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

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

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

graph:

\[
\begin{align*}
  f(M, v) &= Mv \\
  f(U, V) &= UV \\
  f(u) &= u^T
\end{align*}
\]

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

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

graph:

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

\[ f(\mathbf{M}, \mathbf{v}) = \mathbf{M} \mathbf{v} \]

\[ f(\mathbf{U}, \mathbf{V}) = \mathbf{U} \mathbf{V} \]

\[ f(\mathbf{u}) = \mathbf{u}^\top \]

\[ f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v} \]

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

graph:

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

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

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

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

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

\[ b \cdot x \]

\[ c \]
Forward Propagation

graph:

\[ f(x_1, x_2, x_3) = \sum x_i \]
\[ x^\top Ax + b \cdot x + c \]

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

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

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

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

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

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

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

- 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**

```
Alice gave a message to Bob
```

**Phrases**

```
NP The hungry cat
```

**Documents**

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