Frameworks

Natural Language Processing: Jordan Boyd-Graber
University of Maryland

INTRODUCTION

Slides adapted from Chris Dyer, Yoav Goldberg, Graham Neubig
Neural Nets and Language

<table>
<thead>
<tr>
<th>Language</th>
<th>Neural-Nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete, structured (graphs, trees)</td>
<td>Continuous: poor native support for structure</td>
</tr>
</tbody>
</table>

Big challenge: writing code that translates between the {discrete-structured, continuous} regimes
Why not do it yourself?

- Hard to compare with existing models
- Obscures difference between model and optimization
- Debugging has to be custom-built
- Hard to tweak model
Outline

- Computation graphs (general)
- Neural Nets in PyTorch
- Full example
Graph:

\[ \vec{x} \]
Computation Graphs

Expression

\[ \hat{x}^\top \]

- Edge: function argument / data dependency
- A node with an incoming edge is a function \( F \equiv f(u) \) edge’s tail node
- A node computes its value and the value of its derivative w.r.t each argument (edge) times a derivative \( \frac{\partial f}{\partial u} \)
Computation Graphs

Expression

\[ \hat{x}^\top A \]

graph:

\[ f(U, V) = UV \]

\[ f(u) = u^\top \]

Functions can be nullary, unary, binary, … n-ary. Often they are unary or binary.
Computation Graphs

Expression

\[ \tilde{x}^T A x \]

graph:

\[ f(M, v) = Mv \]
\[ f(U, V) = UV \]
\[ f(u) = u^T \]

Computation graphs are (usually) directed and acyclic
Computation Graphs

Expression

\( \tilde{x}^\top A x \)

graph:

\[ f(M, v) = M v \]
\[ f(U, V) = UV \]
\[ f(u) = u^\top \]

\[
\frac{\partial f(x, A)}{\partial x} = (A^\top + A)x
\]
\[
\frac{\partial f(x, A)}{\partial A} = xx^\top
\]
Computation Graphs

Expression

\[ \mathbf{x}^\top A \mathbf{x} + b \cdot \mathbf{x} + c \]

graph:

- \( f(x_1, x_2, x_3) = \sum_i x_i \)
- \( f(M, v) = Mv \)
- \( f(U, V) = UV \)
- \( f(u) = u^\top \)
- \( f(u, v) = u \cdot v \)
Computation Graphs

Expression

\[ y = \vec{x}^\top Ax + b \cdot \vec{x} + c \]

Variable names label nodes
Algorithms

- **Graph construction**
- **Forward propagation**
  - Loop over nodes in topological order
  - Compute the value of the node given its inputs
  - Given my inputs, make a prediction (i.e. “error” vs. “target output”)
- **Backward propagation**
  - Loop over the nodes in reverse topological order, starting with goal node
  - Compute derivatives of final goal node value wrt each edge’s tail node
  - How does the output change with small change to inputs?
Forward Propagation

\[ f(x_1, x_2, x_3) = \sum_i x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^\top \]

\[ f(u, v) = u \cdot v \]
Forward Propagation

\[
f(x_1, x_2, x_3) = \sum_i x_i
\]

\[
f(M, v) = Mv
\]

\[
f(U, V) = UV
\]

\[
f(u) = u^\top
\]

\[
f(u, v) = u \cdot v
\]
Forward Propagation

\[ f(x_1, x_2, x_3) = \sum_i x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]
Forward Propagation

\[ f(x_1, x_2, x_3) = \sum_i x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]
Forward Propagation

\[ f(x_1, x_2, x_3) = \sum_i x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]
Forward Propagation

\[ f(x_1, x_2, x_3) = \sum_i x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]
Forward Propagation

\[ f(x_1, x_2, x_3) = \sum_i x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]
Forward Propagation

\[ f(x_1, x_2, x_3) = \sum_{i} x_i \]

\[ x^\top A x + b \cdot x + c \]

\[ f(M, v) = Mv \]

\[ x^\top A x \]

\[ f(U, V) = UV \]

\[ f(u) = u^\top \]

\[ x^\top A \]

\[ b \cdot x \]

\[ f(u, v) = u \cdot v \]

\[ c \]
### Constructing Graphs

#### Static declaration
- Define architecture, run data through
- PROS: Optimization, hardware support
- CONS: Structured data ugly, graph language

Theano, Tensorflow

#### Dynamic declaration
- Graph implicit with data
- PROS: Native language, interleave construction/evaluation
- CONS: Slower, computation can be wasted

Chainer, Dynet, PyTorch
Constructing Graphs

<table>
<thead>
<tr>
<th>Static declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define architecture, run data through</td>
</tr>
<tr>
<td>PROS: Optimization, hardware support</td>
</tr>
<tr>
<td>CONS: Structured data ugly, graph language</td>
</tr>
</tbody>
</table>

Theano, Tensorflow

<table>
<thead>
<tr>
<th>Dynamic declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph implicit with data</td>
</tr>
<tr>
<td>PROS: Native language, interleave construction/evaluation</td>
</tr>
<tr>
<td>CONS: Slower, computation can be wasted</td>
</tr>
</tbody>
</table>

Chainer, Dynet, PyTorch
Language is Hierarchical
Dynamic Hierarchy in Language

- Language is hierarchical
  - Graph should reflect this reality
  - Traditional flow-control best for processing
- Combinatorial algorithms (e.g., dynamic programming)
- Exploit independencies to compute over a large space of operations tractably
PyTorch

- Torch: Facebook’s deep learning framework
- Nice, but written in Lua (C backend)
- Optimized to run computations on GPU
- Mature, industry-supported framework
Why GPU?
Why GPU?