O P E N
D A T A
S C I E N C E
C O N F E R E N C E_

#ODSC

BOSTON 2015
@opendatasci

RECURRENT NEURAL NETWORKS FOR TEXT ANALYSIS

Alec Radford

Recurrent Neural Networks for text analysis

From idea to practice

BOSTON 2015
@opendatasci
#ODSC

ALEC RADFORD

Follow Along

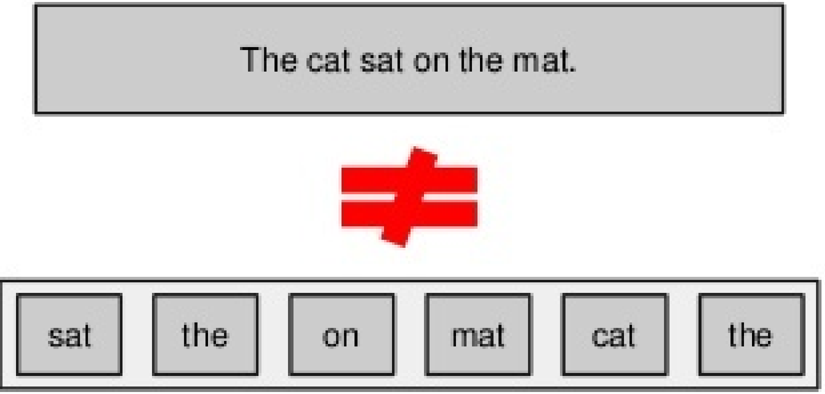
Slides at: http://goo.gl/WLsUWv

How ML

-0.15, 0.2, 0, 1.5	Numerical, great!
A, B, C, D	Categorical, great!
The cat sat on the mat.	Uhhh…….

How text is dealt with (ML perspective)

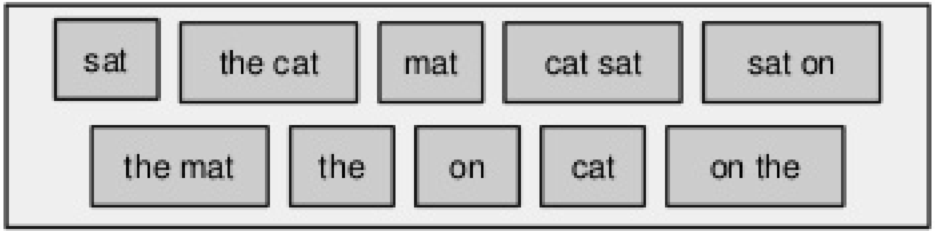
Structure is important!



- Certain tasks, structure is essential:
  - Humor
  - Sarcasm
- Certain tasks, ngrams can get you a long way:
  - Sentiment Analysis
  - Topic detection
- Specific words can be strong indicators
  - useless, fantastic (sentiment)
  - hoop, green tea, NASDAQ (topic)

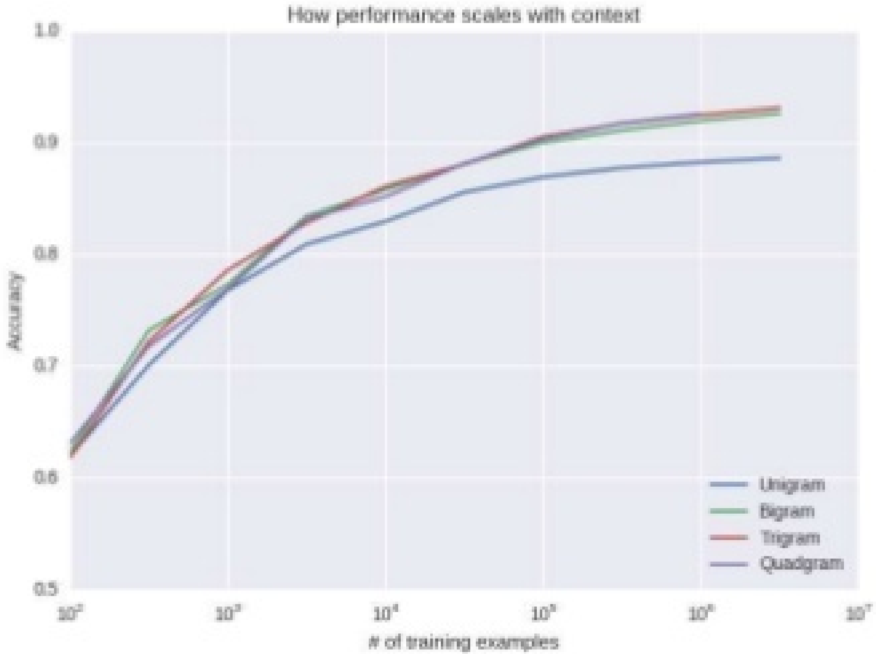
Structure is hard

Ngrams is typical way of preserving some structure.

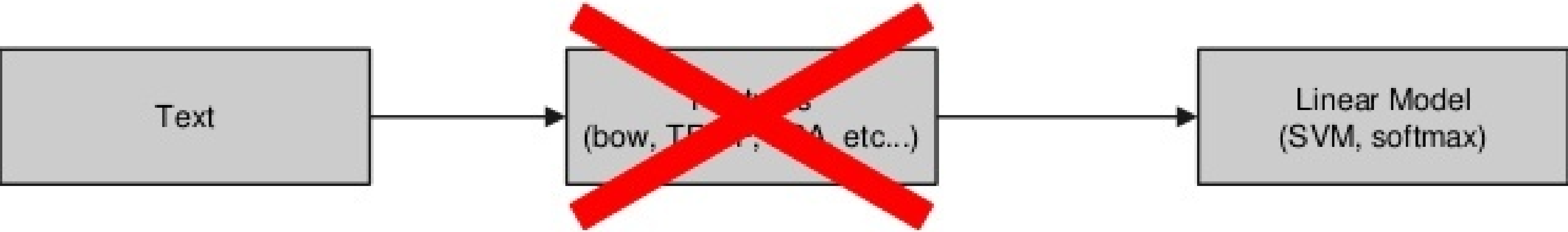


Beyond bi or tri-grams occurrences become very rare and dimensionality becomes huge (1, 10 million + features)

Structure is hard

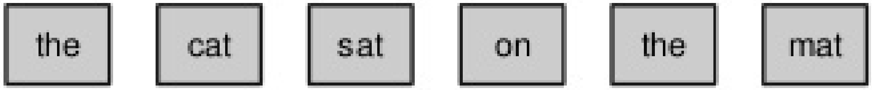


How text is dealt with (ML perspective)

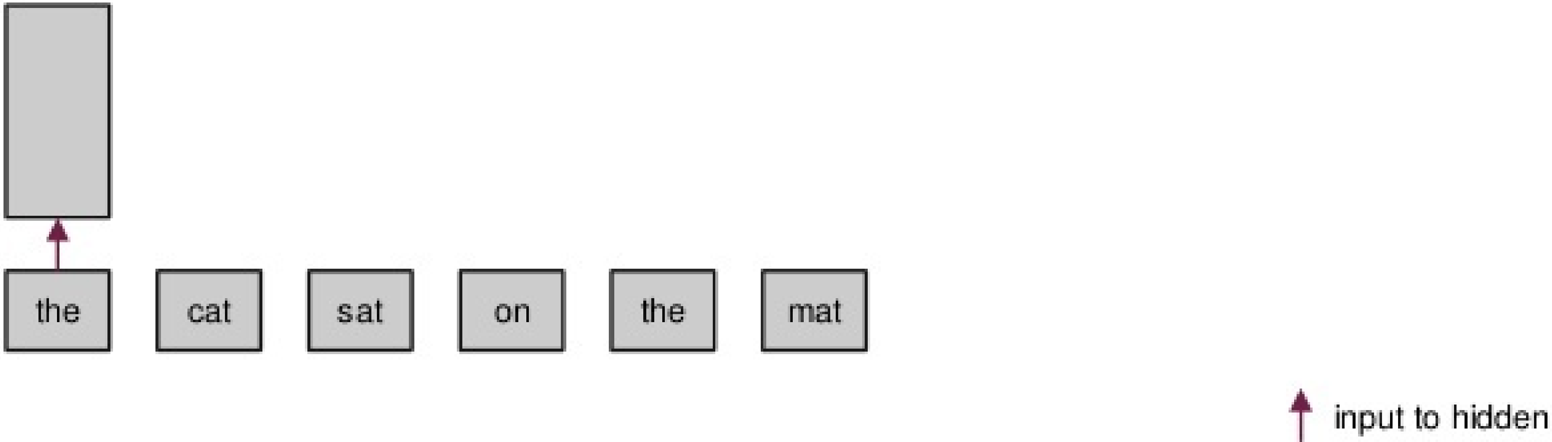


How text should be dealt with?

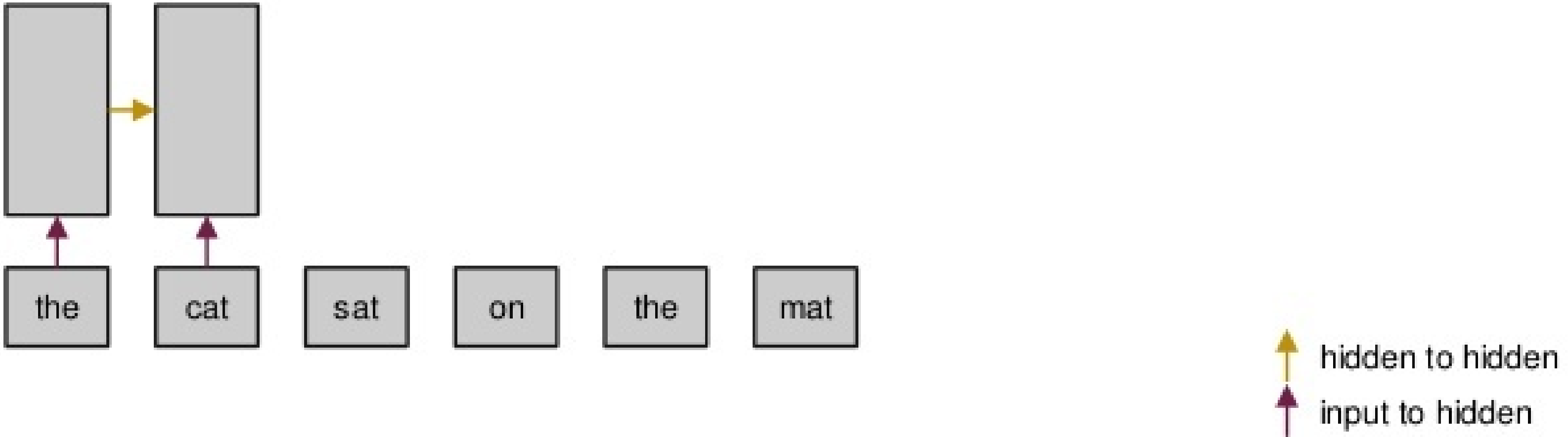
How an RNN works



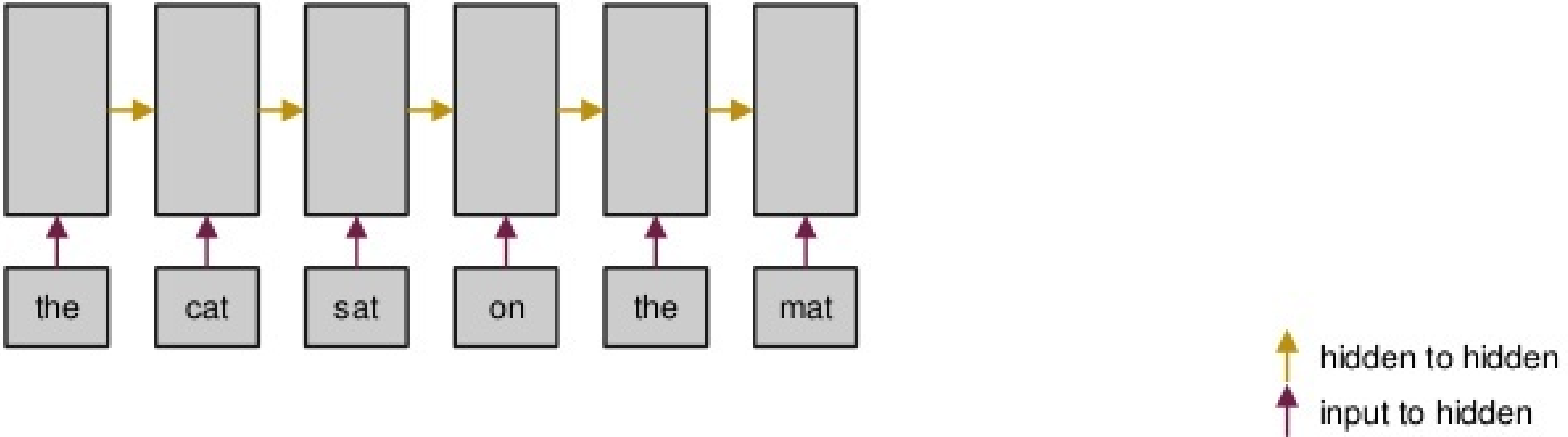
How an RNN works



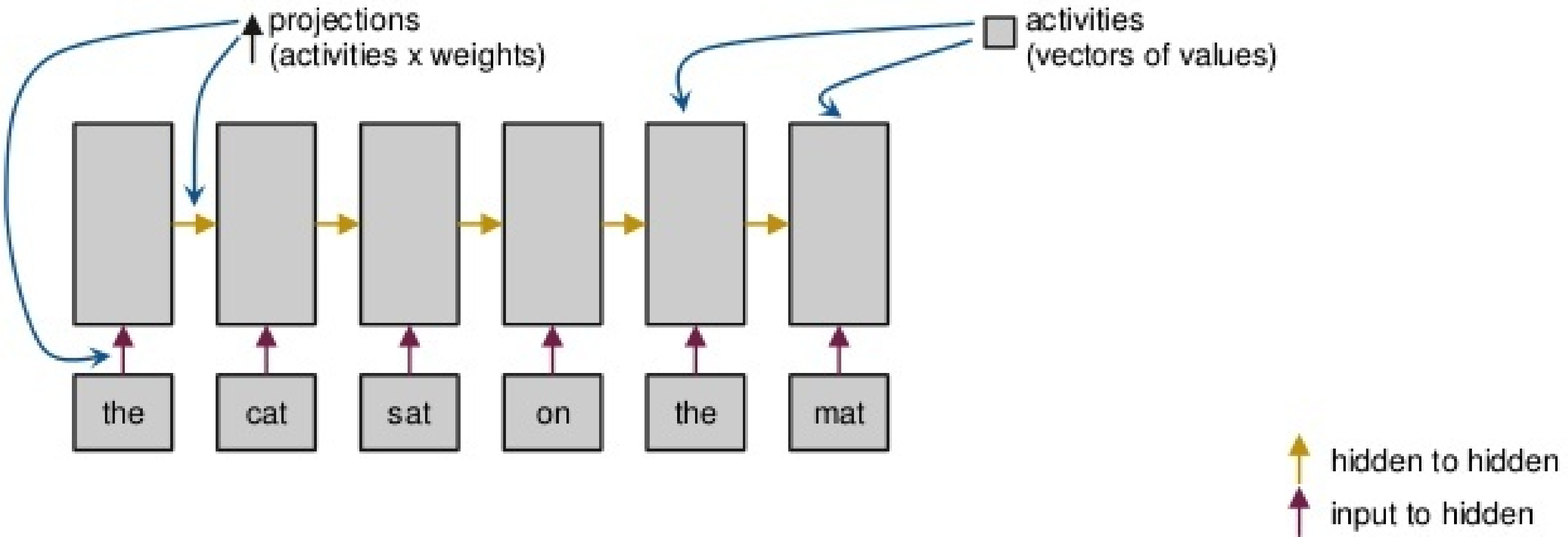
How an RNN works



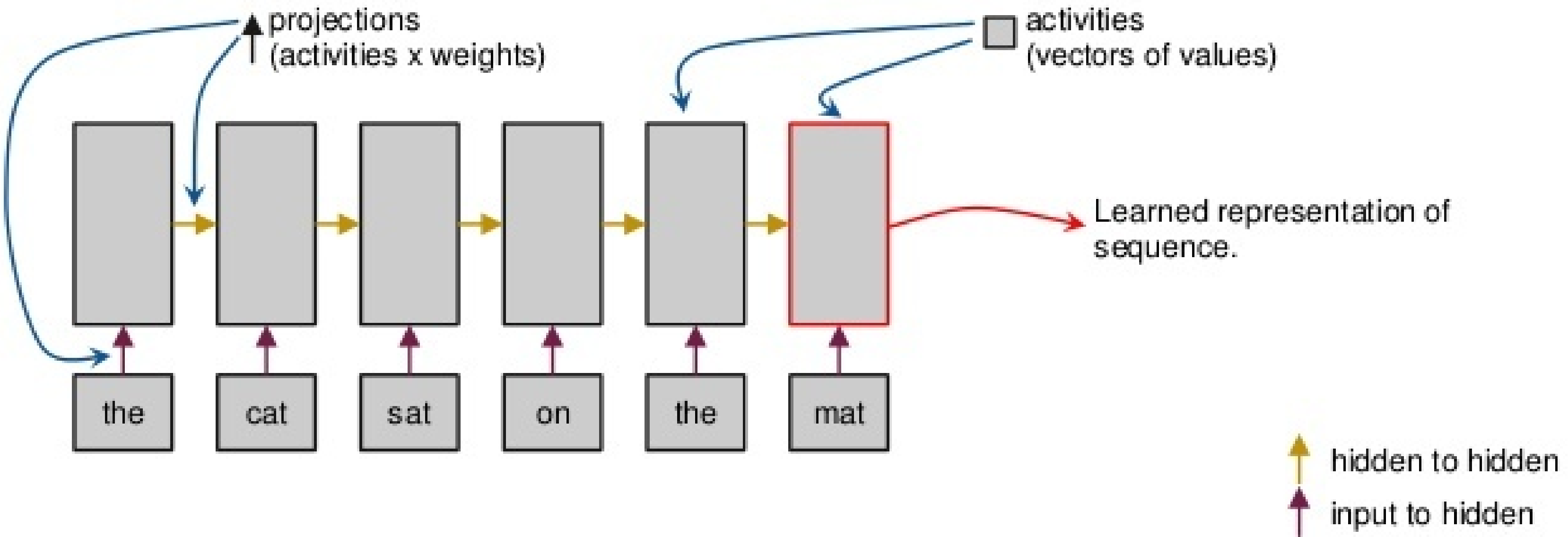
How an RNN works



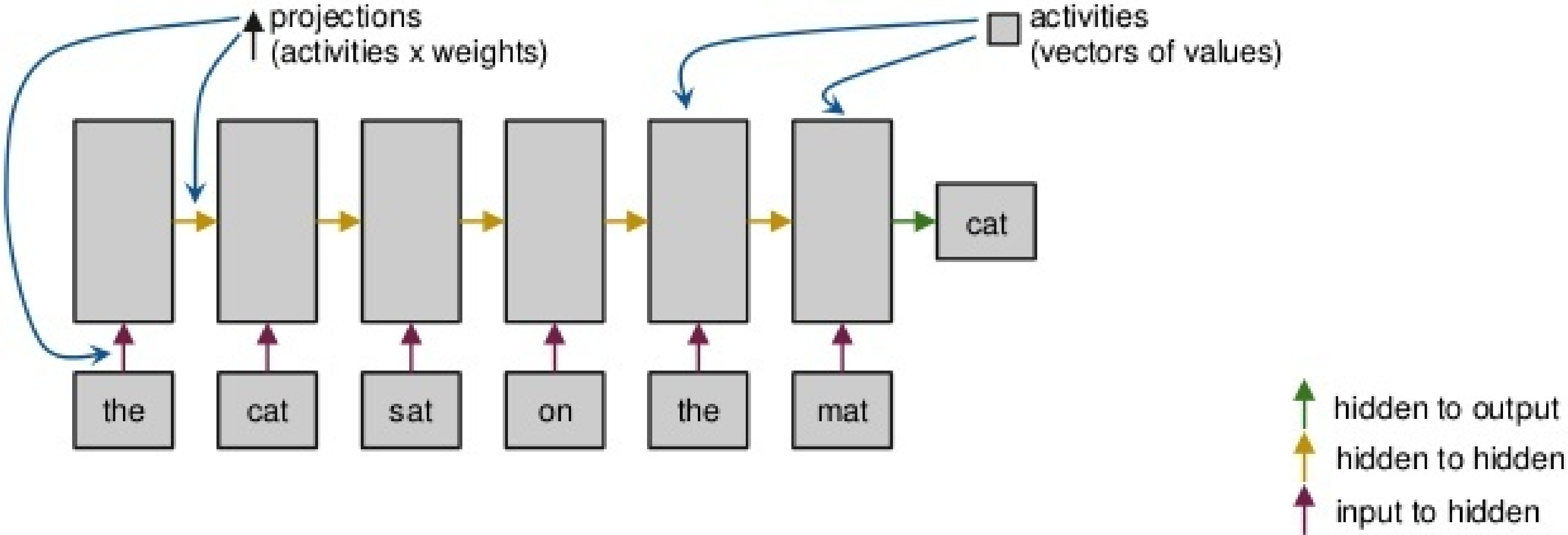
How an RNN works



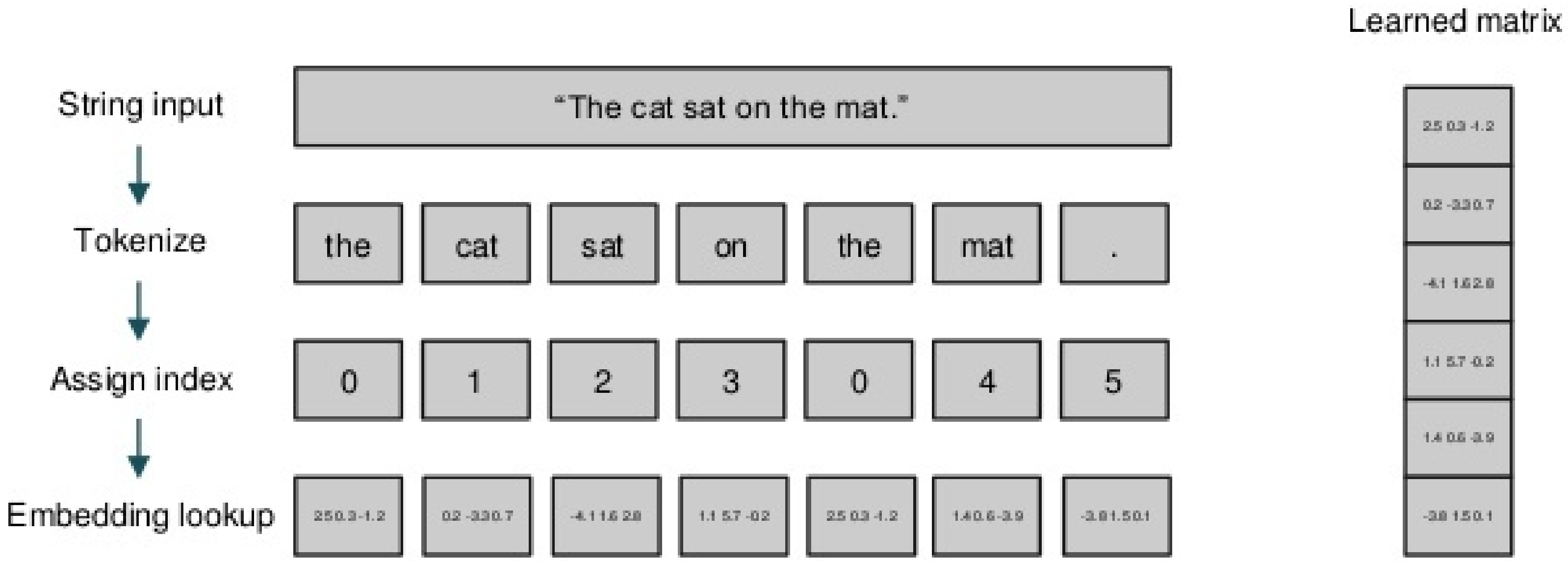
How an RNN works



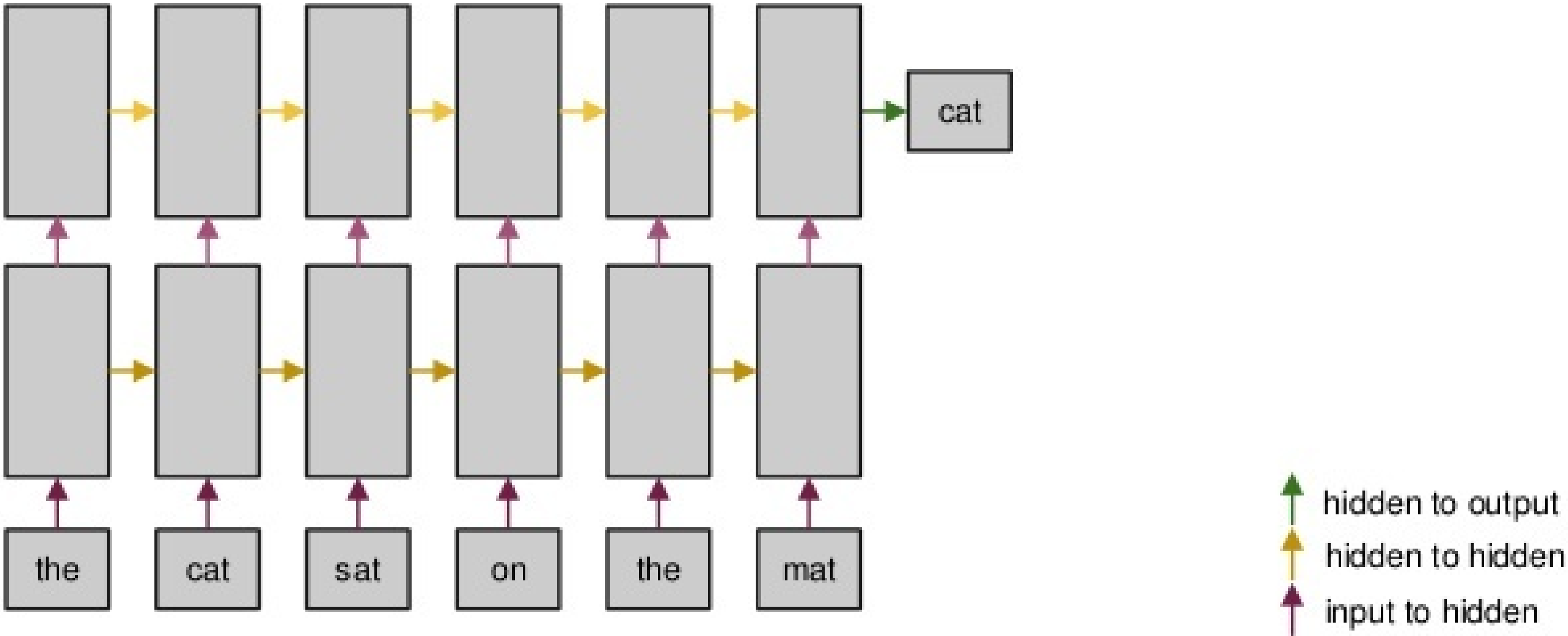
How an RNN works



From text to RNN input



You can stack them too



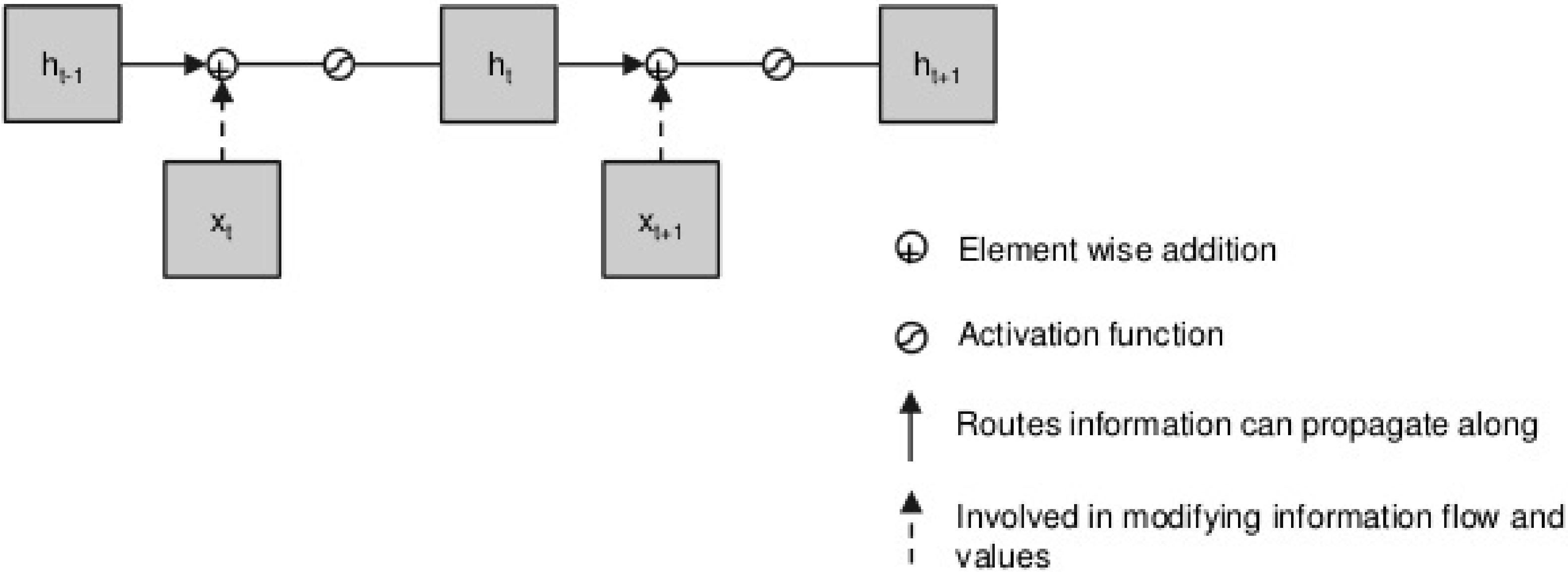
But aren’t RNNs unstable?

Simple RNNs trained with SGD are unstable/difficult to learn.

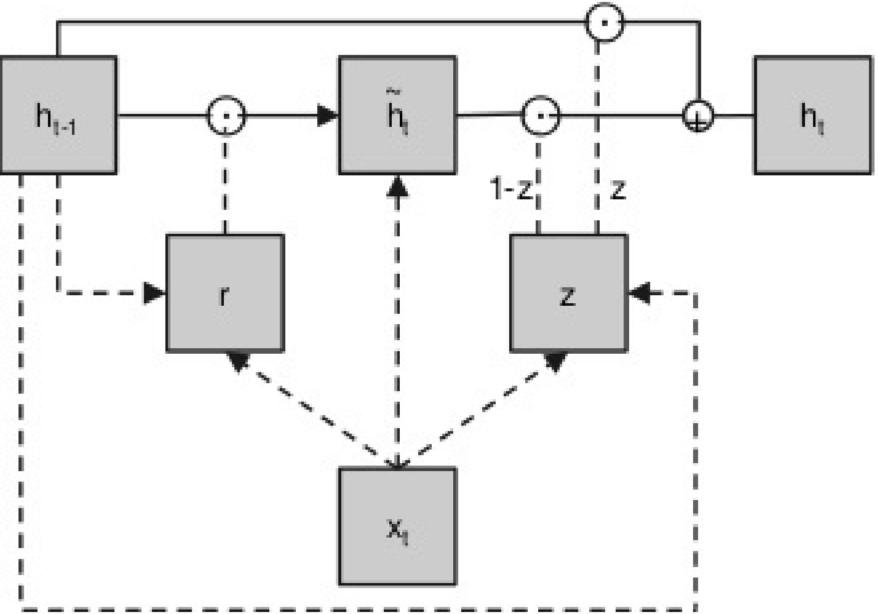
But modern RNNs with various tricks blow up much less often!

- Gating Units
- Gradient Clipping
- Steeper gates
- Better initialization
- Better optimizers
- Bigger datasets

Simple Recurrent Unit



Gated Recurrent Unit - GRU



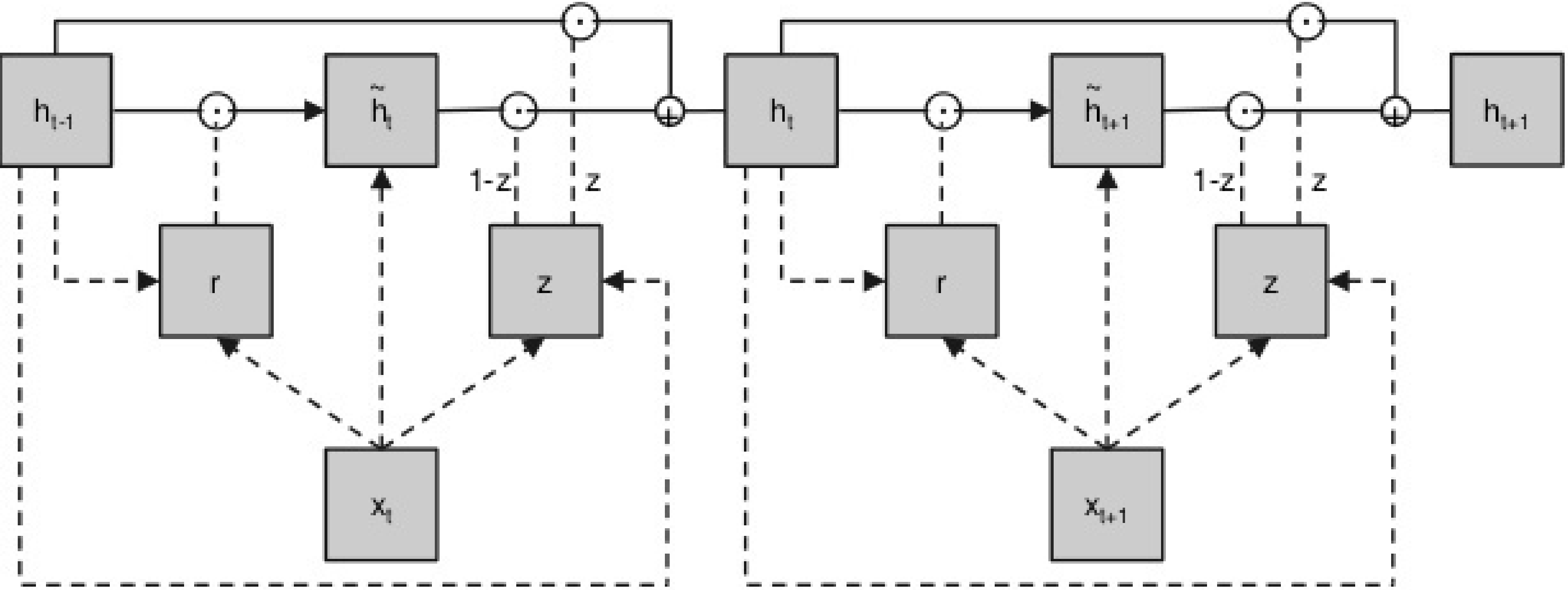
⊕ Element wise addition

⊙ Element wise multiplication

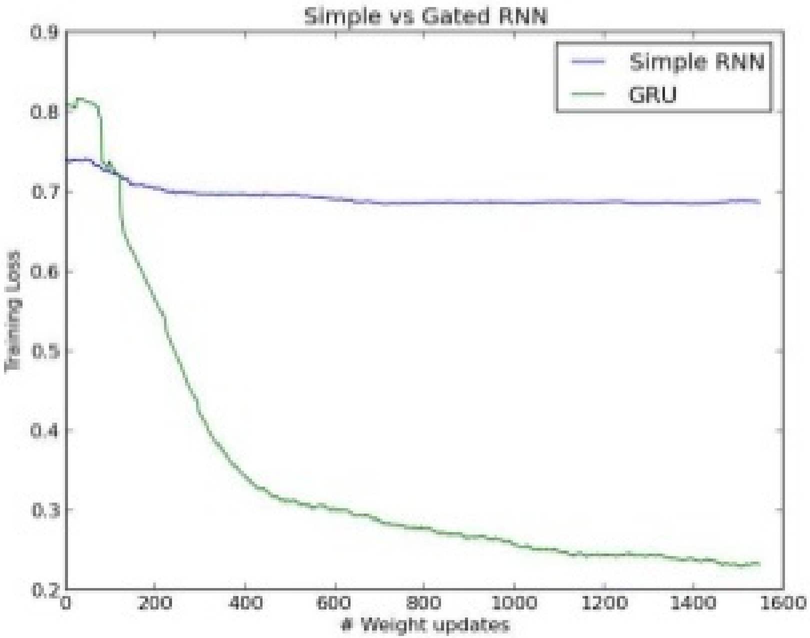
Routes information can propagate along

Involved in modifying information flow and values

Gated Recurrent Unit - GRU



Gating is important



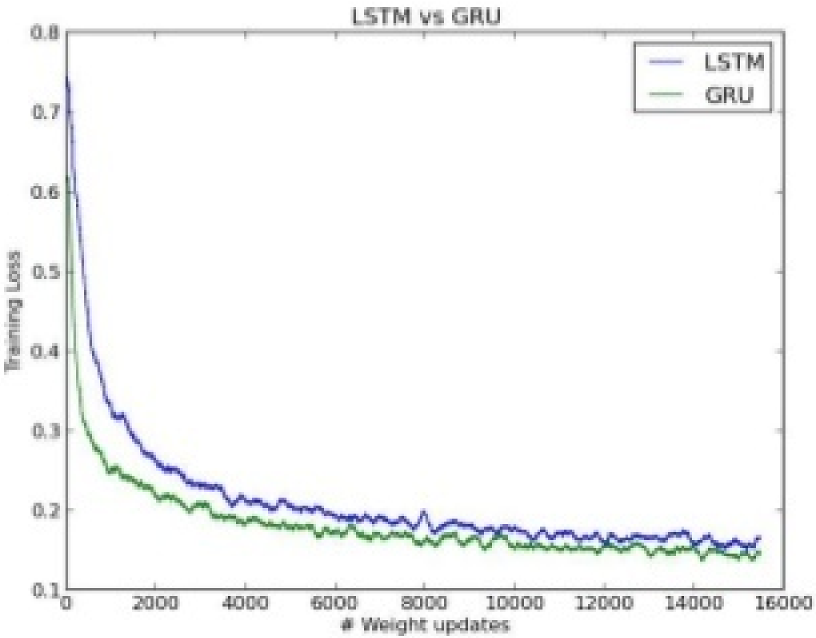
For sentiment analysis of longer sequences of text (paragraph or so) a simple RNN has difficulty learning at all while a gated RNN does so easily.

Which One?

There are two types of gated RNNs:

- Gated Recurrent Units (GRU) by K. Cho, recently introduced and used for machine translation and speech recognition tasks.

- Long short term memory (LSTM) by S. Hochreiter and J. Schmidhuber has been around since 1997 and has been used far more. Various modifications to it exist.

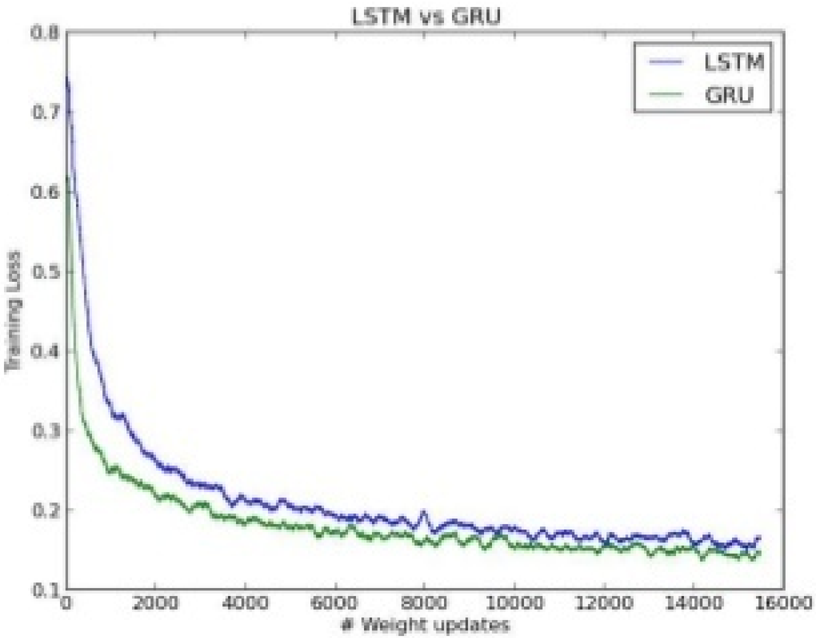


Which One?

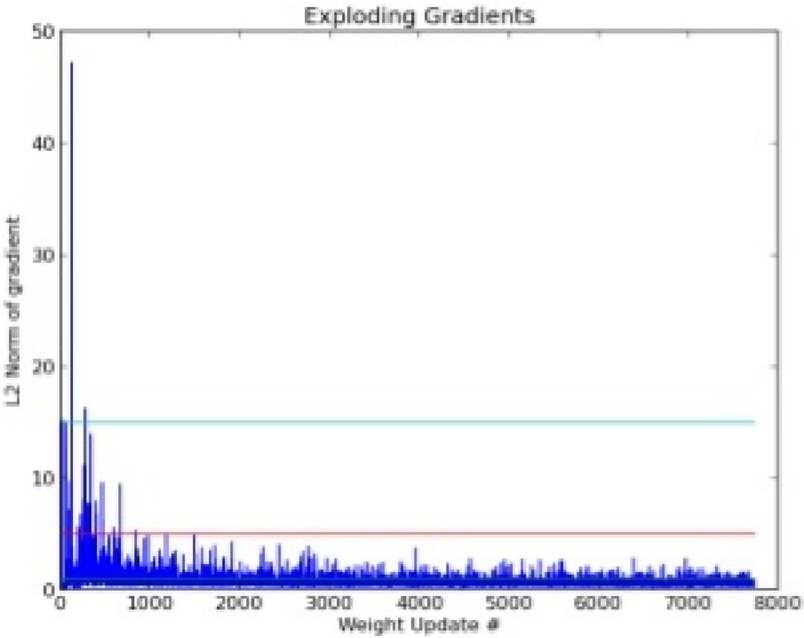
GRU is simpler, faster, and optimizes quicker (at least on sentiment).

Because it only has two gates (compared to four) approximately 1.5-1.75x faster for theano implementation.

If you have a huge dataset and don't mind waiting LSTM may be better in the long run due to its greater complexity - especially if you add peephole connections.



Exploding Gradients?



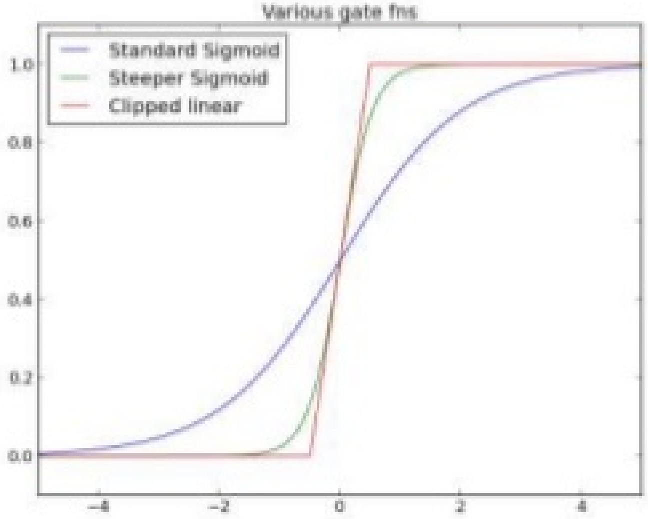
Exploding gradients are a major problem for traditional RNNs trained with SGD. One of the sources of the reputation of RNNs being hard to train.

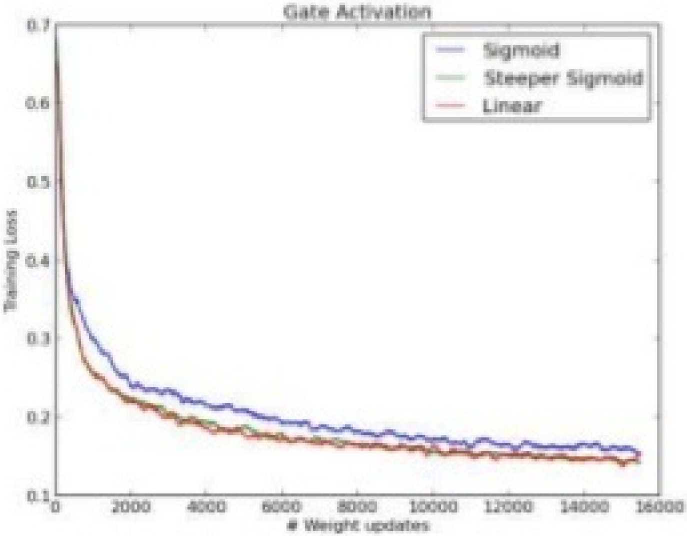
In 2012, R Pascanu and T. Mikolov proposed clipping the norm of the gradient to alleviate this.

Modern optimizers don't seem to have this problem - at least for classification text analysis.

Better Gating Functions

Interesting paper at NIPS workshop (Q. Lyu, J. Zhu) - make the gates “steeper” so they change more rapidly from “off” to “on” so model learns to use them quicker.



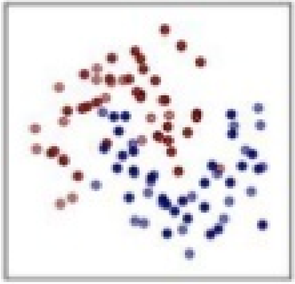


Better Initialization

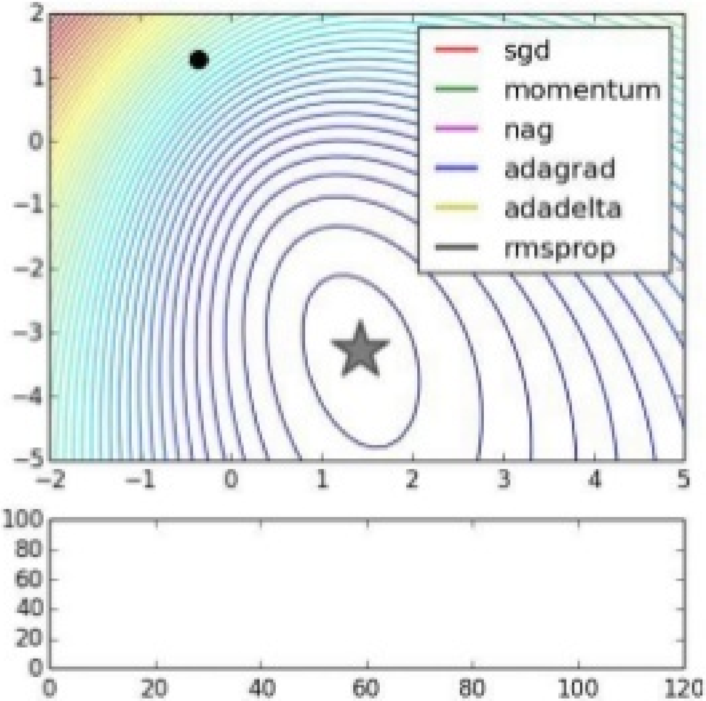
Andrew Saxe last year showed that initializing weight matrices with random orthogonal matrices works better than random gaussian (or uniform) matrices.

In addition, Richard Socher (and more recently Quoc Le) have used identity initialization schemes which work great as well.

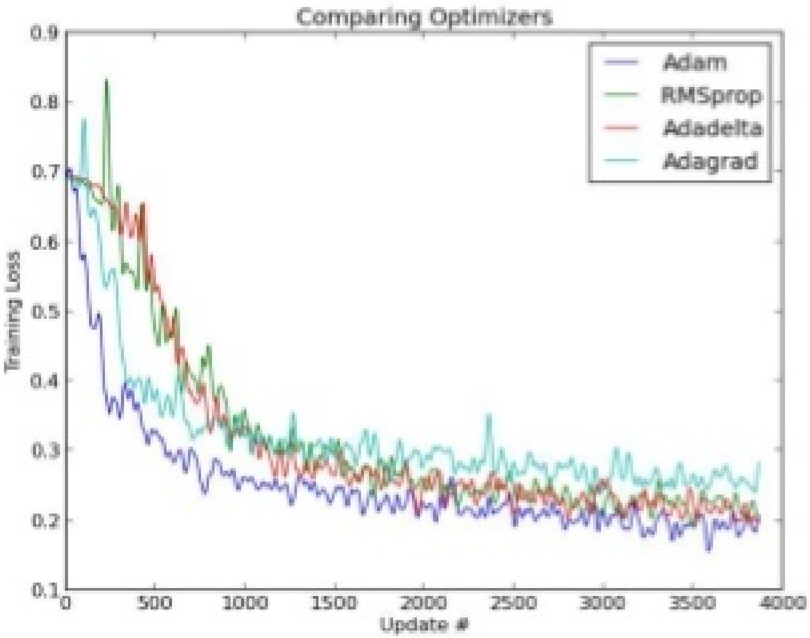
Understanding Optimizers



2D moons dataset
courtesy of scikit-learn



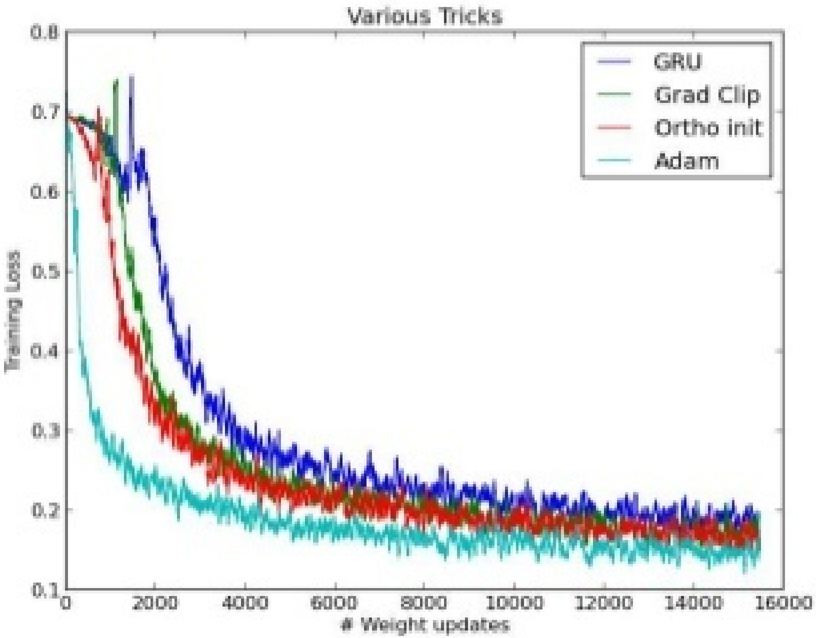
Comparing Optimizers



Adam (D. Kingma) combines the early optimization speed of Adagrad (J. Duchi) with the better later convergence of various other methods like Adadelta (M. Zeiler) and RMSprop (T. Tieleman).

Warning: Generalization performance of Adam seems slightly worse for smaller datasets.

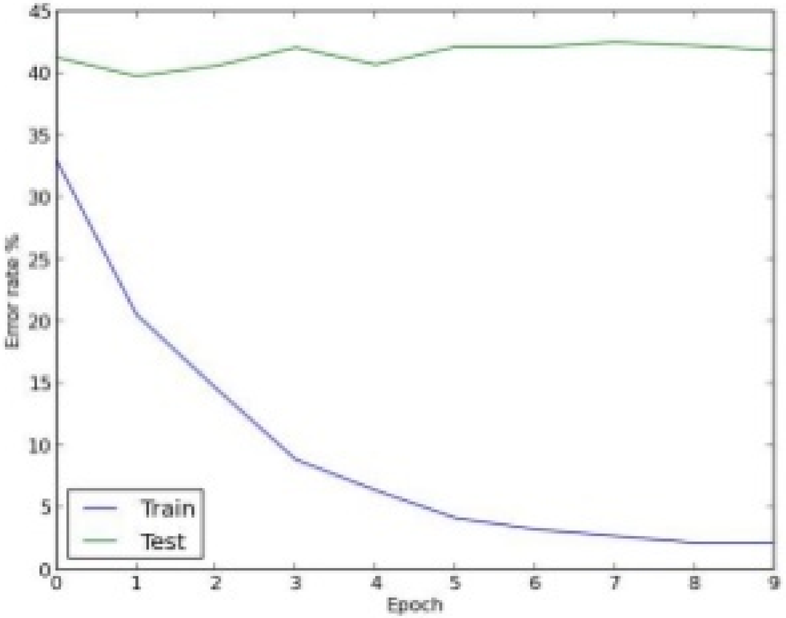
It adds up



Up to 10x more efficient training once you add all the tricks together compared to a naive implementation - much more stable - rarely diverges.

Around 7.5x faster, the various tricks add a bit of computation time.

Too much? - Overfitting



RNNs can overfit very well as we will see. As they continue to fit to training dataset, their performance on test data will plateau or even worsen.

Keep track of it using a validation set, save model at each iteration over training data and pick the earliest, best, validation performance.

The Showdown

Model #1

sklearn.feature_extraction.text.TfidfVectorizer

```
class sklearn.feature_extraction.text.TfidfVectorizer(input=u'content', encoding=u'utf-8', charset=None, decode_error=u'strict', charset_error=None, strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, analyzer=u'word', stop_words=None, token_pattern=u'(?u)\b\w\w+\b', ngram_range=(1, 1), max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<type 'numpy.int64'>, norm=u'l2', use_idf=True, smooth_idf=True, sublinear_tf=False)
```

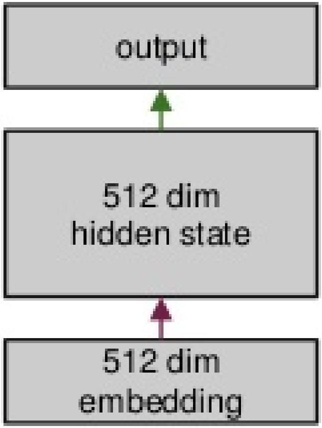
+

sklearn.linear_model.LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None)
```

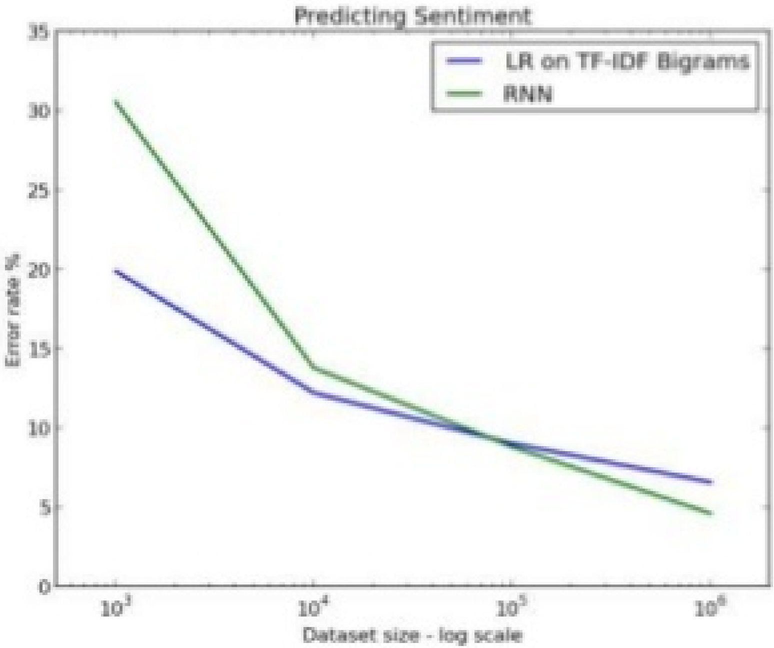
Using bigrams and grid search on min_df for vectorizer and regularization coefficient for model.

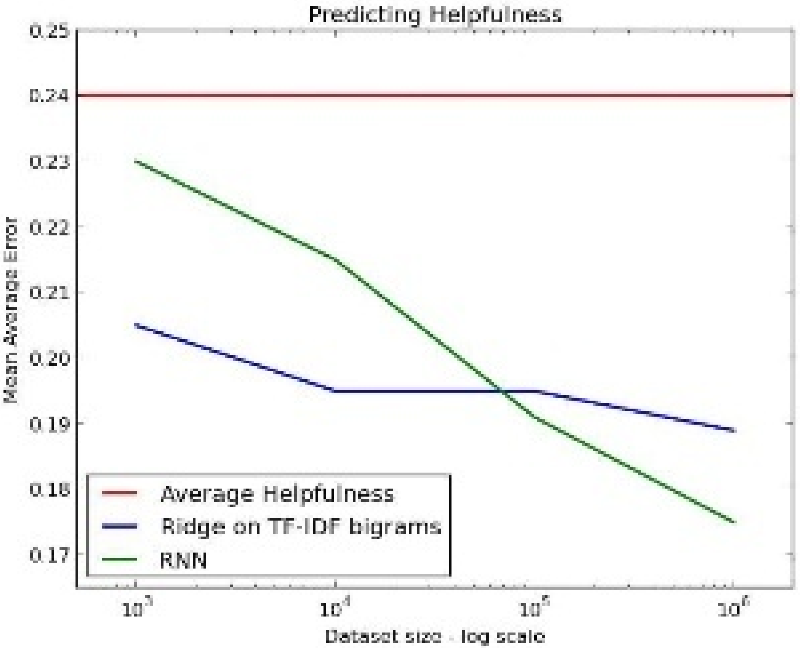
Model #2



Using whatever I tried that worked :)
Adam, GRU, steeper sigmoid gates, ortho/identity init are good defaults

Sentiment & Helpfulness





Effect of Dataset Size

- RNNs have poor generalization properties on small datasets.
  - 1K labeled examples 25-50% worse than linear model…
- RNNs have better generalization properties on large datasets.
  - 1M labeled examples 0-30% better than linear model.
- Crossovers between 10K and 1M examples
  - Depends on dataset.

The Thing we don't talk about

For 1 million paragraph sized text examples to converge:

- Linear model takes 30 minutes on a single CPU core.
- RNN takes 90 minutes on a Titan X.
- RNN takes five days on a single CPU core.

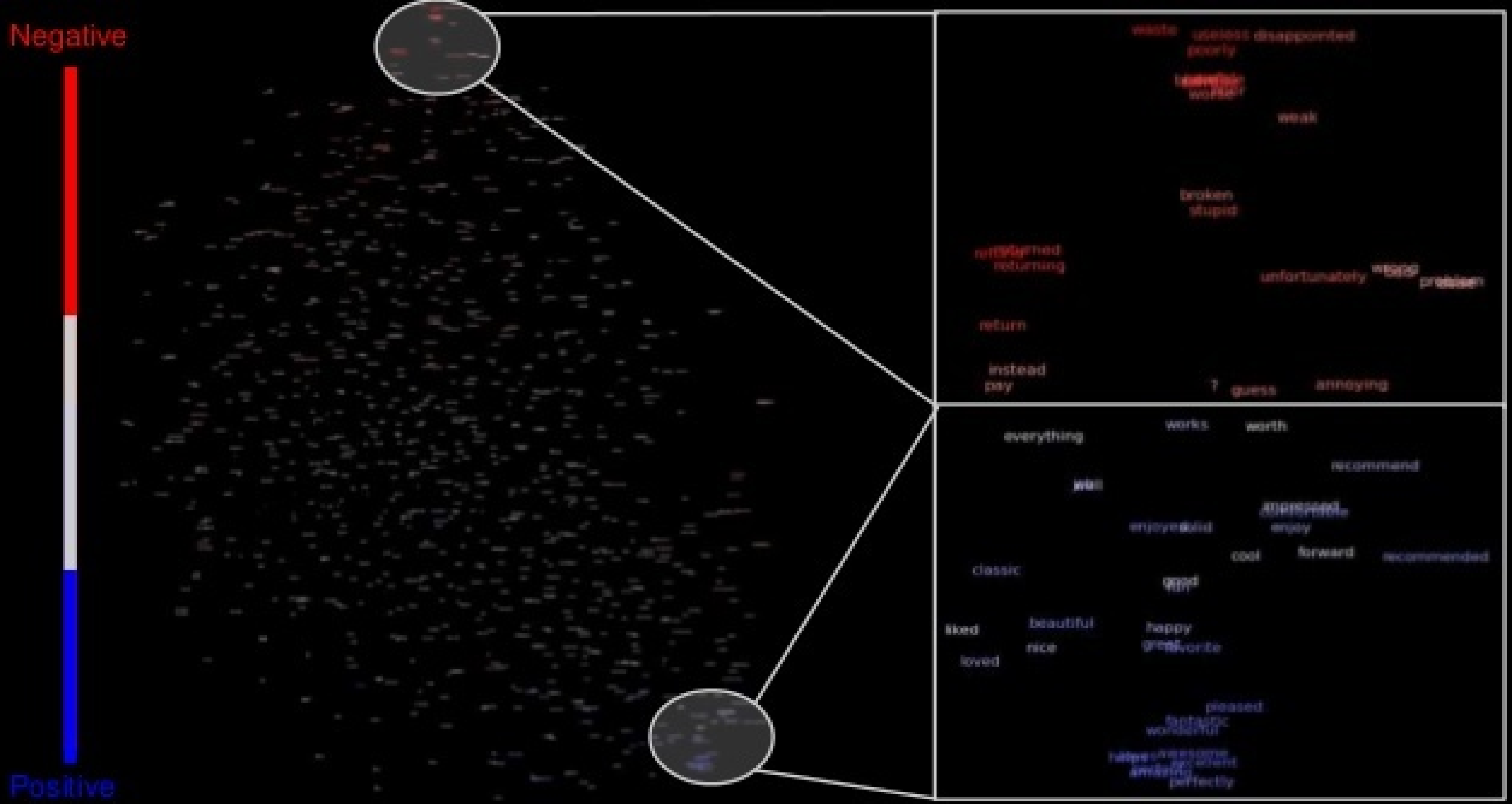
RNN is about 250x slower on CPU than linear model…

This is why we use GPUs

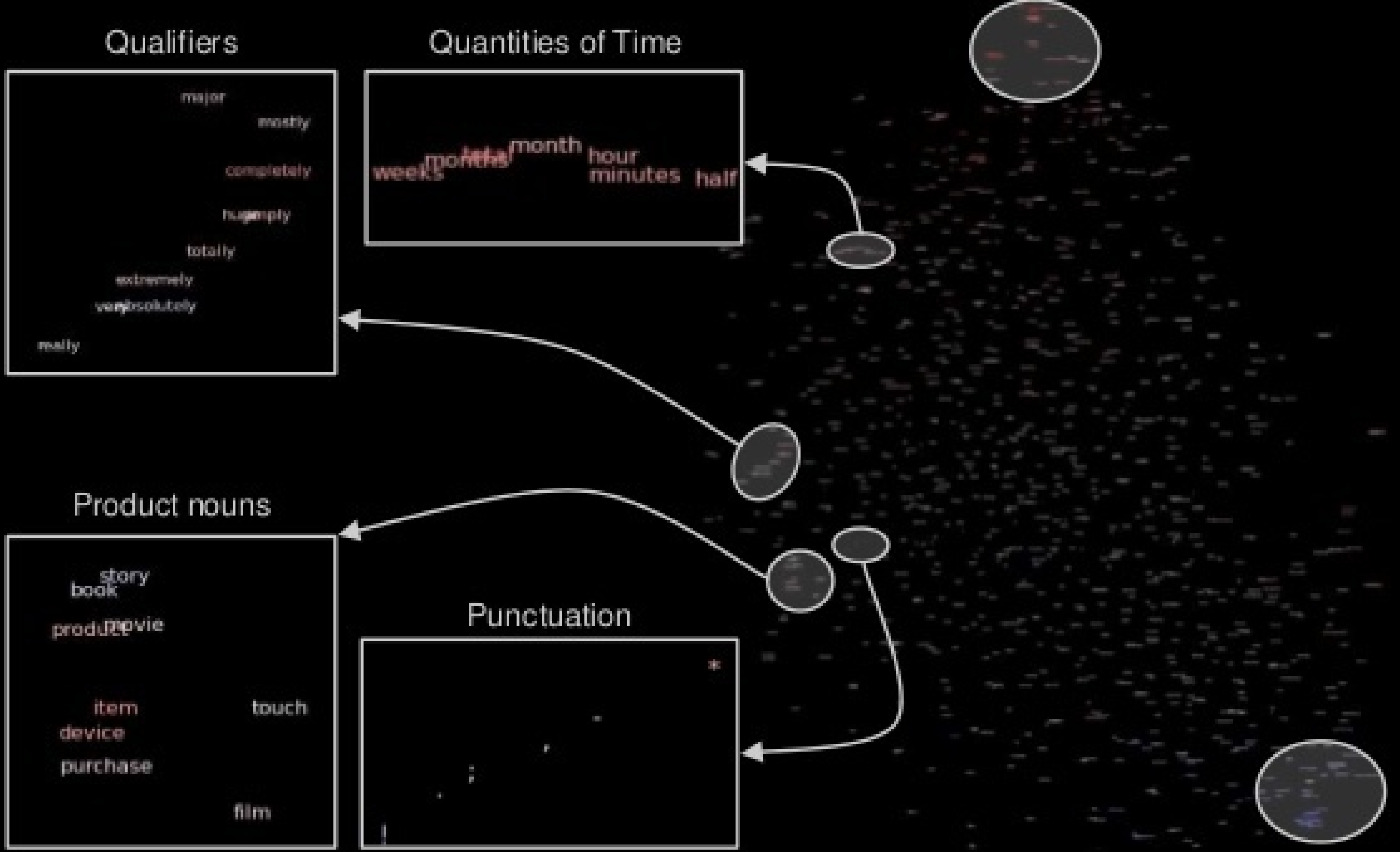
Visualizing representations of words learned via sentiment

Individual words colored by average sentiment

TSNE - L.J.P. van der



Model learns to separate negative and positive words, not too surprising

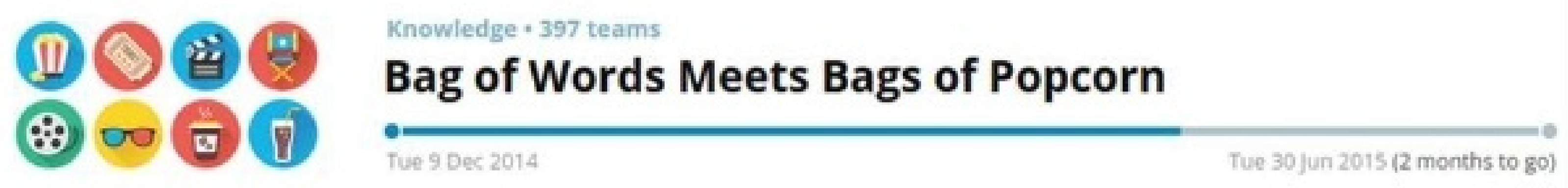


Much cooler, model also begins to learn components of language ***from only binary sentiment labels***

The library - Passage

- Tiny RNN library built on top of Theano
- https://github.com/IndicoDataSolutions/Passage
- Still alpha - we're working on it!
- Supports simple, LSTM, and GRU recurrent layers
- Supports multiple recurrent layers
- Supports deep input to and deep output from hidden layers
    - no deep transitions currently
- Supports embedding and onehot input representations
- Can be used for both regression and classification problems
    - Regression needs preprocessing for stability - working on it
- Much more in the pipeline

An example



Sentiment analysis of movie reviews - 25K labeled examples

```python
import numpy as np
import pandas as pd
from lxml import html

from passage.models import RNN
from passage.updates import Adadelta
from passage.layers import Embedding, GatedRecurrent, Dense
from passage.preprocessing import Tokenizer

# download data at kaggle.com/c/word2vec-nlp-tutorial/data

def clean(texts):
    return [html.fromstring(text).text_content().lower().strip() for text in texts]

if __name__ == "__main__":
    tr_data = pd.read_csv('labeledTrainData.tsv', delimiter='\t')
    trX = clean(tr_data['review'].values)
    trY = tr_data['sentiment'].values

    tokenizer = Tokenizer(min_df=10, max_features=100000)
    trX = tokenizer.fit_transform(trX)

    layers = [
        Embedding(size=256, n_features=tokenizer.n_features),
        GatedRecurrent(size=512, seq_output=False, p_drop=0.75),
        Dense(size=1, activation='sigmoid')
    ]

    model = RNN(layers=layers, cost='bce', updater=Adadelta(lr=0.5))
    model.fit(trX, trY, n_epochs=10)

    te_data = pd.read_csv('testData.tsv', delimiter='\t')
    ids = te_data['id'].values
    teX = clean(te_data['review'].values)
    teX = tokenizer.transform(teX)
    pr_teX = model.predict(teX).flatten()

    pd.DataFrame(np.asarray([ids, pr_teX]).T).to_csv('submission.csv', index=False, header=["id", "sentiment"])
```

```python
import numpy as np
import pandas as pd
from lxml import html

from passage.models import RNN
from passage.updates import Adadelta
from passage.layers import Embedding, GatedRecurrent, Dense
from passage.preprocessing import Tokenizer

# download data at kaggle.com/c/word2vec-nlp-tutorial/data

def clean(texts):
    return [html.fromstring(text).text_content().lower().strip() for text in texts]

if __name__ == "__main__":
    tr_data = pd.read_csv('labeledTrainData.tsv', delimiter='\t')
    trX = clean(tr_data['review'].values)
    trY = tr_data['sentiment'].values

    tokenizer = Tokenizer(min_df=10, max_features=100000)
    trX = tokenizer.fit_transform(trX)

    layers = [
        Embedding(size=256, n_features=tokenizer.n_features),
        GatedRecurrent(size=512, seq_output=False, p_drop=0.75),
        Dense(size=1, activation='sigmoid')
    ]

    model = RNN(layers=layers, cost='bce', updater=Adadelta(lr=0.5))
    model.fit(trX, trY, n_epochs=10)

    te_data = pd.read_csv('testData.tsv', delimiter='\t')
    ids = te_data['id'].values
    teX = clean(te_data['review'].values)
    teX = tokenizer.transform(teX)
    pr_teX = model.predict(teX).flatten()

    pd.DataFrame(np.asarray([ids, pr_teX]).T).to_csv('submission.csv', index=False, header=["id", "sentiment"])
```

RNN imports

```python
import numpy as np
import pandas as pd
from lxml import html

from passage.models import RNN
from passage.updates import Adadelta
from passage.layers import Embedding, GatedRecurrent, Dense
from passage.preprocessing import Tokenizer

# download data at kaggle.com/c/word2vec-nlp-tutorial/data

def clean(texts):
    return [html.fromstring(text).text_content().lower().strip() for text in texts]

if __name__ == "__main__":
    tr_data = pd.read_csv('labeledTrainData.tsv', delimiter='\t')
    trX = clean(tr_data['review'].values)
    trY = tr_data['sentiment'].values

    tokenizer = Tokenizer(min_df=10, max_features=100000)
    trX = tokenizer.fit_transform(trX)

    layers = [
        Embedding(size=256, n_features=tokenizer.n_features),
        GatedRecurrent(size=512, seq_output=False, p_drop=0.75),
        Dense(size=1, activation='sigmoid')
    ]

    model = RNN(layers=layers, cost='bce', updater=Adadelta(lr=0.5))
    model.fit(trX, trY, n_epochs=10)

    te_data = pd.read_csv('testData.tsv', delimiter='\t')
    ids = te_data['id'].values
    teX = clean(te_data['review'].values)
    teX = tokenizer.transform(teX)
    pr_teX = model.predict(teX).flatten()

    pd.DataFrame(np.asarray([ids, pr_teX]).T).to_csv('submission.csv', index=False, header=["id", "sentiment"])
```

RNN imports

preprocessing

```python
import numpy as np
import pandas as pd
from lxml import html
```

RNN imports

```python
from passage.models import RNN
from passage.updates import Adadelta
from passage.layers import Embedding, GatedRecurrent, Dense
from passage.preprocessing import Tokenizer

# download data at kaggle.com/c/word2vec-nlp-tutorial/data
```

preprocessing

```python
def clean(texts):
    return [html.fromstring(text).text_content().lower().strip() for text in texts]

if __name__ == "__main__":
```

load training data

```python
    tr_data = pd.read_csv('labeledTrainData.tsv', delimiter='\t')
    trX = clean(tr_data['review'].values)
    trY = tr_data['sentiment'].values

    tokenizer = Tokenizer(min_df=10, max_features=100000)
    trX = tokenizer.fit_transform(trX)

    layers = [
        Embedding(size=256, n_features=tokenizer.n_features),
        GatedRecurrent(size=512, seq_output=False, p_drop=0.75),
        Dense(size=1, activation='sigmoid')
    ]

    model = RNN(layers=layers, cost='bce', updater=Adadelta(lr=0.5))
    model.fit(trX, trY, n_epochs=10)

    te_data = pd.read_csv('testData.tsv', delimiter='\t')
    ids = te_data['id'].values
    teX = clean(te_data['review'].values)
    teX = tokenizer.transform(teX)
    pr_teX = model.predict(teX).flatten()

    pd.DataFrame(np.asarray([ids, pr_teX]).T).to_csv('submission.csv', index=False, header=["id", "sentiment"])
```

```python
import numpy as np
import pandas as pd
from lxml import html

from passage.models import RNN
from passage.updates import Adadelta
from passage.layers import Embedding, GatedRecurrent, Dense
from passage.preprocessing import Tokenizer

# download data at kaggle.com/c/word2vec-nlp-tutorial/data

def clean(texts):
    return [html.fromstring(text).text_content().lower().strip() for text in texts]

if __name__ == "__main__":
    tr_data = pd.read_csv('labeledTrainData.tsv', delimiter='\t')
    trX = clean(tr_data['review'].values)
    trY = tr_data['sentiment'].values

    tokenizer = Tokenizer(min_df=10, max_features=100000)
    trX = tokenizer.fit_transform(trX)

    layers = [
        Embedding(size=256, n_features=tokenizer.n_features),
        GatedRecurrent(size=512, seq_output=False, p_drop=0.75),
        Dense(size=1, activation='sigmoid')
    ]

    model = RNN(layers=layers, cost='bce', updater=Adadelta(lr=0.5))
    model.fit(trX, trY, n_epochs=10)

    te_data = pd.read_csv('testData.tsv', delimiter='\t')
    ids = te_data['id'].values
    teX = clean(te_data['review'].values)
    teX = tokenizer.transform(teX)
    pr_teX = model.predict(teX).flatten()

    pd.DataFrame(np.asarray([ids, pr_teX]).T).to_csv('submission.csv', index=False, header=["id", "sentiment"])
```

RNN imports

preprocessing

load training data

tokenize data

```python
import numpy as np
import pandas as pd
from lxml import html

from passage.models import RNN
from passage.updates import Adadelta
from passage.layers import Embedding, GatedRecurrent, Dense
from passage.preprocessing import Tokenizer

# download data at kaggle.com/c/word2vec-nlp-tutorial/data

def clean(texts):
    return [html.fromstring(text).text_content().lower().strip() for text in texts]

if __name__ == "__main__":
    tr_data = pd.read_csv('labeledTrainData.tsv', delimiter='\t')
    trX = clean(tr_data['review'].values)
    trY = tr_data['sentiment'].values

    tokenizer = Tokenizer(min_df=10, max_features=100000)
    trX = tokenizer.fit_transform(trX)

    layers = [
        Embedding(size=256, n_features=tokenizer.n_features),
        GatedRecurrent(size=512, seq_output=False, p_drop=0.75),
        Dense(size=1, activation='sigmoid')
    ]

    model = RNN(layers=layers, cost='bce', updater=Adadelta(lr=0.5))
    model.fit(trX, trY, n_epochs=10)

    te_data = pd.read_csv('testData.tsv', delimiter='\t')
    ids = te_data['id'].values
    teX = clean(te_data['review'].values)
    teX = tokenizer.transform(teX)
    pr_teX = model.predict(teX).flatten()

    pd.DataFrame(np.asarray([ids, pr_teX]).T).to_csv('submission.csv', index=False, header=["id", "sentiment"])
```

RNN imports

preprocessing

load training data

tokenize data

configure model

```python
import numpy as np
import pandas as pd
from lxml import html
```

RNN imports

```python
from passage.models import RNN
from passage.updates import Adadelta
from passage.layers import Embedding, GatedRecurrent, Dense
from passage.preprocessing import Tokenizer

# download data at kaggle.com/c/word2vec-nlp-tutorial/data
```

preprocessing

```python
def clean(texts):
    return [html.fromstring(text).text_content().lower().strip() for text in texts]

if __name__ == "__main__":
```

load training data

```python
    tr_data = pd.read_csv('labeledTrainData.tsv', delimiter='\t')
    trX = clean(tr_data['review'].values)
    trY = tr_data['sentiment'].values
```

tokenize data

```python
    tokenizer = Tokenizer(min_df=10, max_features=100000)
    trX = tokenizer.fit_transform(trX)
```

configure model

```python
    layers = [
        Embedding(size=256, n_features=tokenizer.n_features),
        GatedRecurrent(size=512, seq_output=False, p_drop=0.75),
        Dense(size=1, activation='sigmoid')
    ]
```

make and train model

```python
    model = RNN(layers=layers, cost='bce', updater=Adadelta(lr=0.5))
    model.fit(trX, trY, n_epochs=10)

    te_data = pd.read_csv('testData.tsv', delimiter='\t')
    ids = te_data['id'].values
    teX = clean(te_data['review'].values)
    teX = tokenizer.transform(teX)
    pr_teX = model.predict(teX).flatten()

    pd.DataFrame(np.asarray([ids, pr_teX]).T).to_csv('submission.csv', index=False, header=["id", "sentiment"])
```

```python
import numpy as np
import pandas as pd
from lxml import html
```

RNN imports

```python
from passage.models import RNN
from passage.updates import Adadelta
from passage.layers import Embedding, GatedRecurrent, Dense
from passage.preprocessing import Tokenizer

# download data at kaggle.com/c/word2vec-nlp-tutorial/data
```

preprocessing

```python
def clean(texts):
    return [html.fromstring(text).text_content().lower().strip() for text in texts]

if __name__ == "__main__":
```

load training data

```python
    tr_data = pd.read_csv('labeledTrainData.tsv', delimiter='\t')
    trX = clean(tr_data['review'].values)
    trY = tr_data['sentiment'].values
```

tokenize data

```python
    tokenizer = Tokenizer(min_df=10, max_features=100000)
    trX = tokenizer.fit_transform(trX)
```

configure model

```python
    layers = [
        Embedding(size=256, n_features=tokenizer.n_features),
        GatedRecurrent(size=512, seq_output=False, p_drop=0.75),
        Dense(size=1, activation='sigmoid')
    ]
```

make and train model

```python
    model = RNN(layers=layers, cost='bce', updater=Adadelta(lr=0.5))
    model.fit(trX, trY, n_epochs=10)
```

load test data

```python
    te_data = pd.read_csv('testData.tsv', delimiter='\t')
    ids = te_data['id'].values
    teX = clean(te_data['review'].values)
    teX = tokenizer.transform(teX)
    pr_teX = model.predict(teX).flatten()

    pd.DataFrame(np.asarray([ids, pr_teX]).T).to_csv('submission.csv', index=False, header=["id", "sentiment"])
```

```python
import numpy as np
import pandas as pd
from lxml import html
```

RNN imports

```python
from passage.models import RNN
from passage.updates import Adadelta
from passage.layers import Embedding, GatedRecurrent, Dense
from passage.preprocessing import Tokenizer

# download data at kaggle.com/c/word2vec-nlp-tutorial/data
```

preprocessing

```python
def clean(texts):
    return [html.fromstring(text).text_content().lower().strip() for text in texts]

if __name__ == "__main__":
```

load training data

```python
    tr_data = pd.read_csv('labeledTrainData.tsv', delimiter='\t')
    trX = clean(tr_data['review'].values)
    trY = tr_data['sentiment'].values
```

tokenize data

```python
    tokenizer = Tokenizer(min_df=10, max_features=100000)
    trX = tokenizer.fit_transform(trX)
```

configure model

```python
    layers = [
        Embedding(size=256, n_features=tokenizer.n_features),
        GatedRecurrent(size=512, seq_output=False, p_drop=0.75),
        Dense(size=1, activation='sigmoid')
    ]
```

make and train model

```python
    model = RNN(layers=layers, cost='bce', updater=Adadelta(lr=0.5))
    model.fit(trX, trY, n_epochs=10)
```

load test data

```python
    te_data = pd.read_csv('testData.tsv', delimiter='\t')
    ids = te_data['id'].values
    teX = clean(te_data['review'].values)
```

predict on test data

```python
    teX = tokenizer.transform(teX)
    pr_teX = model.predict(teX).flatten()

    pd.DataFrame(np.asarray([ids, pr_teX]).T).to_csv('submission.csv', index=False, header=["id", "sentiment"])
```

The results

#	Δ1w	Team Name * in the money	Score	Entries	Last Submission UTC (Best - Last Submission)
1	—	Rani Nelken *	0.97959	1	Sat, 07 Mar 2015 03:46:43
2	—	honzas	0.97396	19	Thu, 19 Mar 2015 21:19:24 (-13.7h)
3	—	broucek	0.97326	14	Thu, 02 Apr 2015 05:26:23
4	—	sbalajis	0.97119	4	Sun, 26 Apr 2015 00:16:34
5	—	London ML Learners	0.97064	15	Sat, 21 Mar 2015 21:37:38 (-3.8h)
6	new	Mathieu Cliche	0.97009	1	Mon, 20 Apr 2015 12:07:04
7	↓1	pstanisl	0.97003	13	Wed, 08 Apr 2015 15:47:50
8	↑189	shamzarmy	0.96992	18	Wed, 22 Apr 2015 10:32:23
9	↑3	Cristian	0.96980	13	Tue, 21 Apr 2015 04:09:07
10	↓3	Gideon Wulfsohn	0.96974	7	Sun, 26 Apr 2015 02:27:50 (-10.7d)
11	↓3	satomacoto	0.96866	15	Thu, 02 Apr 2015 10:48:15
12	↓3	leotywy	0.96863	6	Thu, 26 Mar 2015 00:58:23 (-11.7h)

Top 10! - barely :)

Summary

- RNNs look to be a competitive tool in certain situations for text analysis.
- Especially if you have a large 1M+ example dataset
    - A GPU or great patience is essential
- Otherwise it can be difficult to justify over linear models
    - Speed
    - Complexity
    - Poor generalization with small datasets

Contact

alec@indico.io

We're hiring!

- Data Engineer
- Infrastructure Engineer

- Interested?
  - contact@indico.io (or talk-to/email me after pres.)

Questions?