Machine Learning & Trilinos

Presented by

Chris Siefert
As per Randall Munroe…

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.

https://xkcd.com/1838/
There’s an interface to Avatar Tools in Trilinos/MueLu

- URL: https://github.com/sandialabs/avatar
- Lead developer: Philip Kegelmeyer.
- Ensembles of decision trees with a ton of bells & whistles.
- C/C++ code (with MPI support).
- Can be used either as TPL or a external package (ala Drekar).

Why ensembles of decision trees?

- Fast & efficient.
- Work well with small-to-moderate size datasets.
- Ensembles give a lot of accuracy-boosting power.
- Usually more explainable than neural networks.
MueLu/Avatar Workflow

**Offline Phase**
- App: Run lots of problems & features
- MueLu: Run lots of options per problem
- Avatar: Trains model on offline data

**Online Phase**
- App: Generates features
- MueLu: Call Avatar
- Avatar: Performs classification
- MueLu: Choose ideal options based on classification
Choose a single MueLu parameter based on 4 mesh features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Training Min/Max</th>
<th>disk1</th>
<th>disk2</th>
<th>Test disk3</th>
<th>expl</th>
<th>tubes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>1.17 / 100</td>
<td>14.9</td>
<td>5.19</td>
<td>4.08</td>
<td>8.25</td>
<td>5.11</td>
</tr>
<tr>
<td>(2)</td>
<td>1.08 / 66</td>
<td>6.88</td>
<td>2.90</td>
<td>2.69</td>
<td>3.23</td>
<td>1.65</td>
</tr>
<tr>
<td>(3)</td>
<td>4.6e-4 / 9.2e-1</td>
<td>1.4e-2</td>
<td>1.2e-1</td>
<td>2.9e-1</td>
<td>1.4e-2</td>
<td>2.9e-1</td>
</tr>
<tr>
<td>(4)</td>
<td>1.08 / 3.91</td>
<td>2.97</td>
<td>2.37</td>
<td>2.21</td>
<td>6.05</td>
<td>2.14</td>
</tr>
</tbody>
</table>

These are five test problems unrelated to the training data.
• Since expl had an out-of-bounds feature no acceptable options were found by the heuristics.
• Defaulted to a “safe” answer (namely 0).
Simple Example (3)

- Iteration counts (low = good)

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>disk1</th>
<th>disk2</th>
<th>disk3</th>
<th>expl</th>
<th>tubes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>37</td>
<td>21</td>
<td>21</td>
<td>30</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>21</td>
<td>21</td>
<td>30</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>37</td>
<td>26</td>
<td>21</td>
<td>30</td>
<td>14</td>
</tr>
<tr>
<td>fixed 0</td>
<td>57</td>
<td>28</td>
<td>24</td>
<td>30</td>
<td>13</td>
</tr>
<tr>
<td>fixed 0.01</td>
<td>29</td>
<td>26</td>
<td>31</td>
<td>26</td>
<td>14</td>
</tr>
<tr>
<td>fixed 0.025</td>
<td>22</td>
<td>21</td>
<td>21</td>
<td>22</td>
<td>14</td>
</tr>
</tbody>
</table>

- Heuristic 2 got optimal results except for out-of-bounds expl.
But what about neural networks?
Basic Neural Networks I

- Building block of a NN is a neuron (or perceptron), often drawn like this:

  \[
  \sigma(Ax + b)
  \]

- Some nonlinear function, \( \sigma \), of a matvec (with weight matrix \( A \)) and vector add.

- Feedforward Network / Multi-Layer Perceptron (MLP) is a bunch of these:
• NNs are often drawn with multiple “blocks” in each layer.

• This means each $A_i$ can be a different size.

• Lots of nonlinear $\sigma_i$ functions are available (e.g. sigmoid, ReLu, etc.).

• Done correctly, this can approximate any continuous function (similar to Stone-Weierstrass for polynomials).
These models are “trained” by choosing parameters $A_i$ and $b_i$ to satisfy some objective, called a “loss function.”

Given training data, output pairs $(d_i, \hat{y}_i)$, for $i=1...m$, you might get:

$$\min_{A_1,...,A_n,b_1,...,b_n} \sum_{i=1}^{m} ||M(d_i) - \hat{y}_i||$$

Where $M$ is the MLP model and $||.||$ is the norm of your choice.

Note that this model is *dense* or “fully connected.”
• Layers need not be dense / fully connected.

• Convolutional networks (CNNs) connect spatially nearby items (tiled over the inputs):

Corresponds to a banded matrix w/ the same values in each row, but shifted.

• Recurrent neural networks (RNN) connect temporally nearby items.
Our real-world problems don’t have fixed meshes!

For unstructured meshes, this is not going to work unless you are very, very witty

Alternative Solution: Graph Neural Networks (GNNs).

The downside here is that we’ve fixed \textit{all} of the dimensions.

- MLP: The input size is fixed.
- CNN: The number of neighbors (and their relations) is fixed.
- RNN: The temporal recurrence length is fixed (though this is less of a problem).

Image from SNL Lab News, March 2014.
Graph Neural Networks I

- Graphs have *vertices* and *edges*.
  - We will consider directed graphs.

- GNNs add...
  - Immutable attributes associated w/ each vertex + edge.
  - Mutable attributes associated with each vertex + edge.
  - Global variables with associated attributes.

Sample GNN with:
- 5 vertices (1 attribute each)
- 7 edges (2 attributes each)
- 2 globals (5 attributes each)
A GNN block has 3 update functions and 3 aggregation functions.

**Updates (trained)**
- $\phi^v$: Update vertex from its attributes, aggregated edge attributes, and globals.
- $\phi^e$: Update edge from its attributes, neighboring vertex attributes, and globals.
- $\phi^u$: Update globals from aggregated edge and vertex attributes.
- **Fixed size inputs and outputs.**

**Aggregation functions (not trained)**
- $\rho^{e\rightarrow v}$: Aggregate edge neighbors to vertex.
- $\rho^{e\rightarrow u}$: Aggregate all edges to globals.
- $\rho^{v\rightarrow u}$: Aggregate all nodes to globals.
- **Variable size inputs and fixed size outputs.**
- **Note:** Since each edge always has two neighboring nodes, node attributes are not aggregated to edges.
These can be composed in almost any order.

Matrix-as-a-Graph: Matrix-Vector Product Example ($Ax = b$)

- Let's consider a very simple example: a sparse matrix vector product.
- Remember: A matrix is a graph (off-diagonal entries are considered edges).

- Vertex $i$ attributes:
  - $a0$: 1 immutable, $A_{ii}$.
  - $a1$: 1 mutable, $x_i$ on input, $b_i$ on output.

- Edge $(i,j)$ attributes:
  - $c0$: 1 immutable, $A_{ij}$.
  - $c1$: 1 mutable, 0 on input, $A_{ij}x_j$ on output.

- Global attributes: none

- Update Edges: $c1 = \phi^e(e_{ij}, v_i, v_j) = c0(e_{ij})a1(v_j)$

- Agg Edge-to-Node: $\bar{e}_i' = \rho^{e \rightarrow v}(e_{ij} \forall j) = \sum_{j \neq i} c1(e_{ij})$

- Nodes: $a1 = \phi^v(\bar{e}_i', v_i) = a0(v_i) a1(v_i) + \bar{e}_i'$

This is a trivial example and has no parameters, but...
Matrix-as-a-Graph: Matrix-Vector Product Example (Ax = b)

• Let’s consider a very simple example: a sparse matrix vector product.
• Remember: A matrix is a graph (off-diagonal nodes are considered edges).

• Vertex $i$ attributes:
  • $a0$: 1 immutable, $A_{ii}$.
  • $a1$: 1 mutable, $x_i$ on input, $b_i$ on output.

• Edge $(i,j)$ attributes:
  • $c0$: 1 immutable, $A_{ij}$.
  • $c1$: 1 mutable, 0 on input, $A_{ij}x_j$ on output.

• Global attributes: none

• Update Edges: $c1 = \phi^e(e_{ij}, v_i, v_j) = c0(e_{ij})a1(v_j)$

• Agg Edge-to-Node: $\overline{e}_i' = \rho^{e\rightarrow v}(e_{ij} \forall j) = \sum_{j \neq i} c1(e_{ij})$

• Nodes: $a1 = \phi^v(\overline{e}_i', v_i) = a0(v_i) \cdot a1(v_i) + \overline{e}_i'$

This is a trivial example and has no parameters, but...
Matrix-as-a-Graph: Matrix-Vector Product Example (Ax = b)

• Let’s consider a very simple example: a sparse matrix vector product.
• Remember: A matrix is a graph (off-diagonal nodes are considered edges).

• Vertex $i$ attributes:
  • $a_0$: 1 immutable, $A_{ii}$.
  • $a_1$: 1 mutable, $x_i$ on input, $b_i$ on output.

• Edge $(i,j)$ attributes:
  • $c_0$: 1 immutable, $A_{ij}$.
  • $c_1$: 1 mutable, 0 on input, $A_{ij}x_j$ on output.

• Global attributes: none

• Update Edges: $c_1 = \phi^e(e_{ij}, v_i, v_j) = c_0(e_{ij})a_1(v_j)$

• Agg Edge-to-Node: $\bar{e}_i' = \rho^{e\rightarrow v}(e_{ij} \forall j) = \sum_{j \neq i} c_1(e_{ij})$

• Nodes: $a_1 = \phi^v(\bar{e}_i', v_i) = a_0(v_i) a_1(v_i) + \bar{e}_i'$

This is a trivial example and has no parameters, but...
Matrix-as-a-Graph: Matrix-Vector Product Example (Ax = b)

- Let’s consider a very simple example: a sparse matrix vector product.
- Remember: A matrix is a graph (off-diagonal nodes are considered edges).

- Vertex $i$ attributes:
  - $a_0$: 1 immutable, $A_{ii}$.
  - $a_1$: 1 mutable, $x_i$ on input, $b_i$ on output.

- Edge $(i,j)$ attributes:
  - $c_0$: 1 immutable, $A_{ij}$.
  - $c_1$: 1 mutable, 0 on input, $A_{ij} x_j$ on output.

- Global attributes: none

- Update Edges: $c_1 = \phi^e(e_{ij}, v_i, v_j) = c_0(e_{ij}) a_1(v_j)$

- Agg Edge-to-Node: $\overline{e}_i' = \rho^{e\rightarrow v}(e_{ij} \forall j) = \sum_{j \neq i} c_1(e_{ij})$

- Nodes: $a_1 = \phi^v(\overline{e}_i', v_i) = a_0(v_i) a_1(v_i) + \overline{e}_i'$

This is a trivial example and has no parameters, but...
Iterative Method: Jacobi Iteration: $x_{k+1} = x_k + \omega D^{-1}(b - Ax_k)$

- Vertex $i$ attributes:
  - $a_0, a_1$: 2 immutable, $A_{ii}, b_i$
  - $a_2$: 1 mutable, $x_i^k$ on input, $x_i^{k+1}$ on output.

- Edge $(i,j)$ attributes:
  - $c_0$: 1 immutable, $A_{ij}$.
  - $c_1$: 1 mutable, 0 on input, $A_{ij}x_i^k$ on output.

- Global attributes:
  - $d_0$: 1 immutable: $\omega$.

• Update Edges: $c_1 = \phi^e(e_{ij}, v_i, v_j) = c_0(e_{ij})a_1(v_j)$

• Agg Edge-to-Node: $\bar{e}_i' = \rho^{e \rightarrow v}(e_{ij} \forall j) = \sum_j c_1(e_{ij})$

• Nodes: $a_1 = \phi^v(\bar{e}_i', v_i) = a_2(v_i) + \frac{d_0(a_1(v_i) - \bar{e}_i')}{a_0(v_i)}$
A Trainable Jacobi Iteration: $x_{k+1} = x_k + \omega D^{-1}(b - A x_k)$

- Vertex $i$ attributes:
  - $a_0$: 1 immutable, $A_{ii}$.
  - $a_1$: 1 mutable, $b_i$ on input, $d_i$ on output.

- Edge $(i,j)$ attributes:
  - $c_0$: 1 immutable, $A_{ij}$.

- Global attributes:
  - $d_0$: 1 immutable: $\omega$.

- Special Edge aggregation:
  - Compute min, mean, sum & max of edges.

- Agg Edge-to-Node: $\bar{e}_i' = \rho^{e \rightarrow v}(e_{ij} \forall j) = [\text{min}, \text{mean}, \text{sum}, \text{max}]$

- Nodes: $a_1 = \phi^v(\bar{e}_i', v_i) = 3 \text{ Level } NN$

Goal: Choose local damping in lieu of doing an EV estimate for $\omega$
A Trainable Jacobi Iteration: $x_{k+1} = x_k + \omega D^{-1}(b - Ax_k)$

- Per-matrix loss function: Damping Factor
  \[ \|I - \omega D^{-1}A\|_2 \]

- $D=GNN(A)$ here is our trained “diagonal.”

- Toy data: 5x5 FEM Laplacians w/ varying y stretch.

- Method: 10 epochs of Adam w/ LR=0.01.

- Test Data:
  - DF Fixed $\omega = 2/3$: 0.82.
  - Trained Diagonal DF: 0.79.
  - Optimal $\omega$ DF: 0.77.

(Done in Matlab)
• Many matrix algorithms can be recast as GNNs!
  • OK. Not anything ordering dependent like Gauss-Seidel, but still.

• Once we have that, we can turn the crank on AI/ML to...
  • Choose parameters.
  • Combine multiple objective functions.
  • Create new data-driven algorithms inspired by (and informed by) old ones.

• GNN software is available in...
  • PyTorch: Geometric, Deep Graph Library.
  • TensorFlow: graph_nets and TensorFlow GNN.
  • Matlab: Deep Learning Toolbox.

• We hope to see this class of algorithms make their way into Trilinos/MueLu!

Sample GNNs Apps
• Node/Graph classification
• Computer vision (object relations)
• Natural language
• Traffic forecasting
• Recommender systems
• Molecular structure

Warning: Software support for GNNs is still a little patchy!