Lecture 9: Computation Graphs and DyNet

2018-2019 Spring
Computational Graphs

• Representing our neural net equations as a graph

  • Source nodes: inputs
  • Interior nodes: operations

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \ (\text{input}) \]
Computational Graphs

- Representing our neural net equations as a graph
  - Source nodes: inputs
  - Interior nodes: operations
  - Edges pass along result of the operation

\[
\begin{align*}
    s &= u^T h \\
    h &= f(z) \\
    z &= Wx + b \\
    x &\text{ (input)}
\end{align*}
\]
Computational Graphs

- Representing our neural net equations as a graph

\[
s = u^T h \\
h = f(z) \\
z = c + b
\]

“Forward Propagation”

How to train the model?
Backpropagation

- Go backwards along edges
  - Pass along gradients

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \text{ (input)} \]
Backpropagation: Single Node

- Node receives an “upstream gradient”
- Goal is to pass on the correct “downstream gradient”

\[ h = f(z) \]
Backpropagation: Single Node

- Each node has a **local gradient**
- The gradient of its output with respect to its input

\[ h = f(z) \]

Can you see the chain rule here?

\[ \frac{\partial s}{\partial z} \quad \frac{\partial h}{\partial z} \quad \frac{\partial s}{\partial h} \]

Downstream gradient  Local gradient  Upstream gradient
Backpropagation: Single Node

- Each node has a **local gradient**
- The gradient of its output with respect to its input
Backpropagation: Single Node

- Each node has a **local gradient**
- The gradient of its output with respect to its input

\[
h = f(z)
\]

- \([\text{downstream gradient}] = [\text{upstream gradient}] \times [\text{local gradient}]\)
Backpropagation: Single Node

- What about nodes with multiple inputs?

\[ z = Wx \]
Backpropagation: Single Node

- Multiple inputs -> multiple local gradients

\[ z = W x \]

\[
\begin{align*}
\frac{\partial s}{\partial W} &= \frac{\partial s}{\partial z} \frac{\partial z}{\partial W} \\
\frac{\partial s}{\partial x} &= \frac{\partial s}{\partial z} \frac{\partial z}{\partial x}
\end{align*}
\]
An Example

\[ f(x, y, z) = (x + y) \max(y, z) \]

\[ x = 1, \ y = 2, \ z = 0 \]
An Example

\[ f(x, y, z) = (x + y) \max(y, z) \]
\[ x = 1, y = 2, z = 0 \]

Forward prop steps

\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

\[ x \]
\[ y \]
\[ z \]
An Example

Forward prop steps

\[ a = x + y \]

\[ b = \max(y, z) \]

\[ f = ab \]

\[
f(x, y, z) = (x + y) \max(y, z)
\]

\[ x = 1, y = 2, z = 0 \]
An Example

Forward prop steps
\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

Local gradients
\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]

\[
f(x, y, z) = (x + y) \max(y, z)
\]
\[ x = 1, y = 2, z = 0 \]
An Example

Forward prop steps
\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

Local gradients
\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]
\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]

\[ f(x, y, z) = (x + y) \max(y, z) \]
\[ x = 1, y = 2, z = 0 \]
An Example

Forward prop steps
\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

Local gradients
\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]
\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]
\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]

\[
 f(x, y, z) = (x + y) \max(y, z) \\
x = 1, y = 2, z = 0
\]
An Example

Forward prop steps

\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

Local gradients

\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]
\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]
\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]

\[ f(x, y, z) = (x + y) \max(y, z) \]
\[ x = 1, y = 2, z = 0 \]
An Example

Forward prop steps
\[ a = x + y \]
\[ b = \text{max}(y, z) \]
\[ f = ab \]

Local gradients
\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]
\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]
\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]

\[ f(x, y, z) = (x + y) \text{max}(y, z) \]
\[ x = 1, y = 2, z = 0 \]

upstream * local = downstream
An Example

Forward prop steps
\[
a = x + y \\
b = \max(y, z) \\
f = ab
\]

Local gradients
\[
\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \\
\frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \\
\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3
\]

\[
f(x, y, z) = (x + y) \max(y, z) \\
x = 1, y = 2, z = 0
\]
An Example

Forward prop steps

\[ a = x + y \]
\[ b = \max(y, z) \]
\[ f = ab \]

Local gradients

\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]
\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]
\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]

upstream * local = downstream
An Example

Forward prop steps

\[ a = x + y \]

\[ b = \max(y, z) \]

\[ f = ab \]

Local gradients

\[ \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1 \]

\[ \frac{\partial b}{\partial y} = 1(y > z) = 1 \quad \frac{\partial b}{\partial z} = 1(z > y) = 0 \]

\[ \frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3 \]
Efficiency: compute all gradients at once

• Incorrect way of doing backprop:
  • First compute $\frac{\partial s}{\partial b}$

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \] (input)
Efficiency: compute all gradients at once

- Incorrect way of doing backprop:
  - First compute $\frac{\partial s}{\partial b}$
  - Then independently compute $\frac{\partial s}{\partial W}$
  - Duplicated computation!

$s = u^T h$
$h = f(z)$
$z = Wx + b$
x (input)
Efficiency: compute all gradients at once

- Correct way:
  - Compute all the gradients at once
  - Analogous to using $\delta$ when we computed gradients by hand

\[
s = u^T h
\]
\[
h = f(z)
\]
\[
z = Wx + b
\]
\[
x \quad \text{(input)}
\]
Summary

- Backpropagation: recursively apply the chain rule along computational graph
  - \([\text{downstream gradient}] = [\text{upstream gradient}] \times [\text{local gradient}]\)
- Forward pass: compute results of operation and save intermediate values
- Backward: apply chain rule to compute gradient
Neural Network Frameworks

Static Frameworks

Theano
Caffe
mxnet

Dynamic Frameworks (Recommended!)

\( \partial y/\text{net} \)
Chainer

TensorFlow

+Gluon
+Fold
Basic Process in Dynamic Neural Network Frameworks

• Create a model

• For each example
  
  • **create a graph** that represents the computation you want
  
  • **calculate the result** of that computation
  
  • if training, perform **back propagation and update**
DyNet

- Examples in this class will be in DyNet:
  - **intuitive**, program like you think (c.f. TensorFlow, Theano)
  - **fast for complicated networks** on CPU (c.f. autodiff libraries, Chainer, PyTorch)
  - has **nice features to make efficient implementation easier** (automatic batching)
Computation Graph and Expressions

```python
import dyten as dy

dy.renew_cg()  # create a new computation graph

v1 = dy.inputVector([1, 2, 3, 4])

v2 = dy.inputVector([5, 6, 7, 8])
# v1 and v2 are expressions

v3 = v1 + v2
v4 = v3 * 2
v5 = v1 + 1

v6 = dy.concatenate([v1, v2, v3, v5])

print v6
print v6.npvalue()
```
import dynet as dy

dy.renew_cg()  # create a new computation graph

v1 = dy.inputVector([1,2,3,4])
v2 = dy.inputVector([5,6,7,8])
# v1 and v2 are expressions

v3 = v1 + v2
v4 = v3 * 2
v5 = v1 + 1

v6 = dy.concatenate([v1,v2,v3,v5])

print v6  # expression 5/1
print v6.npvalue()
import dynet as dy

dy.renew_cg()  # create a new computation graph

v1 = dy.inputVector([1, 2, 3, 4])
v2 = dy.inputVector([5, 6, 7, 8])
# v1 and v2 are expressions

v3 = v1 + v2
v4 = v3 * 2
v5 = v1 + 1

v6 = dy.concatenate([v1, v2, v3, v5])

print v6
print v6.npvalue()
array([  1.,   2.,   3.,   4.,   2.,   4.,   6.,   8.,   4.,   8.,  12.,  16.])
Computation Graph and Expressions

• Create basic expressions.

• Combine them using \textit{operations}.

• Expressions represent \textit{symbolic computations}.

• Use:
  .\texttt{value()}
  .\texttt{npvalue()}
  .\texttt{scalar\_value()}
  .\texttt{vec\_value()}
  .\texttt{forward()}

to perform actual computation.
Model and Parameters

- **Parameters** are the things that we optimize over (vectors, matrices).
- **Model** is a collection of parameters.
- Parameters **out-live** the computation graph.
model = dy.Model()

pW = model.add_parameters((20, 4))
pb = model.add_parameters(20)

dy.renew_cg()
x = dy.inputVector([1, 2, 3, 4])
W = dy.parameter(pW)  # convert params to expression
b = dy.parameter(pb)  # and add to the graph

y = W * x + b
model = dy.Model()

pW = model.add_parameters((4,4))

pW2 = model.add_parameters((4,4), init=dy.GlorotInitializer())

pW3 = model.add_parameters((4,4), init=dy.NormalInitializer(0,1))

pW4 = model.parameters_from_numpu(np.eye(4))
Trainers and Backdrop

• Initialize a **Trainer** with a given model.

• Compute gradients by calling `expr.backward()` from a scalar node.

• Call `trainer.update()` to update the model parameters using the gradients.
Trainers and Backdrop

```python
model = dy.Model()

trainer = dy.SimpleSGDTrainer(model)

p_v = model.add_parameters(10)

for i in xrange(10):
    dy.renew_cg()

    v = dy.parameter(p_v)
    v2 = dy.dot_product(v, v)
    v2.forward()

    v2.backward()  # compute gradients

trainer.update()
```
Trainers and Backdrop

model = dy.Model()

trainer = dy.SimpleSGDTrainer(model)

p_v = model.add_parameters(10)

for i in xrange(10):
    dy.renew_cg()
    v = dy.parameter(p_v)
    v2 = dy.dot_product(v, v)
    v2.forward()
    v2.backward()  # compute gradients

trainer.update()
Training with DyNet

• Create model, add parameters, create trainer.

• For each training example:
  • create computation graph for the loss
  • run forward (compute the loss)
  • run backward (compute the gradients)
  • update parameters
Example Implementation
(in DyNet)

Predicting the next word in a sentence…
Are These Sentences OK?

• Jane went to the store.
• store to Jane went the.
• Jane went store.
• Jane goed to the store.
• The store went to Jane.
• The food truck went to Jane.

How to check whether these sentences are ok or not?
Calculating the Probability of a Sentence

\[ P(X) = \prod_{i=1}^{I} P(x_i \mid x_1, \ldots, x_{i-1}) \]

The big problem: How do we predict

\[ P(x_i \mid x_1, \ldots, x_{i-1}) \]
Review: Count-based Language Models
Count-based Language Models

- Count up the frequency and divide:

\[
P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}
\]

- Add smoothing, to deal with zero counts:

\[
P(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) = \lambda P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) + (1 - \lambda) P(x_i \mid x_{1-n+2}, \ldots, x_{i-1})
\]

- Modified Kneser-Ney smoothing
Continuous Bag of Words (CBOW)

$$I + hate + this + movie = W + bias = scores$$
A Note: “Lookup”

- Lookup can be viewed as “grabbing” a single vector from a big matrix of word embeddings
  
  \[
  \text{lookup}(2) \begin{pmatrix}
  0 \\
  0 \\
  1 \\
  0 \\
  0 \\
  \end{pmatrix}
  \]

- Similarly, can be viewed as multiplying by a “one-hot” vector
  
  \[
  \begin{pmatrix}
  0 \\
  0 \\
  1 \\
  0 \\
  0 \\
  \end{pmatrix} \times \begin{pmatrix}
  \end{pmatrix}
  \]

- Former tends to be faster
Deep CBOW

I hate this movie

+ + +

W + tanh(W₁*h + b₁) + tanh(W₂*h + b₂) = bias scores

Deep CBOW
Neural Language Models

• (See Bengio et al. 2004)

giving a

lookup lookup

tanh(W₁*h + b₁)

W + bias scores = softmax probs
Where is Strength Shared?

**Word embeddings:**

- Similar input words get similar vectors.
- Similar output words get similar rows in the softmax matrix.
- Similar contexts get similar hidden states.

**Equation:**

$$\text{tanh}(W_1 \ast h + b_1)$$

**Diagram:**

- Giving
- A
- Lookup
- Similar output words get similar rows in the softmax matrix
- Similar contexts get similar hidden states

**Softmax Calculation:**

$$W + \text{bias} = \text{scores}$$

$$\text{softmax} \rightarrow \text{probs}$$
Training a Model

• **Reminder:** to train, we calculate a “loss function” (a measure of how bad our predictions are), and move the parameters to reduce the loss.

• The most common loss function for probabilistic models is “negative log likelihood”

If element 3 (or zero-indexed, 2) is the correct answer:

\[
\begin{pmatrix}
0.002 \\
0.003 \\
0.329 \\
0.444 \\
0.090 \\
\ldots
\end{pmatrix}
\]

\[-\log 1.112\]
Parameter Update

• Back propagation allows us to calculate the derivative of the loss with respect to the parameters
  \[ \frac{\partial l}{\partial \theta} \]

• Simple stochastic gradient descent optimizes parameters according to the following rule
  \[ \theta \leftarrow \theta - \alpha \frac{\partial l}{\partial \theta} \]
Choosing a Vocabulary
Unknown Words

- Necessity for UNK words
  - We won’t have all the words in the world in training data
  - Larger vocabularies require more memory and computation time

- Common ways:
  - Frequency threshold (usually UNK <= 1)
  - Rank threshold
REFERENCES

- Introduction to Neural Networks, Philipp Koehn, 3 October 2017
- Artificial Neural Networks and Deep Learning, Christian Borgelt, University of Konstanz, Germany
- Natural Language Processing with Deep Learning, Richard Socher, Kevin Clark, Stanford University, 2017
- Richard Socher, Neural Networks, 2018