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

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

graph:

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

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

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

graph:

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

graph:

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

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

$$x^\top A x + b \cdot x + c$$

diagram:

\[
\begin{align*}
  f(x_1, x_2, x_3) &= \sum_i x_i \\
  f(M, v) &= Mv \\
  f(U, V) &= UV \\
  f(u) &= u^\top \\
  f(u, v) &= u \cdot v
\end{align*}
\]
expression:
\[ y = x^\top A x + b \cdot x + c \]

graph:

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

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

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

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

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

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

graph:

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

- Two senses to processing your data in batch
  - Computing gradients for more than one example at a time to update parameters during learning
  - Processing examples together to utilize all available resources
Batching

- CPU: made of a small number of cores, so can handle some amount of work in parallel
- GPU: made of thousands of small cores, so can handle a lot of work in parallel
- Process multiple examples together to use all available cores
Batching

• Relatively easy when the network looks exactly the same for all examples

• More complex with language data: documents/sentences/words have different lengths

• Frameworks provide different methods to help common cases, but still require work on the developer side

• Key concept is broadcasting: https://pytorch.org/docs/stable/notes/broadcasting.html
The MLP

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

- Input and intermediate results become tensors — batch is another dimension!
- Do not add batch dimension of parameters! What happens then?
No batching

\[
\mathbf{X}^{(j)} = [x_1, \ldots, x_{n(j)}], \; x_i \in 1, \ldots, |\mathcal{V}|
\]
\[
a = \frac{1}{|\mathbf{X}^{(j)}|} \text{sum} \left( \phi(\mathbf{X}^{(j)}) \right)
\]
\[
\mathbf{h}_1 = \sigma(\mathbf{W}_1 a + \mathbf{b}_1)
\]
\[
\mathbf{h}_2 = \mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2
\]
\[
p = \text{softmax}(\mathbf{h}_2)
\]

Batching

\[
\mathbf{X}'^{(j)} = [x'_1, \ldots, x'_M], \; x'_i = \begin{cases} x_i & i \leq n(j) \\ 0 & \text{else} \end{cases}
\]
\[
\mathbf{B} = [\mathbf{X}'^{(j)}, \ldots, \mathbf{X}'^{(j+B)}]
\]
\[
a = \left[ \frac{1}{n(j)}, \ldots, \frac{1}{n(j+B)} \right] \text{sum} \left( \phi(\mathbf{B}) \right)
\]
\[
\mathbf{h}_1 = \sigma(\mathbf{W}_1 a + \mathbf{b}_1)
\]
\[
\mathbf{h}_2 = \mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2
\]
\[
p = \text{softmax}(\mathbf{h}_2)
\]
Hierarchical Structure

Words

Sentences

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 complex to batch complete examples this way

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