Analyzing Software using Deep Learning

Pre-training and Fine-tuning

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2022

Overview

- **Pre-training: General ideas**
- **Models: Transformers, BERT, GPT**
- **Codex and Applications**

Recommended papers:

- ”CodeBERT: A Pre-Trained Model for Programming and Natural Languages.”, EMNLP, 2020
- ”Language Models are Few-Shot Learners”, NeuIPS, 2020
- ”Evaluating Large Language Models Trained on Code”, arXiv, 2021

Pre-training: Motivation

- **Supervised learning:**

  Powerful models for a specific task

- **But:**
  - Need large amounts of labeled training data
  - New architecture and training for each task

Pre-training: Idea

- **Train a model on a "pseudo-task" which**
  - comes with large amounts of data with little effort
    - Self-supervised learning
  - teaches a model to "understand" the data
- **Once pre-trained, apply the model to the actual task**

Pre-training Tasks

- **Auto-regressive language modeling**
  - Given previous tokens, predict next token
- **Next sentence prediction**
  - Predict whether one sentence follows another
- **Token masking**
  - Predict a missing token
- **Replaced token detection**
  - Predict whether a token fits the sequence

Auto-regressive Language Modeling

- Classical lang. modeling task
  - → Given: Sequence of tokens
  - → Objective: Predict next token (e.g., out of fixed vocabulary)

Example (Python):

```
if x is not in units:
    raise ?
          ⋮
          ∨
      ValueError ?
                 ⋮
                 ∨
                 (     etc.
```

Next Sentence Prediction

- Binary classification task
  - → Given: Two sentences s_1, s_2
  - → Objective: Predict whether s_2 follows s_1

- Example (NL)

  s_1: the man went to the store

  s_2: he bought a gallon of milk

  label: is-next (i.e., true)

Token Masking

- Bi-directional lang. modeling task
  - → Given: Sequence of tokens with some (e.g. 15%) masked out
  - → Objective: Predict missing tokens (e.g., out of fixed vocabulary)

- Example (Python):

```
if ? not in units ?:
    raise ?("invalid value" + x)
```

x

ValueError

Replaced Token Detection

- Binary classification task
  - → Given: Sequence of tokens with some tokens (e.g., 15%) replaced with plausible alternatives
    - Generated, e.g., using n-gram model
  - → Objective: Predict which tokens were replaced (fake vs. real)
- Example (Python):

```
if y not in units:
    raise TypeError("invalid value" + x)
```

Using a Pre-trained Model

Once pre-trained, two ways of using a model

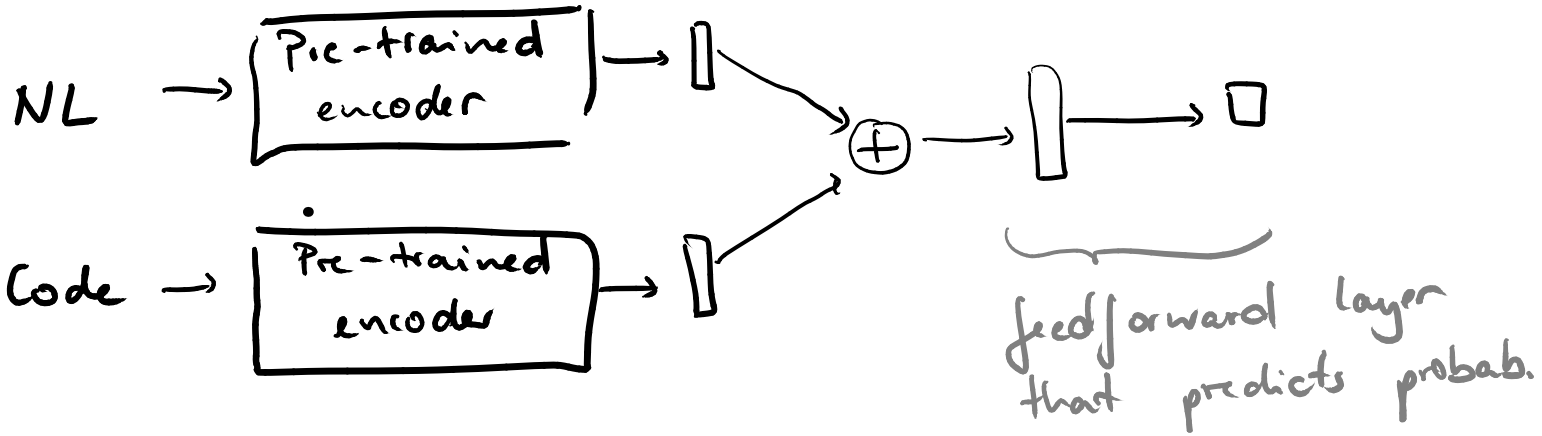
- Option 1: Fine-tune for downstream task
  - Train on (possibly) small set of labeled data
- Option 2: Few-shot learning
  - No additional training at all
  - Instead: Provide a few examples as part of a query

Fine-tuning for Downstream Task

- **Assumption: Labeled dataset for downstream task available**
  - Typically, 1000+ examples
- **Pre-training: Weights and biases tuned for general code understanding**
- **Fine-tuning:**
  - Start from pre-trained encoder
  - Supervised training of model for actual task

Fine-tuning: Example

- Task: NL-to-code search
  - → Given: NL sentence + code example
  - → Objective: Predict if NL describes the code
- Model:



Few-Shot Learning

- **No fine-tuning of the model at all**
- **Construct a "prompt" with**
  - A few examples (e.g., 10) of inputs and expected outputs
  - Another input, without corresponding output
- **Auto-regressive language model completes the prompt with an output**

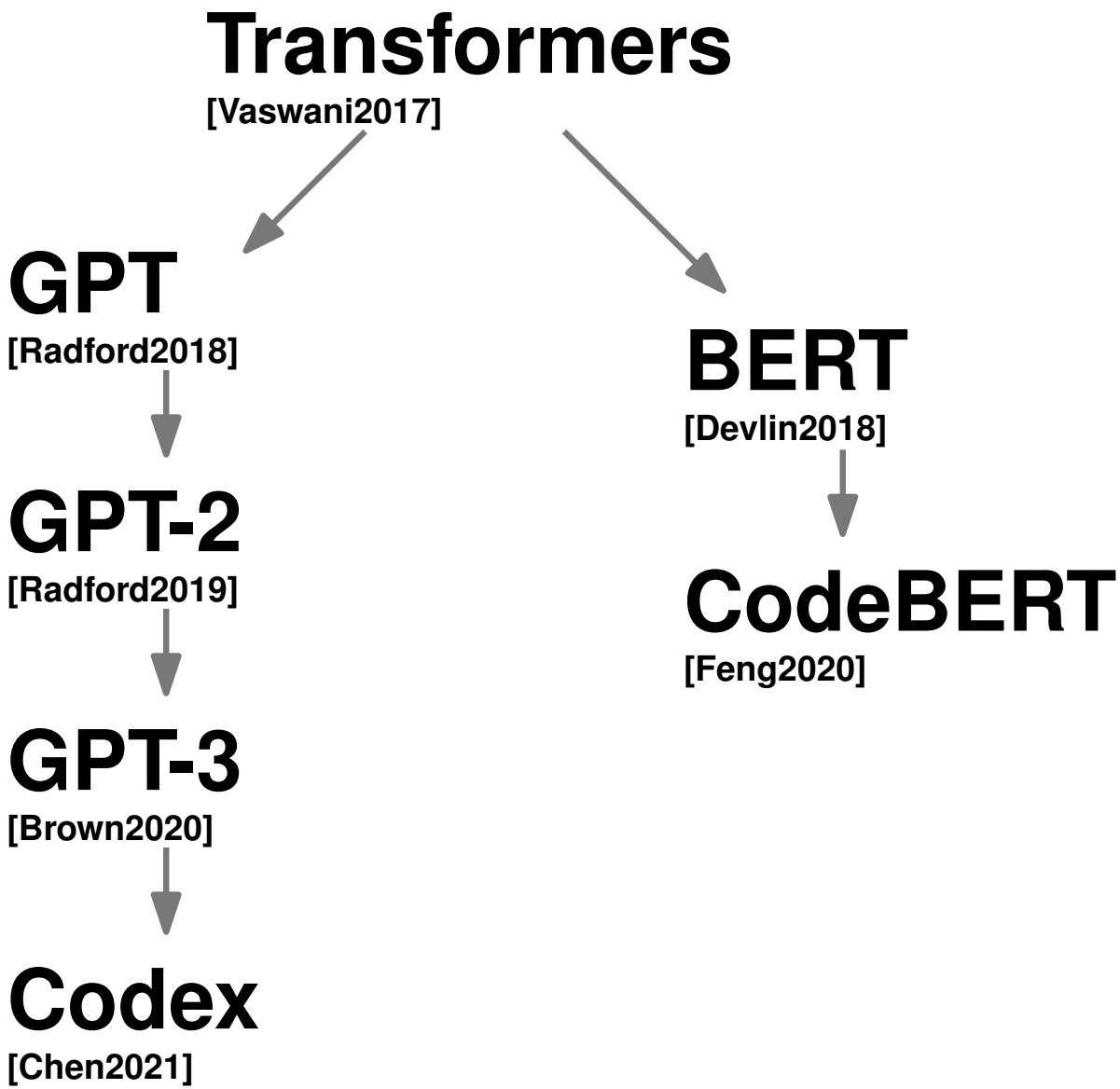
Overview

- **Pre-training: General ideas**
- **Models: Transformers, BERT, GPT**
- **Codex and Applications**

Recommended papers:

- ”CodeBERT: A Pre-Trained Model for Programming and Natural Languages.”, EMNLP, 2020
- ”Language Models are Few-Shot Learners”, NeuIPS, 2020
- ”Evaluating Large Language Models Trained on Code”, arXiv, 2021

Overview of Models



Transformers

- **Model to reason about sequences**
- **Limitations of RNNs**
  - Encoding bottleneck
  - For long sequences (> 100): Details get lost
- **Instead: No recurrence, but self-attention**
  - Learned “attention function” that determines which parts of the input sequence are most relevant

GPT Models

- **Transformer-based, auto-regressive language models**
- **Trained on huge amounts of data**
  - E.g., GPT-3:

    570GB of text crawled from the web
  - Enough to train large models without ever showing same sequence twice

BERT Models

- **Transformer-based sequence encoder**
- **Pre-training objectives**
  - Predict masked tokens
  - Predict whether two sentences occur one after the other

CodeBERT

- **BERT-like, bi-modal model trained on pairs of**
  - Code (functions)
  - Natural language (paragraph of function-level documentation)
- **Pre-training objectives**
  - Predict masked tokens
  - Replaced token detection
- **Trained on six PLs**

Overview

- **Pre-training: General ideas**
- **Models: Transformers, BERT, GPT**
- **Codex and Applications**

Recommended papers:

- ”CodeBERT: A Pre-Trained Model for Programming and Natural Languages.”, EMNLP, 2020
- ”Language Models are Few-Shot Learners”, NeuIPS, 2020
- ”Evaluating Large Language Models Trained on Code”, arXiv, 2021

Codex

- **GPT-3 model trained on code**
  - Initially, trained on natural language
  - Further trained on 159GB of Python code
- **Training objective:**

  Auto-regressive language model

Applications of Codex

- **NL-to-code**
  - Predict functions from docstrings
- **Code mutation**
  - Transform code to introduce a bug
- **Test oracle generation**
  - Turn documentation into executable specifications
- **Unit-level test case generation**

Last three: Our own work (joint with Patrick Bareiß, Beatriz Souza, and Marcelo d'Amorim)

NL-to-Code: Prompt

Input given to the model

```
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

Output predicted by the model

NL-to-Code: Results

- **Dataset of 164 programming problems**
  - Algorithms, math quizzes, etc.
  - Similar to simple software interview questions
  - Validate correctness via unit tests
- **Codex predicts functionally correct program for 29% of the problems**

Code Mutation: Prompt

```
Generate mutations for the following snippets of code.
[[Code]]
long biasedExp = (longBits & DoubleConsts.EXP_BIT_MASK)»
      (DoubleConsts.SIGNIFICAND_WIDTH - 1);
[[Mutations]]
- longBits & DoubleConsts.EXP_BIT_MASK |==> longBits |
      DoubleConsts.EXP_BIT_MASK
- longBits & DoubleConsts.EXP_BIT_MASK) »
      (DoubleConsts.SIGNIFICAND_WIDTH - 1) |==> longBits &
      DoubleConsts.EXP_BIT_MASK) « (DoubleConsts.SIGNIFICAND_WIDTH -
      1)
- 1 |==> 0
- DoubleConsts.SIGNIFICAND_WIDTH - 1 |==>
      DoubleConsts.SIGNIFICAND_WIDTH % 1
[[Code]]
...
(3 more examples)
...
[[Code]]
WeightMass mass = lhsDist.get(classVal);
[[Mutations]]
- classVal |==> classVal + 1
- classVal |==> 0
```

Input

Output

Code Mutation: Results

- **1,194 examples from 32 Java classes**
  - Each example: One line of code
- **63% of the generated mutants compile**
- **90% of the compiling mutants change the behavior**
- **Comparison with an existing code mutation tool (Major)**
  - 18% of the mutants overlap, i.e., complementary

Test Oracle Generation: Prompt

```
### Signature
public static java.lang.String toString(java.lang.Object[] a)
### Comment
...
The value returned by this method is equal to the value that would
be returned by Arrays.asList(a).toString(), unless a is null, in
which case "null" is returned.
### Analysis
This method returns the same thing as the expression
Arrays.asList(a).toString(), therefore they are equivalent
(at least as long as a is not null).
### Equivalence
if (a != null)  toString(a) <-> Arrays.asList(a).toString() ;
...
(3 more examples)
...
### Signature
public double norm2(cern.colt.matrix.DoubleMatrix1D x)
### Comment
Returns the two-norm (aka euclidean norm) of vector x;
equivalent to mult(x,x).
### Analysis
This method is equivalent to the expression mult(x,x).
### Equivalence
norm2(x) <-> mult(x,x);
```

Input

Output

Test Oracle Generation: Results

- **299 metamorphic test oracles from nine Java projects**
- **Effectiveness of Codex**
  - 82% precision, 47% recall
- **Existing, rule-based tool specifically built for this purpose (MeMo)**
  - 64% precision, 54% recall
  - Roughly on par

Test Case Generation: Prompt

```
Suggest a test for a method with the DoubleArrayList
      quantiles(DoubleArrayList percentages) signature.
Helper constructors and methods:
   DynamicBin1D()
   DoubleArrayList()
Method: public synchronized double max() {
   if (! isIncrementalStatValid) updateIncrementalStats();
   return this.max;
}
Test: public static void testMax() {
   double[] temp = new double[2];
   temp[0] = 8.9;
   temp[1] = 1;
   DenseDoubleMatrix1D d1Double = new DenseDoubleMatrix1D(temp);
   hep.aida.bin.DynamicBin1D d1ynamicBin =
         cern.colt.matrix.doublealgo.Statistic.bin(d1Double);
   double max = d1ynamicBin.max();
   System.out.println("max="+ max);
}
–
Method:public DoubleArrayList quantiles(DoubleArrayList percentages)
{
   return Descriptive.quantiles(sortedElements_unsafe(),percentages);
}
Test:public static void testQuantiles() {
   double[] temp = new double[2];
   temp[0] = 8.9;
   temp[1] = 1;
   DenseDoubleMatrix1D d1Double = new DenseDoubleMatrix1D(temp);
   hep.aida.bin.DynamicBin1D d1ynamicBin =
         cern.colt.matrix.doublealgo.Statistic.bin(d1Double);
   DoubleArrayList quantiles = d1ynamicBin.quantiles(new
         DoubleArrayList(new double[] 0.5,0.75));
   System.out.println("quantiles="+ quantiles);
}
```

Input

Output

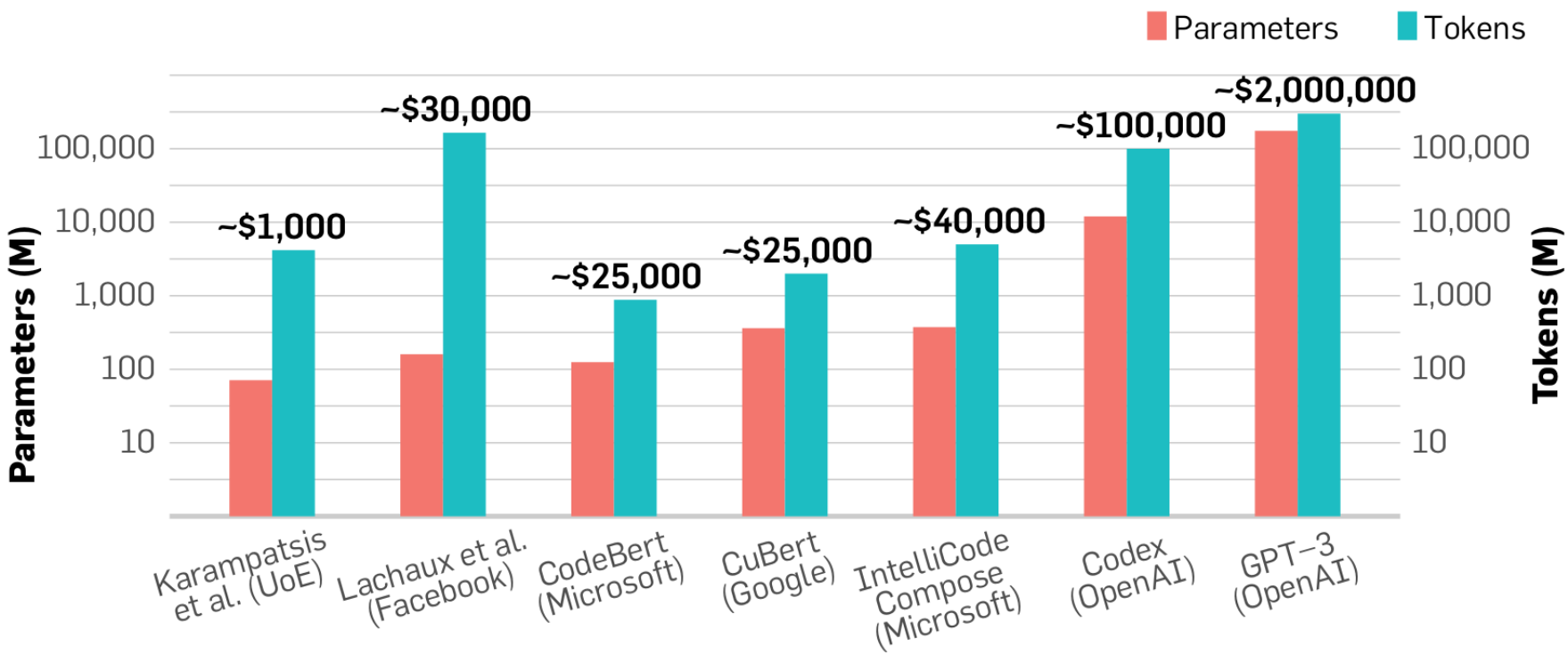
Test Case Generation: Results

- **18 methods under tests from nine Java projects**
  - Generate 100 test cases per method
- **202 test cases compile**
- **Tests achieve 14% line coverage**
- **Comparison with non-learning-based tool (Randoop)**
  - 682 test cases compile
  - Tests achieve 10% line coverage

Cost of Pre-training

- **Best-performing models:**
  Huge computational effort
  - Days or even weeks of training time on high-end GPU clusters
  - Hundreds of GB of training data
- **Trade-off between generality and cost**

Cost of Pre-training (2)



Source: *The Growing Cost of Deep Learning for Source Code*. Hellendoorn, 2022

Summary

- **Pre-training: Powerful work-around for lack of (lots of) labeled training data**
- **Key to success**
  - Large-scale self-supervised training
  - Pseudo task(s) that teach model to "understand" the data
- **Once pre-trained, apply model via**
  - Fine-tuning on downstream task, or
  - Few-shot learning