Using large language models in software engineering

Alex Serban

Radboud University, Software Improvement Group, Leiden University
The Netherlands

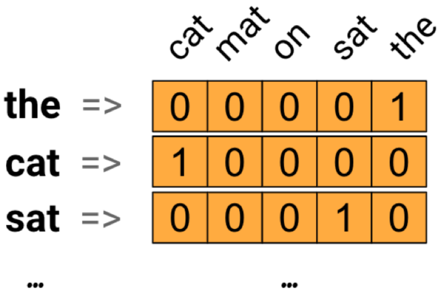
Premises

- Code has all the properties used to analyse natural languages, e.g., form, meaning, context
- Programming languages and code are influenced by social, cultural, historical, and other factors that also influence natural languages
- “Off the shelf” language models are suitable for code tasks
- Text is the simplest level of abstraction

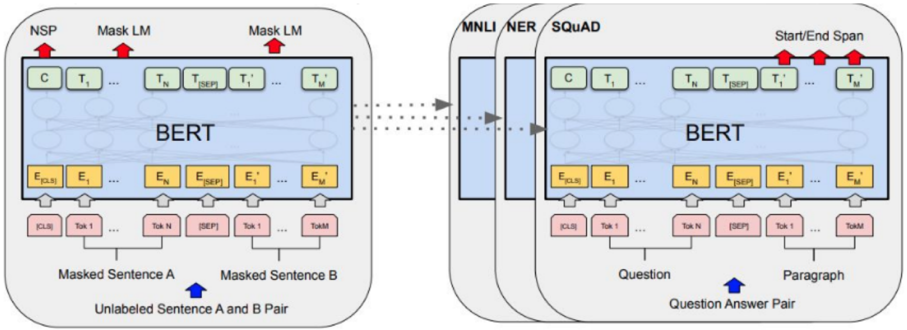
```
(=BA#9"=<;:3y7x54-21q/p-,+*)"!h%B0/.
~P<
<:(8&
66#"!~}|{zyxwvu
gJk
```

Hello world in Malbolge programming language

Large language models



Find a numerical representation of text
(embedding)



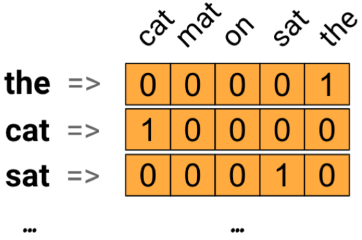
Pretrain a large model on a surrogate task,
where labels can be generated automatically
(self supervised learning)

Fine tune the pretrained model on a (downstream)
task, for which a labeled dataset exists
(supervised learning)

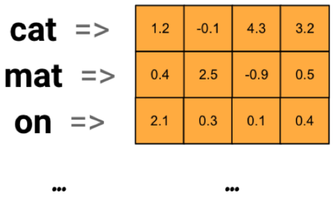
Alex Serban
cs.ru.nl/~aserban

Text as numbers (some examples)

- One hot embedding – inefficient because the vectors are sparsec
- Encode each word with a unique number – the integers assigned to words are arbitrarily
- Both approaches omit *context*
- Learn a representation based on context (embedding)

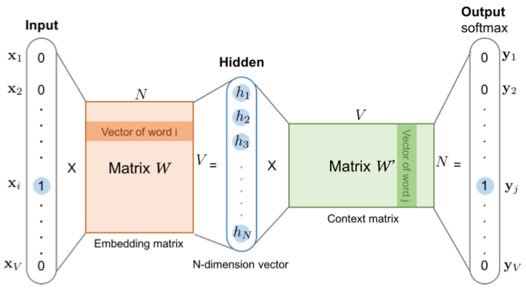


One hot encoding



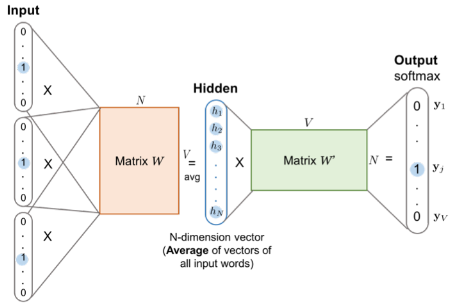
Learned embedding

Text as numbers (Embeddings)



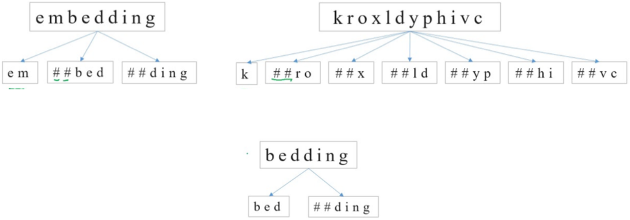
Skip gram

- predicts words within a certain range before and after the word to be represented
- powerful for context representation
- sensibility to rare words



Continuous bag of words

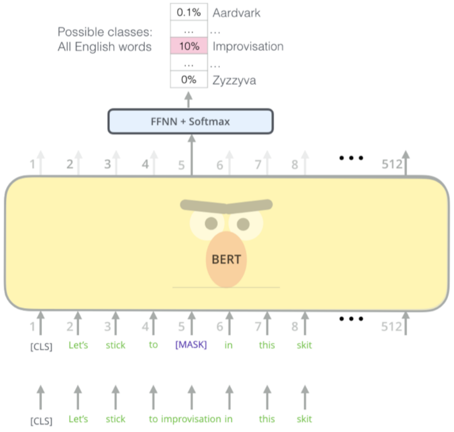
- predicts a middle word from context
- powerful for context representation
- sensibility to rare words



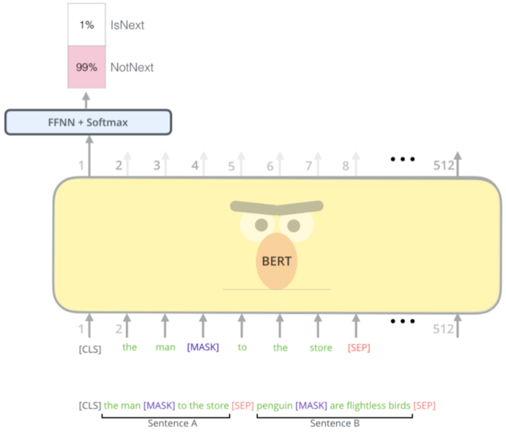
Character based embedding (WordPiece)

- iteratively add word units that increase the likelihood of the trained data
- powerful for rare words

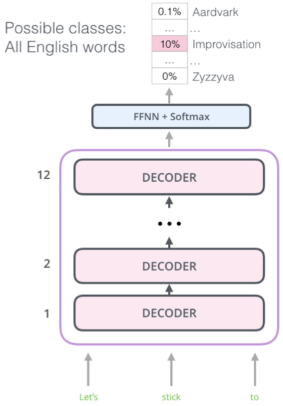
Pretrain a large language model (surrogate tasks)



Masked language modeling

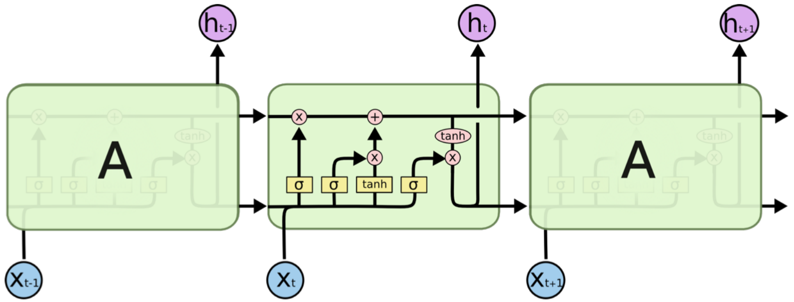


Next sentence prediction



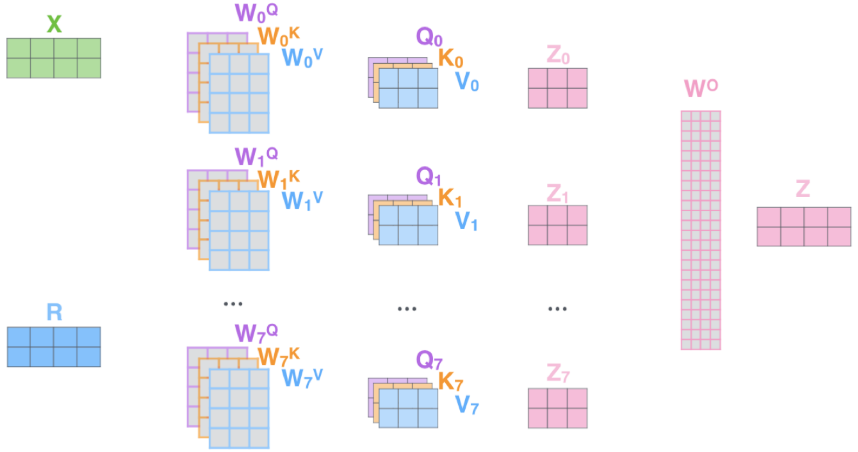
Next word prediction

Pretrain a large language model (building blocks)



Long Short Term Memory (LSTM)

- processes the input word by word
- hard to parallelise

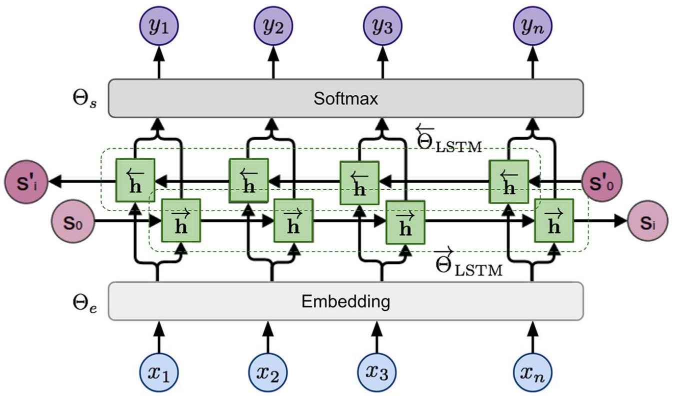


Multihead Attention

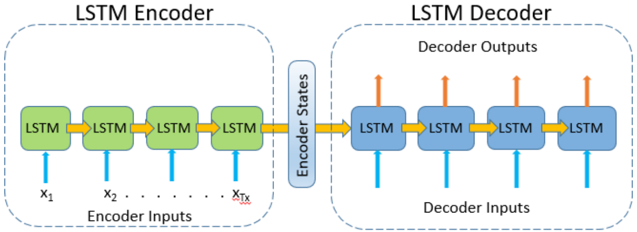
- processes all input words at once
- easy to parallelise

Alex Serban
cs.ru.nl/~aserban

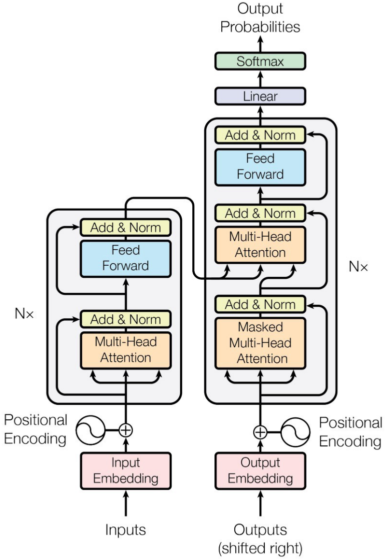
Pretrain a large language model (architectures)



Bidirectional LSTM (ELMO)



Encoder-Decoder (sequence to sequence) LSTM



Transformers

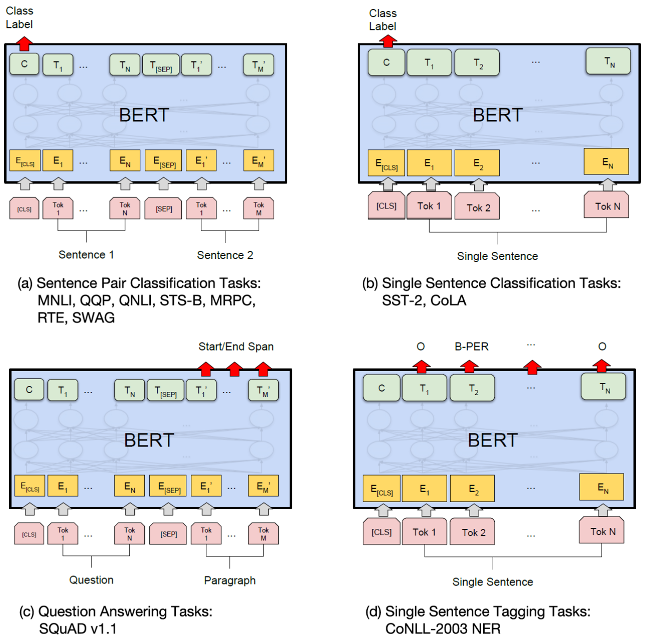
Fine tuning on a downstream task

- After pretraining on a surrogate tasks, language models can be tuned on any NLP task

- Fine tuning requires less resources than training on a surrogate task

- Fine tuning requires less data that training on a surrogate task. However, the data quality is more important (e.g., in terms of labels)

- In some cases (GPT), models are evaluated on downstream tasks in a zero-shot manner



Fine tuning BERT on different tasks

Downstream tasks in software engineering

- Downstream tasks in software engineering are both unimodal (Code-Code, Text-Text) and bimodal (Text-Code, Code-Text)
- The tasks are very distinct in nature, e.g., clone detection, defect detection, code search, code translation, etc.
- Using the WordPiece embeddings makes "off the shelf" language models compatible with SE tasks
- Efforts to create benchmarks similar to NLP are carried out by Microsoft, CodeXGLUE

Category	Task	Dataset Name	Language	Train/Dev/Test Size	Baselines	Task definition
Code-Code	Clone Detection	BigCloneBench	Java	900K/416K/416K	CodeBERT	Predict semantic equivalence for a pair of codes.
		POJ-104	C/C++	32K/8K/12K		Retrieve semantically similar codes.
	Defect Detection	Devign	C	21k/2.7k/2.7k		Identify whether a function is vulnerable.
	Cloze Test	CT-all	Python, Java, PHP, JavaScript, Ruby, Go	-/-/176k		Tokens to be predicted come from the entire vocab.
		CT-max/min	Python, Java, PHP, JavaScript, Ruby, Go	-/-/2.6k		Tokens to be predicted come from {max, min}.
	Code Completion	PY150	Python	100k/5k/50k	CodeGPT	Predict following tokens given contexts of codes.
		GitHub Java Corpus	Java	13k/7k/8k		
	Code Repair	Bugs2Fix	Java	98K/12K/12K	Encoder-Decoder	Automatically refine codes by fixing bugs.
	Code Translation	CodeTrans	Java-C#	10K/0.5K/1K		Translate the codes from one programming language to another programming language.
Text-Code	NL Code Search	CodeSearchNet, AdvTest	Python	251K/9.6K/19K	CodeBERT	Given a natural language query as input, find semantically similar codes.
		CodeSearchNet, WebQueryTest	Python	251K/9.6K/1k		Given a pair of natural language and code, predict whether they are relevant or not.
	Text-to-Code Generation	CONCODE	Java	100K/2K/2K	CodeGPT	Given a natural language docstring/comment as input, generate a code.
Code-Text	Code Summarization	CodeSearchNet	Python, Java, PHP, JavaScript, Ruby, Go	908K/45K/53K	Encoder-Decoder	Given a code, generate its natural language docstring/comment.
Text-Text	Documentation Translation	Microsoft Docs	English-Latvian/Danish/Norwegian/Chinese	156K/4K/4K		Translate code documentation between human languages (e.g. En-Zh), intended to test low-resource multi-lingual translation.

Examples of downstream task from CodeXGLUE

Training language models for software engineering

- In order to better represent both modalities (text and code), we can train language models for SE tasks

- The same methods to define surrogate tasks can be used

- In practice, the masked language model, the replaced toked detection and next token prediction are used

- Efforts to create pretrained language models are carried out by Microsoft, resulting in two models: CodedBert and CodeGPT

```
def _parse_memory(s):
    """
    Parse a memory string in the format supported by Java (e.g. 1g, 200m) and
    return the value in MiB

    >>> _parse_memory("256m")
    256
    >>> _parse_memory("2g")
    2048
    """
    units = {'g': 1024, 'm': 1, 't': 1 << 20, 'k': 1.0 / 1024}
    if s[-1].lower() not in units:
        raise ValueError("invalid format: " + s)
    return int(float(s[:-1]) * units[s[-1].lower()])
```

An example of natural language – programming language task

Results of finetuned pretrained code models

Table 5: Results on the clone detection task.

	BigCloneBench	POJ-104	
Model	F1	MAP	Overall
RtvNN	1.0	-	-
Deckard	3.0	-	-
CDLH	82.0	-	-
ASTNN	93.0	-	-
FA-AST-GMN	95.0	-	-
TBCCD	95.0	-	-
code2vec*	-	1.98	-
NCC*	-	54.19	-
Aroma*	-	55.12	-
MISIM-GNN*	-	82.45	-
RoBERTa	94.9	79.96	87.4
CodeBERT	**96.5**	**84.29**	**90.4**

Table 7: Results on the defect detection task.

Model	Accuracy
BiLSTM	59.37
TextCNN	60.69
RoBERTa	61.05
CodeBERT	**62.08**

Table 8: Results on the code completion task.

Model	PY150			Github Java Corpus			Overall
	token-level	line-level		token-level	line-level		
	Accuracy	EM	Edit Sim	Accuracy	EM	Edit Sim	
LSTM	58.00	17.93	50.05	56.02	10.30	41.55	51.41
Transformer	73.26	36.65	67.51	64.16	15.33	50.39	63.83
GPT-2	74.22	38.55	68.94	74.89	24.30	60.70	69.69
CodeGPT	74.93	39.11	69.69	76.45	25.30	61.54	70.65
CodeGPT-adapted	**75.11**	**39.65**	**69.84**	**77.13**	**26.43**	**63.03**	**71.28**

Table 11: Results on the code repair task.

Method	small			medium			Overall
	BLEU	Acc	CodeBLEU	BLEU	Acc	CodeBLEU	
Naive	**78.06**	0.000	-	90.91	0.000	-	0.000
LSTM	76.76	0.100	-	72.08	0.025	-	0.063
Transformer	77.21	0.147	73.31	89.25	0.037	81.72	0.092
CodeBERT	77.42	**0.164**	**75.58**	**91.07**	**0.052**	**87.52**	**0.108**

Table 9: Results on the code search task.

	AdvTest	WebQueryTest		
Model	MRR	F1	Accuracy	Overall
RoBERTa	18.33	57.49	40.92	33.63
CodeBERT	**27.19**	**58.95**	**47.80**	**40.28**

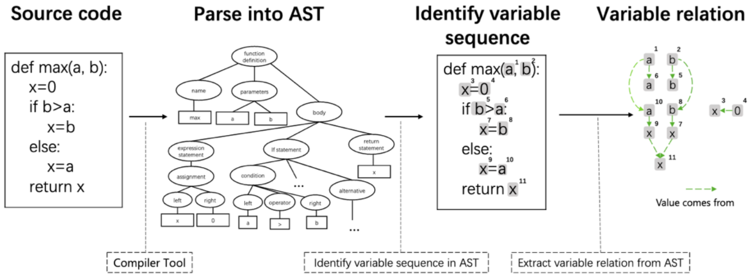
Table 10: Results on the text-to-code generation task.

Model	EM	BLEU	CodeBLEU
Seq2Seq	3.05	21.31	17.61
Seq2Action+MAML	10.05	24.40	20.99
Iyer-Simp+200 idoms	12.20	26.60	-
GPT-2	17.35	25.37	22.79
CodeGPT	18.25	28.69	25.69
CodeGPT-adapted	**20.10**	**32.79**	**27.74**

Data from CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation

Adding more abstractions

- Data flow can be added in training
- Training is done on pairs of source code, comments and data flow
- Data flow tasks such as edge prediction or node alignment are used in pretraining
- The improvements are significant, but marginal



model	Ruby	Javascript	Go	Python	Java	Php	Overall
NBow	0.162	0.157	0.330	0.161	0.171	0.152	0.189
CNN	0.276	0.224	0.680	0.242	0.263	0.260	0.324
BiRNN	0.213	0.193	0.688	0.290	0.304	0.338	0.338
selfAtt	0.275	0.287	0.723	0.398	0.404	0.426	0.419
RoBERTa	0.587	0.517	0.850	0.587	0.599	0.560	0.617
RoBERTa (code)	0.628	0.562	0.859	0.610	0.620	0.579	0.643
CodeBERT	0.679	0.620	0.882	0.672	0.676	0.628	0.693
GraphCodeBERT	**0.703**	**0.644**	**0.897**	**0.692**	**0.691**	**0.649**	**0.713**

Table 1: Results on code search. GraphCodeBERT outperforms other models significantly ($p < 0.01$).

Images from GraphCodeBert: Pretraining code representations with data flow

Resources

- Language models consist of over 100 million parameters

- However, the fine tuning times are in the order of hours, not days (or weeks) given appropriate hardware

- The inference times are reasonable

Table 4: Parameters of CodeBERT and CodeGPT models.

	CodeBERT	CodeGPT
Number of layers	12	12
Max length of position	512	1,024
Embedding size	768	768
Attention heads	12	12
Attention head size	64	64
Vocabulary size	50,265	50,000
Total number of parameters	125M	124M

Task	Dataset Name	Language	Training Cost	Inference Cost
Clone Detection	BigCloneBench	Java	3 hours training on P100 x2	2 hours on p100 x2
	POJ-104	C/C++	2 hours training on P100 x2	10 minutes on p100 x2
Defect Detection	Devign	C	1 hour on P100 x2	2 minutes on p100 x2
Cloze Test	CT-all	Python, Java, PHP, JavaScript, Ruby, Go	N/A	30 minutes on P100-16G x2
	CT-max/min	Python, Java, PHP, JavaScript, Ruby, Go	N/A	1 minute on P100-16G x2
Code Completion	PY150	Python	25 hours on P100 x2	30 minutes on P100 x2
	GitHub Java Corpus	Java	2 hours on P100 x2	10 minutes on P100 x2
Code Repair	Bugs2Fix	Java	24 hours on P100 x2	20 minutes on P100 x2
Code Translation	CodeTrans	Java-C#	20 hours on P100 x2	5 minutes on P100 x2
NL Code Search	CodeSearchnet, AdvTest	Python	5 hours on P100 x2	7 minutes on p100 x2
	CodeSearchNet, WebQueryTest	Python	5 hours on P100 x2	1 minute on P100 x2
Text-to-Code Generation	CONCODE	Java	30 hours on P100 x2	20 minutes on P100 x2
Code Summarization	CodeSearchNet	Python, Java, PHP, JavaScript, Ruby, Go	On average, 12 hours for each PL on P100 x2	On average, 1 hour for each PL on p100 x2
Documentation Translation	Microsoft Docs	English-Latvian/Danish/Norwegian/Chinese	30 hours on P100x2	55 minutes on P100x2

Figure 8: Training and inference time costs for each task, evaluated on two P100 GPUs.

Data from CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation

Reducing the resource footprint

- Because language models consist of a large number of parameters, they have inherent redundancy
- One way to remove this redundancy is to iteratively prune small parameters
- Early results show more that 50% of the parameters are redundant

TABLE I
CLONE DETECTION

		BigCloneBench	POJ-104	
Model	Prune Rate	F1	MAP	Overall
CodeBert	0	96.50	84.29	90.39
CodeBert	90	96.72	83.92	90.32

TABLE II
CODE SEARCH

		AdvTest	WebQueryTest	
Model	Prune Rate	MRR	F1	Accuracy
CodeBert	0	27.19	58.95	47.80
CodeBert	60	26.2	57.62	46.7

Conclusions

- Language models show promising results on software engineering tasks, in particular to unimodal tasks
- Language models trained on code tasks preserve properties of text based language models (e.g., redundancy)
- Adding more abstractions to these models is a promising research avenue
- I think it is interesting to see how these models perform on less curated data

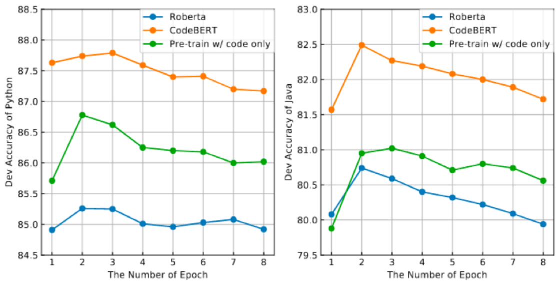


Figure 4: Learning curve of different pre-trained models in the fine-tuning step. We show results on Python and Java.