Embeddings Recurrent Neural Networks, and Sequences (Part 1)

May 11, 2021

ddebarr@uw.edu

http://cross-entropy.net/ML530/Deep_Learning_4.pdf

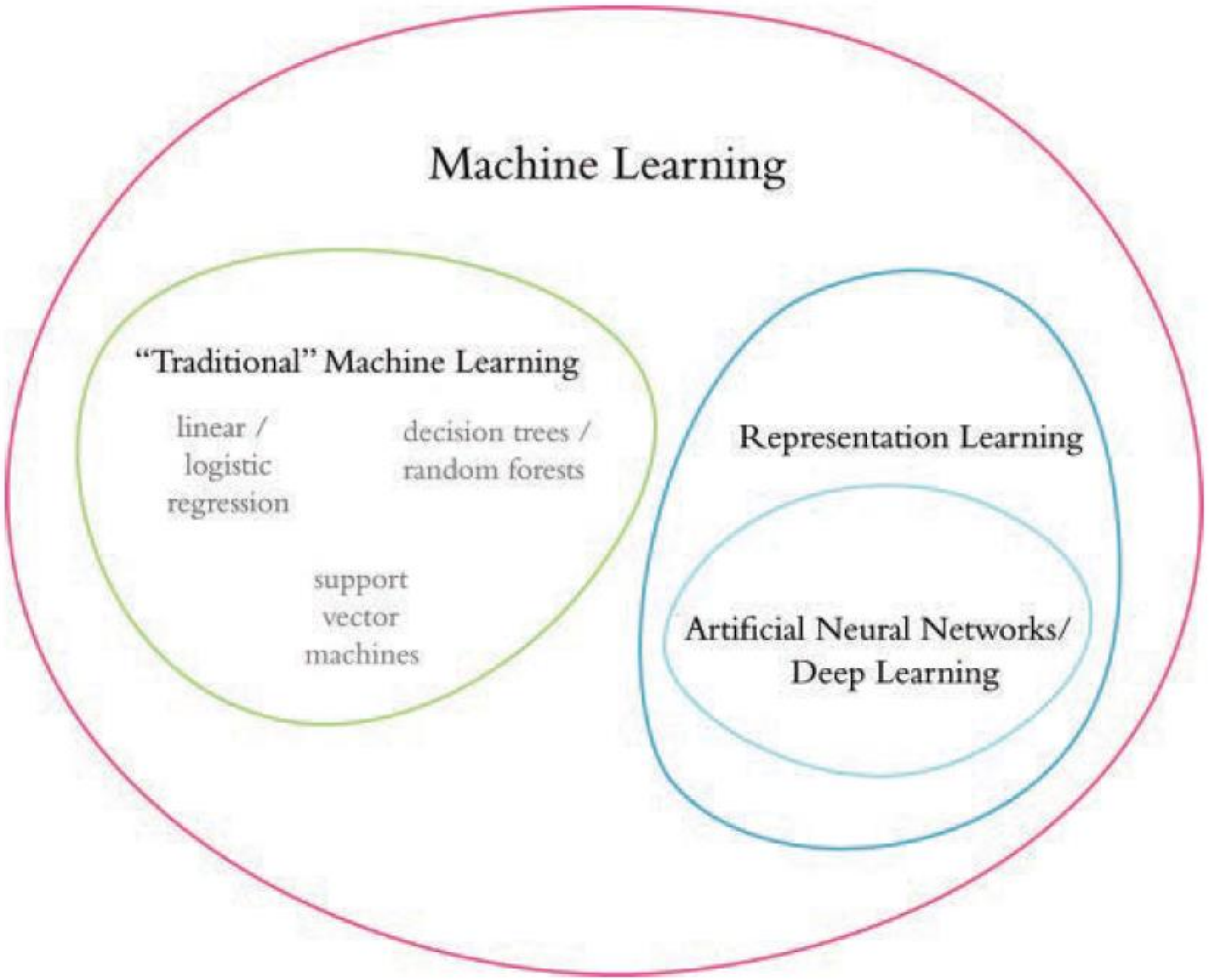
Agenda

- Homework Review
- [DLI] Human and Machine Language
- [DLI] Natural Language Processing

[DLI] Human and Machine Language

- Deep Learning for Natural Language Processing
- Computational Representations of Language
- Elements of Natural Human Language
- Google Duplex
- Summary

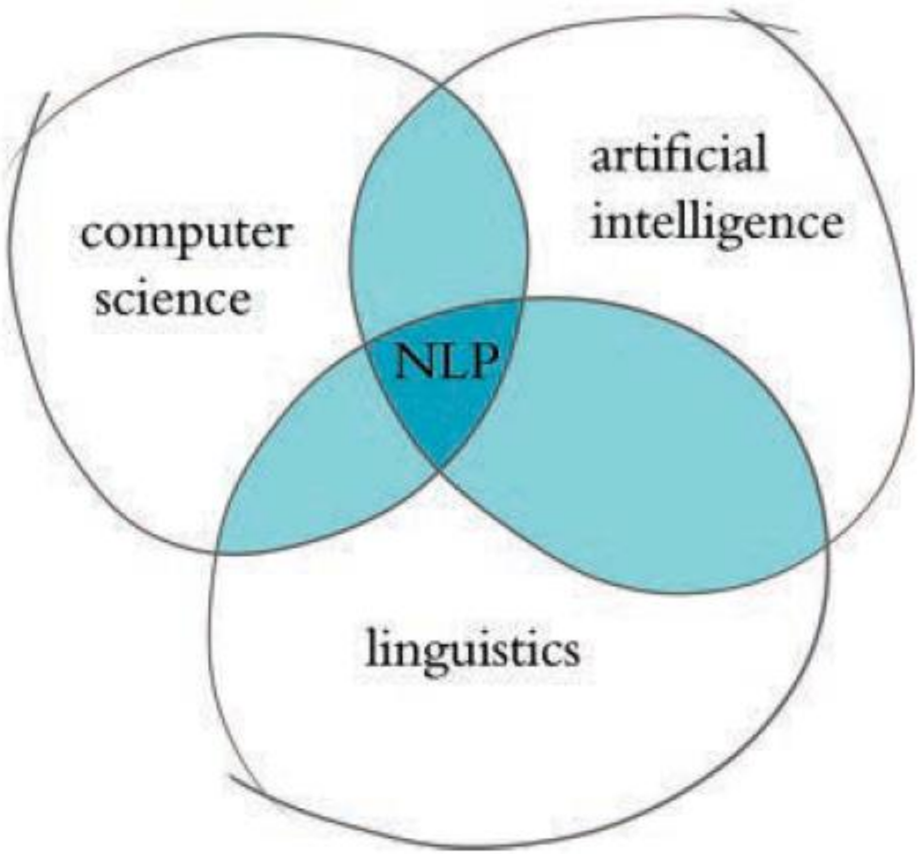
Traditional versus Representation Learning



Natural Language Processing (NLP)

Sits at the intersection of …

- computer science
- artificial intelligence
- linguistics

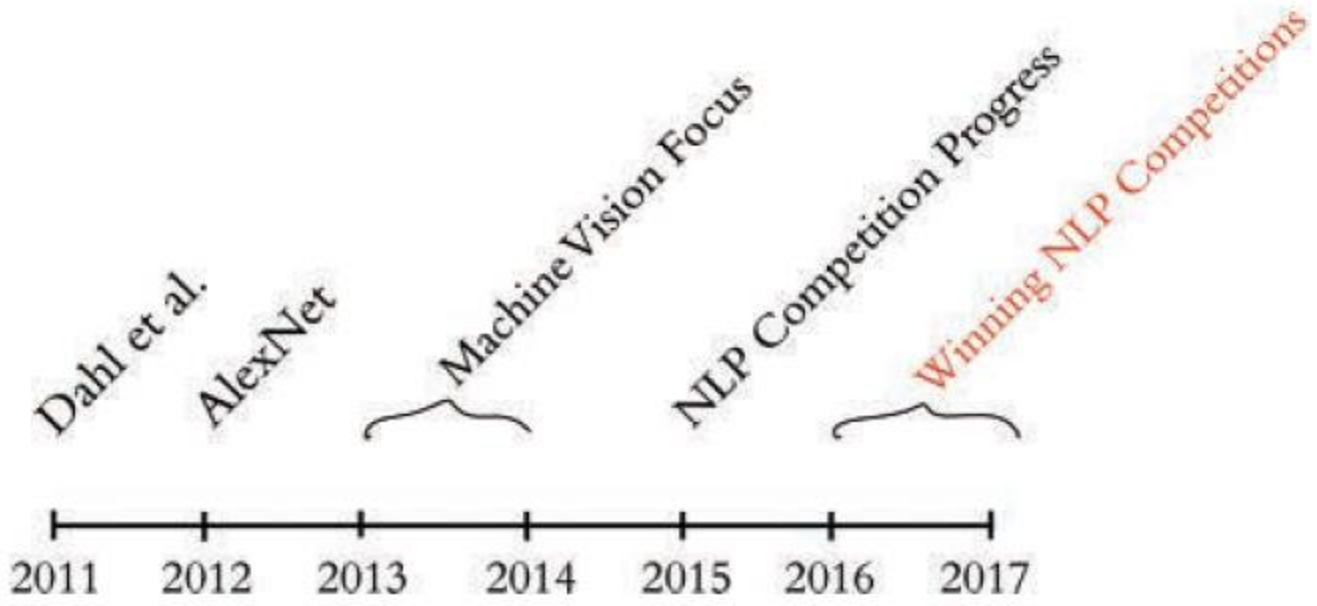


Examples of NLP in Industry

- **Classifying documents:** using the language within a document (e.g., an email, a Tweet, or a review of a film) to classify it into a particular category (e.g., high urgency, positive sentiment, or predicted direction of the price of a company's stock)
- **Machine translation:** assisting language-translation firms with machine-generated suggestions from a source language (e.g., English) to a target language (e.g., German or Mandarin); increasingly, fully automatic—though not always perfect—translations between languages
- **Search engines:** autocompleting users' searches and predicting what information or website they're seeking
- **Speech recognition:** interpreting voice commands to provide information or take action, as with virtual assistants like Amazon's Alexa, Apple's Siri, or Microsoft's Cortana
- **Chatbots:** carrying out a natural conversation for an extended period of time; though this is seldom done convincingly today, they are nevertheless helpful for relatively linear conversations on narrow topics such as the routine components of a firm's customer-service phone calls

Milestones Involving NLP

Before AlexNet, George Dahl and others from Microsoft Research trained a deep neural network to recognize a substantial vocabulary of words from audio recordings of human speech



One-Hot Encoding of Words

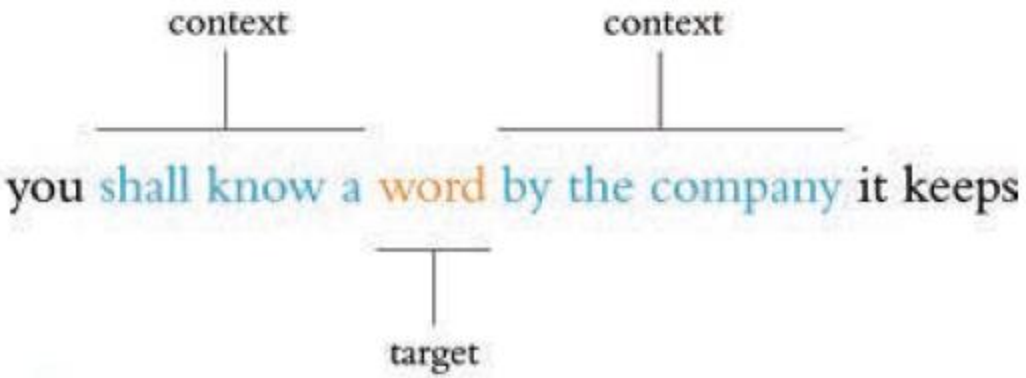
The bat sat on the cat.

words						
the	1	0	0	0	1	0
bat	0	1	0	0	0	0
on	0	0	0	1	0	0
⋮						
n_{unique_words}						

This is a sequence representation, instead of a Bag of Words (BoW)

You Shall Know a Word by the Company It Keeps

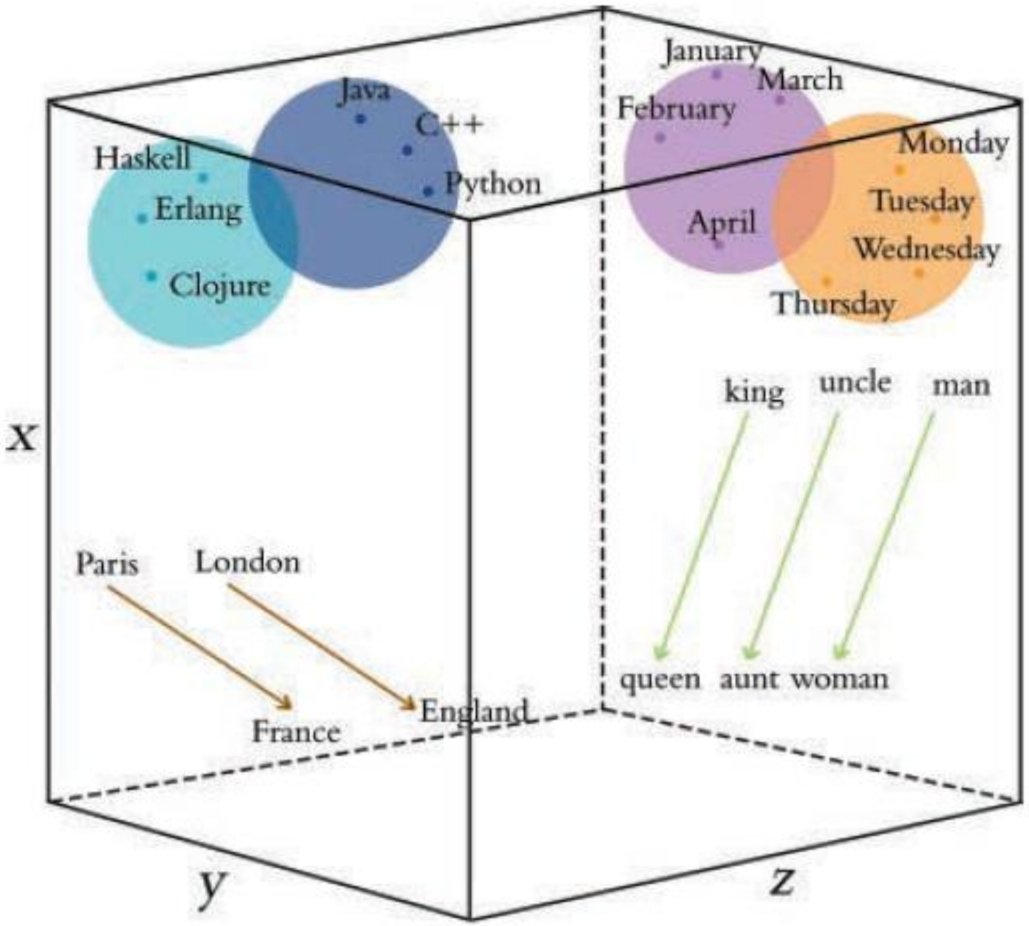
- Originally posed by Ludwig Wittgenstein in 1953: “The meaning of a word is its use in language”
- Captured succinctly by John Rupert Firth in 1957 …



- Word2Vec and Global Vectors (GloVe) exploit context to learn vector representations for words

Example 3-Dimensional Vector Space

- Functional versus imperative programming languages
- Months versus days of the week
- Capital cities versus country
- Feminine versus masculine roles

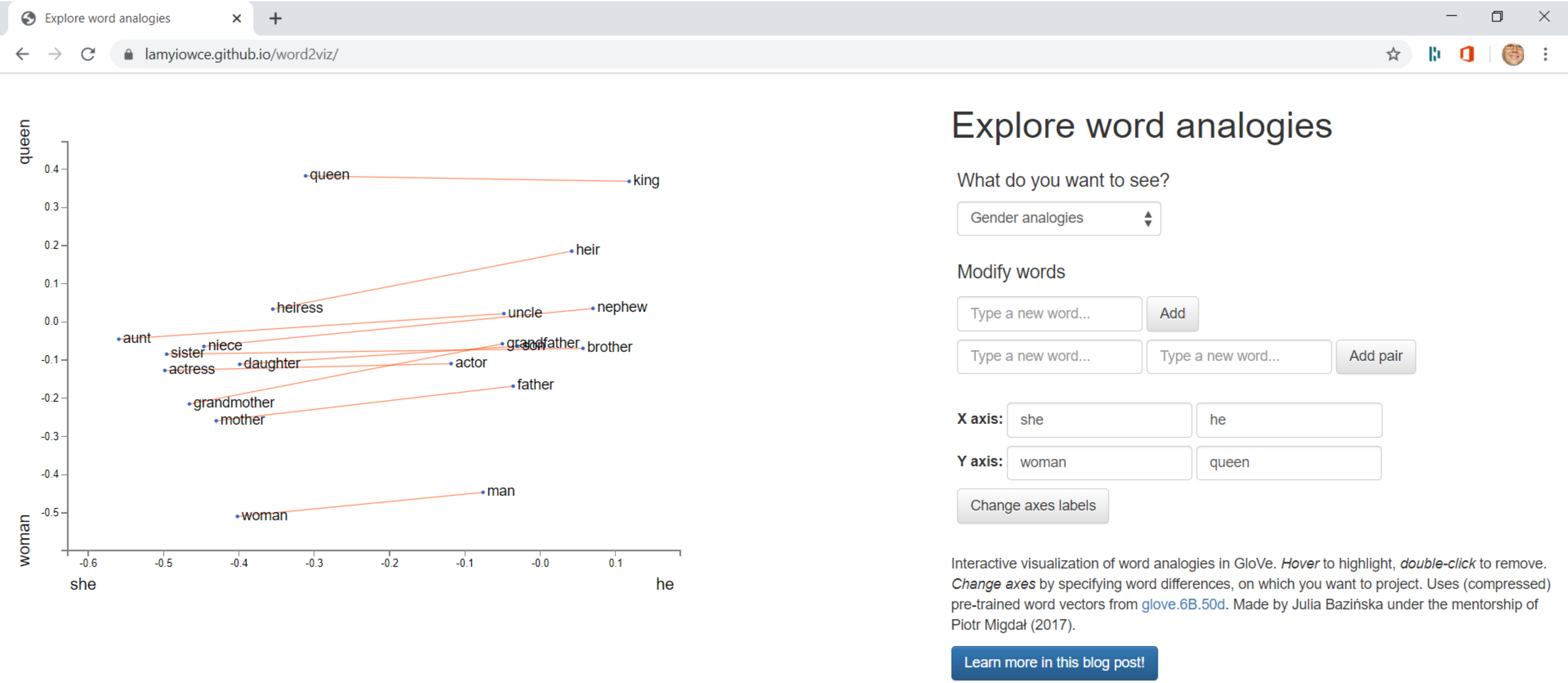


Word Vector Arithmetic

$$V_{king} - V_{man} + V_{woman} = V_{queen}$$
$$V_{bezos} - V_{amazon} + V_{tesla} = V_{musk}$$
$$V_{windows} - V_{microsoft} + V_{google} = V_{android}$$

$$x_{queen} = x_{king} - x_{man} + x_{woman} = -0.9 + 1.1 - 3.2 = -3.0$$
$$y_{queen} = y_{king} - y_{man} + y_{woman} = 1.9 - 2.4 + 2.5 = 2.0$$
$$z_{queen} = z_{king} - z_{man} + z_{woman} = 2.2 - 3.0 + 2.6 = 1.8$$

Word2Viz



What do you want to see?

Gender analogies

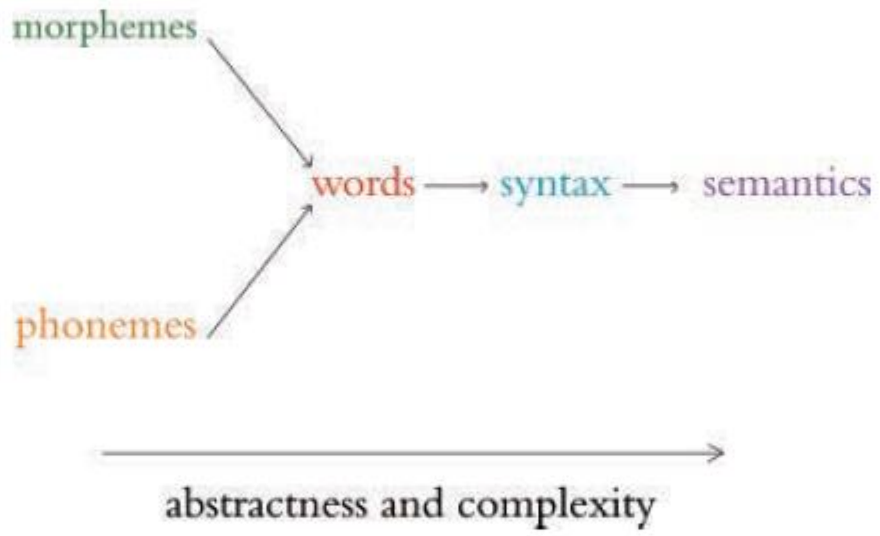
- Gender analogies
- Adjectives analogies
- Numbers say-write analogies
- Up-down linguistic metaphors
- Verb tenses
- Jobs
- God: similar words
- Intelligence
- Nostalgia
- Empty

https://lamyiowce.github.io/word2viz/

One-Hot versus Vector-Based Representation

One-Hot	Vector-Based
Not subtle	Very nuanced
Manual taxonomies	Automatic
Handles new words poorly	Seamlessly incorporates new words
Subjective	Driven by natural language data
Word similarity not represented	Word similarity = proximity in space

Elements of Natural Human Language



- Morphemes are the smallest units of language that contain some meaning; e.g. the three morphemes out, go, and ing combine to form the word outgoing
- Phonemes are the sounds that make up spoken words
- Syntax is the arrangement of words into phrases and phrases into sentences in order to convey meaning in a way that is consistent across the users of a given language

Google Duplex

- The Google Duplex technology was unveiled at the company's I/O developers conference in May 2018. The search giant's CEO, Sundar Pichai, held spectators in rapture as he demonstrated Google Assistant making a phone call to a Chinesefood restaurant to book a reservation.
  https://www.youtube.com/watch?v=D5VN56jQMWM

- Duplex uses a combination of de novo [new] waveform synthesis using Tacotron and WaveNet, as well as a more classical "concatenative" text-to-speech engine
- Tacotron maps sequences of words to corresponding sequences of audio features, which capture subtleties of human speech such as pitch, speed, intonation, and even pronunciation
- These features are then fed into WaveNet, which synthesizes the actual waveform that the restaurateur hears
- This whole system is able to produce a natural-sounding voice with the correct cadence, emotion, and emphasis
- During more-or-less rote moments in the conversation, the simple concatenative TTS engine (composed of recordings of its own "voice"), which is less computationally demanding to execute, is used. The entire model dynamically switches between the various models as needed.

Reminder: today's chatbots are seldom capable of successfully carrying out a natural conversation for an extended period of time

Summary

In this chapter, you learned about applications of deep learning to the processing of natural language. To that end, we described further the capacity for deep learning models to automatically extract the most pertinent features from data, removing the need for labor-intensive one-hot representations of language. Instead, NLP applications involving deep learning make use of vector-space embeddings, which capture the meaning of words in a nuanced manner that improves both model performance and accuracy.

In Chapter 11, you'll construct an NLP application by making use of artificial neural networks that handle the input of natural language data all the way through to the output of an inference about those data. In such "end-to-end" deep learning models, the initial layers create word vectors that flow seamlessly into deeper, specialized layers of artificial neurons, including layers that incorporate "memory." These model architectures highlight both the strength and the ease of use of deep learning with word vectors.

[DLI] Natural Language Processing (NLP)

- Preprocessing Natural Language Data
- Creating Word Embeddings with word2vec
- The Area Under the ROC Curve
- Natural Language Classification with Familiar Networks
- Networks Designed for Sequential Data
- Non-Sequential Architectures: the Keras Functional API
- Summary

Common Natural Language Preprocessing Options

- Tokenization: splitting a document into a list of tokens (words or characters)
- Converting characters to lowercase
- Removing stop words
- Removing punctuation
- Stemming
- Creating n-grams; e.g. treating “New York City” (a tri-gram) as a single token

Suggestions

- Will this preprocessing option help? Maybe; maybe not.
- Maybe useful for small corpora; maybe not as useful for larger corpora
  - Stemming
  - Coverting characters to lowercase
- Maybe depends on task
  - Removing punctuation (including question marks) may harm a question-answering algorithm
  - Removing stop words (e.g. “not”) may harm a sentiment classification algorithm
    - This movie is the bomb
    - This movie is a bomb

Preprocessing Dependencies

```
import nltk
from nltk import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import *
nltk.download('gutenberg')
nltk.download('punkt')
nltk.download('stopwords')
```

```
import string
import genism    # pip install gensim
from gensim.models.phrases import Phraser, Phrases
from gensim.models.word2vec import Word2Vec
from sklearn.manifold import TSNE
import pandas as pd
from bokeh.io import output_notebook, output_file
from bokeh.plotting import show, figure
%matplotlib inline
```

Tokenization

```
from nltk.corpus import gutenberg
# len(gutenberg.fileids()) == 18 books
# len(gutenberg.raw()) == 11,793,318 characters
gberg_sent_tokens = sent_tokenize(gutenberg.raw())
# len(gberg_sent_tokens) == 94,428 sentences
# len(gberg_sent_tokens[91087]) == 6,486 … hmmm
# dozens of lines from Leaves of Grass (commas)
word_tokenize(gberg_sent_tokens[1])
word_tokenize(gberg_sent_tokens[1])[14]
gberg_sents = gutenberg.sents()    # len(gberg_sents) == 98,552
```

['She', 'was', 'the', 'youngest', 'of', 'the', 'two', 'daughters', 'of', 'a', 'most', 'affectionate', ',', 'indulgent', 'father', ';', 'and', 'had', ',', 'in', 'consequence', 'of', 'her', 'sister', "'s", 'marriage', ',', 'been', 'mistress', 'of', 'his', 'house', 'from', 'a', 'very', 'early', 'period', '.']

Converting Characters to Lowercase

list comprehension example (creating a list from a list)

[w.lower() for w in gberg_sents[4]]

```
>>> gberg_sents[4]
['She', 'was', 'the', 'youngest', 'of', 'the', 'two', 'daughters', 'of', 'a', 'most', 'affectionate', ',', 'indulgent', 'father',
';', 'and', 'had', ',', 'in', 'consequence', 'of', 'her', 'sister', "'", 's', 'marriage', ',', 'been', 'mistress', 'of', 'his',
'house', 'from', 'a', 'very', 'early', 'period', '.’]

>>> [ w.lower() for w in gberg_sents[4] ]
['she', 'was', 'the', 'youngest', 'of', 'the', 'two', 'daughters', 'of', 'a', 'most', 'affectionate', ',', 'indulgent', 'father',
';', 'and', 'had', ',', 'in', 'consequence', 'of', 'her', 'sister', "'", 's', 'marriage', ',', 'been', 'mistress', 'of', 'his',
'house', 'from', 'a', 'very', 'early', 'period', '.']
```

Removing Stop Words and Punctuation

```
stpwrds = stopwords.words('english') + list(string.punctuation)
[w.lower() for w in gberg_sents[4] if w.lower() not in stpwrds]
```

```
>>> [ w.lower() for w in gberg_sents[4] ]
['she', 'was', 'the', 'youngest', 'of', 'the', 'two', 'daughters', 'of', 'a', 'most', 'affectionate', ',', 'indulgent', 'father',
';', 'and', 'had', ',', 'in', 'consequence', 'of', 'her', 'sister', "'", 's', 'marriage', ',', 'been', 'mistress', 'of', 'his',
'house', 'from', 'a', 'very', 'early', 'period', '.’]

>>> [w.lower() for w in gberg_sents[4] if w.lower() not in stpwrds]
['youngest', 'two', 'daughters', 'affectionate', 'indulgent', 'father', 'consequence', 'sister', 'marriage', 'mistress',
'house', 'early', 'period']
```

Stemming

stemmer = PorterStemmer()

[stemmer.stem(w.lower()) for w in gberg_sents[4] if w.lower() not in stpwrds]

```
>>> [w.lower() for w in gberg_sents[4] if w.lower() not in stpwrds]
['youngest', 'two', 'daughters', 'affectionate', 'indulgent', 'father', 'consequence', 'sister', 'marriage', 'mistress',
'house', 'early', 'period’]

>>> stemmer = PorterStemmer()
>>> [stemmer.stem(w.lower()) for w in gberg_sents[4] if w.lower() not in stpwrds]
['youngest', 'two', 'daughter', 'affection', 'indulg', 'father', 'consequ', 'sister', 'marriag', 'mistress', 'hous', 'earli',
'period']
```

daughters -> daughter
house -> hous [housing]
early -> earli [earlier; earliest]

https://tartarus.org/martin/PorterStemmer/

GenSim bi-grams

```
>>> from gensim.models.phrases import Phraser, Phrases
>>> input = [ [ 'one', 'two', 'three', 'one', 'two' ], [ 'one', 'two' ] ]
>>> phrases = Phrases(input, min_count = 1, threshold = 1)
>>> bigrams = Phraser(phrases)
>>> bigrams.phrasegrams
{(b'one', b'two'): 1.3333333333333333}
>>> phrases.vocab
defaultdict(<class 'int'>, {b'one': 3, b'two': 3, b'one_two': 3, b'three': 1, b'two_three': 1,
b'three_one': 1})
>>> ((phrases.vocab["one_two".encode()] - phrases.min_count) * len(phrases.vocab)) /
(phrases.vocab["one".encode()] * phrases.vocab["two".encode()])
1.3333333333333333
```

https://radimrehurek.com/gensim/models/phrases.html#gensim.models.phrases.original_scorer

score = ((bigram_count - min_count) * len_vocab) / (word_a_count * word_b_count)

See equation 6 of https://arxiv.org/abs/1310.4546

Handling n-grams

```
>>> phrases = Phrases(gberg_sents)
>>> bigram = Phraser(phrases)
>>> bigram.phrasegrams[("Mount".encode(), "Vesuvius".encode())]
12125.901960784313
>>> ((phrases.vocab["Mount_Vesuvius".encode()] - phrases.min_count) * len(phrases.vocab)) / (phrases.vocab["Mount".encode()] * phrases.vocab["Vesuvius".encode()])
12125.901960784313

>>> tokenized_sentence = "Jon lives in New York City".split()
>>> tokenized_sentence
['Jon', 'lives', 'in', 'New', 'York', 'City']
>>> bigram[tokenized_sentence]
['Jon', 'lives', 'in', 'New_York', 'City']
```

repeat to generate trigrams

Preprocessing the Full Corpus

```
lower_sents = []
for s in gberg_sents:
    lower_sents.append([w.lower() for w in s if w.lower()
                        not in list(string.punctuation)])
lower_bigram = Phraser(Phrases(lower_sents,
                       min_count=32, threshold=64))
clean_sents = []
for s in lower_sents:
    clean_sents.append(lower_bigram[s])
```

```
>>> clean_sents[6]
['sixteen', 'years', 'had', 'miss_taylor', 'been', 'in', 'mr_woodhouse', 's', 'family', 'less', 'as', 'a', 'governess', 'than', 'a', 'friend', 'very', 'fond', 'of', 'both', 'daughters', 'but', 'particularly', 'of', 'emma']
```

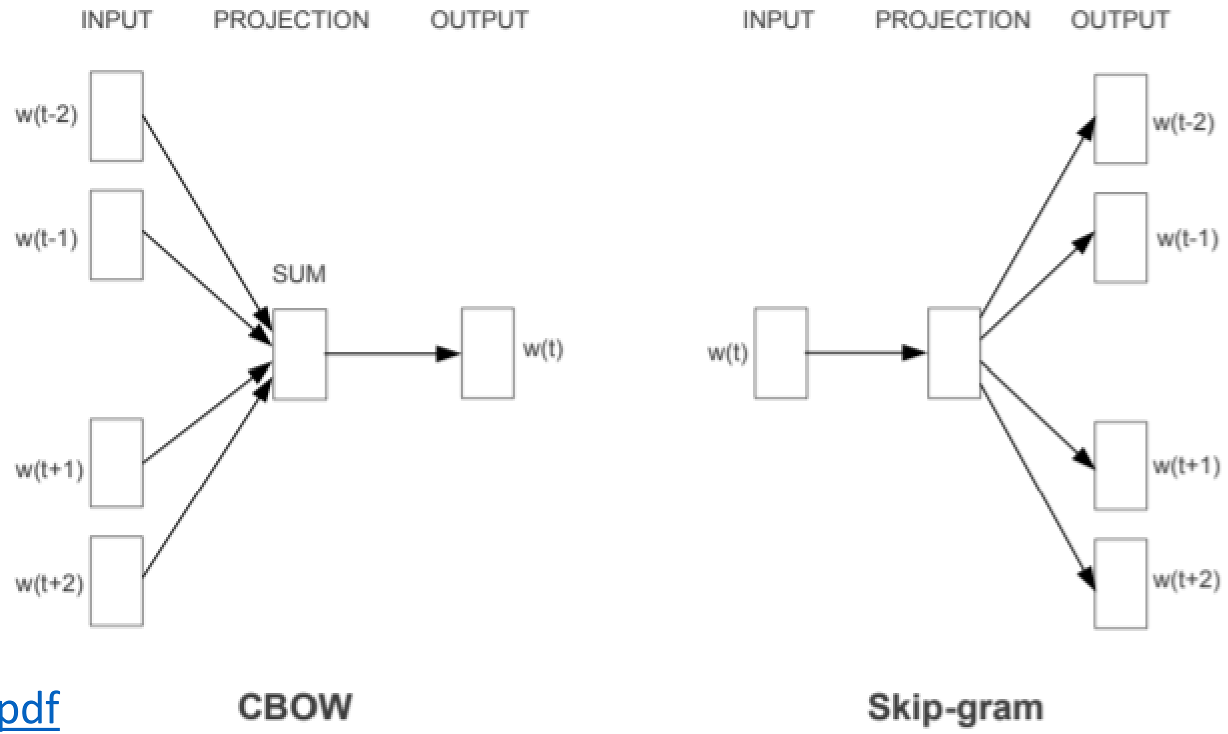
Word2Vec: Continuous Bag Of Words (CBOW) Architecture

`you shall know a word by the company it keeps`

- We take all the context words within the windows to the right and the left of the target word.
- We (figuratively!) throw all of these context words into a bag. If it helps you remember that the sequence of words is irrelevant, you can even imagine shaking up the bag.
- We calculate the average of all the context words contained in the bag, using this average to estimate what the target word could be.

Word2Vec

Objective is to train an embedding matrix, where the number of rows is the size of the vocabulary and the number of columns is the number of dimensions used to represent words in the vocabulary



https://arxiv.org/pdf/1301.3781.pdf

Comparison of Word2Vec Architectures

Architecture	Predicts	Relative Strengths
Skip-gram (SG)	Context words given target word	Better for a smaller corpus; represents rare words well
CBOW	Target word given context words	Multiple times faster; represents frequent words slightly better

Efficient Word2Vec Training: Hierarchical SoftMax versus Negative Sampling

Size of the vocabulary drives the runtime complexity, so we'd like to use something more efficient than a standard softmax layer

- Hierarchical Softmax
  - Tree of binary classification tasks used to reduce the number of nodes to be updated: $O(\log_2(k))$ instead of $O(k)$, where k is the size of the vocabulary
- Negative Sampling
  - Random sample of negatives used for training

https://arxiv.org/abs/1411.2738

Running Word2Vec

```
model = Word2Vec(sentences=clean_sents, size=64,
                 sg=1, window=10, iter=5,
                 min_count=10, workers=4)
# size = 64: 64 floats to represent each word in the vocabulary
# sg = 1: selects skip-gram architecture
# negative sampling used by default
# window = 10: 20 words for context
# iter = 5: sliding window passes over the corpus 5 times
# min_count = 10: word must occur 10 times to be considered
# workers: number of CPU cores to use
```

Reviewing Cosine Similarity Measures

```
>>> model.save('clean_gutenberg_model.w2v')
>>> model = gensim.models.Word2Vec.load('clean_gutenberg_model.w2v')
>>> model.wv.most_similar('father', topn=3)
[('mother', 0.8266161680221558), ('brother', 0.7343044281005859), ('daughter', 0.7092597484588623)]
>>> model.wv.most_similar(positive=['dog'], topn=3)
[('puppy', 0.7803493142127991), ('brahmin', 0.7515953183174133), ('camel', 0.7439272403717041)]
>>> model.wv.most_similar(positive=['eat'], topn=3)
[('bread', 0.833627462387085), ('drink', 0.8167247772216797), ('meat', 0.7876289486885071)]
>>> model.wv.most_similar(positive=['day'], topn=3)
[('morning', 0.7432693839073181), ('night', 0.7180658578872681), ('week', 0.7139558792114258)]
>>> model.wv.most_similar(positive=['ma_am'], topn=3)
[('betty', 0.8646817803382874), ('madam', 0.8590834736824036), ('m_sure', 0.8484251499176025)]
>>> model.wv.doesnt_match("mother father sister brother dog".split())
'dog'
>>> model.wv.similarity('father', 'dog')
0.48039153
>>> model.wv.most_similar(positive=['father', 'woman'], negative=['man'], topn=3)
[('mother', 0.811971127986908), ('daughter', 0.766667902469635), ('sister', 0.7548267841339111)]
>>> model.wv.most_similar(positive=['husband', 'woman'], negative=['man'], topn=3)
[('sister', 0.7030020952224731), ('wife', 0.6938996911048889), ('mother', 0.6863758563995361)]
```

cosine similarity: np.dot(model.wv["father"], model.wv["dog"]) / (np.linalg.norm(model.wv["father"]) * np.linalg.norm(model.wv["dog"]))

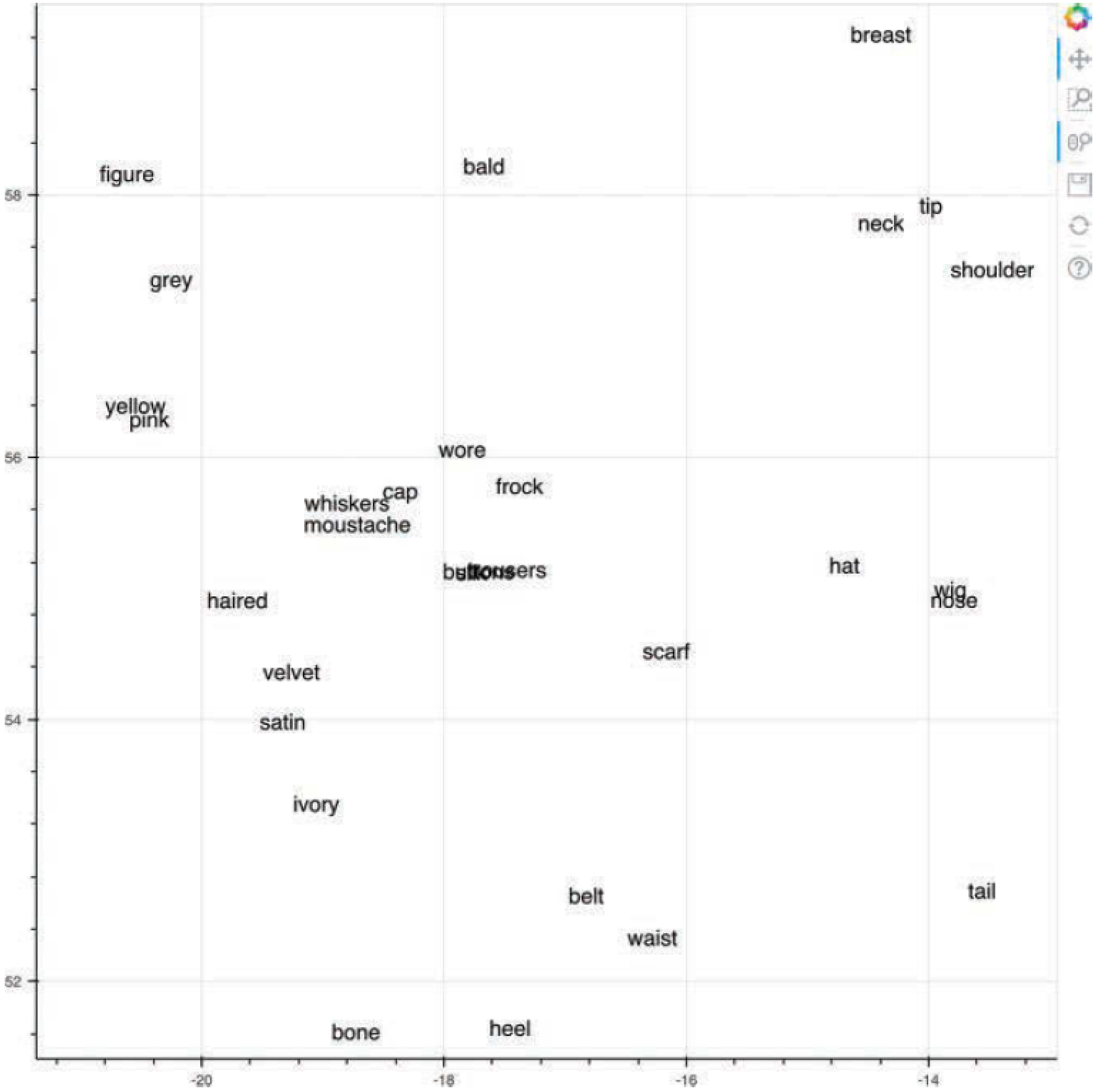
Plotting Word Vectors

```
tsne = TSNE(n_components=2, n_iter=1000)
X_2d = tsne.fit_transform(model.wv[model.wv.vocab])
coords_df = pd.DataFrame(X_2d, columns=['x','y'])
coords_df['token'] = model.wv.vocab.keys()
coords_df.head()
```

	x	y	token
0	62.494060	8.023034	emma
1	8.142986	33.342200	by
2	62.507140	10.078477	jane
3	12.477635	17.998343	volume
4	25.736960	30.876250	i

Bokeh Plot

```
subset_df = coords_df.sample(n=5000)
p = figure(plot_width=800, plot_height=800)
_ = p.text(x=subset_df.x, y=subset_df.y,
      text=subset_df.token)
show(p)
# “Clothing” words, revealed by zooming
# { cap, frock, scarf, hat, belt, trousers? }
```



Confusion Matrix

		actual y = 1	actual y = 0
predicted y	1	True positive	False positive
	0	False negative	True negative

True Positive (TP) Rate: TP / (TP + FN)

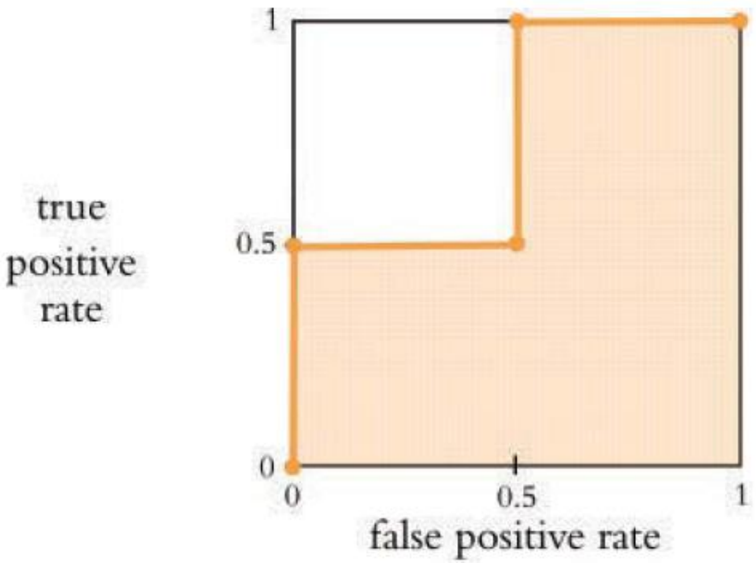
False Positive (FP) Rate: FP / (FP + TN)

As we reduce the classification threshold from 1 to 0, the actual positives and negatives shift from the second row to the first row

Example Receiver Operating Characteristic (ROC) Curve

y	$\hat{y}$	0.3 threshold	0.5 threshold	0.6 threshold
0 (not hot dog)	0.3	0 (TN)	0 (TN)	0 (TN)
1 (hot dog)	0.5	1 (TP)	0 (FN)	0 (FN)
0 (not hot dog)	0.6	1 (FP)	1 (FP)	0 (TN)
1 (hot dog)	0.9	1 (TP)	1 (TP)	1 (TP)

	0.3 threshold	0.5 threshold	0.6 threshold
True Positive Rate $= \frac{TP}{TP+FN}$	$\frac{2}{2+0} = 1.0$	$\frac{1}{1+1} = 0.5$	$\frac{1}{1+1} = 0.5$
False Positive Rate $= \frac{FP}{FP+TN}$	$\frac{1}{1+1} = 0.5$	$\frac{1}{1+1} = 0.5$	$\frac{0}{0+2} = 0.0$



Area Under the Curve is 75%

Loading Sentiment Classifier Dependencies

```
import keras
from keras.datasets import imdb # new!
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from keras.layers import Embedding # new!
from keras.callbacks import ModelCheckpoint # new!
import os # new!
from sklearn.metrics import roc_auc_score, roc_curve # new!
import pandas as pd
import matplotlib.pyplot as plt # new!
%matplotlib inline
```

Sentiment Classifier Hyperparameters

```
# output directory name:
output_dir = 'model_output/dense'
# training:
epochs = 4
batch_size = 128
# vector-space embedding:
n_dim = 64
n_unique_words = 5000
n_words_to_skip = 50
max_review_length = 100
pad_type = trunc_type = 'pre'
# neural network architecture:
n_dense = 64
dropout = 0.5
```

Examining IMDB Data

```
(x_train, y_train), (x_valid, y_valid) = \
    imdb.load_data(num_words=n_unique_words, skip_top=n_words_to_skip)
for x in x_train[0:6]:
    print(len(x))
word_index = keras.datasets.imdb.get_word_index()
word_index = {k:(v+3) for k,v in word_index.items()}
word_index["PAD"] = 0
word_index["START"] = 1
word_index["UNK"] = 2
index_word = {v:k for k,v in word_index.items()}
' '.join(index_word[id] for id in x_train[0])
```

Example Review

```
(all_x_train,_),(all_x_valid,_) = imdb.load_data()
' '.join(index_word[id] for id in all_x_train[0])
```

```
"START this film was just brilliant casting location scenery story direct
ion everyone's really suited the part they played and you could just imag
ine being there robert redford's is an amazing actor and now the same bei
ng director norman's father came from the same scottish island as myself
so i loved the fact there was a real connection with this film the witty
remarks throughout the film were great it was just brilliant so much that
i bought the film as soon as it was released for retail and would recomme
nd it to everyone to watch and the fly fishing was amazing really cried a
t the end it was so sad and you know what they say if you cry at a film i
t must have been good and this definitely was also congratulations to the
two little boy's that played the part's of norman and paul they were just
brilliant children are often left out of the praising list i think becaus
e the stars that play them all grown up are such a big profile for the wh
ole film but these children are amazing and should be praised for what th
ey have done don't you think the whole story was so lovely because it was
true and was someone's life after all that was shared with us all"
```

Dense Model Architecture

```
model = Sequential()
model.add(Embedding(n_unique_words, n_dim,
   input_length=max_review_length))
model.add(Flatten())
model.add(Dense(n_dense, activation='relu'))
model.add(Dropout(dropout))
# model.add(Dense(n_dense, activation='relu'))
# model.add(Dropout(dropout))
model.add(Dense(1, activation='sigmoid'))
```

Dense Model Summary

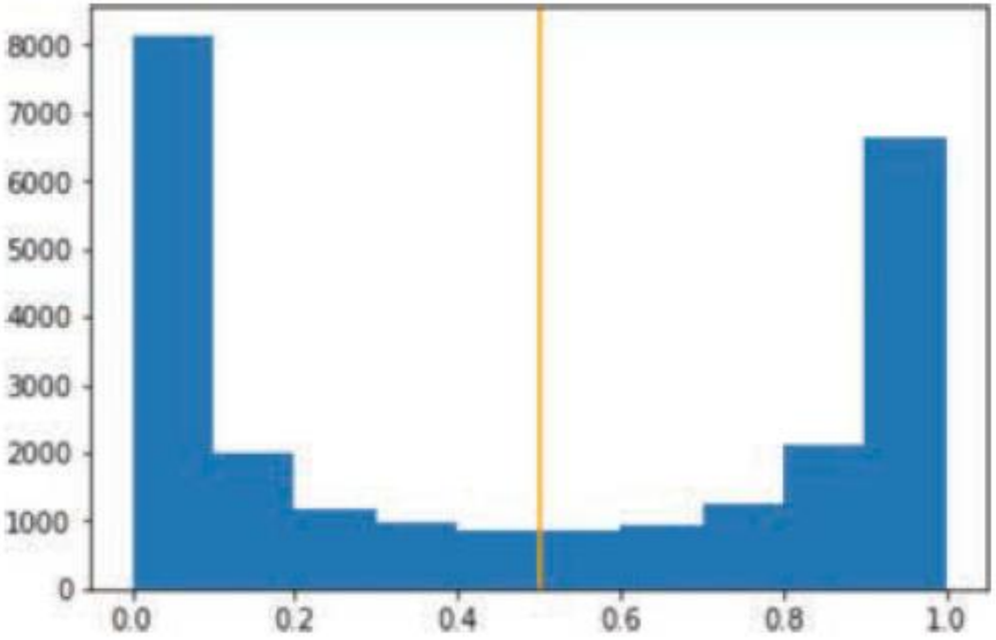
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 64)           320000
_________________________________________________________________
flatten_1 (Flatten)          (None, 6400)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                409664
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 729,729
Trainable params: 729,729
Non-trainable params: 0
```

Dense Model Train/Predict

```
model.compile(loss='binary_crossentropy', optimizer='adam',
    metrics=['accuracy'])
modelcheckpoint = ModelCheckpoint(filepath=output_dir+
    "/weights.{epoch:02d}.hdf5")
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
model.fit(x_train, y_train,
    batch_size=batch_size, epochs=epochs, verbose=1,
    validation_data=(x_valid, y_valid),
    callbacks=[modelcheckpoint])
model.load_weights(output_dir+"/weights.02.hdf5")
y_hat = model.predict(x_valid)
```

Dense Model Results

```
Train on 25000 samples, validate on 25000 samples
Epoch 1/4
25000/25000 [==============================] - 2s 80us/step - loss: 0.5612 - acc: 0.6892 - val_loss: 0.3630 - val_acc: 0.8398
Epoch 2/4
25000/25000 [==============================] - 2s 69us/step - loss: 0.2851 - acc: 0.8841 - val_loss: 0.3486 - val_acc: 0.8447
Epoch 3/4
25000/25000 [==============================] - 2s 70us/step - loss: 0.1158 - acc: 0.9646 - val_loss: 0.4252 - val_acc: 0.8337
Epoch 4/4
25000/25000 [==============================] - 2s 70us/step - loss: 0.0237 - acc: 0.9961 - val_loss: 0.5304 - val_acc: 0.8340
```



plt.hist(y_hat)

AUC ROC (not shown): 92.9%

Example False Positive

"START wow another kevin costner hero movie postman tin cup waterworld bodyguard wyatt earp robin hood even that baseball movie seems like he makes movies specifically to be the center of attention the characters are almost always the same the heroics the flaws the greatness the fall the redemption yup within the 1st 5 minutes of the movie we're all supposed to be in awe of his character and it builds up more and more from there br br and this time the story story is just a collage of different movies you don't need a spoiler you've seen this movie several times though it had different titles you'll know what will happen way before it happens this is like mixing an officer and a gentleman with but both are easily better movies watch to see how this kind of movie should be made and also to see how an good but slightly underrated actor russell plays the hero"

Example False Negative

"START finally a true horror movie this is the first time in years that i had to cover my eyes i am a horror buff and i recommend this movie but it is quite gory i am not a big wrestling fan but kane really pulled the who le monster thing off i have to admit that i didn't want to see this movie my 17 year old dragged me to it but am very glad i did during and after t he movie i was looking over my shoulder i have to agree with others about the whole remake horror movies enough is enough i think that is why this movie is getting some good reviews it is a refreshing change and takes yo u back to the texas chainsaw first one michael myers and jason and no cgi crap"

Convolutional Classifier Hyperparameters

```
from keras.layers import Conv1D, GlobalMaxPooling1D
from keras.layers import SpatialDropout1D

# output directory name:
output_dir = 'model_output/conv'
# training:
epochs = 4
batch_size = 128
# vector-space embedding:
n_dim = 64
n_unique_words = 5000
max_review_length = 400
pad_type = trunc_type = 'pre'
drop_embed = 0.2 # new!
# convolutional layer architecture:
n_conv = 256 # filters, a.k.a. kernels
k_conv = 3 # kernel length
# dense layer architecture:
n_dense = 256
dropout = 0.2
```

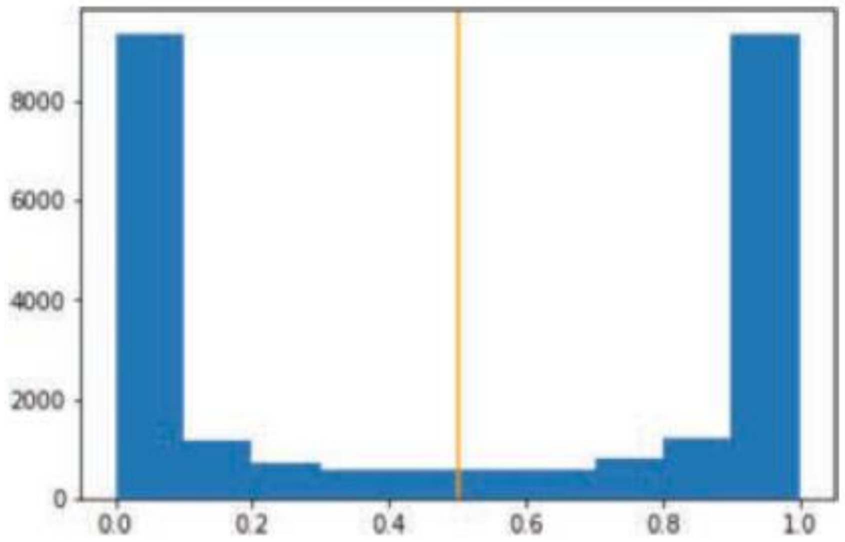
Convolutional Model Architecture

```
model = Sequential()
# vector-space embedding:
model.add(Embedding(n_unique_words, n_dim, input_length=max_review_length))
model.add(SpatialDropout1D(drop_embed))
# convolutional layer:
model.add(Conv1D(n_conv, k_conv, activation='relu'))
# model.add(Conv1D(n_conv, k_conv, activation='relu'))
model.add(GlobalMaxPooling1D())
# dense layer:
model.add(Dense(n_dense, activation='relu'))
model.add(Dropout(dropout))
# output layer:
model.add(Dense(1, activation='sigmoid'))
```

SpatialDropout1D() uses the same dropout mask for all timesteps

Convolutional Model Results

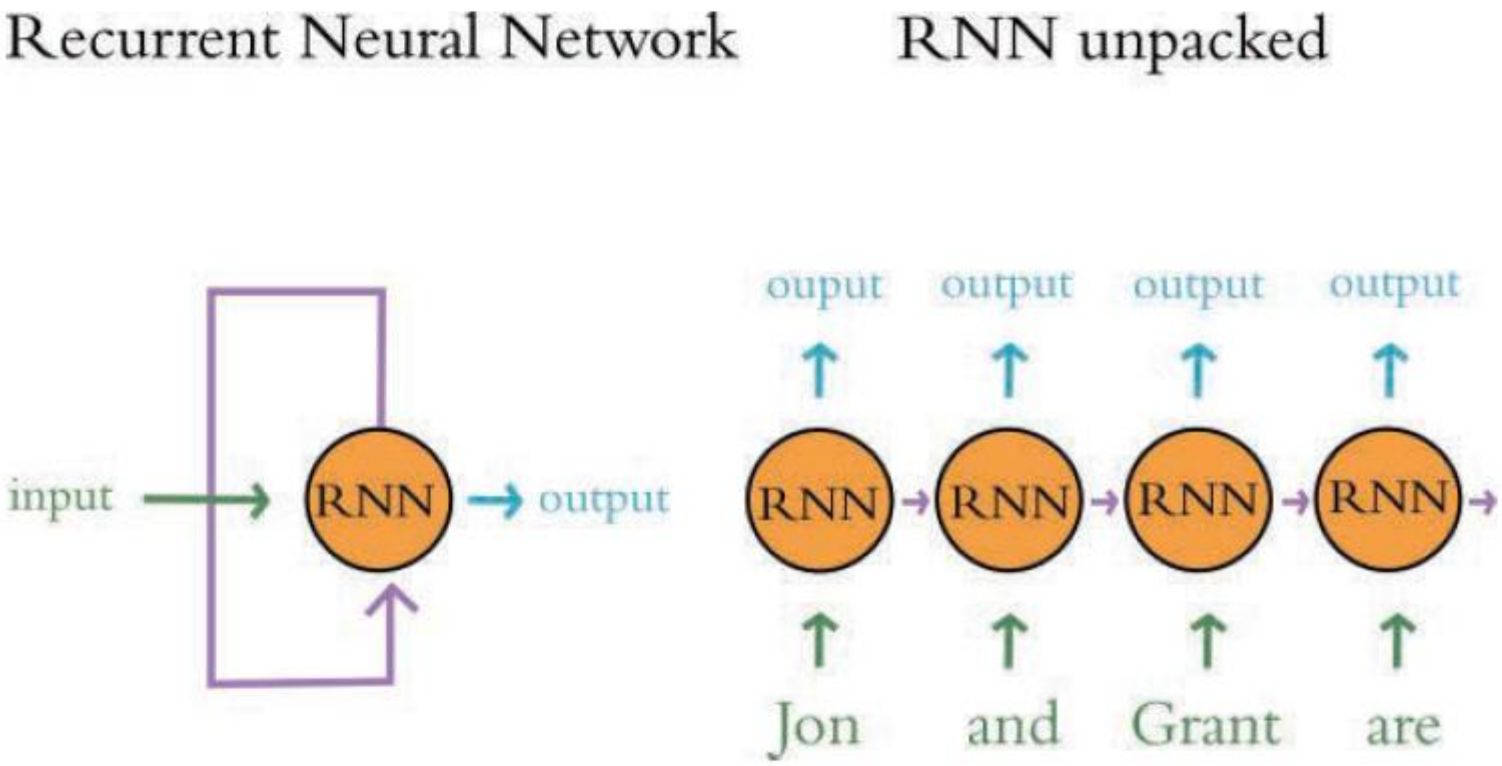
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 400, 64)           320000
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 400, 64)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 398, 256)          49408
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 435,457
Trainable params: 435,457
Non-trainable params: 0
```




```
Train on 25000 samples, validate on 25000 samples
Epoch 1/4
25000/25000 [==============================] - 41s 2ms/step - loss: 0.4894 - acc: 0.7447 - val_loss: 0.2971 - val_acc: 0.8750
Epoch 2/4
25000/25000 [==============================] - 41s 2ms/step - loss: 0.2534 - acc: 0.8972 - val_loss: 0.2604 - val_acc: 0.8914
Epoch 3/4
25000/25000 [==============================] - 41s 2ms/step - loss: 0.1709 - acc: 0.9357 - val_loss: 0.2577 - val_acc: 0.8959
Epoch 4/4
25000/25000 [==============================] - 41s 2ms/step - loss: 0.1151 - acc: 0.9589 - val_loss: 0.2828 - val_acc: 0.8934
```

Diagrams for a Recurrent Neural Network (RNN) Cell

“rolled”



“unrolled”

At each time step, we compute features for the input token and add them to the “memory” (past features)

https://github.com/tensorflow/tensorflow/blob/v2.4.1/tensorflow/python/keras/layers/recurrent.py#L370-L374

RNN Classifier Hyperparameters

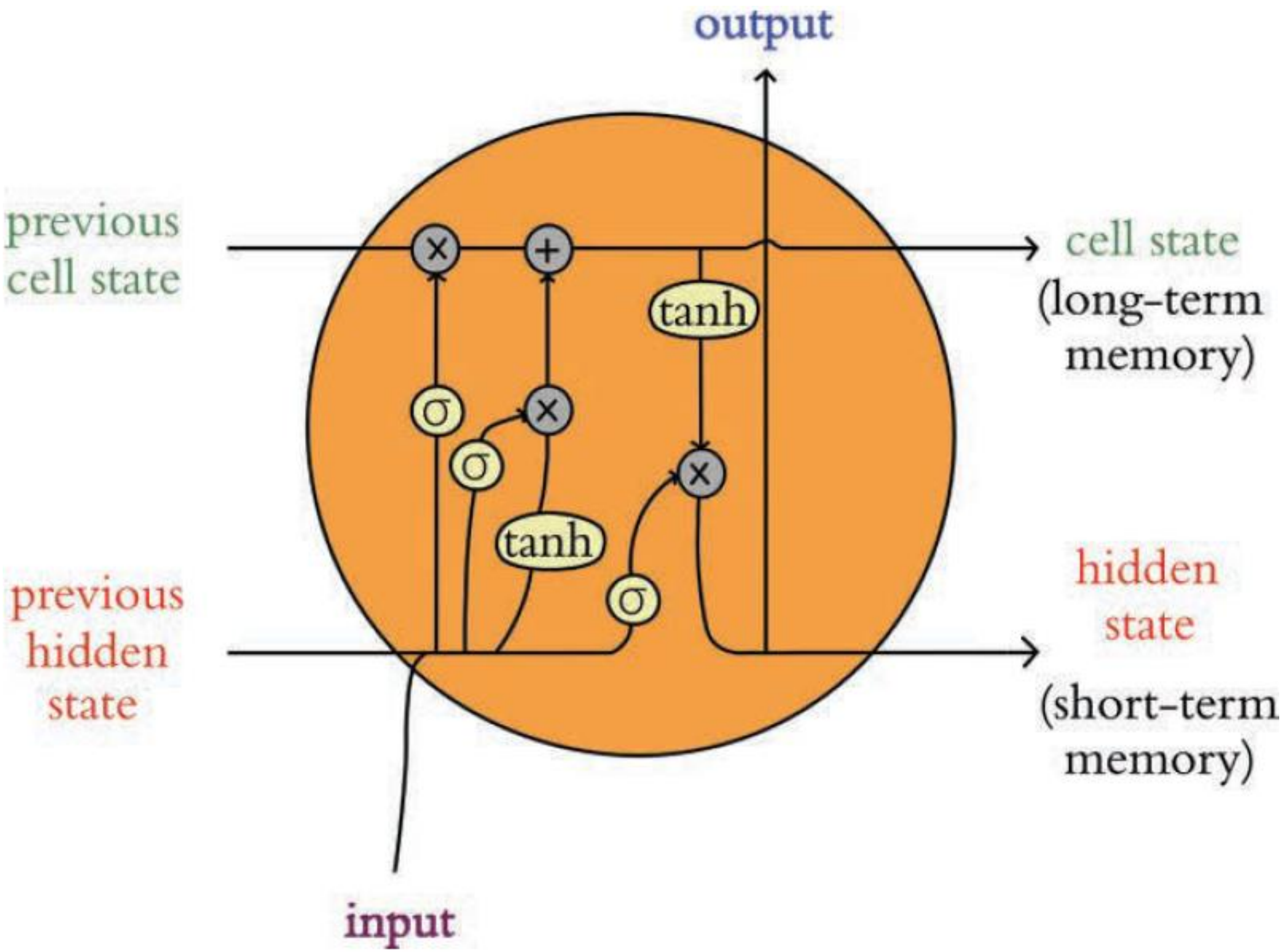
```
# output directory name:
output_dir = 'model_output/rnn'
# training:
epochs = 16 # way more!
batch_size = 128
# vector-space embedding:
n_dim = 64
n_unique_words = 10000
max_review_length = 100
# lowered due to vanishing gradient over time
pad_type = trunc_type = 'pre'
drop_embed = 0.2
# RNN layer architecture:
n_rnn = 256
drop_rnn = 0.2
```

RNN Classifier Architecture

```
from keras.layers import SimpleRNN
model = Sequential()
model.add(Embedding(n_unique_words, n_dim,
   input_length=max_review_length))
model.add(SpatialDropout1D(drop_embed))
model.add(SimpleRNN(n_rnn, dropout=drop_rnn))
model.add(Dense(1, activation='sigmoid'))
```

 https://github.com/tensorflow/tensorflow/blob/v2.4.1/tensorflow/python/keras/layers/recurrent.py#L2404

Diagram for a Long Short-Term Memory (LSTM) Cell



LSTM Classifier Hyperparameters

```
# output directory name:
output_dir = 'model_output/LSTM'
# training:
epochs = 4
batch_size = 128
# vector-space embedding:
n_dim = 64
n_unique_words = 10000
max_review_length = 100
pad_type = trunc_type = 'pre'
drop_embed = 0.2
# LSTM layer architecture:
n_lstm = 256
drop_lstm = 0.2
```

LSTM Classifier Architecture

```
from keras.layers import LSTM
model = Sequential()
model.add(Embedding(n_unique_words, n_dim,
input_length=max_review_length))
model.add(SpatialDropout1D(drop_embed))
model.add(LSTM(n_lstm, dropout=drop_lstm))
model.add(Dense(1, activation='sigmoid'))
```

Bidirectional LSTM Architecture

```
from keras.layers import LSTM
from keras.layers.wrappers import Bidirectional # new!
model = Sequential()
model.add(Embedding(n_unique_words, n_dim,
input_length=max_review_length))
model.add(SpatialDropout1D(drop_embed))
model.add(Bidirectional(LSTM(n_lstm, dropout=drop_lstm)))
model.add(Dense(1, activation='sigmoid'))
```

Stacked Bidirectional Classifier Architecture

```
from keras.layers import LSTM
from keras.layers.wrappers import Bidirectional
model = Sequential()
model.add(Embedding(n_unique_words, n_dim,
    input_length=max_review_length))
model.add(SpatialDropout1D(drop_embed))
model.add(Bidirectional(LSTM(n_lstm_1, dropout=drop_lstm,
    return_sequences=True))) # new!
model.add(Bidirectional(LSTM(n_lstm_2, dropout=drop_lstm)))
model.add(Dense(1, activation='sigmoid'))
```

Gated Recurrent Unit (GRU) Cells

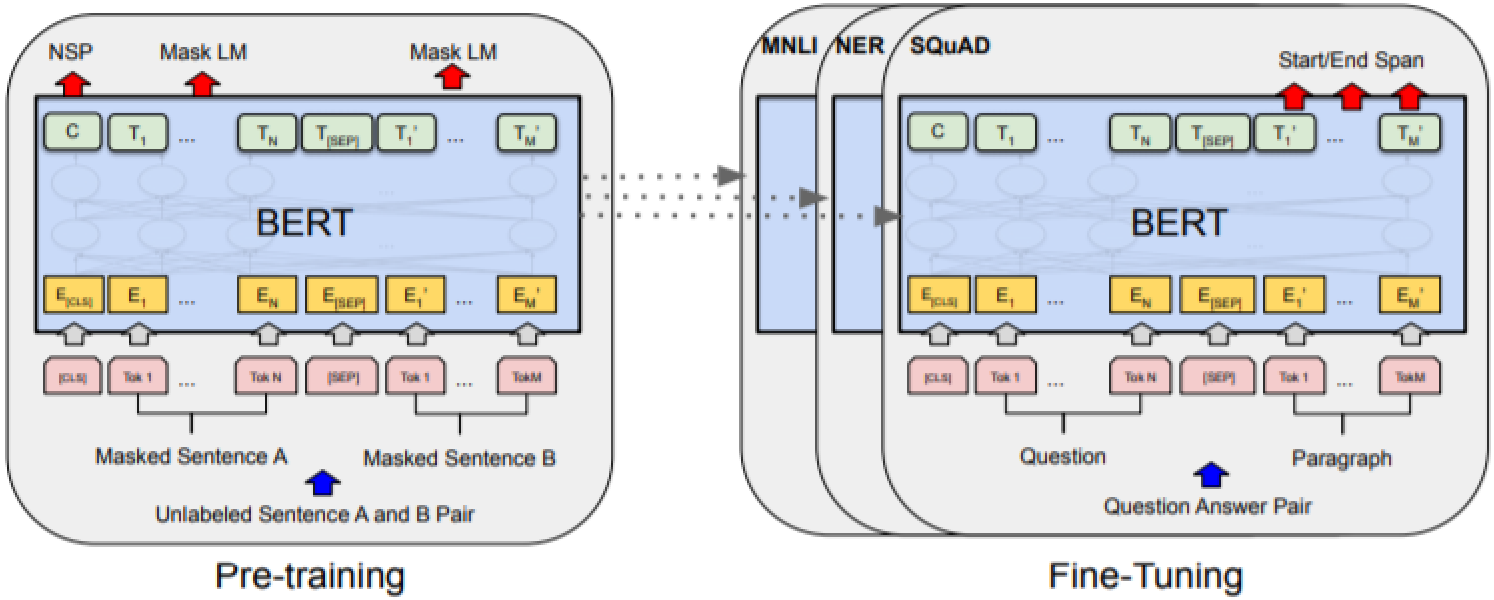
- GRUs are slightly less computationally intensive than LSTMs because they involve only three activation functions, and yet their performance often approaches the performance of LSTMs
- If a bit more compute isn't a deal breaker for you, we see little advantage in choosing a GRU over an LSTM
- If you're interested in trying a GRU in Keras anyway, it's as easy as importing the GRU() layer type and dropping it into a model architecture where you might otherwise place an LSTM() layer

Seq2Seq and Attention

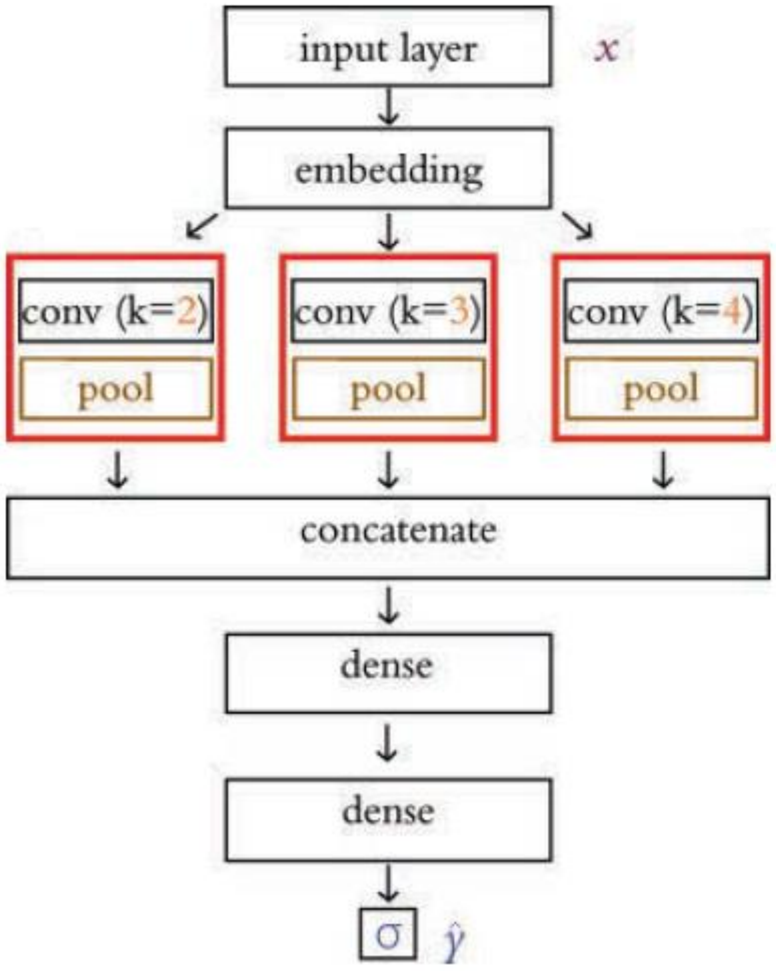
- Sequence-to-Sequence (“seek-to-seek”) models: take a sequence as input and produce a sequence as output
  - Machine translation
  - Chatbots
- Encoder-Decoder architecture
  - Encoder processes the input sequence
  - Decoder produces the output: one position at a time
    - Recurrent cell context can be passed from encoder to decoder
    - Begin with start of sequence “token”
    - End with end of sequence “token”
- Attention
  - Full sequence of outputs can be used
  - Each decoder position gets its own weighted average of encoder outputs

Transfer Learning in NLP

- Pretrain on some large corpus (or corpora)
- Finetune on your application of interest



Non-Sequential Architecture Example



Multi-ConvNet Classifier Hyperparameters

```
# output directory name:
output_dir = 'model_output/multiconv'
# training:
epochs = 4
batch_size = 128
# vector-space embedding:
n_dim = 64
n_unique_words = 5000
max_review_length = 400
pad_type = trunc_type = 'pre'
drop_embed = 0.2
# convolutional layer architecture:
n_conv_1 = n_conv_2 = n_conv_3 = 256
k_conv_1 = 3
k_conv_2 = 2
k_conv_3 = 4
# dense layer architecture:
n_dense = 256
dropout = 0.2
```

Multi-ConvNet Classifier Architecture

```
from keras.models import Model
from keras.layers import Input, concatenate
# input layer:
input_layer = Input(shape=(max_review_length,),
   dtype='int16', name='input')
# embedding:
embedding_layer = Embedding(n_unique_words, n_dim,
   name='embedding')(input_layer)
drop_embed_layer = SpatialDropout1D(drop_embed,
   name='drop_embed')(embedding_layer)
# three parallel convolutional streams:
conv_1 = Conv1D(n_conv_1, k_conv_1,
   activation='relu', name='conv_1')(drop_embed_layer)
maxp_1 = GlobalMaxPooling1D(name='maxp_1')(conv_1)
conv_2 = Conv1D(n_conv_2, k_conv_2,
   activation='relu', name='conv_2')(drop_embed_layer)
maxp_2 = GlobalMaxPooling1D(name='maxp_2')(conv_2)
conv_3 = Conv1D(n_conv_3, k_conv_3,
   activation='relu', name='conv_3')(drop_embed_layer)
maxp_3 = GlobalMaxPooling1D(name='maxp_3')(conv_3)
# concatenate the activations from the three streams:
concat = concatenate([maxp_1, maxp_2, maxp_3])
# dense hidden layers:
dense_layer = Dense(n_dense,
activation='relu', name='dense')(concat)
drop_dense_layer = Dropout(dropout, name='drop_dense')(dense_layer)
dense_2 = Dense(int(n_dense/4),
   activation='relu', name='dense_2')(drop_dense_layer)
dropout_2 = Dropout(dropout, name='drop_dense_2')(dense_2)
# sigmoid output layer:
predictions = Dense(1, activation='sigmoid', name='output')(dropout_2)
# create model:
model = Model(input_layer, predictions)
```

Comparison of IMDB Architectures

Model	ROC AUC (%)	
Dense	92.9	
Convolutional	96.1	
Simple RNN	84.9	
LSTM	92.8	
Bi-LSTM	93.5	
Stacked Bi-LSTM	94.9	
GRU	93.0	
Conv-LSTM	94.5	
Multi-ConvNet	96.2	val_acc = 89.4%

https://github.com/the-deep-learners/deep-learning-illustrated/blob/master/notebooks/multi_convnet_sentiment_classifier.ipynb

Summary

In this chapter, we discussed methods for preprocessing natural language data, ways to create word vectors from a corpus of natural language, and the procedure for calculating the area under the receiver operating characteristic curve. In the second half of the chapter, we applied this knowledge to experiment with a wide range of deep learning NLP models for classifying film reviews as favorable or negative. Some of these models involved layer types you were familiar with from earlier chapters (i.e., dense and convolutional layers), while later ones involved new layer types from the RNN family (LSTMs and GRUs) and, for the first time in this book, a non-sequential model architecture.

A summary of the results of our sentiment-classifier experiments are provided in Table 11.6. We hypothesize that, had our natural language dataset been much larger, the Bi-LSTM architectures might have outperformed the convolutional ones.

Key Concepts

- parameters:
  - weight w
  - bias b
- activation a
- artificial neurons:
  - sigmoid
  - tanh
  - ReLU
  - linear
- input layer
- hidden layer
- output layer
- layer types:
  - dense (fully connected)
  - softmax
  - convolutional
  - max-pooling
  - flatten
  - embedding
  - RNN
  - (bidirectional-)LSTM
  - concatenate
- cost (loss) functions:
  - quadratic (mean squared error)
  - cross-entropy
- forward propagation
- backpropagation
- unstable (especially vanishing) gradients
- Glorot weight initialization
- batch normalization
- dropout
- optimizers:
  - stochastic gradient descent
  - Adam
- optimizer hyperparameters:
  - learning rate η
  - batch size
- word2vec

Word2Vec Tutorial

https://www.tensorflow.org/tutorials/text/word2vec

sampling_table example (for sampling target words):

[0.00315225 0.00315225 0.00547597 0.00741556 0.00912817 0.01068435 0.01212381 0.01347162]

log_uniform_candidate_sampler example (for negative sampling of context words):

[0.31546488 0.18453512 0.13092975 0.10155701 0.08297812 0.07015700 0.06077276 0.05360537]

```
class Word2Vec(Model):
  def __init__(self, vocab_size, embedding_dim):
    super(Word2Vec, self).__init__()
    self.target_embedding = Embedding(vocab_size,
                                      embedding_dim,
                                      input_length=1,
                                      name="w2v_embedding")
    self.context_embedding = Embedding(vocab_size,
                                       embedding_dim,
                                       input_length=num_ns+1)
    self.dots = Dot(axes=(3, 2))
    self.flatten = Flatten()

  def call(self, pair):
    target, context = pair
    word_emb = self.target_embedding(target)
    context_emb = self.context_embedding(context)
    dots = self.dots([context_emb, word_emb])
    return self.flatten(dots)
```

FastText

- The FastText approach to text classification is simple, but it can be reasonably effective: https://arxiv.org/abs/1607.01759
- "The first weight matrix A is a look-up table over the words. The word representations are then averaged into a text representation, which is in turn fed to a linear classifier."

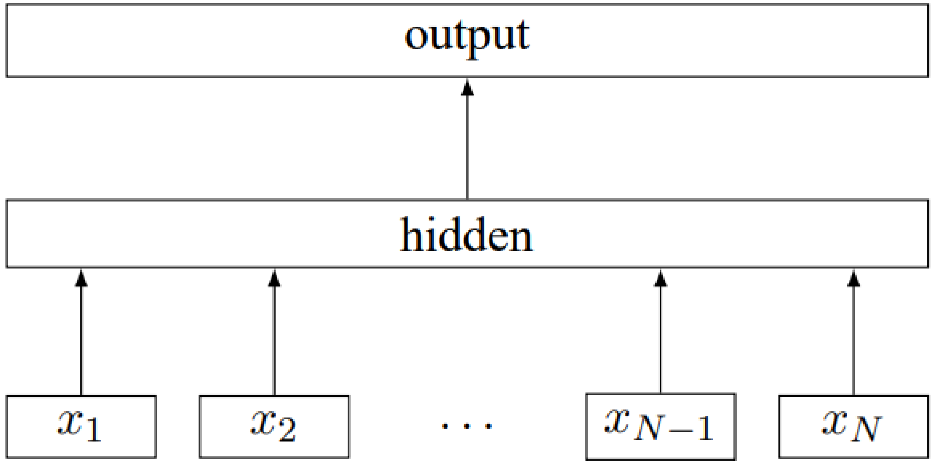


Figure 1: Model architecture of `fastText` for a sentence with N ngram features $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.