M.Sc. Thesis Defense

08.04.2019

LEARNING REPRESENTATIONS OF SOURCE CODE FROM STRUCTURE & CONTEXT

by Dylan Bourgeois

Supervised by

EPFL

Pr. Pierre Vandergheynst
Michaël Defferrard

Stanford University

Pr. Jure Leskovec
Dr. Michele Catasta

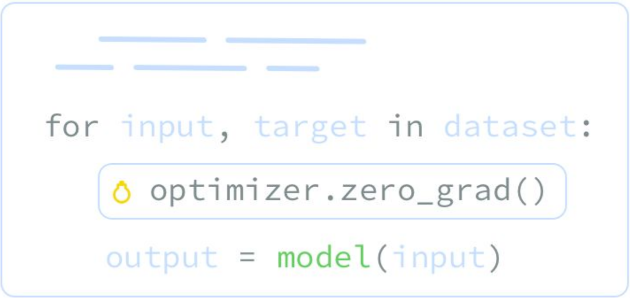
1 Introduction

Capturing similarities of source code

Programming languages offer a unified interface, which is leveraged by programmers. The regularities in coding patterns can be used as a proxy for semantics.

Example applications

- Code recommendation
- Plagiarism detection
- Smarter development tools
- Error correction
- Smart search



Software is ubiquitous

Programming is a human endeavour. It is an intricate process, often repetitive, time-consuming and error-prone.

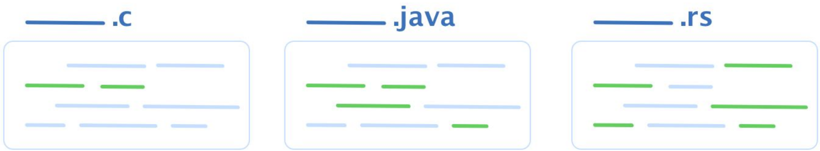
```
def forward(self, x):
        x1 = F.relu(self.conv1(x))
        x1 = F.max_pool2d(x1, 2, 2)
        # ctrl-c // ctrl-v
        x2 = F.relu(self.conv2(x))
        x2 = F.max_pool2d(x2, 2, 2)
        return F.log_softmax(x2, dim=1)
```

Should be x1!

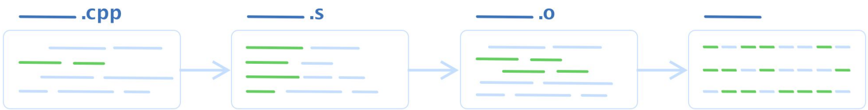
Software is multimodal

The idiosyncrasies of source code are not trivial to deal with. Software is also inherently composable, reusable and hierarchical, it has side-effects.

Software is multilingual.



It exists through several representations...



and multiple abstractions.

Existing work

Most work has focused on solving specific tasks, less so on capturing rich representations of source code.

1 Heuristic-based

Leveraging the strong logic encoded by PL to create formal verification tools, memory safety checkers, ...

2 Contextual regularities

Capturing common patterns in the input representation, typically used in code editors.

Our approach

We propose a ***hybrid*** approach, which leverages both **heuristics** and **regularities**.

Specifically, we hypothesise that **structure** is an informative heuristic.

HEURISTICS (STRUCTURE)

We provide evidence for the importance of leveraging structure in the representation of source code.

REGULARITIES (CONTEXT)

We show that patterns in the input provide a decent signal.

HYBRID (OURS)

We propose a model which learns to recognize both structural and lexical patterns.

2 Code: a structured language with natural properties

[Shannon, 1950, Harris, 1954, Deerwester et al, 1990, Bengio et al. 2003, Collobert and Weston, 2008]

Capturing the regularities of language

A Language Model (LM) defines a probability distribution over sequences of words:

$$p(t) = p(w_1 ... w_n)$$

This probability is estimated from a corpus, and can be parameterized through different forms:

- n-gram $P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$
- Bidirectional / Bi-linear
- Neural Network $P(w_i | \text{context}) \, \forall \, i \in V$

[Hindle et al., 2012]

On the *naturalness* of software

Source code starts out as **text**: as such it can present the same kind of regularities as **natural language**.

Its restricted vocabulary, strong grammatical rules and composability properties encourage regularity and hence predictability.

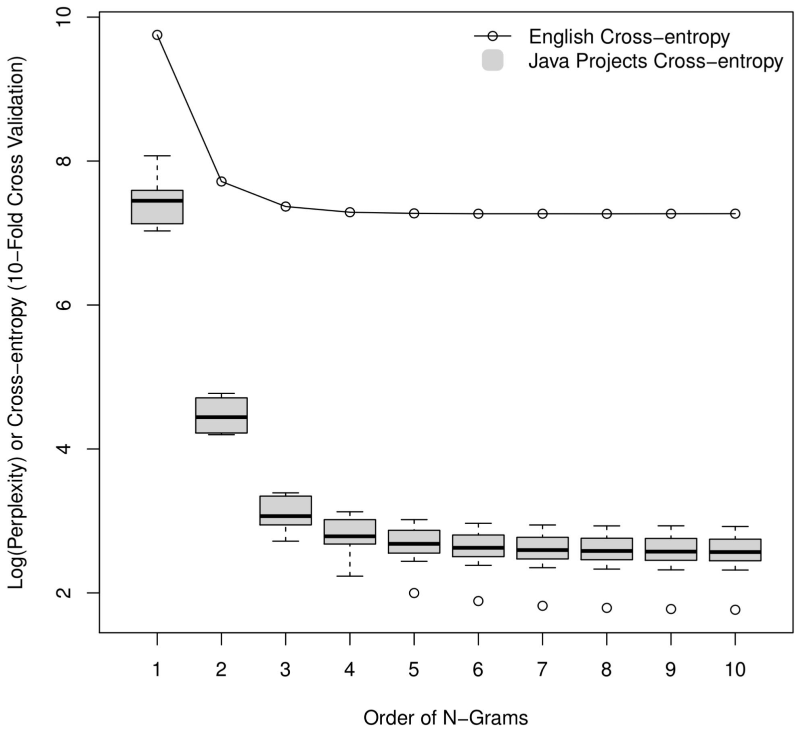


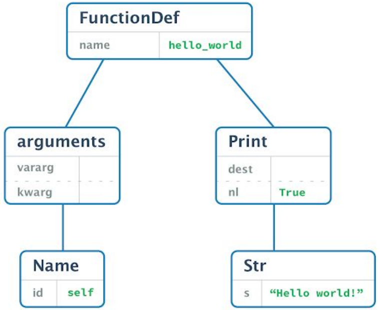
Figure 1. Comparison of English cross-entropy versus the code cross-entropy of 10 Java projects.

Representations of source code

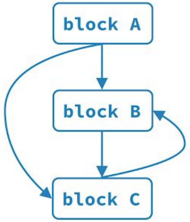
Each representation has inherent properties and abstraction levels associated to it.

```
def hello_world():
    print("Hello world!")
```

1. RAW CODE



2. AST



3. CFG

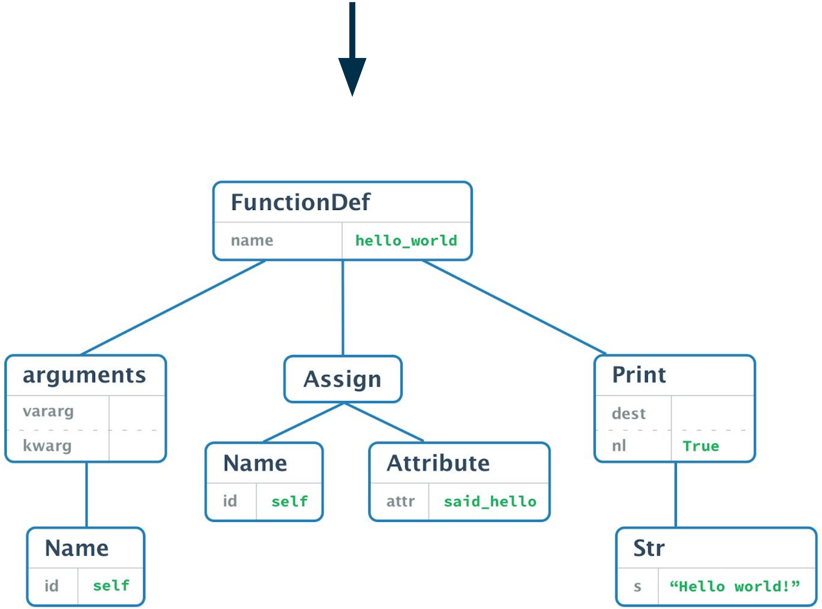
```
0 LOAD_GLOBAL     0  print
2 LOAD_CONST      1  ("Hello world!")
4 CALL_FUNCTION   1
```

4. ByteCode

Code represented as a structured language

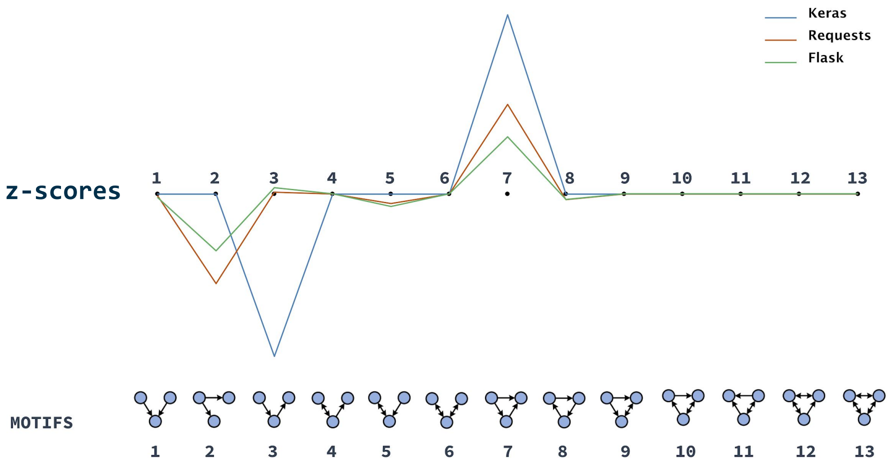
The Abstract Syntax Tree (AST) provides a universally-available, deterministic and rich structural representation of source code.

```
def hello_world(self):
    self.said_hello = True
    print("Hello world!")
```



The regularities of structured representations

Similar to what was found by [Hindle et al., 2012] on free-form text, we see both common patterns *(e.g. motif #7)* and project specific patterns *(e.g. motif #3).*

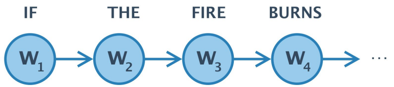


3 Leveraging context and structure in representations of source code

3.1 **Learning from context**

Linear Language Models

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$



Linear Graphical Model
Markov Chain

The n-gram model can be represented as a Markov Chain, simplifying the joint probability by assuming that the likelihood of a word depends only on its history.

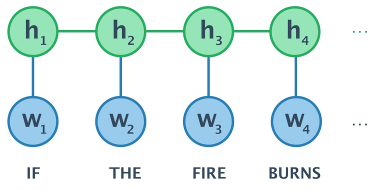
[Mikolov et al., 2013, Peters et al., 2018]

Generalized language models

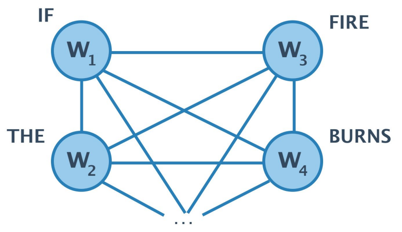
However, in order to integrate more complex models of language, it is necessary to allow more complex models of context.

In order to model polysemy, this context should also modulate the representation of a given word.

$$P(w_i | \text{context}) \forall i \in V$$



Contextual Graphical Model
Markov Random Field (ELMo)

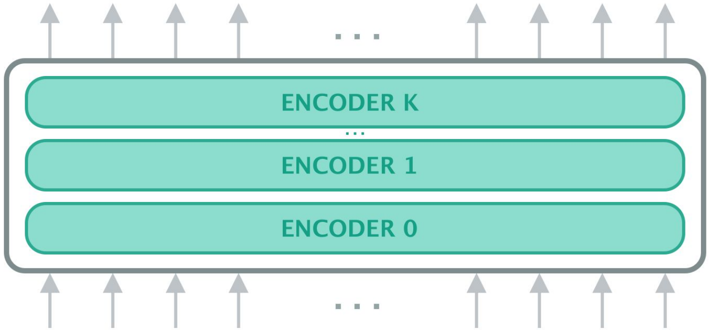


Fully Connected Graphical Model
BERT's Markov Random Field

The Transformer

Many of these insights are captured in the Transformer architecture
[Vaswani et al., 2017].

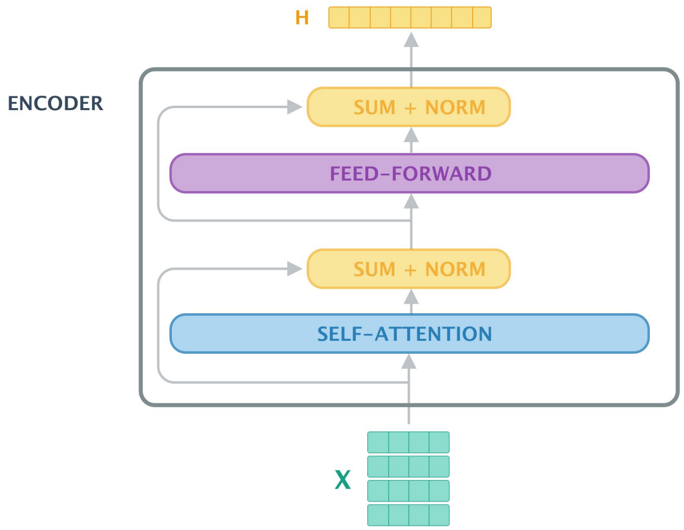
It is a deep, feed-forward, attentive architecture showing strong results compared to recurrent architectures. It is now the building block for most state-of-the-art architectures in NLP.
[Radford et al., 2018, Devlin et al. 2018]



[Vaswani et al., 2017]

The Transformer

The encoder embeds input sequences. Several of these blocks are then stacked to create deeper representations.

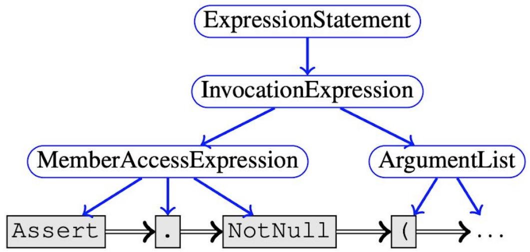


3.2 **Learning from structure**

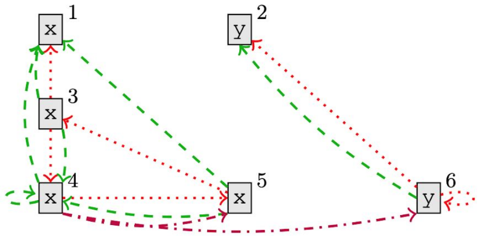
[Allamanis et al., 2018]

Leveraging structured representations of code

Recent work has built on the powerful Graph Neural Networks, running on semantically augmented representations.



(a) Simplified syntax graph for line 2 of Fig. 1, where blue rounded boxes are syntax nodes, black rectangular boxes syntax tokens, blue edges Child edges and double black edges NextToken edges.



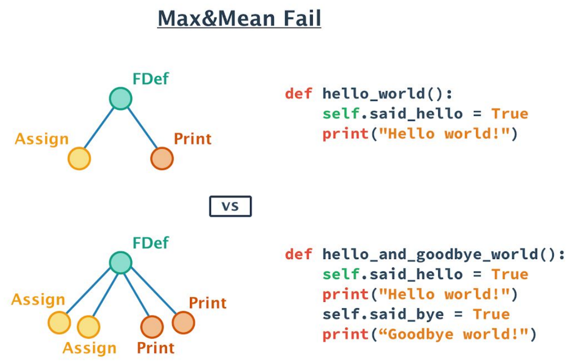
(b) Data flow edges for `(x¹,y²) = Foo(); while (x³ > 0) x⁴ = x⁵ + y⁶` (indices added for clarity), with red dotted LastUse edges, green dashed LastWrite edges and dashdotted purple ComputedFrom edges.

Limitations of the approach

Unfortunately, we found the purely structural approach to have limited results.

INSIGHTS

- A limited vocabulary means contexts are averaged across too many usages to be semantically meaningful.
- Learning a representation for each token has the inverse problem: not enough co-occurrences.
- Some aggregators can have issues with common motifs in code [Xu et al, 2019].

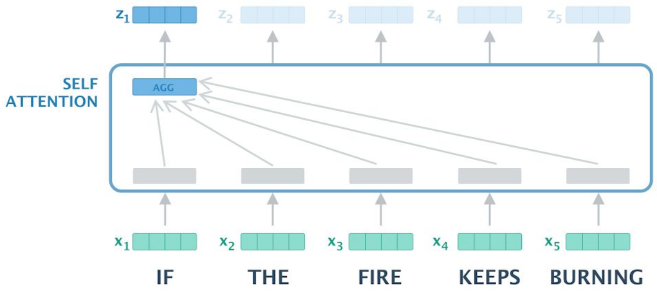


3 Learning from context and structure

INSIGHT

The Transformer: a GNN perspective

No assumptions are made on the underlying structure: the attention module can attend to all the elements in the sequence.

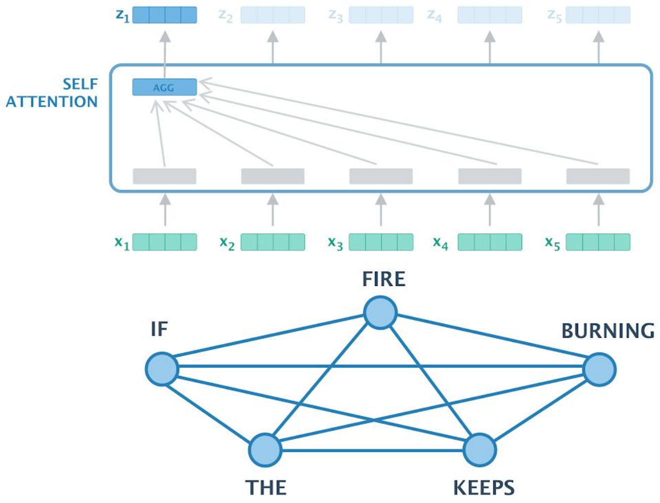


INSIGHT

The Transformer: a GNN perspective

No assumptions are made on the underlying structure: the attention module can attend to all the elements in the sequence.

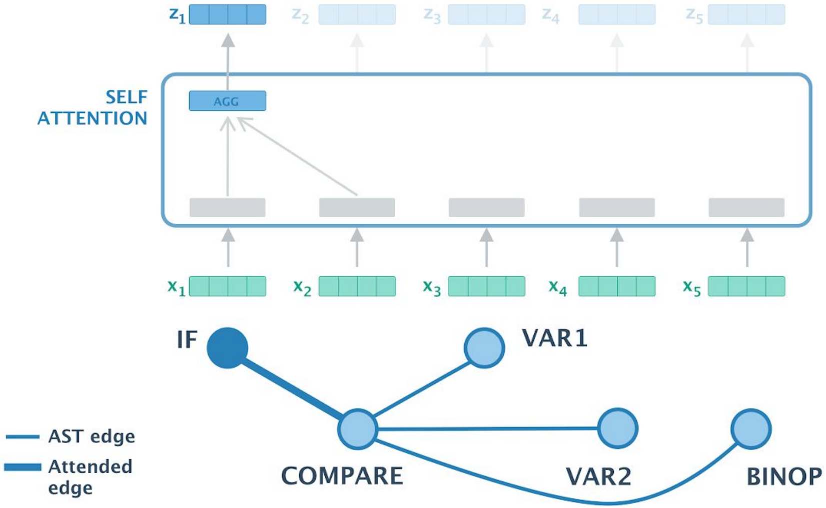
This can be seen as a message-passing GNN on a fully connected input graph.



OUR APPROACH

Generalizing to arbitrarily structured data

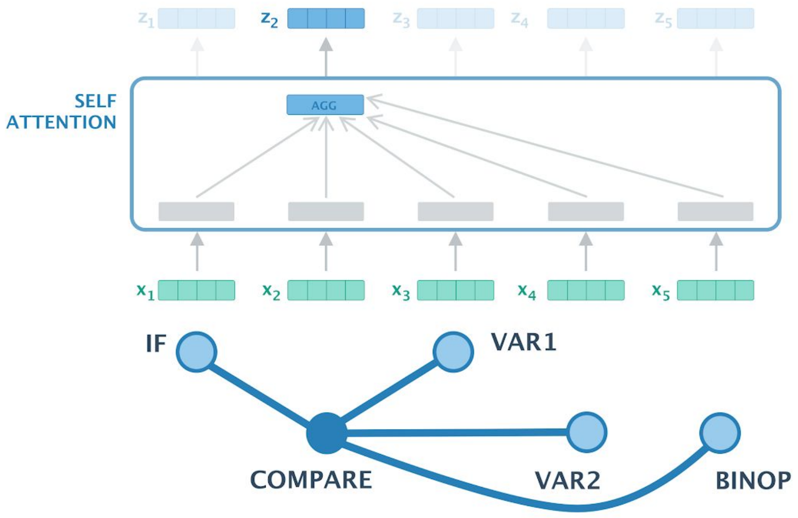
The message-passing edges can be restricted to a priori edges, e.g. syntactic relationships. This enables the treatment of arbitrary graph structures as input.



OUR APPROACH

Generalizing to arbitrarily structured data

The message-passing edges can be restricted to a priori edges, e.g. syntactic relationships. This enables the treatment of arbitrary graph structures as input.



OUR APPROACH

Generalizing to arbitrarily structured data

The aggregation scheme can be replaced by any message-passing aggregation architecture!

GCN-based aggregation

$$\textsc{Aggregate}_k(u) = \sigma\left(\sum_{v \in \mathcal{N}(u)} \frac{1}{c_{uv}} W^k \cdot \mathbf{h}_u^k\right)$$

GAT-based aggregation

$$\textsc{Aggregate}_k(u) = \sigma\left(\sum_{v \in \mathcal{N}(u)} \alpha_{uv}^k \cdot W^k \cdot \mathbf{h}_u^k\right)$$

where $\alpha_{uv}^k = \textsc{Softmax}\left(\phi(u, v)\right)$

Masked Dot-Product Attention

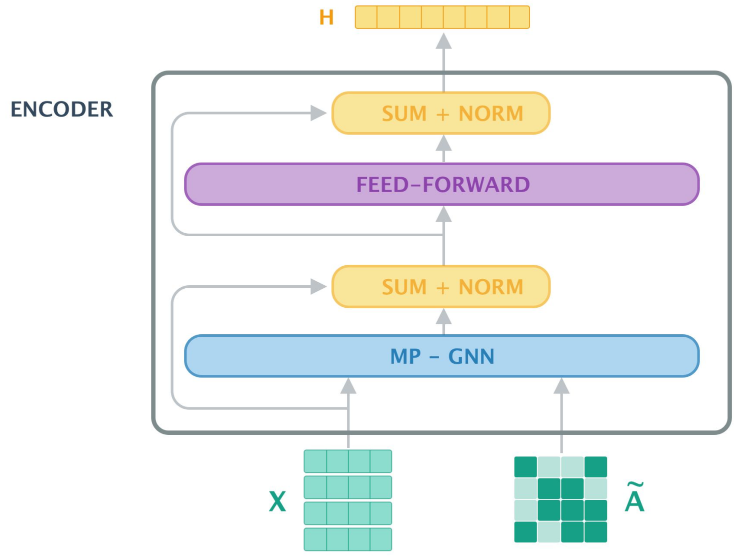
$$\textsc{Aggregate}_k(u) = \sum_{v \in \mathcal{N}(u)}^{N} \textsc{Softmax}\left(\frac{\mathbf{q}_u^k \cdot \mathbf{k}_v^k}{\sqrt{d_k}}\right) \cdot \mathbf{v}_u^k$$

Semantic Aggregation?

OUR APPROACH

Generalizing to arbitrarily structured data

For example, with the masked attention formulation, we can modify a Transformer encoder block to run on arbitrarily structured inputs.

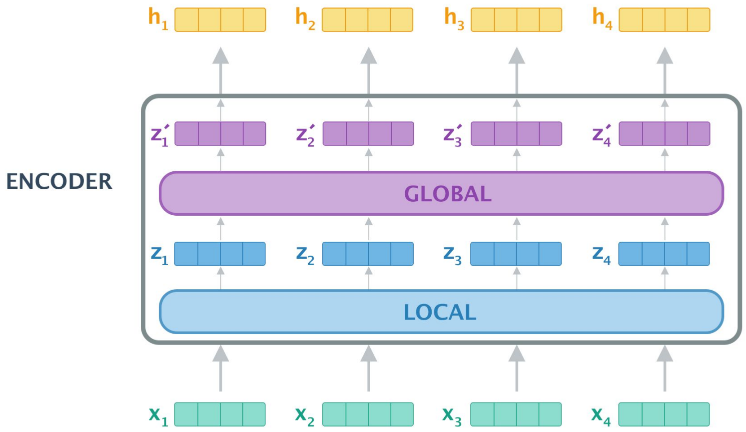


OUR APPROACH

A hybrid approach to aggregating context

With this formulation, we can jointly learn to compose **local** and **global** context, obtaining a deep contextualized node representation.

This helps to learn **structural** and **contextual** regularities.

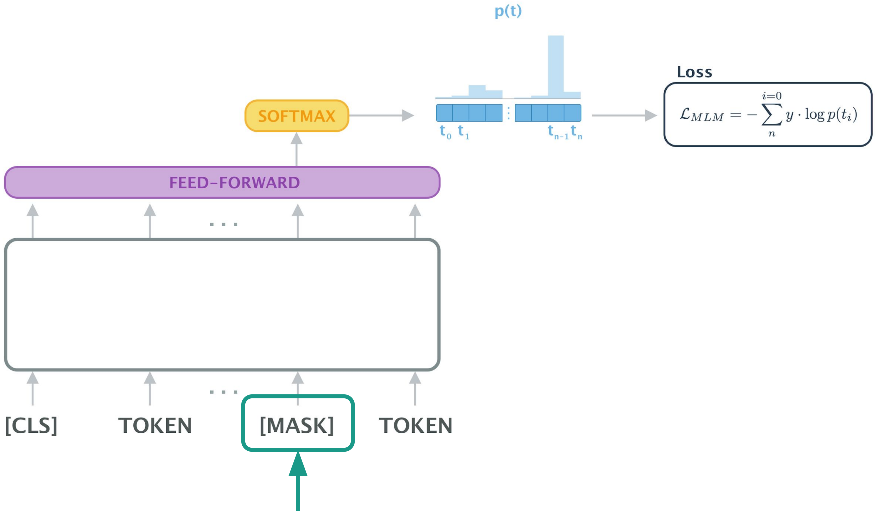


3.4 Learning from context and structure

Model pre-training: a semi-supervised approach

Great success in NLP applications to first model the input data.

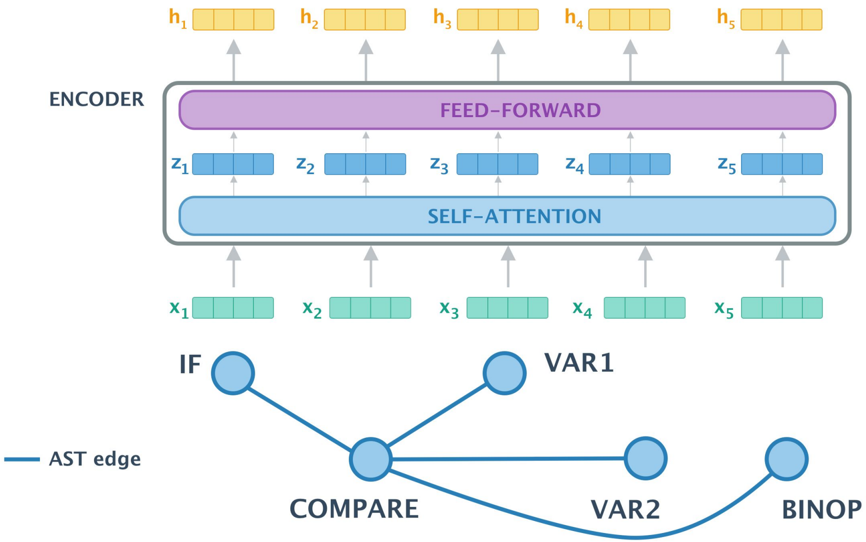
Similar approach to auto-encoders, but only the masked input is reconstructed.



Source code provides abundant training data

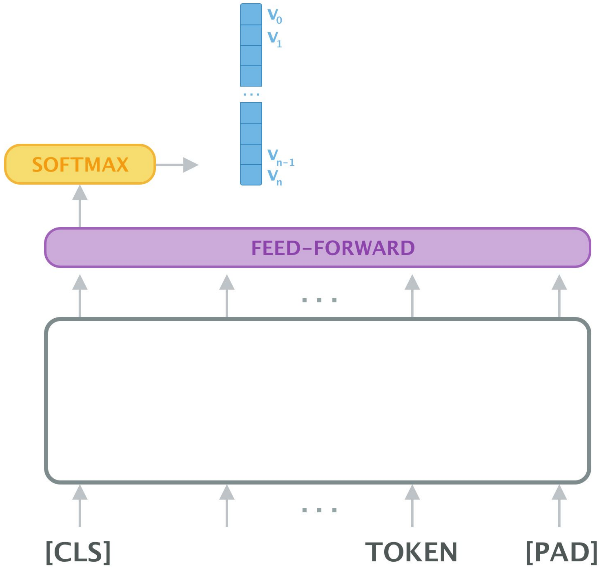
Structure is readily available and deterministic, unlike parse trees of natural language.

The masked language model is similar to a node classification task on graphs.



Transfer learning capabilities

Once the model is pre-trained, it can be fine-tuned to produce labels through a pooling token `[CLS]` or used as a rich feature extractor.



4 Experiments

4.1 **Learning from structure**

Graph-based tasks

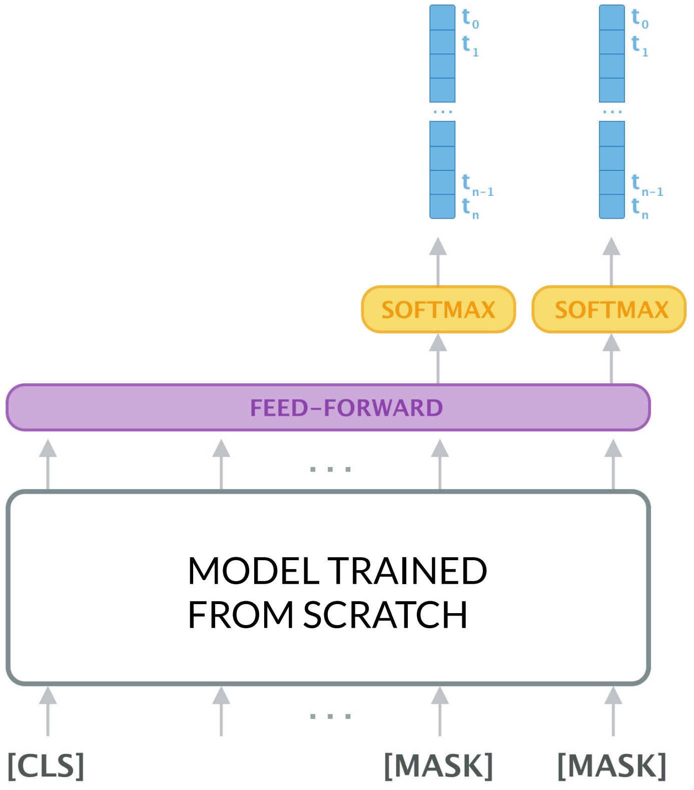
Node classification

The structure is similar to the pre-training task.

	CORA			
	Ours	Freq	L-GCN	GCN
Test acc.	**0.83**†	0.16	**0.83**	0.81

† Label Propagation setting

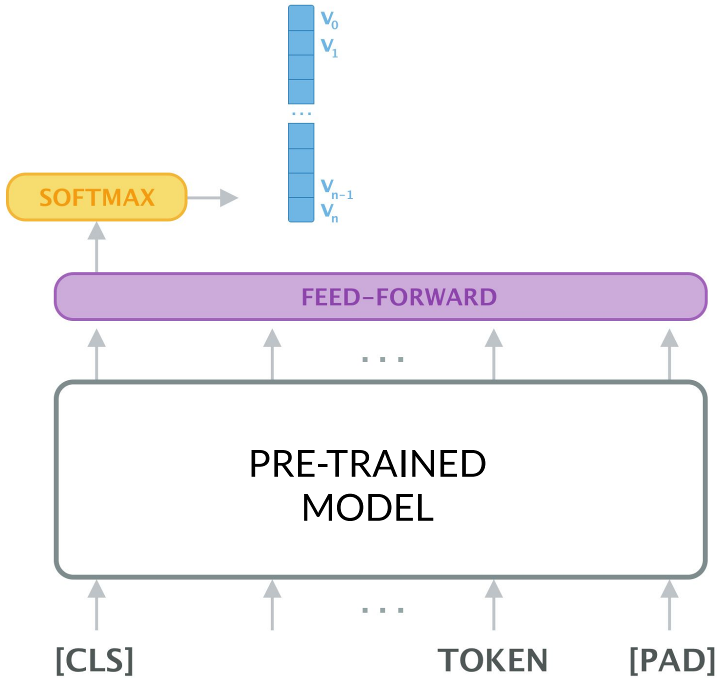
Table 4.10 – Results on node classification



Graph-based tasks

Graph classification

In this case, we use the pooled representation of the input graph to make a prediction.



Graph classification

Our approach is competitive with state-of-the-art results on classic graph classification datasets.

ENZYMES
Predicting one of **6** classes of chemical properties on molecular graphs.

MSRC 21
Predicting one of **21** semantic labels *(e.g. building, grass, …)* on image super-pixel graphs.

MUTAG
Predicting the mutagenicity of chemical compounds (**binary**).

	ENZYMES					
	Ours	Freq	GCN	GraphSAGE	DiffPool	WL
Test Acc.	**0.68**	0.16	0.64	0.54	0.62	0.53

	MSRC-21		
	Ours	Freq	GCN
Test Acc.	0.90	0.05	**0.92**
@3	**1.0**	0.15	-

	MUTAG				
	Ours	Freq	GCN	DGCNN	WL
Test Acc.	0.81	0.55	0.76	**0.85**	0.80

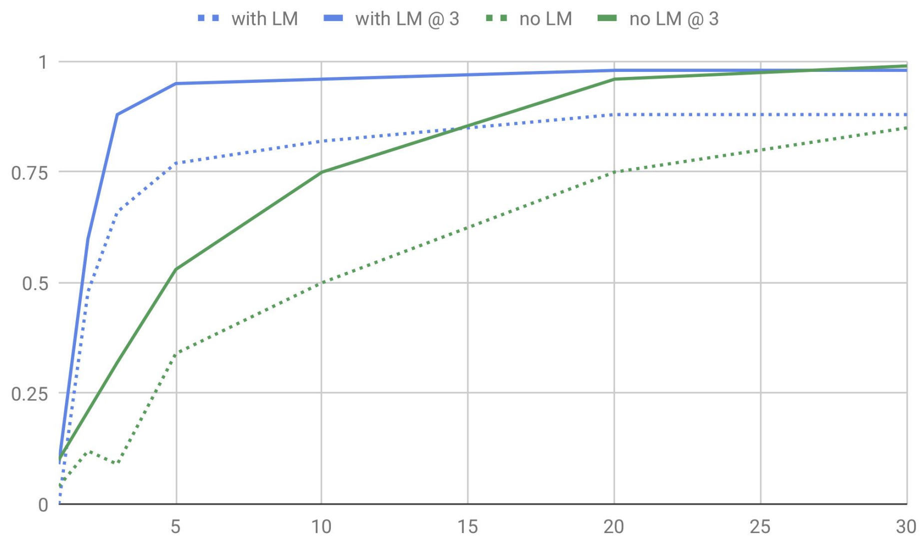
Table 4.9 – Results for the Graph Classification dataset

Transfer learning on graphs

Pre-training the model seems to enable faster training. For better accuracy, the model can be trained on multiple related tasks.

MSRC 21 [Winn et al. 2005]

Dataset of MRFs connecting super-pixels of an image, where the goal is to predict one of **21** labels *(e.g. building, grass, ...)*.

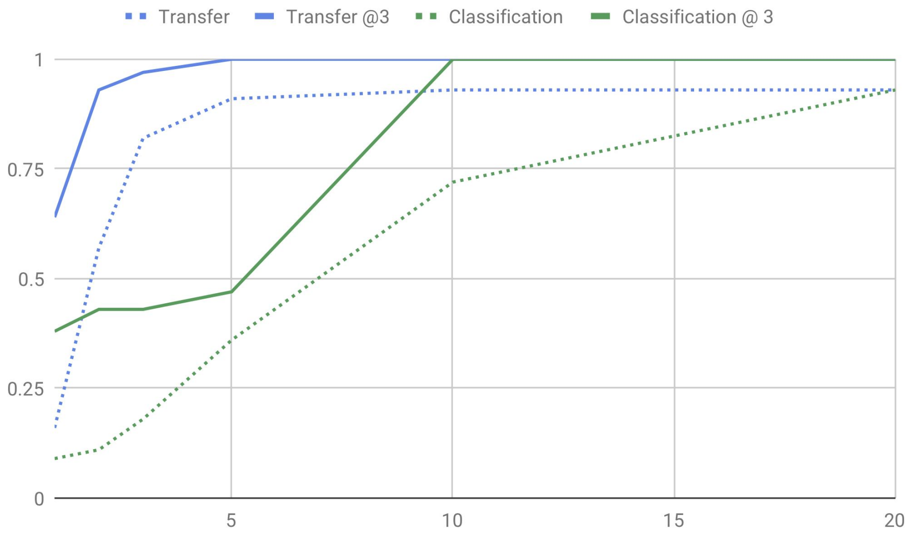


Transfer learning on graphs

Pre-training the model seems to enable faster training. For better accuracy, the model can be trained on multiple related tasks.

MSRC 21/9 [Winn et al. 2005]

Dataset of MRFs connecting super-pixels of an image, where the goal is to predict one of **21/9** labels *(e.g. building, grass, …)*.



4.2 **Learning from structure and context**

Datasets

We collect code from online repositories into three datasets at different scales.

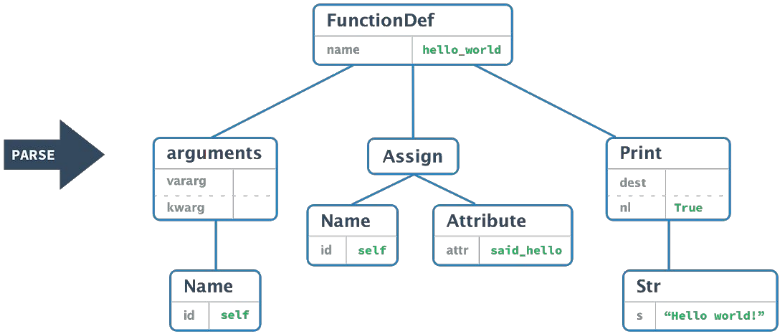
A fourth very large (3TB!) dataset is currently being curated.

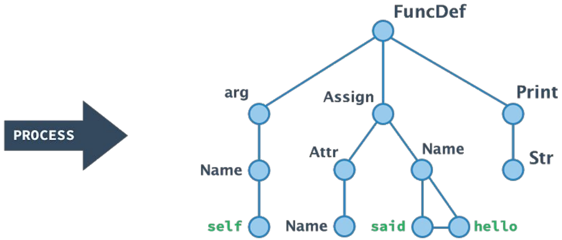
	LoC	# Snip.	# Tokens	# Unique Tok.	Avg. node deg.	Hash ID
keras	38,139	7,142	173,696	1,156	2.09 / 4.69	3e6db0e
sk-learn	192,663	35,228	776,365	3,581	2.07 / 4.61	611254d
pytorch	17,163	2,384	59,803	740	2.06 / 4.70	f3a860b
ansible	428,144	95,846	2,168,605	5,847	2.06 / 4.65	0b579a0
requests	5,036	699	11,508	452	2.06 / 4.72	2820839
django	121,188	22,892	337,444	3,413	2.05 / 4.71	cf826c9
httpie	3,919	612	8,886	421	2.06 / 4.65	358342d
youtube-dl	131,960	25,742	371,753	2,248	2.04 / 4.69	794c1b6
flask	7,750	804	13,086	490	2.05 / 4.64	4f3dbb3
BERT	5,928	1,967	17,805	480	2.06 / 4.60	bee6030
CORPUS-SM	65,225	7,142	173,696	1,146	2.09 / 4.69	
CORPUS-MID	247,965	44,754	1,009,864	3,823	2.07 / 4.67	
CORPUS-LG	951,890	193,316	3,938,951	9,769	2.06 / 4.67	

Table 4.1 – Dataset Statistics

Processing the data

```python
def hello_world(self):
    self.said_hello = True
    print("Hello world!")
```





1. RAW CODE

2. AST REPRESENTATION

3. PROCESSED AST

Preparing the data for pre-training

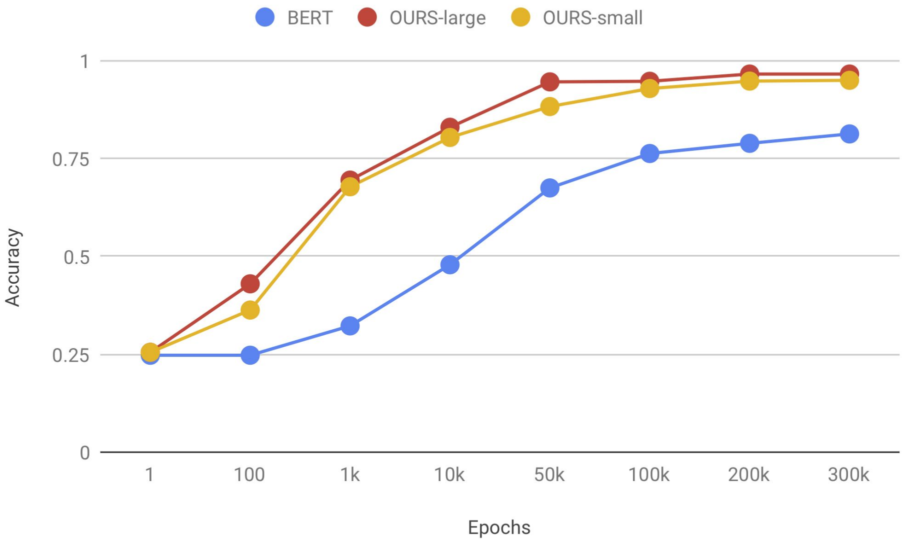
We generate a set of code snippets, defined as valid code subgraphs, and perturb the dataset for reconstruction in the Masked Language Model task.

Name	Range	Description
`nb_masked_tokens`	1-10	Number of tokens masked in training instance
`mask_probability`	0.15	Probability for uniform sampling of masked token
`noise_factor`	0.1	Probability of adding a random incorrect token to the training instance
`dupe_factor`	50	Number of generated training instances from each input instance
`max_seq_length`	64-128	Maximum length (resp. number of nodes) of input sequence (resp. graph)

Table 4.2 – Dataset Generation Hyperparameters

Pre-training: a semi-supervised task

Our syntax-aware model significantly outperforms BERT [Devlin et al, 2018] , providing some evidence that the addition of structure helps the model capture regularities.

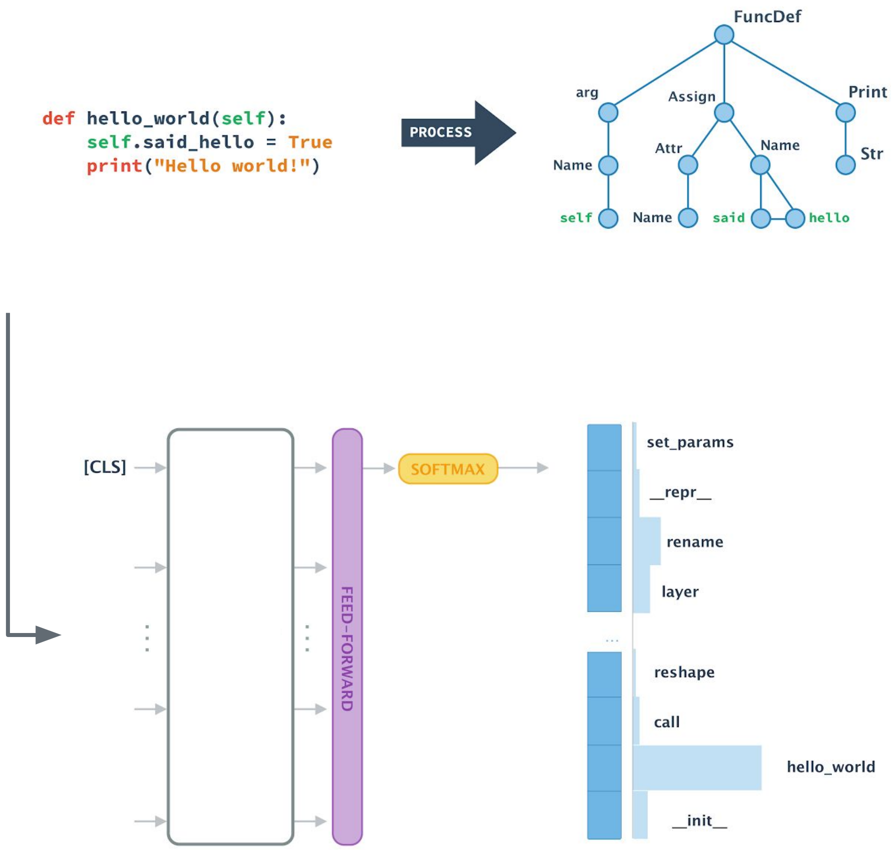


4.3 **Supervised tasks**

Supervised fine-tuning

We fine-tune the model on two standard tasks in the field of machine learning on source code:

1. **Method Naming**
2. Variable Naming



Method Naming

The addition of structural information seems to help outperform traditional LM architectures.

	F1-Macro*	F1-Weighted*	Subtoken Accuracy @1*
OURS			
Corpus-Sm	0.82	0.85	0.86
Corpus-Mid	0.68	0.76	0.81
Corpus-Lg	0.53	0.76	0.76
BERT			
Corpus-Sm	0.03	0.12	0.21

Table 4.5 – Method Naming Results

- * Points for partial match, at a token level
- * Exact match

Method Naming

We outperform State-of-the-art results, showing a 20% relative improvement to [Alon et al, 2019].

	Reported	Description
[Iyer et al., 2016]	0.275	RNN+Attention on textual representation of JAVA source code. Original work is done on C#/SQL ([Alon et al., 2019] for reported).
[Allamanis et al., 2016]	0.473	CNN+Attention run on JAVA source code.
[Alon et al., 2018]	0.511	Learning a CRF on paths generated from Python AST code (Accuracy measured @7).
[Alon et al., 2019]	0.633	RNN+attention embedding of paths on the AST, run on a filtered subset of JAVA code.
Ours	0.76	Generalized TRANSFORMER model run on Python code (CORPUS-lg).

Table 4.4 – Method Naming Results - Literature.

Method Naming

1

```
def deserialize(config, custom_objects=None):
    return deserialize_keras_object(config,
                                   module_objects=globals(),
                                   custom_objects=custom_objects,
                                   printable_module_name='regularizer')
```

Predictions
```
0. deserialize (1.0)
1. model_from_config (0.0)
2. from_config (0.0)
```

2

```
def __init__(self, minval=-0.05, maxval=0.05, seed=None):
      self.minval = minval
      self.maxval = maxval
      self.seed = seed
```

Predictions
```
0. __init__ (1.0)
1. on_train_begin (0.0)
2. preprocess_input (0.0)
```

3

```
def __call__(self, shape, dtype=None):
        return K.random_uniform(shape,
                                self.minval,
                                self.maxval,
                                dtype=dtype,
                                seed=self.seed)
```

Predictions
```
0. __call__ (0.995)
1. truncated_normal (0.001)
2. transform (0.0)
```

4

```
def get_config(self):
       return {
           'mean': self.mean,
           'stddev': self.stddev,
           'seed': self.seed
       }
```

Predictions
```
0. get_config (1.0)
1. _updated_config (0.0)
2. _preprocess_conv3d_kernel (0.0)
```

Method Naming

Failure modes reveals that interesting semantic information is being captured.

1

```
def glorot_normal(seed=None):
    return VarianceScaling(scale=1.,
                           mode='fan_avg',
                           distribution='normal',
                           seed=seed)
```

Predictions

```
0. he_normal (0.209)
1. lecun_normal (0.198)
2. lecun_uniform (0.198)
3. glorot_uniform (0.193)
4. he_uniform (0.19)
```

2

```
def call(self, x):
        output = K.dot(x, self.W)
        if self.bias:
            output += self.b
        output = K.max(output, axis=1)
        return output
```

Predictions

```
0. __call__ (0.554)
1. call (0.434)
2. recurrent_conv (0.001)
```

3

```
def add(inputs, **kwargs):
    return Add(**kwargs)(inputs)
```

Predictions

```
0. average (0.343)
1. maximum (0.326)
2. minimum (0.323)
```

Method Naming

The model can leverage both co-occurrence based semantics as well as structural similarities.

```
def sigmoid(x):
    return 1. / (1. + np.exp(-x))
```

Predictions

0. tanh (0.525)

```
def tanh(x):
    return np.tanh(x)
```

1. softplus (0.335)

```
def softplus(x):
    return np.log(1. + np.exp(x))
```

2. softsign (0.104)

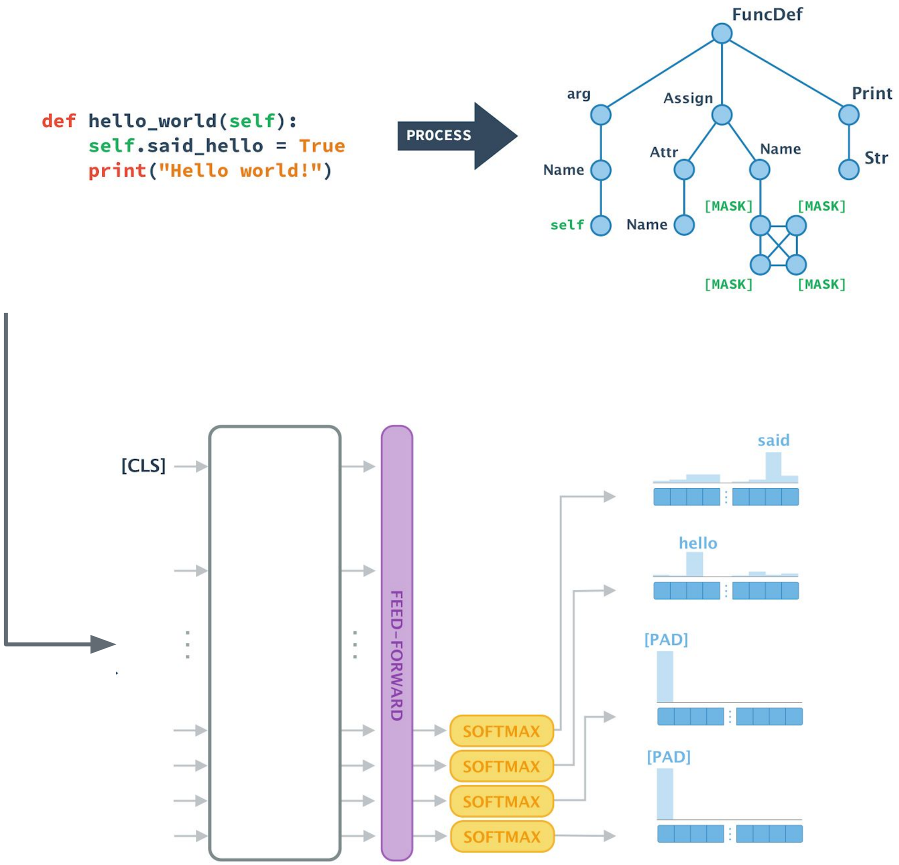
```
def softsign(x):
    return x / (1 + np.abs(x))
```

Supervised fine-tuning

We fine-tune the model on two standard tasks in the field of machine learning on source code:

1 Method Naming

2 Variable Naming



Variable Naming

We show clear improvements with the addition of structure, as well as state-of-the art results.

	Accuracy			
	@1	@3	@5	@7
BERT	0.3	0.43	0.48	0.52
OURS	**0.59**	**0.792**	**0.833**	**0.849**
[Alon et al., 2018]				
Assumed @1	0.567	-	-	-
[Allamanis et al., 2018b]				
PYTHON	0.536	-	-	-

Table 4.6 – Variable Naming Results

Variable Naming

1

```
for layer in model._input_layers:
  input_tensor = Input(batch_shape=layer.batch_input_shape,
                       dtype=layer.dtype,
                       sparse=layer.sparse,
                       name=layer.name)
  input_tensors.append(input_tensor)
  # Cache newly created input layer.
  newly_created_input_layer = input_tensor._keras_history[0]
```

Predictions ['layer', '[PAD]', '[PAD]', '[PAD]']

2

```
def selu(x):
    alpha = 1.6732632423543772848170429916717
    scale = 1.0507009873554804934193349852946
    return scale * K.elu(x, alpha)
```

Predictions [‘x’, '[PAD]', '[PAD]', '[PAD]']

3

```
def __call__(self, shape, dtype=None):
        return K.constant(0, shape=shape, dtype=dtype)
```

Predictions [‘self’, '[PAD]', '[PAD]', '[PAD]']

4

```
   for cell in self.cells:
     if isinstance(cell, Layer):
       trainable_weights += cell.trainable_weights
```

Predictions [‘cell’, '[PAD]', '[PAD]', '[PAD]']

4.4 **Sanity checks**

Permutation invariance

We shuffle the token input sequence order but preserve edges, ensuring that the model actually learns on the message-passing edges and not local co-occurrences in the flattened representation.

	Accuracy				MRR		
	@1	@3	@5	@7	@3	@5	@7
Standard	0.63	0.66	0.66	0.69	0.73	0.49	0.37
Random Permutations	0.628	0.65	0.67	0.68	0.72	0.478	0.36

Table 4.7 – METHODNAMING results, with and without permutations.

Syntactic correctness

To test the model's properties we evaluate the syntactic correctness of the predicted tokens, as defined by the language's grammar.

Token Type - 2 classes

- Language keyword
- User-provided token

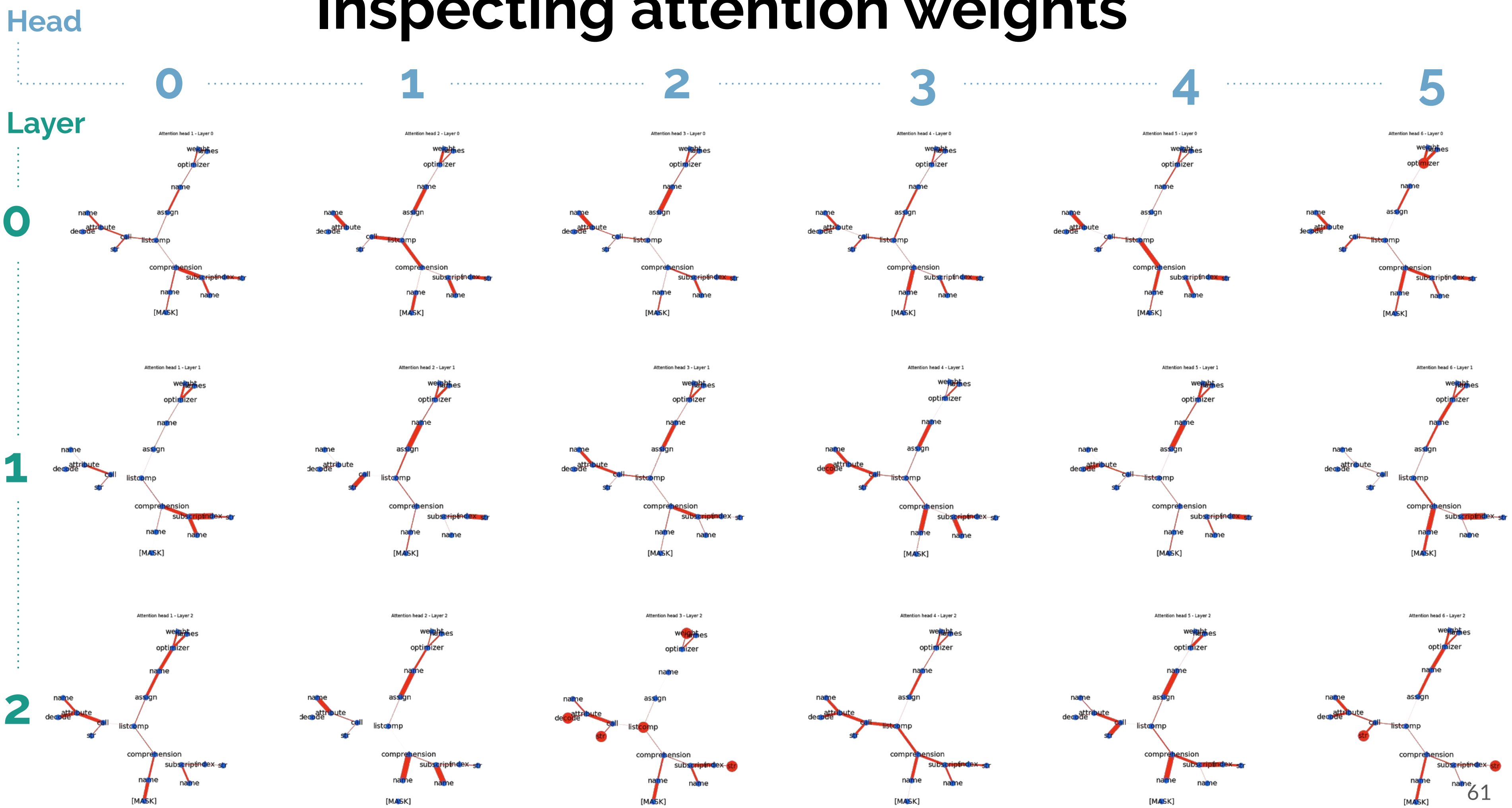
Token Class - 14 classes

- BoolOp - `And, Or`
- Expression - `Lambda, Yield, Num, Str, ...`
- Statement - `FuncDef, Return, If, While...`
- ...

	Token Type	Token Class		
	Accuracy	Accuracy	F-1 Macro	F-1 Weighted
BERT 200k iterations	0.990	0.979	0.92	0.91
OURS 200k iterations	**0.997**	**0.994**	**0.96**	**0.96**

Table 4.8 – Assessing the syntactical correctness of Masked Language Model predictions.

Inspecting attention weights



Inspecting attention weights

Head

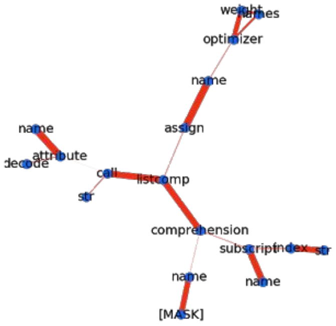
0 1 2 3 4 5

Layer

0

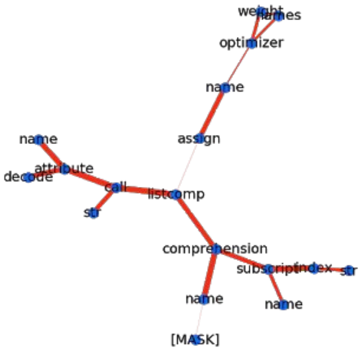
1

2



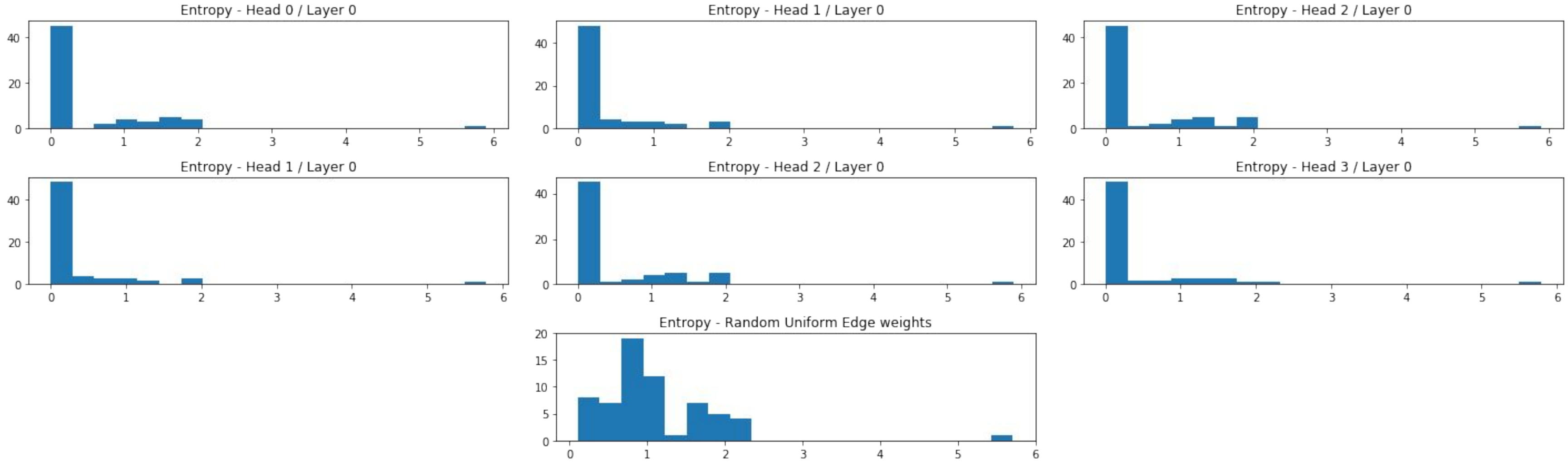
In early layers, the model has a receptive field that extends only to its immediate neighbours.

More complex attentive chains form in later layers.



Inspecting the entropy of attention weights

We measure the entropy of attention weights to see if the model is able to weigh different neighbours differently based on their importance, comparing it to uniform weights (all neighbours are equally important).



5 Conclusion

- We propose a model leveraging both **structural** and **contextual** information to embed graph-structured input.
- We show that adding structure provides strong semantic signals for the representations of source code.
- We present a model that can extend to several related tasks on graphs, encouraging re-use of prior knowledge.

Future Work

Reproducibility

The field of ML4Code could benefit from explicitly designed datasets, serving as diagnostics or evaluations on a standardized benchmark.

Architecture

Design more complex aggregation schemes, possibly incorporating more domain-specific information, global feature information or recursively aggregating at larger scales.

Similarity

Proxy tasks validate the approach but the final goal is to measure similarity in software. This requires designing a better evaluation of similarity, and extending to other languages and applications.

Thank you!

Questions?

Dylan Bourgeois
@dtsbourg

Additional slides

A - Reproducibility in ML4Code
B - Other work

A

Reproducibility Checklist

Inspired by the influential reproducibility checklist by Joëlle Pineau (adopted for NeurIPS this year!), we propose a specific version for ML4Code.

Data

- Is the data available? If yes, in which form?
  - ☐ Raw data.
  - ☐ Pre-processed data.
  - ☐ Output data.
- Is the pre-processing pipeline explicit?
  - ☐ What filters are applied? (*e.g.* removing low-frequency elements)
  - ☐ Which assumptions are made when generating the data? (*e.g.* snippets should be valid bits of code)
  - ☐ What transformations are applied to the original dataset?
  - ☐ What is the final representation that is passed to the model?
- Is the meta-data fully specified?
  - ☐ What is the origin of the corpus.
  - ☐ If the raw source forming the dataset is available online, are hashes or fingerprints of its version shared?
  - ☐ Is the programming language specified, including its version?
  - ☐ What are the Train / Test / Validation splits?

A

Reproducibility Checklist

Inspired by the influential reproducibility checklist by Joëlle Pineau (adopted for NeurIPS this year!), we propose a specific version for ML4Code.

Code

- Is the entire pipeline available? This includes the following components:
  - ☐ Data collection.
  - ☐ Data pre-processing.
  - ☐ Main algorithm loop and architecture.
  - ☐ *(Optional)* Post-processing steps.
  - ☐ Output in a form matching that of reported results.
- Is there a runnable version of the code provided? This includes the specification of:
  - ☐ The source platform and hardware specifications.
  - ☐ Dependency version information.
    or
  - ☐ A reproducible container which packages the entire project.

A

Reproducibility Checklist

Inspired by the influential reproducibility checklist by Joëlle Pineau (adopted for NeurIPS this year!), we propose a specific version for ML4Code.

Model

- Is the algorithm fully specified?
  - ☐ Hyperparameter sets.
  - ☐ Computational Cost analysis.
  - ☐ Number of iterations to convergence.
  - ☐ Ablation study.
  - ☐ Pre-trained model.
- Is the evaluation task fully specified?
  - ☐ Objective
  - ☐ Metric
  - ☐ Labels

A

SCUBA

Semantics of Code and Understanding BenchmArk

We would also like to propose a standardized benchmark dataset, whose development is in process, complete with an online leaderboard and diagnostics tasks.

Inspired by the GLUE benchmark.
[Wang et al. 2018]

Inference Tasks

Predicting a label or property of a set of tokens from the input, similar to a node classification.

Snippet-level evaluation

Predicting a label or property for an entire chunk of the input, similar to graph classification.

Similarity measures

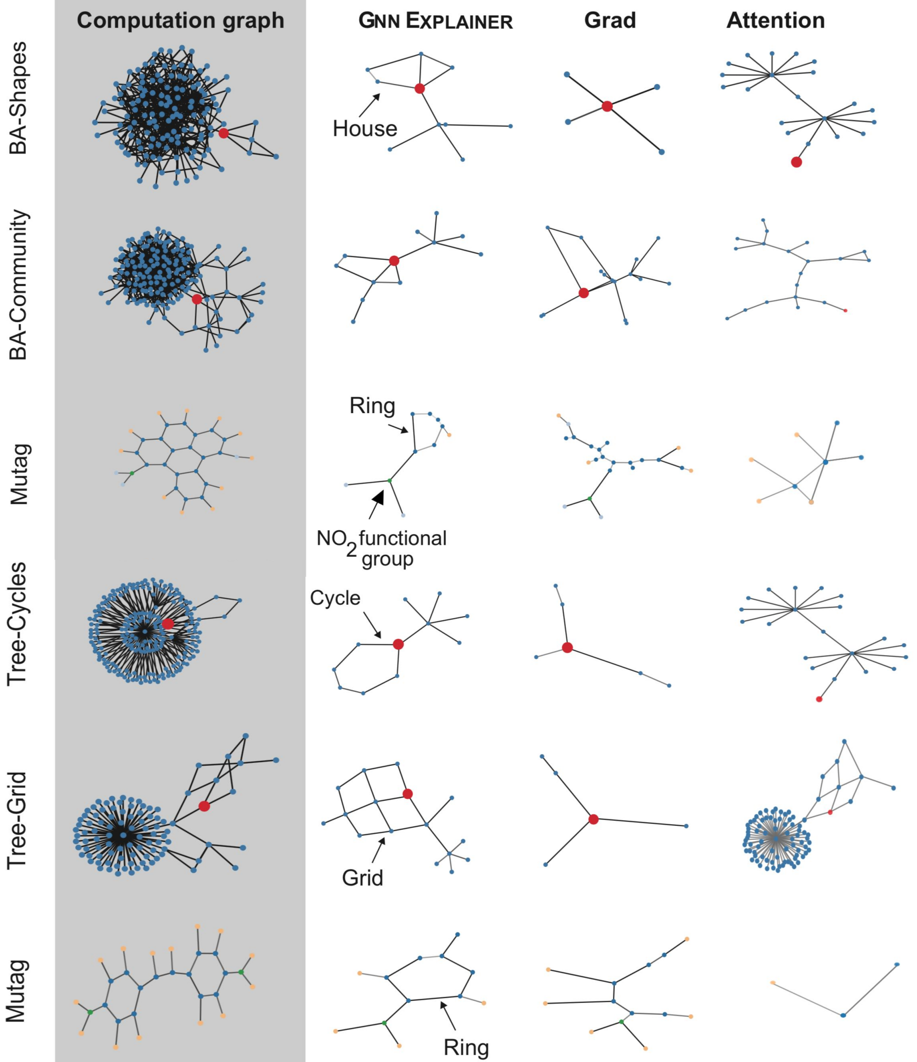
Predicting labels for sets of inputs, from similarity to link prediction.

B

GNN-Explainer: A tool for post-hoc interpretation of Graph Neural Networks

R. Ying, D. Bourgeois, J. You, M. Zitnik, J. Leskovec

KDD’19 (submitted)
arxiv:1903.03894

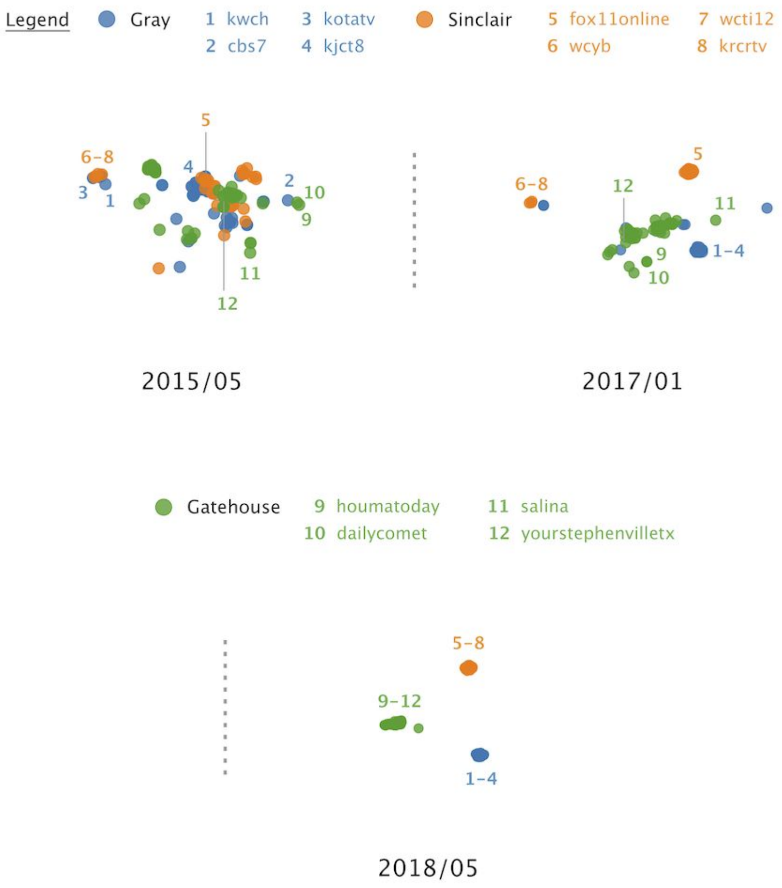


B

A dynamic embedding model of the media landscape

J. Rappaz*, D. Bourgeois*, K. Aberer

WebConf'19



Bibliography

[Allamanis, 2018] Allamanis, M. (2018). *The adverse effects of code duplication in machine learning models of code.* arxiv:1812.06469.

[Allamanis et al., 2015] Allamanis, M., Barr, E. T., Bird, C., and Sutton, C. (2015). *Suggesting accurate method and class names*. ESEC/FSE 2015, pages 38–49

[Alon et al., 2019] Alon, U., Zilberstein, M., Levy, O., and Yahav, E. (2019). *Code2vec: Learning distributed representations of code*. POPL.

[Allamanis et al., 2018a] Allamanis, M., Barr, E. T., Devanbu, P. T., and Sutton, C. A. (2018a). *A survey of machine learning for big code and naturalness*. ACM Comput. Surv., 51:81:1–81:37.

[Allamanis et al., 2018b] Allamanis, M., Brockschmidt, M., and Khademi, M. (2018b). *Learning to represent programs with graphs.* ICLR.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). *A neural probabilistic language model*. J. Mach. Learn. Res., 3:1137–1155.

Bibliography

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). *A unified architecture for natural language processing: Deep neural networks with multitask learning.* ICML '08.

[Deerwester et al.,1990] Deerwester,S.C.,Dumais,S.T.,Landauer,T.K.,Furnas,G.W.,and Harshman, R. A. (1990). *Indexing by latent semantic analysis.* JASIS, 41:391–407.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). *BERT: Pre-training of deep bidirectional transformers for language understanding*. Arxiv:1810.04805.

[Firth,1957] Firth,J.R.(1957).*A synopsis of linguistic theory* 1930-55. Studies in Linguistic Analysis (special volume of the Philological Society), 1952-59:1–32.

[Hindle et al., 2012] Hindle, A., Barr, E. T., Su, Z., Gabel, M., and Devanbu, P. (2012). *On the naturalness of software*. In ICSE '12, pages 837–847, IEEE Press.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). *Efficient estimation of word representations in vector space*. ICLR'13

Bibliography

[Peters et al., 2018] Peters,M.E.,Neumann,M.,Iyyer,M.,Gardner,M.,Clark,C.,Lee,K.,and Zettlemoyer, L. S. (2018). *Deep contextualized word representations*. In NAACL-HLT.

[Radford et al., 2018] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). *Improving language understanding by generative pre-training.* OpenAI.

[Shannon, 1950] Shannon, C. (1950). *Prediction and entropy of printed english*. Bell Systems Technical Journal.

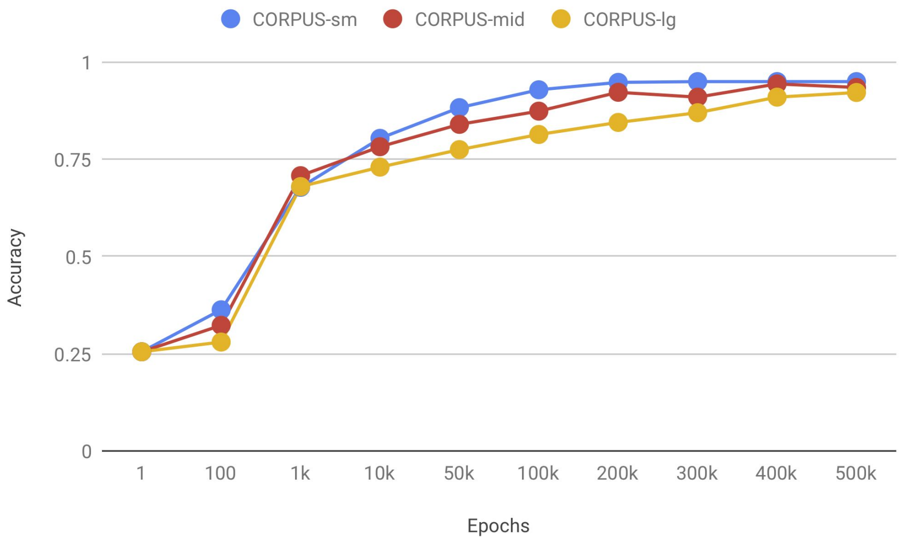
[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). *Attention is all you need.* In NeurIPS.

[Wang et al., 2018] Wang,A.,Singh,A.,Michael,J.,Hill,F.,Levy,O.,andBowman,S.R.(2018). *GLUE: A multi-task benchmark and analysis platform for natural language understanding*. arXiv:1804.07461.

[Xu et al., 2019] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). *How powerful are graph neural networks?* In ICLR'19.

Pre-training: a semi-supervised task

The results are consistent across corpora.



Multi-task capabilities