Supervised sentiment analysis: RNN classifiers

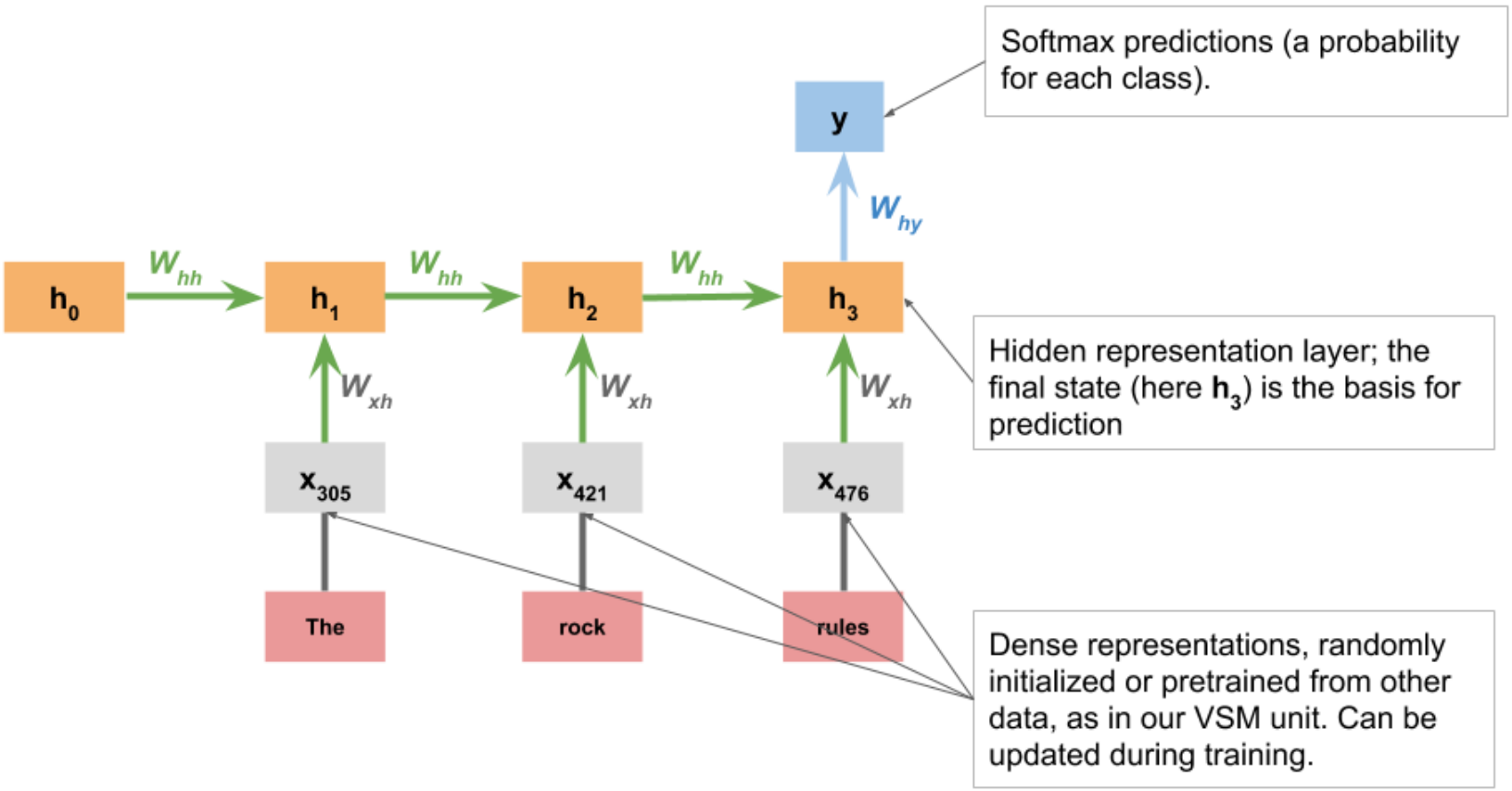
Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding

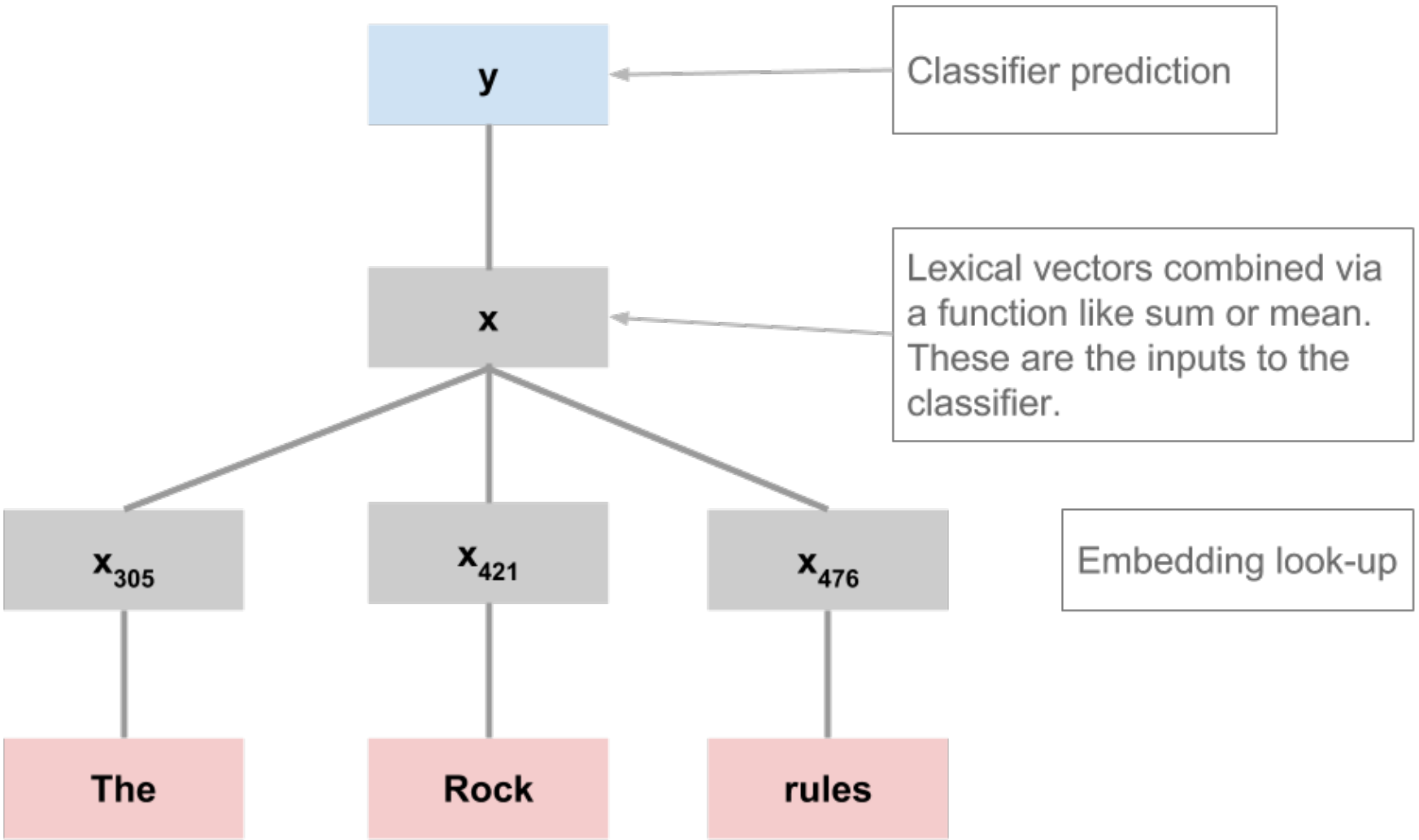
CC BY

Model overview



For complete details, see the reference implementation `np_rnn_classifier.py`

Distributed representations as features



Standard RNN dataset preparation

Standard RNN dataset preparation

Examples

[a, b, a]
[b, c]

Standard RNN dataset preparation

Examples

[a, b, a]
[b, c]

Indices

[1, 2, 1]
[2, 3]

Standard RNN dataset preparation

Examples

[a, b, a]
[b, c]

⇓

Indices

[1, 2, 1]
[2, 3]

	Embedding		
1	−0.42	0.10	0.12
2	−0.16	−0.21	0.29
3	−0.26	0.31	0.37

Standard RNN dataset preparation

Examples

[a, b, a]
[b, c]

⇓

Indices

[1, 2, 1]
[2, 3]

⇓

Vectors

$$\Big[[-0.42\ 0.10\ 0.12], [-0.16\ -0.21\ 0.29], [-0.42\ 0.10\ 0.12]\Big]$$
$$\Big[[-0.16\ -0.21\ 0.29], [-0.26\ 0.31\ 0.37]\Big]$$

Embedding

1	−0.42	0.10	0.12
2	−0.16	−0.21	0.29
3	−0.26	0.31	0.37

A note on LSTMs

1. Plain RNNs tend to perform poorly with very long sequences; as information flows back through the network, it is lost or distorted.

2. LSTM cells are a prominent response to this problem: they introduce mechanisms that control the flow of information.

3. We won't review all the mechanism for this here. I instead recommend these excellent blog posts, which include intuitive diagrams and discuss the motivations for the various pieces in detail:
   - Towards Data Science: Illustrated Guide to LSTM's and GRU's: A step by step explanation
   - colah's blog: Understanding LSTM networks

Code snippets

[1]:
```
import os
from torch_rnn_classifier import TorchRNNClassifier
import torch.nn as nn
import sst
import utils
```

[2]:
```
GLOVE_HOME = os.path.join('data', 'glove.6B')
SST_HOME = os.path.join('data', 'sentiment')
```

[3]:
```
GLOVE_LOOKUP = utils.glove2dict(os.path.join(GLOVE_HOME, 'glove.6B.50d.txt'))
```

[4]:
```
def rnn_phi(text):
    return text.lower().split()
```

[5]:
```
def fit_rnn(X, y):
    sst_train_vocab = utils.get_vocab(X, mincount=2)
    glove_embedding, sst_glove_vocab = utils.create_pretrained_embedding(
        GLOVE_LOOKUP, sst_train_vocab)
    mod = TorchRNNClassifier(
        sst_glove_vocab,
        eta=0.01,
        embedding=glove_embedding,
        batch_size=1028,
        hidden_dim=50,
        l2_strength=0.001,
        bidirectional=True,
        max_iter=50,
        early_stopping=True)
    mod.fit(X, y)
    return mod
```

[6]:
```
rnn_experiment = sst.experiment(
    sst.train_reader(SST_HOME),
    rnn_phi,
    fit_rnn,
    vectorize=False)
```