Computation Graphs

Instructor: Yoav Artzi
Computation Graphs

• The descriptive language of deep learning models
• Functional description of the required computation
• Can be instantiated to do two types of computation:
  • Forward computation
  • Backward computation
expression:
  x

graph:

A **node** is a \{tensor, matrix, vector, scalar\} value
An **edge** represents a function argument (and also data dependency). They are just pointers to nodes.

A **node** with an incoming **edge** is a **function** of that edge’s tail node.

A **node** knows how to compute its value and the value of its derivative w.r.t each argument (edge) times a derivative of an arbitrary input \( \frac{\partial F}{\partial f(u)} \).

\[
\begin{align*}
  f(u) &= u^T \\
  \frac{\partial f(u)}{\partial u} \frac{\partial F}{\partial f(u)} &= \left( \frac{\partial F}{\partial f(u)} \right)^T
\end{align*}
\]
expression:
\[ x^\top A \]

graph:

Functions can be nullary, unary, binary, ... \( n \)-ary. Often they are unary or binary.

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

\[ f(u) = u^\top \]
expression:
\[ x^T A x \]

graph:

Computation graphs are directed and acyclic (usually)
expression:
\[ x^T A x \]

graph:

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

\[ \frac{\partial f(x, A)}{\partial x} = (A^T + A)x \]
\[ \frac{\partial f(x, A)}{\partial A} = xx^T \]
expression:
\[ x^\top Ax + b \cdot x + c \]

graph:
expression:
\[ y = x^\top A x + b \cdot x + c \]

graph:

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

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

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

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

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

variable names are just labelings of 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

graph:

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

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

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

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

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

graph:

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

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

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

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

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

graph:

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

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

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

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

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

graph:

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

graph:

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

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

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

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

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

graph:

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

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

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

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

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

graph:

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

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

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

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

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

graph:

\[ f(x_1, x_2, x_3) = \sum x_i \]
\[ x^\top A x + b \cdot x + c \]
\[ f(M, v) = Mv \]
\[ f(U, V) = UV \]
\[ f(u) = u^\top \]
\[ f(u, v) = u \cdot v \]
The MLP

\[ h = \tanh(Wx + b) \]
\[ y = Vh + a \]
The MLP

\[ h = \tanh(Wx + b) \]
\[ y = Vh + a \]

\[ f(u, v) = u + v \]
\[ f(M, v) = Mv \]
\[ f(u) = \tanh(u) \]
Constructing Graphs
Two Software Models

• **Static declaration**
  
  • Phase 1: define an architecture
    (maybe with some primitive flow control like loops and conditionals)

  • Phase 2: run a bunch of data through it to train the model and/or make predictions

• **Dynamic declaration**

  • Graph is defined implicitly (e.g., using operator overloading) as the forward computation is executed
This film was completely unbelievable.

The characters were wooden and the plot was absurd.

That being said, I liked it.
Static Declaration

• Pros
  • Offline optimization/scheduling of graphs is powerful
  • Limits on operations mean better hardware support

• Cons
  • Structured data (even simple stuff like sequences), even variable-sized data, is complex
  • You effectively learn a new programming language (“the Graph Language”) and you write programs in that language to process data.

• Examples: Torch, Theano, TensorFlow
Dynamic Declaration

• **Pros**
  
  • Library is less invasive
  
  • The forward computation is written in your favorite programming language with all its features, using your favorite algorithms
  
  • Interleave construction and evaluation of the graph

• **Cons**

  • Little time for graph optimization
  
  • If the graph is static, effort can be wasted

• Examples: Chainer, *most automatic differentiation libraries*, DyNet
Dynamic Structure?

• Hierarchical structures exist in language
  • We might want to let the network reflect that hierarchy
  • Hierarchical structure is easiest to process with traditional flow-control mechanisms in your favorite languages

• Combinatorial algorithms (e.g., dynamic programming)
  • Exploit independencies to compute over a large space of operations tractably