All models are wrong, some are useful
George Box
Supervised Machine Learning

- Given an i.i.d. dataset \(\{(x_1, y_1), \ldots, (x_N, y_N)\}\)
- Pick \(\theta\) that minimize Loss \(\mathcal{L}(\theta) = \frac{1}{N} \sum_i L(f_{\theta}(x_i), y_i)\)
Gradient Descent: Learning Model Parameters $\theta$

while not converged
• Compute (estimate of) derivative $g \approx \nabla_\theta \mathcal{L}(\theta)$
• Update $\theta \leftarrow \theta - q(g)$
Generalization

Underfitting

Overfitting

graphs from http://antianti.org/?p=175
Distributed Vector Representations

Local representation (1-hot)

\[ r = E \mathbb{I}_w \] with \( E \) a \( D \times V \) matrix

"vocabulary"

\( \mathbb{I}_w \)
Graph Notation

• Nodes/Vertices

• Edges/Links

\[ G = (V, E) \]
Graph Neural Networks

and Neural Message Passing
Graph Neural Networks

Graph Representation of Problem

Initial Representation of each node
Graph Neural Networks

Initial Representation of each node

Output Representations of each Node

Task Specific Stuff + Loss
Graph Neural Networks

Initial Representation of each node

Output Representations of each Node

Task Specific Stuff + Loss
Graph Neural Networks

GNN

Task Specific Stuff + Loss
Neural Message Passing

Current Neighbor States

Prepare "Message"

Summarize Received Information

Next Node State

Current Node State
Each node $n$ initially has the state $h_0^n = x_n$, for all $n$.

For $t = 0, 1, \ldots, T$ do:
  
  For each node $n$, do:
  
  $a^n_t = Aggregate(h^v_t | v \in N(n))$
  
  $h^n_{t+1} = Update(a^n_t, h^n_t)$

Return $\{h^n_T\}$

$h^n_T = q_t(h^n_{t-1}, \bigcup_{n:j:n_j \rightarrow n} f_t(h^n_{t-1}, k, h^n_j))$
Graph Neural Networks: Message Passing
GNNs: Synchronous Message Passing
Graph Neural Networks: Output

- node selection
- node classification
- graph classification

https://github.com/microsoft/ptggnn/
https://github.com/microsoft/tf2-gnn/
Example: Node [Binary] Classification

\[ x_n = \sigma(w^T h^n_t + b) \]

Binary cross entropy

\[ \mathcal{L}(x_n, y_n) = y_n \cdot \log x_n + (1 - y_n) \log(1 - x_n) \]
Gated GNNs

\[ m = \sum_{n_j:n_j \rightarrow n} E_k h_{t-1}^{n_j} \]

GCNs

\[
\hat{h}_t^n = \sigma \left( \frac{1}{\text{numNeighbors} + 1} W_t \left( h_{t-1}^n + \sum_{n_j : n_j \neq n} h_{t-1}^{n_j} \right) \right)
\]

Trick 1: Backwards Edges
Expressing GGNNs as Matrix Operations
Graph Notation (2) — Adjacency Matrices

\[ A = \begin{bmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
1 & 1 & 0
\end{bmatrix} \]
Graph Notation (2) — Adjacency Matrices

\[ A = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \quad N = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \]

\[ A \cdot N = \begin{bmatrix} 0 \\ a \\ a + b \end{bmatrix} \]
Graph Notation (2) — Adjacency Matrices

\[ A_0 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \]

\[ A_1 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \]
GGNN as Matrix Operation

Node States

\[ H_t = \begin{bmatrix} h_t^{n_0} \\ \vdots \\ h_t^{n_K} \end{bmatrix} \text{ (num_nodes x D)} \]

Messages to-be sent

\[ M_t^k = E_k H_t \text{ (num_nodes x M)} \]

Received Messages

\[ R_t = \sum_k A_k M_t^k \text{ (num_nodes x M)} \]

Update \( H_{t+1} = GRU(H_t, R_t) \)

If we used a vanilla RNN instead

\[ H_{t+1} = \sigma(UH_t + WR_t) \]
Expressing GNN Matrix Operations as Code
\[ C = \text{np.einsum}(\text{`td, qd->tq`, A, B}) \quad \# \quad C_{t,q} = \sum_d A_{t,d} B_{q,d} \]

\[ D = \text{np.einsum}(\text{`abc, be, abq->cqe`, A, B, C}) \]

\[ \# \quad D_{c,q,e} = \sum_b \sum_a A_{a,b,c} B_{b,e} C_{a,b,q} \]
def GGNN(initial_node_states, adj, num_steps):
    node_states = initial_node_states  # [N, D]

    for i in range(num_steps):
        messages = {}
        for k in range(num_message_types):
            messages[k] = einsum('nd, dm->nm', node_states, edge_transform[k])  # [N, M]

        received_messages = zeros(num_nodes, M)  # [N, M]
        for k in range(num_message_types):
            received_messages += einsum('mm, nl->lm', messages[k], adj[k])

        node_states = GRU(node_states, received_messages)

    return node_states
def GGNN(initial_node_states, adj, num_steps):
    node_states = initial_node_states  # [N, D]

    for i in range(num_steps):
        messages = {}
        for k in range(num_message_types):
            messages[k] = einsum("nd, dm->nm", node_states, adj * messages[k])

        received_messages = zeros(num_nodes, M)  # [N, M]
        for k in range(num_message_types):
            received_messages += einsum("nm, nl->lm", messages[k], adj * received_messages)

        node_states = GRU(node_states, received_messages)

    return node_states
GGNN as Pseudocode

def GGNN(initial_node_states, adj, num_steps):
    node_states = initial_node_states  # [N, D]

    for i in range(num_steps):
        messages = {}
        for k in range(num_message_types):
            messages[k] = einsum('nd, dm->nm', node_states, edge_transform[k])  # [N, M]

        received_messages = zeros(num_nodes, M)  # [N, M]
        for k in range(num_message_types):
            received_messages += einsum('nm,nl->lm', messages[k], adj[k])

        node_states = GRU(node_states, received_messages)

    return node_states
def GGNN(initial_node_states, adj, num_steps):
    node_states = initial_node_states  # [N, D]

    for i in range(num_steps):
        messages = {}
        for k in range(num_message_types):
            messages[k] = einsum('nd,dm->nm', node_states, edge_transform[k])  # [N, M]

        received_messages = zeros(num_nodes, M)  # [N, M]
        for k in range(num_message_types):
            received_messages += einsum('nm,nl->lm', messages[k], adj[k])

        node_states = GRU(node_states, received_messages)

    return node_states
GGNN as Pseudocode: Sparsity

def GGNN(initial_node_states, adj, num_steps):
    node_states = initial_node_states  # [N, D]

    for i in range(num_steps):
        messages = {}
        for k in range(num_message_types):
            messages[k] = einsum('nd, dm->nm', node_states, edge_transform[k])  # [N, M]

        received_messages = zeros(num_nodes, M)  # [N, M]
        for k in range(num_message_types):
            received_messages += einsum('nm, nl->lm', messages[k], adj[k])

        node_states = GRU(node_states, received_messages)

    return node_states
Other Models as Special Cases of GNNs
Special Case 1: Convolutions (CNN)

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

Image from: https://stats.stackexchange.com/questions/235032
Special Case 2: “Deep Sets”

Set of Objects

Representing a variable-sized set of objects
Special Case 2: Self-Attention ($\approx$ Transformers)
GNNs in Software Engineering
Programs are Graphs
Assert.NotNull(clazz);

Programs as Graphs: Syntax

```
Assert.NotNull(clazz);
```
Programs as Graphs: Data Flow

```java
(x, y) = Foo();
while (x > 0)
    x = x + y;
```

- **Last Write**
- **Last Use**
- **Computed From**
Representing Program Structure as a Graph

Additional Edge Types:
• ReturnsTo

```java
int foo(int sum) {
    ...
    return x;
}
```
Representing Program Structure as a Graph

Additional Edge Types:
• ReturnsTo
• FormalArgName

```c
void foo(int sum) { ... }
```

```
b = foo(result);
      \downarrow
  sum

```
Representing Program Structure as a Graph

Entities in Programs $\rightarrow$ Nodes

Relationships among entities $\rightarrow$ Edges

```c
int SumPositive(int[] arr, int lim) {
    int sum = 0;
    for (int i = 0; i < lim; i++)
        if (arr[i] > 0)
            sum += arr[i];
    return sum;
}
```
Tasks Explored in Literature

- **VarMisuse** Allamanis *et al.* 2018, Cvitkovic *et al.* 2019, Hellendoorn *et al.* 2020

- **Method Naming** Fernandes *et al.* 2019

- **Code Generation or Repair** Brockschmidt *et al.* 2019, Yasunaga *et al.* 2020

- **Predicting Program Properties**
  - **Type Annotations in Dynamic Languages** Wei *et al.* 2020, Allamanis *et al.* 2020, Schrouff *et al.* 2019
  - **Performance Characteristics of Programs** Tomczak *et al.* 2019
Variable Misuse Task

```csharp
var clazz = classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first = classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

Allamanis et al. 2018, Cvitkovic et al. 2019, Hellendoorn et al. 2020
Initial Node Representations

Label: outFilePrefix
Variable Misuse
Initial Representation of each node

Output Representations of each Node

GNN
Output Representations of each Node

Localization objective: Pick the right node

Probabilities:
- Probability of A: 0.03
- Probability of B: 0.02
- Probability of C: 0.10
- Probability of D: 0.05
- Probability of E: 0.40
- Probability of F: 0.30
- Probability of G: 0.10

The right node is selected using the softmax function: $\text{softmax}(r^T n)$
Initial Representation of each node

Localization objective: Pick the right node

Probabilities:
- softmax($r^T n$)
  - $A$: 0.03
  - $B$: 0.02
  - $C$: 0.10
  - $D$: 0.05
  - $E$: 0.40
  - $F$: 0.30
  - $G$: 0.10
GNN Layers

Initial Representation of each node

Output Representations of each Node
<table>
<thead>
<tr>
<th>Model Family</th>
<th>Class. Accuracy ≤ 250</th>
<th>Class. Accuracy ≤ 1000</th>
<th>Loc &amp; Rep Accuracy ≤ 250</th>
<th>Loc &amp; Rep Accuracy ≤ 1000</th>
<th>Parameters</th>
<th>Training Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN¹</td>
<td>71.8%</td>
<td>70.6%</td>
<td>44.4%</td>
<td>42.5%</td>
<td>4.3M</td>
<td>31.3h</td>
</tr>
<tr>
<td>Transformer</td>
<td>75.9%</td>
<td>73.2%</td>
<td>67.7%</td>
<td>63.0%</td>
<td>3.7M</td>
<td>41.5h</td>
</tr>
<tr>
<td>GGNN</td>
<td>81.4%</td>
<td>79.2%</td>
<td>64.0%</td>
<td>60.9%</td>
<td>5.5M</td>
<td>241h</td>
</tr>
<tr>
<td>RNN Sandwich</td>
<td><strong>82.5%</strong></td>
<td><strong>81.9%</strong></td>
<td><strong>75.8%</strong></td>
<td><strong>73.8%</strong></td>
<td>12.6M</td>
<td>109h</td>
</tr>
<tr>
<td>Transformer Sandwich</td>
<td>81.1%</td>
<td>78.1%</td>
<td>74.5%</td>
<td>71.4%</td>
<td>10M</td>
<td>161h</td>
</tr>
<tr>
<td>GREAT</td>
<td>80.1%</td>
<td>76.9%</td>
<td><strong>76.4%</strong></td>
<td>73.1%</td>
<td>7.9M</td>
<td>120h</td>
</tr>
</tbody>
</table>

1: a stronger version of the model proposed in Vasic et al. (2019) (previous SOTA).
Machine Learning in Practice
Common Architecture of Deep Learning Code

Data Extraction → Compute Metadata → Convert to Tensors → Create Minibatches → Training Loop

Build ML Model → Convert to Tensors → Create Minibatches

hyperparameters
THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
Practical(?) Tips on Debugging Machine Learning Models

Model Capacity *(what can the model learn?)*
- Overtrain on a small dataset
- Synthetic data

Optimization Issues *(can we make the model learn?)*
- Look at learning curves
- Monitor gradient update ratios
- Hand-pick parameters for synthetic data

Other model “bugs” *(is the model doing what I want it to do?)*
- Generate samples from your model (if you can)
- Visualize learned representations *(e.g. embeddings, nearest neighbors)*
- Error analysis *(examples where the model is failing, most “confident” errors)*
- Simplify the problem/model
- Increase capacity, sweep hyperparameters *(e.g. increase size of $h$ in LSTM)*

See also: https://youtu.be/oMB24_ao05A
Learning Curve
Graph Neural Networks in SE Research

Neural Message Passing

Current Neighbor States

Prepare “Message”

Summarize Received Information

Current Node State

Next Node State

Programs as Graphs: Data Flow

Variable Misuse Task

@miltos1

miltos.allamanis.com