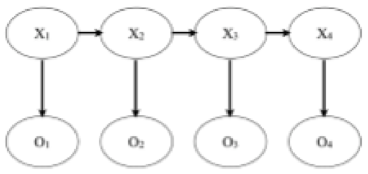
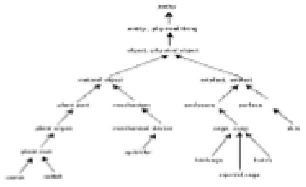
Deep Learning for Text Processing with Focus on Word Embedding: Concept and Applications

Mohamad Ivan Fanany, Dr. Eng.,

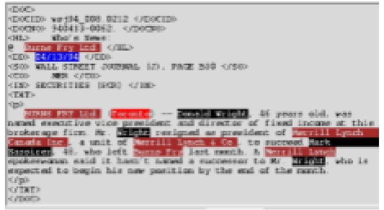
Deep Learning



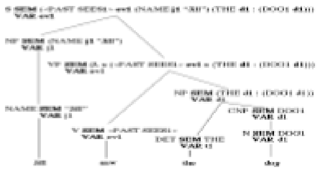
NER



WordNet



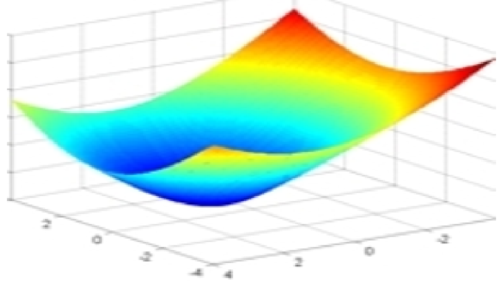
SRL



Parser

Most current machine learning works well because of human-designed representations and input features

Machine learning becomes just optimizing weights to best make a final prediction



Representation learning attempts to automatically learn good features or representations

Deep learning algorithms attempt to learn multiple levels of representation of increasing complexity/abstraction

A Deep Architecture

Mainly, work has explored deep belief networks (DBNs), Markov Random Fields with multiple layers, and various types of multiple-layer neural networks

Output layer

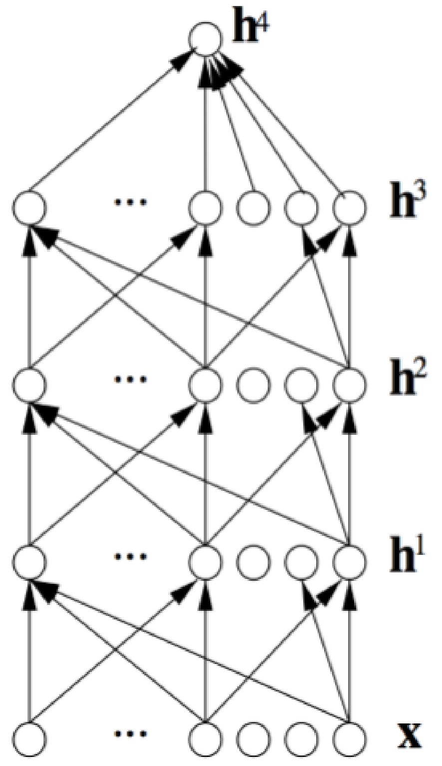
Here predicting a supervised target

Hidden layers

These learn more abstract representations as you head up

Input layer

Raw sensory inputs (roughly)



Five Reasons to Explore Deep Learning

#1 Learning representations

Handcrafting features is time-consuming

The features are often both over-specified and incomplete

The work has to be done again for each task/domain/…

We must move beyond handcrafted features and simple ML

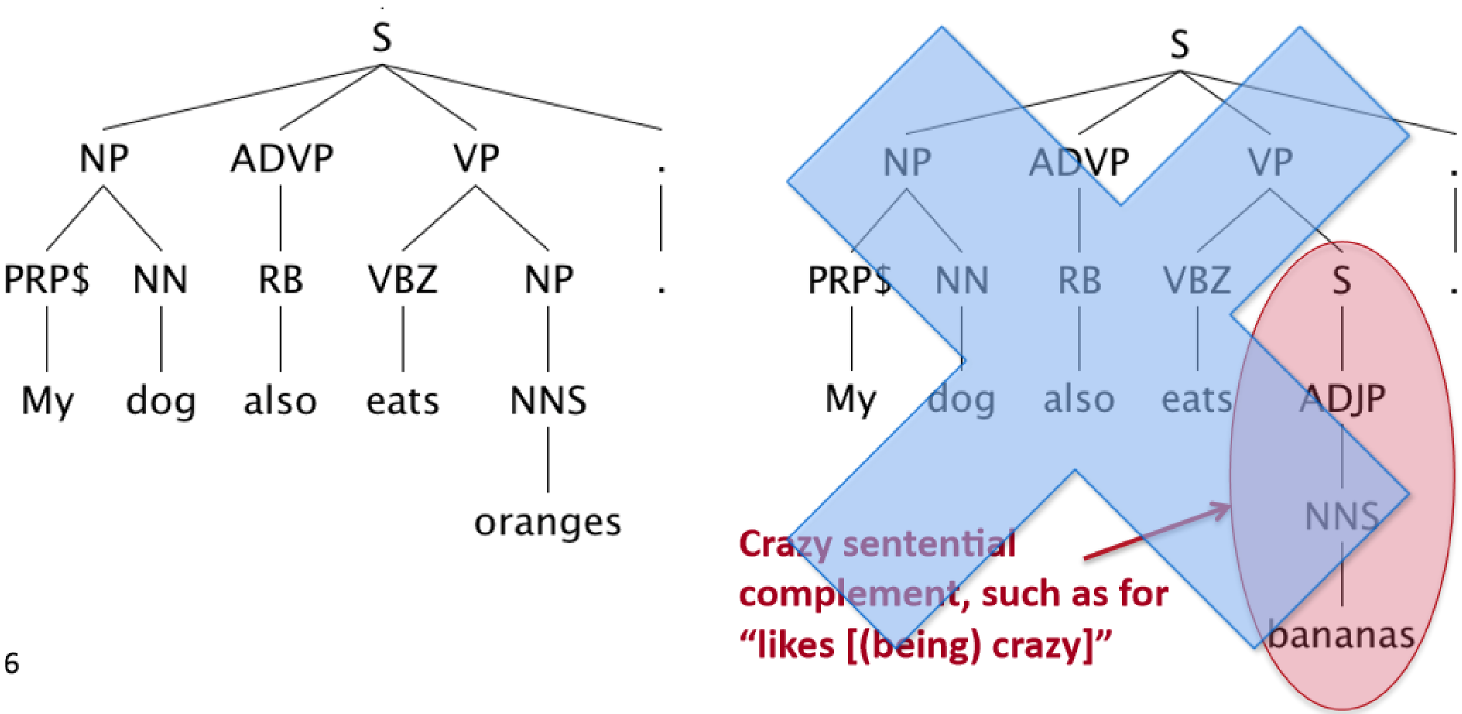
Humans develop representations for learning and reasoning

Our computers should do the same

Deep learning provides a way of doing this

#2 The need for distributed representations

Current NLP systems are incredibly fragile because of their atomic symbol representations



#2 The need for distributional & distributed representations

Learned word representations help enormously in NLP

They provide a powerful similarity model for words

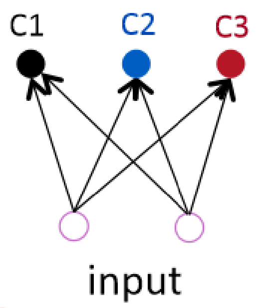
Distributional similarity based word clusters greatly help most applications

+1.4% F1 Dependency Parsing 15.2% error reduction (Koo & Collins 2008, Brown clustering)

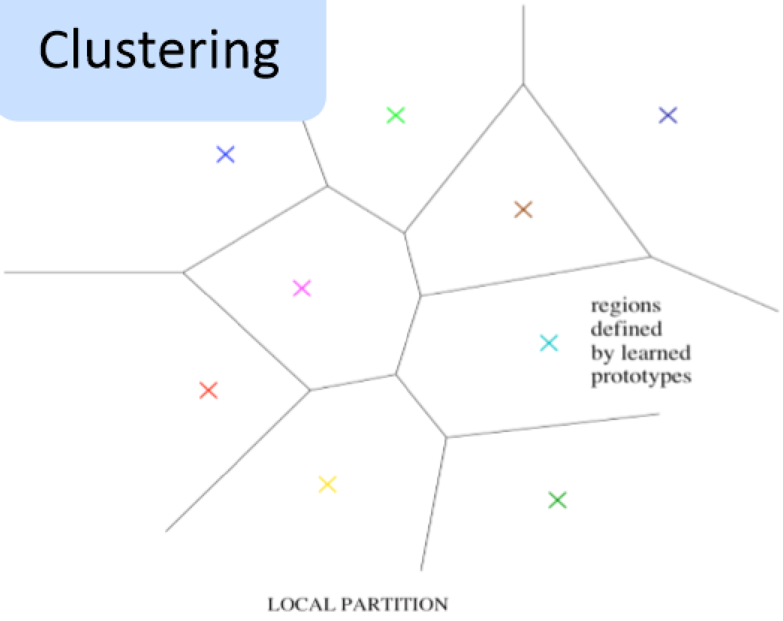
+3.4% F1 Named Entity Recognition 23.7% error reduction (Stanford NER, exchange clustering)

Distributed representations can do even better by representing more dimensions of similarity

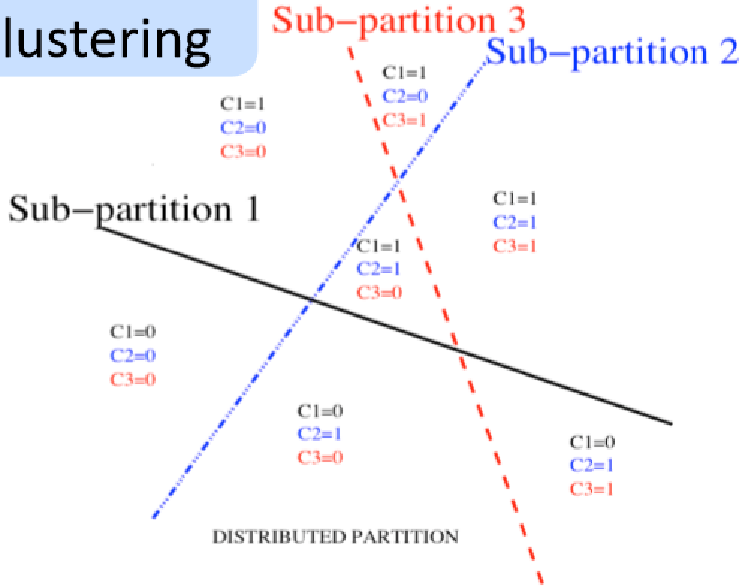
#2 The need for distributed representations



Clustering



Multi-Clustering



Learning features that are not mutually exclusive can be exponentially more efficient than nearest-neighbor-like or clustering-like models

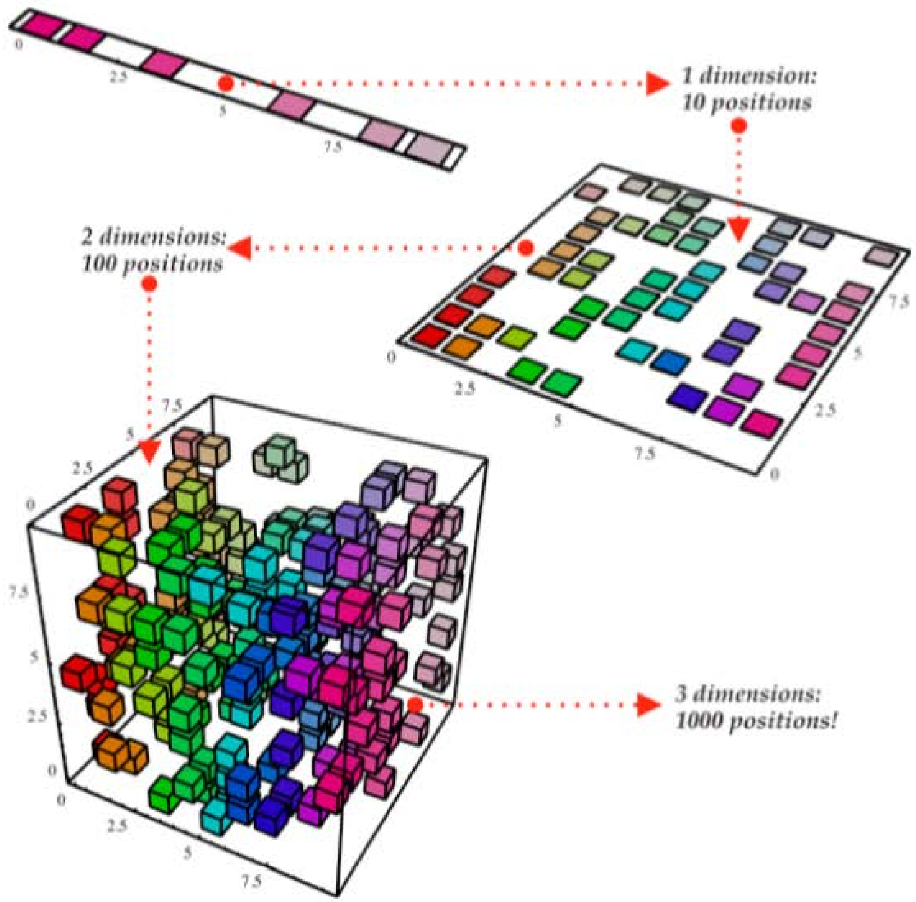
Distributed representations deal with the curse of dimensionality

Generalizing locally (e.g., nearest neighbors) requires representative examples for all relevant variations!

Classic solutions:

- Manual feature design
- Assuming a smooth target function (e.g., linear models)
- Kernel methods (linear in terms of kernel based on data points)

Neural networks parameterize and learn a "similarity" kernel



#3 Unsupervised feature and weight learning

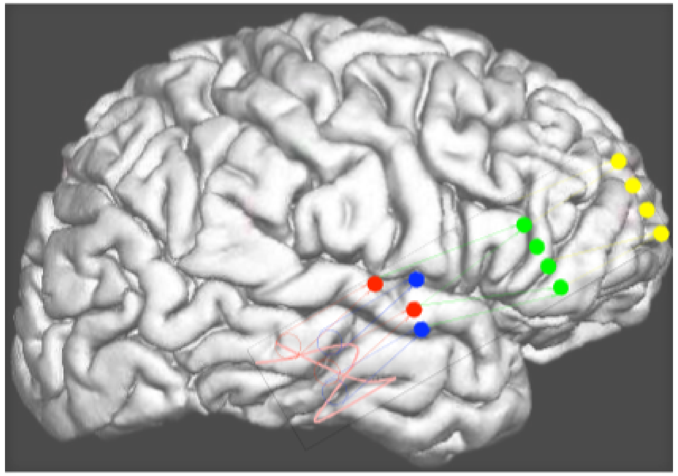
Today, most practical, good NLP& ML methods require labeled training data (i.e., supervised learning)

- But almost all data is unlabeled

Most information must be acquired **unsupervised**

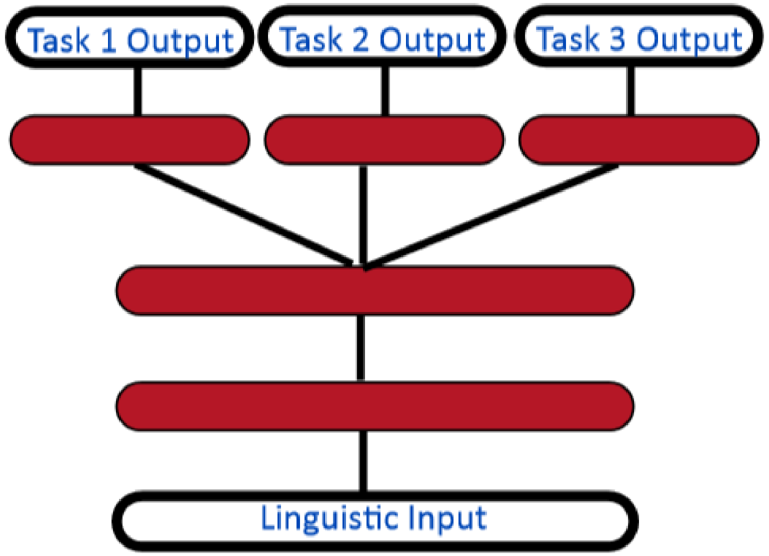
- Fortunately, a good model of observed data can really help you learn classification decisions

#4 Learning multiple levels of representation



Biologically inspired learning

- The cortex seems to have a generic learning algorithm
- The brain has a deep architecture



We need good intermediate representations that can be shared across tasks

Multiple levels of latent variables allow combinatorial sharing of statistical strength

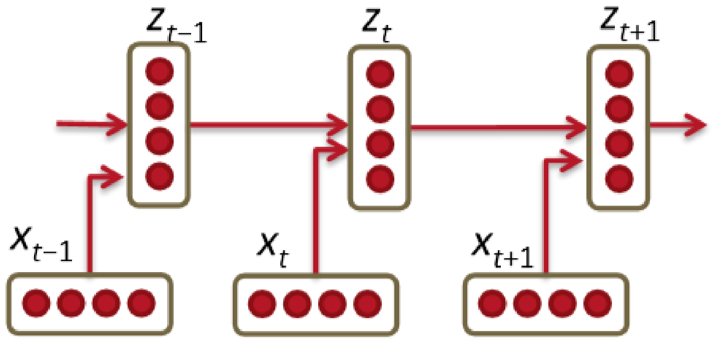
- Insufficient model depth can be exponentially inefficient

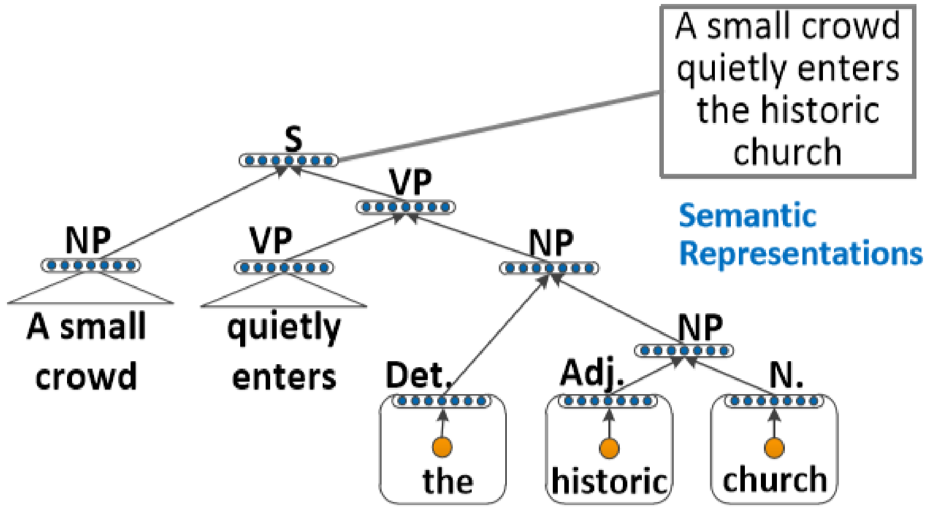
Handling the recursivity of human language

Human sentences are composed from words and phrases

We need compositionality in our ML models

Recursion: the same operator (same parameters) is applied repeatedly on different components





#5 Why now?

Despite prior investigation and understanding of many of the algorithmic techniques …

Before 2006 training deep architectures was **unsuccessful** ☹

What has changed?

- New methods for unsupervised pre-training have been developed (Restricted Boltzmann Machines = RBMs, autoencoders, contrastive estimation, etc.)
- More efficient parameter estimation methods
- Better understanding of model regularization

Contents

Natural Language Processing

- Mostly works with text data.
- Could be applied to music, bioinformatics, speech, etc.
- *Machine learning* perspective: NL is a sequence of variable-length sequences of high-dimensional vectors.

How to best represent the word for learning?

Word Embeddings

- A set of language modeling and feature learning techniques in natural language processing.
- Mapped words or phrases from the vocabulary to vectors of real numbers.
- Mathematical embedding from a space with one dimension per word to a continuous vector space with much lower dimension.

One-Hot Encoding

```
V = {zebra, horse, school, summer}

 v(zebra) = [1, 0, 0, 0]
 v(horse) = [0, 1, 0, 0]
v(school) = [0, 0, 1, 0]
v(summer) = [0, 0, 0, 1]
```

(+) Pros:

Simplicity

(-) Cons:

Can be memory inefficient

Notion of "word similarity" is undefined

Distributional Representation

Is there a representation that preserves the similarities of word meanings?

$$d(v(\text{zebra}), v(\text{horse})) < d(v(\text{zebra}), v(\text{summer}))$$

“You shall know a word by the company it keeps” - John Rupert Firth

Paris is the capital of France.

Berlin is the capital of Germany.

Paris : France :: Berlin : Germany

$$v(\text{Paris})-v(\text{France}) \approx v(\text{Berlin})-v(\text{Germany})$$

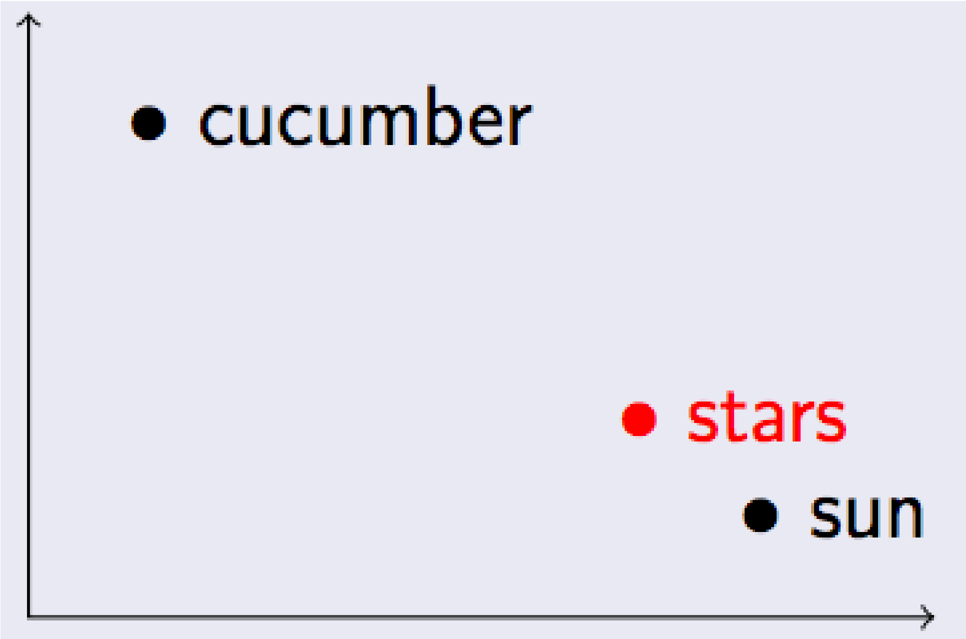
Distributional Representation

```
he curtains open and the stars shining in on the barely
ars and the cold , close stars " . And neither of the w
rough the night with the stars shining so brightly , it
made in the light of the stars . It all boils down , wr
 surely under the bright stars , thrilled by ice-white
sun , the seasons of the stars ? Home , alone , Jay pla
m is dazzling snow , the stars have risen full and cold
un and the temple of the stars , driving out of the hug
 in the dark and now the stars rise , full and amber a
bird on the shape of the stars over the trees in front
 But I could n't see the stars or the moon , only the
 they love the sun , the stars and the stars . None of
r the light of the shiny stars . The plash of flowing w
man 's first look at the stars ; various exhibits , aer
rief information on both stars and constellations, inc
```

www.cs.ox.ac.uk/files/6605/aclVectorTutorial.pdf

Distributional Representation

	shining	bright	trees	dark	look
stars	38	45	2	27	12



www.cs.ox.ac.uk/files/6605/aclVectorTutorial.pdf

Distributional Representation

(+) Pros:

Simplicity (BOW assumption)

Has notion of word similarity

(-) Cons:

Can be memory inefficient

Latent Semantic Analysis, Latent Dirichlet Allocation, Self-organizing map, Hyperspace Analog to Language, Independent Component Analysis, Random Indexing.

Distributed Representation

$\boldsymbol{V}$ is a vocabulary

$$\boldsymbol{w}_i \in \boldsymbol{V}$$

$$\boldsymbol{v}(\boldsymbol{w}_i) \in \boldsymbol{R}^n$$

$\boldsymbol{v}(\boldsymbol{w}_i)$ is a low-dimensional, learnable, dense word vector

Distributed Representation

colah.github.io/posts/2014-07-NLP-RNNs-Representations

Distributed Representation

(+) Pros:

Has notion of word similarity

Memory Efficient (low dimensional)

(-) Cons:

Computationally intensive

Distributed Representation as a Lookup Table

$\boldsymbol{W}$ is a matrix whose rows are $\boldsymbol{v}(\boldsymbol{w}_i) \in \boldsymbol{R}^n$

$\boldsymbol{v}(\boldsymbol{w}_i)$ returns $\boldsymbol{i}^{th}$ row of $\boldsymbol{W}$

Statistical Language Model

A sentence $\boldsymbol{s} = (x_1, x_2, \cdots, x_T)$

How likely is $\boldsymbol{s}$?

$p(x_1, x_2, \cdots, x_T)$

According to the chain rule (probability)

$$p(x_1, x_2, \cdots, x_T) = \prod_{t=1}^{T} p(x_t | x_1, x_2, \cdots, x_T)$$

N-gram Models

n-th order Markov assumption

$$p(x_1, x_2, \cdots, x_T) \approx \prod_{t=1}^{T} p(x_t | x_{t-n}, \cdots, x_{t-1})$$

Bigram model of $\boldsymbol{s} = (a, cute, bird, is, on, the, tree, .)$

1. How likely does ‘a’ follow ‘<S>’?
2. How likely does ‘cute’ follow ‘a’?
3. How likely does ‘is’ follow ‘bird’?
4. …

n-gram Models

n-th order Markov assumption

$$p(x_1, x_2, \cdots, x_T) \approx \prod_{t=1}^{T} p(x_t | x_{t-n}, \cdots, x_{t-1})$$

Bigram model of $\boldsymbol{s} = (a, cute, bird, is, on, the, tree, .)$

$$p\left(w_i | w_{i-(n-1)}, \cdots, w_{i-1}\right) = \frac{count\left(w_{i-(n-1)}, \cdots, w_{i-1}, w_i\right)}{count\left(w_{i-(n-1)}, \cdots, w_{i-1}\right)}$$

the counts are obtained from a training corpus

n-gram Models

(+) Pros:

- Computationally efficient

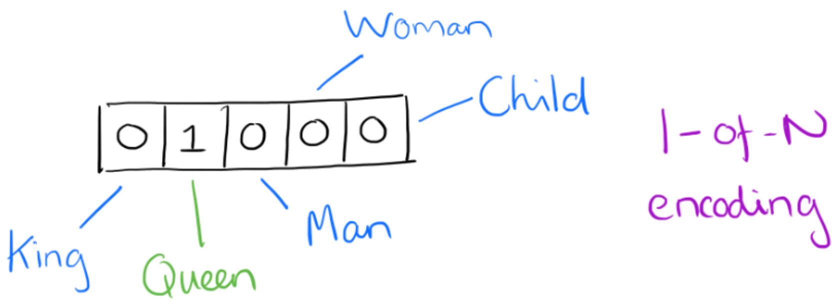
(-) Cons:

- Data sparsity
- Lack of generalization:
  - [ride a *horse*], [ride a *llama*], [ride a *zebra*]

Word Embedding versus Other Representations

- They use words as their context
- More natural form of semantic similarity
- Human understanding perspective
- The technique of choice for vectorizing text for NLP Tasks:
  - Text classification
  - Document clustering
  - Part of speech tagging
  - Named entity recognition
  - Sentiment analysis
  - …

word2vec vs one-hot

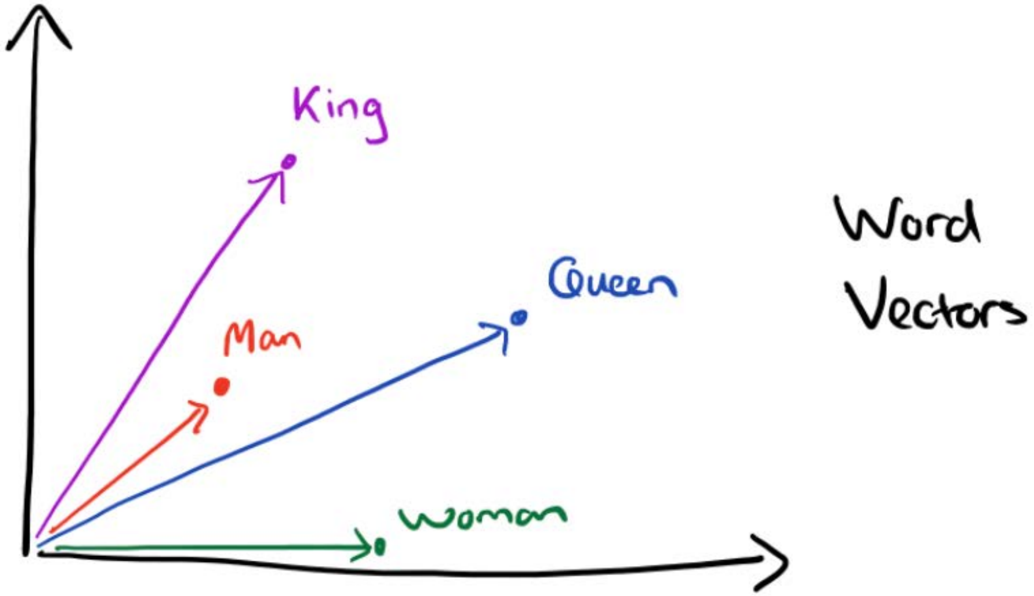


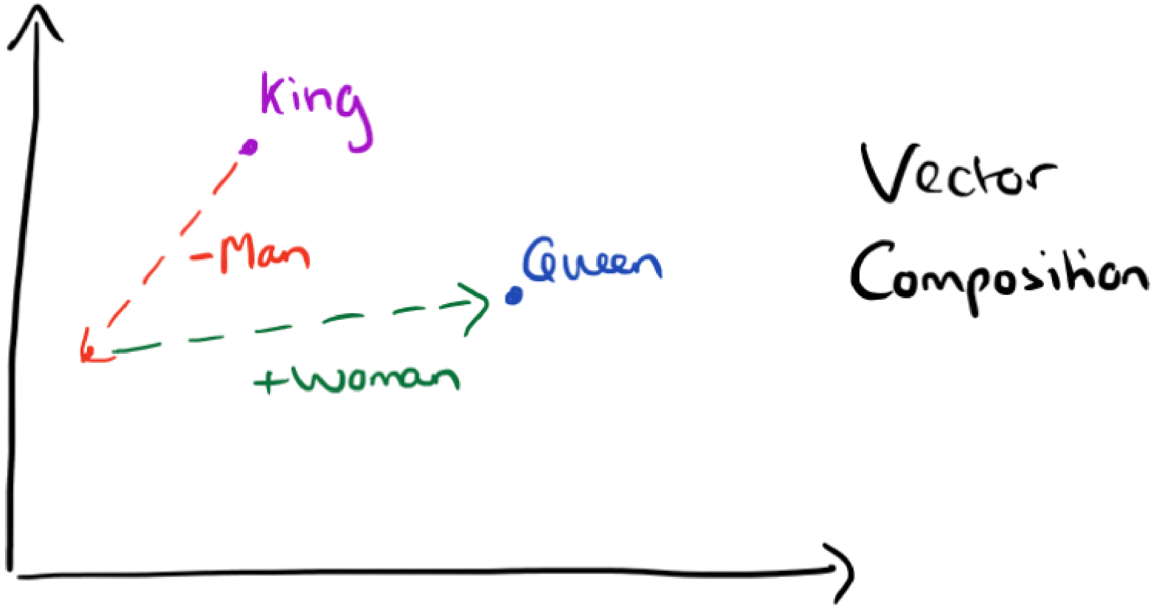
one-hot

word2vec

https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

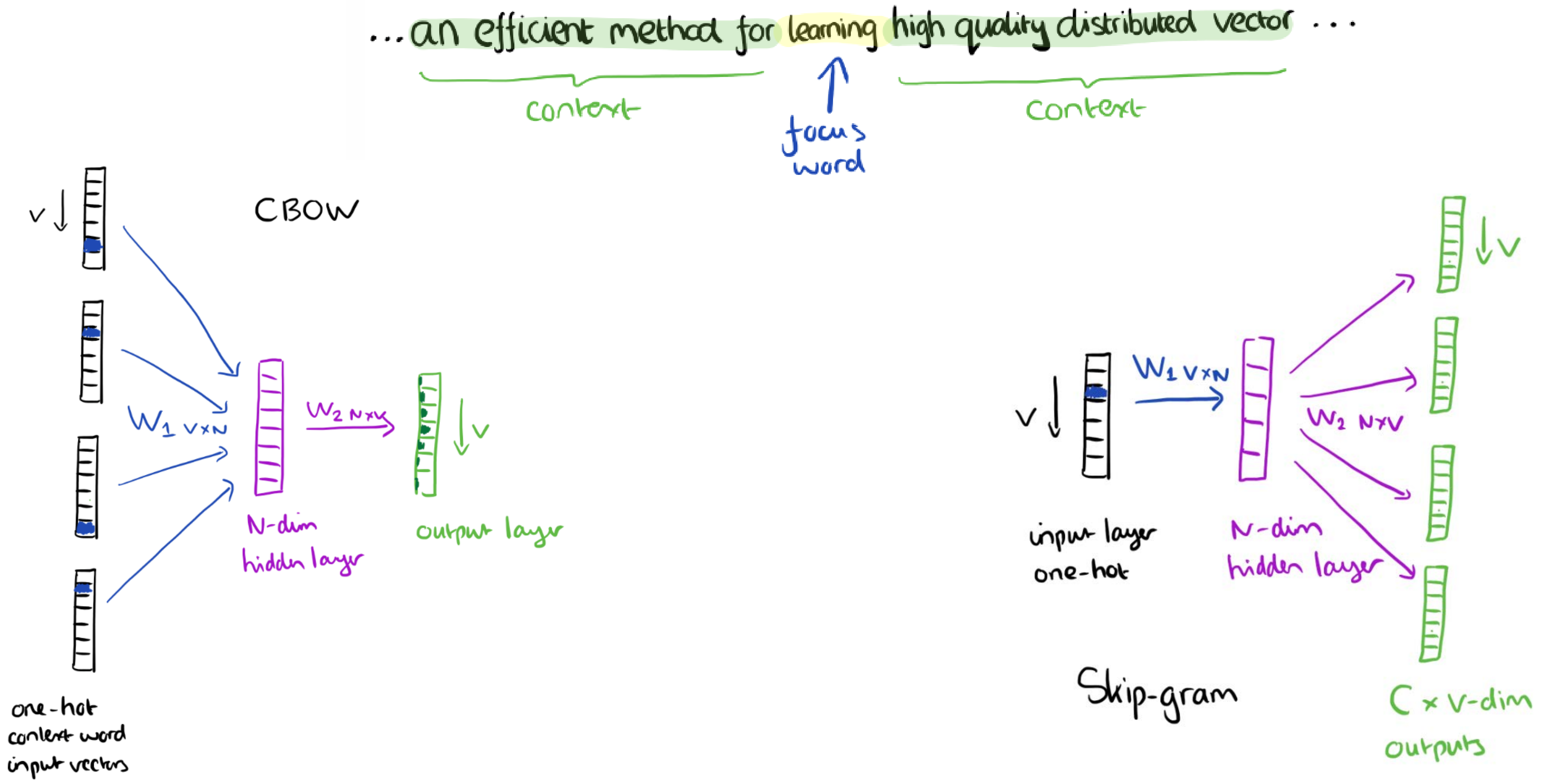
Reasoning with word vectors





https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

word2vec



https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

CBOW vs Skip-gram

- CBOW:
  - Predicts the current word given a window of surrounding words (context)
  - The order of the context words does not influence the prediction (bag of words assumption)

- Skip-gram:
  - Predict the surrounding words (context) given the center word.

CBOW is faster but skip-gram does a better job at predicting infrequent words.

Skip-gram implementation in Keras

I love green eggs and ham.

([I, green], love)
([love, eggs], green)
([green, and], eggs)
...

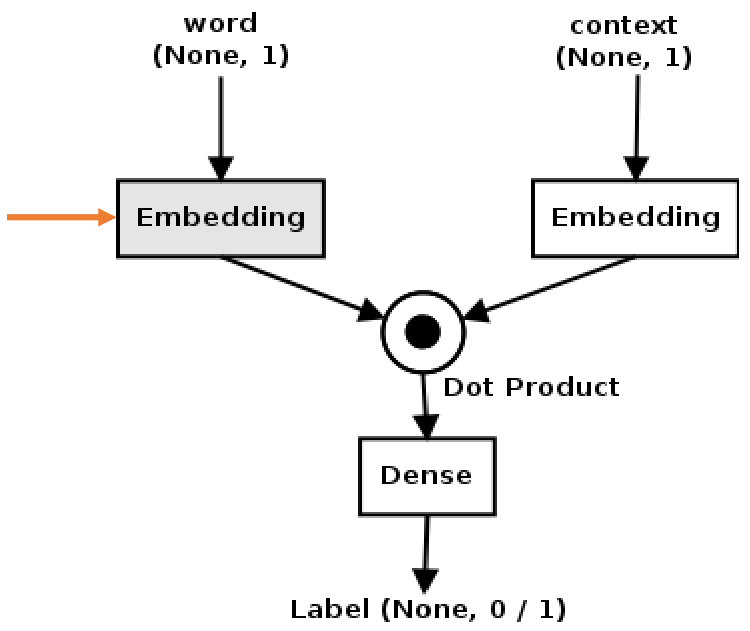
Training Data (Pairs)

(love, I), (love, green),
(green, love), (green, eggs),
(eggs, green), (eggs, and),
...

Expected Prediction

Takes in a word vector and a context vector, learns to predict one or zero depending whether it sees a positive or negative sample.

Deliverables



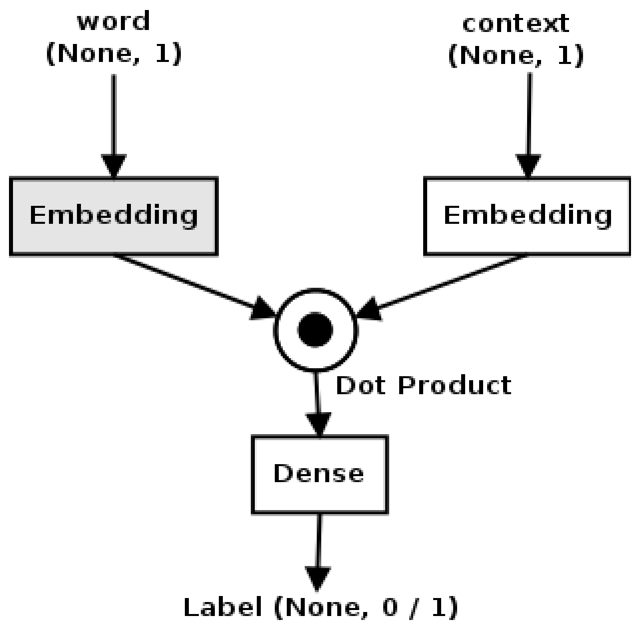
https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

Skip-gram implementation in Keras

```
from keras.layers import Merge
from keras.layers.core import Dense, Reshape
from keras.layers.embeddings import Embedding
from keras.models import Sequential

vocab_size = 5000
embed_size = 300
```

```
word_model = Sequential()
word_model.add(Embedding(vocab_size, embed_size,
                         embeddings_initializer="glorot_uniform",
                         input_length=1))
word_model.add(Reshape((embed_size, )))
```

```
context_model = Sequential()
context_model.add(Embedding(vocab_size, embed_size,
                  embeddings_initializer="glorot_uniform",
                  input_length=1))
context_model.add(Reshape((embed_size,)))
```



https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

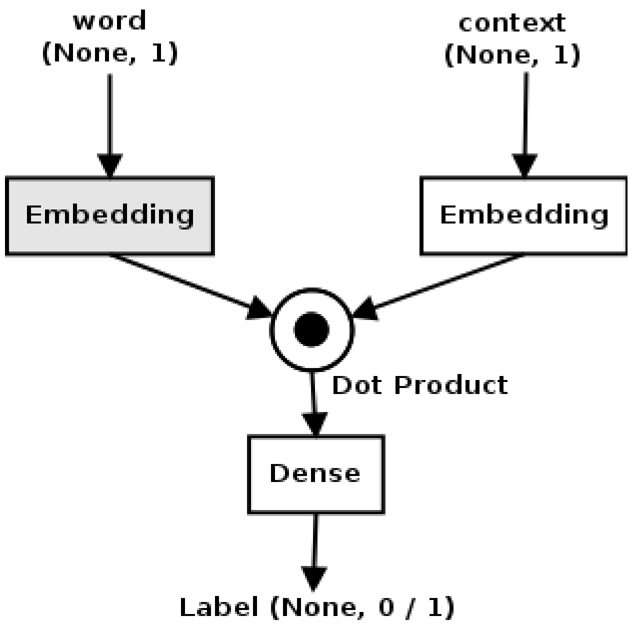
Skip-gram implementation in Keras

```
model = Sequential()
model.add(Merge([word_model, context_model], mode="dot"))
model.add(Dense(1, init="glorot_uniform", activation="sigmoid"))
model.compile(loss="mean_squared_error", optimizer="adam")
```

```
from keras.preprocessing.text import *
from keras.preprocessing.sequence import skipgrams

text = "I love green eggs and ham ."
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
```

```
word2id = tokenizer.word_index
id2word = {v:k for k, v in word2id.items()}
```



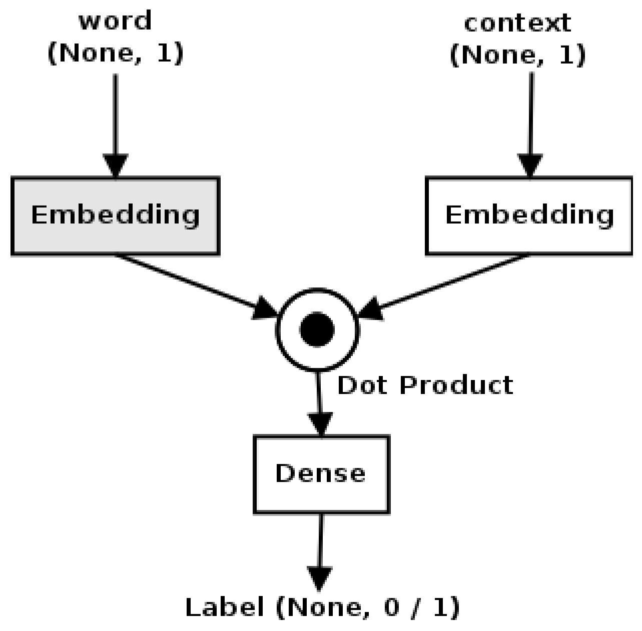
https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

Skip-gram implementation in Keras

```
wids = [word2id[w] for w in text_to_word_sequence(text)]
pairs, labels = skipgrams(wids, len(word2id))
print(len(pairs), len(labels))
for i in range(10):
    print("({:s} ({:d}), {:s} ({:d})) -> {:d}".format(
          id2word[pairs[i][0]], pairs[i][0],
          id2word[pairs[i][1]], pairs[i][1],
          labels[i]))
```

```
(and (1), ham (3)) -> 0
(green (6), i (4)) -> 0
(love (2), i (4)) -> 1
(and (1), love (2)) -> 0
(love (2), eggs (5)) -> 0
(ham (3), ham (3)) -> 0
(green (6), and (1)) -> 1
(eggs (5), love (2)) -> 1
(i (4), ham (3)) -> 0
(and (1), green (6)) -> 1
```

The first 10 of 56 (pair, label)



https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

CBOW implementation in Keras

I love green eggs and ham.

([I, green], love)
([love, eggs], green)
([green, and], eggs)
...

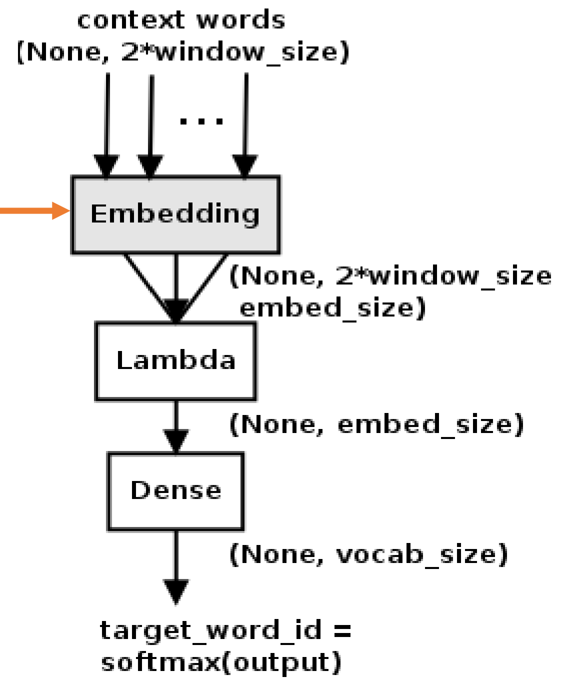
Training Data (Pairs)

(love, I), (love, green),
(green, love), (green, eggs),
(eggs, green), (eggs, and),
...

Expected Prediction

Takes the context words as input
Predicts the target word

Deliverables

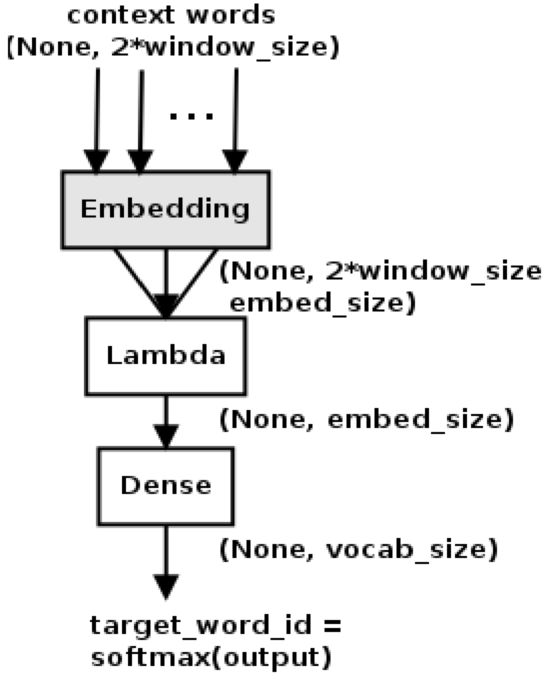


https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

CBOW implementation in Keras

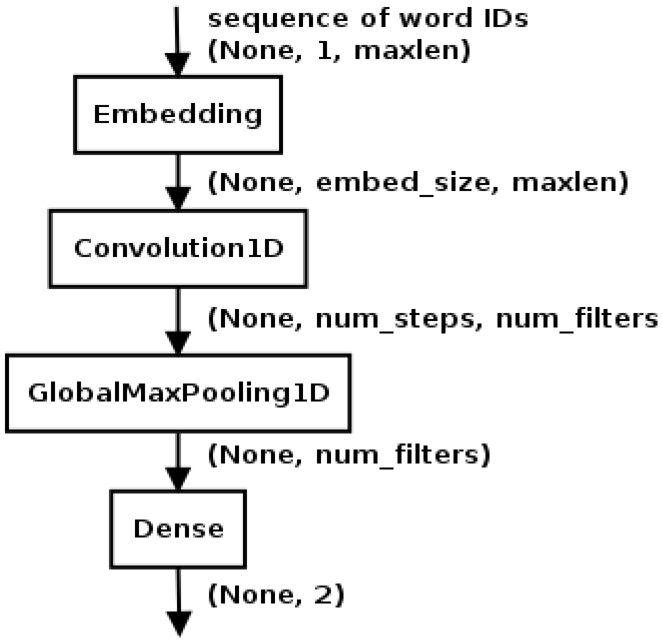
```
from keras.models import Sequential
from keras.layers.core import Dense, Lambda
from keras.layers.embeddings import Embedding
import keras.backend as K

vocab_size = 5000
embed_size = 300
window_size = 1
```

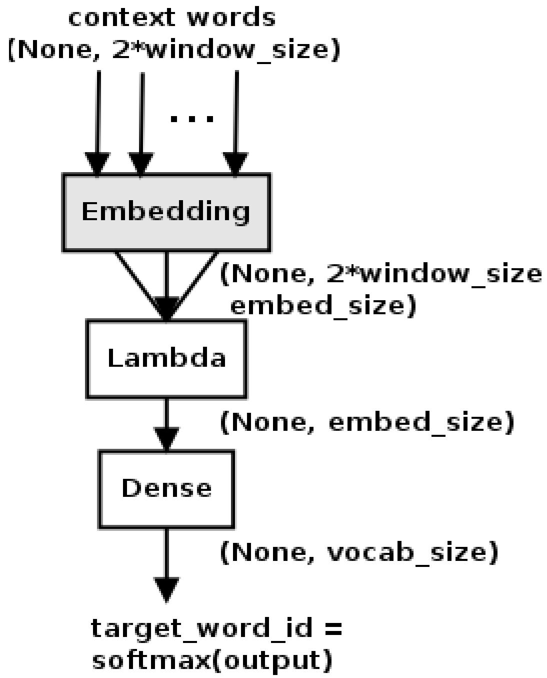
```
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embed_size,
                    embeddings_initializer='glorot_uniform',
                    input_length=window_size*2))
model.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=  (embed_size,)))
model.add(Dense(vocab_size, kernel_initializer='glorot_uniform', activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer="adam")
```



https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

CBOW from the scratch in Keras



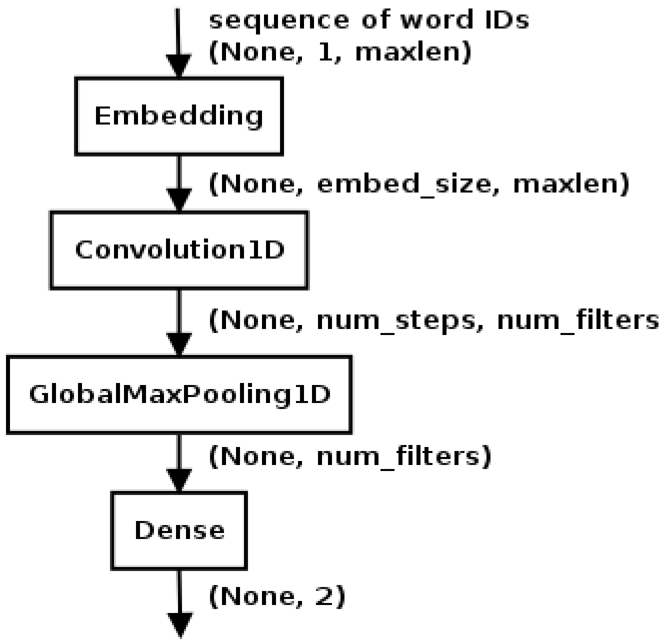


https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

CBOW from the scratch in Keras

```
from keras.layers.core import Dense, Dropout, SpatialDropout1D
from keras.layers.convolutional import Conv1D
from keras.layers.embeddings import Embedding
from keras.layers.pooling import GlobalMaxPooling1D
from kera
s.models import Sequential
from keras.preprocessing.sequence import pad_sequences
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
import collections
import matplotlib.pyplot as plt
import nltk
import numpy as np

np.random.seed(42)
```

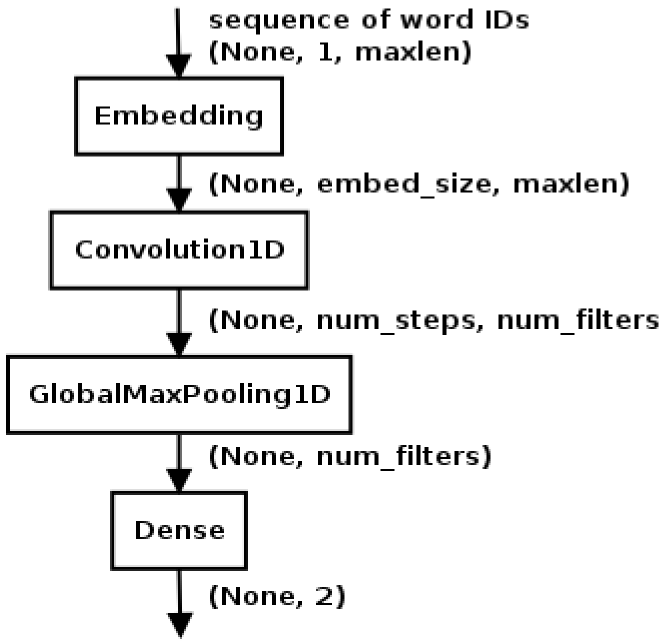
```
INPUT_FILE = "../data/umich-sentiment-train.txt"
VOCAB_SIZE = 5000
EMBED_SIZE = 100
NUM_FILTERS = 256
NUM_WORDS = 3
BATCH_SIZE = 64
NUM_EPOCHS = 20
```



https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

CBOW from the scratch in Keras

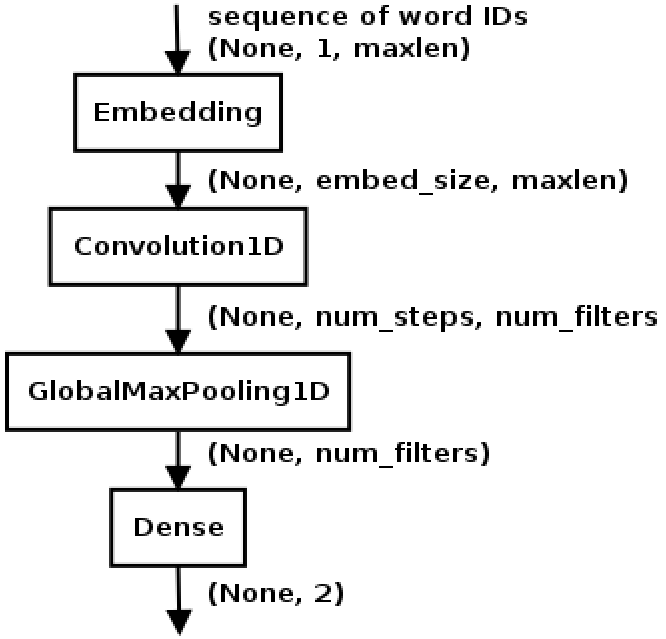
```
counter = collections.Counter()
fin = open(INPUT_FILE, "rb")
maxlen = 0
for line in fin:
    _, sent = line.strip().split("t")
    words = [x.lower() for x in   nltk.word_tokenize(sent)]
    if len(words) > maxlen:
        maxlen = len(words)
    for word in words:
        counter[word] += 1
fin.close()

word2index = collections.defaultdict(int)
for wid, word in enumerate(counter.most_common(VOCAB_SIZE)):
    word2index[word[0]] = wid + 1
vocab_size = len(word2index) + 1
index2word = {v:k for k, v in word2index.items()}
```



https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

CBOW from the scratch in Keras

```
xs, ys = [], []
fin = open(INPUT_FILE, "rb")
for line in fin:
    label, sent = line.strip().split("t")
    ys.append(int(label))
    words = [x.lower() for x in nltk.word_tokenize(sent)]
    wids = [word2index[word] for word in words]
    xs.append(wids)
fin.close()
X = pad_sequences(xs, maxlen=maxlen)
Y = np_utils.to_categorical(ys)
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.3, random_state=42)
```

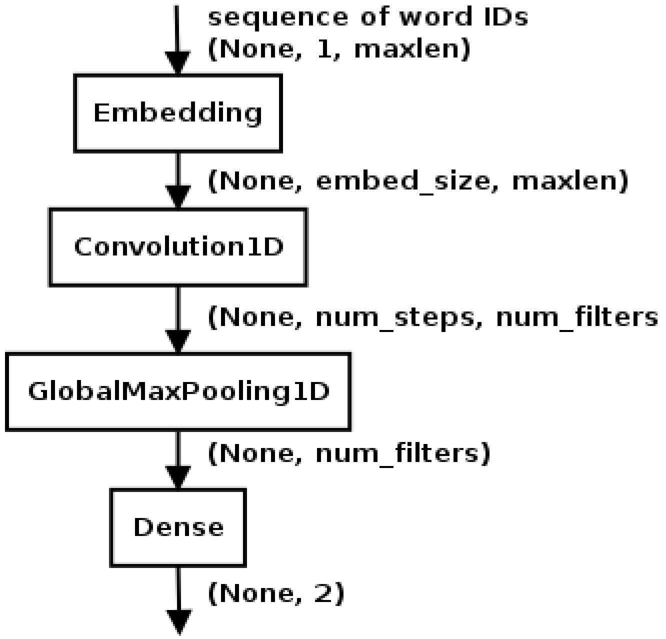


https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

CBOW from the scratch in Keras

```
model = Sequential()
model.add(Embedding(vocab_size, EMBED_SIZE, input_length=maxlen)
model.add(SpatialDropout1D(Dropout(0.2)))
model.add(Conv1D(filters=NUM_FILTERS, kernel_size=NUM_WORDS,
activation="relu"))
model.add(GlobalMaxPooling1D())
model.add(Dense(2, activation="softmax"))
```

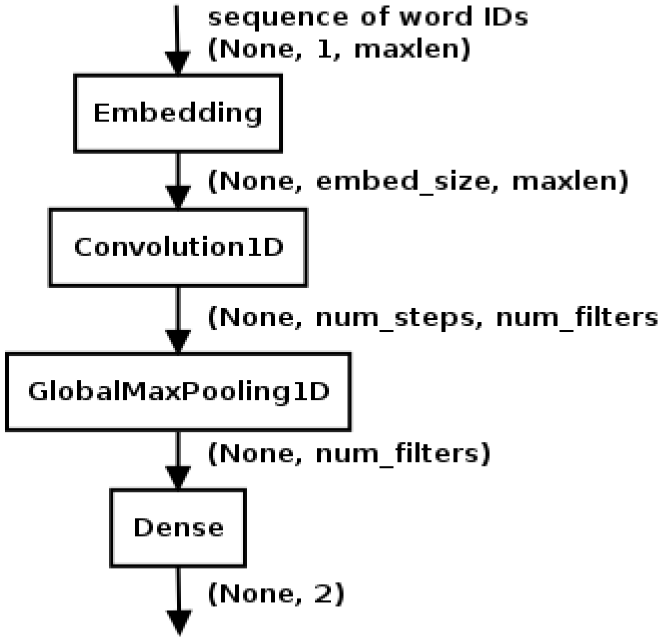
```
model.compile(loss="categorical_crossentropy", optimizer="adam",
              metrics=["accuracy"])
history = model.fit(Xtrain, Ytrain, batch_size=BATCH_SIZE,
                    epochs=NUM_EPOCHS,
                    validation_data=(Xtest, Ytest))
```



https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

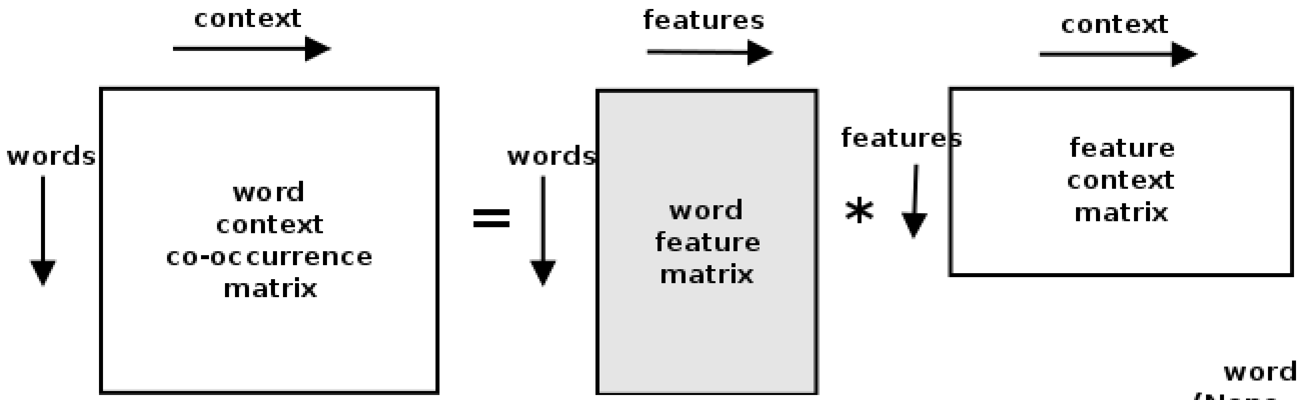
CBOW from the scratch in Keras

```
Epoch 9/20
4960/4960 [==============================] - 3s - loss: 0.0337 - acc: 0.9855 - val_loss: 0.0263 - val_acc: 0.9882
Epoch 10/20
4960/4960 [==============================] - 3s - loss: 0.0369 - acc: 0.9843 - val_loss: 0.0277 - val_acc: 0.9878
Epoch 11/20
4960/4960 [==============================] - 3s - loss: 0.0331 - acc: 0.9881 - val_loss: 0.0303 - val_acc: 0.9878
Epoch 12/20
4960/4960 [==============================] - 3s - loss: 0.0289 - acc: 0.9879 - val_loss: 0.0291 - val_acc: 0.9882
Epoch 13/20
4960/4960 [==============================] - 3s - loss: 0.0261 - acc: 0.9901 - val_loss: 0.0305 - val_acc: 0.9878
Epoch 14/20
4960/4960 [==============================] - 3s - loss: 0.0261 - acc: 0.9895 - val_loss: 0.0310 - val_acc: 0.9859
Epoch 15/20
4960/4960 [==============================] - 3s - loss: 0.0355 - acc: 0.9857 - val_loss: 0.0307 - val_acc: 0.9873
Epoch 16/20
4960/4960 [==============================] - 3s - loss: 0.0247 - acc: 0.9893 - val_loss: 0.0283 - val_acc: 0.9868
Epoch 17/20
4960/4960 [==============================] - 3s - loss: 0.0249 - acc: 0.9891 - val_loss: 0.0329 - val_acc: 0.9854
Epoch 18/20
4960/4960 [==============================] - 3s - loss: 0.0299 - acc: 0.9895 - val_loss: 0.0285 - val_acc: 0.9882
Epoch 19/20
4960/4960 [==============================] - 3s - loss: 0.0282 - acc: 0.9887 - val_loss: 0.0287 - val_acc: 0.9882
Epoch 20/20
4960/4960 [==============================] - 3s - loss: 0.0401 - acc: 0.9839 - val_loss: 0.0311 - val_acc: 0.9878

2126/2126 [==============================] - 0s
Test score: 0.031, accuracy: 0.986
```



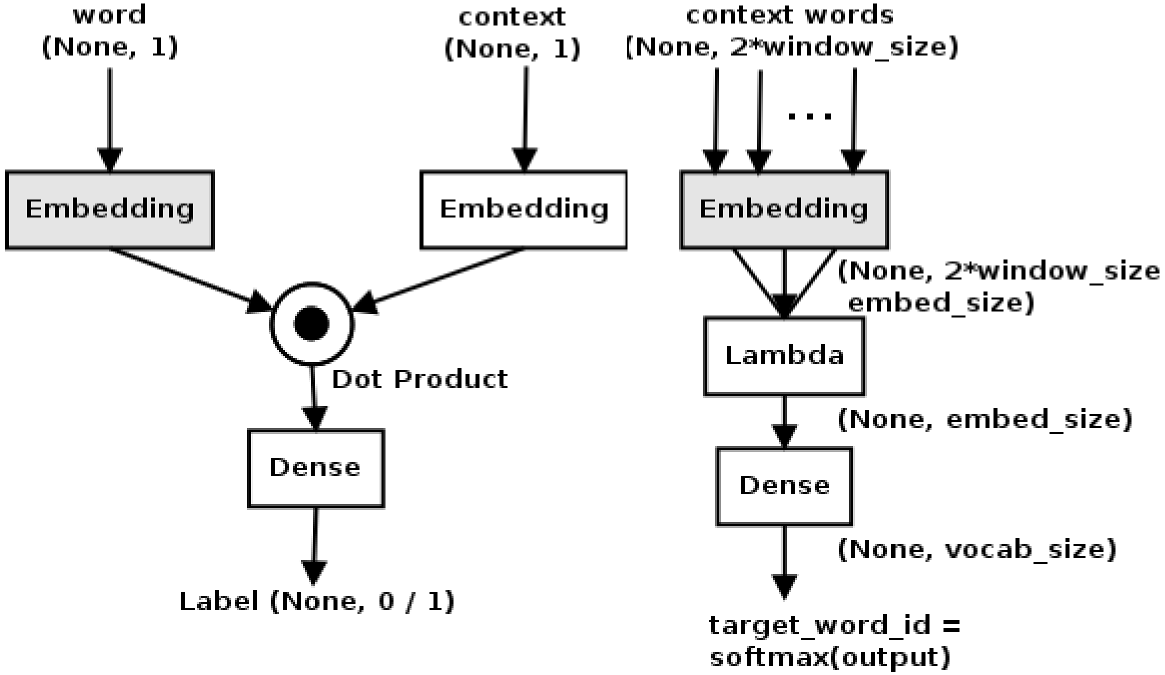
https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

Glove vs Word2vec



- Glove :
  - Count-based
  - Capture global co-occurrences statistics
  - Requires upfront pass through entire dataset.

- Both are fundamentally similar:
  - Capture local co-occurrence statistics (neighbors)
  - Capture distance between embedding vector (analogies)



Glove vs Word2vec

- GloVe generally shows higher accuracy than word2vec.
- GloVe is faster to train if use parallelization
- Python tooling for GloVe is not as mature as for word2vec.

  - The only tool available to do this as of the time of writing is the GloVe-Python project (https://github.com/maciejkula/glove-python), which provides a toy implementation for GloVe on Python

Fine-tune learned embedding (word2vec)

```
INPUT_FILE = "../data/umich-sentiment-train.txt"
WORD2VEC_MODEL = "../data/GoogleNews-vectors-negative300.bin.gz"
VOCAB_SIZE = 5000
EMBED_SIZE = 300
NUM_FILTERS = 256
NUM_WORDS = 3
BATCH_SIZE = 64
NUM_EPOCHS = 10
```

New Training Data

Learned Model

```
# load word2vec model
word2vec = Word2Vec.load_word2vec_format(WORD2VEC_MODEL, binary=True)
embedding_weights = np.zeros((vocab_sz, EMBED_SIZE))
for word, index in word2index.items():
    try:
        embedding_weights[index, :] = word2vec[word]
    except KeyError:
        pass
```

https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

Fine-tuned learned embedding (word2vec)

```
model = Sequential()
model.add(Embedding(vocab_sz, EMBED_SIZE, input_length=maxlen,
          weights=[embedding_weights]))
model.add(SpatialDropout1D(Dropout(0.2)))
model.add(Conv1D(filters=NUM_FILTERS, kernel_size=NUM_WORDS,
                        activation="relu"))
model.add(GlobalMaxPooling1D())
model.add(Dense(2, activation="softmax"))
```

```
model.compile(optimizer="adam", loss="categorical_crossentropy",
              metrics=["accuracy"])
history = model.fit(Xtrain, Ytrain, batch_size=BATCH_SIZE,
                    epochs=NUM_EPOCHS,
                    validation_data=(Xtest, Ytest))

score = model.evaluate(Xtest, Ytest, verbose=1)
print("Test score: {:.3f}, accuracy: {:.3f}".format(score[0], score[1]))
```

https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

Fine-tuned learned embedding (word2vec)

```
((4960, 42), (2126, 42), (4960, 2), (2126, 2))
 Train on 4960 samples, validate on 2126 samples
 Epoch 1/10
 4960/4960 [==============================] - 7s - loss: 0.1766 - acc: 0.9369 - val_loss: 0.0397 - val_acc: 0.9854
 Epoch 2/10
 4960/4960 [==============================] - 7s - loss: 0.0725 - acc: 0.9706 - val_loss: 0.0346 - val_acc: 0.9887
 Epoch 3/10
 4960/4960 [==============================] - 7s - loss: 0.0553 - acc: 0.9784 - val_loss: 0.0210 - val_acc: 0.9915
 Epoch 4/10
 4960/4960 [==============================] - 7s - loss: 0.0519 - acc: 0.9790 - val_loss: 0.0241 - val_acc: 0.9934
 Epoch 5/10
 4960/4960 [==============================] - 7s - loss: 0.0576 - acc: 0.9746 - val_loss: 0.0219 - val_acc: 0.9929
 Epoch 6/10
 4960/4960 [==============================] - 7s - loss: 0.0515 - acc: 0.9764 - val_loss: 0.0185 - val_acc: 0.9929
 Epoch 7/10
 4960/4960 [==============================] - 7s - loss: 0.0528 - acc: 0.9790 - val_loss: 0.0204 - val_acc: 0.9920
 Epoch 8/10
 4960/4960 [==============================] - 7s - loss: 0.0373 - acc: 0.9849 - val_loss: 0.0221 - val_acc: 0.9934
 Epoch 9/10
 4960/4960 [==============================] - 7s - loss: 0.0360 - acc: 0.9845 - val_loss: 0.0194 - val_acc: 0.9929
 Epoch 10/10
 4960/4960 [==============================] - 7s - loss: 0.0389 - acc: 0.9853 - val_loss: 0.0254 - val_acc: 0.9915
 2126/2126 [==============================] - 1s
 Test score: 0.025, accuracy: 0.993
```

https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

Fine-tune learned embedding (GloVe)

```
GLOVE_MODEL = "../data/glove.6B.300d.txt"
word2emb = {}
fglove = open(GLOVE_MODEL, "rb")
for line in fglove:
    cols = line.strip().split()
    word = cols[0]
    embedding = np.array(cols[1:], dtype="float32")
    word2emb[word] = embedding
fglove.close()
```

Learned Model

```
embedding_weights = np.zeros((vocab_sz, EMBED_SIZE))
for word, index in word2index.items():
    try:
        embedding_weights[index, :] = word2emb[word]
    except KeyError:
        pass
```

https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

Fine-tune learned embedding (GloVe)

```
((4960, 42), (2126, 42), (4960, 2), (2126, 2))
Train on 4960 samples, validate on 2126 samples
Epoch 1/10
4960/4960 [==============================] - 7s - loss: 0.1748 - acc: 0.9240 - val_loss: 0.0390 - val_acc: 0.9840
Epoch 2/10
4960/4960 [==============================] - 7s - loss: 0.0859 - acc: 0.9649 - val_loss: 0.0431 - val_acc: 0.9845
Epoch 3/10
4960/4960 [==============================] - 7s - loss: 0.0586 - acc: 0.9754 - val_loss: 0.0528 - val_acc: 0.9779
Epoch 4/10
4960/4960 [==============================] - 8s - loss: 0.0565 - acc: 0.9798 - val_loss: 0.0386 - val_acc: 0.9873
Epoch 5/10
4960/4960 [==============================] - 8s - loss: 0.0792 - acc: 0.9683 - val_loss: 0.0233 - val_acc: 0.9892
Epoch 6/10
4960/4960 [==============================] - 8s - loss: 0.0618 - acc: 0.9746 - val_loss: 0.0247 - val_acc: 0.9911
Epoch 7/10
4960/4960 [==============================] - 7s - loss: 0.0569 - acc: 0.9752 - val_loss: 0.0266 - val_acc: 0.9906
Epoch 8/10
4960/4960 [==============================] - 8s - loss: 0.0419 - acc: 0.9829 - val_loss: 0.0211 - val_acc: 0.9920
Epoch 9/10
4960/4960 [==============================] - 7s - loss: 0.0371 - acc: 0.9849 - val_loss: 0.0206 - val_acc: 0.9920
Epoch 10/10
4960/4960 [==============================] - 9s - loss: 0.0422 - acc: 0.9815 - val_loss: 0.0266 - val_acc: 0.9906
2126/2126 [==============================] - 1s
Test score: 0.027, accuracy: 0.991
```

https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras

Look up embeddings

```
INPUT_FILE = "../data/umich-sentiment-train.txt"
GLOVE_MODEL = "../data/glove.6B.100d.txt"
VOCAB_SIZE = 5000
EMBED_SIZE = 100
BATCH_SIZE = 64
NUM_EPOCHS = 10
```

```
model.add(Embedding(vocab_sz, EMBED_SIZE, input_length=maxlen,
                    weights=[embedding_weights],
                    trainable=False))
model.add(SpatialDropout1D(Dropout(0.2)))
```

Set this!

Using third-party Implementations (Gensim)

- Gensim library provides an implementation of word2vec.
- Keras does not provide any support for word2vec.
- Integrating the genism implementation into Keras is common practice.

Using third-party Implementations (Gensim)

```
from gensim.models import KeyedVectors
import logging
import os
```

```
class Text8Sentences(object):
  def __init__(self, fname, maxlen):
    self.fname = fname
    self.maxlen = maxlen

  def __iter__(self):
    with open(os.path.join(DATA_DIR, "text8"), "rb") as ftext:
      text = ftext.read().split(" ")
      sentences, words = [], []
      for word in text:
        if len(words) >= self.maxlen:
          yield words
          words = []
          words.append(word)
          yield words
```

```
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

DATA_DIR = "../data/"
sentences = Text8Sentences(os.path.join(DATA_DIR, "text8"), 50)
model = word2vec.Word2Vec(sentences, size=300, min_count=30)
```

Using third-party Implementations (Gensim)

```
>>> model.vocab.keys()[0:10]
['homomorphism',
'woods',
'spiders',
'hanging',
'woody',
'localized',
'sprague',
'originality',
'alphabetic',
'hermann']
```

```
>>> model.most_similar("woman")
 [('child', 0.7057571411132812),
 ('girl', 0.702182412147522),
 ('man', 0.6846336126327515),
 ('herself', 0.6292711496353149),
 ('lady', 0.6229539513587952),
 ('person', 0.6190367937088013),
 ('lover', 0.6062309741973877),
 ('baby', 0.5993420481681824),
 ('mother', 0.5954475402832031),
 ('daughter', 0.5871444940567017)]
```

Using third-party Implementations (Gensim)

```
>>> model.most_similar(positive=['woman', 'king'], negative=['man'], topn=10)
 [('queen', 0.6237582564353943),
 ('prince', 0.5638638734817505),
 ('elizabeth', 0.5557916164398193),
 ('princess', 0.5456407070159912),
 ('throne', 0.5439794063568115),
 ('daughter', 0.5364126563072205),
 ('empress', 0.5354889631271362),
 ('isabella', 0.5233952403068542),
 ('regent', 0.520746111869812),
 ('matilda', 0.5167444944381714)]
```

```
>>> model.similarity("girl", "woman")
 0.702182479574
 >>> model.similarity("girl", "man")
 0.574259909834
 >>> model.similarity("girl", "car")
 0.289332921793
 >>> model.similarity("bus", "car")
 0.483853497748
```

Neural-based Predictive Models

- Goal: Predict Text using Learned Embedding Vectors
- Word2vec:
  - Shallow neural network
  - Local: nearby words predict each other
  - Fixed word embedding vector size (i.e., 300)
  - Optimizer: Mini-batch SGD
- SyntaxNet:
  - Deep(er) neural network
  - More global
  - Not an RNN!
  - Can Combine with BOW-based models (i.e., word2vec CBOW)

Word2vec Library

- Gensim:
  - Python only
  - Most popular

- Spark ML
  - Python + Jawa/Scala
  - Supports only synonyms

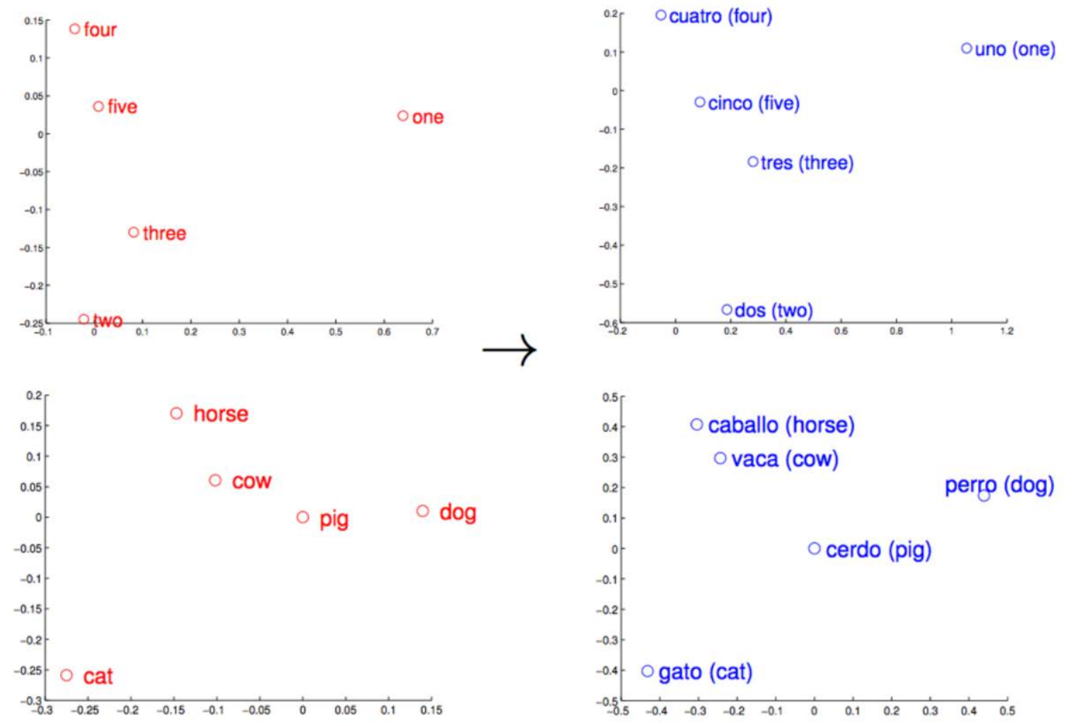
*2vec

- lda2vec:
  - LDA (Global) + word2vec (local)

- Like2vec:
  - Embedding-based Recommender

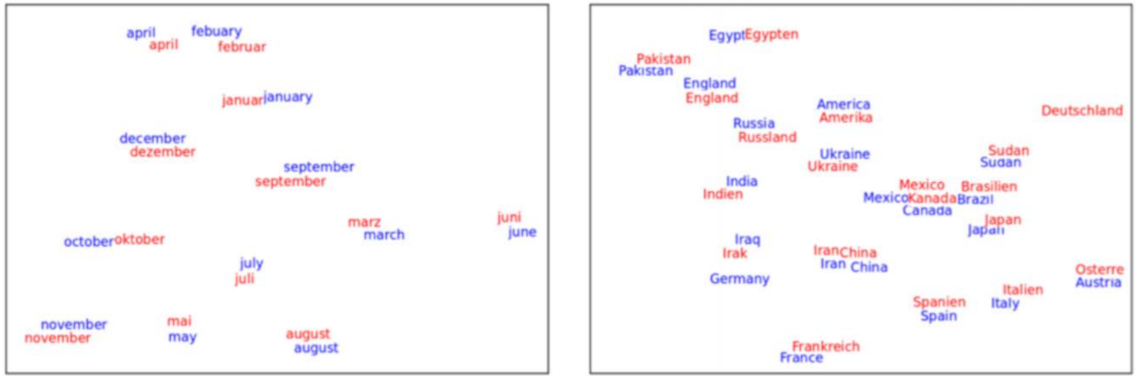
Word Embeddings Applications (Machine Translation)

Word Embeddings for MT: Mikolov (2013)



Mikolov, T., Le, V. L., Sutskever, I. (2013) .
Exploiting Similarities among Languages for Machine Translation

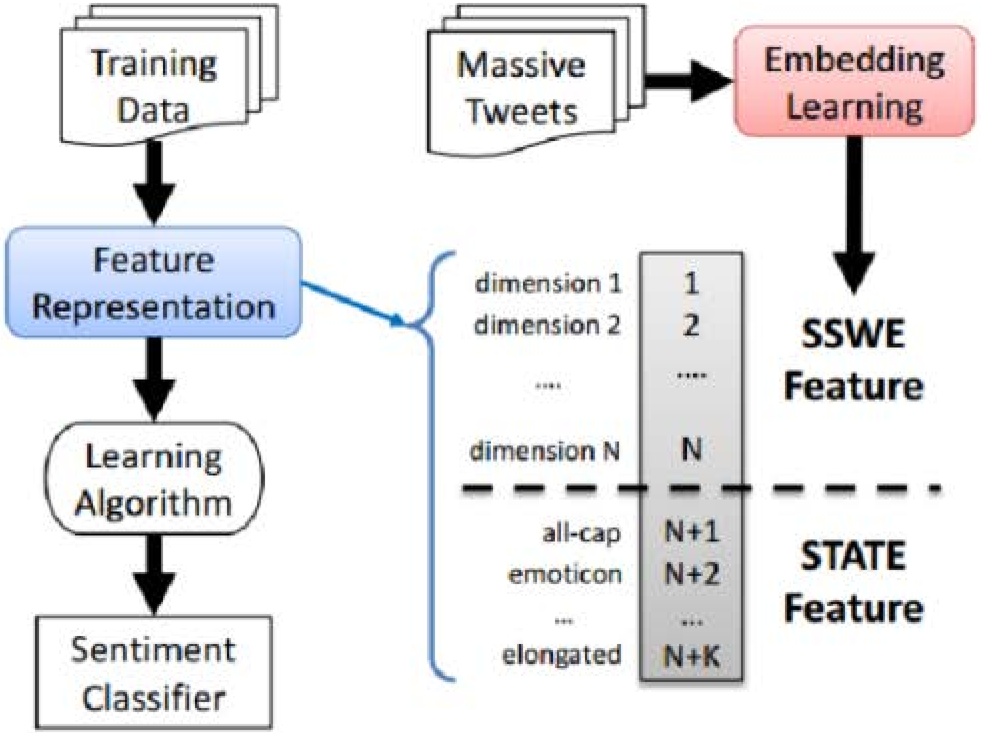
Word Embeddings for MT: Kiros (2014)

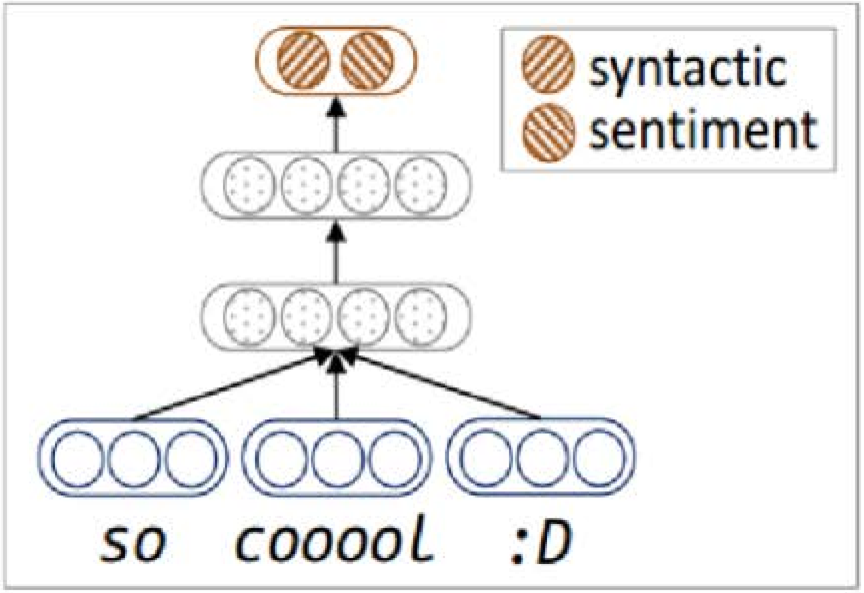


(a) Months (b) Countries

Kiros, R., Zemel, R. S., Salakhutdinov, R. (2014) .
A Multiplicative Model for Learning Distributed Text-Based Attribute Representations

Word Embeddings Applications (Sentiment Analysis)





Thank you!
Q/A Sessions

All source codes and datasets are available!
The DLwK sources are fixed and modified to run
on Python 3.5 and Keras 2.2 in Windows 10 with GPU

Please ask Panitia INACL 2017