Inverse Text Normalization- Literature Review

ZHANG MENG AO

(WEEK 1)

Current Approaches

1. Traditional rule-based approach by using Weighted Finite-State Transducers (WFST)

   Pros: complete control over the generated output, easy to debug and make correction

   Cons: require much labor and linguistic knowledge, hard to scale up and generalize, usually don't consider the context information.

2. Neural network (NN)

   Pros: don't need to define every rule, data-driven; could generalize better than WFST given enough data;

   Cons: May make unrecoverable errors and requires lots of data. Black box, hard to debug and make correction

3. Hybrid models – Combine WFST and NN. WFST constrains the prediction of the NN when it has low confidence in he prediction or corrects common mistakes in the output of the NN model.

4. ChatGPT - Using prompts.

Current Research Works

- Found 18 research works on Inverse Text Normalization spanning from 2008 to 2023
- Most works are published between 2020 to 2022
- The most recent works
  - LANGUAGE AGNOSTIC DATA-DRIVEN INVERSE TEXT NORMALIZATION (24/01/2023)
  - Focus on improving performance of ITN on low-resource language
  - AdapITN: A FAST, RELIABLE, AND DYNAMIC ADAPTIVE INVERSE TEXT NORMALIZATION (04/06/2023)
  - Focus on handling both semiotic phrases and phonetization phrases (Vietnamese)

NeMo ITN

NeMo Inverse Text Normalization: From Development To Production

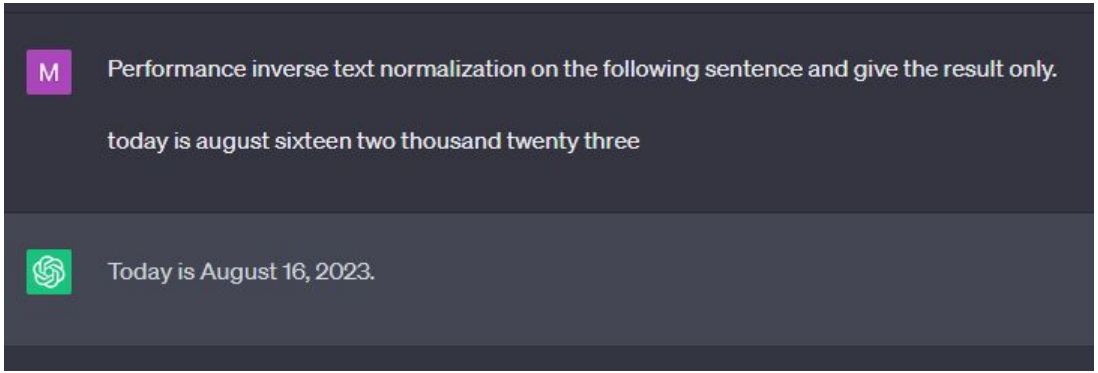
- Pure WFST
- Sentence-level accuracy: 73.73% / 75.65% (evaluated in the second work)

A UNIFIED TRANSFORMER-BASED FRAMEWORK FOR DUPLEX TEXT NORMALIZATION

- Combines TN and ITN. Two steps: Tagging (determine the span needs to be normalized); Normalizing
- Sentence-level accuracy: 93.17% / 97.31% (evaluated in the third work)

Thutmose Tagger: Single-pass neural model for Inverse Text Normalization

- Simpler design. Treat ITN as tagging problem.
- Sentence-level accuracy: 97.43%



ChatGPT

- Performance not known – need more testing
- Highly flexible, can be configured by simply changing the prompt

Challenges and Plan for the Next Week

- Hard to determine the state-of-the-art models because the evaluation data is not unified for different works and the language may also differ.
  - Need to select a publicly available ITN dataset or use our own evaluation data to check the performance.
  - Choose the language to focus on.
- Compare other methods with NeMo on the chosen dataset

Inverse Text Normalization – Literature Review

ZHANG MENG AO

(WEEK 2)

Inverse Text Normalization Dataset

- There is no large and public dataset curated for ITN available.
- Some works used Google Text Normalization Dataset [1] by swapping the input and target
  - Not annotated by human
  - Generated by using Google TTS system's Kestrel text normalization system
- Other works proposes their own ways to generate spoken-form text from written-form text
  - Regular expression [2]
  - WFST [3]
- One work [4] used internal dataset – Microsoft

[1] *RNN Approaches to Text Normalization: A Challenge* https://arxiv.org/ftp/arxiv/papers/1611/1611.00068.pdf
[2] *AdapITN: A FAST, RELIABLE, AND DYNAMIC ADAPTIVE INVERSE TEXT NORMALIZATION*
[3] *FOUR-IN-ONE: A JOINT APPROACH TO INVERSE TEXT NORMALIZATION, PUNCTUATION, CAPITALIZATION, AND DISFLUENCY FOR AUTOMATIC SPEECH RECOGNITION* https://arxiv.org/pdf/2210.15063.pdf
[4] *Streaming, fast and accurate on-device Inverse Text Normalization for Automatic Speech Recognition* https://arxiv.org/pdf/2211.03721.pdf

Problems on Current Dataset

- Not annotated by human, thus cannot make sure 100% correct
- May not cover all semiotic classes
- Don’t cover cases in different languages or even different cultural habits for the same language
- As a result, the model’s performance may degrade in the production environment.

Baseline

- Choose NeMo's 2nd or 3rd approaches to start
  - Publicly available, easy to reproduce
  - Tested on Google TN dataset, easy to compare with other methods
- Test the performance of Large Language Models on Google TN dataset

WeNet

- End-to-end ASR system; didn't consider the problem of text normalization or inverse text normalization
- Whether inverse text normalization is done may depend on the dataset
  - E.g. LibriSpeech: book text(written form) □□ reading audio

Plan for the Next Week

- Compare NeMo with other open-source models on Google TN dataset
- Test LLM's performance on Google TN dataset
- Investigate Whisper's architecture

Inverse Text Normalization – Reproducing NeMo

ZHANG MENGAO

WEEK 3

	semiotic_class	written	spoken
0	PLAIN	It	<self>
1	PLAIN	can	<self>
2	PLAIN	be	<self>
3	PLAIN	summarized	<self>
4	PLAIN	as	<self>
5	PLAIN	an	<self>
6	PUNCT	"	sil
7	PLAIN	error	<self>
8	PLAIN	driven	<self>
9	PLAIN	transformation	<self>
10	PLAIN	based	<self>
11	PLAIN	tagger	<self>
12	PUNCT	"	sil
13	PUNCT	.	sil
14	<eos>	<eos>	None
15	PLAIN	This	<self>
16	PLAIN	plan	<self>
17	PLAIN	was	<self>
18	PLAIN	first	<self>
19	PLAIN	enacted	<self>

Google Text Normalization Dataset

- Text source: Wikipedia
- Annotation: Kestrel system – Google's TTS system
- 1.1 billion English tokens
- 100 files in total
- Each record of the data consists of three fields: Semiotic class, spoken, written
- Test set in [1]- The first 100,002 tokens of the 99th file.

[1] RNN Approaches to Text Normalization: A Challenge

Google Text Normalization Dataset

```
PLAIN          67894
PUNCT          17746
<eos>           7551
DATE            2832
LETTERS         1409
CARDINAL        1037
VERBATIM        1001
MEASURE          142
ORDINAL          103
DECIMAL           92
ELECTRONIC        49
DIGIT             44
TELEPHONE         37
MONEY             37
FRACTION          16
TIME               8
ADDRESS            4
Name: semiotic_class, dtype: int64
```

- Distribution of semiotic classes
- Most of the tokens doesn't need to change (PLAIN)
- Not balanced
- Thutmose Tagger sampled a more challenging test set that contains 1000 tokens of each class

NeMo – Thutmose Tagger

- Training script available
- Each token in the input (spoken domain) is tagged with <self>, <delete>, or a token in the vocabulary for replacement
- How the vocabulary is got: count the frequency of the written fragments in GTN and predefine some common number tokens
- Model architecture: pre-trained BERT encoder + MPL + softmax layer
- Need to process the training dataset such that each input token is aligned with a tag (Giza++)
- Limitation: assume monotonical alignment
  - Most examples in GTN satisfy this requirement.
  - Drop most non-monotonical example except some special cases
  - Solution to the special cases: add angle brackets to the tag to indicate positional change and do post-processing

Thutmose Tagger

- Successfully done inference on the test set
- Haven’t done evaluating
- The code is integrated into the NeMo repository
- May need to extract the relevant code to increase the speed

Whisper

- Minimized data processing
- Depend on the training set scale and capability of the model to generate natural transcript
- As a result, no inverse text normalization is involved.

Plan for Next Week

- Evaluate Thusmose Tagger and the Duplex model
- Estimate the time taken to do inference for both models
- Explore the options of LLM

Inverse Text Normalization – Evaluating

ZHANG MENG AO

WEEK 4

Thutmose Tagger

- Default test set:
  - WER: 3.74%
  - Sentence accuracy: 97.48% 7277/7465
  - digit errors: 0.33% 25/7465
  - other errors: 2.18% 163/7465
- Hard test set: (at least 1000 examples for each semiotic classes)
  - WER: 8.95%
  - Sentence accuracy: 87.72% 8435/9616
  - digit errors: 3.16% 304/9616
  - other errors: 9.12% 877/9616
- ~20-30 ms/example (CPU)
- ~2 ms/example (GPU)
- GPU memory - 1618 MB

Duplex Model

- Pretrained model not compatible with code
  - Might because of package incompatible
- May need to train the model by ourselves
- Many bugs
  - Depend on nemo_text_processing package

Large Language Model APIs

- ChatGPT (GPT family) – OpenAI
  - Free credits for each phone number. New accounts with the same phone number will not get new credits.
- BARD (LaMDA & PaLM family) – Google
  - API not yet publicly available. Only a limited number of users can use.
- Claude – Anthropic
  - Need to request for access
- LLAMA – Meta
  - Need to request for access
- Cohere
  - Performance not good.

Open-Source Large Language Models

- GPT-Neo-1.3B
- GPT-Neo-2.7B
- GPT-J-6B
- GPT-NeoX-20B

Plan for the next week

- Start building the website using Thutmose Tagger as backend.
- Explore other method's performance

Inverse Text Normalization - Development

ZHANG MENGAO

WEEK 5

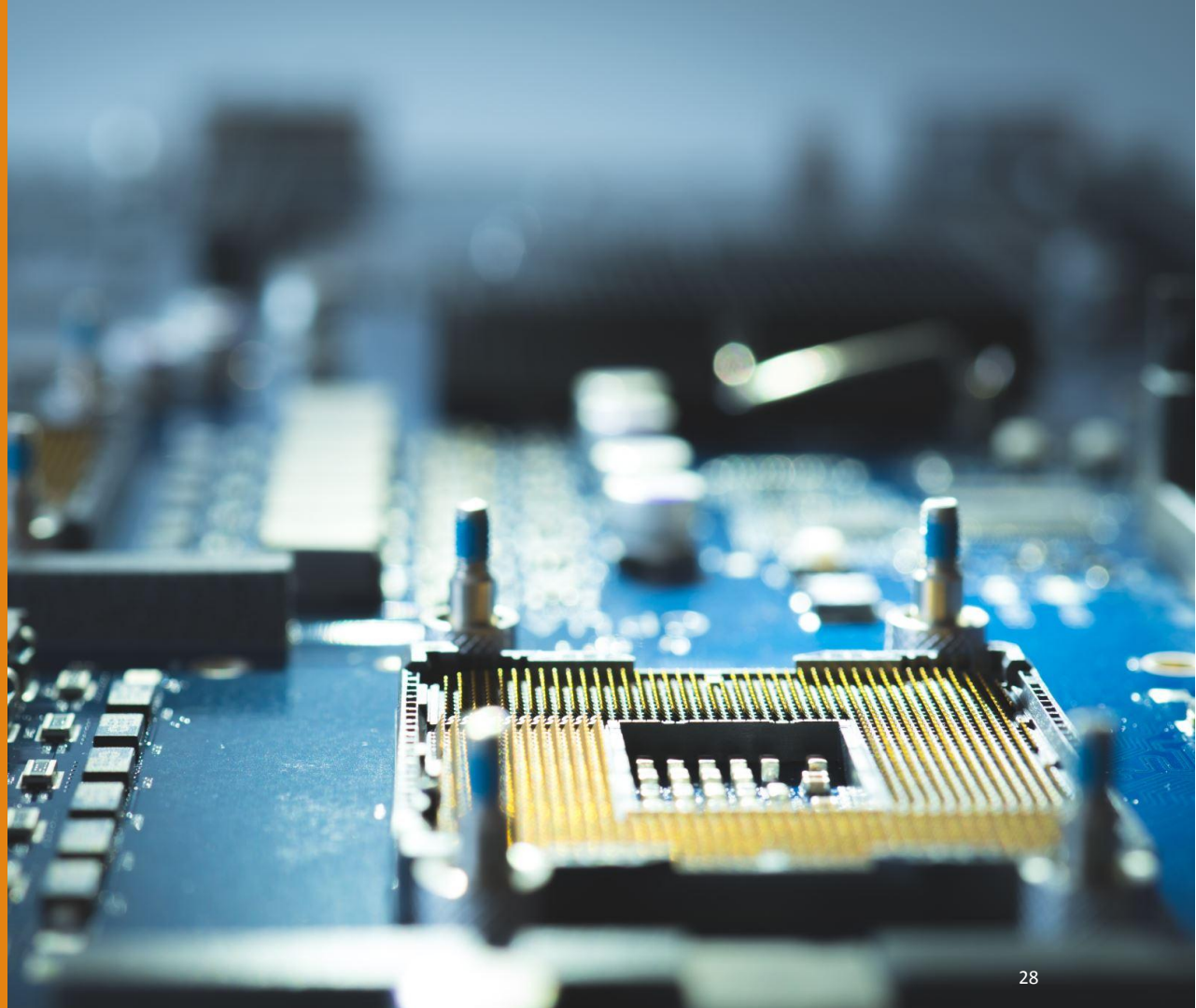
Thutmose Tagger Performance

```
input:          gilstead road bedok north and one north link
answer:         gilstead road bedok north and one north link
model output:   gilstead road bedok north and 1 north link

input:          jurong island terminal miller street and aljunied m r t station
answer:         jurong island terminal miller street and aljunied mrt station
model output:   jurong island terminal miller street & aljunied mrt station

input:          geylang lorong eight bukit ho swee link and jalan basong
answer:         geylang lorong 8 bukit ho swee link and jalan basong
model output:   geylang lorong 8, bukit ho swee link and jalan basong

input:          oh oh so would you like to settle the er hong kong dollar outstanding balance with your us dollar
answer:         oh oh so would you like to settle the er HK$ outstanding balance with your US$
model output:   oh oh so would you like to settle the er hong kong dollar outstanding balance with your us $

input:          remit hong kong dollars three thousand five hundred
answer:         remit HK$ 3500
model output:   remit hong hkd 3500

input:          five hundred and fifty pound okay against usd
answer:         £550 okay against US$
model output:   five hundred and fifty pound okay against usd

input:          um hum the second one is uh australian dollar three eight one nine five uh
answer:         um hum the second one is uh AUD 38195 uh
model output:   um hum the second one is uh australian $ 38195 uh

input:          eight hundred sixty thousand eight six six sixty thousand
answer:         860000 8 6 6 60000
model output:   868 u. s. 60 6 60,000

input:          this is for service fees seventeen of april to sixteen of july
answer:         this is for service fees 17th of April to 16th of July
model output:   this is for service fees 17 of april to 16 of july

input:          so we will remit us dollars two hundred and three thousand
answer:         so we will remit US$ 203000
model output:   so we will remit us dollars 200 and 3,000

input:          point ninety three usd so so we'll do that uh i mean the buy back the euros
answer:         0.93 usd so so we'll do that uh i mean the buy back the euros
model output:   . 93 usd so so we'll do that uh i mean the buy back the euros

input:          so we'll be doing this for value tomorrow so the indicative spot rate right now is one point uh three
answer:         so we'll be doing this for value tomorrow so the indicative spot rate right now is 1. uh 3
model output:   so we'll be doing this for value tomorrow so the indicative spot rate right now is 1 point uh 3

input:          okay good yeap and the second would be the canadian dollars er canadian dollars ten thousand
answer:         okay good yeap and the second would be the CAD er CAD 10000
model output:   okay good yeap and the second would be the canadian dollars er canadian dollars 10.000
```

```
input:          yeah you would like to do it one week or two week
answer:         yeah you would like to do it 1 week or 2 week
model output:   yeah you would like to do it one week or two week

input:          ter the iban for this transfer is fr seven six three zero zero seven seven zero four eight nine four three nine
answer:         ter the iban for this transfer is fr 77300770489439
model output:   ter the iban for this transfer is fr 763 0 0 7 7 0 489439

input:          you need to remit fifteen thousand us dollars to bank united new york branch the beneficiary
answer:         you need to remit US$ 15000 to bank united new york branch the beneficiary
model output:   you need to remit 15,000 us dollars to bank united new york branch the beneficiary

input:          ya eighty thousand three hundred and fifty
answer:         ya 800350
model output:   ya 800300 and 50

17
```

Live Demo of the Website

Average response time ~0.1 s (CPU on server)



Large Language Models

Fine-tuning a LLM (GPT family, LLaMA) is an option given enough resources

Not applicable if we deploy it on a CPU machine given response constraint

Plan for the next week

- Check other methods' performances
  - *Four-in-one*
- Test time taken on local machine
- Test the inference time of LLM on local machine

Inverse Text Normalization-Four–in–one

ZHANG MENGAO

WEEK 6

Literature Review

- *JOINT PREDICTION OF TRUECASING AND PUNCTUATION FOR CONVERSATIONAL SPEECH IN LOW-RESOURCE SCENARIOS*
  - Jointly consider capitalization and punctuation.

- *CONTROLLABLE TIME-DELAY TRANSFORMER FOR REAL-TIME PUNCTUATION PREDICTION AND DISFLUENCY DETECTION*
  - Jointly consider punctuation and disfluency detection

- No other work unifying the four tasks.

Method

Stage 1: Joint labeling of ITN, punctuation, capitalization, and disfluency

- First, tokenize the spoken-form text.
- Then use a transformer encoder to learn a shared representation of the input.
- Four task-specific classification heads – corresponding to ITN, punctuation, capitalization, and disfluency – predict four token-level tag sequences from the shared representation
- Each classification head consists of a dropout layer followed by a fully connected layer.
- loss: cross entropy

$$CE_{joint} = \frac{CE_i + CE_p + CE_c + CE_d}{4} \qquad (1)$$

Method

Stage 2: Tag application

- First, convert token-level to word-level text.
- Different strategies are applied to different tasks.
- ITN: the tags are the categories of the token. The consecutive tokens with the same tag are treated as a unit for the WFST conversion. If multiple spoken-form tokens are mapped to a single WFST output, only the first capitalization tag and last punctuation tag are applied.
  - WFST grammar: only mentioned in the acknowledge part
- Punctuation: append the indicated punctuation tags to the corresponding words
- Capitalization: capitalize the first letter or entirety of words, as tagged.
- Disfluencies, simply remove the disfluency tagged words from the text sequence

DATA PROCESSING PIPELINE

- Various datasets are used which include both written form text and spoken form text. The written-form datasets take a very large proportion.

Dataset	Distribution
OpenWebText	22.8%
Stack Exchange	13.6%
OpenSubtitles2016	3.3%
MAEC	2.9%
NPR Podcast	0.6%
SwDA + SWB + Fisher	0.3%
Web-crawled ITN	56.4%
Conversational Disfluency	0.1%

Table 1. Data distribution by number of words per dataset

Data processing of written form

- Process each of our sets to contain token-level tags for each of the four tasks
- Cleaned and filtered the datasets by preserving natural sentences or paragraphs and removing characters apart from alphanumeric, punctuation, and necessary mid-word symbols such as hyphens.
- ITN: used WFST to do text normalization and get the tags
- Punctuation: comes from the text
- Capitalization: comes from the text
- Disfluencies: as written form text doesn't have disfluencies, all text is tagged with a non-disfluency tag.

Data processing of spoken form

- Get the written form via a commercial formatting service
- Apply the same process with written form text to get ITN, punctuation, capitalization tags
- Disfluencies:
  - SwDA already contains dialog act annotations, so we translate these to token-level disfluency tags.
  - SWB and Fisher are not annotated for disfluencies. The self-training approach is used Improving Disfluency Detection by Self-Training a Self-Attentive Model
  - Self-training: We have a small amount of labeled data. We use these data to train a model, and then predict the unlabeled data. Add the data with high confidence into the training set. Retrain the model and repeat this cycle.

Tag types

- ITN
  - Five classes (alphanumeric, numeric, ordinal, money, time);
  - ‘O’ representing non-ITN
- Punctuation
  - 4 tag categories: comma, period, question mark, and ‘O’ for no punctuation
  - Each punctuation tag represents the punctuation that appears appended to the corresponding token of text
- Capitalization
  - 3 tag categories: all uppercase (‘U’), capitalize only the first letter (‘C’), and all lowercase (‘O’)
- Disfluency
  - 7 tag categories: correction of repetition (‘C_RT’), reparandum repetition (‘R_RT’), correction (‘C’), reparandum (‘R’), filler word (‘F’), all other disfluency (‘D’), and non-disfluency (‘O’)

Experiment

Metrics

- word level precision (P), recall (R), and F1 scores

Model architecture

- 12-layer transformers with 16 attention heads, 1024-dimension word embeddings, 4096-dimension fully connected layers, and 8-dimension projection layers between the transformer encoder and the decoder that maps to the tag classes

Settings

- TASK represents the model trained on specific tasks
- JOINT represents the models jointly trained with four heads
- TASK-SMALL have less parameters

Conclusion

Our joint transformer tagging model, trained from scratch on spoken- and written-form data, matches or exceeds the performance of task-specific models on all four tasks across test domains – despite training a fraction of the parameters.

Plan for the Next Week

- Dockerize the system and test its performance on local PC CPU
- Test the inference time of LLMs with different sizes on local PC CPU

Inverse Text Normalization – review of CulturaX and code

WEEK 7

ZHANG MENGAO

CulturaX Dataset

Title:

- CulturaX: A Cleaned, Enormous, and Multilingual Dataset for Large Language Models in 167 Languages

Research Gap:

- LLM requires cleaned deduplicate data to get high-performance
- Lack of transparency for current LLMs' training data
- More severe for multilingual scenario

Dataset: CulturaX

- 6.3 trillion tokens.
- 167 languages.
- Stages for preparation: language identification, URL-based filtering, metric-based cleaning, document refinement, and data deduplication

CulturaX Dataset Construction

How the data is obtained:

- Combining two datasets: mC4, OSCAR
- Both are from web-crawled data (Common Crawl)

Data Preparation Pipeline:

- Data cleaning: language identification, URL-based filtering, metric-based cleaning, and document refinement
  - Language identification
  - URL-based filtering: using a blacklist of websites
  - metric-based cleaning
  - document refinement
- Data Deduplication:
  - MinHash Deduplication
  - URL-based Deduplication

Cleaning Code and Dockerization

Cleaning

- It contains only NeMo code + README.file
- I included directory structure, instructions on how to install and how to run scripts.

Dockerization

- I try to dockerize it but I face the problem of lacking disk space.
- Any available workstation for dockerization?

Plan for the Next Week

Transfer code to AISG

Need a workstation to test the inference time because my own PC don’t have enough space.

Check other method’s performance.

Inverse Text Normalization

ZHANG MENGAO

WEEK 8

Dockerization

- Live demo
- 2.8 Gigabytes
- It takes a few seconds to load
- Around 0.2 second to do inference

LLM – Inference on Local CPU

- LLM used: GPT-Neo-1.3B (relatively small)
- Takes more than 8 GB RAM to load
- Takes around 6 GB RAM after loading is done
- Takes about 8 seconds to generate a sequence of 30 tokens
- Not suitable on PC CPU

Other Methods

- Most methods has their limitations
- Most methods don’t open-source their code and model
- One interesting method [1]
  - It propose a new data preparation pipeline.
  - It can solve punctuation restoration, Capitalization, and ITN
  - Open-sourced
  - However, the evaluation metric is WER. Cannot directly compare with Thutmose Tagger.

[1] Benjamin Suter, Josef Novak, *Neural Text Denormalization for Speech Transcripts.* Interspeech2021

LibriHeavy Dataset

- Data source: LibriVox project (audiobook)
- Derived from LibriLight Dataset
  - LibriLight Dataset don’t have labels.
  - Used for unsupervised development
- How to get label:
  - Text and audio are both available.
  - The hard part is alignment: they used an ASR model to get raw transcript then apply some algorithm to get alignment.
  - Raw transcript is available in the dataset
  - Might be helpful for ITN
  - Hard part to adopt it for ITN: selecting data & determining ITN span

Plan for the Next Week

- Check the performance of Neural Text Denormalization for speech scripts
- Work on Json API
- Get the statistic of LibriHeavy Dataset

Inverse Text Normalization

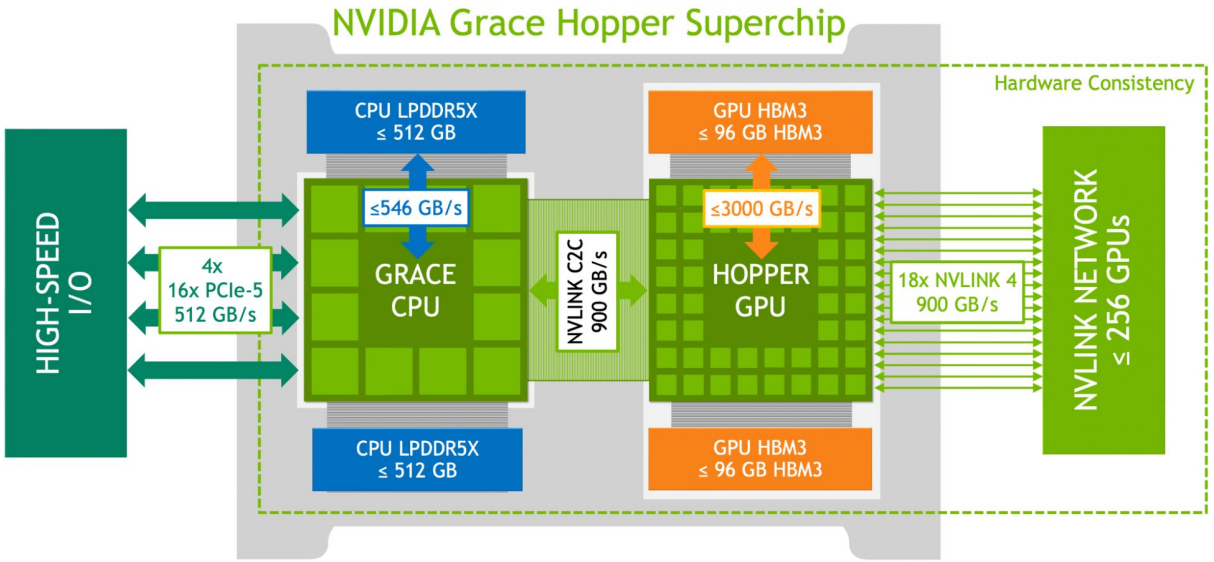
ZHANG MENGAO

WEEK 9

Dockerization

- Current version support API calls
- Instructions on how to run is added to the overview of the repository

GH200 Grace Hopper Superchip Platform



- Faster connection
- 96 GB GPU memory
- High extensibility
- More powerful than A100 GPU

Neural Text Denormalization

- Methodology: prepare data based on GTN, then train a machine translation model on the data
- Authors' pre-trained models are expired on github
- Same issue was mentioned in its github page- but no response
- A live website demo is available in https://denormalizer.spitch.ch/app
- ASR test results show that it is good at numbers, capitalization, but not so good at converting currency units
- Need to train from scratch.

Statistics of LibriHeavy

- Three splits for training set:
  - Small: 122526 (7578 with numeric terms)
  - Medium: 1101040 (57432)
  - Large: 11156939 (798950)
- Evaluation sets:
  - Development set: 5348 (315)
  - Test-clean: 2557 (66)
  - Test-clean-large: 26127 (913)
  - Test-other: 2815 (237)
  - Test-other-large: 24681 (1965)

Checking Numbers in LibriHeavy

- Most are invalid: “[Illustration: image249]”, “(Vv. 3663 3930.)”, “[1]”, or random number not pronounced.
- Valid:
  - Mostly time & dates
  - Some numbers are with units to express temperature, weight, etc.
  - Normal numbers appears mainly for chapters index

Plan for the Next Week

- Train neural text denormalization model
- Transfer the docker image to AISG
- Figure out how to clean LibriHeavy

Inverse Text Normalization

ZHANG MENGAO

WEEK 11

Docker

- For docker image, here's the link to the docker hub page:
  - https://hub.docker.com/repository/docker/mario6666667/itn/general
- The overview in this page contains the instructions on how to run the image as well as how to call the api.

Text Denormalization Model

- Trained using its training script and GTN dataset
- Results on ASR output is similar compared to the pre-trained model on the web demo

Evaluating Text Denormalization Model

- The text denormalization model adds punctuation mark to the output
- Thutmose Tagger does not add punctuation mark but will add symbols that carry semantic meanings, e.g. &, %
- Evaluate Using metrics in text denormalization paper vs Thutmose Paper
  - Thutmose Paper's evaluation method have the reference files ready and have multiple references on each semiotic instances. To use this method, we need to remove punctuation in the denormalization model output.
  - Text denormalization paper's evaluation method calculates different WER, i.e., overall WER, copy word WER (cWER), punctuation marks (pWER), digit sequence (dWER), capitalized word (uWER). They define the punctuation marks as all characters from the set {.,:;!?/-}. May need to modify a lot of code to evaluate Thutmose tagger's output properly.

Evaluation on Text Denormalization Model using Thutmose method

- A set of rules for removing symbols:
  - https://docs.google.com/document/d/13iw0CYgtIVmEupFPlCi8mB1s7j0Hc5icKLMyhrtukRA/edit

Plan for the Next Week

- Apply the rules to evaluate text denormalization model with Thutmose tagger
- Evaluate text denormalization according to the original paper to check whether its performance is similar as reported.
- Apply NeMo TN and ITN on Libriheavy dataset.

Inverse Text Normalization

ZHANG MENGAO

WEEK 12

Evaluating Text Denormalization Model - Evaluation protocol

- : ; ( ) ? are removed from both the reference and the model output
- Model output changes:
  - Y= ⟶ ¥
  - ! is removed if it is not following Yahoo or yahoo.
  - . is removed if it is the last or the second last character in the string.
  - , is removed if the letter immediately before it is not numbers.
- Reference changes:
  - £ ⟶ ps, µm ⟶ um
  - , is removed if the characters surrounding it are all numbers

Evaluating Text Denormalization Model - Default test set

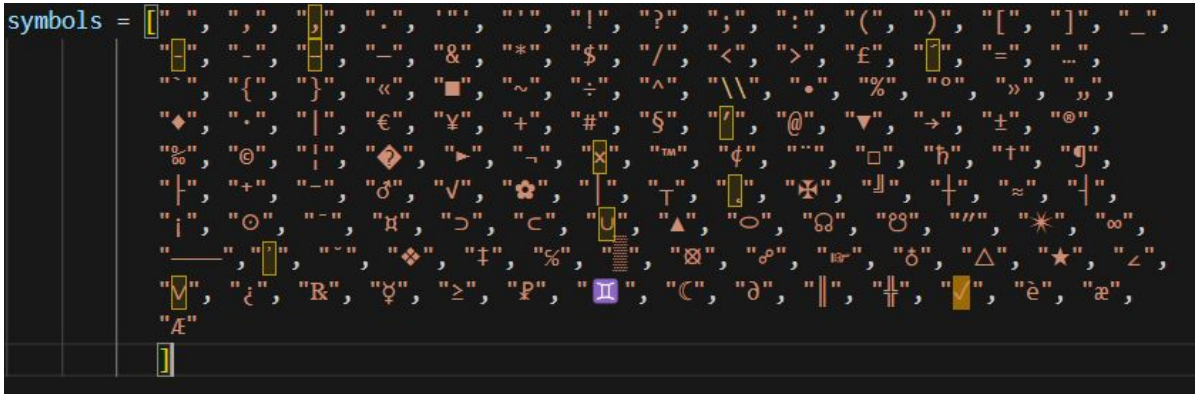
- Text Denormalization Model:
  - sentence accuracy: 88.92% (6638/7465)
  - digit errors: 0.28% (21/7465)
  - other errors: 10.80% (806/7645)
    - quote errors: 6.19% (473/7645) (text denormalization model tend to convert ‘ into random words)
    - hyphen errors: 1.03% (79/7645) (text denoramlization model tend to add hyphen more than needed, e.g. date)
- Thutmose Tagger:
  - sentence accuracy: 97.49% (7278/ 7465)
  - digit errors: 0.33% (25/7465)
  - other errors: 2.17% (162/7645)

Evaluating Text Denormalization Model - Hard test set

- Text Denormalization Model:
  - sentence accuracy: 78.34% (7533/9616)
  - digit errors: 3.16% (304/9616)
  - other errors: 18.50% (1779/9616)
    - quote errors: 6.06% (583/9616)
    - hyphen errors: 4.79% (461/9616)
- Thutmose Tagger:
  - sentence accuracy: 87.93% (8455/ 9616)
  - digit errors: 3.16% (304/9616)
  - other errors: 8.91% (857/9616)

Libriheavy Preprocessing

- Over 100 kinds of symbols.
  - Some of them need to be removed
  - Some of them are pronounced
  - Some of them need to be converted to other common words.



Libriheavy Preprocessing

- Current approach of finding components that are not pronounced:
  - Calculate character level edit distance from written form to spoken form with higher cost on deletion
- Found that if there are “illustration” or completely numbers inside “[]”, “[]” and the content inside it is not read and pronounced
- Found that if numbers or roman numeral appears at the beginning of the sequence, it is not pronounced.

Plan for the Next Week

- Should we restrict the split of Libriheavy to explore and to use in the future?
  - medium for now
- Keep preprocessing Libriheavy
- check instruction on audiobook website
- Evaluate text denormalization model using the method in the original paper
- check text denormalization model’s performance after correcting false errors
- Check inference time of text denormalization model

Inverse Text Normalization

ZHANG MENGAO

WEEK 13

Processing LibriHeavy – Guide from Web

- Only said something about footnote
- Footnote policy may vary depend on book coordinator.
  - All read out
  - All ignore
  - Specific policy

Processing LibriHeavy

- Common symbols: " ,.!?-_\"\'*/^=\\|:;()[]{}<>$&#%~+`@"
- Uncommon symbol: ––-¯„,…«»±×÷⁺†φΦØ¢→▶νΛ´¨ʻˆˇ⁰°·•□♦▼■🞐¶§
  £€¥áâàäåãĀÃÁÂÀÄÃÅćčĆČéêèëĖÉÊĚÈĖĒîìíîĨïĩÍÎĬIκķKMñÑoõóóôòöõŌÒÖÔÖTūúûŭùüůυύÚÙÜŰŰÜχýŶæÆŒœçÇÇðαά
  άάάàβγθεέéĖėĖèĖσђηήήήήρλμπωϙώ™®©¼½⅝¾¹¡²₂³⁴⁶⁸⁹‰°¦⊢τιι¬Γζξ♂
- Most uncommon symbols don’t appear often.
- Strategy to handle uncommon symbols:
  - Remove the entire example if it contains many uncommon symbols
  - Remove the symbols if it is not pronounced.
  - Convert it into common symbols if it carries similar meaning.

Evaluating Denormalization Model

- Inference time on CPU:
  - 15 ms to 25 ms depending on the input length
- The model now handles ‘ well.
- soft errors are introduced to check the errors that are not very serious
  - We consider an error as soft error if the edit distance between input sentence and target sentence is only 1.
  - Further classify as errors related to substitution and symbols

Evaluating Denormalization Model

- Using the method in original paper
  - The paper didn’t specify the evaluation dataset used.
  - Assume used GTN test set
  - Processed GTN test set by concatenating words into sentence because GTN only contain words for each line
  - Need additional processing on the evaluation dataset

Evaluating Text Denormalization Model - Default test set

- Text Denormalization Model:
  - sentence accuracy: 94.39% (7046/7465)
  - digit errors: 0.28% (21/7465)
  - other errors: 5.33% (398/7645)
    - soft errors: 3.05% (233/7645)
      - soft error related to subsitution: 0.09% (7/7645)
      - soft error related to symbols: 1.15% (88/7645)
- Thutmose Tagger Model:
  - sentence accuracy: 97.50% (7278/7465)
  - digit errors: 0.33% (25/7465)
  - other errors: 2.17% (162/7645)
    - soft errors: 0.73% (56/7645)
      - soft error related to subsitution: 0.01% (1/7645)
      - soft error related to symbols: 0.40% (31/7645)

Evaluating Text Denormalization Model - Hard test set

- Text Denormalization Model:
  - sentence accuracy: 83.60% (8039/9616)
  - digit errors: 3.03% (291/9616)
  - other errors: 13.37% (1286/9616)
    - soft errors: 6.08% (585/9616)
      - soft error related to subsitution: 0.12% (12/9616)
      - soft error related to symbols: 2.15% (207/9616)

- Thutmose Tagger Model:
  - sentence accuracy: 87.93% (8455/9616)
  - digit errors: 3.16% (304/9616)
  - other errors: 8.91% (857/9616)
    - soft errors: 3.04% (293/9616)
      - soft error related to subsitution: 0.24% (24/9616)
      - soft error related to symbols: 0.48% (46/9616)

Plan for the Next Week

- Get accurate result for evaluation on denormalization model using the method in original paper
- Keep preprocessing Libriheavy
- packing denormalization model into docker image

Inverse Text Normalization

ZHANG MENGAO

WEEK 14

Denormalization performance

TYPE	ERRORS	TOTAL	RATE
Sentence	1836268	4290238	42.80%
Word	5051282	51509573	9.81%
Copy Word	840051	24348049	3.45%
Uppercase Word	2029441	14954218	13.57%
Digit	650656	3602745	18.06%
Punctuation	1762211	8132809	21.67%
Symbol	54712	107213	51.03%

- TYPE indicate the type of metric
- ERRORS indicate the number of errors
- TOTAL represents the total number of instances of each type
- RATE is the error rate

Denormalization performance

TYPE	ERRORS	TOTAL	RATE
Sentence	1836268	4290238	42.80%
Word	5051282	51509573	9.81%
Copy Word	840051	24348049	3.45%
Uppercase Word	2029441	14954218	13.57%
Digit	650656	3602745	18.06%
Punctuation	1762211	8132809	21.67%
Symbol	54712	107213	51.03%

- For the metrics below, no special treatment is applied, i.e., it directly compares model output and reference sentence.
- Sentence represents sentence accuracy considering all error types.
- Word is on word-level. Rate is the word error rate.

Denormalization performance

TYPE	ERRORS	TOTAL	RATE
Sentence	1836268	4290238	42.80%
Word	5051282	51509573	9.81%
Copy Word	840051	24348049	3.45%
Uppercase Word	2029441	14954218	13.57%
Digit	650656	3602745	18.06%
Punctuation	1762211	8132809	21.67%
Symbol	54712	107213	51.03%

- For the metrics below, only words satisfying respective standards are selected from model output and reference sentence.
- Example: Calculating punctuation:
  Und auch wer nur hier im Land operiert , zahlt nur zehn Prozent Steuern . → [, .] (reference)

  Und auch wer nur hier im Land operiert zahlt nur 10 % Steuern . → [.] (hypothesis)

Denormalization performance

TYPE	ERRORS	TOTAL	RATE
Sentence	1836268	4290238	42.80%
Word	5051282	51509573	9.81%
Copy Word	840051	24348049	3.45%
Uppercase Word	2029441	14954218	13.57%
Digit	650656	3602745	18.06%
Punctuation	1762211	8132809	21.67%
Symbol	54712	107213	51.03%

```
CHARACTER SETS:
Copy chars:     abcdefghijklmnopqrstuvwxyz
Punctuation:    ,;:.?!/-()"
Symbols:        % & ° § + EUR CHF USD GBP m mm cm km m2 m3 km2 l dl ml g kg
```

- Copy Word are words that should identical in both written and spoken form. The word is selected if all characters of it belongs to the copy chars in charset
- Uppercase Word is literally the words that needs to be capitalized. The word is selected if there is at least one character is capitalized.
- Digit indicates the words that contain digits. The word is selected if there is at least one digit.
- punctuations and symbols are defined in the character set in the attachment. The word is selected if the word is in the respective charset

Comment on performance of denormalization model

- The result of the model we trained (large version) is very similar to the small version in the original paper except for digits.
- The model we trained made more mistakes on digits.
- This might because that the test set I used is the whole test set of GTN where the test set in the original paper is a subset of GTN and it is not specified.

TYPE	ERRORS	TOTAL	RATE
Sentence	1836268	4290238	42.80%
Word	5051282	51509573	9.81%
Copy Word	840051	24348049	3.45%
Uppercase Word	2029441	14954218	13.57%
Digit	650656	3602745	18.06%
Punctuation	1762211	8132809	21.67%
Symbol	54712	107213	51.03%

	BLEU	words (WER)			copy words (cWER)			punctuation (pWER)			digits (dWER)			capitalization (uWER)		
		errors	tokens	rate	errors	tokens	rate	errors	tokens	rate	errors	tokens	rate	errors	tokens	rate
EN-S	82.0	10 425	117 171	8.90	1 944	56 216	3.46	3 659	17 658	20.72	542	7 742	7.00	5 430	34 556	15.71
EN-L	87.2	7 452	117 171	6.36	1 304	56 216	2.32	2 602	17 658	14.74	390	7 742	5.04	3 948	34 556	11.42

Docker image

- The docker image is of size 9.3GB
- Reason:
  - It depends on fairseq library which is an integrated and heavy library.
  - Fairseq library depend on a lot of other libraries.
- Possible solution:
  - Uninstall some of the dependencies after installing fairseq
  - Search for lightweight version of fariseq or other library compatible with fairseq

Plan for the next week

- figure out how to reduce the size of the image
- continue preprocessing libriheavy

Inverse Text Normalization

ZHANG MENGAO

WEEK 15

Docker

- Total size (including the model itself): 2.8 Gigabytes
- How did I shrinked the size:
  - Fairseq depend on PyTorch
  - If not installed before the installation of fairseq, it will automatically install GPU version of PyTorch
  - I first installed the CPU version, then installed fairseq
- link and instructions:
  - https://hub.docker.com/repository/docker/mario6666667/itn/general

Libriheavy Preprocessing

- Strategy:
  - Three subsets:
    - Alphanumeric characters: a-z A-Z 0-9
    - Common Symbols: , . ! ? - _ \ " \ ' * / ^ = \ \ | : ; ( ) [ ] { } < > $ & # % ~ + ` @
    - Uncommon Symbols: – — - ¯ „ , … « » ± × ÷ ⁺ † φ Φ Ø ¢ → ▶ ν Λ ´ ¨ ′ ˆ ˜ ₀ ° ° · • □ Δ ♦ ▼ ■ ◆ ¶ $
      £ € ¥ á â à ä å ã Ā Ă Â À Ä Å Ą ć č Ć Č é ê è ë Ė É Ê È Ë Ę ī î ì ï Ī Í Î Ì Ï ı ķ Ķ M ń Ñ ο õ ó ô ò ö ō Õ Ò Ö Ô Ő T ū ú û ù ü Ů ų Ú Ù Ü Ű Ů χ ý Y æ Æ Œ œ ç Ç ð α ά á à ä å ã β γ θ
      ε é ế ë ê è σ ђ η ή ń ǹ ρ λ μ π ω ų ŵ ™ ® © ¼ ½ ¾ ¹ ₁ ² ₂ ³ ⁴ ₅ ⁶ ⁸ ⁹ ‰ ° ¦ ├ τ ι ¬ Γ ζ ξ Ξ σ
  - Deal with uncommon symbols first.
    - Count the frequency of each uncommon symbols
    - Filter out the instances that contains very rare symbols (appears less than a threshold)
    - iteratively do the filtering until a manageable number of symbols are kept. Keep the interesting symbols while do the filtering.
  - Then deal with symbols in a case by case manner.

Libriheavy Preprocessing

- Interesting finding:
  - Most uncommon symbols come from other languages – discarding the instance would be helpful if we focus on English
  - Other uncommon symbols can be directly deleted.
  - Some of the uncommon symbols are just not standard ASCII characters – convert to standard ASCII characters.

Plan

- Keep working on preprocessing Libriheavy

Inverse Text Normalization

ZHANG MENGAO

WEEK 16

Preprocessing Libriheavy

- Removed instances containing the following symbols: 243 instances in total
  - Chinese Characters
  - ¯±†Φˆˇ₀□Δ¶ĀȦćČΈEi̇i̒ìîķKòõÒTůÚŰȕάἕὲȅήῳ¼⅝₈₅⁶₆⁸₈⁹‰├♂⁺˙˙⁰▼EË̃ĩὺὐüΥάθέέ ℏ ἡήωῷ⁴ºΓζξ¢ ►åÏlóύάὴ¾³Ξ×→íχὰἐ¥AÅMδἀ¦φčí₂Ôμυ♦ Ȇ̇ÜÇηπ¹₁κςγετἵΐη²ονλσ
- All of these appear very rarely: less than 20 times
- Some of them are Latin letters
- Some of them appear in the written form because of some weird error, i.e., it converted the normal word to random letters. This case is discovered by checking the transcript.
- Some appear as mathematical or chemcal symbols.
- These instances are removed because they are rare and does not help the training.

Preprocessing Libriheavy

- Remaining uncommon symbols: ρ®·ãÀÃÈÍÑù©Λý„™’òūŒìβ  α½ØŭÖĚ€îäûÉ§Ć÷Ō■…Áç» ‚Äï´ÆŌ«àü•öëÂê¬°úâœô£íñóæá-è–é—
- Keep: αβρ£°´
- Remove Instance: ®ù©„™’  €■»« ([´)
- Remove Symbol in the sentence: (SCHÖN.) (‚ÄĚ) • (Â…) (Â Â Â Â) (¬)
- Convert:
  - · → . Λv→w ŌĆØ → ’ ŌĆ£ → ” ÷ → o
  - Œ → oe Æ → e? œ → oe æ→ae

Preprocessing Libriheavy

- Unsure:
  - ãàÁÀÃÄÈÍÑòūìŭúüîïäûÉçïōöôëêéèâñóáý (accent / named entity) (replace/keep)
  - ½ (half) .5 (1 / 2) check slash, date?
  - § (part/paragraph/not pronounced) (we remove symbol in written and part/paragraph in spoken)
  - … (punction) (remove)
  - - (connecting two words to get an adjective (convert)/ many - to get tabular content (remove))
  - – (Appear at the beginning of a book in the book number. Transcription will also appear: read by xxx for librivox, 2539 frequency) (get alignment and remove the beginning of both forms)
  - — (Explaination, punctuation?) (convert to ,)
  - -

Plan

- Keep working on preprocessing Libriheavy – common symbols
- get alignment (reusable)
- get percentage of dropped instances

Inverse Text Normalization

ZHANG MENGAO

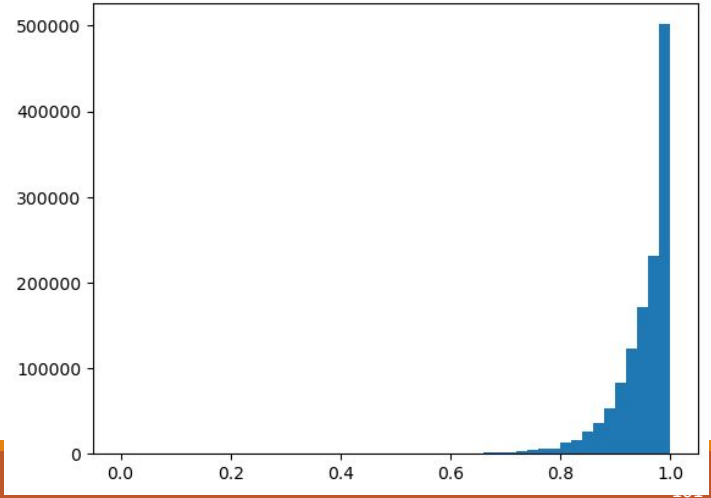
WEEK 17

Preprocessing Libriheavy

- Applied most rules we discussed last week
- Initial number of lines (all subsets excluding large): 1,285,094
- After cleaning: 1,284,094
- Dropped lines: 1,000
- Dropped percentage: 0.08%
- Left undone:
  - ãàÁÀÃÄÈÍÑòūìŭúüîïäûÉçïōöôëêéèâñóáý (accent / named entity) (keep because appeared only about ~10,000 times. <1%)
  - § (part/paragraph/not pronounced) (using the alignment to remove accordingly)
  - – (Appear at the beginning of a book in the book number. Transcription will also appear: read by xxx for librivox, 2539 frequency) (get alignment and remove the beginning of both forms)

Alignment

- difflib.SequenceMatcher (Ratcliff/Obershelp algorithm)
- basic idea:
  - find the longest contiguous matching subsequence (or "snake") between two sequences
  - After found a snake, apply the same step on the sequences before and after the snake
- It return a similarity ratio between the two strings
  - The distribution of similarity ratio is as shown
  - left skewed
  - Most spoken and written pairs are similar
  - ~ 500,000 instances are identical



Alignment

- With this method, we can get the alignment between two strings.
- Example:
  - ratio: 0.925
  - written: His manner was rather boyish and diffident, and wholly apologetic, and the All in One glistened in his hand like a razor, or a revolver, or anything terrible and destructive that a startled camp cook might make it out to be.
  - spoken: his manner was rather boyish and diffident and wholly apologetic and the only one glistened in his hand like a racer or revolver or anything terrible and destructive that a startled camp cook might make it out to be
  - {all in} ⟶ {only}
  - {razor} ⟶ {racer}
  - {a} ⟶ {}

Alignment

- With this, we can find the cases we are interested and customize the alignment method.
- For example, we can to find mismatches that contain uncommon symbols.
- We can also find mismatches start at the beginning
- A lot more to explore
  - The correctness of the alignment
  - How to customize the method according to our need

Plan

- Explore the usage of the alignment method and clean Libriheavy
- Combine the preprocessing steps with Changsong

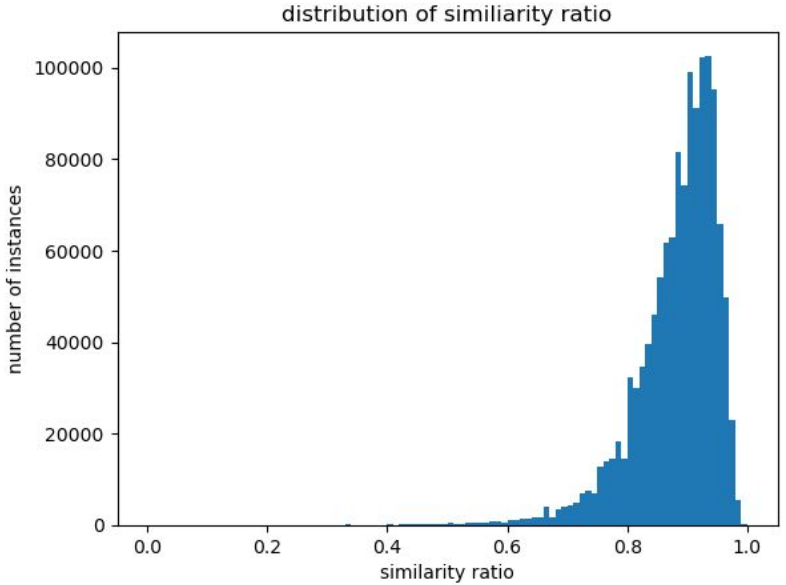
Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 18-19

Alignment

- This method (difflib.SequenceMatcher (Ratcliff/Obershelp algorithm)) can detect all the differences between two strings, i.e. it doesn't cause false negative.
- Thus, the "not real" differences (false positive) is the focus.
- Method:
  - 1. Convert both strings to lower case
  - 2. insert spaces around symbols (to separate it from the word)
  - 3. Use this method to find the alignment
  - 4. find interesting cases
- Distribution: left-skewed gaussian distribution

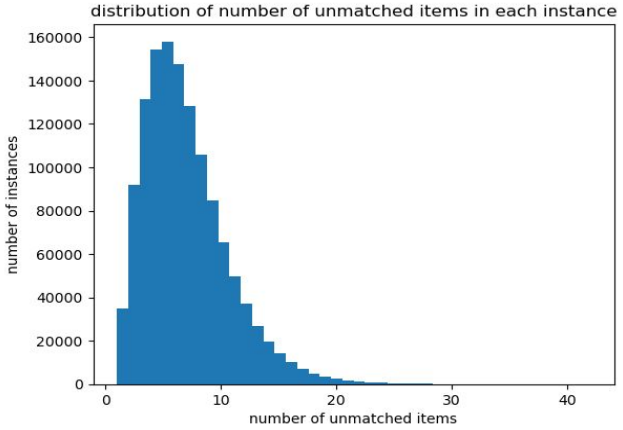


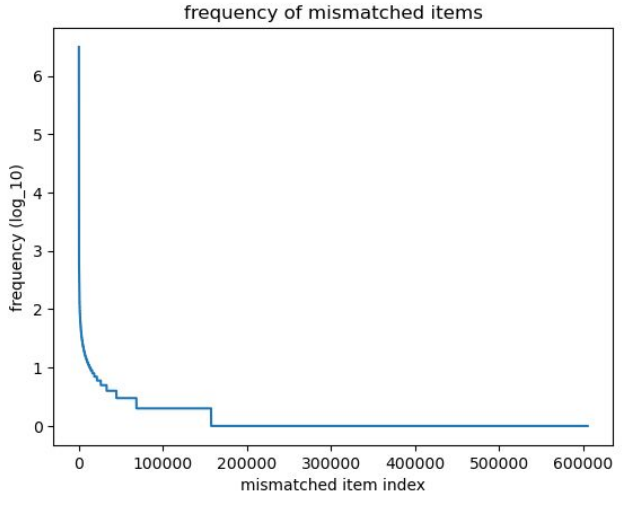
Alignment

- With this method, we can get the alignment between two strings.
- Example:
  - ratio: 0.925
  - written: His manner was rather boyish and diffident**,** and wholly apologetic**,** and the **All in** One glistened in his hand like a **razor,** or **a** revolver**,** or anything terrible and destructive that a startled camp cook might make it out to be**.**
  - spoken: his manner was rather boyish and diffident and wholly apologetic and the **only** one glistened in his hand like a **racer** or revolver or anything terrible and destructive that a startled camp cook might make it out to be
  - {,} → {}, {,} → {}, {all in} → {only}, {razor} → {racer}, {,} → {}, {a} → {}, {,} → {},

Alignment

- Total number of instances: 1,284,094
- Total number of mismatches: 8,515,488 (words %)
- Mismatches that doesn't contain any alphabetics: 7,020,211 (82.44%)
- Number of unique mismatches: 605,522
- Most mismatched items appear rarely
- Unique Mismatched items with frequency less than 10: 593,164 (97.96%)
- Strategy:
  - Consider the known cases first
  - Explore common patterns of the mismatches
  - Ignore infrequent mismatches if there's no common pattern





Alignment - Known cases

- ãàÁÀÃÄÈÍÑòūìŭúüîïäûÉçïōöôëéèâñóáý (Can be replaced automatically)
- §
  - examples: {§ 1}->{one}, {§ 1}->{part/paragraph/section one}, {§ 173 .}->{}
  - Processing:
    - remove § and part/paragraph/section
    - remove number if the spoken form is empty
- –
  - Sometimes the same as - (Convert to -)
  - Sometimes appear at the beginning of a book in the book number (remove that part)
  - Also need to check whether the second scenario will happen on normal -

Alignment - Previously unknown cases

- Current observation(Normal cases):
  - {<punctuation>} → {}
  - common written to spoken conversion ({mr .} → {mister})
  - american & british style
- Current observation (Unnormal cases):
  - {<simple word>} → {} or {} → {<simple word>}
  - {round}->{around}, {toward}->{towards}, {goin '}->{going}
  - {"}->{quote}
- P.s. <punctuation> refers to periods or other punctuations not its literally meaning. same applies to <simple word>

Alignment - Previously unknown cases

- Possible directions to explore:
  - Check common patterns of the mismatched items.
  - Check frequent mismatched items
  - Check long mismatched items
  - Check mismatched items related to symbols
- Problem:
  - requires much labor
  - current way of counting frequency of mismatched items follows exact string matching. One possibility: many mismatches may follow the same pattern but may only appear once. As a result, such cases got buried in the infrequent mismatches.

Alignment – Finding common patterns

- One solution: sequential pattern mining
  - Treat mismatch {x y} ⟶ {z} as two sequences. x, y and z are the words or symbols and are regarded as items
  - Find frequent sequential patterns
- Example:
  - We have the following mismatches and respective frequency count:
    - ({x} ⟶ {z}, 100), ({y} ⟶ {z}, 50), ({x, y} ⟶{z}, 30)
  - Then {x} ⟶ {z} will have the support 100+30 = 130, and {y} ⟶ {z} will have the support 50+30 =80.
  - Using this, we can detect sequential patterns like: {[Illustration: xxxx]} ⟶ {}. Although the words following Illustration are different across mismatches.

Plan

- Keep exploring the cases that need to be cleaned
  - manual check
  - Sequential pattern mining
- Discuss with Changsong on how to improve preprocessing steps using alignment

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

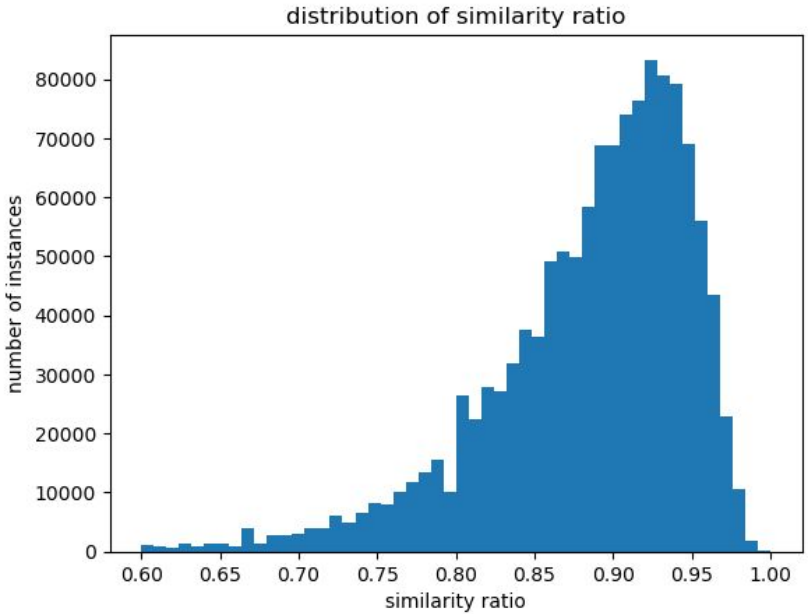
WEEK 20

Preprocessing Libriheavy - Overview

- Remove/convert non-ASCII characters (Done)
  - Rare, erroneous and not pronounced characters are removed.
  - Characters that carries special meanings are kept (αβρ£°)
  - Latin characters are converted into Alphabetics (e.g. é)
  - Other characters are converted to ASCII characters according to specific rules.
- Remove not aligned fragments in written and spoken text (In progress)
  - Written: remove not pronounced fragments
  - Spoken: remove misaligned fragments at the beginning/end
  - Challenge: hard to find the pattern of misaligned fragments

Preprocessing Libriheavy - Current progress

- Finished processing all non-ASCII characters.
  - Relevant codes are refactored and well documented
- Processing utilizing the alignment
  - Dropped instances with similarity ratio less than 0.6. Because there are very few instances satisfying this and most of them are corrupted
  - Total number of dropped instances: 7867 (can drop more)
  - Percentage: 0.61%
  - ratio of mismatched words/symbols to all words/symbols: 4.013781304056187%



Preprocessing Libriheavy - sequential pattern mining

- Public library available
- Current found sequential patterns:
  - {} → {read, by} (appear at the end of a book, “read by xxx”)
  - {} → {end, chapter} (appear at the end of a chapter, “end of chapter xxx”)
  - { [, illustration,] } → {}
  - {42} → {}, {002} → {}, {003} → {}, {004} → {}, {005} → {} (needs investigation)
  - {page} → {} (Page number are not read out)

Plan

- Keep exploring the sequential patterns
  - Needs more exploration and verification
- tidy up the newly written code

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 21

Preprocessing Libriheavy - summary

- 1. Found all uncommon symbols slide 91
- 2. Remove instances that contains rare symbols (with frequency less than 10) slide 95
- 3. Apply specific rules on each special symbol (rules: slide 96 slide 97)
- 4. Using alignment and sequential pattern mining to find frequent misaligned patterns and corrupted instances.
  - Alignment example: slide 107
  - Using alignment to help with applying rules in step 3: slide 109
  - Observed patterns using only alignment: slide 110
  - Observed patterns using both alignment and sequential pattern: slide 121
- 5. Labeling punctuations by Changsong (For punctuation restoration)

Preprocessing Libriheavy - sequential pattern mining

- {} → {read, by} (appear at the end of a book, “read by xxx”)
- {} → {end, chapter} (appear at the end of a chapter, “end of chapter xxx”)
- { [, illustration,] } → {}
- { [, FN, #,] } → {}
- {42:00x:xxx} → {}
- {page} → {} (Page number are not read out)

Pros and Cons of current preprocessing method

- Pros:
  - control over the rules to process. e.g. how to deal with each symbol and its effect
  - Reserved most of the data while cleaned noise in the data
- Cons:
  - Many steps are laborious and specific to Libriheavy.
  - Not automated, requires manual checking
  - May not be generalizable to other collected parallized corpus.

Possible Directions

- Improve the preprocessing method to make it more generalizable.
  - Rules for uncommon symbols are less generalizable
  - Alignment and sequential pattern mining are more generalizable to find the pattern, but the patterns need to be manually filtered.
- Combine with Changsong's work, i.e., use the preprocessed dataset to train Large Language Model not only to do punctuation restoration but also ITN, and Capitalization.

Plan

- Start training models on the dataset
  - Translation model
  - Four in one multi task model
- Clean up the code and write documentation

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 22

Last week’s plan

- Ways to make my method generalizable
- Train denormalization model on the processed dataset

How to make the data preparation method generalizable

- The problems trying to solve:
  - Yufei's work: Try to build a dataset for ITN with word-level alignment. Similar to Google Text Normalization Dataset. Assuming each word or phrase in written form corresponds to a word or phrase in spoken form.
  - My method (Alignment + Sequential Pattern Mining): Tries to eliminate the cases where some words in written form doesn't have corresponding words in spoken forms and vice versa.
  - Thus, these two methods are complementary.

How to make the data preparation method generalizable

- Yufei's work:
  - Statistical approach, no manual checking;
  - Can get word-level alignment
  - Assume correspondence is always present. This case might be true if the collected data is clean.
  - The resultant dataset can be used for all kinds of models
- Mine:
  - exact match, will detect any form of misalignment including normal transformations between spoken and written.
  - Can clear words that have no correspondence in the other form.
  - The word-level alignment may not be accurate because it is not a statistical method.
  - The resultant dataset can only be used in Translation models.

How to make the data preparation method generalizable

- Alignment + Sequential pattern mining: The word-level alignment may not be accurate because it is not a statistical method.
- Example for the statement:
  - Written form: I was born in 1969. I may be too old for this
  - Spoken form: i was born in umm nineteen sixty nine i may be too old for this lah
  - My method will first find all the strings that are identical and consider the the rest to be misalignments. The output will be:
    - {I was born in} → {i was born in} (Match)
    - {1969 .} → {umm nineteen sixty nine} (Mismatch) (We can't do much in this case, but if "umm" appear very often, we can remove it)
    - {I may be too old for this} → {I may be too old for this} (Match)
    - {} → {lah} (Mismatch) (We can remove lah in this case, because the left side is empty)
- In conclusion, our method is helpful to remove some of the redundant words but only the frequent ones.

Training denormalization model on Libriheavy

- Example of trained model output:

```
sheikhr el bahr el bahr el bahr el khr r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r
r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r r o o o o o o o o o o o o o o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o !
```

- Reason:
  - 1. The Libriheavy still have noise after cleaning, i.e. The written form text have additional text that are not pronounced and it is very hard to find all of them
  - 2. The denormalization model’s output follows greedy decoding method, i.e. it will select the token with the highest probability. In model’s point of view, some random words will appear in the target output (written form) and there’s no sign of it in the input (spoken form). As a result, common words that are not pronounced will have higher probability than correct words and result in this erroneous pattern.

Training denormalization model on Libriheavy

- Solution:
  - Clean the dataset in a more aggressive way: clean all mismatches that are likely not pronounced, e.g. clean all mismatches with the following format {<any words>} → {<empty>}
  - This might solve the problem but may erase a lot of useful training data
  - Ideally, the not pronounced words should appear in the format shown in bullet point one. However, if such cases combines with legal conversions, there is still possibility that the not pronounced words remain in the written form, e.g. {Mr . r} → {Mister . } where the r is not pronounced.
  - If this solution doesn't work, Libriheavy may not be a good dataset for ITN because of noise in data.

Plan

- Clean Libriheavy more aggressively and retrain the model.
- Simplify and refactor the code
  - adjust the steps for dealing with non-ascii characters
    - remove all instances with non-ascii characters (check dropped rate)
    - use simple heuristic

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 23

Dataset Noise Types

- The noise in the dataset can be of the following types where Book Text produces the most noise.
- 1. Text that is present in Book Text but not in the ASR transcript (Red case):
  - This include: footnotes, illustration, weird formatting strings (e.g., 42:001:000), or any thing not pronounced
  - This will greatly affect the translation model's performance because the random text pops up in the target output with no signal in Input
- 2. Text that is present in ASR transcript but not in Book Text (Blue case) :
  - This could be due to ASR error or the reader put additional conjunctions or disfluency.
- Example:
  - Written form: Author: abc. I was b@rn in 1969 **[1]**, have a nice day. The price is $1
  - Spoken form: read by edf i was born in **ummm** nineteen sixty nine **and** have a nice day the price is on

Dataset Noise Types

- 3. Incomplete start and end in Book Text and ASR transcript for each instance (Red case)
- 4. Wired erroneous words in Book Text, e.g. `tne$r` The correct word should be “their” (Blue case)
  - Might because the Book Text is obtained by scanning paper books and that scanning process cause errors.
  - Usually appear together with strange symbols
- Example:
  - Written form: **Author: abc.** I was b@rn in 1969 [1], have a nice day. The price is **$1**
  - Spoken form: **read by edf** i was born in ummm nineteen sixty nine and have a nice day the price is **on**

Data preparation pipeline (revised)

- Previous methods too complicated and cannot get clean enough data
- 1. Discard instances that contains non-ascii characters (with exceptions because it has special meaning: αβρ£°´) and bad ascii characters (because the instances containing these symbols are erroneous: +=@\|). Reason:
  - a.didn't drop too much instances: 63927 (4.97%)
  - b. Previously hand crafted rules may not properly handle all cases. By only defining the symbols we want to keep, we can better reduce the noise while saving labor.
- 2. Clean the Book text using regular expression with simple heuristics, e.g. replace multiple space with a single space.
- 3. Clean the Book text and ASR Transcript using alignment (details provided in the next slide)
- 4. Clean the Book text and ASR Transcript using sequential pattern mining

Clean using alignment

- Remove mismatches at the beginning and end of sentences (Red case)
- Remove mismatches that either one side is empty (Blue case) (First just remove mismatch in written)
- Remove mismatches that contains symbols not in the allowable set (orange case) (count the number of instances dropped)
- Possible drawbacks: may remove useful information. Is it acceptable?
- Written form: Author: abc. I was **b@rn** in 1969 [1], have a nice day. The price is $1
- Spoken form: read by edf i was born in **ummm** nineteen sixty nine and have a nice day the price is on
- The purple case could be solved by sequential pattern mining

Plan

- The code for the first three steps are rewritten and done. Finish the last step - sequential pattern mining
- Train the Denoramlization model to see the effect.

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 24

Effect of First Three Steps

- Kept symbols:
  - ASCII: !"$&',.:;?
  - Non-ASCII: αβρ£°´
- Number of instances:
  - Before cleaning: 1,285,094
  - After cleaning: 1, 203,004
  - Dropped (total) : 82,090 (6.39%)
  - Dropped because of ascii characters: 12,424 (0.97%)
  - Dropped because of non-ascii characters: 69,666 (5.42%)

Sequential Pattern Mining- Explained

- Terminology:
  - Item: the most basic element in the pattern. Words or symbols in this case, e.g., “and”, “?”, “101”
  - Sequence: formed by items with order, e.g. [“an”, “apple”, “tree”]
  - Subsequence: sequence 1 is a subsequence of sequence 2 if and only if:
    - length of sequence 1 is not longer than sequence 2
    - all items in sequence 1 appeared in sequence 2 in monotonic order
  - Subsequence example: Sequence: [“a”, “b”, “a”, “c”]; subsequences:
    - [“a”], [“b”], [“c”], [“a”, “a”], [“a”, “b”], [“a”, “c”], [“b”, “c”], [“b”, “a”, “c”], [“a”, “b”, “a”, “c”]
- Given a database of sequences, sequential pattern mining tries to find the subsequences that appears frequently (high support).

Sequential Pattern Mining- Explained

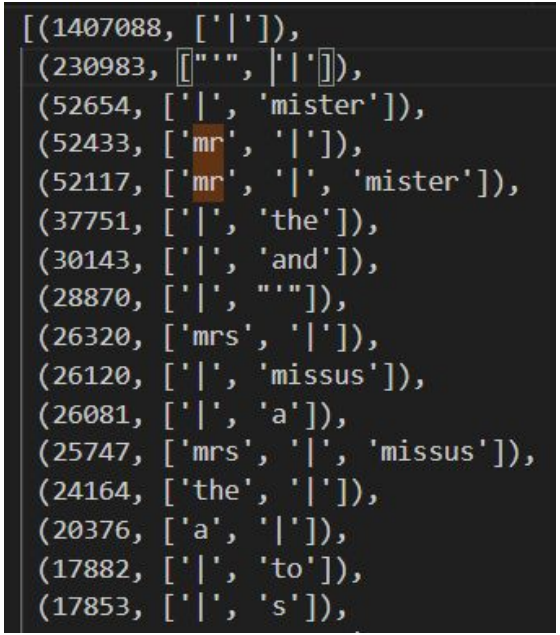
- Procedure:
  - Iterate through each sequence.
  - For each sequence, find all its subsequences and increase their support (frequency) by one.
  - After iterating all the sequences, we get the support for every subsequence
  - Set a threshold to filter out subsequences with low support.
- Example:
  - Sequences: [1, 2], [1], [2, 3]
  - Support for subsequence (underlined): ([1], 2), ([2], 2), ([3], 1), ([1,2], 1), ([2, 3], 1)
- Property: the support of subsequence 1 is not higher than subsequence 2 if subsequence 2 is a subsequence of subsequence 1, e.g., support for [1,2] must not higher than [1]

Sequential Pattern Mining- How to Utilize

- Consider mismatches as sequences
- Each word or symbol in the mismatch is an item
- Filter out the symbols we want to keep, e.g. period, comma
- Focus on words and single quote because it also appears frequently in ASRTranscript
- To separate Book Text from ASR transcript in the sequence, add a “|” in between the two.
- Example conversion from mismatch to sequence:

  ({“1”, “mr”, “.”} → {“mister”}) —> [“1”, “mr”, “|”, “mister”]
- As a result, the sequential patterns will cover all frequent mismatched items with ordering information.

Example of found patterns



Sequential Pattern Mining- How to Utilize

- What we can get from sequential patterns
  - The words in Book Text (including the empty case) that are frequently mapped to specific words in ASR Transcript (including the empty case)
- What we want from sequential patterns
  - The words in Book Text that are frequently mapped to nothing in ASR Transcript (more important?) and vice versa. (one side is empty)
  - Hard to determine the correctness when both sides are not empty. e.g. ['t', '|', 'to'], ['duane', '|', 'twain'], ['tomorrow', '|', 'to', 'morrow'] (pronounced similarly)

Sequential Pattern Mining- How to Utilize

- However, because of property mentioned in slide 142, the patterns are overlapped.
  - Example:

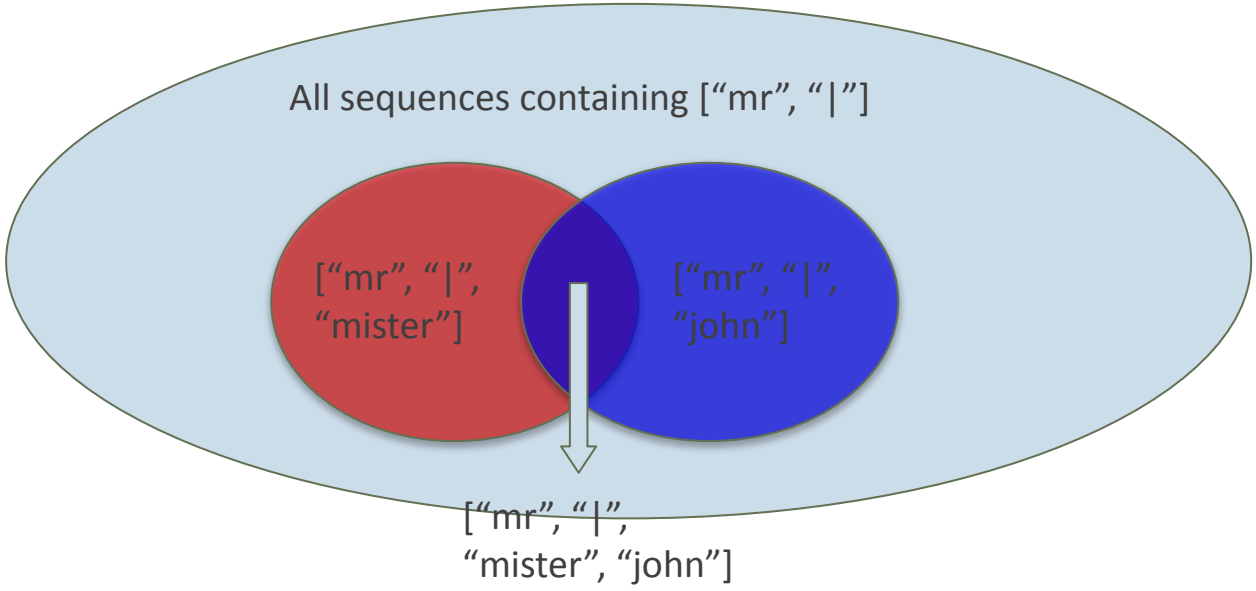
```
(52433, ['mr', '|']),
(52117, ['mr', '|', 'mister']),
```

- We want to seperate the patterns and get their real support.
- Method I can think of:
  - 1. Sort the patterns based on the length, from longest to shortest
  - 2. Iterate through the patterns starting from the longest pattern. Check whether other patterns are subsequence of the current pattern. If yes, decrease the support of subsequence by the support of current pattern.
    - Example:

```
(316, ['mr', '|']),
(52117, ['mr', '|', 'mister']),
```

Sequential Pattern Mining- How to Utilize

- However, the method above is wrong because the subsequences are also overlapped.
- Need to construct a tree structure and propagate the support calculation accordingly



Model performance without using Sequential Pattern Mining for cleaning

- Cannot convert some words correctly, output weird words for specific input words
- Cannot translate numbers
- Can restore punctuation
- Can do Capitalization
- Possible reasons:
  - training data doesn’t contain much ITN examples (Find new dataset) (select instances with ITN)
  - training hyperparameters are not tuned well
  - Training set not large enough

Tentative Plan

- Review Sequential Patterns mining
- Train on smaller set with more ITN
- Try new dataset
- Try TN on BookText

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 25

Finding ITN instances

- For reference: semiotic_classes: ["LETTERS", "CARDINAL", "VERBATIM", "ORDINAL", "DECIMAL", "ELECTRONIC", "DIGIT", "MONEY", "FRACTION", "TIME", "ADDRESS"]
- Checked GTN dataset, and also by commonsence: most semiotic class instances must include digits except the following cases:
  - LETTERS- letter sequence. Example: MRT or M.R.T.
  - Used Roman Numerals instead. Example: XIVth, V
- Considering to the cases above, the following instances in Libriheavy is picked for ITN:
  - Instances with digits
  - Instances with roman numerals (May include cases that contains a single V, C, M)
  - Instances with words that are fully capitalized (May include cases where capitalized words are just emphasis)

Finding ITN instances

- Statistics of ITN instances (Only small and medium splits):
  - Instances with digits: 31,461 (2.75%)
  - Instances with roman numerals: 12,165 (1.06%)
  - Instances with fully capitalized words: 23,082 (2.02%)
- Statistics of ITN instances (including small, medium and large splits):
  - Instances with digits: 341,697 (3.01%)
  - Instances with roman numerals: 119,249 (1.05%)
  - Instances with fully capitalized words: 202,723 (1.78%)

Training model

- Used all ITN instances found in small, medium and large splits. Total number of instances: 663,669
- Tested on ASR samples (37 samples)
- Insights:
  - Able to convert numbers
  - Perform badly on incomplete sentence and Will map the incomplete sentence into fully capitalized version without converting numbers
    - E.g.: compassvale drive the chevrons and aljunied avenue five → COMPASSVALE DRIVE THE CHEVRONS AND ALJUNIED AVENUE FIVE.
  - Disfluency affect the result and If the number is at the beginning or end of the sentence, it will not convert part of it, might because I removed all mismatches at the beginning and end.
    - E.g.: figure on the right

```
I: ya the total number is eighty thousand three hundred and fifty
time used:  0.10821700096130371
O: YA. The total number is eighty thousand three hundred and fifty.

I: the total number is eighty thousand three hundred and fifty
time used:  0.12587523460388184
O: The total number is 80,300 and fifty.

I: the total number is eighty thousand three hundred and fifty one
time used:  0.09295654296875
O: The total number is 80,350 one.

I: the total number is eighty thousand three hundred and fifty two kids
time used:  0.1061868667602539
O: The total number is 80,352 kids.
```

Review Sequential Pattern Mining

- Does the order of words matters?
  - By intuition, the order of words determines the meaning of the phrase
  - Another algorithm: Frequent Itemset mining will ignore order information
- Intuition: a pronounced phrase or word in the Book Text should have one or multiple mappings into the ASR Transcript.
  - itself
  - one fixed mapping: e.g., 1 → one, favorite → favourite
  - Multiple mappings: e.g., 'll → will or shall
  - If the word or phrase is not mapped to fixed mappings, it is likely not pronounced.
- Group the patterns that have the same items on the Book Text side, i.e., before "|"
- For each group, order the patterns according to support

Review Sequential Pattern Mining

- For each group, the sequence with the highest support will have no items on ASR Transcript side.
- If the group only have one sequence → no fixed mapping. it means such words or phrases are not pronounced.
  - Remove such cases in the dataset
- For each group, if the ratio between the support of the second sequence and the top sequence is very high, it means the words or phrases in the BookText have one fixed mapping.
  - E.g.:

```
(52433, ['mr', '|']),
(52117, ['mr', '|', 'mister']),
```

  - Leave it in the dataset as it is legal mapping
- For the rest of the groups, it could be that the BookText have multiple mappings or because some common ASR error
  - (Maybe) Build a tree structure according to the items after "|" to count the frequency (get the percentage of such cases)

```
'be': [(1502, ['be', '|']),
 (106, ['be', '|', 'he']),
 (98, ['be', '|', 'been']),
 (76, ['be', '|', 'maybe']),
 (57, ['be', '|', 'being']),
 (49, ['be', '|', 'have']),
 (47, ['be', '|', 'before']),
 (46, ['be', '|', 'have', 'been']),
 (41, ['be', '|', 'is']),
 (35, ['be', '|', 'become']),
 (34, ['be', '|', 'between']),
 (32, ['be', '|', 'the']),
 (31, ['be', '|', 'because']),
 (30, ['be', '|', 'me']),
 (24, ['be', '|', 'b']),
 (24, ['be', '|', 'behind']),
 (22, ['be', '|', 'we']),
 (22, ['be', '|', 'believe'])],
```

Plan

- Improve sequential pattern mining (libriheavy worth? preprocess)
- Try SPGI dataset?
- Try TN on BookText

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 26

Comparison between libriheavy and GTN

- Number of ITN instances in GTN: 77,244,691
- Number of ITN instances in Libriheavy: 663,669 (0.8% compared to GTN)
- Randomly sampled 100 ITN instances in the Libriheavy:
  - Instances with numbers: 50 (mostly dates, chapter xx, and few cardinal numbers)
  - Instances with LETTER sequence (abbreviation, etc.): 7
  - Instances that don't contain ITN:
    - Because of capitalized words: 31
    - Example: Caption: "AND SO, ALL THROUGH THE LONG, WEARY SUMMER, HENRY ADAMS SAT, HEAD IN HAND, WONDERING IF DARWIN WAS RIGHT. TO HIM THE GLACIAL EPOCH SEEMED LIKE A YAWNING CHASM BETWEEN A UNIFORMITARIAN WORLD AND HIMSELF.
    - Contains numbers but is erroneous: 12
    - Examples: beloved by my fellowsand 80 it came to pass → beloved by my fellows and so it came to pass
    - neglig6es → negligees

Comparison between libriheavy and GTN

- Problems with the Libriheavy for ITN task:
  - Limited number of valid ITN instances in Libriheavy: 663,669 * 57% = ~ 380,000 (0.5% compared to GTN)
  - Semiotic classes not diverse
  - Two sources of error (errounous book text and ASR transcription error) make the dataset noisy
- The defects of models's output:
  - Will convert incomplete sentences into fully capitalized version
  - cannot convert beginning and end of the sentence
  - Poped-up words are rare and not the main defects of the model output → sequential pattern mining may not be helpful in solving the main defects.

SPGISpeech Dataset

- SPGISpeech details:
  - Audio source: real corporate oral presentations
  - Transcription source: mannually transcribed by professionals and it is fully-formatted.
- Number of ITN instances in GTN: 77,244,691
- Number of instances in SPGISpeech Dataset: 1,966,109 (around 2.5% compared to GTN)
- Number of ITN instances in SPGISpeech Dataset: 373,175 (around 0.5% compared to GTN)
  - Instances with number: 257,396
  - Instances with abbreviations: 115,779
  - Doesn't contain symbols except punctuations and % (percent) → currency not converted to symbols in the transcript, just plain word
  - Other semiotic classes are present in the dataset except "MONEY"

Discussion & Plan

- Use SPGI dataset to train the denormalization model?
- Use both datasets to train the denormalization model?
- What should be the evaluation dataset? GTN?
- Tanmay shared some feedback about denormalization model trained on GTN
- Current goal? Use Denormalization model or Tagging model?
- Plan:
  - train on both ASR generated pair and NEMO TN with audio guide generated pair
  - Use denormalization first train on GTN then finetune on SPGI
  - Evaluation: GTN and SPGI

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 27

Denormalization approaches

- Definition of text denormalization from paper [1]: include punctuation restoration, capitalization and ITN
- Did not get much attention in the academia
  - [1] first proposed this task
  - No other papers mentioned this task (checked the references of [1] and the papers citing [1])
  - The only paper that combines PR, capitalization and ITN is [2] which also takes disfluency removal into consideration
- Approaches
  - Sequence to sequence ([1] - transformers)
  - Tagging ([2] - encoder-only transformers + multiple classification head)

[1] Neural Text Denormalization for Speech Transcripts
[2] FOUR-IN-ONE: A JOINT APPROACH TO INVERSE TEXT NORMALIZATION, PUNCTUATION, CAPITALIZATION, AND DISFLUENCY FOR AUTOMATIC SPEECH RECOGNITION

Approaches for Individual Tasks

- ITN
  - WFST
  - Neural network: sequence to sequence or tagging
  - WFST + Neural Network
- PR, Capitalization, disfluency
  - Neural network: Sequence to sequence or tagging

Novelty in Denormalization/ITN

- Data preparation
  - New data preparation method
  - Utilize Large Language Model: data cleaning, data generation
- Model
  - Consider it as a new task other than sequence-to-sequence and tagging
  - New architecture
  - Utilize Large Language Model: finetune
- Language
  - Explore to new language or multilingual case

Plan

- train on both ASR generated pair and NEMO TN with audio guide generated pair
- Use denormalization model, first train on GTN then finetune on SPGI
- Evaluation: GTN and SPGI

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 28

Transcribing Audios

- Toolkit and model used: NeMo parakeet-tdt-1.1b
- Didn't use canary-1b because it is too slow
- The resultant transcript doesn't contain numbers, capitalization, and punctuations
- May automatically remove disfluency–like words, e.g., and, so

Leaderboard | Metrics | ✨ Request a model here!

model	Average WER	RTF (1e-3)	AMI	Earnings22	Gigaspeech	LS Clean
nvidia/canary-1b	6.67	9.8	14	12.25	10.19	1.49
nvidia/parakeet-tdt-1.1b	6.95	1.7	15.9	14.65	9.55	1.39
nvidia/parakeet-rnnt-1.1b	7.04	2.4	17.1	14.15	9.96	1.46
nvidia/parakeet-ctc-1.1b	7.58	0.6	15.6	13.69	10.27	1.83
nvidia/parakeet-rnnt-0.6b	7.63	2	17.56	14.9	10.07	1.62
openai/whisper-large-v3	7.7	7.45	16.01	11.3	10.02	2.03

Experiment setting

- Models:
  - Model trained on GTN
  - Model first trained on GTN then fine-tuned on SPGI (parallel corpus: raw ASR transcript and manually labelled transcript)
- Evaluation set:
  - GTN: took first 100,000 samples from the test set
  - SPGI: the whole val split

Experiment setting

- Evaluation Metric (adapted from paper [1]):
  - Sentence accuracy
  - Word error rate (WER)
  - Specialized word error rate: copy words, punctuation, digit, capitalization, symbols
  - Calculation process for specialized WER: first split the sentence by space, then pick the units that satisfy specific standard in each case, then calculate the WER. Ignored absolute position, focus the order and presence.

[1] Neural Text Denormalization for Speech Transcripts

Experiment Result - GTN testset

Pretrained

TYPE	ERRORS	TOTAL	RATE
Sentence	36788	100000	36.79%
Word	101855	1203159	8.47%
Copy Word	18927	569908	3.32%
Uppercase Word	40087	348739	11.49%
Digit	12469	83618	14.91%
Punctuation	34683	190165	18.24%
Symbol	1247	2558	48.75%

Finetuned

TYPE	ERRORS	TOTAL	RATE
Sentence	71688	100000	71.69%
Word	195373	1203159	16.24%
Copy Word	21783	569908	3.82%
Uppercase Word	68450	348739	19.63%
Digit	17470	83618	20.89%
Punctuation	85678	190165	45.05%
Symbol	1837	2558	71.81%

Experiment Result - SPGI dataset

Pretrained

TYPE	ERRORS	TOTAL	RATE
Sentence	7343	7706	95.29%
Word	36600	229370	15.96%
Copy Word	8631	181752	4.75%
Uppercase Word	10698	18911	56.57%
Digit	4042	6595	61.29%
Punctuation	10296	18806	54.75%
Symbol	171	693	24.68%

Finetuned

TYPE	ERRORS	TOTAL	RATE
Sentence	6346	7706	82.35%
Word	20885	229370	9.11%
Copy Word	5675	181752	3.12%
Uppercase Word	6147	18911	32.50%
Digit	539	6595	8.17%
Punctuation	7451	18806	39.62%
Symbol	151	693	21.79%

Experiment Result Analysis

- GTN testset
  - The performance of the pretrained model is comparable with the model in the paper
  - The performance dropped after fine-tuning on SPGI
  - The transcript generated by ASR may contain errors and influence the performance of the fine-tuned model
- SPGI testset
  - The model pretrained on GTN performs badly on SPGI
  - The performance after finetuning improved significantly especially for digits.
  - The error in ASR transcript may increases WER for both models.

Plan

- Try TN with ASR transcript guidance to get better parallel corpus for SPGI dataset
- Check out typical errors of both models
- Test on in-house ASR testset
- Try on only SPGI

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 29

Experiment Setting

- Datasets:
  - GTN: train split for training and validation during training; first 100,000 instances in test split for testing
  - SPGI: Manual transcript as written-form text; applied audio-based TN on manual transcript to get spoken-form text.
- Models:
  - Trained on GTN: M_G
  - Trained on GTN, then finetuned on SPGI: M_{GS}
  - Trained on SPGI: M_S
- Evaluation datasets:
  - GTN, SPGI, in-house ASR testset

Results on in-house ASR testset

- M_G
  - Most are correct/acceptable
  - Insert comma in between long numbers and sometimes incorrect
- M_{GS}:
  - Most are correct/acceptable
  - May convert oh/um into 0
  - Long numbers sometimes incorrect
- M_S:
  - Not good in general
  - May lose words
  - Not good at numbers

Results on GTN testset

M_{g} on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	36788	100000	36.79%
Word	101855	1203159	8.47%
Copy Word	18927	569908	3.32%
Uppercase Word	40087	348739	11.49%
Digit	12469	83618	14.91%
Punctuation	34683	190165	18.24%
Symbol	1247	2558	48.75%

M_{gs} on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	65454	100000	65.45%
Word	191978	1203159	15.96%
Copy Word	24996	569908	4.39%
Uppercase Word	82872	348739	23.76%
Digit	23598	83618	28.22%
Punctuation	61301	190165	32.24%
Symbol	1785	2558	69.78%

M_{s} on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	96322	100000	96.32%
Word	483283	1203159	40.17%
Copy Word	39469	569908	6.93%
Uppercase Word	251070	348739	71.99%
Digit	69506	83618	83.12%
Punctuation	93237	190165	49.03%
Symbol	3024	2558	118.22%

- M_G performs the best on GTN, M_{GS}'s performance dropped after fine-tuning.
- M_S cannot generalize well because GTN contains patterns not in SPGI

Typical Errors on GTN

- Common
  - Missing or unnecessary capitalization in the mid of sentences – Many named entity in GTN
  - Missing or unnecessary comma
  - Date format error
- M_{GS}
  - Missing period
- M_S
  - May convert nineteen to 2019 because the SPGI dataset contains such abbreviation. Causing the model to convert nineteen fifty two into 2019, 52
  - Not good at convert digits – not enough numbers?
  - Not good at punctuation and capitalization – many incomplete sentences in SPGI dataset.

Results on SPGI testset

M_{g} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	7152	7706	92.81%
Word	29614	228652	12.95%
Copy Word	5653	181730	3.11%
Uppercase Word	9844	18746	52.51%
Digit	3835	6417	59.76%
Punctuation	8846	18440	47.97%
Symbol	20	693	2.89%

M_{gs} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	4743	7706	61.55%
Word	11813	228652	5.17%
Copy Word	2133	181730	1.17%
Uppercase Word	3896	18746	20.78%
Digit	456	6417	7.11%
Punctuation	5217	18440	28.29%
Symbol	0	693	0.00%

M_{s} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	6722	7706	87.23%
Word	25768	228652	11.27%
Copy Word	5659	181730	3.11%
Uppercase Word	8997	18746	47.99%
Digit	1197	6417	18.65%
Punctuation	8846	18440	47.97%
Symbol	18	693	2.60%

- M_{GS} performs the best. M_S performs relatively good at digits
- M_G performs not good but is better than on M_S GTN → GTN make model relatively more generalizable.
- The particular high error rate in capitalization and punctuation might because of the SPGI contains many incomplete sentences.

Typical Errors on GTN

- M_G
  - Additional period
  - formattings errors (formats present in GTN but not SPGI)
- M_{GS}
  - Missing/additional period/comma
  - Capitalizations
- M_S
  - Not good at converting numbers
  - Missing/additional period/comma
  - Capitalizations

Discussion and Plan

- Pretrain on GTN and then finetune on specific dataset performs the best on specific dataset
- The model directly trained on SPGI might not sufficiently trained. Current: 10 epochs. Will try to tune the hyperparameters.
- Errors in Punctuation and capitalization often co-occur
- Tanmay's feedback on the model
  - space before n't – can be solved by post-processing
  - Capitalization mid sentence – didn't observe that in the fine-tuned model
  - Single word (noun) duplication – didn't observe that in the fine-tuned model
- Plan:
  - Finetune more
  - forward the model to tanmay
  - try generated data talk to bennett
  - Maybe get data from aisg

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 30

Dataset from AISG

- Only the train split
- Done standardization, upsampling, duplication, and augmentation
- Removed punctuation – cannot be used for finetuning denormalization model
- A small subset of GTN ~300 MB compared to ~18 GB of full GTN
- Maybe adopt a part of the data pipeline after they finished their work.

Hyperparameter Tuning

- Performed hyperparameter tuning when finetuning M_{GS} on SPGI
- Tuned three hyperparameters:
  - Learning rate: 1e-5 ~ 1e-3
  - Weight decay: 0 ~ 1e-2
  - Maximum epochs: 5 ~ 15
- Used optuna library
- Modified argument using sys.argv
- Validation loss cannot be returned
  - write and read external file.

```python
def objective(trial):
    lr = trial.suggest_float("lr", 1e-5, 1e-3)
    weight_decay = trial.suggest_float("weight_decay", 0, 1e-2)
    epochs = trial.suggest_int("epochs", 5, 15)
    print(sys.argv)
    args = sys.argv
    args[args.index("--lr") + 1] = str(lr)
    args[args.index("--weight-decay") + 1] = str(weight_decay)
    args[args.index("--max-epoch")+1] = str(epochs)
    sys.argv = args
    print(sys.argv)
    cli_main()
    with open("/home/mazhang/pt/txtDenormalization/denormalizer/valid_losses.txt", "r") as file:
        loss = file.readline().strip()
    return float(loss)

if __name__ == '__main__':
    sys.argv[0] = re.sub(r'(-script\.pyw|\.exe)?$', '', sys.argv[0])
    study = optuna.create_study(direction="minimize")
    study.optimize(objective, n_trials=20)
    sys.exit(cli_main())
```

Hyperparameter Tuning

- Performed 20 trials
- Validation loss: 1.944 – 1.988 (no big difference)
- Best hyperparameter:
  - Learning rate: 1e-3
  - Weight decay: 8.4e-3
  - Maximum epochs: 13
- Used this new set of hyperparameters to finetune a new model M_{GS}'

Results on GTN testset

M_{g} on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	36788	100000	36.79%
Word	101855	1203159	8.47%
Copy Word	18927	569908	3.32%
Uppercase Word	40087	348739	11.49%
Digit	12469	83618	14.91%
Punctuation	34683	190165	18.24%
Symbol	1247	2558	48.75%

M_{gs} on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	65454	100000	65.45%
Word	191978	1203159	15.96%
Copy Word	24996	569908	4.39%
Uppercase Word	82872	348739	23.76%
Digit	23598	83618	28.22%
Punctuation	61301	190165	32.24%
Symbol	1785	2558	69.78%

M_{gs}' on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	77909	100000	77.91%
Word	251089	1203159	20.87%
Copy Word	32180	569908	5.65%
Uppercase Word	103256	348739	29.61%
Digit	35194	83618	42.09%
Punctuation	74824	190165	39.35%
Symbol	1904	2558	74.43%

- M_G performs the best on GTN
- The performance of M_{GS} dropped after fine-tuning, and M_{GS}' dropped further

Results on SPGI testset

M_{g} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	7152	7706	92.81%
Word	29614	228652	12.95%
Copy Word	5653	181730	3.11%
Uppercase Word	9844	18746	52.51%
Digit	3835	6417	59.76%
Punctuation	8846	18440	47.97%
Symbol	20	693	2.89%

M_{gs} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	4743	7706	61.55%
Word	11813	228652	5.17%
Copy Word	2133	181730	1.17%
Uppercase Word	3896	18746	20.78%
Digit	456	6417	7.11%
Punctuation	5217	18440	28.29%
Symbol	0	693	0.00%

M_{gs}' on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	4560	7706	59.17%
Word	9698	228652	4.24%
Copy Word	2095	181730	1.15%
Uppercase Word	3451	18746	18.41%
Digit	84	6417	1.31%
Punctuation	4737	18440	25.69%
Symbol	0	693	0.00%

- M_{GS}' performs the better than M_{GS} in the cost of losing some knowledge in GTN

Plan

- Add samples from GTN into the validation set when doing hyperparameter tuning because:
  - Current validation set only contains samples from SPGI
  - May overfit to SPGI and loss knowledge in GTN
- Combine GTN (subset?) + SPGI + optionally synthetic data from bennett to train model instead of finetuning after trained on GTN
- Forward model to Tanmay (which version?) Mgs

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 31

Work done for last week

- Forwarded M_G and M_{GS} and denormalization model training scripts to Tanmay
- Haven't received synthetic data from bennett
- Took a subset of GTN:
  - Ten semiotic classes, sampled 100,000 samples for each classes from training split as training set → 1,444,793 sentences → ~100MB → ~ 2 times the size of SPGI training set
  - Took first 40,000 samples from validation split as validation set. (validation set of SPGI ~35k samples)
- Combined GTN subset with SPGI (for both training set and validation set)
- Trained a model $M_{combine}$ on the combined dataset using the hyperparameters tuned before.

Results on GTN testset

M_{g} on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	36788	100000	36.79%
Word	101855	1203159	8.47%
Copy Word	18927	569908	3.32%
Uppercase Word	40087	348739	11.49%
Digit	12469	83618	14.91%
Punctuation	34683	190165	18.24%
Symbol	1247	2558	48.75%

M_{gs} on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	65454	100000	65.45%
Word	191978	1203159	15.96%
Copy Word	24996	569908	4.39%
Uppercase Word	82872	348739	23.76%
Digit	23598	83618	28.22%
Punctuation	61301	190165	32.24%
Symbol	1785	2558	69.78%

M_combine on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	47170	100000	47.17%
Word	148521	1203159	12.43%
Copy Word	32689	569908	5.76%
Uppercase Word	66235	348739	19.00%
Digit	17558	83618	21.00%
Punctuation	50234	190165	26.45%
Symbol	1676	2558	66.35%

- M_G performs the best on GTN
- $M_{combine}$ performs better than M_{GS} on GTN but not as good as M_G because only a subset of GTN is used.

Results on SPGI testset

M_{g} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	7152	7706	92.81%
Word	29614	228652	12.95%
Copy Word	5653	181730	3.11%
Uppercase Word	9844	18746	52.51%
Digit	3835	6417	59.76%
Punctuation	8846	18440	47.97%
Symbol	20	693	2.89%

M_{gs} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	4743	7706	61.55%
Word	11813	228652	5.17%
Copy Word	2133	181730	1.17%
Uppercase Word	3896	18746	20.78%
Digit	456	6417	7.11%
Punctuation	5217	18440	28.29%
Symbol	0	693	0.00%

M_{combine} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	4987	7706	64.72%
Word	11292	228652	5.01%
Copy Word	2407	181730	1.34%
Uppercase Word	4122	18746	21.99%
Digit	154	6417	2.40%
Punctuation	5279	18440	28.63%
Symbol	2	693	0.29%

- $M_{combine}$ performs good at SPGI and comparable with M_{GS} but not as good as M_{GS}‘

M_{gs}' on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	4560	7706	59.17%
Word	9698	228652	4.24%
Copy Word	2095	181730	1.15%
Uppercase Word	3451	18746	18.41%
Digit	84	6417	1.31%
Punctuation	4737	18440	25.69%
Symbol	0	693	0.00%

Results on in-house ASR testset

- Still not good at long digits and currency units
- Lose content for very long sentence (check previous models)
- May convert “oh” to 0 (caused by SPGI)

Plan

- Perform hyperparameter tuning on the new combined training set (in progress)
- Try adjusting the proportion of GTN? (sample more digit and money)
- Use synthetic data to imporve it’s performance on long digits
- clean up code, reproducible, documentation.
- slides to explain how to run code

Inverse Text Normalization

ZHANG MENGAO (Year 4 undergraduate, part-time)

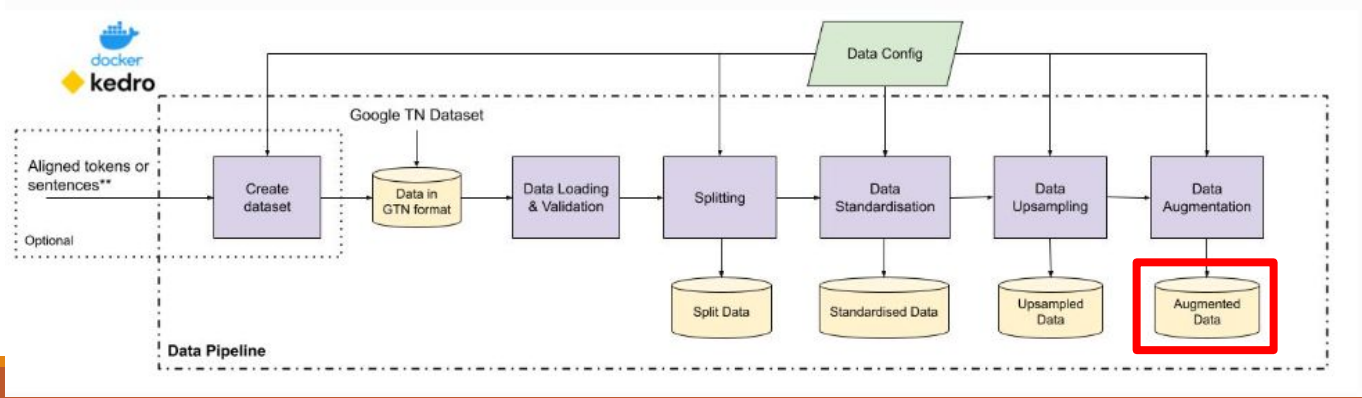
WEEK 32 & 33

Recap

- $\mathbf{M_G}$: Model trained on the full GTN dataset. (forwarded to Tanmay)
- $\mathbf{M_S}$: Model trained on SPGI dataset.
- $\mathbf{M_{GS}}$: Model first trained on GTN and finetuned on SPGI. (forwarded to Tanmay)
- $\mathbf{M_{GS}'}$: Same as Mgs except that hyperparameter tuning is conducted when doing finetuning.
- $\mathbf{M_{combine}}$: Model trained on a combined dataset – SPGI and a subset of GTN
- Performance Ranking
  - GTN testset: $\mathbf{M_G > M_{combine} > M_{GS} > M_{GS}' > M_S}$
  - SPGI testset: $\mathbf{M_{GS}' > M_{combine} \approx M_{GS} > M_G > M_S}$
- Shortcomes of all models: Cannot convert long digits well.

Data from AISG

- Took augmented data.
- In order to keep punctuation, modified data standardisation module. (check punctuation)
- training set: 1,191,681 instances (previously sampled data from GTN 1,444,793 instances)
- Took first 40,000 samples from validation split as validation set. (validation set of SPGI ~35k samples)
- Testset samples each semiotic classes at least 1,000 instances



Training New Model

- I was tuning hyperparameters for $M_{combine}$.
  - Unfortunately, the validation losses never goes less than the initial one.
  - Thus, I stick to the initial hyperparameters found when doing hyperparameter tuning for M_{gs}'.
- Combined GTN data from AISG with SPGI (for both training set and validation set)
- Trained a new model M_{AS} on the combined data. (A for AISG, S for SPGI)

Results of M_{AS}' on In-house ASR testset

- May lose digits for long numbers
- May lose words for long sentence (also observed in $M_{combine}$)
  - Not observed for M_G and M_{GS} and M_{GS}'
  - This could because the training data size is not big enough

Results of M_{AS}' on GTN

M_{g} on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	36788	100000	36.79%
Word	101855	1203159	8.47%
Copy Word	18927	569908	3.32%
Uppercase Word	40087	348739	11.49%
Digit	12469	83618	14.91%
Punctuation	34683	190165	18.24%
Symbol	1247	2558	48.75%

M_{gs} on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	65454	100000	65.45%
Word	191978	1203159	15.96%
Copy Word	24996	569908	4.39%
Uppercase Word	82872	348739	23.76%
Digit	23598	83618	28.22%
Punctuation	61301	190165	32.24%
Symbol	1785	2558	69.78%

M_{gs}' on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	77909	100000	77.91%
Word	251089	1203159	20.87%
Copy Word	32180	569908	5.65%
Uppercase Word	103256	348739	29.61%
Digit	35194	83618	42.09%
Punctuation	74824	190165	39.35%
Symbol	1904	2558	74.43%

M_combine on GTN

TYPE	ERRORS	TOTAL	RAT
Sentence	47170	100000	47.17
Word	148521	1203159	12.43
Copy Word	32689	569908	5.76
Uppercase Word	66235	348739	19.00
Digit	17558	83618	21.00
Punctuation	50234	190165	26.45
Symbol	1676	2558	66.35

M_{AS} on GTN

TYPE	ERRORS	TOTAL	RATE
Sentence	66830	100000	66.83%
Word	222743	1194686	18.64%
Copy Word	19496	567635	3.43%
Uppercase Word	106357	348551	30.51%
Digit	18431	83618	22.04%
Punctuation	79810	189951	42.02%
Symbol	2419	2526	95.76%

- $M_G > M_{combine} > M_{GS} > M_{GS}'$
- M_{AS} performs comparably with M_g on copy word but not very good at other cases.
- Might because the data from aisg is not sampled according to ITN, instead, it contains a lot of instances that are purely copy words.
- Another possibility is that the output format is not matched with testset

Results of M_{AS}' on SPGI

M_{g} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	7152	7706	92.81%
Word	29614	228652	12.95%
Copy Word	5653	181730	3.11%
Uppercase Word	9844	18746	52.51%
Digit	3835	6417	59.76%
Punctuation	8846	18440	47.97%
Symbol	20	693	2.89%

M_{gs} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	4743	7706	61.55%
Word	11813	228652	5.17%
Copy Word	2133	181730	1.17%
Uppercase Word	3896	18746	20.78%
Digit	456	6417	7.11%
Punctuation	5217	18440	28.29%
Symbol	0	693	0.00%

M_{gs}' on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	4560	7706	59.17%
Word	9698	228652	4.24%
Copy Word	2095	181730	1.15%
Uppercase Word	3451	18746	18.41%
Digit	84	6417	1.31%
Punctuation	4737	18440	25.69%
Symbol	0	693	0.00%

M_{combine} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	4987	7706	64.72%
Word	11292	228652	5.01%
Copy Word	2407	181730	1.34%
Uppercase Word	4122	18746	21.99%
Digit	154	6417	2.40%
Punctuation	5279	18440	28.63%
Symbol	2	693	0.29%

M_{AS} on SPGI

TYPE	ERRORS	TOTAL	RATE
Sentence	6573	7706	85.30%
Word	23416	225502	10.38%
Copy Word	3437	179488	1.91%
Uppercase Word	7466	18744	39.83%
Digit	668	6417	10.41%
Punctuation	11410	18440	61.88%
Symbol	21	693	3.03%

- $M_{GS}' > M_{combine} \approx M_{GS} > M_G$
- M_{AS} performs better than M_g but not good compared to other models
- M_{AS} ouptut usually missing comma
- M_{AS} sometimes loses digits

Results of M_{AS}' on GTN testset from AISG

M_g on GTN_aisg

TYPE	ERRORS	TOTAL	RATE
Sentence	7085	9151	77.42%
Word	38216	134267	28.46%
Copy Word	2934	59991	4.89%
Uppercase Word	8585	39524	21.72%
Digit	11494	16402	70.08%
Punctuation	8953	17399	51.46%
Symbol	384	851	45.12%

M_gs on GTN_aisg

TYPE	ERRORS	TOTAL	RATE
Sentence	8261	9151	90.27%
Word	42605	135101	31.54%
Copy Word	3862	59991	6.44%
Uppercase Word	13327	39524	33.72%
Digit	9159	16402	55.84%
Punctuation	9690	17399	55.69%
Symbol	374	851	43.95%

M_GS' on GTN_aisg

TYPE	ERRORS	TOTAL	RATE
Sentence	8465	9151	92.50%
Word	45246	135101	33.49%
Copy Word	4473	59991	7.46%
Uppercase Word	14748	39524	37.31%
Digit	9178	16402	55.96%
Punctuation	10298	17399	59.19%
Symbol	426	851	50.06%

M_AS on GTN_aisg

TYPE	ERRORS	TOTAL	RATE
Sentence	7404	9151	80.91%
Word	29790	135101	22.05%
Copy Word	2972	59991	4.95%
Uppercase Word	15164	39524	38.37%
Digit	4881	16402	29.76%
Punctuation	7096	17399	40.78%
Symbol	282	851	33.14%

- The checkpoints of $M_{combine}$ got accidentally deleted. The result is not ready yet.
- M_{AS}' performs better compared to other model, except capitalization.
- Possible reason: the format is standardized in the new testset.

Discussion & Plan

- clean up code, experiment records, reproducible, documentation.
- slides to explain how to run code
- Handover session
- Upload to google drive (big file)
- Summarize and find directions

Inverse Text Normalization - Summary

ZHANG MENGAO (Year 4 undergraduate, part-time)

WEEK 34

General things to note in the ITN

- ITN only applies to the output of non end-to-end ASR systems
  - Some end-to-end ASR systems trys to directly convert audio into readable (include ITN, PR, Capitalization, Disfluency removal) transcript
  - May need to compare the performance difference between such end-to-end ASR system and standalone modules for post-processing. (to see whether current end-to-end ASR systems performs already well)
- In the scenario where ITN needs to be applied, we typically also need PR, Capitalization and Disfluency Removal
  - It is good to combine them together
- ITN task faces the problem of lacking data
  - Currently, there are no large scale human-labeled data available.
  - Other tasks may have utilized crowd sourcing to obtain data e.g., machine translation. ITN is not the case

General things to note in the ITN

- ITN evaluation is tricky
  - Usually, the ITN instances can be into multiple formats. → hard to tell whether the model's output is correct
  - Hard to find high-quality evaluation set as no large scale human-labeled data is available
  - The ground-truth/label of ITN instances may change with respect to scenarios, languages, culture, etc.
- If we want to customize ITN formats, we need:
  - Either customize WFST grammar
  - Or collect data with the expected formats (Use neural networks)
  - Or standardize the existing data (Use neural networks)
  - Or design the prompt carefully (Use LLM)

Current approaches for ITN

- WFST – rule-based approach
  - Pros: complete control over the generated output, easy to debug and make correction
  - Cons: require much labor and linguistic knowledge, hard to scale up and generalize, usually don't consider the context information. (can use language model to improve context awareness)
- Neural Networks
  - Pros: don't need to define every rule, data-driven; could generalize better than WFST given enough data;
  - Cons: May make unrecoverable errors and requires lots of data. Black box, hard to debug and make correction
- Mixed
  - Could be in multiple approaches, e.g., use neural networks to detect the ITN span, then use WFST to do the conversion.
- LLM
  - Given that the LLM has been trained on a huge amount of data, the model may have learned how to do the conversion under 0-shot or few-shot settings (No paper has tried this before)

Current approaches for ITN – Neural Networks

- As reported by the paper [1], neural networks have better result compared to WFST
- Duplex model [1]
  - performs both TN and ITN
- Thutmose Tagger [2]
  - performs ITN, better compared to duplex
- Denormalization [3]
  - performs ITN, PR and Capitalization
- Four-in-one [4]
  - performs ITN, PR, Capitalization, and Disfluency removal

[1] A UNIFIED TRANSFORMER-BASED FRAMEWORK FOR DUPLEX TEXT NORMALIZATION
[2] Thutmose Tagger: Single-pass neural model for Inverse Text Normalization
[3] Neural Text Denormalization for Speech Transcripts
[4] FOUR-IN-ONE: A JOINT APPROACH TO INVERSE TEXT NORMALIZATION, PUNCTUATION, CAPITALIZATION, AND DISFLUENCY FOR AUTOMATIC SPEECH RECOGNITION

Method tried

- Thutmose Tagger [1]
  - pre-trained model – on GTN dataset
- Denormalization [2]
  - Trained on GTN
  - have comparable performance on ITN instances compared to Thutmose Tagger
  - Advantage: could also perform PR, capitalization.
  - Disadvantage: not good at long digits. Need high-quality and large amount of data to perform well because it is trained from scratch.

[1] Thutmose Tagger: Single-pass neural model for Inverse Text Normalization
[2] Neural Text Denormalization for Speech Transcripts

Method tried – Denormalization

- Trained libriheavy
  - Don’t have good performance
  - Libriheavy contains too much noise and too few ITN instances
- Trained on SPGI
  - First train on GTN, then fine-tune on SPGI gives the best result on SPGI
  - Feasible way:
    - train denormalization model on GTN (transformer model needs large amount of data)
    - fine-tuning it on the data with desired format (change the style of the output or fix common defects occurs to the pretrained model)

Possible direction

- How to properly evaluate the model's performance
  - No widely unified evaluation method in the literature
- How to get data except GTN
- How to possibly integrate LLM into the task
  - As LLM as been proven very powerful for many NLP tasks
  - Could be used to prepare data or make prediction

Plan

- Summarize the models and results
- Working on cleaning code
- Handover session – next Thursday