Frameworks

Advanced Machine Learning for NLP

Jordan Boyd-Graber

INTRODUCTION

Slides adapted from Chris Dyer, Yoav Goldberg, Graham Neubig

Neural Nets and Language

Language

Discrete, structured (graphs, trees)

Neural-Nets

Continuous: poor native support for structure

Big challenge: writing code that translates between the {discrete-structured, continuous} regimes

Outline

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

Expression

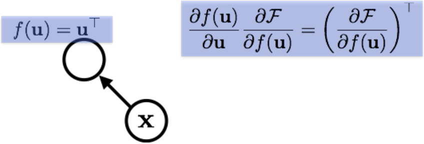
$\vec{x}$

graph:

Computation Graphs

Expression

$\vec{x}^\top$

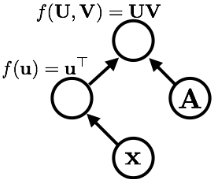


- Edge: function argument / data dependency
- A node with an incoming edge is a function $F \equiv f(u)$ edge's tail node
- A node computes its value and the value of its derivative w.r.t each argument (edge) times a derivative $\frac{\partial f}{\partial u}$

Expression

$\vec{x}^\top A$

graph:



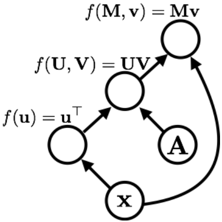
Functions can be nullary, unary, binary, . . . n-ary. Often they are unary or binary.

Computation Graphs

Expression

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

graph:

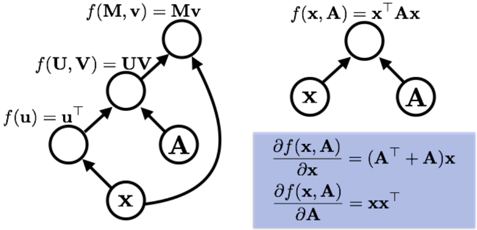


Computation graphs are (usually) directed and acyclic

Expression

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

graph:

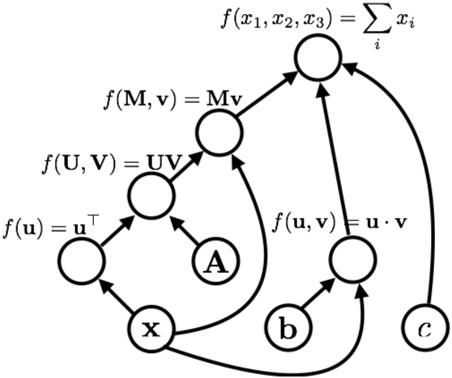


Computation Graphs

Expression

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

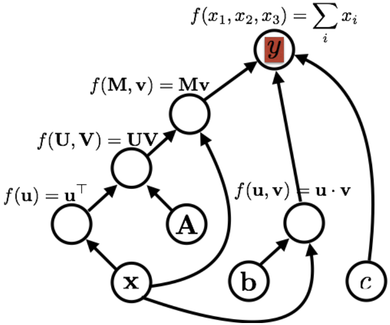
graph:



Expression

$y = \vec{x}^{\top} A x + b \cdot \vec{x} + c$

graph:

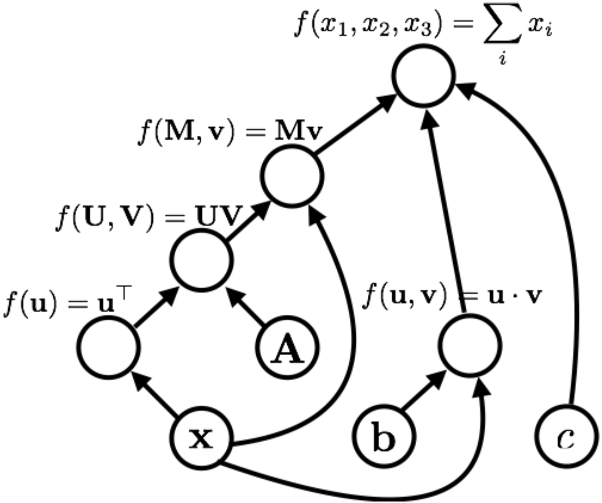


Variable names label nodes

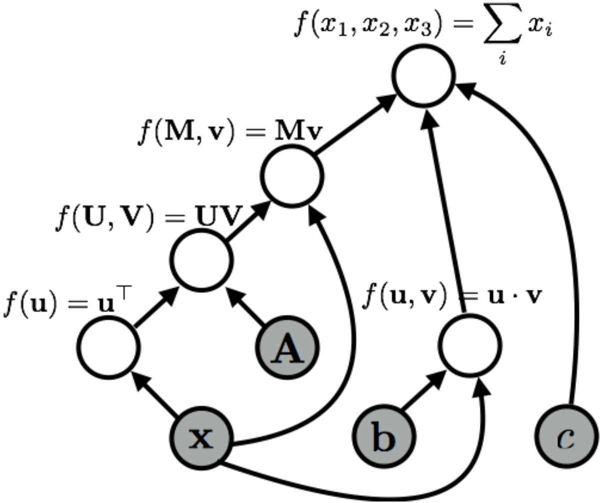
Algorithms

- 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 edgeâĂŹs tail node
  - How does the output change if I make a small change to the inputs?

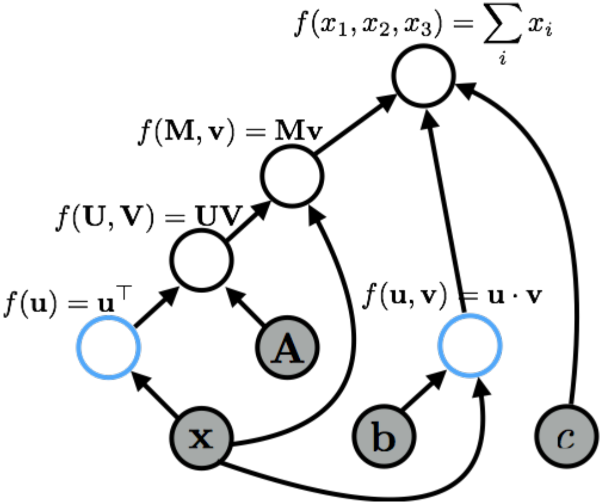
Forward Propagation



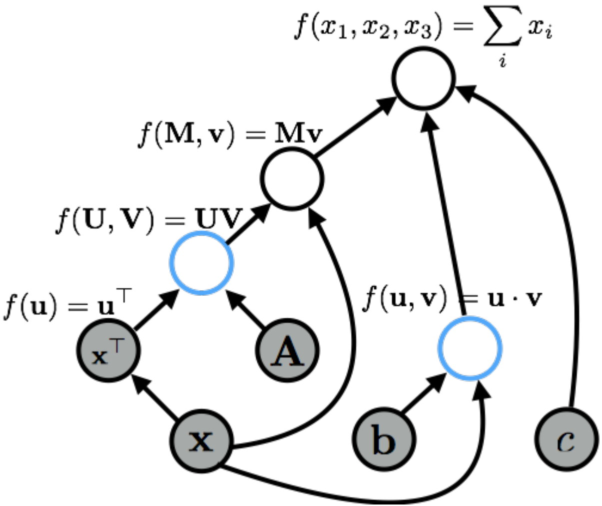
Forward Propagation



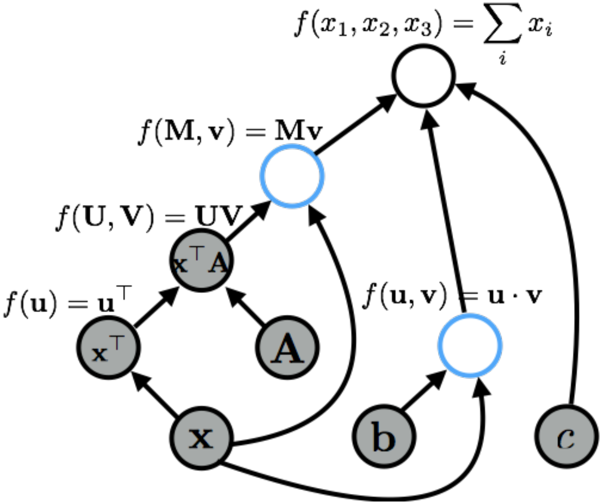
Forward Propagation



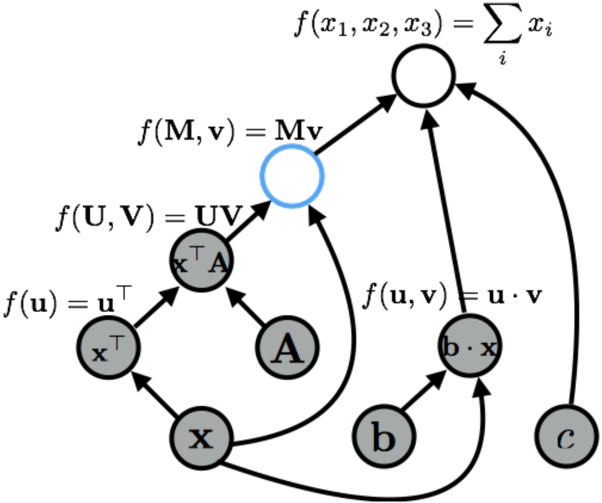
Forward Propagation



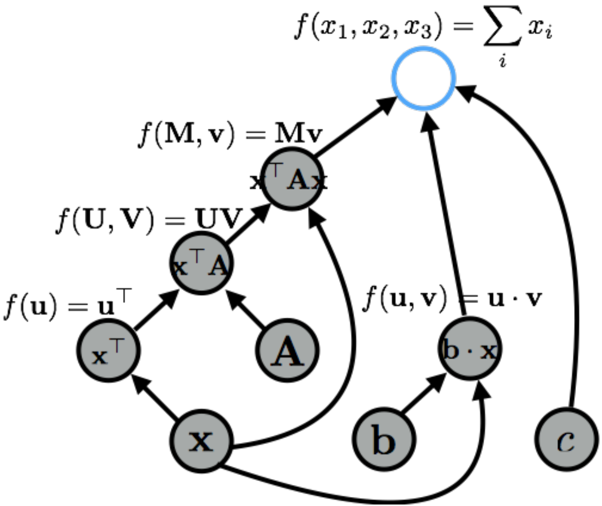
Forward Propagation



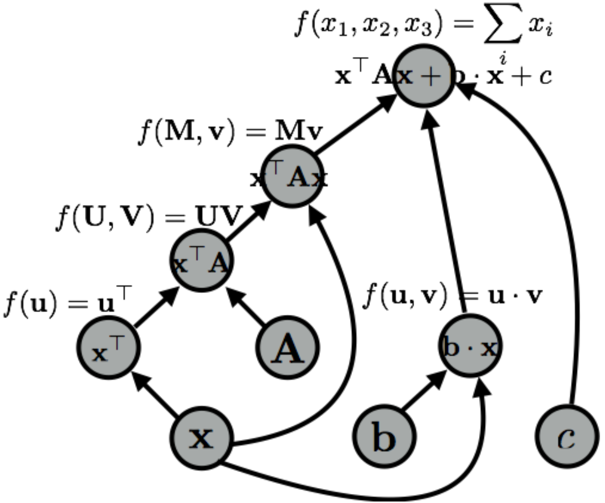
Forward Propagation



Forward Propagation



Forward Propagation



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

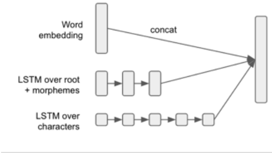
Dynamic Hierarchy in Language

- Language is hierarchical

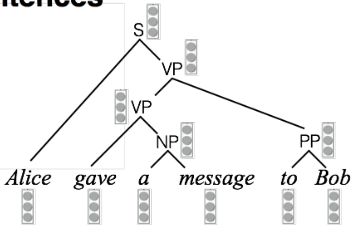
Dynamic Hierarchy in Language

- Language is hierarchical

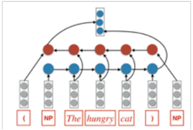
Words



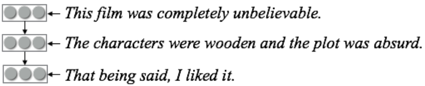
Sentences



Phrases



Documents



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[1]
  - 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