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 $\frac{\partial F}{\partial f(u)}$.

$$f(u) = u^\top$$

$$\frac{\partial f(u)}{\partial u} \frac{\partial F}{\partial f(u)} = \left( \frac{\partial F}{\partial f(u)} \right)^\top$$
Functions can be nullary, unary, binary, ... n-ary. Often they are unary or binary.
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:

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

graph:

$$f(x_1, x_2, x_3) = \sum_i x_i$$

$$f(M, v) = Mv$$

$$f(U, V) = UV$$

$$f(u) = u^\top$$

$$f(u, v) = u \cdot v$$
expression:
\[ y = x^\top Ax + b \cdot x + c \]

graph:

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

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

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

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

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

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

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

graph:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

graph:

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

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

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

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

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

\[ x^T A x \]

\[ x^T A \]

\[ A \]

\[ b \cdot x \]

\[ b \]

\[ c \]
Forward Propagation

graph:

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

\[ h^1 = \sigma([\phi(x_l); \phi(x_r)]W^1 + b^1) \]
\[ h^2 = \sigma(h_1 W^2 + b^2) \]
\[ p = \text{softmax}(h^2 W^3 + b^3) \]
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
Batching

• Packing a few examples together has significant computational benefits

• CPU: helpful

• GPU: you get to use all the GPU cores —> world changing!

• Easy with simple networks, but gets harder as the architecture becomes more complex
The MLP

\[ h = \tanh(Wx + b) \]

\[ y = Vh + a \]

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

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

\[ f(u) = \tanh(u) \]

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

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

- Input and intermediate results become tensors — batch is another dimension!
- Do not add batch dimension to parameters! What happens then?
Hierarchical Structure

Words
- Word embedding
- LSTM over root + morphemes
- LSTM over characters

Concat

Sentences
- NP
- VP
- PP

S

Alice
- gave
- a
- message
- to
- Bob

Phrases

Documents

This film was completely unbelievable.
The characters were wooden and the plot was absurd.
That being said, I liked it.
Batching with Complex Networks

- Complex networks may include different parts with varying length (more about this later)

- It is very hard to batch complete examples this way

- But: you can still batch sub-parts across examples, so you alternate between batched and non-batched computations