Research Presentation at UKP

Mitodru Niyogi,
M.Sc., B.Tech.

Interdisciplinary Center for Scientific Computing,
Heidelberg University, Heidelberg, Germany
`niyogi@cl.uni-heidelberg.de`

November 21, 2022

Outline

About Myself

- Born and raised in Calcutta, India.
- Bachelor of Technology in Information Technology in Calcutta, India (2017).
- Worked at IIT Kanpur, India as Research Engineer from 2017 - 2018.
- MSc. in Scientific Computing, Heidelberg University, Germany (December 2021).
- Worked at SAP SE in Walldorf/Berlin, Germany as a Developer and Data Scientist.
- Working as Machine Learning Engineer at GFK SE, Nuremberg.
- Past affiliations:

SAP 2019-2021

2018-2021

dkfz. GERMAN CANCER RESEARCH CENTER IN THE HELMHOLTZ ASSOCIATION Fall 2019

INDIAN INSTITUTE OF TECHNOLOGY KANPUR 2017-2018

INDIAN INSTITUTE OF MANAGEMENT CALCUTTA 2016-2017

Research Interests

- NLP (text generation, language models, NLU, multilingual, low resource languages, summarization, information extraction, knowledge graphs)
- ML4Code
- Information Retrieval (cross-lingual, neural IR, multimodal IR, legalIR)
- Deep Learning & AI4SocialGood

ML4Code Objective?

- Want to fill the skill gap between industry & market supply by creating AI models that can mimic as amateur software developer.
- Want to create AI model that will generate REST API services and/or web/mobile applications.

Learning Multilingual Embeddings for Cross-Lingual Information Retrieval in the Presence of Topically Aligned Corpora [Niyogi et al. 2018]

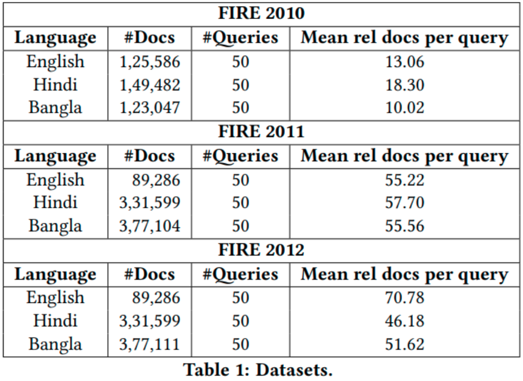
Problem Defintion

- Cross-lingual information retrieval is a challenging task in the absence of aligned parallel corpora.
- Most of the previous work on cross-lingual IR [1, 2] require sentence-aligned parallel data and other language specific resources such as dictionaries.
- Vulic et al. [3] removed this extremely constraining requirement and learnt bilingual word embedding using only document-aligned comparable corpora.

Our Approach

- We addressed the general ad-hoc information retrieval task where the query is in any of the n languages, and retrieval can be from any of the remaining languages
- We presented a multi-lingual setup where we build a cross-lingual IR system that requires no such aligned corpora or language specific resources.
- Unlike Vulic et al.[3], we proposed to build a multi-lingual embedding on the same setup.
- No separate building embeddings for collection pairs in a cross-lingual retrieval paradigm. Instead, this single multi-lingual embedding will leverage automatic cross-lingual retrieval between any two pairs of languages.

Dataset



FIRE 2010			
Language	**#Docs**	**#Queries**	**Mean rel docs per query**
English	1,25,586	50	13.06
Hindi	1,49,482	50	18.30
Bangla	1,23,047	50	10.02
FIRE 2011			
Language	**#Docs**	**#Queries**	**Mean rel docs per query**
English	89,286	50	55.22
Hindi	3,31,599	50	57.70
Bangla	3,77,104	50	55.56
FIRE 2012			
Language	**#Docs**	**#Queries**	**Mean rel docs per query**
English	89,286	50	70.78
Hindi	3,31,599	50	46.18
Bangla	3,77,111	50	51.62

Table 1: Datasets.

Figure: Datasets

Proposed Method

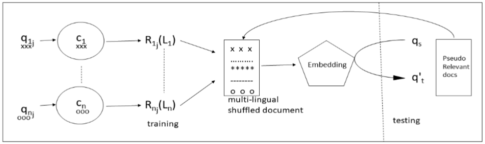
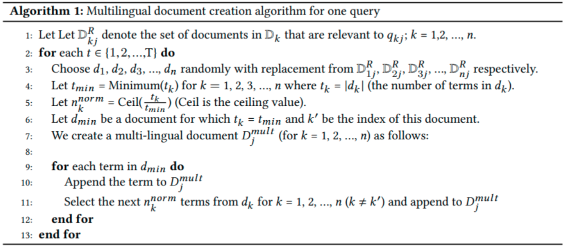


Figure: Proposed System Architecture



Algorithm 1: Multilingual document creation algorithm for one query

```
1:  Let Let $\mathbb{D}^R_{kj}$ denote the set of documents in $\mathbb{D}_k$ that are relevant to $q_{kj}$; $k$ = 1,2, ..., $n$.
2:  for each $t \in \{1, 2, ..., T\}$ do
3:      Choose $d_1, d_2, d_3, ..., d_n$ randomly with replacement from $\mathbb{D}^R_{1j}, \mathbb{D}^R_{2j}, \mathbb{D}^R_{3j}, ..., \mathbb{D}^R_{nj}$ respectively.
4:      Let $t_{min}$ = Minimum($t_k$) for $k = 1, 2, 3, ..., n$ where $t_k = |d_k|$ (the number of terms in $d_k$).
5:      Let $n^{norm}_k$ = Ceil($\frac{t_k}{t_{min}}$) (Ceil is the ceiling value).
6:      Let $d_{min}$ be a document for which $t_k = t_{min}$ and $k'$ be the index of this document.
7:      We create a multi-lingual document $D^{mult}_j$ (for $k$ = 1, 2, ..., $n$) as follows:
8:
9:      for each term in $d_{min}$ do
10:         Append the term to $D^{mult}_j$
11:         Select the next $n^{norm}_k$ terms from $d_k$ for $k$ = 1, 2, ..., $n$ ($k \neq k'$) and append to $D^{mult}_j$
12:     end for
13: end for
```

Cross-lingual query generation

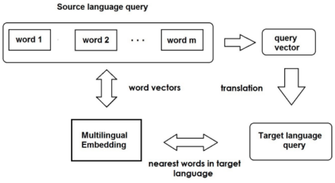


Figure 2: Embedding based cross-lingual query generation.

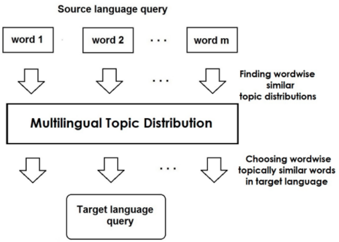
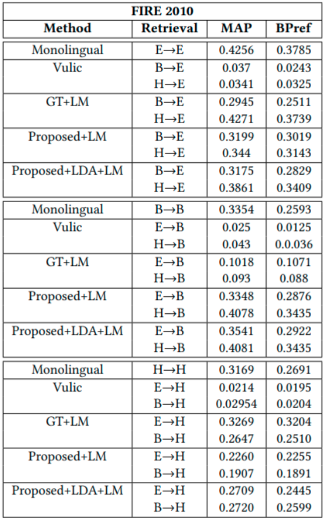
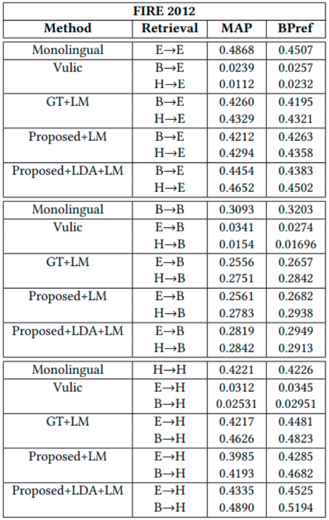

Source language query
word 1
word 2
· · ·
word m
Finding wordwise similar topic distributions
Multilingual Topic Distribution
Choosing wordwise topically similar words in target language
Target language query

Figure: Embedding & Topic modeling based cross-lingual query generation

Retrieval Results

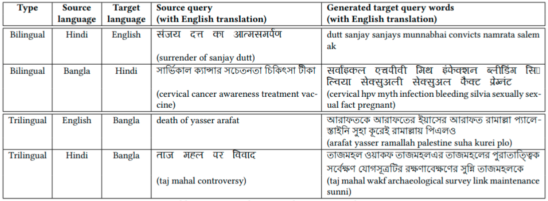


FIRE 2010			
Method	**Retrieval**	**MAP**	**BPref**
Monolingual	E→E	0.4256	0.3785
Vulic	B→E	0.037	0.0243
	H→E	0.0341	0.0325
GT+LM	B→E	0.2945	0.2511
	H→E	0.4271	0.3739
Proposed+LM	B→E	0.3199	0.3019
	H→E	0.344	0.3143
Proposed+LDA+LM	B→E	0.3175	0.2829
	H→E	0.3861	0.3409
Monolingual	B→B	0.3354	0.2593
Vulic	E→B	0.025	0.0125
	H→B	0.043	0.0.036
GT+LM	E→B	0.1018	0.1071
	H→B	0.093	0.088
Proposed+LM	E→B	0.3348	0.2876
	H→B	0.4078	0.3435
Proposed+LDA+LM	E→B	0.3541	0.2922
	H→B	0.4081	0.3435
Monolingual	H→H	0.3169	0.2691
Vulic	E→H	0.0214	0.0195
	B→H	0.02954	0.0204
GT+LM	E→H	0.3269	0.3204
	B→H	0.2647	0.2510
Proposed+LM	E→H	0.2260	0.2255
	B→H	0.1907	0.1891
Proposed+LDA+LM	E→H	0.2709	0.2445
	B→H	0.2720	0.2599



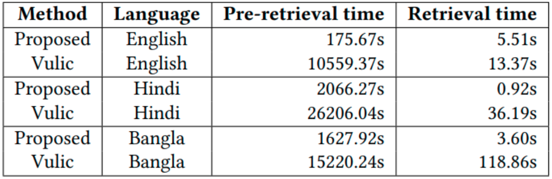
FIRE 2012			
Method	**Retrieval**	**MAP**	**BPref**
Monolingual	E→E	0.4868	0.4507
Vulic	B→E	0.0239	0.0257
	H→E	0.0112	0.0232
GT+LM	B→E	0.4260	0.4195
	H→E	0.4329	0.4321
Proposed+LM	B→E	0.4212	0.4263
	H→E	0.4294	0.4358
Proposed+LDA+LM	B→E	0.4454	0.4383
	H→E	0.4652	0.4502
Monolingual	B→B	0.3093	0.3203
Vulic	E→B	0.0341	0.0274
	H→B	0.0154	0.01696
GT+LM	E→B	0.2556	0.2657
	H→B	0.2751	0.2842
Proposed+LM	E→B	0.2561	0.2682
	H→B	0.2783	0.2938
Proposed+LDA+LM	E→B	0.2819	0.2949
	H→B	0.2842	0.2913
Monolingual	H→H	0.4221	0.4226
Vulic	E→H	0.0312	0.0345
	B→H	0.02531	0.02951
GT+LM	E→H	0.4217	0.4481
	B→H	0.4626	0.4823
Proposed+LM	E→H	0.3985	0.4285
	B→H	0.4193	0.4682
Proposed+LDA+LM	E→H	0.4335	0.4525
	B→H	0.4890	0.5194

Results



Type	Source language	Target language	Source query (with English translation)	Generated target query words (with English translation)
Bilingual	Hindi	English	संजय दत्त का आत्मसमर्पण (surrender of sanjay dutt)	dutt sanjay sanjays munnabhai convicts namrata salem ak
Bilingual	Bangla	Hindi	সার্ভিকাল ক্যান্সার সচেতনতা চিকিৎসা টীকা (cervical cancer awareness treatment vaccine)	सर्वाइकल एचपीवी मिथ इंफेक्शन ब्लीडिंग सि लिविया सेक्सुअली सेक्सुअल फैक्ट प्रेग्नंट (cervical hpv myth infection bleeding silvia sexually sexual fact pregnant)
Trilingual	English	Bangla	death of yasser arafat	আরাফতকে আরাফতের ইয়াসের আরাফত রামাল্লা প্যালেস্তাইনি সুহা কুরেই রামাল্লায় পিএলও (arafat yasser ramallah palestine suha kurei plo)
Trilingual	Hindi	Bangla	ताज महल पर विवाद (taj mahal controversy)	তাজমহল ওয়াকফ তাজমহলএর তাজমহলের পুরাতাত্ত্বিক সর্বেক্ষণ যোগসূত্রটির রক্ষণাবেক্ষণের সুন্নি তাজমহলকে (taj mahal wakf archaeological survey link maintenance sunni)

Figure: Target and generated query examples



Method	Language	Pre-retrieval time	Retrieval time
Proposed	English	175.67s	5.51s
Vulic	English	10559.37s	13.37s
Proposed	Hindi	2066.27s	0.92s
Vulic	Hindi	26206.04s	36.19s
Proposed	Bangla	1627.92s	3.60s
Vulic	Bangla	15220.24s	118.86s

Table 2: Time requirements, averaged over three datasets.

Introduction

Can AI write code? **YES**!

- Natural language (NL) to code suggestion systems assist developers in programming IDEs by translating NL utterances into compilable code snippet.
- These systems reduce the need for developers to search online sources or prevalent documentation for helpful code snippets.

Current Limitations:

- The current approaches are unable to extract semantic information from the coding intents of the developer.
- In earlier NL to code systems, researchers focused mainly on the task of semantic parsing.
- These systems made heavy use of hand-crafted rules and could only work on a limited examples with a restricted NL syntax.
- Hence they are in-extensible and expensive for real-world deployment.

Problem Definition

The problem can be further thought of developing an AI system that

- translates NL into code on the go, such as by assisting the developer by generating source code given NL intent.
- The model should understand the context of the intent and the source code of the program.
- Extending its usability for an assistive code completion feature to predict the next code tokens given the previous tokens.

Goals

Natural language to code translation

- Aims to develop a versatile Seq2Seq architecture for both objectives of translating text to code (NL2Code) and of generating comments, docstring, method documentation from source code input (Code2NL).
- Aims to use various probabilistic subword tokenizers models to incorporate the contextual embeddings of the input.
- Aims at performing an ablation study to gauge the importance of the crucial components of the developed AI system.
- Aims to develop transfer learning and data augmentation techniques to generate more diverse and accurate source code translations.

Code completion

- Aims to develop a novel RoBERTa based neural language model for source code.
- Aims at investing the performance of the model for the fill-in-mask task and compare the predicted masked tokens with the ground truth.

Challenges

- **Lack of big (NL,code) pair corpora**
  - The absence of a proper large (NL, code) pair annotated dataset limited us in exploring the full capacity of our proposed developed deep learning model.
- **GPU memory limitations**
- **Syntax decoding and lack of diverse output**
  - Generative models often suffer from the lack of diverse and repetitive text generation.
- **Evaluation Metric**
  - Both BLEU and ROUGE metrics neglect the important syntactic and semantic features of codes.
  - Perfect accuracy is too strict to consider the different correct outputs with the same semantic logic.

Background: BERT

- Attention-based bidirectional language model.
- Single encoder-style transformer block consisting of a multi-headed attention block followed by a small fully-connected network.

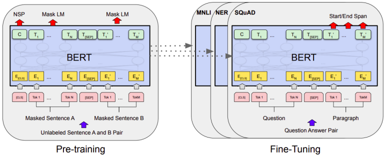


Figure: Overall pre-training and fine-tuning procedures for BERT. Figure taken from [6]

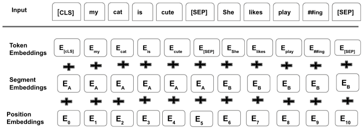


Figure: Encoding representation in BERT. Figure drawn from [6].

RoBERTa

The BERT architecture was modified to develop RoBERTa as follows:

- training the BERT model longer over more data, with larger batches
- removing the next sentence prediction (NSP) objective from BERT
- training on longer sequences with dynamic masking

BART: BERT Encoder + GPT decoder + Noisy Transformations

- BART is a denoising autoencoder built with a sequence-to-sequence model.
- BART has Transformer based bidirectional Encoder and an autoregressive decoder that is applicable to a very wide range of end tasks.
- BART is trained by corrupting documents and then optimizing a reconstruction loss—the cross-entropy between the decoder's output and the original document.
- In total, BART model has roughly 10% more parameters than BERT.

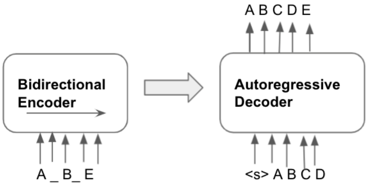


Figure: A schematic representation of BART. Figure drawn from [12].

BART: Pre-training objectives

Pretraining has two stages:

- text is corrupted with an arbitrary noising function.
- A sequence-to-sequence model is learned to reconstruct the original text.

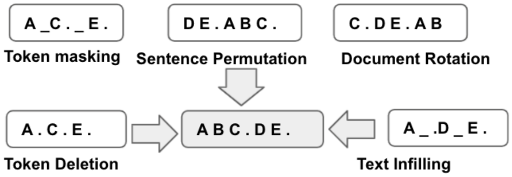


Figure: Transformations as part of Pre-training objectives for noising the input in BART. Figure drawn from [12].

Evaluation Metrics

BLEU

- BLEU (bilingual evaluation understudy) [15] measures the translation closeness by counting matches of n-grams in candidate and reference translation.
- BLEU metric does not take into account the intelligibility or grammatical correctness of a translated text.
- The BLEU is defined by

$$BP = \begin{cases} 1 & \text{if } c > r \\ \exp(1 - \frac{r}{c}) & \text{if } c \leq r \end{cases} \tag{1}$$

$$BLEU = BP \cdot \exp(\sum_{n=1}^{N} \frac{1}{N} \log p_n) \tag{2}$$

- **Sentence-BLEU**: computes the BLEU metric on a single sentence pair. It calculates the averaging of the macro-average precision.

ROUGE

- **ROUGE-N** [13] : measures unigram, bigram, trigram, and higher-order n-gram overlap.
- **ROUGE-L**: measures the longest matching sequence of words using Longest Common Subsequence (LCS) algorithm.

System Design

Dataset

Intent	How can I convert a tensor into a numpy array in TensorFlow?
Code	print(type(tf.Session().run(tf.constant([1, 2, 3]))))
Rewritten Intent	Convert a tensor with list of constants ‘[1, 2, 3]‘ into a numpy array in tensorflow

Table: CoNaLa Dataset Sample

Dataset	Number of samples
Train	1903
Validation	476
Test	500
Mined 100k train set	96179
Mined 100k valid set	10687
Mined 30k train set	31741
Mined 30k valid set	7935

Table: CoNaLa Dataset

Dataset Statistics

Average length of nl intent	46.53
Max length of nl intent	122.00
Median length of nl intent	45.00
Mode length of nl intent	46.00
Average length of code snippet	39.77
Max length of code snippet	232.00
Median length of code snippet	38.00
Mode length of code snippet	33.00

Table: Length statistics of CoNaLa data attributes..

Augmented Dataset Creation

- To make the training data larger, we used the idea to generate back translation by reversing the training objective, i.e., Code2NL, generating natural language intent from the code snippets.
- This made us augment both the training and the validation set by 1x, 2x, or even kx depending on the top-k predictions retrieved from the Code2NL model.

Proposed System Architecture

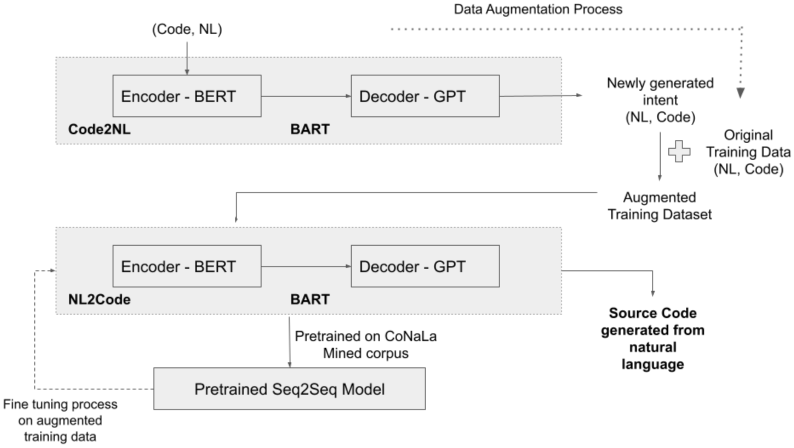

Data Augmentation Process
(Code, NL)
Encoder - BERT
Decoder - GPT
Code2NL
BART
Newly generated intent (NL, Code)
Original Training Data (NL, Code)
Augmented Training Dataset
Encoder - BERT
Decoder - GPT
NL2Code
BART
Source Code generated from natural language
Pretrained on CoNaLa Mined corpus
Pretrained Seq2Seq Model
Fine tuning process on augmented training data

Figure: Proposed System Architecture.

Experimental Results and Discussion

RQ1. Results & Analysis: How well does the developed architectures perform on the NL2Code objective in comparison to the state-of-the-art?

Models trained on Dataset/Augmented Datasets	**Test BLEU**
Seq2Seq-BART on 3x size of CoNaLa dataset	25.7710
Seq2Seq-BART on 5x size of CoNaLa dataset	25.1601
Seq2Seq-BART on CoNaLa dataset	24.2990
Fine-tuned Seq2Seq-BART on CoNaLa,, pretrained on mined100k corpus	26.5379
Fine-tuned Seq2Seq-BART on 3x CoNaLa, pretrained on mined100k corpus	**27.8235**
Fine-tuned Seq2Seq-BART on 5x CoNaLa, pretrained on mined100k corpus	25.3153
Vanilla Seq2Seq on CoNaLa	13.3270
Transformer-CoNaLa code tokenizer on CoNaLa	15.3834
Transformer-BPE on CoNaLa	19.3402
Transformer-Unigram on CoNaLa	20.9678
Transformer-WordPiece on CoNaLa	17.3237
Seq2Seq-RoBERTa without pretraining	17.0032
Fine-tuned Seq2Seq-RoBERTa on CoNaLa, pretrained on mined30k corpus	18.8853
TranX on CoNaLa	25.1050

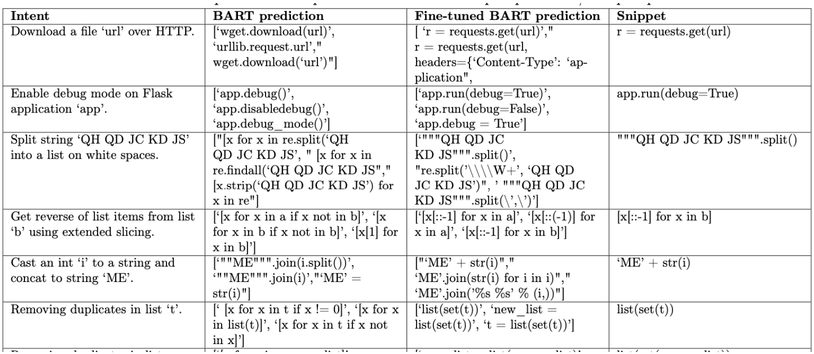
Table: Overall Comparison of all models on Test set.

Qualitative Evaluation

Model	Exact match (Sentence-BLEU >0.9)	Mostly Correct (Sentence-BLEU >= 0.6 and <= 0.9)	Marginally Correct (Sentence-BLEU >=0.4 and <=0.6)	Semantically Equivalent (Sentence-BLEU >=0.2 and <=0.4)	Number of Parsable Snippets
Fine-tuned Seq2Seq-BART	**23**	**66**	101	142	**425**
Seq2Seq-BART w/o pretraining	16	54	**110**	141	340
Fine-tuned Seq2Seq-RoBERTa	5	35	89	148	326
Seq2Seq-RoBERTa w/o pretraining	3	16	68	170	277
Transformer-CoNaLa tokenizer	2	22	80	179	138
Vanilla Seq2Seq	1	15	75	191	73
TranX	N/A	N/A	N/A	N/A	206

Table: Comparison on classifying the translations into categories.

Generated Translations: Seq2Seq-BART

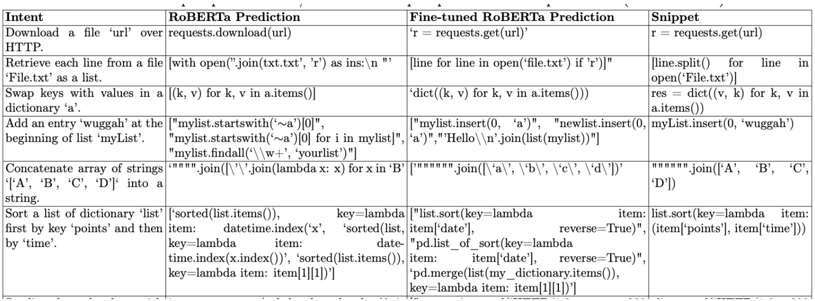


Intent	BART prediction	Fine-tuned BART prediction	Snippet
Download a file ‘url’ over HTTP.	[‘wget.download(url)’, ‘urllib.request.url’," wget.download(‘url’)"]	[‘r = requests.get(url)’," r = requests.get(url, headers={‘Content-Type’: ‘ap-plication",	r = requests.get(url)
Enable debug mode on Flask application ‘app’.	[‘app.debug()’, ‘app.disabledebug()’, ‘app.debug_mode()’]	[‘app.run(debug=True)’, ‘app.run(debug=False)’, ‘app.debug = True’]	app.run(debug=True)
Split string ‘QH QD JC KD JS’ into a list on white spaces.	["[x for x in re.split(‘QH QD JC KD JS’, " [x for x in re.findall(‘QH QD JC KD JS"," [x.strip(‘QH QD JC KD JS’) for x in re"]	[‘"""QH QD JC KD JS""".split()’, "re.split(’\\\\W+’, ‘QH QD JC KD JS’)", ’ """QH QD JC KD JS""".split(\’,\’)’]	"""QH QD JC KD JS""".split()
Get reverse of list items from list ‘b’ using extended slicing.	[‘[x for x in a if x not in b]’, ‘[x for x in b if x not in b]’, ‘[x[1] for x in b]’]	[‘[x[::-1] for x in a]’, ‘[x[::(-1)] for x in a]’, ‘[x[::-1] for x in b]’]	[x[::-1] for x in b]
Cast an int ‘i’ to a string and concat to string ‘ME’.	[‘"""ME""".join(i.split())’, ‘"""ME""".join(i)’,"‘ME’ = str(i)"]	["‘ME’ + str(i)"," ‘ME’.join(str(i) for i in i)"," ‘ME’.join(’%s %s’ % (i,))"]	‘ME’ + str(i)
Removing duplicates in list ‘t’.	[‘ [x for x in t if x != 0]’, ‘[x for x in list(t)]’, ‘[x for x in t if x not in x]’]	[‘list(set(t))’, ‘new_list = list(set(t))’, ‘t = list(set(t))’]	list(set(t))

Figure: Comparison of test predictions: fine-tuned Seq2Seq-BART v/s Seq2Seq-BART.

- The fine-tuned model worked well for intents like complicated regex, dictionary, and Pandas dataframe operations; advanced vector & list slicing operations.
- The fine-tuned model could even infer the change in the variable name or the arguments to a data structure/method expressed in the intent.

Generated Translations: Seq2Seq-RoBERTa

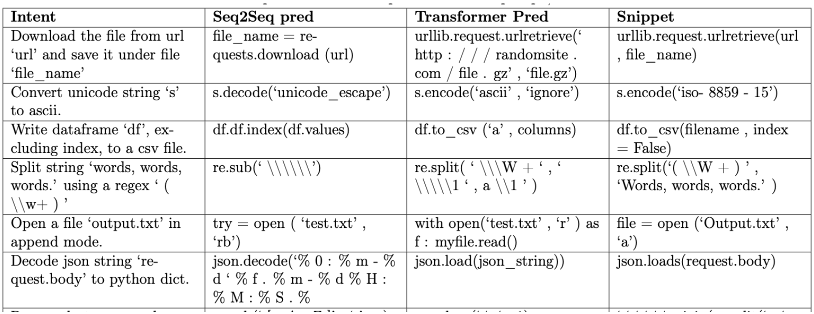


Intent	RoBERTa Prediction	Fine-tuned RoBERTa Prediction	Snippet
Download a file ‘url’ over HTTP.	requests.download(url)	‘r = requests.get(url)’	r = requests.get(url)
Retrieve each line from a file ‘File.txt’ as a list.	[with open(”.join(txt.txt’, ’r’) as ins:\n "’	[line for line in open(‘file.txt’) if ’r’)]"	[line.split() for line in open(‘File.txt’)]
Swap keys with values in a dictionary ‘a’.	[(k, v) for k, v in a.items()]	‘dict((k, v) for k, v in a.items()))	res = dict((v, k) for k, v in a.items())
Add an entry ‘wuggah’ at the beginning of list ‘myList’.	["mylist.startswith(‘∼a’)[0]", "mylist.startswith(‘∼a’)[0] for i in mylist]", "mylist.findall(‘\\w+’, ‘yourlist’)"]	["mylist.insert(0, ‘a’)", "newlist.insert(0, ‘a’)","’Hello\\n’.join(list(mylist))"]	myList.insert(0, ‘wuggah’)
Concatenate array of strings ‘[‘A’, ‘B’, ‘C’, ‘D’]‘ into a string.	‘"""".join([\’\’.join(lambda x: x) for x in ‘B’	[’"""""".join([\‘a\’, \‘b\’, \‘c\’, \‘d\’])’	"""""".join([‘A’, ‘B’, ‘C’, ‘D’])
Sort a list of dictionary ‘list’ first by key ‘points’ and then by ‘time’.	[‘sorted(list.items()), key=lambda item: datetime.index(‘x’, ‘sorted(list, key=lambda item: date-time.index(x.index())’, ‘sorted(list.items()), key=lambda item: item[1][1])’]	["list.sort(key=lambda item: item[‘date’], reverse=True)", "pd.list_of_sort(key=lambda item: item[‘date’], reverse=True)", ‘pd.merge(list(my_dictionary.items()), key=lambda item: item[1][1])’]	list.sort(key=lambda item: (item[‘points’], item[‘time’]))

Figure: Comparison of test predictions: fine-tuned Seq2Seq-RoBERTa v/s Seq2Seq-RoBERTa.

- The model worked well for list & tuples operations such as finding the length of a list, converting a list into a tuple, concatenating arrays, sorting lists, clock operations, and file operations such as read and open.
- The non fine-tuned model didn’t work well for complicated regex intents, os system calls, multiple operations with dictionaries, dataframe operations, etc.

Generated Translations: Transformer and Seq2Seq



Intent	Seq2Seq pred	Transformer Pred	Snippet
Download the file from url 'url' and save it under file 'file_name'	file_name = requests.download (url)	urllib.request.urlretrieve(' http : / / / randomsite . com / file . gz' , 'file.gz')	urllib.request.urlretrieve(url , file_name)
Convert unicode string 's' to ascii.	s.decode('unicode_escape')	s.encode('ascii' , 'ignore')	s.encode('iso- 8859 - 15')
Write dataframe 'df', excluding index, to a csv file.	df.df.index(df.values)	df.to_csv ('a' , columns)	df.to_csv(filename , index = False)
Split string 'words, words, words.' using a regex ' (\\w+) '	re.sub(' \\\\\\')	re.split(' \\\W + ' , ' \\\\\1 ' , a \\1 ')	re.split('(\\W +) ' , 'Words, words, words.')
Open a file 'output.txt' in append mode.	try = open ('test.txt' , 'rb')	with open('test.txt' , 'r') as f : myfile.read()	file = open ('Output.txt' , 'a')
Decode json string 'request.body' to python dict.	json.decode('% 0 : % m - % d ' % f . % m - % d % H : % M : % S . %	json.load(json_string))	json.loads(request.body)

Figure: Comparison of test predictions: fine-tuned Seq2Seq v/s Transformer.

- The transformer model worked well for the intents related to the os path and system calls, simple regex calls, dictionary operations such as sorting dictionaries, removing none values from dictionaries, etc.
- The transformer model had failed to infer the variable, parameters, function names, and types from the NL intent.

RQ2. What did we find from the ablation studies of Seq2Seq-RoBERTa?

Ablation study: Seq2Seq-RoBERTa on self-attention heads

Attention heads	Decoding method	Test BLEU	Test Rouge1 Precision	Test Rouge1 Recall	Test Rouge1 F1-score
2	Greedy	12.8621	0.3469	0.3259	0.3139
	Beam size = 4	12.8621	0.3469	0.3259	0.3139
	Beam size = 7	12.6930	0.3371	0.3221	0.3066
	Beam size = 10	12.8602	0.3375	0.3227	0.3072
	Beam size = 15	12.7773	0.3380	0.3243	0.3076
4	Greedy	13.2852	0.3357	0.3054	0.2988
	Beam size = 4	13.9182	0.3273	0.3144	0.3000
	Beam size = 7	14.0908	0.3291	0.3216	0.3050
	Beam size = 10	13.9419	0.3279	0.3211	0.3045
	Beam size = 15	13.8604	0.3275	0.3202	0.3039
6	Greedy	13.3928	0.3197	0.3110	0.2938
	Beam size = 4	13.3928	0.3197	0.3110	0.2938
	Beam size = 7	13.3976	0.3085	0.3106	0.2871
	Beam size = 10	13.5370	0.3110	0.3128	0.2897
	Beam size = 15	13.4991	0.3088	0.3111	0.2878
8	Greedy	13.4887	**0.3544**	0.3075	0.3109
	Beam size = 4	13.6894	0.3366	0.3103	0.3048
	Beam size = 7	13.8962	0.3365	0.3125	0.3050
	Beam size = 10	13.8202	0.3365	0.3114	0.3050
	Beam size = 15	13.8844	0.3364	0.3143	0.3062
12	Greedy	13.3493	0.3474	0.3179	0.3117
	Beam size = 4	13.8000	0.3399	0.3261	0.3126
	Beam size = 7	13.9197	0.3340	0.3253	0.3095
	Beam size = 10	**14.0733**	0.3337	0.3260	0.3096
	Beam size = 15	14.0307	0.3350	**0.3265**	**0.3150**

Table: Ablation study on the number of self-attention heads for DistilRoBERTa while keeping layers fixed at 1. (batch_size=8, 50 epochs)

Ablation study: Depth of Seq2Seq-RoBERTa layers

Number of layers	Decoding Method	Test BLEU	Rouge1 F1	Rouge2 F1	RougeL F1
3	Greedy	14.2945	0.3238	0.1148	0.3106
	Beam size= 4	15.1319	0.3258	0.1240	0.3127
	Beam size = 10	15.3172	0.3268	0.1250	0.3139
	Beam Size = 15	15.2510	0.3254	0.1246	0.3125
5	Greedy	15.2415	0.3506	0.1248	**0.3349**
	Beam size= 4	15.9160	0.3520	0.1267	0.3334
	Beam size= 7	15.9965	**0.3525**	**0.1276**	0.3340
	Beam Size = 10	16.0659	**0.3525**	0.1256	0.3337
	Beam Size = 15	**16.0700**	0.3508	0.1274	0.3330

Table: Ablation study on the depth of RoBERTa layers in the hybrid Se2Seq while keeping the attention heads to be 8. (batch_size=8, 50 epochs)

RQ3. Does the SentencePiece tokenizer improve the performance of the Transformer model?

Tokenizer	BLEU	Rouge1 F1	Rouge2 F1	Rouge4 F1	RougeL F1	Token Accuracy
WordPiece	17.3237	0.6786	0.5339	0.2448	0.6786	9.1647
Unigram	**20.9678**	0.6718	0.5214	0.2376	0.6718	12.5242
BPE	19.3402	**0.6937**	**0.5495**	**0.2567**	**0.6937**	10.1486
CoNaLa code tokenizer	15.3834	0.5449	0.2628	0.1311	0.5347	**14.7041**

Table: Test metrics for Transformer using SentencePiece tokenizers.

- The transformer-Unigram performed better by 36.30%, the transformer-WordPiece performed better by 12.6% and the transformer- BPE performed better by 25.72% over the CoNaLa code tokenizer.

RQ4. Do data augmentation and pretraining techniques improve the results of the proposed hybrid Seq2Seq-BART architecture?

- Transfer learning is a means of extracting knowledge from a source setting and applying it to a different target setting.
- A pretrained model is a saved network that was previously trained on a large unannotated dataset, typically on large-scale (NL, code) pairs for our task.
- These weights can be reused to fine-tune the pretrained model on the smaller training annotated dataset.

Yes, they do!

Seq2Seq-BART on Augmented Datasets	Test BLEU
Seq2Seq-BART on $3x$ size of original dataset	25.7710
Seq2Seq-BART on $5x$ size of original dataset	25.1601
Seq2Seq-BART on CoNaLa dataset	24.2990
Fine-tuned Seq2Seq-BART on CoNaLa, pretrained on mined100k corpus	26.5379
Fine-tuned Seq2Seq-BART on $3x$ CoNaLa, pretrained on mined100k corpus	**27.8235**
Fine-tuned Seq2Seq-BART on $5x$ CoNaLa, pretrained on mined100k corpus	25.3153
TranX on CoNaLa train dataset	25.1050
Seq2Seq-RoBERTa on CoNaLa dataset	17.0032
Fine-tuned Seq2Seq-RoBERTa on CoNaLa, pretrained on mined30k corpus	18.8853

Table: Results of Seq2Seq-BART models on Augmented Datasets.

- The fine-tuning of these pretrained hybrid Seq2Seq architectures on the training set improved the test BLEU score metric by 9.2% over non-pretrained Seq2Seq-BART architecture and by 11.2% over non-pretrained Seq2Seq-RoBERTa.

RQ5. Does the proposed hybrid Seq2Seq-BART architecture work bidirectionally to reverse the hypothesis?

- We reversed the input and the output for the Seq2Seq-BART architecture.
- We used the same values for the hyperparameters used for the Nl2Code-BART model.
- We observed that the proposed architecture worked quite well for the reversed hypothesis.

BLEU	**Rouge1 F1**	**Rouge2 F1**	**RougeL F1**
21.80291	0.45516	0.21644	0.407032

Table: Test metrics for the Code2NL.

RQ6. How well does the code completion task work with the use of neural language model?

A code completion system suggests pieces of code by understanding the context of the incomplete code snippet written by the developer.
Approach:

- We derived a contextual embedding and language modeling of source code by training a RoBERTa model on Algorithms' Python code repositories.
- We used the subword tokenizer, ByteLevel BPE, to model the source code then trained the RoBERTa tokenizer for source code.
- We finally performed the fill-in mask token task to judge the performance of the model.

Fill Mask task	Prediction/Output	Score
”<mask>os”	‘import os’	**0.9979**
if (x is not None) <mask>(x > 1)	'if (x is not None) and (x>1)	**0.6732**
	'if (x is not None) / (x>1)	0.0770
	'if (x is not None) | (x>1)	0.0769
	if (x is not None) or (x>1)	0.0291
if self.graph[u].count([w, v]) <mask>0:	if self.graph[u].count([w, v]) == 0:	**0.5888**
	'if self.graph[u].count([w, v])!= 0:'	0.3329
	if self.graph[u].count([w, v]) >0:	0.0360
sum = a <mask>b	sum = a * b	**0.7289**
	sum = a + b	0.1288
	'sum = a - b	0.0186
	sum = a // b'	0.0118
mdist = [<mask>for i in range(V)]	mdist = [0 for i in range(V)]	**0.7845**
	mdist = [i for i in range(V)]	0.0978
	mdist = [True for i in range(V)]	0.0320
	mdist = [False for i in range(V)]	0.0235
	mdist = [1 for i in range(V)]	0.0088

Table: Fill-mask task for code completion.

Conclusion

- Parsable Python source code snippets can be directly generated from NL intent using deep learning architectures, without necessarily using heavily feature engineered semantic parsers.
- The proposed Seq2Seq-BART architecture has exceeded the performance the state-of-the-art neural semantic parser, TranX on BLEU-4 metric score by 10.82% for NL2Code on CoNaLa dataset.
- The code generation of the proposed algorithm can be improved by using the pretraining approach and the data augmentation techniques.
- The output from the Transformer architecture can be improved for the Nl2Code task by using subword tokenizers (BPE).
- RoBERTa based language model for Python source code has been highly effective for code completion task.

Future Work

- Explore our novel hybrid Seq2Seq architecture on more datasets such as Django [14], WikiSQL [27] to see how well the proposed architecture generalizes to other programming languages and language-specific domains.
- Incorporate Abstract Syntax Trees (ASTs) into our proposed architecture and devise strategies for more conditioned text generation.
- Evaluate the model using BERTSCORE [26] as an evaluation metric for the translated code snippets.
- Allow the proposed system to explore multi-line contextual awareness by creating a dataset of (NL, method definition) to train our proposed architecture to generate multi-line source code snippets as generating an entire function.

Fact checking for low-resource languages with code switching capabilities

Code Switched Fake News	English Interpretation
WHO ne Bharat ko Karuna free declare Kar Diya hai. Humare PM narinder Modi is being awarded best PM award by UNESCO. WHO Bharot keh Korona free boleche. koreche ভারতকে করানো মুক্ত ঘাষেণ koreche	WHO has declared India as Corona free. Our Prime minister Narindra Modi is being awarded best PM award by UNESCO.. বিশ্ব স্বাস্থ্য সংস্থা ভারতকে করানো মুক্ত ঘাষেণা করেছে।
Nahi bole toh bhi kam se kam 250-300 nuske bata chuke hai yeh log Corona khatam karne ke liye. Aur WHO ainwayi keh raha hai ke Corona ke liye vaccine nahi bani abtak	These people have suggested atleast 250-300 prescriptions to cure CORONA. And WHO, without any reason, is saying we have no vaccine for CORONA
WHO boleche jeh Bharat naki Corona free. Amader PM Modi keh UNESCO shrestho PM award diyeche.	

Why?

- In the non-homogeneous world, multilingual society necessitates the development of multilingual fact-checking systems where fake news spread in multilingual languages over social media and messaging platforms and often claims get unverified and led to irreparable consequences in the society.
- Although false information transcends geographical and linguistic boundaries, the majority of research in the area has been on English.
- Besides the spread of misinformation in low-resource languages in the developing world, one of the other way of spreading misinformation is the code switching phenomenon in the social media and popular messaging platforms.
- This requires to develop fact-checking methods for multilingual languages which considers code switching representation of claims.

RQ1: How to develop multilingual fact-checking systems?

Background

- Very few works have been done so far on multilingual fact-checking.
- [5] Dementieva and Panchenko [2020] used translation system to develop fake news detection system based on multilingual evidence but we need more relevant multilingual datasets for testing the performance of multilingual models.
- A multilingual fact-checking dataset for 25 languages was introduced by [8] Gupta and Srikumar [2021],

Approach

- Initially, I plan to research about the state-of-the-art fact-checking systems and report their performance on multilingual datasets.
- Thereafter, I plan to develop fact-checking system that will not only incorporate cross-lingual transfer capabilities but also handle code switching representation of misinformation representation.
- I plan to use the dataset introduced by Kazemi et al. [2021] for Indian language(Bengali, Hindi, Malayalam, Tamil) and also the FakeCovid dataset introduced by Gupta and Srikumar [2021] for the evaluation of newly developed methods for multilingual fact-checking.
- I also plan to create a bigger curated annotated dataset by curating posts from Facebook and tweets from Twitter which are predominantly popular in India following the steps of [Kazemi et al., 2021]. This will allow to learn to align and leverage resources from multilingual datasets for fact-checking.

RQ2: How to efficiently build language independent models that handles multi-tasks learning capabilities for low-resource starved Indian & African languages?

Background

- Multilingual models(Devlin et al. [2019], Conneau et al. [2020a], Xue et al. [2021] [6], [4], [24]) are generally trained on large multilingual datasets with shared vocabulary.
- Pires et al. [2019][16] concluded that zero-shot cross-lingual generalization is made possible by using a shared vocabulary where they fine-tuned mBERT on downstream task in the source language and evaluated on the same task in the target language.
- However, K et al. [2019][9] showed that multilinguality can also be achieved in a joint-training setting without sharing joint vocabulary of two languages.
- Conneau et al. [2020b][3] and Artetxe et al. [2020][1] concluded that instead of multilingual pre-training, shared statistical features of language spaces primarily allow zero-shot cross-lingual transfer.

Approach

- In my research stay, I would like to explore and apply the technique of cross-lingual transfer learning from a monolingual model to multilingual as shown by de Vries and Nissim [2021][22] to Indian languages.
- I would also like to explore meta learning (Bengio et al. [1991])[2] to learn shareable structure across multiple tasks with annotated data and develop uniform cross-lingual transfer methods for unseen languages.
- I would also like to find out to what extent pixel-based representations (Rust et al. [2022][19]) of language can be leveraged for cross-lingual transfer.

RQ3: How to improve code switching representation for multilingual fact checking?

Background

- In multilingual communities, code-switching is a common phenomenon in which a person speaks more than one language during a conversation.
- Multilingual language models (Devlin et al. [2019][6] and Conneau et al. [2020a][4]) have achieved SOTA performance in various monolingual and cross-lingual NLP tasks but limited focused on code-switching.
- To represent code-switching text, one can train the language model with the word embeddings of the primary language (PL) and the embedded language (EL) from FastText [Grave et al., 2018].
- In order to handle noisy and out-of-vocabulary mixed words tokens (PL+EL), subword-level embeddings from FastText [Grave et al., 2018] and SentencePiece [Kudo and Richardson, 2018] can be utilized to generate the OOV tokens representations.
- Winata et al. [2021][23] found that pre-trained multilingual models do not always ensure high-quality representations on code-switching, whereas using meta-embeddings yields comparable results with a lot less complexity.

Approach

- To the best of my knowledge, there does not exist a single fact-checking multi-lingual dataset with code-switching representation.
- In order to address low resource data pairs, I would like to collect a corpus of code-switched text claims from social media using a combination of sets of anchor words that exist in one language and sentence-level language taggers.
- During my research stay at TU-Darmstadt, I want to develop language agnostic representational learning approaches which will improve the generalization of language models on code-switched data especially for low-resource language pairs.
- Explore meta-embeddings (methods to combine multilingual meta-embeddings such as concat, linear, and self-attention) and hierarchical meta-embeddings (HME) [Winata et al., 2019b] model for code-switched representational learning for low-resource fact checking system.
- For the task of language modeling, conduct an exhaustive analysis of the relationship between cognate words and code-switching for Indian languages, although it might not so benefit as majority of Indian languages don't share the same script.

Pragmatic Program Synthesis

Background:

- Ambiguity of NL expression and the inability of humans to describe their coding intentions properly poses challenge for synthesizing program generation from NL descriptions.
- As many different synthesized programs could be generated from the same input but there are few ways to decide heuristically which one is correct.

Problem Statement

- The problem is analogous to speaker-listener communication, where the listener has to interpret the in- tended meaning of the speaker.
- The "Rational Speech Act model" (RSA) [Goodman and Frank, 2016][7] has been effective to resolve ambiguities in speaker-listener communication by reasoning about what speaker could have expressed (but chose not to) from listeners' perspective.
- Various models of cooperative communication and recursive reasoning of intents can be applied for pragmatic program generation.
- The program synthesis system would construct the alternate programs while generatively reasoning about what a human would have said if they intended to communicate each of those programs for a given (ambiguous) human instruction.
- This way, all programs which would have better been described differently can be discarded.
- Pu et al., 2020 [17] applied this pragmatic reasoning approach to program synthesis from examples.
- However, to the best of my knowledge, this approach is still not well developed for program synthesis and has not yet been applied to language-based program synthesis

Approach

- I would like to consider using an encoder-decoder approach (similar to machine translation).
- These models would then be the basis for the recursive reasoning model.
- As for exploring the benefits regarding program comprehension and education, I can resort to a substantial methodological arsenal, including controlled experiments, think-aloud studies, eye tracking, and neuro-imaging (EEG, fMRI).

References I

Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. "On the Cross-lingual Transferability of Monolingual Representations." In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 4623–4637. DOI: 10.18653/v1/2020.acl-main.421. URL: https://aclanthology.org/2020.acl-main.421.

Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. "Learning a synaptic learning rule." In: *IJCNN-91-Seattle International Joint Conference on Neural Networks* ii (1991), 969 vol.2-.

Alexis Conneau et al. "Emerging Cross-lingual Structure in Pretrained Language Models." In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 6022–6034. DOI: 10.18653/v1/2020.acl-main.536. URL: https://aclanthology.org/2020.acl-main.536.

References II

Alexis Conneau et al. “Unsupervised Cross-lingual Representation Learning at Scale.” In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 8440–8451. DOI: 10.18653/v1/2020.acl-main.747. URL: https://aclanthology.org/2020.acl-main.747.

Daryna Dementieva and Alexander Panchenko. “Fake News Detection using Multilingual Evidence.” In: *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. 2020, pp. 775–776. DOI: 10.1109/DSAA49011.2020.00111.

Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

References III

Noah D. Goodman and Michael C. Frank. “Pragmatic Language Interpretation as Probabilistic Inference.” In: *Trends in Cognitive Sciences* 20.11 (2016), pp. 818–829. ISSN: 1364-6613. DOI: https://doi.org/10.1016/j.tics.2016.08.005. URL: https://www.sciencedirect.com/science/article/pii/S136466131630122X.

Ashim Gupta and Vivek Srikumar. “X-Fact: A New Benchmark Dataset for Multilingual Fact Checking.” In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 675–682. DOI: 10.18653/v1/2021.acl-short.86. URL: https://aclanthology.org/2021.acl-short.86.

Karthikeyan K et al. *Cross-Lingual Ability of Multilingual BERT: An Empirical Study*. 2019. DOI: 10.48550/ARXIV.1912.07840. URL: https://arxiv.org/abs/1912.07840.

References IV

Ashkan Kazemi et al. "Claim Matching Beyond English to Scale Global Fact-Checking." In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 4504–4517. DOI: 10.18653/v1/2021.acl-long.347. URL: https://aclanthology.org/2021.acl-long.347.

Taku Kudo. *Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates*. 2018. arXiv: 1804.10959 [cs.CL].

Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL].

Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries." In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://www.aclweb.org/anthology/W04-1013.

References V

Yusuke Oda et al. "Learning to Generate Pseudo-code from Source Code Using Statistical Machine Translation." In: *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ASE '15. Lincoln, Nebraska, USA: IEEE Computer Society, Nov. 2015, pp. 574–584. ISBN: 978-1-5090-0025-8. DOI: 10.1109/ASE.2015.36. URL: https://doi.org/10.1109/ASE.2015.36.

Kishore Papineni et al. "BLEU: A Method for Automatic Evaluation of Machine Translation." In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL '02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://doi.org/10.3115/1073083.1073135.

Telmo Pires, Eva Schlinger, and Dan Garrette. "How Multilingual is Multilingual BERT?" In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 4996–5001. DOI: 10.18653/v1/P19-1493. URL: https://aclanthology.org/P19-1493.

References VI

Yewen Pu et al. "Program Synthesis with Pragmatic Communication." In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.

Alec Radford et al. "Language Models are Unsupervised Multitask Learners." In: 2019.

Phillip Rust et al. *Language Modelling with Pixels*. 2022. arXiv: 2207.06991 [cs.CL].

Mike Schuster and Kaisuke Nakajima. "Japanese and Korean voice search." In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pp. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079.

Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. 2016. arXiv: 1508.07909 [cs.CL].

References VII

Wietse de Vries and Malvina Nissim. "As Good as New. How to Successfully Recycle English GPT-2 to Make Models for Other Languages." In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Association for Computational Linguistics, 2021. DOI: 10.18653/v1/2021.findings-acl.74. URL: https://doi.org/10.18653%2Fv1%2F2021.findings-acl.74.

Genta Indra Winata et al. *Are Multilingual Models Effective in Code-Switching?* 2021. DOI: 10.48550/ARXIV.2103.13309. URL: https://arxiv.org/abs/2103.13309.

Linting Xue et al. "mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer." In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 483–498. DOI: 10.18653/v1/2021.naacl-main.41. URL: https://aclanthology.org/2021.naacl-main.41.

References VIII

Pengcheng Yin and Graham Neubig. "TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation." In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 7–12. DOI: 10.18653/v1/D18-2002. URL: https://www.aclweb.org/anthology/D18-2002.

Tianyi Zhang et al. *BERTScore: Evaluating Text Generation with BERT*. 2020. arXiv: 1904.09675 [cs.CL].

Victor Zhong, Caiming Xiong, and Richard Socher. "Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning." In: *CoRR* abs/1709.00103 (2017).

Thank you for listening!

Appendix

Seq2Seq

Model hyperparameters	**Values**
The input dimension of the model, INPUT_DIM	len(nl_src.vocab)
The output dimension of the model, OUTPUT_DIM	len(code_target.vocab)
The encoder embedding dimension, ENC_EMB_DIM	256
The decoder embedding dimension, DEC_EMB_DIM	256
The encoder hidden layer dimension, ENC_HID_DIM	512
The decoder hidden layer dimension, DEC_HID_DIM	512

Table: Seq2Seq model configuration.

Transformer model configuration

Model hyperparameters	Values
The dimension of the model, “D_MODEL”	256
The number of self-attention heads, “N_HEADS”	3
The hidden size of the encoder, and decoder, “HIDDEN_SIZE”	512
The maximum length of the input sequence, “MAX_LEN”	50

Table: Transformer model configuration.

Se2Seq-RoBERTa model configuration

Model hyperparameters	Values
"encoder_max_length"	512
"decoder_max_length"	256
"is_encoder_decoder"	True
"length_penalty"	2.0
"max_length"	64
"model_type"	"encoder-decoder"
"*no_repeat_ngram_size*"	3
"num_beams"	4
"tie_encoder_decoder"	True
"vocab_size"	50265
"attention_probs_dropout_prob"	0.1
"hidden_act"	"gelu"
"hidden_dropout_prob"	0.1
"hidden_size"	768
"layer_norm_eps"	1e-05
"max_position_embeddings"	514
"min_length"	0
"model_type"	"roberta"
"num_attention_heads"	12
"num_beam_groups"	1
"num_hidden_layers"	6
"num_return_sequences"	3
"top_k"	50
"top_p"	1.0

Table: Seq2Seq-RoBERTa model configuration.

Seq2Seq-BART model configuration

Model hyperparameters	Values
"max_length"	128
"min_length"	12
"do_sample"	True
"early_stopping"	True
"length_penalty"	1.0
"temperature"	1.0
"*no_repeat_ngram_size*"	3
"num_beams"	4
"encoder_no_repeat_ngram_size"	0
"repetition_penalty"	1.0
"attention_probs_dropout_prob"	0.1
"bos_token_id"	0
"pad_token_id"	1
"eos_token_id"	2
"use_cache"	True
"decoder_start_token_id"	2
"output_hidden_states"	False
"diversity_penalty"	0.0
"output_attentions"	False
"num_beam_groups"	1
"output_scores"	False
"num_return_sequences"	3
"top_k"	50
"top_p"	1.0

Table: Seq2Seq-BART model configuration.

What did we find from the ablation studies of the developed architectures?

Ablation study of the self-attention heads of Transformer

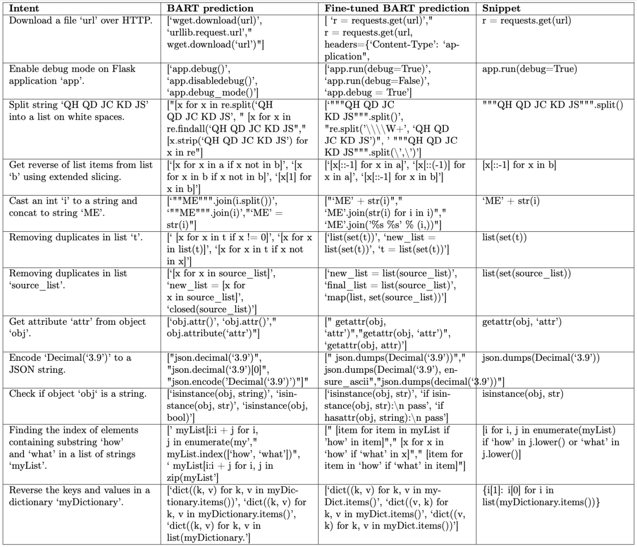
Attention heads	Validation BLEU	Test BLEU	Validation RougeL F1	Test RougeL F1	Validation token accuracy	Test token accuracy
2	20.6100	16.7480	0.5730	0.5497	16.5211	13.5680
4	21.6802	17.3580	0.5871	0.5598	**18.4361**	14.3770
8	**23.5436**	17.6857	**0.6074**	0.5707	17.2164	14.1223
16	22.8340	17.2390	0.5957	0.5594	16.8025	13.4464
32	22.3587	**17.7187**	0.5977	**0.5725**	18.3544	**15.0941**

Table: Ablation study of the self-attention heads of Transformer while keeping the number of layers fixed at 3.

Ablation study on the number of layers of Transformer

Number of layers	Validation BLEU	Test BLEU	Validation RougeL F1	Test RougeL F1	Validation token accuracy	Test token accuracy
1	18.7609	15.2000	0.5673	0.5400	15.7220	14.3319
2	21.4045	16.1740	0.5779	0.5469	18.0871	15.5930
3	21.9000	**16.2420**	**0.5976**	**0.5716**	**20.1941**	**16.6751**
4	21.1420	15.0200	0.5892	0.5539	18.9563	15.5323
5	**22.1571**	15.0414	0.5899	0.5493	19.7900	16.4548
6	19.1737	16.0100	0.5743	0.5642	15.6427	14.2967

Table: Ablation study on the number of layers of Transformer while keeping the attention heads fixed at 8.



Intent	BART prediction	Fine-tuned BART prediction	Snippet
Download a file 'url' over HTTP.	['wget.download(url)', 'urllib.request.url'," wget.download('url')"]	['r = requests.get(url)'," r = requests.get(url, headers={'Content-Type': 'ap-plication",	r = requests.get(url)
Enable debug mode on Flask application 'app'.	['app.debug()', 'app.disabledebug()', 'app.debug_mode()']	['app.run(debug=True)', 'app.run(debug=False)', 'app.debug = True']	app.run(debug=True)
Split string 'QH QD JC KD JS' into a list on white spaces.	["[x for x in re.split('QH QD JC KD JS', " [x for x in re.findall('QH QD JC KD JS"," [x.strip('QH QD JC KD JS') for x in re"]	['"""QH QD JC KD JS""".split()', "re.split('\\\\W+', 'QH QD JC KD JS')", ' """QH QD JC KD JS""".split(\',\')']	"""QH QD JC KD JS""".split()
Get reverse of list items from list 'b' using extended slicing.	['[x for x in a if x not in b]', '[x for x in b if x not in b]', '[x[1] for x in b]']	['[x[::-1] for x in a]', '[x[::(-1)] for x in a]', '[x[::-1] for x in b]']	[x[::-1] for x in b]
Cast an int 'i' to a string and concat to string 'ME'.	['"""ME""".join(i.split())', '"""ME""".join(i)',"'ME' = str(i)"]	["'ME' + str(i)"," 'ME'.join(str(i) for i in i)"," 'ME'.join('%s %s' % (i,))"]	'ME' + str(i)
Removing duplicates in list 't'.	[' [x for x in t if x != 0]', '[x for x in list(t)]', '[x for x in t if x not in x]']	['list(set(t))', 'new_list = list(set(t))', 't = list(set(t))']	list(set(t))
Removing duplicates in list 'source_list'.	['[x for x in source_list]', 'new_list = [x for x in source_list]', 'closed(source_list)']	['new_list = list(source_list)', 'final_list = list(source_list)', 'map(list, set(source_list))']	list(set(source_list))
Get attribute 'attr' from object 'obj'.	['obj.attr()', 'obj.attr()'," obj.attribute('attr')"]	[" getattr(obj, 'attr')","getattr(obj, 'attr')", 'getattr(obj, attr)']	getattr(obj, 'attr')
Encode 'Decimal('3.9')' to a JSON string.	["json.decimal('3.9')", "json.decimal('3.9')[0]", "json.encode('Decimal('3.9')')"]"	[" json.dumps(Decimal('3.9'))"," json.dumps(Decimal('3.9'), en-sure_ascii","json.dumps(decimal('3.9'))"]	json.dumps(Decimal('3.9'))
Check if object 'obj' is a string.	['isinstance(obj, string)', 'isin-stance(obj, str)', 'isinstance(obj, bool)']	['isinstance(obj, str)', 'if isin-stance(obj, str):\n pass', 'if hasattr(obj, string):\n pass']	isinstance(obj, str)
Finding the index of elements containing substring 'how' and 'what' in a list of strings 'myList'.	[' myList[i:i + j for i, j in enumerate(my'," myList.index(['how', 'what'])", ' myList[i:i + j for i, j in zip(myList']	[" [item for item in myList if 'how' in item]"," [x for x in 'how' if 'what' in x]"," [item for item in 'how' if 'what' in item]"]	[i for i, j in enumerate(myList) if 'how' in j.lower() or 'what' in j.lower()]
Reverse the keys and values in a dictionary 'myDictionary'.	['dict((k, v) for k, v in myDic-tionary.items())', 'dict((k, v) for k, v in myDictionary.items()', 'dict((k, v) for k, v in list(myDictionary.']	['dict((k, v) for k, v in my-Dict.items()', 'dict((v, k) for k, v in myDict.items()', 'dict((v, k) for k, v in myDict.items())']	{i[1]: i[0] for i in list(myDictionary.items())}

Figure: Comparison of test predictions: fine-tuned Seq2Seq-BART v/s Seq2Seq-BART.

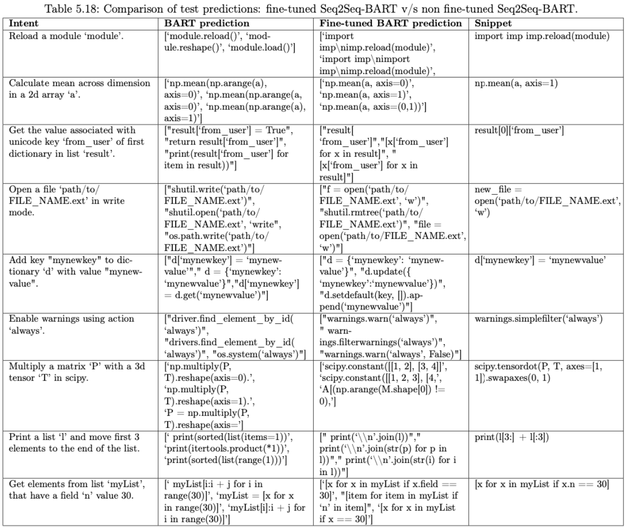


Table 5.18: Comparison of test predictions: fine-tuned Seq2Seq-BART v/s non fine-tuned Seq2Seq-BART.

Intent	BART prediction	Fine-tuned BART prediction	Snippet
Reload a module ‘module’.	[‘module.reload()’, ‘module.reshape()’, ‘module.load()’]	[‘import imp\nimp.reload(module)’, ‘import imp\nimport imp\nimp.reload(module)’,	import imp imp.reload(module)
Calculate mean across dimension in a 2d array ‘a’.	[‘np.mean(np.arange(a), axis=0)’, ‘np.mean(np.arange(a, axis=0)’, ‘np.mean(np.arange(a), axis=1)’]	[‘np.mean(a, axis=0)’, ‘np.mean(a, axis=1)’, ‘np.mean(a, axis=(0,1))’]	np.mean(a, axis=1)
Get the value associated with unicode key ‘from_user’ of first dictionary in list ‘result’.	["result[‘from_user’] = True", "return result[‘from_user’]", "print(result[‘from_user’] for item in result))"]	["result[‘from_user’]","[x[‘from_user’] for x in result]", " [x[‘from_user’] for x in result]"]	result[0][‘from_user’]
Open a file ‘path/to/ FILE_NAME.ext’ in write mode.	["shutil.write(‘path/to/ FILE_NAME.ext’)", "shutil.open(‘path/to/ FILE_NAME.ext’, ‘write", "os.path.write(‘path/to/ FILE_NAME.ext’)"]	["f = open(‘path/to/ FILE_NAME.ext’, ‘w’)", "shutil.rmtree(‘path/to/ FILE_NAME.ext’)", "file = open(‘path/to/FILE_NAME.ext’, ‘w’)"]	new_file = open(‘path/to/FILE_NAME.ext’, ‘w’)
Add key "mynewkey" to dictionary ‘d’ with value "mynewvalue".	["d[‘mynewkey’] = ‘mynewvalue’"," d = {‘mynewkey’: ‘mynewvalue’}","d[‘mynewkey’] = d.get(‘mynewvalue’)"]	["d = {‘mynewkey’: ‘mynewvalue’}", "d.update({ ‘mynewkey’:‘mynewvalue’})", "d.setdefault(key, []).append(‘mynewvalue’)"]	d[‘mynewkey’] = ‘mynewvalue’
Enable warnings using action ‘always’.	["driver.find_element_by_id(‘always’)", "drivers.find_element_by_id(‘always’)", "os.system(‘always’)"]	["warnings.warn(‘always’)", " warnings.filterwarnings(‘always’)", "warnings.warn(‘always’, False)"]	warnings.simplefilter(‘always’)
Multiply a matrix ‘P’ with a 3d tensor ‘T’ in scipy.	[‘np.multiply(P, T).reshape(axis=0).’, ‘np.multiply(P, T).reshape(axis=1).’, ‘P = np.multiply(P, T).reshape(axis=’]	[‘scipy.constant([[1, 2], [3, 4]]’, ‘scipy.constant([[1, 2, 3], [4,’, ‘A[(np.arange(M.shape[0]) != 0),’]	scipy.tensordot(P, T, axes=[1, 1]).swapaxes(0, 1)
Print a list ‘l’ and move first 3 elements to the end of the list.	[‘ print(sorted(list(items=1))’, ‘print(itertools.product(*1))’, ‘print(sorted(list(range(1)))’]	[" print(‘\\n’.join(l))"," print(‘\\n’.join(str(p) for p in l))"," print(‘\\n’.join(str(i) for i in l))"]	print(l[3:] + l[:3])
Get elements from list ‘myList’, that have a field ‘n’ value 30.	[‘ myList[i:i + j for i in range(30)]’, ‘myList = [x for x in range(30)]’, ‘myList[i]:i + j for i in range(30)]’]	[‘[x for x in myList if x.field == 30]’, "[item for item in myList if ‘n’ in item]", ‘[x for x in myList if x == 30]’]	[x for x in myList if x.n == 30]

Figure: Comparison of test predictions: fine-tuned Seq2Seq-BART v/s Seq2Seq-BART.

Advantages of using subword unit LM for code

- Firstly, as the model had a smaller vocabulary size due to reduced level of data sparsity, it might have better performance over the non-subword tokenizers.
- Secondly, the model could handle the *OoV* problem by synthesizing the missing *OoV* tokens that were seen in the training data using the smaller subtoken units.

One cycle training policy

The one cycle policy follows the Cyclical Learning Rate (CLR) to obtain faster training time with regularization effect but with a slight modification, thus producing very fast results when training complex models. The original one cycle policy has three steps:

- Initially, the learning rate is progressively increased from lr_{max}/div_{factor} to lr_{max} and at the same time, the momentum is decreased from mom_{max} to mom_{min}.
- Then, the learning rate is decreased from lr_{max} to lr_{max}/div_{factor} and at the same time, the momentum is increased progressively from mom_{min} to mom_{max}.
- At last, the learning rate is decreased further from lr_{max}/div_{factor} to $lr_{max}/(div_{factor} \times 100)$ and the momentum is kept steady at mom_{max}.

How TranX works?

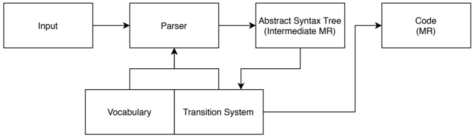


Figure: The overview of the workflow of TranX. Figure drawn from [25]

- TranX [25] is a neural semantic parser having a transition system, and a neural encoder-decoder network to compute action probabilities.
- The transition system is extendable to new programming languages (PL) while the neural network is PL agnostic, i.e., independent of the specific PL.
- The transition system of TranX maps the given input NL utterance x into an AST z using a series of three tree-construction actions.
- TranX utilizes ASTs as the intermediate meaning representations (MRs) to extract over the domain-specific structure of MRs.
- The parsing process involves the conversion of the intermediate generated AST z into a domain-specific meaning representation y.
- TranX also uses a probabilistic neural network model $p(z \mid x)$ to score each hypothesis AST and to reflect the topology of ASTs.

Evaluation Metrics: BLEU

- BLEU (bilingual evaluation understudy) [15] measures the translation closeness by counting matches of n-grams in candidate and reference translation.
- BLEU metric does not take into account the intelligibility or grammatical correctness of a translated text.
- The BLEU is defined by

$$BP = \begin{cases} 1 & \text{if } c > r \\ \exp(1 - \frac{r}{c}) & \text{if } c \leq r \end{cases} \tag{3}$$

$$BLEU = BP \cdot \exp(\sum_{n=1}^{N} \frac{1}{N} \log p_n) \tag{4}$$

$$p_n = \frac{\sum_{c \ \epsilon \ \{candidates\}} \sum_{n-gram \ \epsilon \ c} count_{clip}(n-gram)}{\sum_{c' \ \epsilon \ \{candidates\}} \sum_{n-gram \ \epsilon \ c'} count(n-gram')} \tag{5}$$

- where p_n measures the modified n-gram precision between a document with candidate translations and a set of human authored reference documents, and the brevity penalty (BP) down-scales the score for outputs shorter than the reference.
- Candidates are the set of sentences to be evaluated. $count(n-gram')$ counts the number of times the n-gram appears in the candidate sentence, and $count_{clip}(n-gram)$ is the same albeit clipped such that it does not exceed the number of times it appears in one of the reference sentences (which may be zero).

Corpus-BLEU

- It calculates a single corpus-level BLEU score (aka. system-level BLEU) for all the hypotheses and their respective references.
- The original BLEU metric [15] accounts for the micro-average precision (i.e., summing the numerators and denominators for each hypothesis-reference(s) pairs before the division).

Sentence-BLEU

- It computes the BLEU metric on a single sentence pair. It calculates the averaging of the macro-average precision.
- However, the meaning of the sentence-level BLEU is more or less a special case of the corpus-level BLEU, and it can easily get zero value without smoothing function.

Rouge Scores

- ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metric [13] measures the n-gram overlap between generated translation and its reference translation. It is a widely used evaluation metric for summarization.
- As the ROUGE score only measures token hard-match, in some cases, the ROUGE score penalizes two sentences that convey the same semantic information, but this metric highly rewards sentences with completely different semantics yet in similar surface forms.
- **ROUGE-N**: measures unigram, bigram, trigram, and higher-order n-gram overlap.
- **ROUGE-L**: measures the longest matching sequence of words using Longest Common Subsequence (LCS) algorithm.
- The advantage of LCS is that it reflects the sentence-level word order as it does not require consecutive matches but in-sequence matches. Since it automatically considers the longest in-sequence common n-grams, there is no need for predefined n-gram length.

Tokenization Models: Byte Pair Encoding

In subword tokenization algorithms, rare words are decomposed into meaningful subwords instead of frequently used words split into subwords.

- Byte pair encoding (BPE) for word segmentation [21], the algorithm relies on a pre-tokenizer that splits the training data into words.
- The algorithm then creates a set of unique words and their frequency of occurrence in the training data after the pre-tokenization step.
- Then, a base vocabulary is created which consists of all symbols that occur in the set of unique words and the tokenizer learns the merge rules to form a new symbol from given pair of symbols in the base vocabulary.
- The tokenizer gets trained until the vocabulary size reaches the defined vocabulary size which is a hyperparameter assigned for the training of the tokenizer.

WordPiece

- WordPiece is a subword tokenization algorithm introduced in [20]. This algorithm is quite similar to BPE. WordPiece tokenizer is used in BERT, DistilBERT architectures.
- This algorithm first includes all the characters present in the training data in its vocabulary then learns the given number of merge rules progressively.
- WordPiece picks up the symbols which maximize the likelihood of the training data if those symbols are added to the vocabulary, whereas BPE picks up the most frequent symbol pairs.
- The training of WordPiece tokenizer aims to find the symbol pairs that maximize the likelihood of the training data and for which the probability of the merged pairs divided by the probabilities of its first symbol, followed by its second symbol is the highest among all symbol pairs.
- E.g., symbol pairs such as "a", followed by "b" will be merged if the probability of "ab" divided by "a" and "b" is the highest among all the symbol pairs..

Unigram

- Unigram [11] is not based on merge rules such as BPE, or WordPiece. This means that the algorithm has several ways of tokenizing new text after training.
- At first, Unigram initializes its base vocabulary to a large number of symbols with pre-tokenized words and the most common substrings then it progressively trims down each symbol to reduce the size of the vocabulary but keeping the base characters so that any word can be tokenized.
- Given the unigram language model and the vocabulary, the Unigram algorithm computes the log-likelihood loss over the training data at each training step.
- Then, for each symbol in the vocabulary, the algorithm computes the increase in the training loss if the selected symbol is removed from the vocabulary.
- The algorithm removes p percent (p usually being 10% or 20%) of symbols for which the increase in the training loss is the lowest. This process is repeated unless the vocabulary has reached the desired size.
- Assuming that the training data consists of the words $x_1, \ldots, x_N$ and that the set of all possible tokenizations for a word x_i is defined as S(x_i), then the overall loss is defined as $-\sum_{i=1}^{N} \log \sum_{x \epsilon S(x_i)} p(x)$.

Byte-level BPE

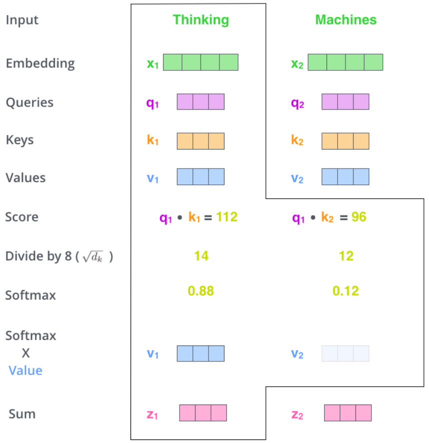
- GPT-2 [18] uses bytes as a base vocabulary instead of including all possible Unicode base characters.
- This forces the base vocabulary to be of size 256 while ensuring that every base character is included in the vocabulary.
- GPT2's tokenizer does not need the $< unk >$ token to tokenize every text.
- The vocabulary size of the GPT-2 is 50,257, which corresponds to the 256 bytes base tokens, a special end-of-text token. The tokenizer learns the symbols with 50,000 merges.

RoBERTa Tokenizer

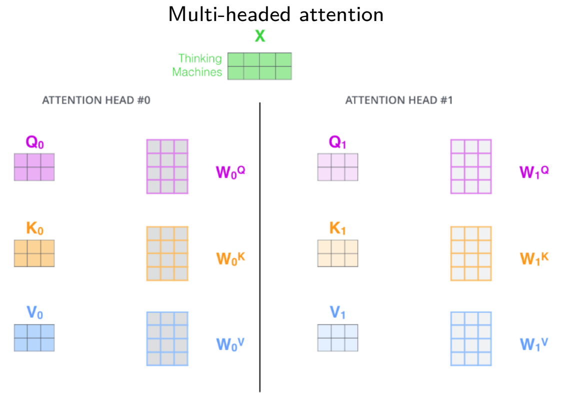
- HuggingFace's "RoBERTaTokenizerFast" is implemented from the GPT-2 tokenizer, using byte-level Byte-Pair-Encoding.
- The tokenizer is trained to treat spaces like parts of the tokens (a bit like SentencePiece) so that a word will be encoded differently independent of the position of the word, i.e., whether the word is at the beginning of the sentence (without space) or not.

BART Tokenizer The HuggingFace BART tokenizer uses byte-level Byte-Pair-Encoding. It is identical to the "RobertaTokenizerFast" tokenizer as discussed before.

Self Attention calculation:

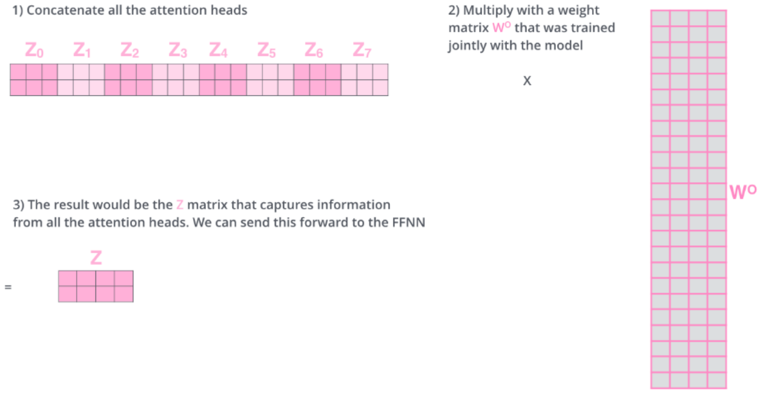


Multi-headed attention:



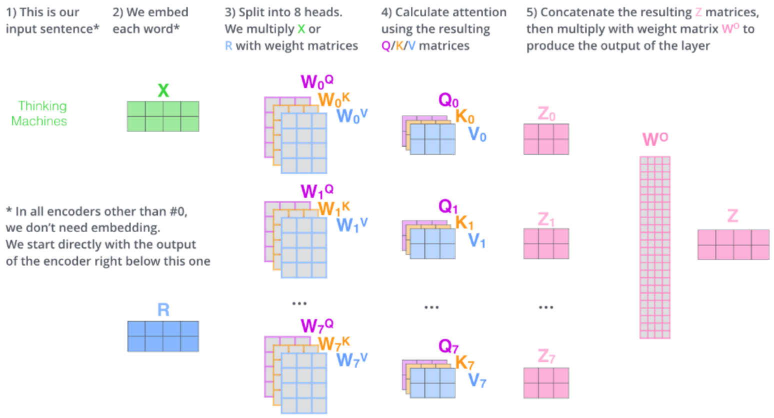
With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.

Multi-headed attention calculation:



1) Concatenate all the attention heads
Z0 Z1 Z2 Z3 Z4 Z5 Z6 Z7
2) Multiply with a weight matrix W^O that was trained jointly with the model
X
W^O
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
=
Z

Multi-head attention in one snapshot


1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0 Z1 Z7
WO
Z
...

How Transformer uses multi-head attention?

The Transformer uses multi-head attention in three different ways:

- In the "encoder-decoder attention" layers, the memory keys and the values come from the output of the encoder, and the queries come from the previous decoder layer. This allows every position in the decoder to attend all positions in the input sequence.
- The self-attention layers in the encoder allow each position in the encoder to attend to all positions in the previous layer of the encoder. In a self-attention layer, all of the queries, keys, values, come from the output of the previous layer in the encoder.
- Each position in the decoder can attend to all positions in the decoder up to and including that position via self-attention layers. The leftward information flow in the decoder is prevented to preserve the auto-regressive property. Therefore, the authors masked out (setting to $-\infty$) all values in the input of the *softmax* that correspond to illegal connections inside of the scaled dot-product attention as shown in **??**.