CS11-747 Neural Networks for NLP

Intro/
Why Neural Nets for NLP?

Graham Neubig

Carnegie Mellon University
Language Technologies Institute

Site
https://phontron.com/class/nn4nlp2017/
Neural Networks:
A Tool for Doing Hard Things
An Example Prediction Problem: Sentence Classification

I hate this movie

I love this movie
A First Try: Bag of Words (BOW)
Build It, Break It

I don’t love this movie

There’s nothing I don’t love about this movie

Build It, Break It
The Language Edition
https://bibinlp.umiacs.umd.edu
Combination Features

• Does it contain “don’t” and “love”?

• Does it contain “don’t”, “i”, “love”, and “nothing”?
Basic Idea of Neural Networks (for NLP Prediction Tasks)

I
lookup

hate
lookup

this
lookup

movie
lookup

some complicated function to extract combination features (neural net)

scores

probs

softmax
Computation Graphs
The Lingua Franca of Neural Nets
expression:
   x

graph:

A **node** is a \{tensor, matrix, vector, scalar\} value
   x
An **edge** represents a function argument (and also an 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:

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

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

graph:

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

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

\[ x^\top Ax + b \cdot x + c \]

dependent nodes:

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

\[ y = x^\top A x + b \cdot x + c \]

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 (1)

- Graph construction
- Forward propagation
  - In topological order, compute the value of the node given its 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\[ x^T A x \]

\[ x^T A \]

\[ A \]

\[ b \cdot x \]

\[ b \]

\[ x \]

\[ c \]
Forward Propagation

graph:

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

\[ f(M, v) = Mv \]
\[ f(U, V) = UV \]

\[ f(u) = u^\top \]
\[ f(u, v) = u \cdot v \]
Algorithms (2)

- **Back-propagation:**
  - Process examples in reverse topological order
  - Calculate the derivatives of the parameters with respect to the final value
    (This is usually a “loss function”, a value we want to minimize)

- **Parameter update:**
  - Move the parameters in the direction of this derivative

  \[ W \leftarrow \alpha \times \frac{dL}{dW} \]
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)
import dy 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()
Computation Graph and Expressions

```python
import 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.npvalue())
```

import 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 *operations*.
- Expressions represent *symbolic computations*.
- Use:
  - `.value()`
  - `.npvalue()`
  - `.scalar_value()`
  - `.vec_value()`
  - `.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 and Parameters

```python
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

ty = W * x + b
```
Parameter Initialization

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 \texttt{expr.backward()} from a scalar node.
- Call \texttt{trainer.update()} to update the model parameters using the gradients.
Trainers and Backdrop

```python
define = 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

```python
model = dy.Model()
trainer = dy.SimpleSGDTrainer(model,...)
p_v = model.params
trainer = dy.MomentumSGDTrainer(model,...)
for i in x:
    trainer = dy.AdagradTrainer(model,...)
    dy.renew
    v = dy.DeltaVector()
    v2 = dy.DeltaVector()
    v2.for_each
    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)
Bag of Words (BOW)
Continuous Bag of Words (CBOW)
Deep CBOW

I $\rightarrow$ hate $\rightarrow$ this $\rightarrow$ movie

$= \begin{cases} 
\text{tanh}(W_1h + b_1) & \text{tanh}(W_2h + b_2) 
\end{cases}
$

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