# intel AI DEVCON 2018

# Semantic Parsing: Natural Language Understanding in Python

Milind S. Pandit
Machine Learning Solutions Manager, AI Products Group

# THINGS TO KEEP IN MIND

1. You'll get access to the information covered in this session after the conference

2. Visit the Intel® AI Academy for additional resources, training materials and videos related to today's presentation.
software.intel.com/AI

3. Try the Intel Distribution for Python! (https://software.intel.com/en-us/distribution-for-python)

4. Check out more examples of Intel AI/Movidius NCS/Intel AI DevCloud in action on DevMesh – Intel's Developer Network
https://devmesh.intel.com/

# REFERENCES


- Stanford CS 224U: Natural Language Understanding
- Liang, Percy and Potts, Christopher, Bringing machine learning and compositional semantics together. *Annual Review of Linguistics* 1(1): 355–376, 2014.
- Original SippyCup Github Repository
- Fork for this class: https://github.com/mspandit/sippycup

# SEMANTIC PARSING

A computation which takes a linguistic expression and returns as output a structured, machine-interpretable representation of its meaning, known as the *semantic representation*

https://github.com/mspandit/sippycup

# EXAMPLE: QUESTION ANSWERING APPLICATION

"How tall is Obama?"

→

(/person/height /m/02mjmr)

https://github.com/mspandit/sippycup

# EXAMPLE: QUESTION ANSWERING APPLICATION

Linguistic Expression "How tall is Obama?" → Semantic Parser → Semantic Representation (/person/height /m/02mjmr) → Executor → Denotation 1.85 m

https://github.com/mspandit/sippycup

# WHY SEMANTIC PARSING IS HARD

- Multiple linguistic expressions can have the same meaning
  - Example: “nyc population,” “How many people live in New York City?”
  - Canonicalization: Same meaning → Same semantic representation
- A single linguistic expression can have multiple meanings—depending on the context
  - Example: “How big is New York?” (area, population) X (city, state)
  - Ambiguity resolution: Different meanings → Different semantic representations

# WHY SEMANTIC PARSING IS HARD

- Linguistic expressions can be messy with typos, misspellings, loose syntax: “where r u”
- Internationalization compounds the problem
- Scale of the problem demands machine learning

# NATURAL LANGUAGE ARITHMETIC

# THE PROBLEM

- Interpret natural language arithmetic expressions
  - "one plus one"
  - "minus three minus two" (lexical ambiguity)
  - "three plus three minus two"
  - "two times two plus three" (syntactic ambiguity)
- Small, closed vocabulary
- Limited variety of syntactic structures

https://github.com/mspandit/sippycup

# SEMANTIC REPRESENTATION: BINARY EXPRESSION TREES

| | |
|---|---|
| one plus one | `('+', 1, 1)` |
| minus three minus two | `('-', ('~', 3), 2)` |
| three plus three minus two | `('-', ('+', 3, 3), 2)` |
| two times two plus three | `('+', ('*', 2, 2), 3)` |

# executor.py

# CONSTITUENCY STRUCTURE

How we group words into larger and larger phrases.

# SYNTACTIC PARSING

Build a tree structure (a *parse*) over the input which describes its constituency structure. Assign *categories* to each word and phrase.

# EXAMPLE INPUT AND DENOTATION

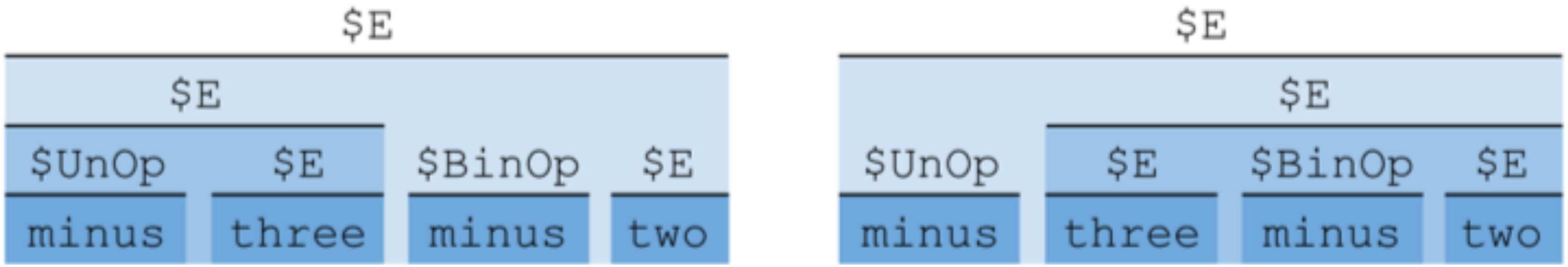
| | | | |
|---|---|---|---|
| minus three minus two | | | -5 |
| minus three minus two | | | -1 |

# EXAMPLE WITH WORDS GROUPED

| | | | |
|---|---|---|---|
| minus three minus two | ((minus three) minus two) | | -5 |
| minus three minus two | (minus (three minus two)) | | -1 |

# EXAMPLE WITH CATEGORIES ASSIGNED

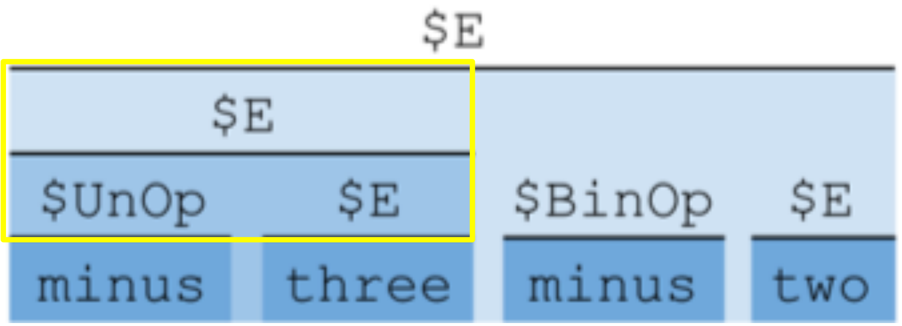
| | | | |
|---|---|---|---|
| minus three minus two | ((minus three) minus two) | ($E ($E ($UnOp minus) ($E three)) ($BinOp minus) ($E two)) | -5 |
| minus three minus two | (minus (three minus two)) | ($E ($UnOp minus) ($E ($E three) ($BinOp minus) ($E two))) | -1 |

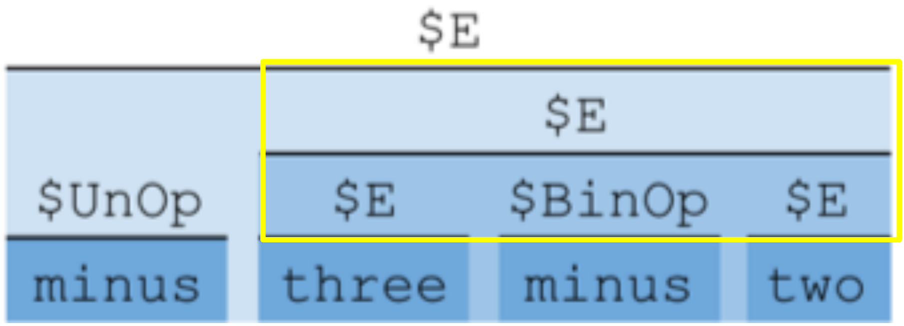



| Category | Definition |
|---|---|
| $E | Expression |
| $UnOp | Unary Operator |
| $BinOp | Binary Operator |

# EXAMPLE WITH LOCAL SUBTREES HIGHLIGHTED

| | | | |
|---|---|---|---|
| minus three minus two | ((minus three) minus two) | ($E ($E ($UnOp minus) ($E three)) ($BinOp minus) ($E two)) | -5 |
| minus three minus two | (minus (three minus two)) | ($E ($UnOp minus) ($E ($E three) ($BinOp minus) ($E two))) | -1 |




| Category | Definition |
|---|---|
| $E | Expression |
| $UnOp | Unary Operator |
| $BinOp | Binary Operator |

https://github.com/mspandit/sippycup

# PARTIAL CONTEXT FREE GRAMMAR RULES

| Left Hand Side | Right Hand Side |
| --- | --- |
| | |
| $E | two |
| $E | three |
| | |
| $UnOp | minus |
| $BinOp | minus |
| | |
| | |
| $E | $UnOp $E |
| $E | $E $BinOp $E |

# COMPLETE CONTEXT FREE GRAMMAR RULES

| Left Hand Side | Right Hand Side |
|---|---|
| $E | one |
| $E | two |
| $E | three |
| $E | four |
| $UnOp | minus |
| $BinOp | minus |
| $BinOp | plus |
| $BinOp | times |
| $E | $UnOp $E |
| $E | $E $BinOp $E |

https://github.com/mspandit/sippycup

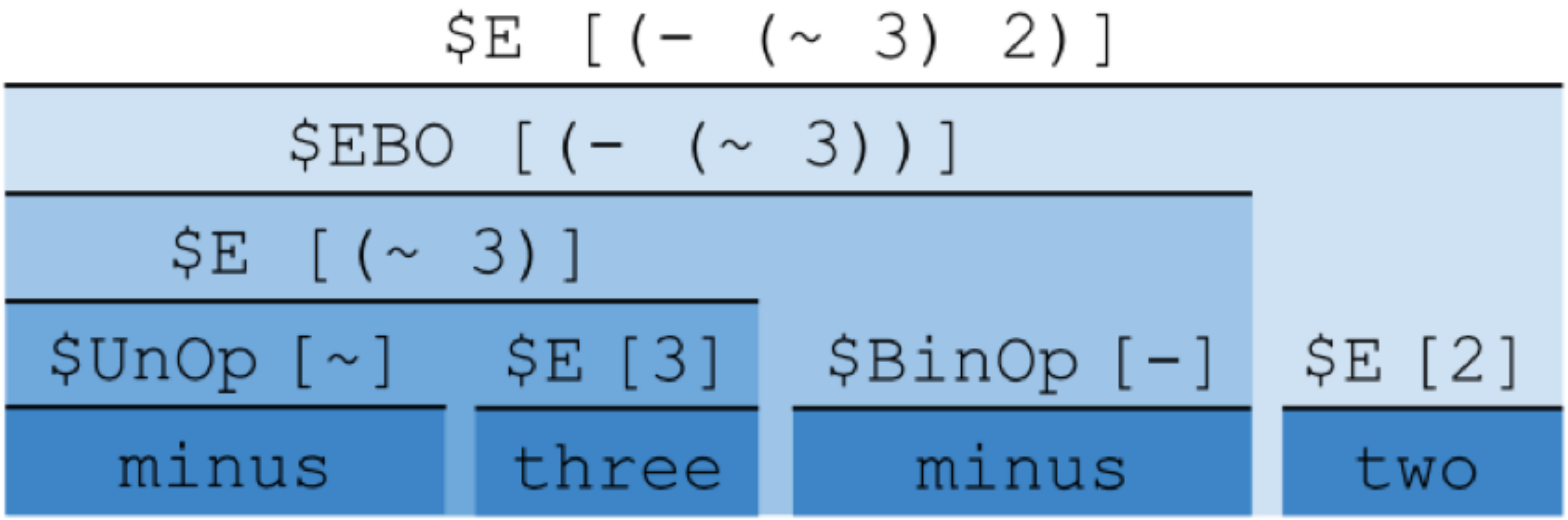
# CHOMSKY NORMAL FORM (BINARIZED) CFG RULES

| Left Hand Side | Right Hand Side |
|---|---|
| $E | one |
| $E | two |
| $E | three |
| $E | four |
| $UnOp | minus |
| $BinOp | minus |
| $BinOp | plus |
| $BinOp | times |
| $E | $UnOp $E |
| **$EBO** | **$E $BinOp** |
| **$E** | **$EBO $E** |

https://github.com/mspandit/sippycup

# unit1_tests.py

# SEMANTICS



# THE PRINCIPLE OF COMPOSITIONALITY

The meaning of a compound expression is a function of the meanings of its parts and the manner of their combination.

# unit1_tests.py

# STATUS

In every example, we produced some correct parse

In three examples, the parse at position 0 was incorrect

**Conclusion: Rank candidate parses so that correct parses are likely to appear higher in the list.**

# LINEAR SCORING FUNCTION

- Define multiple *feature functions* $\phi_i(p)$, each taking a parse $p$ as input and returning a real number as output.
- Store a *weight* $w_i$ for each feature function.
- For parse $p$:
- $score(p) = \sum_i w_i \cdot \phi_i(p)$

# unit1_tests.py

# LINEAR SCORING FUNCTION

- Define multiple *feature functions* $\phi_i(p)$, each taking a parse $p$ as input and returning a real number as output.
- Store a *weight* $w_i$ for each feature function.
- For parse $p$:
- $score(p) = \sum_i w_i \cdot \phi_i(p)$
- **What if there are many features? Learn weights from training data!**

# unit1_tests.py

# NATURAL LANGUAGE ARITHMETIC—SUMMARY

- Grammar with rules in Chomsky Normal (Binarized) Form
- Semantic representation derived from syntactic parses
- Feature functions for parses
- Machine learning of feature weights from semantics or denotation
- Performance improvement on ranking parses

# DEEP NEURAL NETWORKS

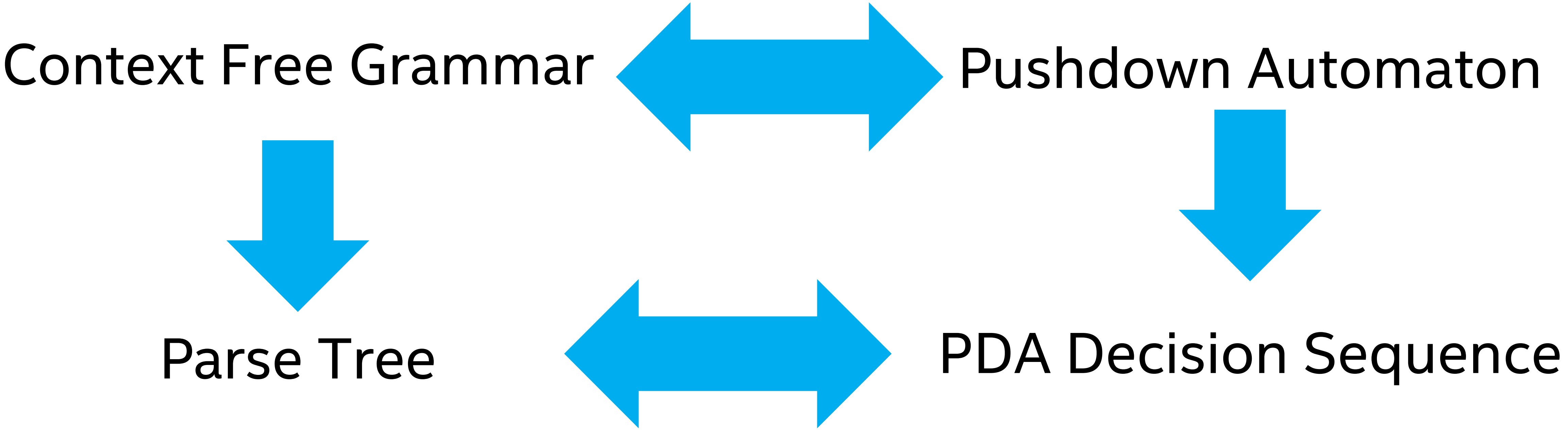
# POTENTIAL BENEFITS

- Recognize words absent from training vocabulary using word embeddings?
- Learn rules from examples?
- Probability distributions *across parses*?
- Fast, parallel generation of parses?

# DEEP LEARNING FOR SEMANTIC PARSING?

- Announcing SyntaxNet: The World's Most Accurate Parser Goes Open Source
  - *Globally Normalized Transition-Based Networks*
    - *Grammar as a Foreign Language*
      - *Sequence to sequence learning with neural networks*
- Translation with a Sequence to Sequence Network and Attention

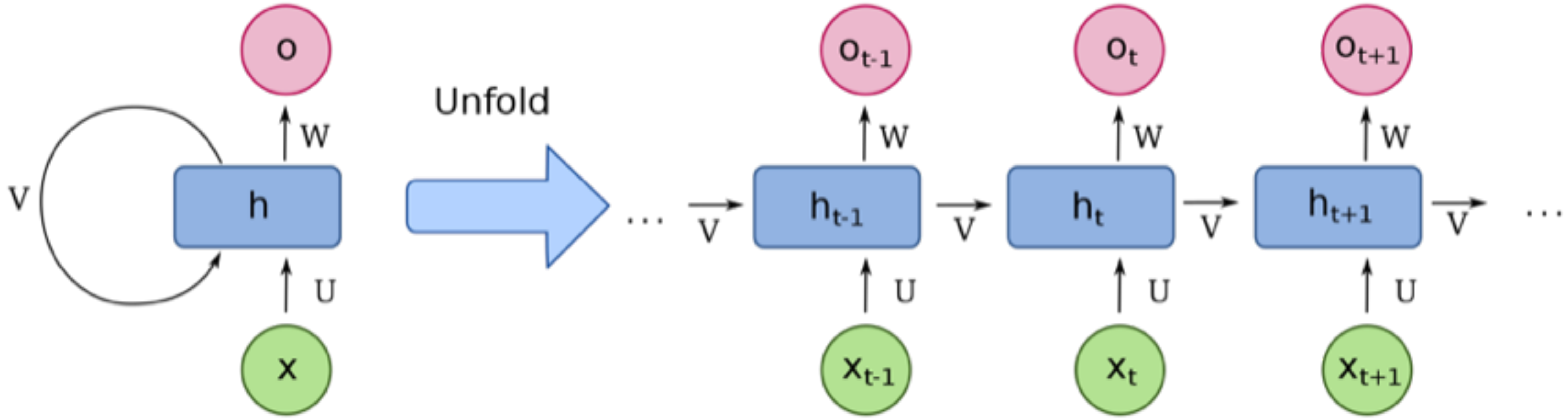
# INSIGHTS


Context Free Grammar
Pushdown Automaton
Parse Tree
PDA Decision Sequence

# EXAMPLES

| Expression | Constituent Structure | Categorized Constituent Structure | PDA Decision Sequence |
|---|---|---|---|
| minus three minus two | ((minus three) minus two) | ($E ($EBO ($E ($UnOp minus) ($E three)) ($BinOp minus)) ($E two)) | PUSH $UnOp PUSH $E $E PUSH $BinOp $EBO PUSH $E $E |
| minus three minus two | (minus (three minus two)) | ($E ($UnOp minus) ($E ($EBO ($E three) ($BinOp minus)) ($E two))) | PUSH $UnOp PUSH $E PUSH $BinOp $EBO PUSH $E $E $E |

# RECURRENT NEURAL NETWORK



# unit4_tests.py

# SUMMARY

Generating grammar produces random examples for training a sequence-to-sequence neural network

Given a sequence of inputs, an RNN can generate a sequence of probability distributions across PDA decisions

Sequence of *most likely* PDA decisions = a valid parse tree

Only one parse tree comes out of sequence of most likely PDA decisions

👉 Use *beam search* to consider other sequences of PDA decisions

https://github.com/mspandit/sippycup

# SO... WHAT'S NEXT?

1. Visit the Intel® AI Academy for additional resources, training materials and videos related to today's presentation. software.intel.com/AI
2. Try the Intel Distribution for Python! (https://software.intel.com/en-us/distribution-for-python)
3. Build useful chatbots and voice interfaces using semantic parsers!
4. Check out more examples of Intel AI/Movidius NCS/Intel AI DevCloud in action on Intel's Developer Network https://devmesh.intel.com/

# DISCLAIMER

intel AI