# An introduction to automatic program repair

Luke Braithwaite

Peterhouse Outreach Event — 1st February 2024

# 👋 Hello & Welcome

# Outline

- What are software bugs
- What is automatic program repair?
- Modelling code using graphs
- Deep learning on graphs

# The effect of bad code…

The largest miscarriage of justice in British history

The Economist

Menu | Weekly edition | The world in brief | Search

Britain | Justice in the mail

## Britain’s worst miscarriage of justice sparks outrage at last

A TV drama shines a spotlight on a Post Office scandal that has been known about for years

Source: The Economist

Loss of a $125 million martian rover

Los Angeles Times

A **bug** is incorrect or undesirable program behaviour caused by an error or mistake in the code.

# Bug types

| Bug type | Description |
|---|---|
| **Syntax error** | Code breaks the language's rules e.g. typos. |
| **Semantic error** | Code is valid but breaks type/semantic rules e.g. using a string to index an array. |
| **Logical bugs** | The program runs but output is incorrect/not desired e.g. loop condition is incorrect. |
| **Performance bug** | The code has a performance issue e.g. code uses too much memory or CPU time. |
| **Security bug** | The code is insecure e.g. improper verification or buffer overflow. |

# Your turn

Spot the bugs!

```
name = imput('What is your name:')
age = input('What is your age:')
choice = int(input('Pick a drink (0-3):'))

drinks = ['wine', 'beer', 'port']
can_drink = age > 21
print(f'Can {name} drink: {can_drink}')
print(drinks[choice])
```

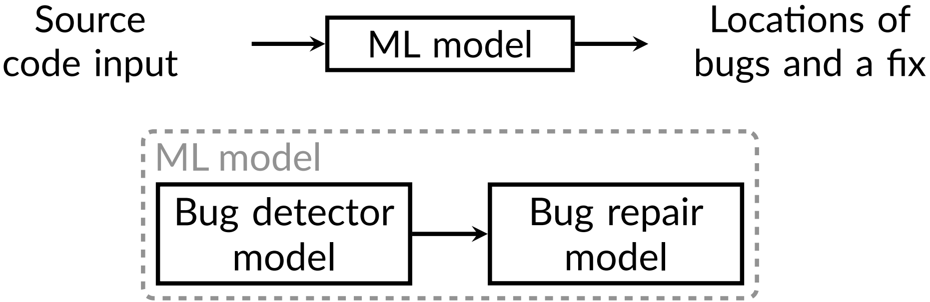
# Answers

Spot the bugs!

```
name = input('What is your name:')
age = int(input('What is your age:'))
choice = int(input('Pick a drink (0-2):'))

drinks = ['wine', 'beer', 'port']
can_drink = age >= 18
print(f'Can {name} drink: {can_drink}')
print(drinks[choice])  # check that 0 <= choice <= 2
```

# Automatic program repair

Using machine learning (ML) to automatically **detect** and **repair** bugs in source code.



# Can we just use ChatGPT?

# Source code as natural language

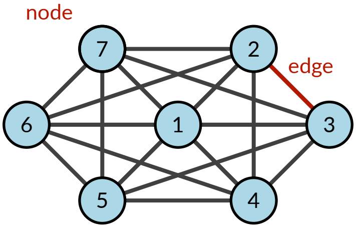
- We could treat source code as sequence of tokens and then apply techniques from NLP.
  - ChatGPT and other generative models can do this (sort of)
- But important semantic information is encoded in the structure and relationships in the code.
  - e.g. variable referencing, function calls, type relationships.
- Want a way to encode this relational information.

**Use a graph**

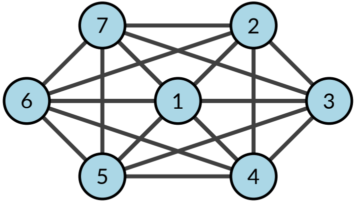
# What is a graph?

Graphs are a way to model relational data.

**Nodes** model the entities we care about and **edges** model the relationships between nodes.



# Examples of graphs



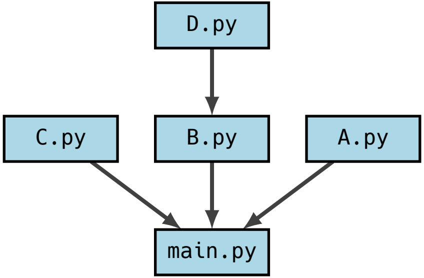

- Computer networks
- Social networks
- Airspace and airways
- Source code

# Common graph representation of code

# Dependency graphs

Dependency graphs model the dependency structure

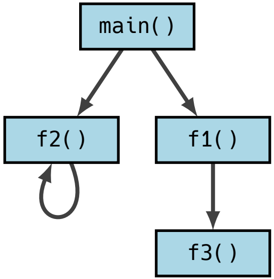
```python
# main.py
import A
import B
import C

# B.py
import D
```



# Call graphs

Call graphs model the function caller-callee structure

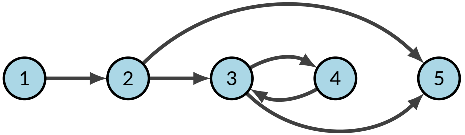
```python
def f2():
    return f2()

def f1():
    return f3()

def main():
    f1()
    f2()
```



# Control flow graph

Control flow graphs model the execution paths through a program

```python
i = ...
if i == 1:
  for j in range(10):
    print(j)
print("finished")
```



# Other graph representations

- Dependency structure over tokens (Raychev et al., 2015)

$$\texttt{a} = \texttt{b} \implies \texttt{b} \xrightarrow{\text{dependency}} \texttt{a}$$

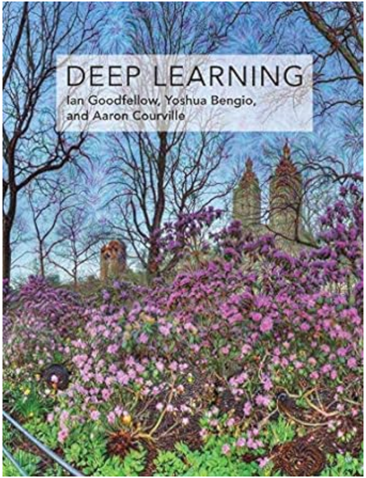
- Type dependencies (Wei et al., 2019)

$$\texttt{a} : \texttt{int} = \texttt{1} \implies \texttt{int} \xrightarrow{\text{type}} \texttt{a}$$

- Combining all of the above (Allamanis et al., 2021)
- Consider higher-order relations (Georgiev et al., 2022)

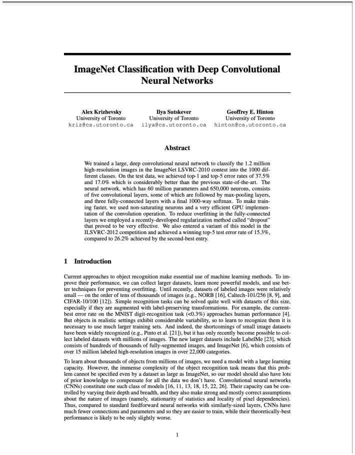
# Deep learning for APR

# An (abridged) introduction to deep learning




- Machine Learning — using computers to **learn from data**
- Recent advances in ML is due to **deep learning**
- Deep learning uses **large neural networks** to learn from examples and then generalise to unseen data

# A (modern) history of deep learning



## ImageNet Classification with Deep Convolutional Neural Networks


Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca



### Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.


### 1 Introduction

Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting. Until recently, datasets of labeled images were relatively small — on the order of tens of thousands of images (e.g., NORB [16], Caltech-101/256 [8, 9], and CIFAR-10/100 [12]). Simple recognition tasks can be solved quite well with datasets of this size, especially if they are augmented with label-preserving transformations. For example, the current-best error rate on the MNIST digit-recognition task (<0.3%) approaches human performance [4]. But objects in realistic settings exhibit considerable variability, so to learn to recognize them it is necessary to use much larger training sets. And indeed, the shortcomings of small image datasets have been widely recognized (e.g., Pinto et al. [21]), but it has only recently become possible to collect labeled datasets with millions of images. The new larger datasets include LabelMe [23], which consists of hundreds of thousands of fully-segmented images, and ImageNet [6], which consists of over 15 million labeled high-resolution images in over 22,000 categories.

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. However, the immense complexity of the object recognition task means that this problem cannot be specified even by a dataset as large as ImageNet, so our model should also have lots of prior knowledge to compensate for all the data we don't have. Convolutional neural networks (CNNs) constitute one such class of models [16, 11, 13, 18, 15, 22, 26]. Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

1




## Attention Is All You Need


Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com



### Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.


*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.


31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

arXiv:1706.03762v7 [cs.CL] 2 Aug 2023

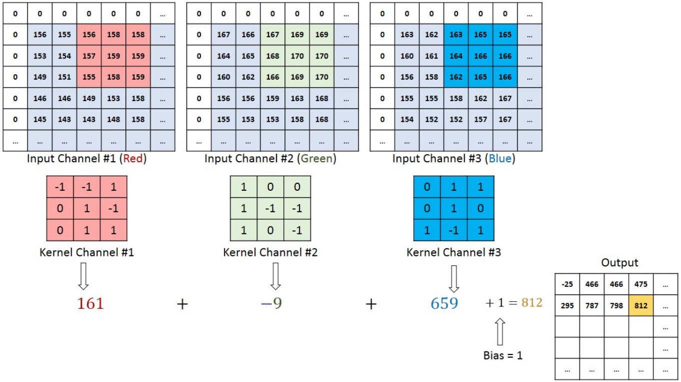
# Motivating graph neural networks (GNNs)

# Motivation I: The locality principle for images

**A pixel’s value depends on the value of its neighbours**

- Key idea that underpins large amount of image processing.
- Basis for **convolution**
    - Image is just a grid of pixels
    - **Convolution** lets you apply a kernel over the image to detect features such as edges
    - A pixels values is calculated by finding the weighted sum of its neighbours
    - 3blue1brown has a very good youtube video about it if interested

# Motivation II: Convolution

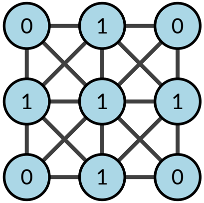



Source: A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way

# Motivation III: Images are graphs

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |

$$\Longleftrightarrow$$



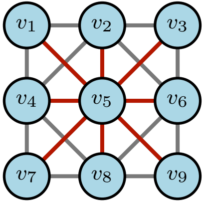

🤔 Can we perform convolutions on graphs?

**Yes**

# Message passing on graphs

- Message passing generalises convolutions to graphs

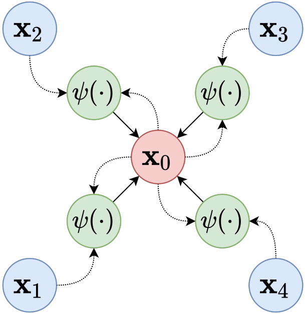
- We can define image convolution using this




$$v'_5 = \sum_{i=1}^{9} \underbrace{w_i \cdot v_i}_{\text{message from } v_i \text{ to } v_5}$$

- We use this to define **message passing layers** to process graphs
  - This in turn is used to define **Graph Neural Networks (GNNs)**

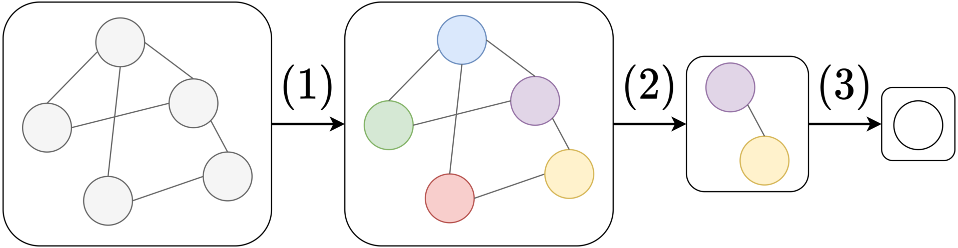
# Message passing layers




Source: Wikipedia

1. Compute message sent from each neighbour by calculating $\psi(\mathbf{x}_0, \mathbf{x}_i)$
2. Aggregate all the messages using a permutation (order) invariant function
3. Pass the aggregated representation through a non-linear activation function $\phi(\cdot)$

# Graph Neural Networks (GNNs)




Source: Wikipedia

1. **Message passing (convolution)** to update representation
2. **Local pooling** to coarsen or downscale the graph
3. **Global pooling (readout)** to get model output

# GNNs for automatic program repair

## GNNs outperform other methods on APR tasks

### Self-Supervised Bug Detection and Repair

**Miltiadis Allamanis, Henry Jackson-Flux,* Marc Brockschmidt**
Microsoft Research, Cambridge, UK
{miallama, mabrocks}@microsoft.com

**Abstract**

Machine learning-based program analyses have recently shown the promise of integrating formal and probabilistic reasoning towards aiding software development. However, in the absence of large annotated corpora, training these analyses is challenging. Towards addressing this, we present BUGLAB, an approach for self-supervised learning of bug detection and repair. BUGLAB co-trains two models: (1) a detector model that learns to detect and repair bugs in code, (2) a selector model that learns to create buggy code for the detector to use as training data. A Python implementation of BUGLAB improves by up to 30% upon baseline methods on a test dataset of 2374 real-life bugs and finds 19 previously unknown bugs in open-source software.

Allamanis et al. (2021)

### HEAT: Hyperedge Attention Networks

**Dobrik Georgiev*** *dgg30@cam.ac.uk*
*Department of Computer Science and Technology, University of Cambridge, UK*

**Marc Brockschmidt** *mmjb@google.com*
*Microsoft Research, Cambridge, UK†*

**Miltiadis Allamanis** *mallamanis@google.com*
*Microsoft Research, Cambridge, UK†*

**Reviewed on OpenReview:** https://openreview.net/forum?id=gCmQK6McbR

**Abstract**

Learning from structured data is a core machine learning task. Commonly, such data is represented as graphs, which normally only consider (typed) binary relationships between pairs of nodes. This is a substantial limitation for many domains with highly-structured data. One important such domain is source code, where hypergraph-based representations can better capture the semantically rich and structured nature of code.

In this work, we present HEAT, a neural model capable of representing typed and qualified

Georgiev et al. (2022)

# Next steps

- More data and experiments
- Improving user experience (UX)
- Developing better model architectures
- Accounting for higher-order (multi-node) relationships and structures — hypergraphs
- Language-agnostic graph representations

# Conclusions

- ChatGPT is not always the answer
- Structure and relationships are useful
- Graphs are a natural way to model code
- We can use GNNs to automate bug detection and repair
- GNNs outperform NLP and traditional ML methods
- Graphs are useful in other problems and domains

# Questions

# References I


- Allamanis, Miltiadis, Henry Jackson-Flux, and Marc Brockschmidt (2021). "Self-Supervised Bug Detection and Repair". In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., pp. 27865–27876.
- Georgiev, Dobrik Georgiev, Marc Brockschmidt, and Miltiadis Allamanis (2022). "HEAT: Hyperedge Attention Networks". In: *Transactions on Machine Learning Research*. ISSN: 2835-8856.
- Raychev, Veselin, Martin Vechev, and Andreas Krause (2015). "Predicting Program Properties from "Big Code"". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '15. New York, NY, USA: Association for Computing Machinery, pp. 111–124. ISBN: 978-1-4503-3300-9.

# References II

Wei, Jiayi et al. (2019). “LambdaNet: Probabilistic Type Inference using Graph Neural Networks”. In: International Conference on Learning Representations.