CS11-747 Neural Networks for NLP

Intro/
Why Neural Nets for NLP?

Graham Neubig

Site
https://phontron.com/class/nn4nlp2017/
Language is Hard!
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.
Engineering Solutions

- 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.

Create a grammar of the language

Consider morphology and exceptions

Semantic categories, preferences

And their exceptions
Are These Sentences OK?

• ジェインは店へ行った。
• は店行ったジェインは。
• ジェインは店へ行った。
• 店はジェインへ行った。
• 屋台はジェインのところへ行った。
Phenomena to Handle

- Morphology
- Syntax
- Semantics/World Knowledge
- Discourse
- Pragmatics
- Multilinguality
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)

I hate this movie

lookup + lookup + lookup + lookup + bias = scores

softmax

probs
I don’t love this movie

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

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 hate this movie

lookup lookup lookup lookup

some complicated function to extract combination features (neural net)

scores

softmax

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

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

\[ \frac{\partial f(u)}{\partial u} \frac{\partial F}{\partial f(u)} = \left( \frac{\partial F}{\partial f(u)} \right)^\top \]
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 (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:
\[ \mathbf{x}^\top \mathbf{A} \mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c \]

graph:

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

graph:

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

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:

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

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

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

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

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

graph:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\[ x^T A \]

\[ A \]

\[ b \cdot x \]

\[ x \]

\[ b \]

\[ c \]
Forward Propagation

graph:

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

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

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

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

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

\[ x^T A \]

\[ x^T A \]

\[ b \cdot x \]

\[ c \]
Forward Propagation

graph:

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

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

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

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

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

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

\[ W' = \alpha \times \frac{dl}{dW} \]
A Concrete Example
Neural Network Frameworks

Static Frameworks
- theano
- Caffe
- mxnet

Dynamic Frameworks (Recommended!)
- PyTorch
- Chainer
- dyNet
- 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)
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()
import dyne 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 dy"net 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

- 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
code
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
```
Parameter Initialization

```python
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

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

```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()
```

- `dy.SimpleSGDTrainer(model,...)`
- `dy.MomentumSGDTrainer(model,...)`
- `dy.AdagradTrainer(model,...)`
- `dy.AdadeltaTrainer(model,...)`
- `dy.AdamTrainer(model,...)`
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)

I hate this movie

lookup lookup lookup lookup + bias scores = probs

softmax
Continuous Bag of Words (CBOW)

I hate this movie

\[ \text{scores} = \text{bias} \]

\[ \text{lookup} + \text{lookup} + \text{lookup} + \text{lookup} \]
Deep CBOW

Deep CBOW

I

hate

this

movie

bias

scores
Class Format/Structure
Class Format

- **Reading**: Before the class

- **Quiz**: Simple questions about the required reading (should be easy)

- **Summary/Elaboration/Questions**: Instructor or TAs will summarize the material, elaborate on details, and field questions

- **Code Walk**: The TAs (or instructor) will walk through some demonstration code
Assignments

• Course is group (2-3) assignment/project based

• **Assignment 1:** Survey the field and implement a baseline model

• **Assignment 2:** Re-implement and reproduce results from a state-of-the-art model

• **Project:** Perform a unique research project that either (1) improves on state-of-the-art, or (2) applies neural net models to a unique task
Instructors/Office Hours

- **Instructor:** Graham Neubig  
  (Mon., 4:00-5:00PM GHC5409)

- **TAs:**
  - Zhengzhong (Hector) Liu (Mon. 1:00-2:00PM, GHC5517)
  - Xuezhe (Max) Ma (Tue. 12:00-1:00PM, GHC5517)
  - Daniel Clothiaux (Fri. 9:00-10:00AM, GHC5505)

- **Piazza:** [http://piazza.com/cmu/fall2017/cs11747/home](http://piazza.com/cmu/fall2017/cs11747/home)
Class Plan
Section 1: Models of Words

- Word representations using context
- Word representations using word form
- Speed tricks for neural networks
Section 2: Models of Sentences

- Bag of words, bag of n-grams, convolutional nets
- Recurrent neural networks and variations
- Applications of sentence modeling
Sec.3: Sequence-to-sequence Models

I hate this movie

- Encoder decoder models
- Attentional models
Section 4: Structured Prediction Models

- Structured perceptron, structured max margin
- Conditional random fields
Section 5:
Models of Tree Structure

• Shift reduce, minimum spanning tree parsing

• Tree structured compositions

• Models of graph structures
Section 6: Advanced Learning Techniques

- Variational Auto-encoders
- Adversarial Networks
- Marginal Likelihood, Reinforcement Learning
- Semi-supervised and Unsupervised Learning
Section 7: Neural Networks and Knowledge

- Learning from/for Relational Databases
- Interfacing with Relational Databases
- Machine Reading Models
- Reasoning with Neural Nets
Section 8: Multi-task and Multilingual Learning

- Multi-task Learning Models
- Multilingual Learning of Representations
- Universal Analysis Models

I hate this movie

この映画が嫌い

PRP VB DT NN
Section 9: Advanced Search Techniques

- Beam search and its variants
- A* search
Any Questions?