Frameworks

Advanced Machine Learning for NLP
Jordan Boyd-Graber

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
Outline

- Computation graphs (general)
- Neural Nets in DyNet
- RNNs
- New functions
Computation Graphs

Expression
\( \vec{x} \)

graph:
Computation Graphs

Expression

\( \vec{x}^\top \)

- 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 \( \frac{\partial f}{\partial u} \)
Expressions can be nullary, unary, binary, ... n-ary. Often they are unary or binary.
Computation Graphs

Expression
\[ \hat{x}^T A x \]

graph:

Computation graphs are (usually) directed and acyclic
Computation Graphs

Expression

$\tilde{x}^\top A x$

graph:

\[
\begin{align*}
  f(M, v) &= Mv \\
  f(U, V) &= UV \\
  f(u) &= u^\top \\
  f(x, A) &= x^\top A x \\
  \frac{\partial f(x, A)}{\partial x} &= (A^\top + A)x \\
  \frac{\partial f(x, A)}{\partial A} &= xx^\top
\end{align*}
\]
Computation Graphs

Expression

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

graph:

- \( f(x_1, x_2, x_3) = \sum_i x_i \)
- \( f(U, V) = UV \)
- \( f(u) = u^T \)
- \( f(u, v) = u \cdot v \)

\( f(M, v) = Mv \)
Computation Graphs

Expression

\[ y = \hat{x}^T A x + b \cdot \hat{x} + c \]

graph:

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

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

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

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

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

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 (or compute an “error” with respect to a “target output”)
- Backward propagation
  - Loop over the nodes in reverse topological order starting with a final goal node
  - Compute derivatives of final goal node value with respect to each edge’s tail node
  - How does the output change if I make a small change to the 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\[ x^\top Ax + b^\top x + c \]

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

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

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

\[ f(u, v) = u \cdot v \]
## Constructing Graphs

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

Torch, Theano, Tensorflow

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

Stan, Chainer, DyNet
Constructing Graphs

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

Torch, Theano, Tensorflow

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

Stan, Chainer, DyNet
Dynamic Hierarchy in Language

- Language is hierarchical
Dynamic Hierarchy in Language

- Language is hierarchical

**Words**

**Sentences**

**Phrases**

**Documents**

- This film was completely unbelievable.
- The characters were wooden and the plot was absurd.
- That being said, I liked it.
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
DyNet

- Before DyNet:
  - AD libraries are fast and good, lack deep learning must-haves (GPUs, optimization algorithms, primitives for implementing RNNs, etc.)
  - Deep learning toolkits don’t support dynamic graphs well
- DyNet is a hybrid between a generic autodiff library and a Deep learning toolkit
  - It has the flexibility of a good AD library
  - It has most obligatory DL primitives
  - Useful for RL over structure (need this later)

---

1 Although the emphasis is dynamic operation, it can run perfectly well in “static mode”. It’s quite fast too! But if you’re happy with that, probably stick to TensorFlow/Theano/Torch.
DyNet

- C++ backend based on Eigen (like TensorFlow)
- Custom ("quirky") memory management
- A few well-hidden assumptions make the graph construction and execution very fast.
- Thin Python wrapper on C++ API