Neural Code Synthesis and Completion

Presented by: Mitodru Niyogi[1]

Supervised by: Prof. Dr. Artur Andrzejak[1] and Dr. Dejan Kovachev[2]

[1]Interdisciplinary Center for Scientific Computing,
Heidelberg University

[2]SAP Berlin
mitodru.niyogi@stud.uni-heidelberg.de

August 25, 2021

Outline

Introduction

Can AI write code? YES!

- The use of natural language (NL) by a developer to express coding intention and get it translated into code segments, is an interesting problem that,
- if solved, can 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.

Goal

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 neural-based subword tokenizers 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.
- Each block uses A self-attention heads and hidden dimension H.

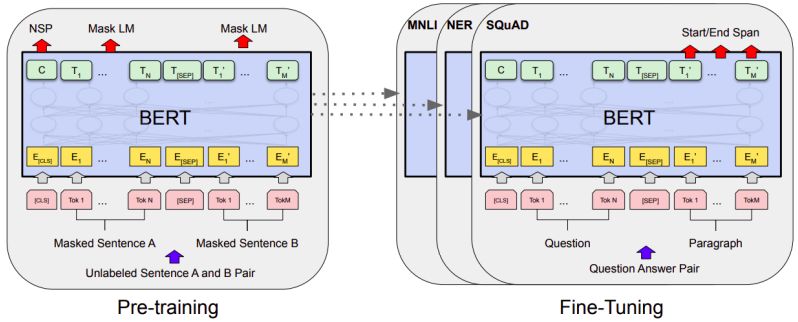
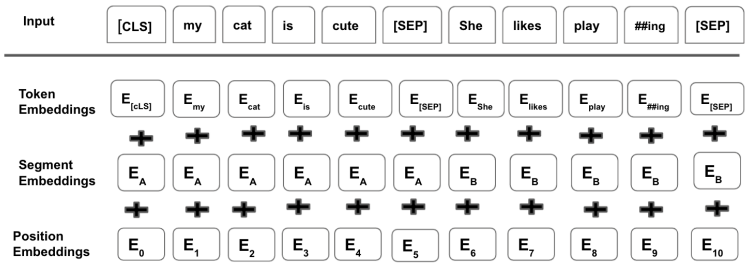


Figure: Overall pre-training and fine-tuning procedures for BERT. Figure taken from [1]



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
- dynamic masking pattern applied to the training data
- dynamic masking is where the masking pattern was generated every time for every input sequence to avoid using the same mask for each training instance in every epoch.
- Specifically, RoBERTa is trained with dynamic masking, FULL-SENTENCES without NSP loss pretraining objective, and large mini-batches.
- FULL-SENTENCES: each input is packed with full sentences sampled contiguously from one or more documents, such that the total length is at most 512 tokens.

BART

- 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.

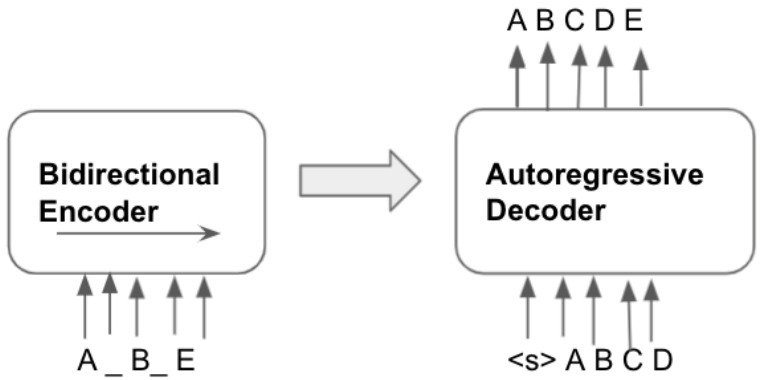


Figure: A schematic representation of BART. Figure drawn from [3].

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.

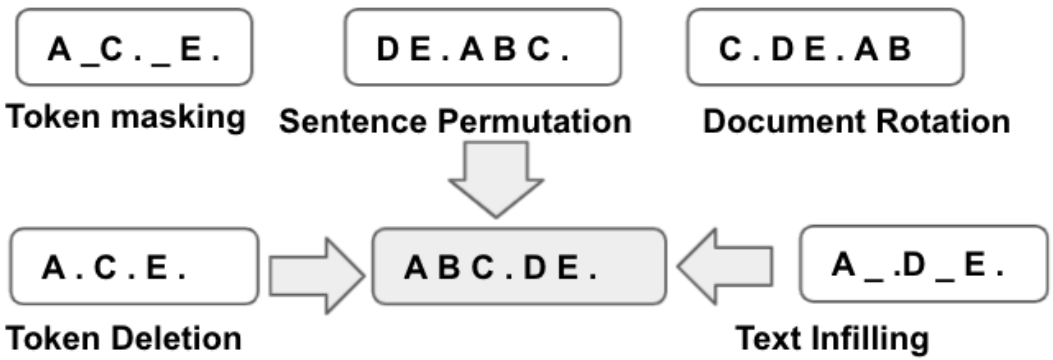


Figure: Transformations as part of Pre-training objectives for noising the input in BART. Figure drawn from [3].

Evaluation Metrics

BLEU

- BLEU (bilingual evaluation understudy) [6] 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

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

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

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

ROUGE

- ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metric [4] measures the n-gram overlap between generated translation and its reference translation.
- **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.

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.
- **How?** We first trained a model for the Code2NL objective and using this model, we generated the predicted natural language intent from the code snippets for both the training and the validation set
- 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.

Augmented curated datasets using back-translation:
Top-1 from intent and top-1 from rewritten intent resulted into $3x$ dataset.
Top-2 from intent and top-2 from rewritten intent resulted into $5x$ dataset.
Top-1 from rewritten intent resulted into $2x$ dataset.
Top-2 from rewritten intent resulted into $3x$ dataset.

Proposed System Architecture

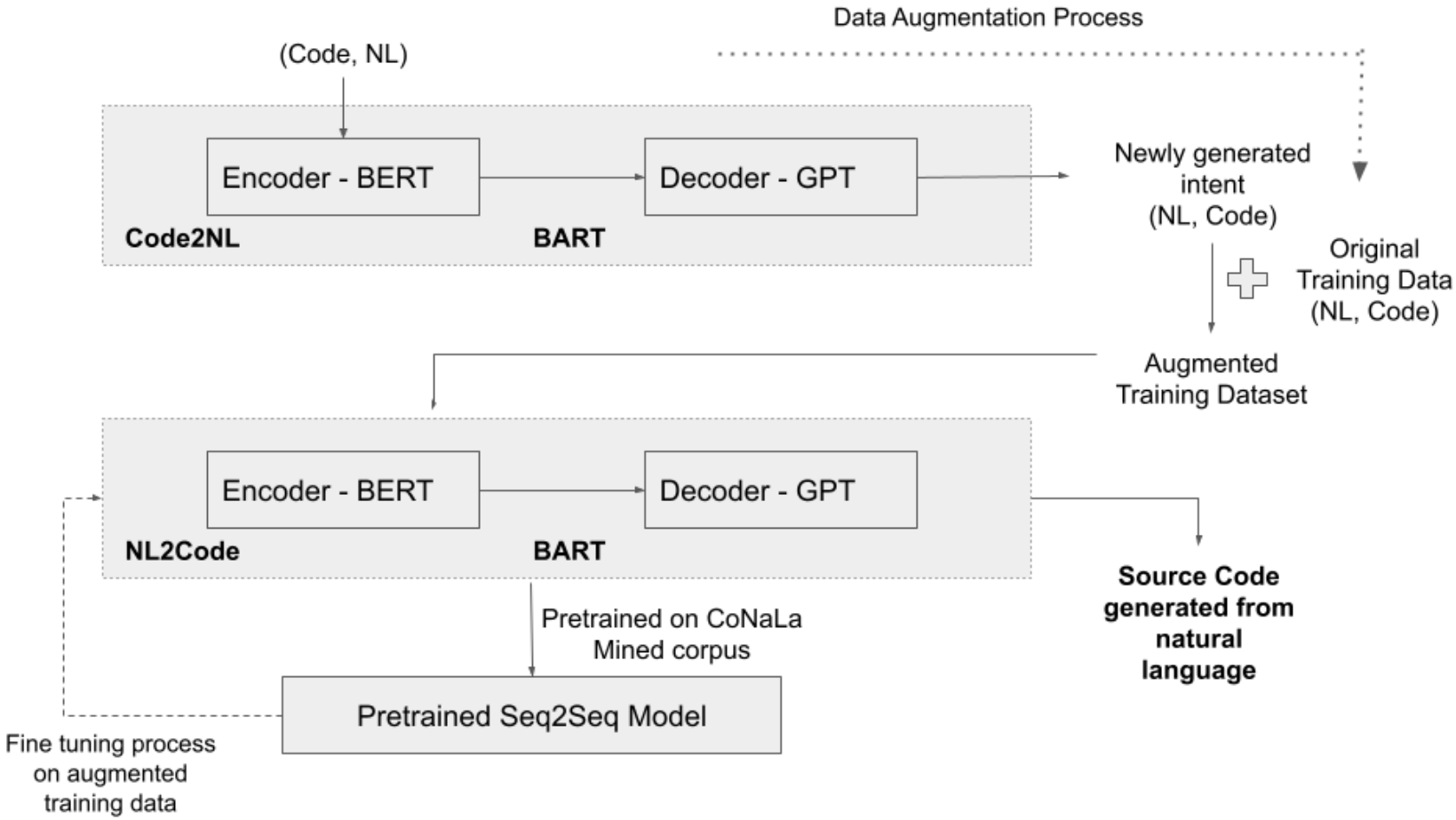


Figure: Proposed System Architecture

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 Compilable Snippets
Fine-tuned Seq2Seq-BART	23	66	101	142	326
Seq2Seq-BART w/o pretraining	16	54	110	141	340
Fine-tuned Seq2Seq-RoBERTa	5	35	89	148	425
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

Table: Comparison on classifying the translations into categories.

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': 'application",	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'), ensure_ascii","json.dumps(decimal('3.9'))"]	json.dumps(Decimal('3.9'))
Check if object 'obj' is a string.	['isinstance(obj, string)', 'isinstance(obj, str)', 'isinstance(obj, bool)']	['isinstance(obj, str)', 'if isinstance(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 myDictionary.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 myDict.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.

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')", "os.path.write('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]

Hinweise und FAQs für BSc+MSc Abschlussarbeiten-2.pdf

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

RQ2. 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.

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.
- 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.

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.

Do pretraining knowledge and transfer learning help to improve the results?

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

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

- The fine-tuning 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	METEOR
21.80291	0.45516	0.21644	0.407032	0.3050

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 resulted in generating better code translations from NL and has exceeded the performance of the vanilla Seq2Seq, Transformer, the proposed Seq2Seq-RoBERTa architectures, and the state-of-the-art neural semantic parser, TranX on BLEU-4 metric score 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 [5], WikiSQL [12] to see how well the proposed architecture generalizes to other programming languages and language-specific domains.
- Incorporate ASTs into our proposed architecture.
- Evaluate the model using BERTSCORE [11] as an evaluation metric for the translated code snippets.
- Allow the proposed system to explore contextual awareness over multiple lines of source code 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 code.

References I

[1] 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.

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

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

[4] 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 II

[5] 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.

[6] 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.

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

[8] 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.

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

References III

[10] 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.

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

[12] 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.

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?

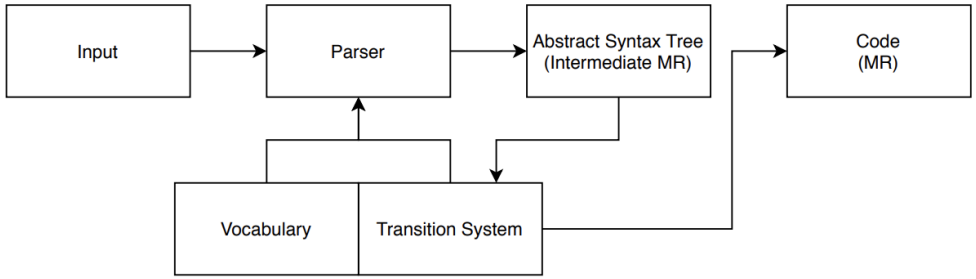


Figure: The overview of the workflow of TranX. Figure drawn from [10]

- TranX [10] 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) [6] 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

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

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

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

- 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 [6] 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 [4] 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 [9], 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 [8]. 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 [2] 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

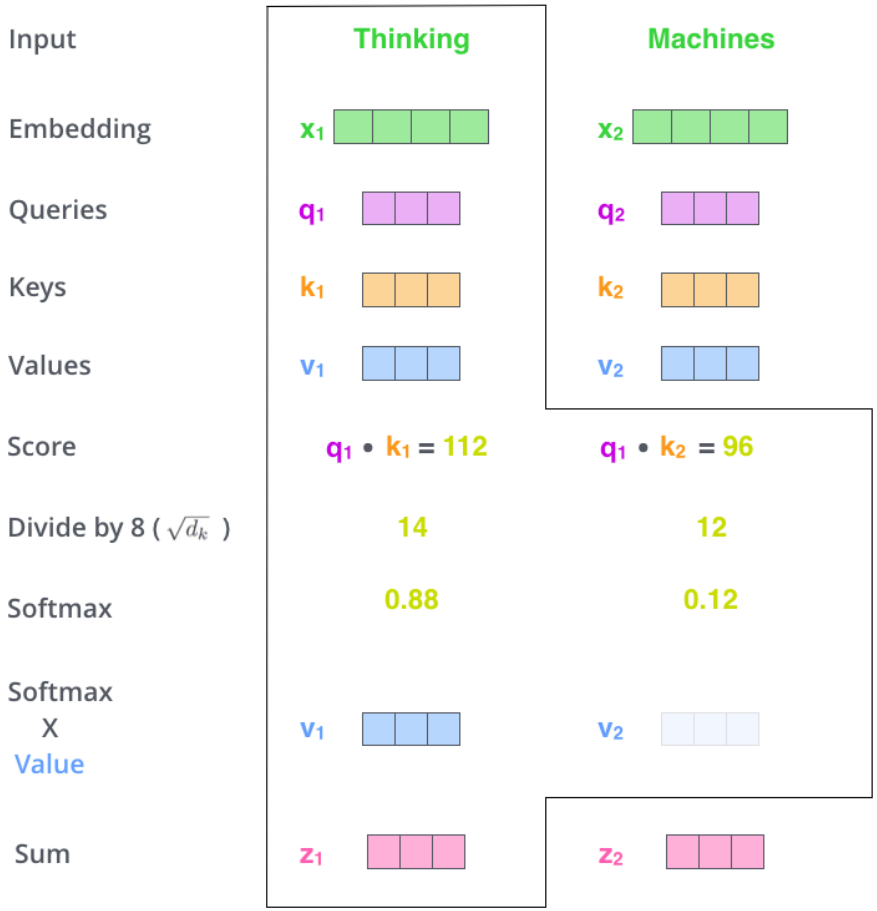
- GPT-2 [7] uses bytes as a base vocabulary instead of including all possible Unicode base characters.
- This forces the base vocabulary to be of size 256 while ensuring that every base character is included in the vocabulary.
- GPT2's tokenizer does not need the $< unk >$ token to tokenize every text.
- 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

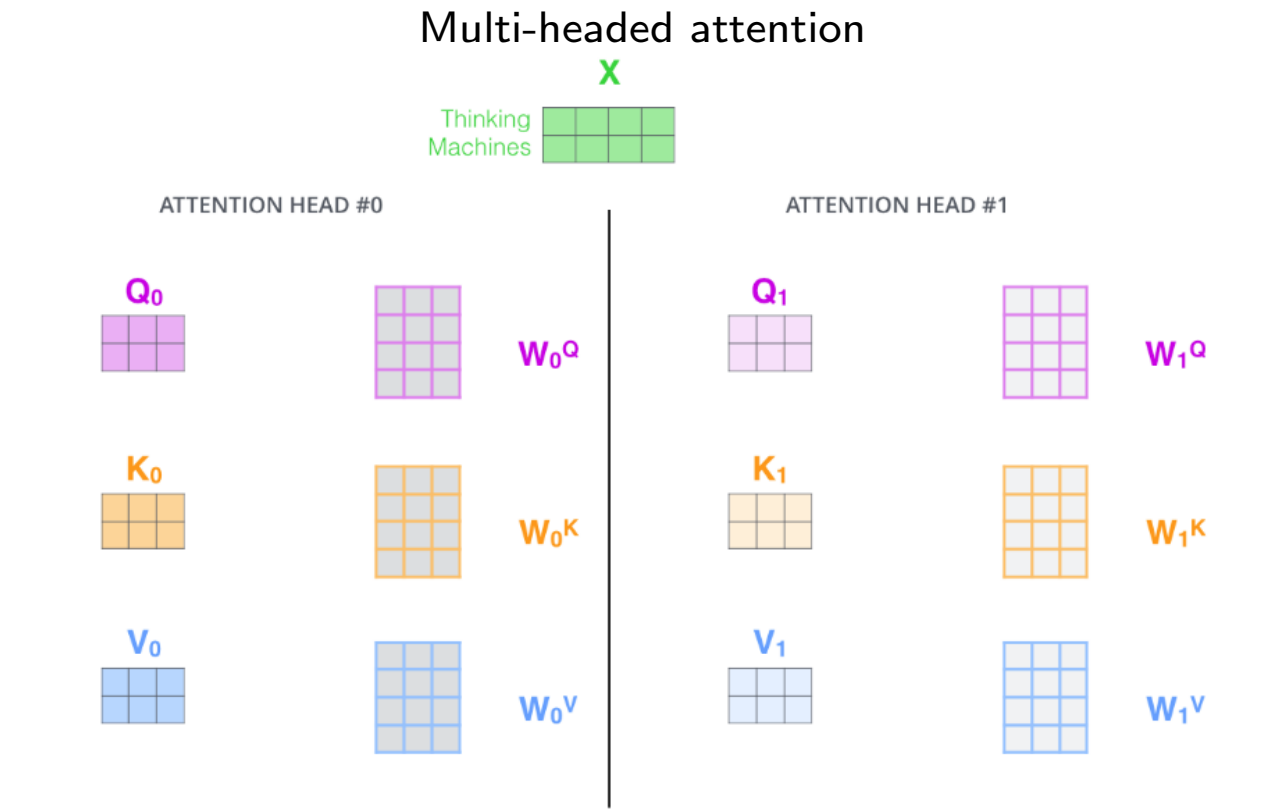
- HuggingFace's "RoBERTaTokenizerFast" is implemented from the GPT-2 tokenizer, using byte-level Byte-Pair-Encoding.
- The tokenizer is trained to treat spaces like parts of the tokens (a bit like SentencePiece) so that a word will be encoded differently independent of the position of the word, i.e., 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:



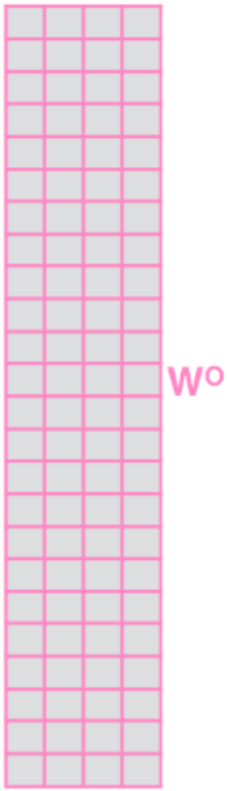
With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.

Multi-headed attention calculation:

1) Concatenate all the attention heads

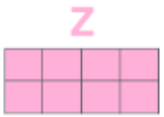
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



Multi-head attention in one snapshot