transforming numerals as single digit-based tokens do not have a visible impact, however, the lack of impact of alternative way of numeral tokenization might partly be explained by the bias in the provided dataset, that many numerals in dev and test set have appeared in training set. Among all hyperparameters, we find the learning rate has a strong correlation with performance. All the other hyperparameters that we consider do not show a clear pattern of correlation with the performance.

8 Limitations

In the original notation where each numeral is treated as a token, due to the infinite nature of numerals, the vocabulary computed from the training set would be insufficient in recognizing the numerals in the dev set and test set. For QNLI problems, the numerals play a crucial part. If the models only learned to deduce the answer from the non-numeral tokens, we may lack confidence in model robustness. In the character notation, where each numeral is split into multiple single-digit tokens, the problem of recognizing dev set and test set numerals as unknown tokens is lifted, since all digits can be recognized. However, each single-digit token is no different from a non-numeral token, adding or removing one single-digit token is adding or removing one ordinary token in the vocabulary to the model, but it will cause the change of the underlining number by a factor of 10 and might have critical impact on the answer of QNLI problem. Even with all single-digit tokens recognized, the numeracy structure is still lost.

Transforming the numeral tokens into scientific notation and splitting the scientific notation into single-digit-based tokens as characterized scientific notation might preserve more numeracy structure. They are indeed provided dataset from Chen et al., 2023, however, we discovered the data quality of these two more promising notations to be corrupted, see Appendix A.3, and due to time constrain we dropped them.

Another direction of improvement is to compute the embeddings of numerals and non-numerals separately: Non-numerals are embedded in the same way as one token in the vocabulary, while the embedding of numerals is computed in a way that preserves their numeracy, such as DICE Sundararaman et al. (2020) and NEKG Duan et al. (2021).

9 Ethical Considerations

Numeracy-literate models may have access to vast amounts of sensitive numerical data, raising con-

best model	entailment_recall	neutral_recall	contradict_recall
FFNN	29.32%	58.33%	55.56%
RNN	63.89%	98.46%	93.21%
BiRNN	68.83%	96.30%	88.89%
Transformer	58.95%	84.57%	88.27%

Table 8: Accuracy on train/dev/test sets of the best-performed model of all model architecture.

cerns about privacy and surveillance. If not properly regulated, these models could exploit personal or financial information without consent, leading to breaches of privacy and potential misuse of data. In addition, models trained in numeracy may exacerbate existing social and economic inequalities by favoring those with access to high-quality numerical data or resources for model development. This could widen the gap between those who benefit from numerical literacy and those who are marginalized or disadvantaged.

References

Chung-Chi Chen, Hiroya Takamura, Ichiro Kobayashi, and Yusuke Miyao. 2023. Improving numeracy by input reframing and quantitative pre-finetuning task. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 69–77, Dubrovnik, Croatia. Association for Computational Linguistics.

Hanyu Duan, Yi Yang, and Kar Yan Tam. 2021. Learning numeracy: A simple yet effective number embedding approach using knowledge graph. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2597–2602, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Sanda Harabagiu and Andrew Hickl. 2006. Using scenario knowledge in automatic question answering. In *Proceedings of the Workshop on Task-Focused Summarization and Question Answering*, pages 32–39, Sydney, Australia. Association for Computational Linguistics.

Bill MacCartney and Christopher D. Manning. 2008. Modeling semantic containment and exclusion in natural language inference. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 521–528, Manchester, UK. Coling 2008 Organizing Committee.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. 2022. NumGLUE: A suite of fundamental yet challenging mathematical reasoning tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3505–3523, Dublin, Ireland. Association for Computational Linguistics.

Abhilasha Ravichander, Aakanksha Naik, Carolyn Rose, and Eduard Hovy. 2019. EQUATE: A benchmark evaluation framework for quantitative reasoning in natural language inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 349–361, Hong Kong, China. Association for Computational Linguistics.

Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. 2020. Methods for numeracy-preserving word embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4742–4753, Online. Association for Computational Linguistics.

A Appendices

A.1 Personal Contribution in Teamwork

My learning goal for this project is to consolidate my theoretical and practical knowledge of RNN, Transformer, and methodologies of hyperparameter tuning, plus brush up on necessary tools such as `Git`, `Latex`, and `Docker`. Apart from `Docker`, I have achieved all my learning goals.

I have implemented RNN on top of PyTorch built-in RNN module, and Transformer from scratch. As I am more experienced in programming within the team, I also provided the scaffolding of the codebase for my teammates such as hyperparameter tuning, data loading and transformation, command line argument parsing, etc. I shared my experiences with teamwork methodologies such as SCRUM and Git workflow.

I started from a position of having a vague understanding of the basics such as how embeddings work, with almost no experience with PyTorch, and completely no experience of reading NLP papers. In the end, I have a much better understanding of NLP workflow, a clearer idea of how researchers work, a mental picture to work with high-dimensional tensors, and gained practical experience to code with PyTorch.

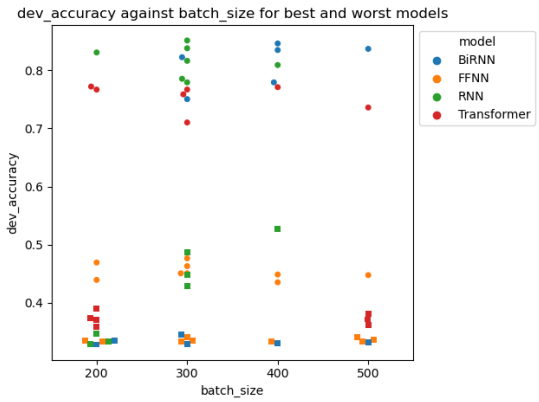

Figure 7: Dev set accuracy against batch size

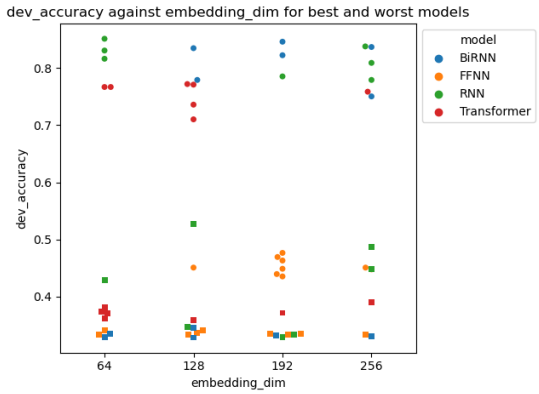

Figure 8: Dev set accuracy against embedding dimension

A.2 Dev Set Accuracy against Various Hyperparameters

In the following figures, filled circles denote best-performed models, filled squares denote worst-performed models. Color encodes the different model architecture.

A.3 Data Quality Issue

In exploratery data analysis we find the number of tokens per example of characterized scientific notation blows up by a factor of 100 (see Figure 13, compared to uncharacterized scientific notation.

We locate the problem to be in scientific notation, more than necessary zeros are added to the number, and subsequently the characterized scientific notation is also corrupted. When we find this problem, we have run out of time to clean the data and test on these two notations again.

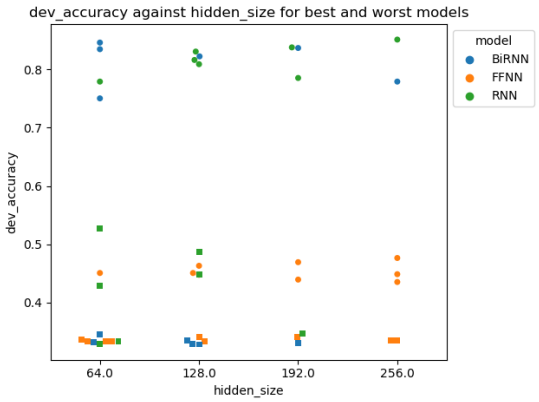

Figure 9: Dev set accuracy against hidden size, only applicable to FFNN, and (Bi)RNN.

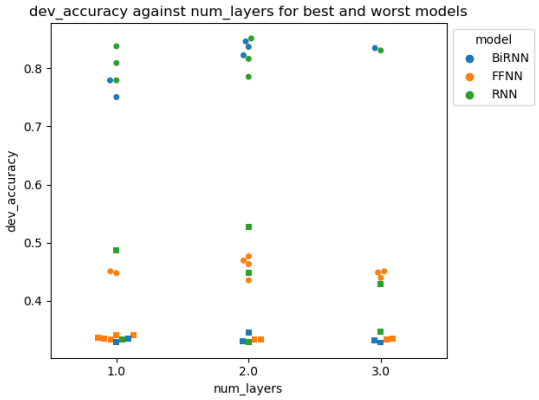

Figure 10: Dev set accuracy against number of layers, only applicable to FFNN and (Bi)RNN.

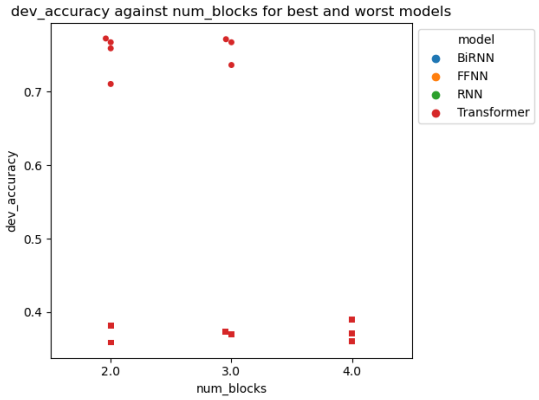

Figure 11: Dev set accuracy against number of blocks, only applicable to Transformer.

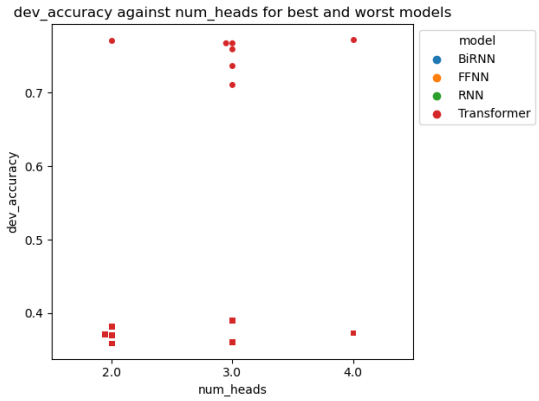

Figure 12: Dev set accuracy against number of heads in multi-head self-attention, only applicable to Transformer.

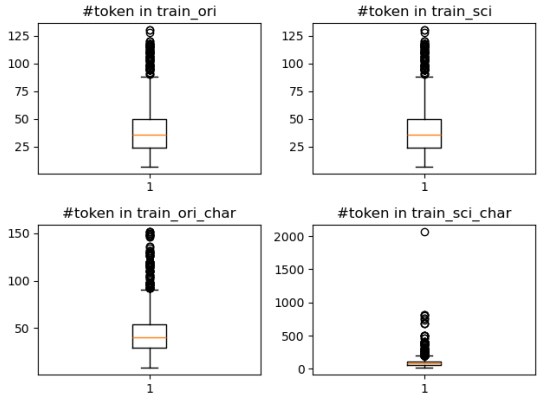

Figure 13: Boxplot of number of tokens per example across different notation of numerals.

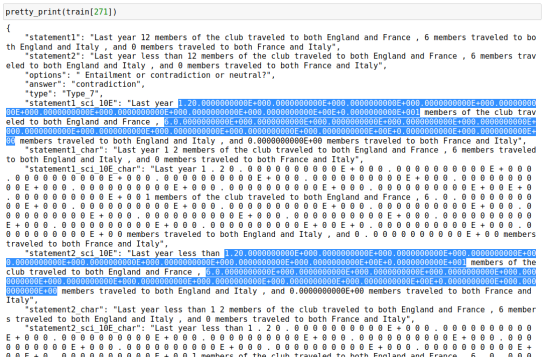

Figure 14: One example of currupted training example.