

Natural Language Processing in Finance

May 2022

Artificial Intelligence Finance Institute

NYU Courant



Our papers

- 1.  [Deep Learning for Equity Time Series Prediction](#)

Number of pages: 31 • Posted: 24 Nov 2020
Miquel Noguer i Alonso, [Gilberto Batres-Estrada](#) and [Aymeric Moulin](#)
Artificial Intelligence in Finance Institute, Trell Technologies and Jp Morgan
[View PDF](#) | [Download](#) | [Show Abstract](#)

Downloads **1,863**
(12,315)

- 2.  [Deep Reinforcement Learning for Asset Allocation in US Equities](#)

Number of pages: 29 • Posted: 19 Oct 2020
Miquel Noguer i Alonso and [Sonam Srivastava](#)
Artificial Intelligence in Finance Institute and Wright Research
[View PDF](#) | [Download](#) | [Show Abstract](#)

Downloads **1,516**
(16,961)

- 3.  [The Shape of Performance Curve in Financial Time Series](#)

Number of pages: 12 • Posted: 20 Dec 2021
Miquel Noguer i Alonso and [Sonam Srivastava](#)
Artificial Intelligence in Finance Institute and Wright Research
[View PDF](#) | [Download](#) | [Show Abstract](#)

Downloads **502**
(77,867)

- 4.  [A Meta-Learning approach to Model Uncertainty in Financial Time Series](#)

Number of pages: 27 • Posted: 30 Mar 2021
Miquel Noguer i Alonso, [Gilberto Batres-Estrada](#) and [Ghozlane Yahiaoui](#)
Artificial Intelligence in Finance Institute, Trell Technologies and *affiliation not provided to SSRN*
[View PDF](#) | [Download](#) | [Show Abstract](#)

Downloads **487**
(80,794)

- 5.  [Deep Signature models for Financial Equity Time Series prediction](#)

Number of pages: 7 • Posted: 13 May 2022
Miquel Noguer i Alonso, [Himanshu Agrawal](#) and [Sonam Srivastava](#)
Artificial Intelligence in Finance Institute, Q1 Cap Markets LLP and Wright Research
[View PDF](#) | [Download](#) | [Show Abstract](#)

Downloads **277**
(152,412)

Our papers

□	1.	 Sustainable Investment - Exploring the Linkage between Alpha, ESG, and SDG's	Downloads 912 (31,511)	Citation 1
		Number of pages: 34 • Posted: 11 Jun 2020 • Last Revised: 12 Oct 2020		
		Madelyn Antoncic , Geert Bekaert , Richard V Rothenberg and Miquel Noguer		
		Global AI Corp, Columbia Business School - Finance and Economics, Global AI Co and Global AI Corp		
		View PDF Download Show Abstract		
1.	arXiv:2111.00526 [pdf, other]	cs.CL	q-fin.CP	q-fin.PM
	FinEAS: Financial Embedding Analysis of Sentiment			
	Authors: Asier Gutiérrez-Fandiño , Miquel Noguer i Alonso , Petter Kolm , Jordi Armengol-Estepá			
	Abstract: We introduce a new language representation model in finance called Financial Embedding Analysis of Sentiment (FinEAS). In financial markets, news and investor sentiment are significant drivers of security prices. Thus, leveraging the capabilities of modern NLP approaches for financial sentiment analysis is a crucial component in identifying patterns and trends that are useful for market participan... ▼ More			
	Submitted 19 November, 2021; v1 submitted 31 October, 2021; originally announced November 2021.			
2.	arXiv:2010.04404 [pdf, other]	q-fin.PM	q-fin.CP	
	Deep Reinforcement Learning for Asset Allocation in US Equities			
	Authors: Miquel Noguer i Alonso , Sonam Srivastava			
	Abstract: Reinforcement learning is a machine learning approach concerned with solving dynamic optimization problems in an almost model-free way by maximizing a reward function in state and action spaces. This property makes it an exciting area of research for financial problems. Asset allocation, where the goal is to obtain the weights of the assets that maximize the rewards in a given state of the market... ▼ More			
	Submitted 9 October, 2020; originally announced October 2020.			
	Comments: Submitting to Journal of Machine Learning in Finance			
	MSC Class: 91-10 ACM Class: I.2.m			

Lecture Outline

1. NLP Definitions
2. NLP in Finance Use Cases
3. NLP Essentials
4. Traditional Methods
5. Deep Learning Methods : LSTM, BERT and TRANSFORMERS
6. FinEAS : Financial Embedding Analysis of Sentiment

What Is NLP?

NLP is a sub-field of artificial intelligence ('AI'), which seeks to program computers to process, understand and analyse large amounts of human (or 'natural') language.

Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural-language generation.

Natural Language Processing Applications

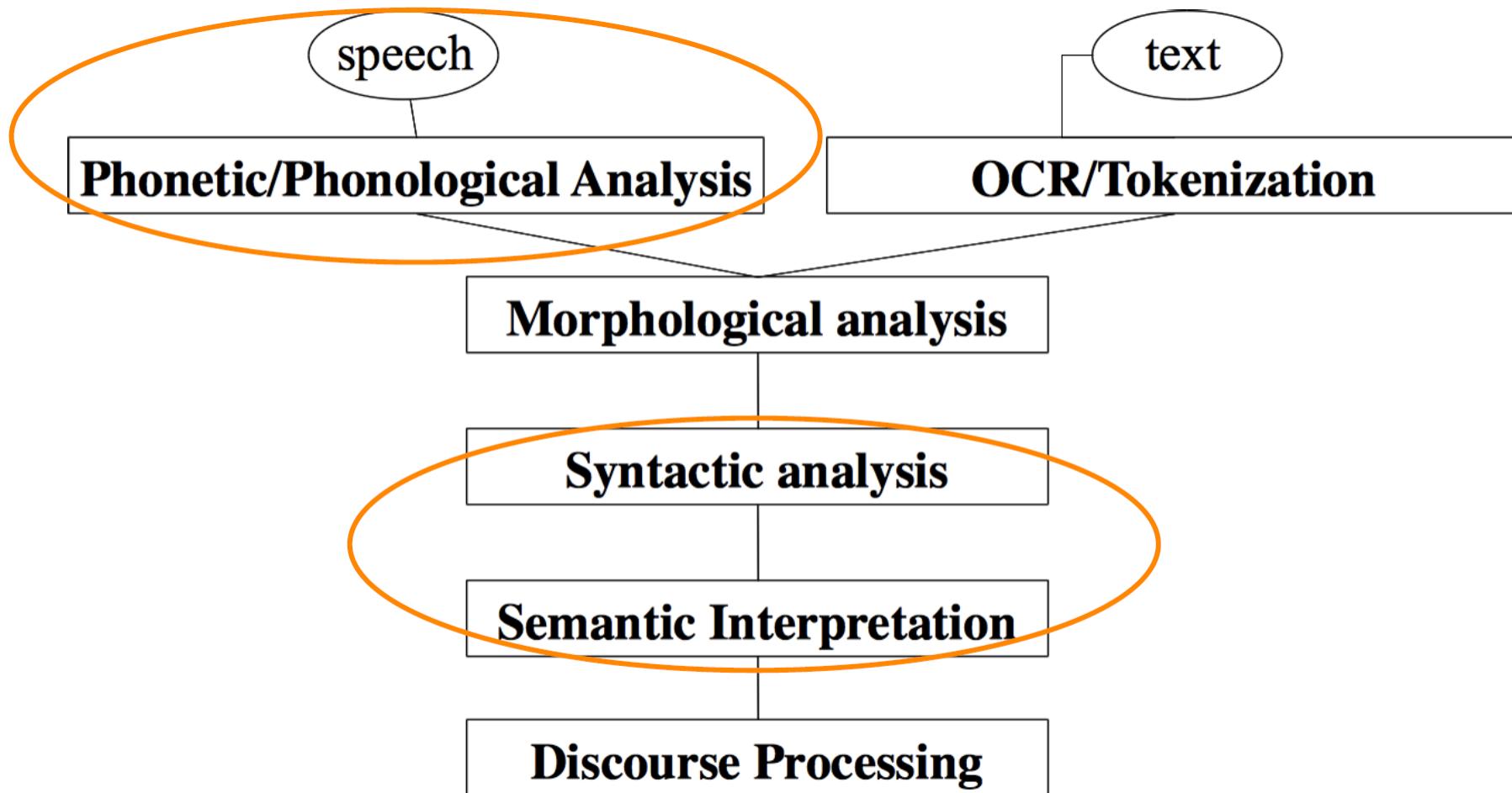
Applications range from simple to complex:

- Spell checking, keyword search, finding synonyms
- Extracting information from websites such as product price, dates, location, people or company names
- Classifying: reading level of school texts, positive/negative sentiment of longer documents
- Machine translation
- Spoken dialog systems
- Complex question answering

NLP in Industry

- Online advertisement matching
- Search
- Automated/assisted translation
- Sentiment analysis for marketing or finance/trading
- Speech recognition
- Chatbots / Dialog agents
- Automating customer support
- Controlling devices
- Ordering goods

Natural Language Processing Levels

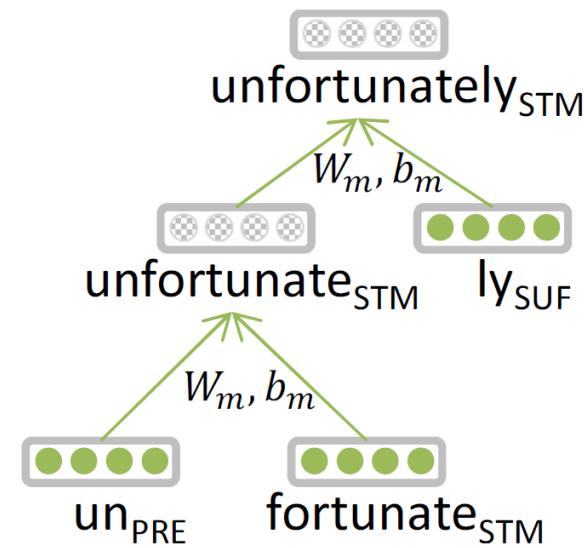


Representations of NLP Levels: Morphology

- Traditional: Words are made of morphemes

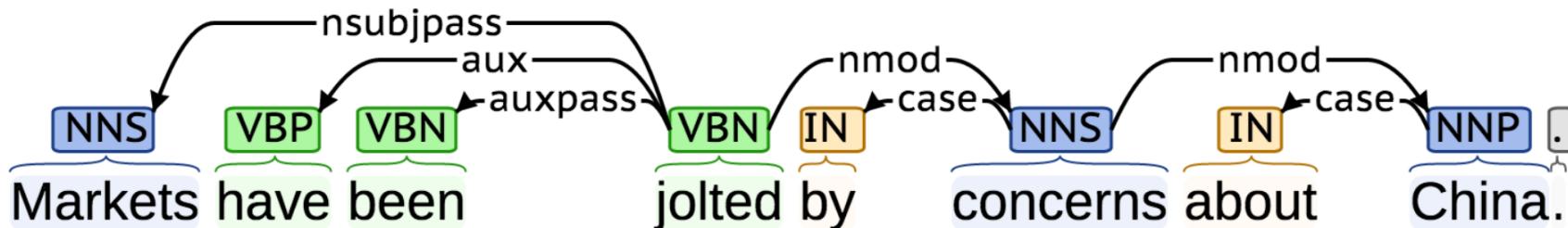
prefix stem suffix
un interest ed

- DL:
 - every morpheme is a vector
 - a neural network combines two vectors into one vector
 - Luong et al. 2013



NLP Tools: Parsing for sentence structure

Neural networks can accurately determine the structure of sentences, supporting interpretation



Softmax layer:

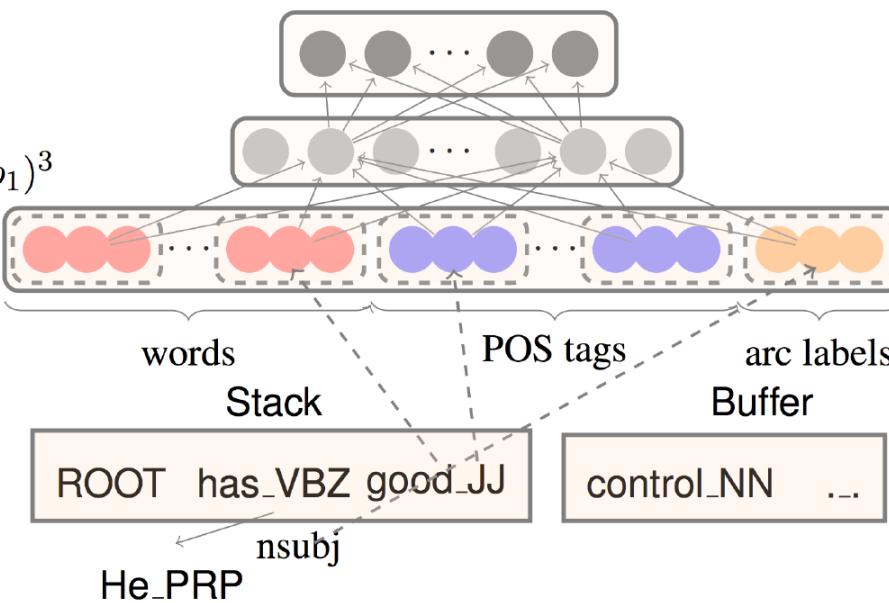
$$p = \text{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

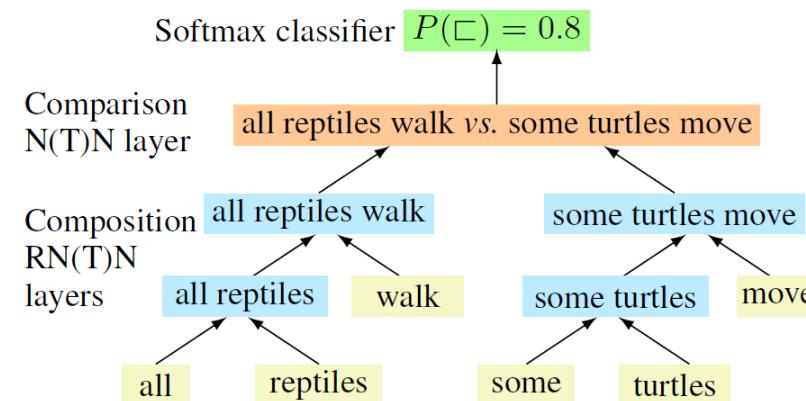
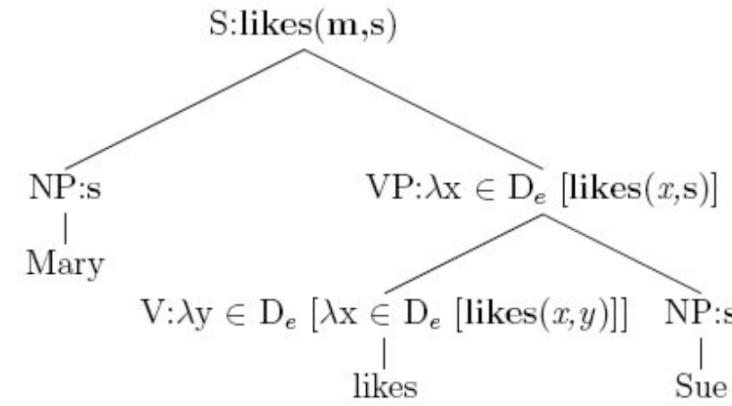
Input layer: $[x^w, x^t, x^l]$

Configuration



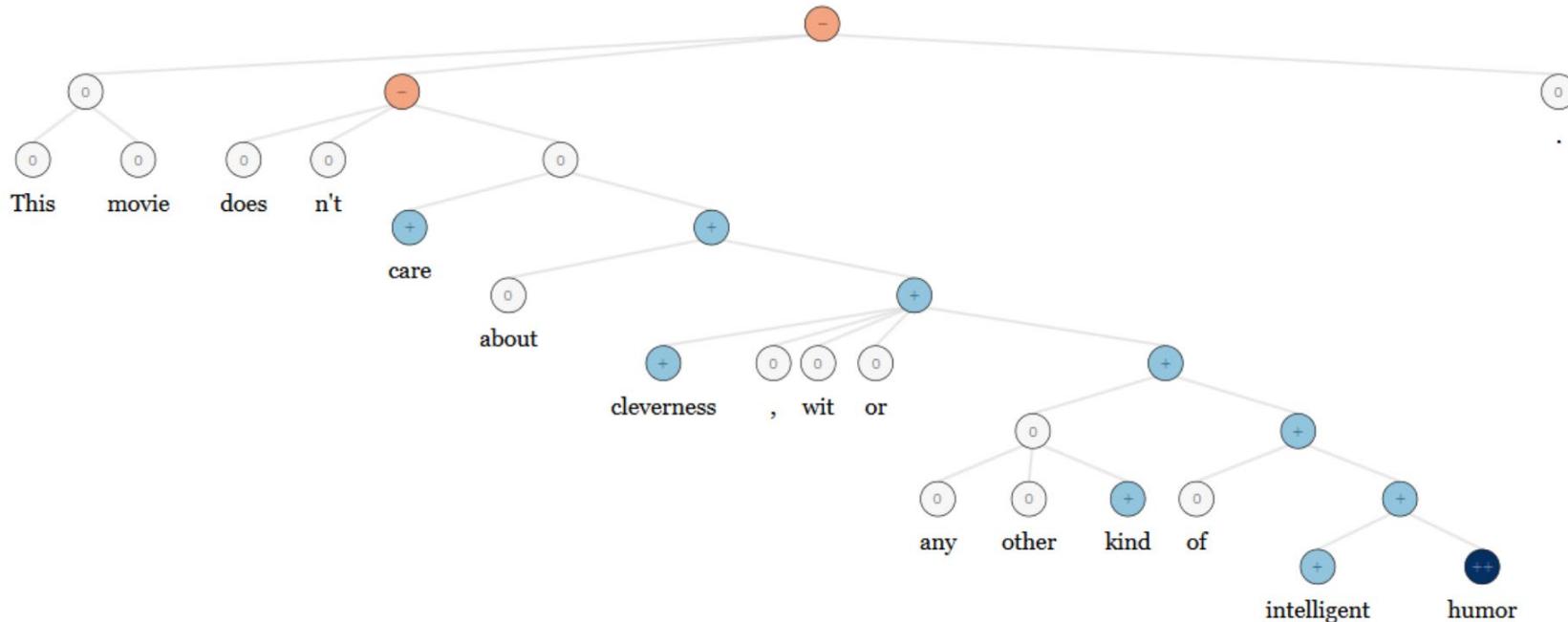
Representations of NLP Levels: Semantics

- Traditional: Lambda calculus
 - Carefully engineered functions
 - Take as inputs specific other functions
 - No notion of similarity or fuzziness of language
- DL:
 - Every word and every phrase and every logical expression is a vector
 - a neural network combines two vectors into one vector
 - Bowman et al. 2014



NLP Applications: Sentiment Analysis

- Traditional: Curated sentiment dictionaries combined with either bag-of-words representations (ignoring word order) or handdesigned negation features (ain't gonna capture everything)
- Same deep learning model that was used for morphology, syntax and logical semantics can be used! - RecursiveNN



GLUE Leaderboard

GLUE SuperGLUE Paper </> Code Tasks Leaderboard i FAQ Diagnostics Submit

Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-r
1	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6	73.5	97.2	94.0/92.0	93.0/92.4	76.1/91.0	91.6	9
2	ERNIE Team - Baidu	ERNIE	↗	90.4	74.4	97.5	93.5/91.4	93.0/92.6	75.2/90.9	91.4	9
3	Alibaba DAMO NLP	StructBERT	↗	90.3	75.3	97.1	93.9/91.9	93.0/92.5	74.8/91.0	90.9	9
4	T5 Team - Google	T5	↗	90.3	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2	9
5	Microsoft D365 AI & MSR AI & GATECHMT-DNN-SMART		↗	89.9	69.5	97.5	93.7/91.6	92.9/92.5	73.9/90.2	91.0	9
6	Zihang Dai	Funnel-Transformer (Ensemble B10-10-10H1024)	↗	89.7	70.5	97.5	93.4/91.2	92.6/92.3	75.4/90.7	91.4	9
7	ELECTRA Team	ELECTRA-Large + Standard Tricks	↗	89.4	71.7	97.1	93.1/90.7	92.9/92.5	75.6/90.8	91.3	9
8	Huawei Noah's Ark Lab	NEZHA-Large		89.1	69.9	97.3	93.3/91.0	92.4/91.9	74.2/90.6	91.0	9
9	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)	↗	88.4	68.0	96.8	93.1/90.8	92.3/92.1	74.8/90.3	91.1	9
10	Junjie Yang	HIRE-RoBERTa	↗	88.3	68.6	97.1	93.0/90.7	92.4/92.0	74.3/90.2	90.7	12

NLP Tools

https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

<https://huggingface.co/>

<https://www.tensorflow.org/tutorials>

https://pytorch.org/tutorials/beginner/deep_learning_nlp_tutorial.html

Natural Language Processing in Finance

Natural Language Processing in Finance – MAN AHL Report

1. Introduction

“We want Facebook to be somewhere where you can start meaningful relationships,” Mark Zuckerberg said on 1 May, 2018.

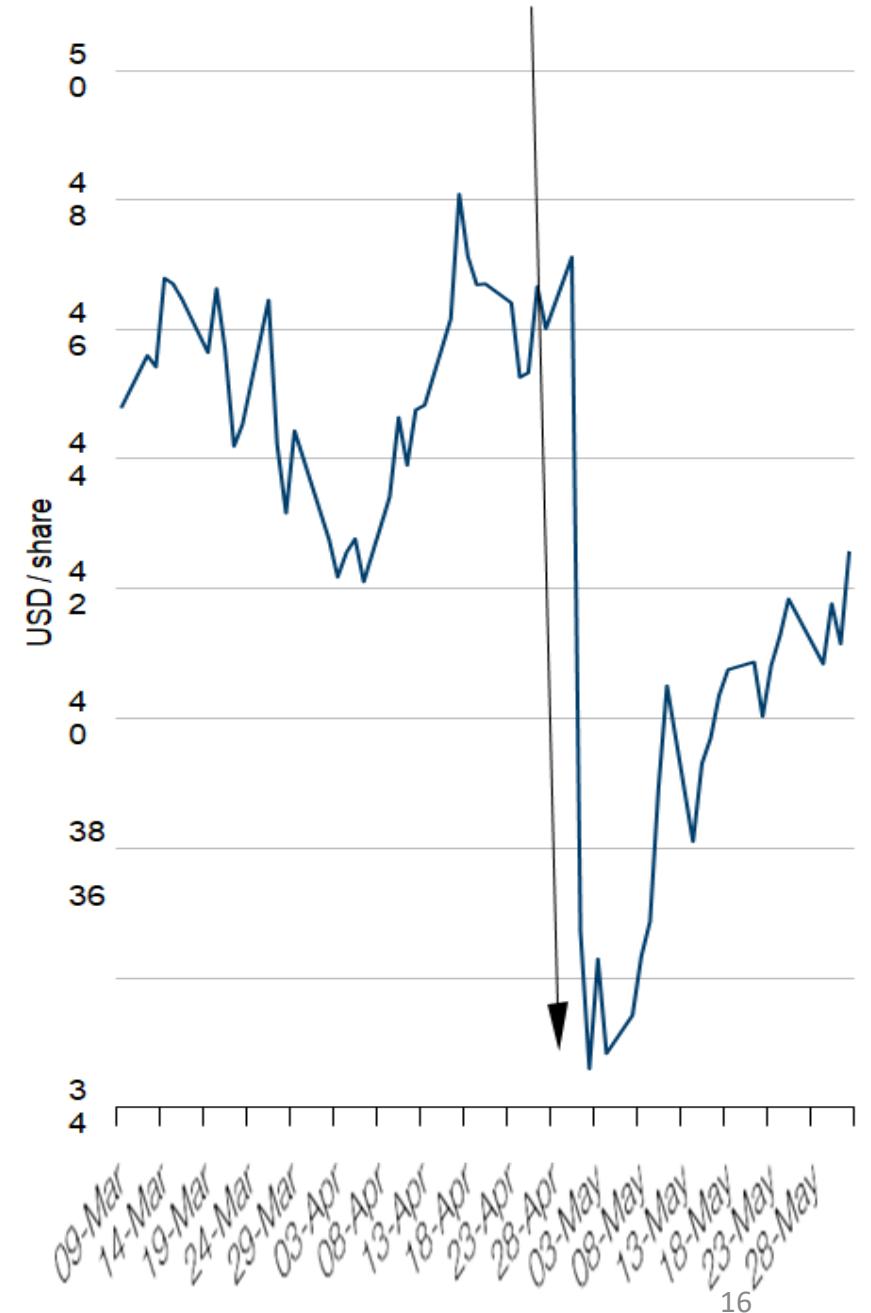
The announcement sparked gasps – not just from the crowd in front of whom Zuckerberg was talking – but also in financial markets. The share price of Match Group (the company that owns Match.com, Tinder and other dating websites) plunged by more than 20%.

Natural Language Processing in Finance

Figure 1. Facebook's Announcement Creates Ripples in Match.com

Share price of Match.com plunged on 1 May, 2018, as Facebook announced that it would integrate a feature for online dating directly onto its app.

Source: Bloomberg; between 9 March, 2018 and 1 June, 2018.



What Is NLP?

NLP is a sub-field of artificial intelligence ('AI'), which seeks to program computers to process, understand and analyse large amounts of human (or 'natural') language. How is this useful in finance?

1. Detecting Material Events

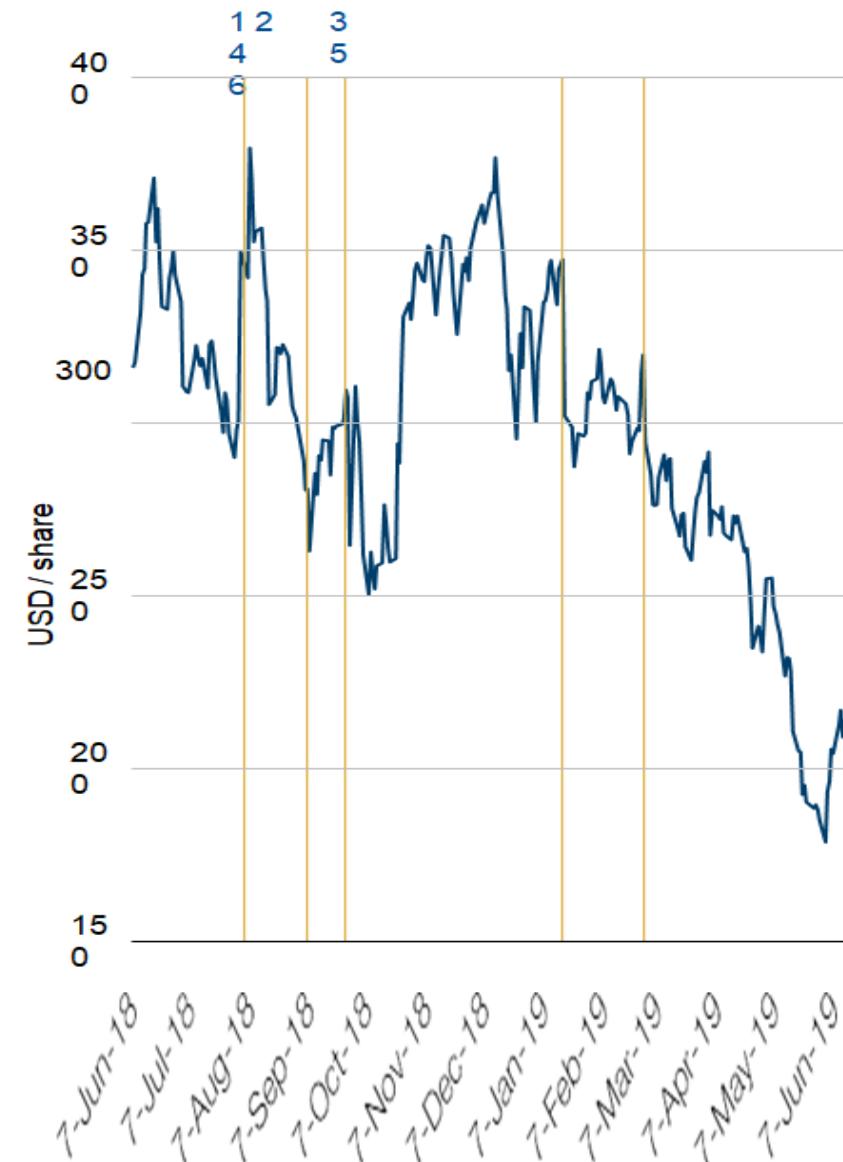
As we saw in the Facebook example, it's useful in uncovering market-moving events. Facebook unveiled a new product – like Apple unveiling the iPhone – and that resulted in a very strong market move. Numerous such events happen in financial markets all the time. Indeed, for a lot of them, text, or even the spoken word, is the primary source. As such, methods from NLP can be used to automate this process: monitoring many text data streams and automatically issuing notifications upon the emergence of market-moving events.

Material Events

Source: Bloomberg; as of 21 June, 2019.

1. 2 August, 2018: Shares soar as Tesla says production of its lower-cost Model 3 sedan is growing and CEO Elon Musk says the company does not expect to need to raise more money from investors.
2. 7 August, 2018: Musk announces on Twitter that he wants to take Tesla private in a deal that would value the company at USD70 billion.
3. 8 September, 2018: Just hours after Musk finishes smoking marijuana in a more than 2 1/2-hour podcast with comedian Joe Rogan, it is confirmed that both his chief accounting officer and head of human resources are leaving. Shares plunge.
4. 27 September, 2018: Shares fall as the SEC accuses Musk of misleading investors with his 7 August tweet about taking Tesla private, raising questions about Musk's future.
5. 18 January, 2019: Shares fall sharply as Musk warns that Tesla could struggle to make a profit in the first quarter and as he cuts more than 3,000 jobs from the electric carmaker.
6. 1 March, 2019: Shares slide as Musk confirms that Tesla will not be profitable in the first quarter.

Figure 2. An Example of How Events Affected Tesla



2. Understanding Document Tone

Perhaps one of the most common applications of NLP in finance is measuring document tone, also known as sentiment. The idea is simple: get the machine to ‘read’ a document and assign it a score from -10 (very negative) to +10 (very positive).

Take the sentence below:

French Cosmetics giant L’Oreal said strong demand for luxury skin creams helped it beat fourth-quarter sales forecasts - another company reporting better-than-feared demand from China after LVMH last week.¹

This would maybe get a score of 8. Now take this sentence:

Construction was a weak spot with Denmark’s Rockwool sinking 13% after full-year earnings missed expectations, and Sweden’s Skanska losing 7.8% after it cut its dividend and lagged profit estimates.

3. Modelling Document Topics

To be successful, NLP systems in finance often need to automatically extract a document's topic structure. Consider this snippet from a news article:

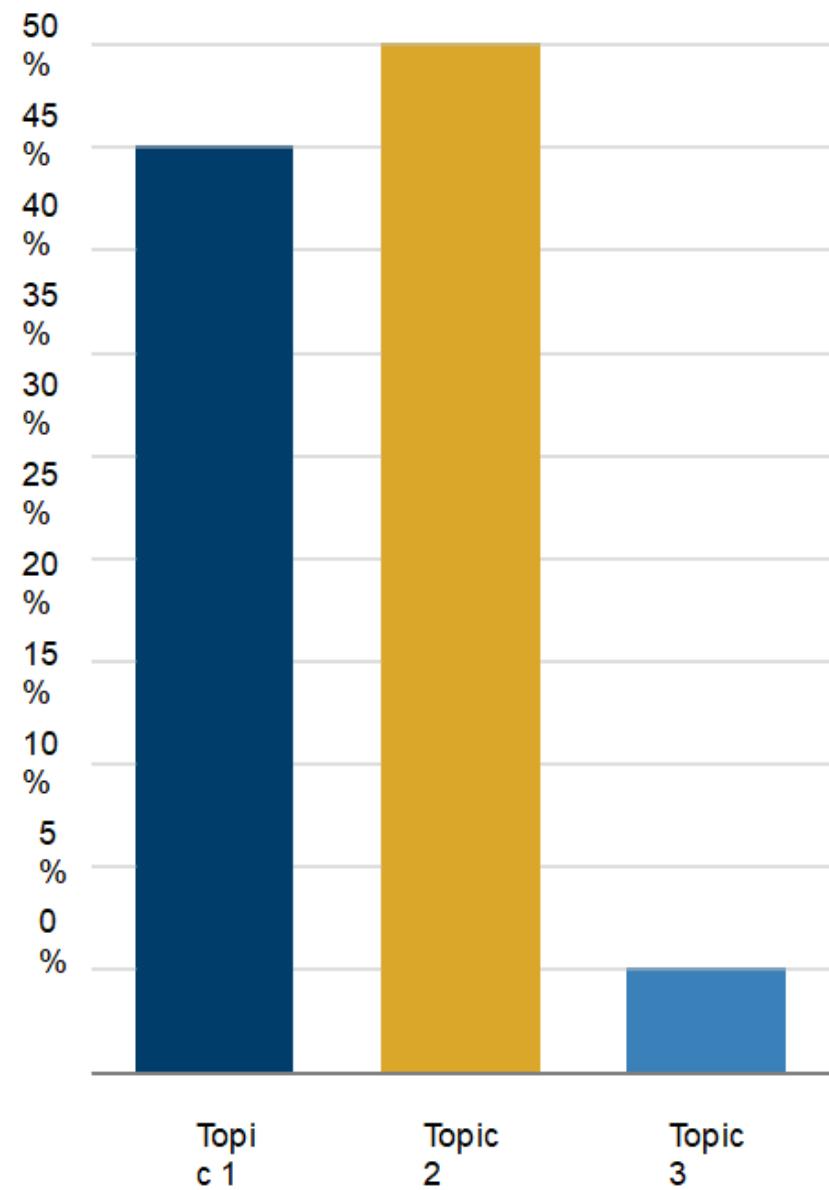
“Oil prices fell on Monday after climbing to their highest this year earlier in the session as China reported automobile sales in January fell for a seventh month, raising concerns about fuel demand in the world’s second-largest oil user.”

Often, the important information in a document is not just the tone, but its focus. In this example, there are two key topics: the first is oil, with words such as “oil”, “prices”, “fuel”, “fell” and “climb”; the second is the global economy, with words such as “world”, “China”, “demand” and “sales”. Understanding the topic structure of a document helps identifying events, informs the correct attribution of sentiment and allows to assess document similarity on a semantic level.

Figure 3. An Example of Machine Learning Models Inferring Topic Structure From a Document

Oil prices fell on Monday after climbing to their highest this year earlier in the session as China reported automobile sales in January fell for a seventh month, raising concerns about fuel demand in the world's second-largest oil user

The above example also highlights another subtle, but important, aspect of quantifying text data: timeliness. Even if we correctly identified the document's topics, there are two timeframes mentioned: "fell on Monday" and "climbing to their highest this year earlier in the session". Clearly, these two moves were attributed to a single entity – oil prices – yet they have opposite directions. Correctly identifying the evolution of events can be a crucial task for computer algorithms.



Source: Man Group; for illustration purposes only. The model has determined that the sentence is 45% about Topic 1, 50% about Topic 2, and 5% about other topics. We have explicitly labelled Topic 1 as Oil and Topic 2 as the Global Economy based on the most probable words associated with each of the topics.

4. Detecting Subtle Change

This theme of subtlety is quite prevalent in NLP research. The information contained in text data is sometimes very obvious to the human eye (a new product launch in the news; lots of positive words by a company executive), but can just as often be buried. One example application of NLP is measuring textual change: comparing the same documents over time, and finding subtle differences.

For example, in IBM's 2016 annual report, the company had a snippet related to its brand risks under a risk factor called "Failure of Innovation Initiatives". In the following year's annual report, IBM decided to extract it as a separate risk factor called "Damage to IBM's Reputation", and explicitly listed eight broad categories of example sources of reputation risk.

Comparing IBM's Annual Reports

IBM's 2016 annual report (left column)

IBM's 2017 annual report (right column)

Such subtle changes can be tricky and painstaking for a human to identify, especially given the typical length of annual reports and an investible universe of thousands of companies. Yet, to a machine, these changes are obvious: an algorithm can automatically scan through millions of documents and identify the added, deleted, or modified risk factors, classify them according to their topic, and even check which other companies have modified their risk factors in similar ways.

Another example is the transcripts from the Federal Open Market Committee ('FOMC') on US interest rate policy, where the market typically reacts not to the current transcript, but rather to slight changes in wording between the current and previous ones.

[...] IBM has one of the strongest brand names in the world, and its brand and overall reputation could be negatively impacted by many factors, including if the company does not continue to be recognized for its industry-leading technology and solutions and as a cognitive leader. If the company's brand image is tamished by negative perceptions, its ability to attract and retain customers could be impacted.

Damage to IBM's Reputation Could Impact the Company's Business: IBM has one of the strongest brand names in the world, and its brand and overall reputation could be negatively impacted by many factors, including if the company does not continue to be recognized for its industry-leading technology and solutions and as a

cognitive leader. IBM's reputation is potentially susceptible to damage by events such as significant disputes with clients, product defects, internal control

deficiencies, delivery failures, cybersecurity incidents, government investigations or legal proceedings or actions of current or former clients, directors, employees, competitors,

vendors, alliance partners or joint venture partners. If the company's

brand image is tamished by negative perceptions, its ability to attract and retain customers could be impacted.

5. Working Across Multiple Languages

All of the above examples are in English. While documents in English are convenient to consider because there is a vast amount of academic research in the area, it clearly isn't the case that all market-moving information originates in English. Consider the volume of important documents in any other language – be it Chinese or Russian, Japanese or Portuguese. To be able to leverage text from different languages and sources, one has to either develop models specific to that language (e.g. a Chinese sentiment model), or translate documents into English and then apply an English model.

Indeed, both applications are currently a heavy focus of NLP.

6. Going Beyond Written Text

Finally, all examples so far assume that the text we are interested in already exists in written form. That is not always the case. For example, every quarter, many global public companies host earnings conference calls – the timeliest source of financial results.⁵ Techniques from speech recognition research can be used to automatically transcribe documents as the call is progressing, or even analyse the subtle nuances of management tone to measure emotions.⁶

All of the above – and many others – are central research topics within NLP. Indeed, various methods – many of which may provide potential sources of quant alpha – have been developed over the last seven decades. In addition, these methods can be used to quickly uncover emerging risk factors. The ball doesn't just stop there, however. The same ideas can be used to scan for investment ideas by discretionary managers or give early warnings of key developments to companies in their investment portfolio.

4. Challenges When Using NLP

Working with language can inherently be more difficult than working with well-structured numerical data.

4.1. Scale and Technology

The first obvious challenge is scale. Unlike many numerical datasets, text data can be very large and thus requires significant investments in data storage and computation capacities. To be effective, large-scale distributed computation resources (hardware) are typically required, along with enough storage for all raw and intermediate data (with data stored efficiently) so that ideas can be iterated quickly. In addition, specialist software may need to be developed, to help visualise the complexities of the NLP research stages and aid research.

Sentiment Analysis

As an illustration of how we may think about applying NLP to finance, consider the following intuitive alpha idea: when people talk about companies using more positive language, these companies may be more likely to generate higher stock-market returns than companies with more negative language. People in this setting will be kept deliberately generic: these may be journalists, company executives, analysts, individual investors, clients, or political figures.

To test this idea, we would first need to collect a large dataset of documents associated with our tradable universe of stocks. These may be news, annual reports, earnings calls, blog posts, product reviews, social media posts, etc. We would then need to analyse all text documents related to each company and rank all stocks based on how positive its associated documents are. Finally, we are going to test going long the companies with the most positive language, and short the companies with the most negative language, continually rebalancing our portfolio as new documents appear.

Traditional Sentiment Model : Bag of Words +Word counting

To begin with, let's look at the following two news headlines:

Disney CEO is upbeat about deal with Fox.²²

Germany's tightening labor market might spell more trade trouble.

As humans, why do we think the first one is positive, and the second one is negative? At a basic level, it is because of words such as “upbeat” and “trouble”. Throughout our lives, education and work, we have assembled very large internal databases of word meaning, and when we see a sentence we apply these instantaneously and automatically.

Our first sentiment model is then simply going to replicate this human intuition at scale: count positive and negative words. What if we had a long list of all the positive and negative words that could appear in our documents? We could then simply get a computer to count these words and compute an aggregate score:

Traditional Sentiment Model : Bag of Words +Word counting

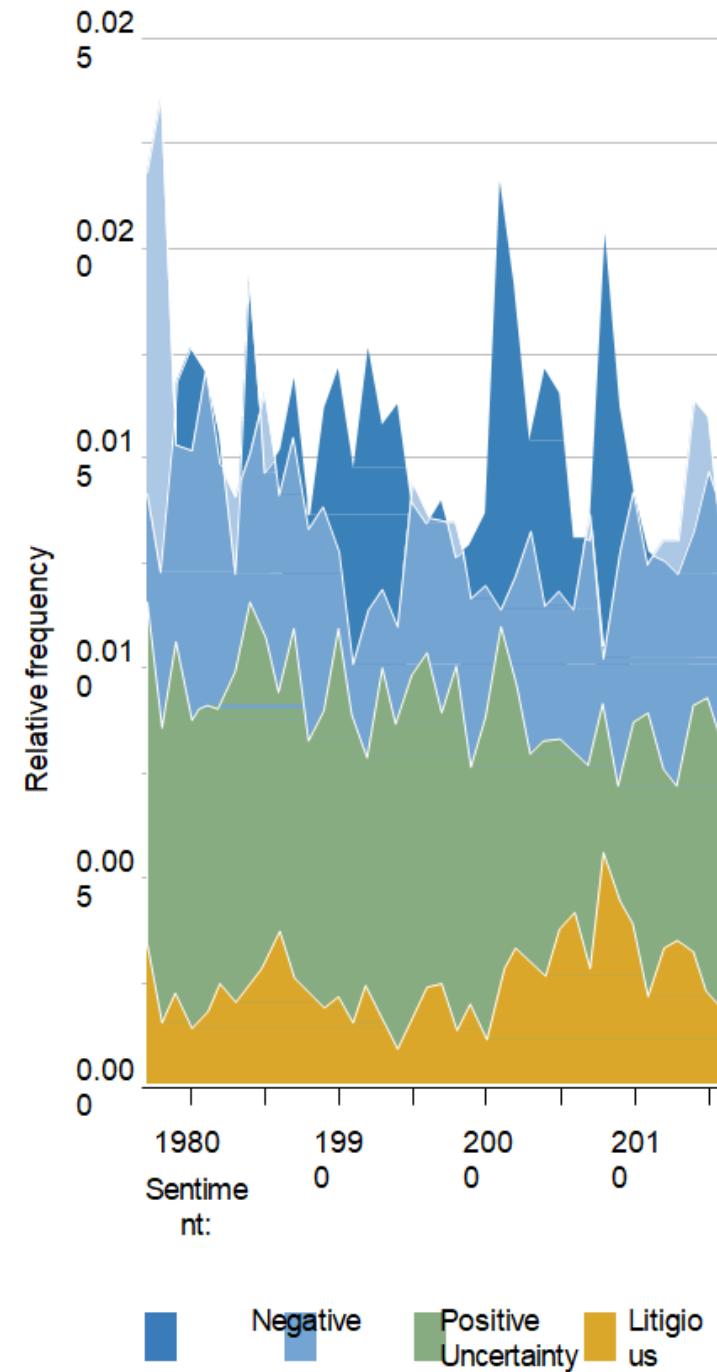
$$\text{sentiment} = \frac{\text{num}_{\text{positive}} - \text{num}_{\text{negative}}}{\text{num}_{\text{total}}}$$

That is, we simply subtracted the number of negative words from the number of positive words, and normalised this score by the total number of words in a document.

In fact, this is exactly the approach taken by one of the first highly influential academic papers in the fields of text analysis in finance. In 2007, Professor Paul Tetlock showed that counting negative words in a particular column of the Wall Street Journal had predictive power over the future price moves of the Dow Jones Industrial Average Index and the daily volume traded on the New York Stock Exchange.

Finance-Specific Dictionaries

Figure 7. Sentiment Analysis of Warren Buffett's Shareholder Letters (Using the Loughran-McDonald Lexicon)



Source: <https://juliasilge.com/blog/tidytext-0-1-3/>; as of 18 June, 2017.

Challenges with the Traditional Models

To see why, let's look at the following sentence:

Our Board of Directors has approved a new \$100 billion share repurchase authorization as well as a 16% increase in our quarterly dividend.²⁶

If you run this – and many other similar sentences – through a computer program that counts words from the LM dictionary, you will get a sentiment score of exactly zero. This is unfortunate, because we, as humans, see sentences like these as highly positive. Is that an isolated example or a more fundamental flaw? Is it simply the case the LM word lists have a few missing words we can add and move on?

If we look at why we as humans think the above sentence is positive, we quickly notice that none of the words above are positive or negative in isolation. A “dividend” or an “increase” are on their own neutral; it is the combination “increase in dividend” that makes us think the sentence is positive.

Consequentially, even if we had the most comprehensive and accurate word lists in our domain, any word-counting method will be unable to capture such sentences. To solve this problem, we leave the world of academic finance and enter the world of machine learning, where sentiment classification has been studied for a long time, and where there have been some exciting recent advancements.²⁸

Deep Learning and Targeted Sentiment

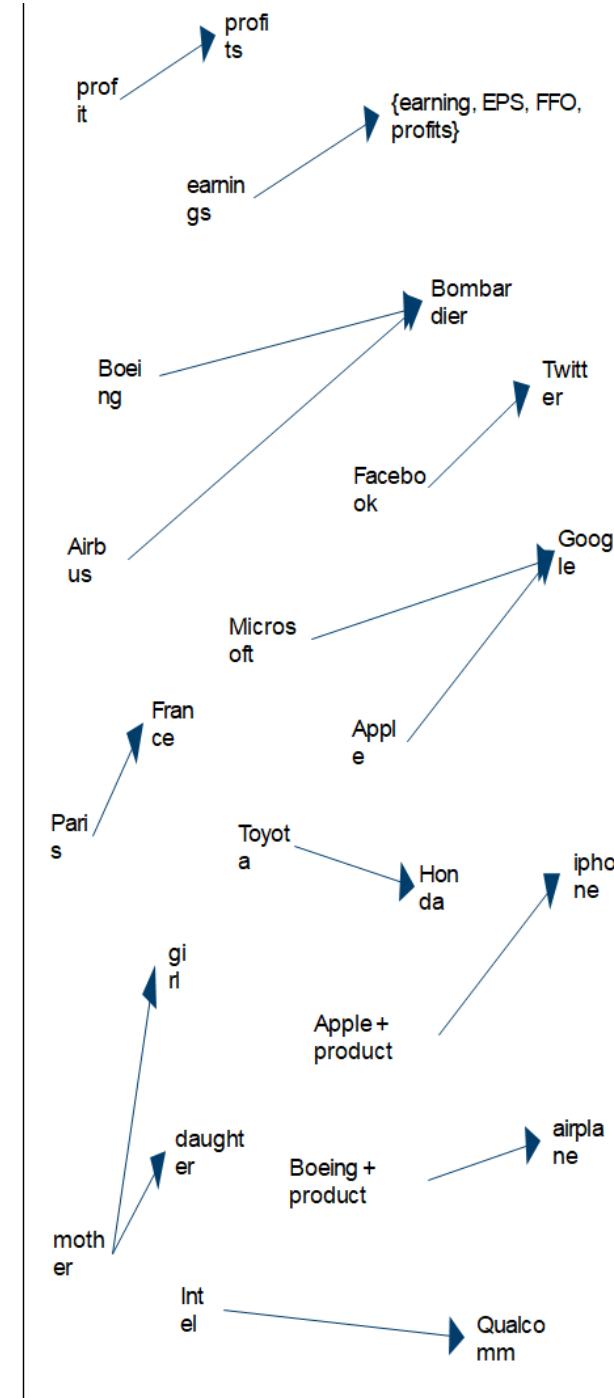
To tackle sentences like the one above, ‘targeted sentiment’ methods are used, i.e. given a target (like “dividend”), models are built that will tell us whether a sentence is positive or negative with respect to this target. In the example above, our model should tell us that the sentence is positive with respect to both “repurchase” and “dividend”.

To do this, three problems have to be solved: how to compile the list of interesting targets, how to find all words in a sentence related to a target, and how to come up with the words that are positive or negative given a target. It turns out that recent advances in machine learning are very helpful in solving these problems.

The first and last tasks – coming up with lists of targets of interest, and positive/negative word lists for each target – look remarkably similar to what Loughran and McDonald did in their 2011 work. In their case, their research group manually and painstakingly went through tens of thousands of words, reviewing each one manually and deciding whether each word was positive, negative or neutral. While that can be done, it would take vast amounts of time. Instead, a recent technique in machine learning called word embeddings can be used to automatically generate similar words given a set of seed words.

Word Vectors – Word2Vec

One of the most popular word embedding models is called word2vec because it can learn to convert any given word to a sequence of numbers. To do that, word2vec – a simple neural network model published in 2013 by a research group at Google – plays a game with itself. It goes through a body of text (e.g. all financial news) looking at all possible sequences of, say, seven words. It then ‘hides’ the middle word from itself, and tries to guess what it is based on its context (or vice versa – tries to predict the context from the middle word). By doing this painstakingly hundreds of millions of times (computers are good at this: they don’t get bored easily), it ends up implicitly learning word similarity. This is because words that appear in similar contexts end up represented as similar vectors of numbers. For example, “EBIT” is the most similar to “earnings” and “EBITDA” as all of these get used in fairly similar contexts.



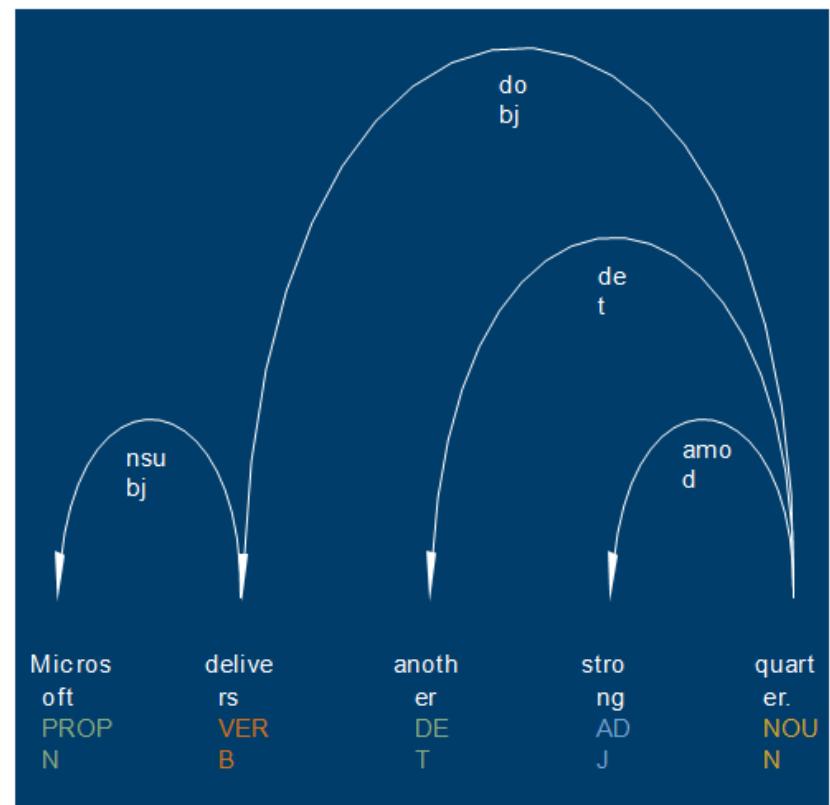
Dependency Parsing and POS Tagging

Given a sentence, a dependency parser would automatically identify the relationships between the words.

In Figure 9, the dependency structure is illustrated by the arrows. They indicate that “quarter” is the direct object of the verb “delivers”, and that “Microsoft” is its subject. They tell us that “strong” is an adjective modifying the noun “quarter”. The colourful words in uppercase are known as part of speech tags; these speech tags show that “delivers” is a verb, “strong” is an adjective and “quarter” is a noun.

Note that the annotations in the above figure were not generated by a human – they were generated by a neural network. These models are nowadays trained on huge amounts of data and are surprisingly accurate.

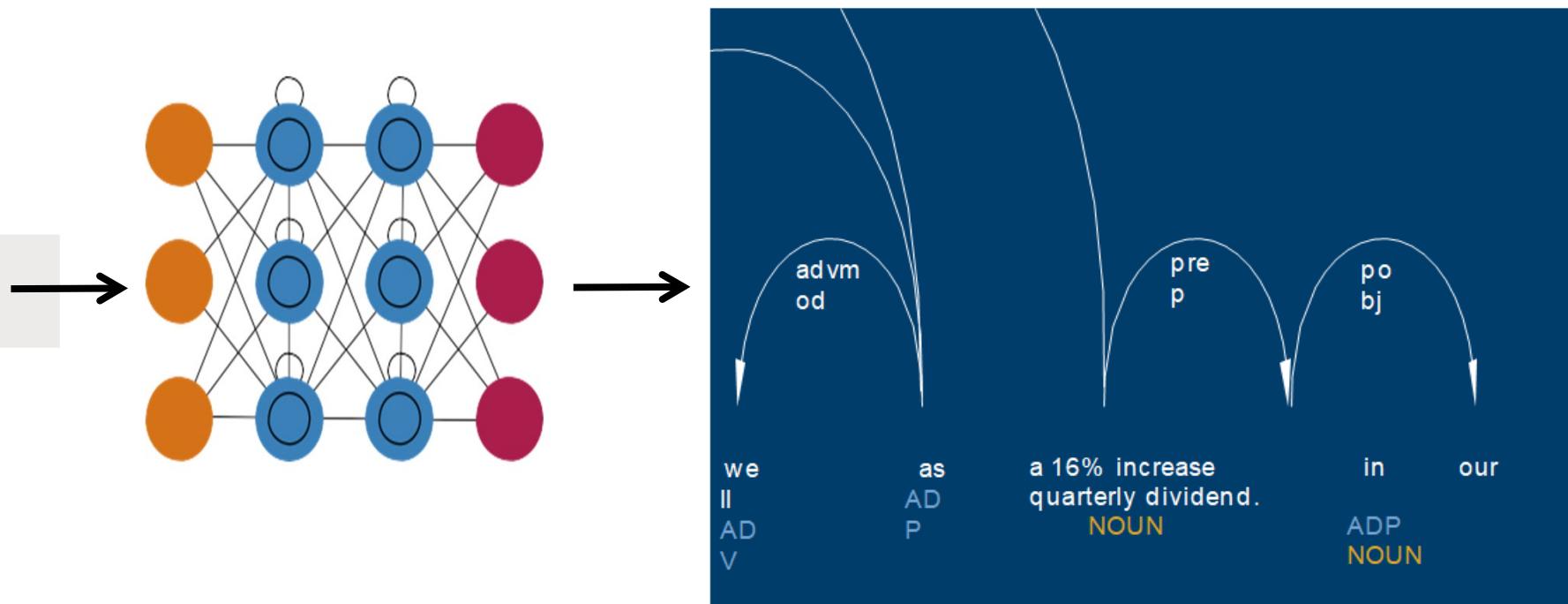
Figure 9. Dependency Parsing



Quote source: M.W. De Oliveira (2018/05/01). Microsoft Does It Again (SeekingAlpha). Image generated with spacy: <https://spacy.io/>

Dependency Parsing and POS Tagging

"Our Board of Directors has approved a new USD 100 billion share repurchase authorization as well as a 16% increase in our quarterly dividend."



Quote source: Apple second-quarter results - earnings call transcript. Image sources: Man Group (visualisation using <https://spacy.io/>) and <http://www.asimovinstitute.org/neural-network-zoo/>

Dependency Parsing and POS Tagging

All we need to do now to create our third sentiment model, one that can correctly capture targeted sentiment, is to put all these bits together.

We start with a set of seed targets (“EBITDA”, “repurchase”, “dividend”) and use word embeddings to generate expanded lists of targets of interest. We then scan each sentence and check if any of the targets of interest is in it. If so, we use a neural network to identify the dependency structure of the sentence and find all words related to our target.

In this neighborhood, we count the target-dependent positive or negative words (again, constructed by taking a set of seed sentiment words and expand them using our word embeddings).

Dependency Parsing and POS Tagging

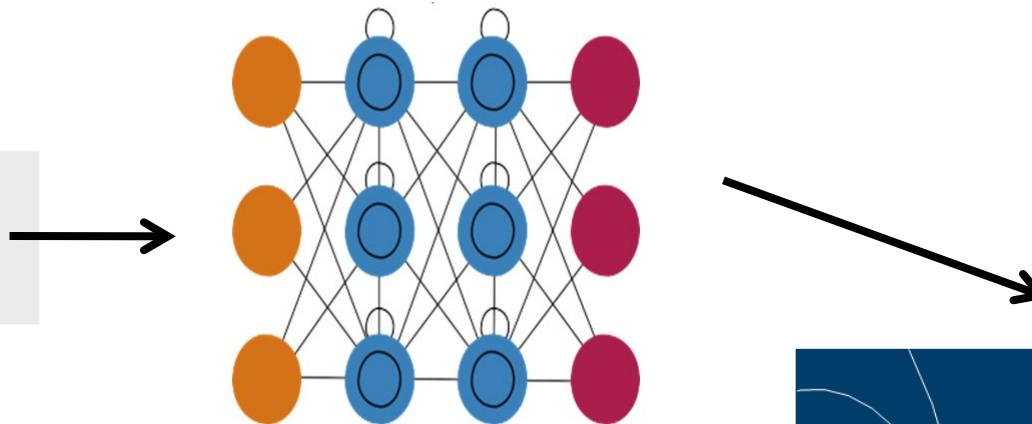
Our final sentiment score is going to be computed as before:

$$\text{sentiment} = \frac{\text{num positive} - \text{num negative}}{\text{num total}}$$

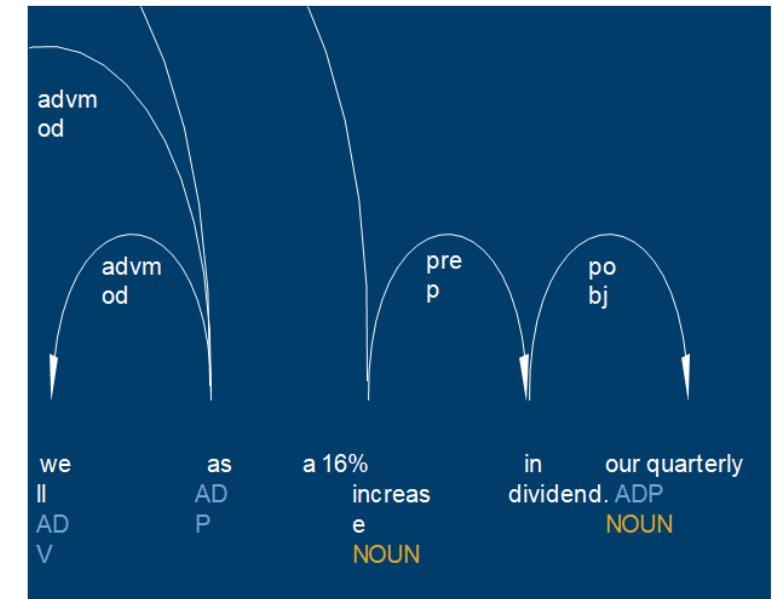
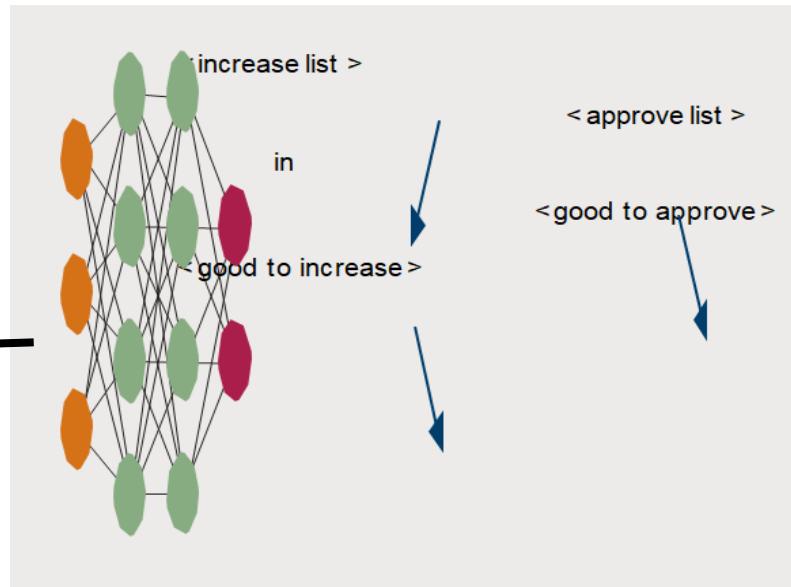
However, there is one minor modification: instead of simply counting positive and negative words, we count positive and negative targeted phrases.

Targeted Phrases

"Our Board of Directors has approved a new USD 100 billion share repurchase authorization as well as a 16% increase in our quarterly dividend."



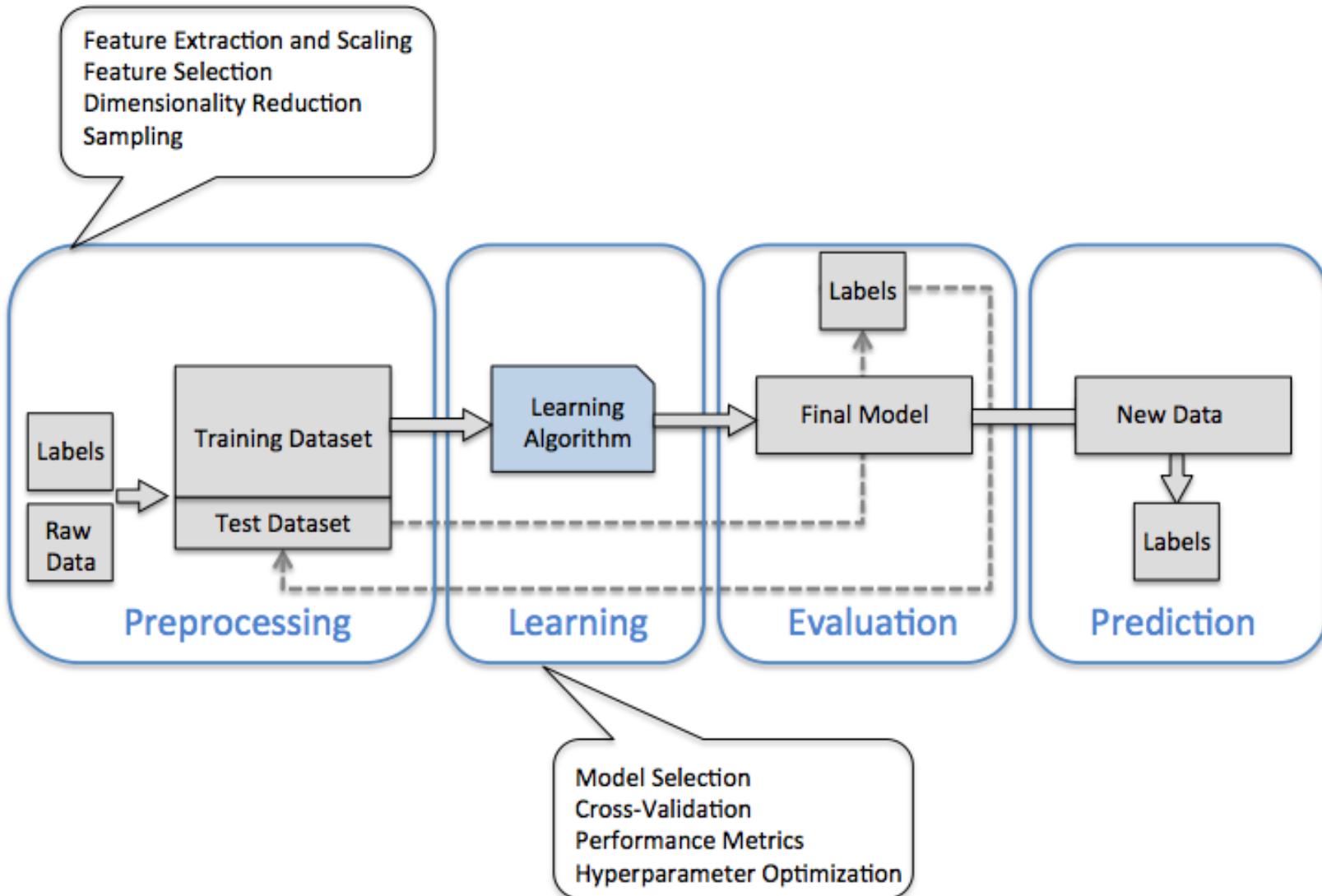
- 1) approved -> share repurchase authorization
- 2) increase in -> our quarterly dividend



Natural Language Processing in Finance

Essentials

Machine Learning Modeling

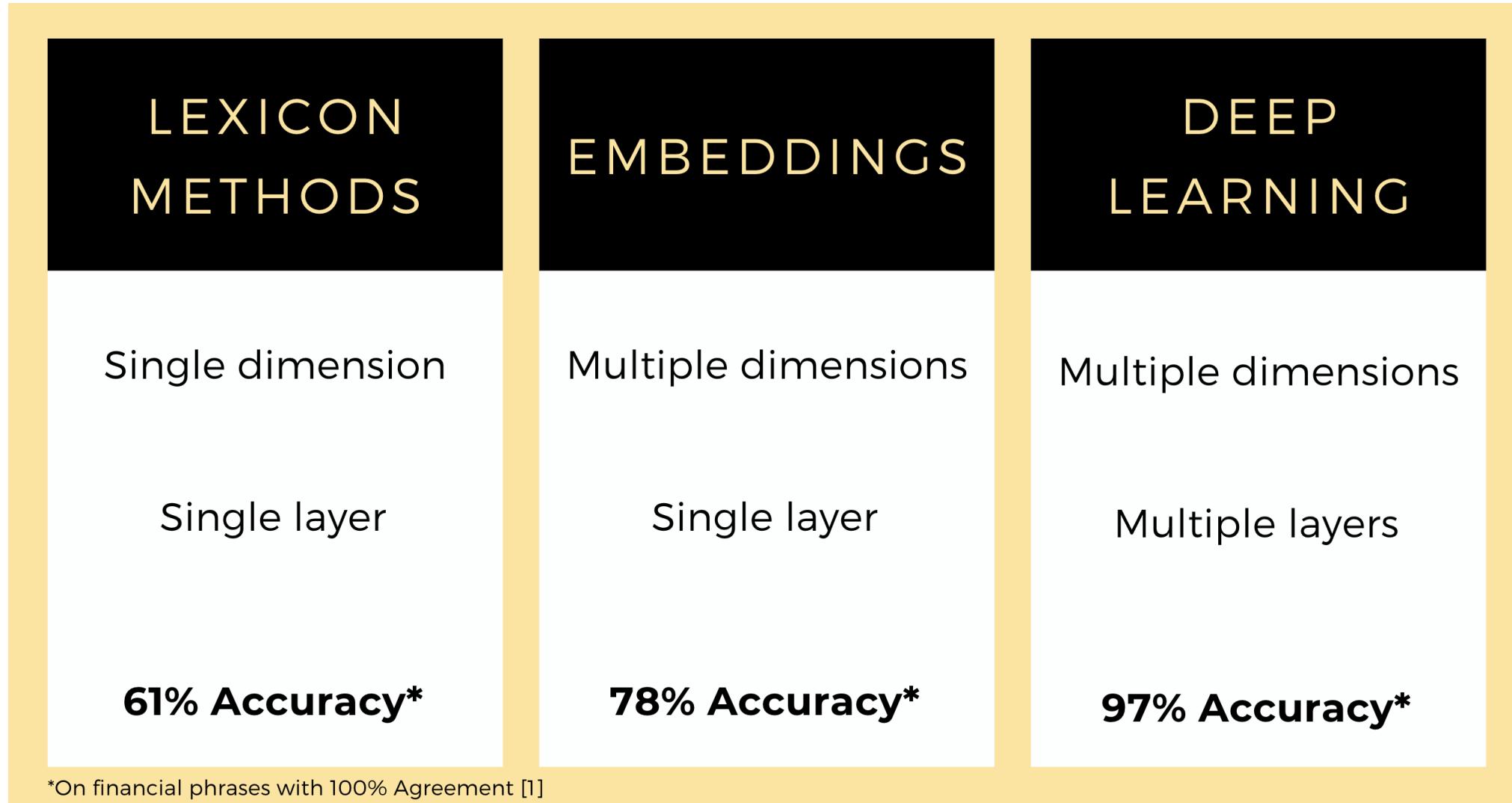


WORKFLOW SUMMARY OUTLINE



Historical Evolution Sentiment Analysis

[Financial Phrase Bank](#)[



Natural Language Processing – a Classical Approach

- NLP pipeline
 - Classical Approach
 - Modern Approach via Neural Networks (Deep learning)
- Text-preprocessing
- Words Tokenization
- Words Feature Engineering
 - Binary Based Representation
 - Count Based Representation
 - Term Frequency–inverse Document Frequency (TF-IDF)
- Machine Learning Modeling: shallow vs deep learning

NLP pipeline – Modern Approach

It differs from the classical approach from the “feature engineering” and modeling steps.

By replacing the “count” based methods and using the **word2vec**, the modern approach use the neural networks to find and build features for us.

So why do we learn the “unsex” traditional approach?

- Amount of data available for training. For example, a simple RNN model’s parameters could go beyond 100k.
- Requirements of the problem, e.g., computing time, interpretabilities, and so on

NLP pipeline – Text preprocessing

Definitions

Tokens: words or entities (e.g., punctuations, numbers, URLs, etc.) present in a text.

Tokenization: the process of converting a text (a list of strings) into tokens.

Why text preprocessing?

Reduce the total number of vocabularies and make the tokenization steps easier.

- Converting all letters to lower or upper case
- Converting numbers into words or removing numbers
- Removing special characters
- Removing punctuations, accent marks and other diacritics
- Removing stop words
- Removing Entity (person or organization) Names
- Correcting misspellings or abbreviations – not so necessary

NLP pipeline – Text preprocessing Con't

Summary of text preprocessing:

- **Noise removal** is cleaning of textual data by stripping away anything that is not relevant to the task at hand including but not limited to the removal of acronyms, punctuations and numbers
- **Lexicon normalization** is the method of standardizing multiple representations that are exhibited by a single word (e.g., different inflections of a word: e.g., play, plays, played, etc.).
- **Object standardization** is the process of converting shorthand forms or variant spellings to the formal spelling (e.g., luv to love).

NLP pipeline – Words Tokenization

It is a process of breaking the preprocessed strings into N-grams, where N can be 1, 2 or more.

E.g., today can not be better .

1-gram: today, can, not, be, better

2-gram: today can, can not, not be, be better

3-gram: today can not, can not be, not be better

4-gram: today can not be; can not be better

5-gram: today can not be better

In classical NLP, we can use the combinations of N-grams to fit the models, usually with 1 - and 2 – grams.

NLP pipeline - Modeling

Finishing the word representations of each document, in order to do supervised machine learning, we need to define the target variable.

For sentiment analysis, the usual target will be positive (1), negative (-1) and neutral (0)

Doc Id	increase	decrease	great	weather	price	fruit	rice		Target
1	4	0	5	0	4	0	0	...	1
2	1	1	3	4	3	0	2	...	0
3	0	7	0	3	4	2	0	...	-1

But the article did not come with the target/label. So how do we label it? Manually.

Is it necessary? Fortunately, maybe not.

Now it is time to load your favorite machine learning models to train the data.

NLP pipeline – Words Feature Engineering

Binary representation is a way of turning the large amount of text for each record into useful features.

We can create a matrix that uses each word as a feature and keeps track of whether or not a word appears in a document/record. You can do this in sklearn with a *CountVectorizer()* method and setting *binary* parameter to be true

NLP pipeline – Words Feature Engineering

Count representation is very similar to binary representation, but rather than with 0 and 1 values, it counts the actual frequencies of each word in a document/record.

We can do this in sklearn with a *CountVectorizer()* method and setting *binary* parameter to be False.

However, this will cause some problems sometimes.

- The frequent words are not necessarily meaningful in NLP analysis, e.g., “very”, “much”.
- In addition, longer documents will have higher average count values than shorter documents, though the two documents might belong to same category

NLP pipeline – Words Feature Engineering

Term Frequency–inverse Document Frequency (TF-IDF) representation is the solution to the problems arisen from count based representation. Rare terms are more *informative* than frequent terms.

$$\text{TF - IDF} = f_{t,d} * \text{idf}(t, d)$$

In simplest form, term frequency is to use the raw count of a term in a document $f_{t,d}$, where t stands for frequency and d for documents.

However, this again is biased for long documents.

Alternatively, we could have:

- $f_{t,d} / (\text{number of words in } d)$ – not implemented in sklearn
- $1 + \log(f_{t,d})$ – just set the **sublinear_tf = True** in this method

NLP pipeline – Words Feature Engineering

Inverse Document Frequency is a measure of how much information the word provides, defined by:

$$idf(t, d) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

, where N is total number of documents in the data and $|\{d \in D : t \in d\}|$ is the number of documents where the t term appears. To ensure the denominator not being 0, we can set **smooth_idf = True** in the method.

NLP pipeline – Words Feature Engineering

Therefore, the TF – IDF is defined as:

$$\text{TF - IDF} = f_{t,d} * \text{idf}(t, d)$$

Warning: there might be data leakage among training and testing data.

We can realize this word representation in sklearn with a *TfidfVectorizer()* method.

Consider a document containing 100 words wherein the word *finance* appears 3 times.

The term frequency (i.e., tf) for *finance* is then $(3 / 100) = 0.03$. Now, assume we have 10 million documents and the word *finance* appears in one thousand of these.

Then, the inverse document frequency (i.e., idf) is calculated as $\log(10,000,000 / 1,000) = 4$.

Thus, the Tf-idf weight is the product of these quantities: $0.03 * 4 = 0.12$.

Natural Language Processing in Finance

Traditional Methods

NLP Sentiment Analysis – Case Study S&P Global Research

Details on Loughran and McDonald (2011) Financial Dictionary

There are many ways to define sentiment. We use a bag-of-words approach where the sentiment word lists are from the Loughran and McDonald (2011) financial dictionary. Their dictionary has become the de facto financial dictionary for NLP analysis due to its accessibility, its comprehensiveness, its financial-specific context, its lack of dependency on the transitory nature of its words and, lastly and perhaps most importantly, its unambiguous and singularly connotated words. Details below.

Accessibility – their word lists are readily accessible because they are freely posted online.

Comprehensiveness – the dictionary is comprehensive such that it is difficult for managers to game the system (i.e., circumvent certain words that have empirically been shown to lead to future stock underperformance) because they start with every conceivable English word with all inflections of a word, totaling 80,000+ distinct words in the master word list.

NLP Sentiment Analysis – Case Study S&P Global Research

Financial-specific context - they filter their initial master word list down to their sentiment word lists by examining 10-K filings between 1994 and 2008 inclusively.

Permanence of words - their master and sentiment word lists are less transitory because they start with the most comprehensive list of English words possible and, more importantly, the master word list doesn't rely on transitory terms such as iphone.

Unambiguous and Singularly Connotated Words - they arrive at their sentiment word lists containing unambiguous and singularly connotated words by looking at the most frequently occurring words in the 10-Ks from the master word list. From there, they went word-by-word and assessed each of the word's meaning in a business context. At the end of their process, the words that ended up in their word lists are less ambiguous in their meaning with singular connotation.

NLP Sentiment Analysis – Case Study S&P Global Research

Their three most important lists of words for our use cases are the master word list, positive and negative sentiment word lists with distinct word counts of 80,000+, 350+ and 2300+, respectively. Examples of positive words are able, abundance, acclaimed, accomplish and so forth. Examples of negative words are abandon, abdicate, aberrant, abetting and so forth.

NLP Sentiment Analysis – S&P Global Research

Sentiment Change vs. Forward Returns

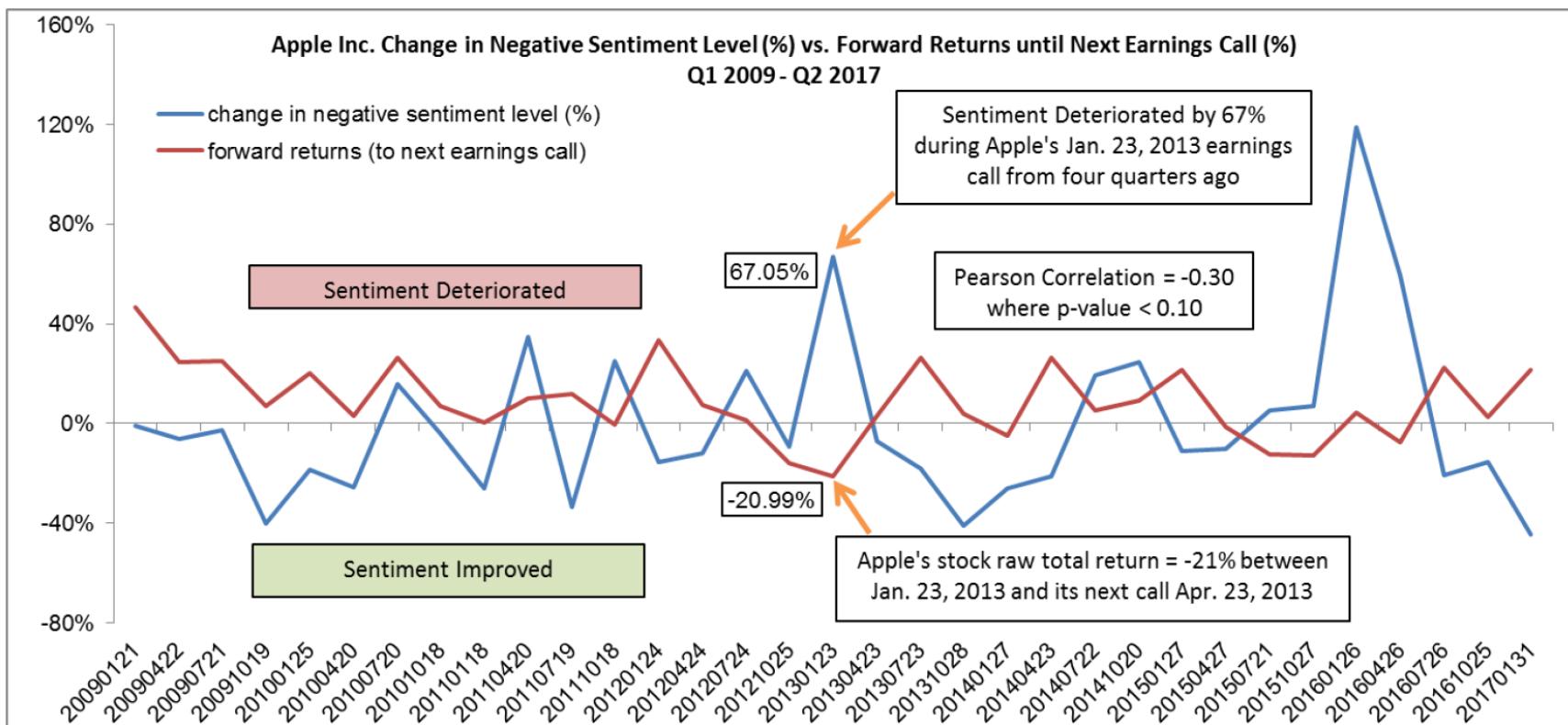
The first use case is to capture the historical relationship of Apple Inc.'s sentiment level changes from its earnings calls and its forward returns until the next call (Exhibit next slide). The changes in sentiment are defined as quarter-over-quarter (QoQ) changes from four quarters ago (to account for seasonality).

The sentiment of each of Apple's earnings calls is defined by the proportion of negative words in its earnings call transcript where the classification of both the negative and the master word list is based on the Loughran and McDonald (2011) financial dictionary.

Because sentiment in this use case is measured with negative words, positive (negative) changes reflect sentiment deterioration (improvement). Apple's forward returns until its future calls have been shifted back a quarter such that its sentiment changes and its forward returns are aligned vertically in the Exhibit. One promising observation is that the Pearson correlation is about -0.30 since Q1 2009, which suggests that Apple's forward returns historically go down when its sentiment deteriorates.

NLP Sentiment Analysis – S&P Global Research

Sentiment Change vs. Forward Returns Con't



Note: Sentiment is defined as the proportion of negative words in an earnings call using [Lougrahan and McDonald](#)

NLP Sentiment Analysis – S&P Global Research

S&P 500 Trends in Sentiment Change

The second use case provides a heat map showing sentiment trends for S&P 500 GICS industry groups. Exhibit next slide shows quarterly sentiment changes for 24 GICS industry groups by calendar quarter between Q3 2016 and Q2 2017 inclusively.

Sentiment is defined as the proportion of negative words in an earnings call using Lougrahan and McDonald (2011). Sentiment changes are measured quarter-over-quarter from four quarters ago where the values are multiplied by -1 to make results easier to interpret. Industry group level values are rolled up equal-weighted from the stock-level. Source: Data as of 08/08/2017.

NLP Sentiment Analysis – S&P Global Research

S&P 500 Trends in Sentiment Change

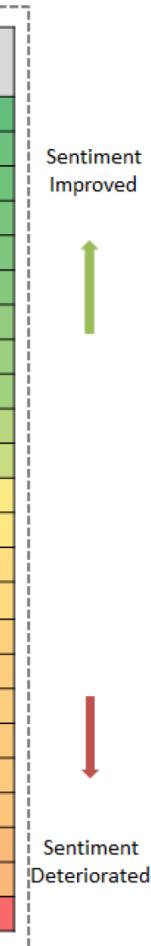
The industry groups are sorted by their sentiment changes from Q2 2017 in descending order. Similar to the Apple's example from above, all the sentiment values are QoQ changes from four quarters ago where each industry group's sentiment is aggregated, on an equal-weighted basis, from the stock-level where the sentiment is measured using the proportion of negative words in a stock's earnings call. In order to make the interpretation more intuitive, we multiplied the values by negative one so green (red) values denote improvement (deterioration) in sentiment for the industry groups and the different color shades reflect the magnitude of sentiment changes.

One could easily visualize sentiment trends for an industry group and spot potential inflection points and accelerations. For example, the sentiment improved substantially for banks between calendar quarter Q4 2016 and Q1 2017. Investors would notice this as an inflection point to the upside and could potentially use the insight as an additional piece of information in their investment decision making process.

NLP Sentiment Analysis – S&P Global Research

S&P 500 Trends in Sentiment Change

GICS Industry Groups	Calendar Quarters			
	Q3 2016	Q4 2016	Q1 2017	Q2 2017
Consumer Services	-4.1%	-4.0%	16.9%	20.1%
Software & Services	-0.2%	4.1%	-3.9%	19.2%
Transportation	-3.6%	-2.8%	14.6%	19.1%
Diversified Financials	5.4%	13.8%	16.5%	18.0%
Technology Hardware & Equipment	4.8%	7.1%	18.1%	17.5%
Banks	-3.7%	-0.4%	18.1%	16.8%
Capital Goods	-4.9%	9.1%	16.9%	15.5%
Commercial & Professional Services	7.6%	-4.4%	-6.2%	14.6%
Utilities	-4.6%	7.2%	7.0%	14.1%
Insurance	0.0%	14.4%	10.2%	11.9%
Energy	0.1%	13.6%	25.8%	10.3%
Real Estate	-11.5%	-2.4%	0.5%	5.5%
Media	-11.6%	-12.6%	5.0%	4.2%
Materials	5.7%	-1.8%	12.2%	1.4%
Semiconductors & Semiconductor Equipment	6.3%	-3.7%	14.5%	0.5%
Retailing	-18.4%	-0.3%	2.4%	-1.6%
Consumer Durables & Apparel	10.7%	0.2%	-4.7%	-2.6%
Pharmaceuticals, Biotechnology & Life Sciences	-5.7%	-3.6%	-1.3%	-2.9%
Household & Personal Products	-3.3%	-2.9%	-7.8%	-3.9%
Food & Staples Retailing	17.7%	-0.8%	-16.5%	-4.0%
Food, Beverage & Tobacco	2.2%	0.4%	-3.6%	-5.0%
Health Care Equipment & Services	-8.2%	-2.7%	-1.0%	-7.5%
Autos & Components	-14.6%	-4.3%	34.2%	-8.5%
Telecommunication Services	9.1%	-13.6%	-36.9%	-29.4%



Note: Sentiment is defined as the proportion of negative words in an earnings call using [Lougrahan and McDonald \(2011\)](#). Sentiment changes are measured quarter-over-quarter from four quarters ago where the values are multiplied by -1 to make results easier to interpret. Industry group level values are rolled up equal-weighted from the stock-level. Source: S&P Global Market Intelligence Quantamental Research. Data as of 08/08/2017.

NLP Sentiment Analysis – S&P Global Research

- **Sentiment-based signals:** Firms whose executives and analysts exhibited the highest positivity in sentiment² during earnings calls outperformed their counterparts by 4.14% annually with significance at the 1% level. Firms with the largest year-over-year positive sentiment change and firms with the strongest positive sentiment trend outperformed their respective counterparts by 3.07% and 3.96% annually with significance at the 1% level.
- **Behavioral-based signals:** Firms whose executives provided the most transparency by using the simplest language and by presenting results with numbers outperformed their respective counterparts by 2.11% and 4.43% annually with significance at the 1% level.
- **Sentiment- and behavioral-based signals are not subsumed by commonly used alpha and risk signals.** After adjustments³, the signals generated excess long-short returns ranging from 1.65% to 3.64% annually with significance at the 1% level (Exhibit 14). The sentiment- and behavioral-based signals had some of the highest information ratios among all considered strategies (Exhibit 4) and are lowly and negatively correlated with each other.

NLP Sentiment Analysis – S&P Global Research

- Positive language from the **unscripted responses by the executives during the Q&A** drove the overall predictability of the positive sentiment signal.
- **The sentiment of CEOs** has historically been more important than the sentiment of other executives (Exhibit 9). A strategy based on the sentiment of CEOs generated 3.63% per year on a long-short basis with significance at the 1% level.
- The aggregate sentiment of analysts historically enhanced the predictability of the 3-month FY1 EPS analyst revision signal. A strategy using the aggregate sentiment of analysts from earnings calls yielded 4.24% per year on a long-short basis with significance at the 1% level.

NLP Sentiment Analysis – S&P Global Research

- **Topic Identification** - Firms That Referenced the Most Positive Descriptors around Their Financials Outperformed Historically. Firms whose executives most frequently articulated references to growth- and expansion-related descriptors around I) revenue II) earnings or III) profitability topics outperformed their counterparts by 9.16%, 8.60% and 6.76% per year, respectively.
- **Transparency** - Firms That Provided Greater Call Transparency Exhibited by Executives' Behaviors and Decisions Outperformed Historically. Firms whose executives I) referenced the most instances of guidance or II) used the most similar language between calls outperformed their counterparts by 5.84% and 3.75% per year, respectively. Firms whose executives III) introduced numerical values earlier or IV) blamed exogenous elements the least outperformed their counterparts by 3.97% and 2.39% per year, respectively.
- **Weighted Average Sentiment** - Quantifying Call Sentiment Using A Weighted Average Construct Relative to A Simple Average Led to Better Returns and Less Volatility Historically. Firms whose executives used the most positive language in the most recent call that was scarcely used in the previous calls outperformed their counterparts by 5.18% per year. A weighted average sentiment, relative to a simple average, improved the annualized information ratio to 1.81 from 1.19 historically .
- **Additive Forecasting Power** - The Newly Introduced Signals Demonstrated Additive Forecasting Power above Commonly Used Alpha and Risk Signals Historically. The additive economic performance ranged from 2.02% to 6.69% per year after controlling for market, size, value, quality, momentum, earnings surprise and analyst revision.

Abstract



Environmental, Social and Governance (ESG) investing has been one of the most important topics in asset management this past decade. Yet, for all the attention, only a fraction of asset managers truly consider ESG issues when making investment decisions.

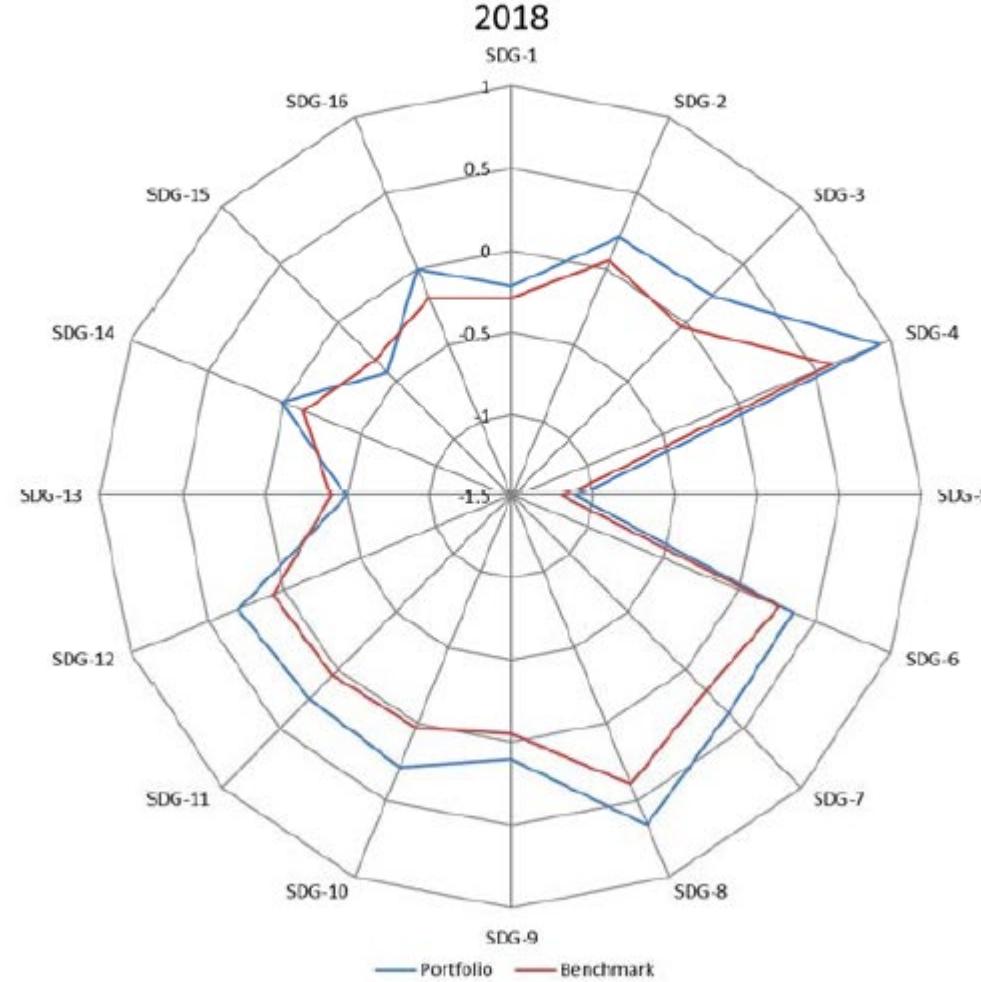
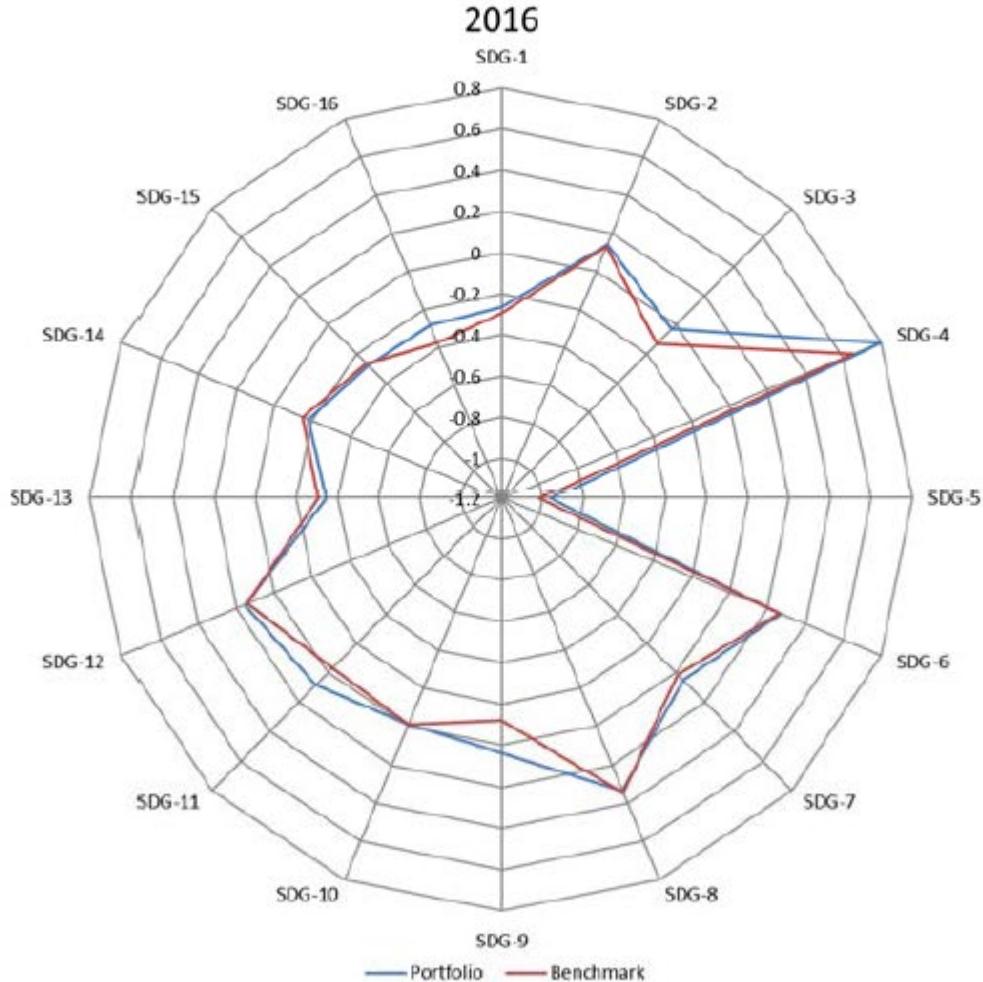
This is partly due to the perceived conflict of ESG investing with an asset manager's fiduciary duty and partly due to low-quality ESG data despite the near ubiquity of sustainability reports. We analyze the relationship between alpha generation and ESG metrics, and measure the impact companies have on the U.N.'s Sustainable Development Goals (SDG's). First, we construct a sector-neutral portfolio using MSCI ESG momentum scores from 2013 to 2018, and determine that it is feasible to generate positive alpha vis a vis the MSCI US index. Second, we utilize structured and unstructured data to determine a company's net influence on the SDGs, what we call its SDG 'footprint.'

We show that an ESG momentum portfolio both outperforms the MSCI US index and has a relatively better SDG footprint than that of the index. Third, we establish a positive contemporaneous connection between the portfolio's ESG ratings momentum and its SDG footprint. Thus, a positive linkage exists between ESG, alpha, and the SDG's.

See: Sustainable Investment – exploring the linkage between Alpha, ESG, and SDG's SSRN Working Paper
<http://ssrn.com/abstract=3623459>

SDG Footprints of Momentum Portfolio

Contemporaneous Raw SDG Scores



Lazy Prices: Cosine Similarity

- Paper Lazy Prices: Cosine Similarity. Short companies whose 10-Ks change drastically year-over-year, long companies with few changes

$$\text{Sim_Cosine} = \frac{D_1^{TF} \cdot D_2^{TF}}{\|D_1^{TF}\| \times \|D_2^{TF}\|}$$

D_A : We expect demand to increase.

D_B : We expect worldwide demand to increase.

$T(D_A, D_B) = [\text{we, expect, worldwide, demand, to, increase}]$

$D_A^{TF} = [1, 1, 0, 1, 1, 1]; D_B^{TF} = [1, 1, 1, 1, 1, 1]$

$$\begin{aligned}\text{Sim_Cosine} &= \frac{D_1^{TF} \cdot D_2^{TF}}{\|D_1^{TF}\| \times \|D_2^{TF}\|} \\ &= \frac{(1 \times 1 + 1 \times 1 + 0 \times 1 + 1 \times 1 + 1 \times 1 + 1 \times 1)}{(\sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2}) \times (\sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2})} \\ &= 0.91\end{aligned}$$

Step 1: Define our Texts

Step 2: Find all unique words in each text, then construct the union of these two sets

Step 3: Turn our texts into vectors

Step 4: Follow the formula to calculate cosine similarity!

Quick intro to cluster demeaning

- Since not all industries have the same business practices, the returns and signals for companies tend to group together
- Want to find these clusters and control for them via cluster demeaning
- Can do so using Item 1: Business by embedding these descriptions into a vector and calculating pairwise distances

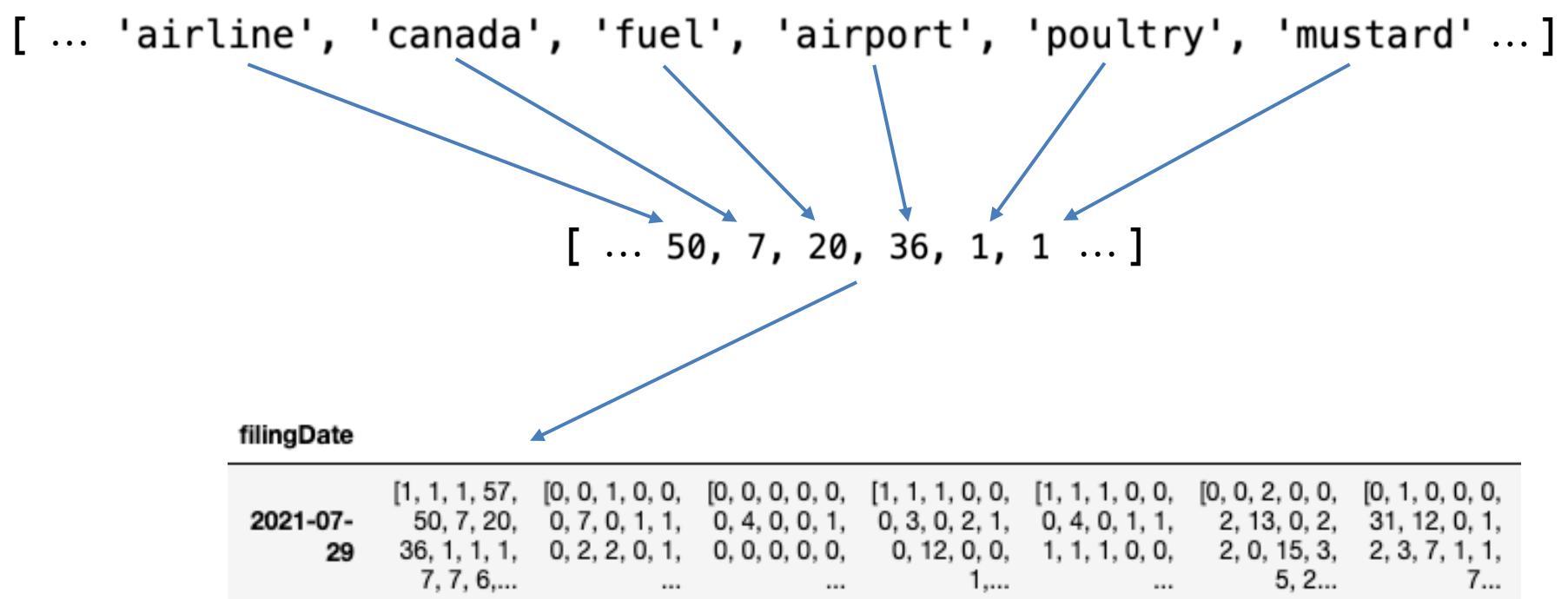


Quick intro to cluster demeaning

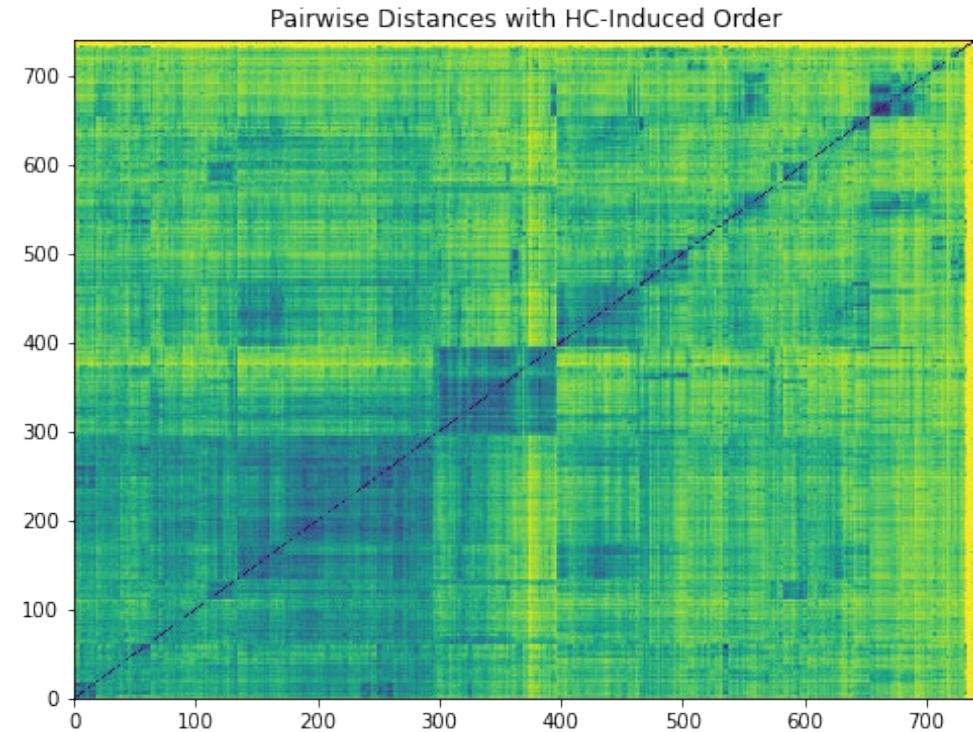
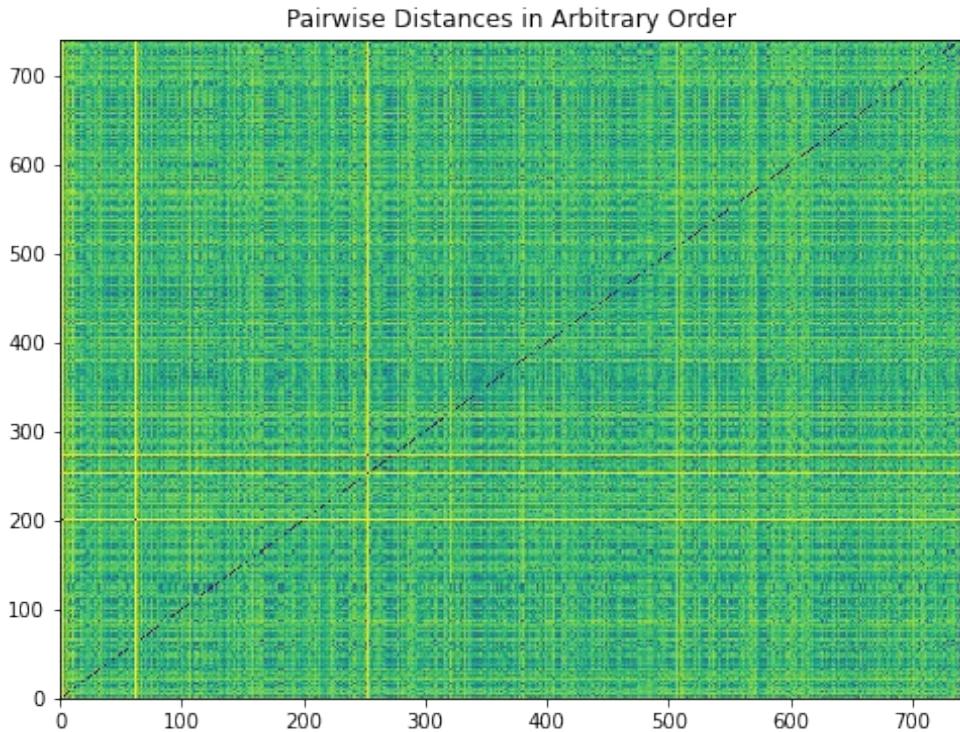
Superset of all words used (cleaned)

Create a vector containing the frequency of each word in superset within the business description

Do this for each business
description, for each
applicable date



Cluster demeaning



- By using hierarchical clustering, we can rearrange the order of companies to find some interesting groupings seen above, and remove their means from the signals

Cosine similarity signal construction

Date	Stock 1	Stock 2	Stock 3
Date 1	0.2	0.5	0.83
Date 2	0.35	0.42	0.90
Date 3	0.40	0.37	0.92

Step 1: Define your Score

$$p_i \sim \mathcal{U}[-1, 1]$$

$$\sum_i p_i = 0$$

Step 6: Enjoy the nice statistical properties of your portfolio weights!

Date	Stock 1	Stock 2	Stock 3
Date 1	1	2	3
Date 2	1	2	3
Date 3	2	1	3

Step 2: Rank your score along the rows

Date	Stock 1	Stock 2	Stock 3
Date 1	1/2	3/2	5/2
Date 2	1/2	3/2	5/2
Date 3	3/2	1/2	5/2

Step 3: Subtract 1/2

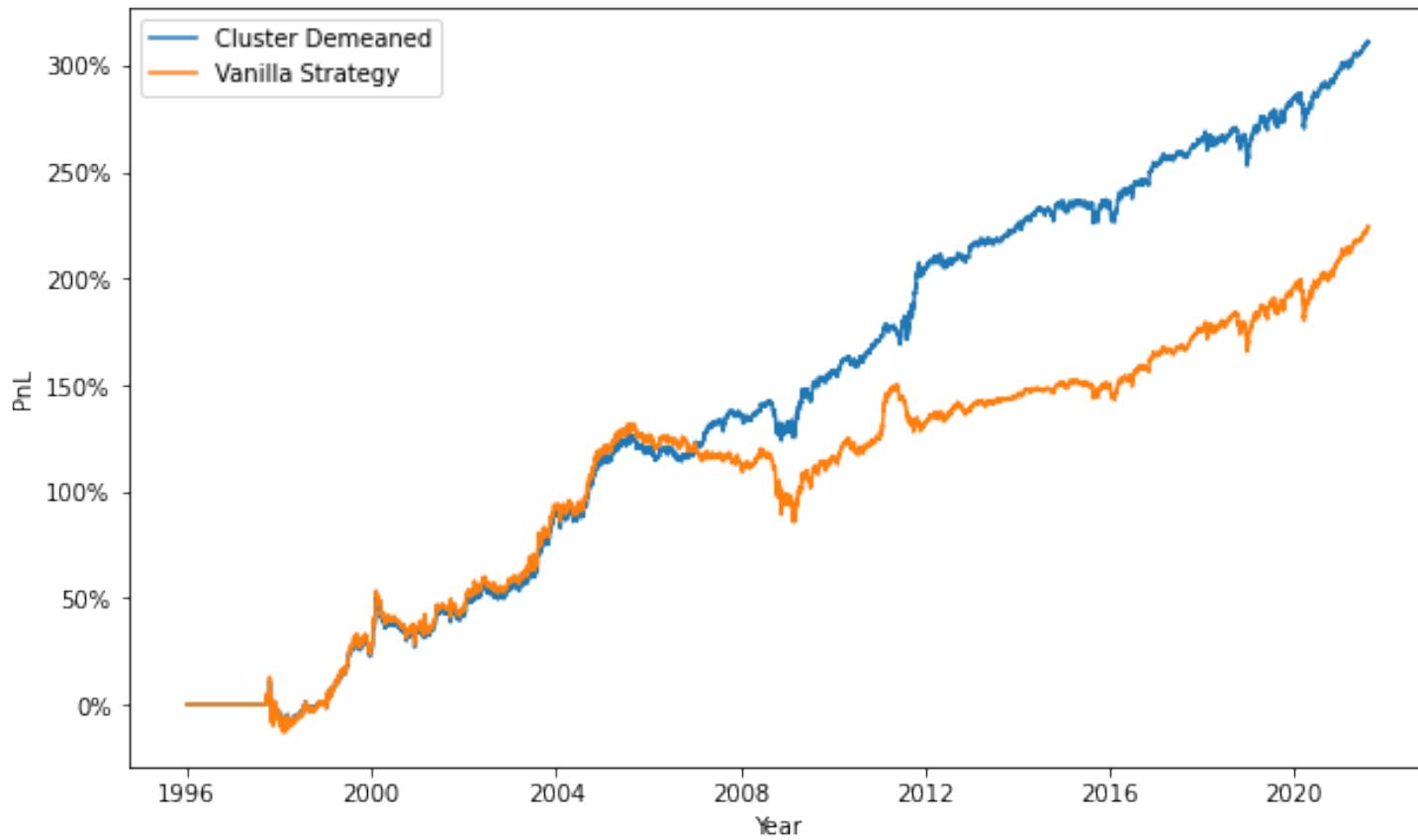
Date	Stock 1	Stock 2	Stock 3
Date 1	1/6	1/2	5/6
Date 2	1/6	1/2	5/6
Date 3	1/2	1/6	5/6

Step 4: Divide by number of scores in each row

Date	Stock 1	Stock 2	Stock 3
Date 1	-2/3	0	2/3
Date 2	-2/3	0	2/3
Date 3	0	-2/3	2/3

Step 5: Multiply by 2 and subtract 1

Constructing a portfolio using cosine similarity



	De-Meanned	Vanilla
PnL	311.17%	224.09%
Ann. Return	6.1%	5.0%
Sharpe	1.00	0.73
Max DD	(26.21%)	(29.88%)

Natural Language Processing Advanced

Deep Learning

Natural Language Processing

Word2Vec

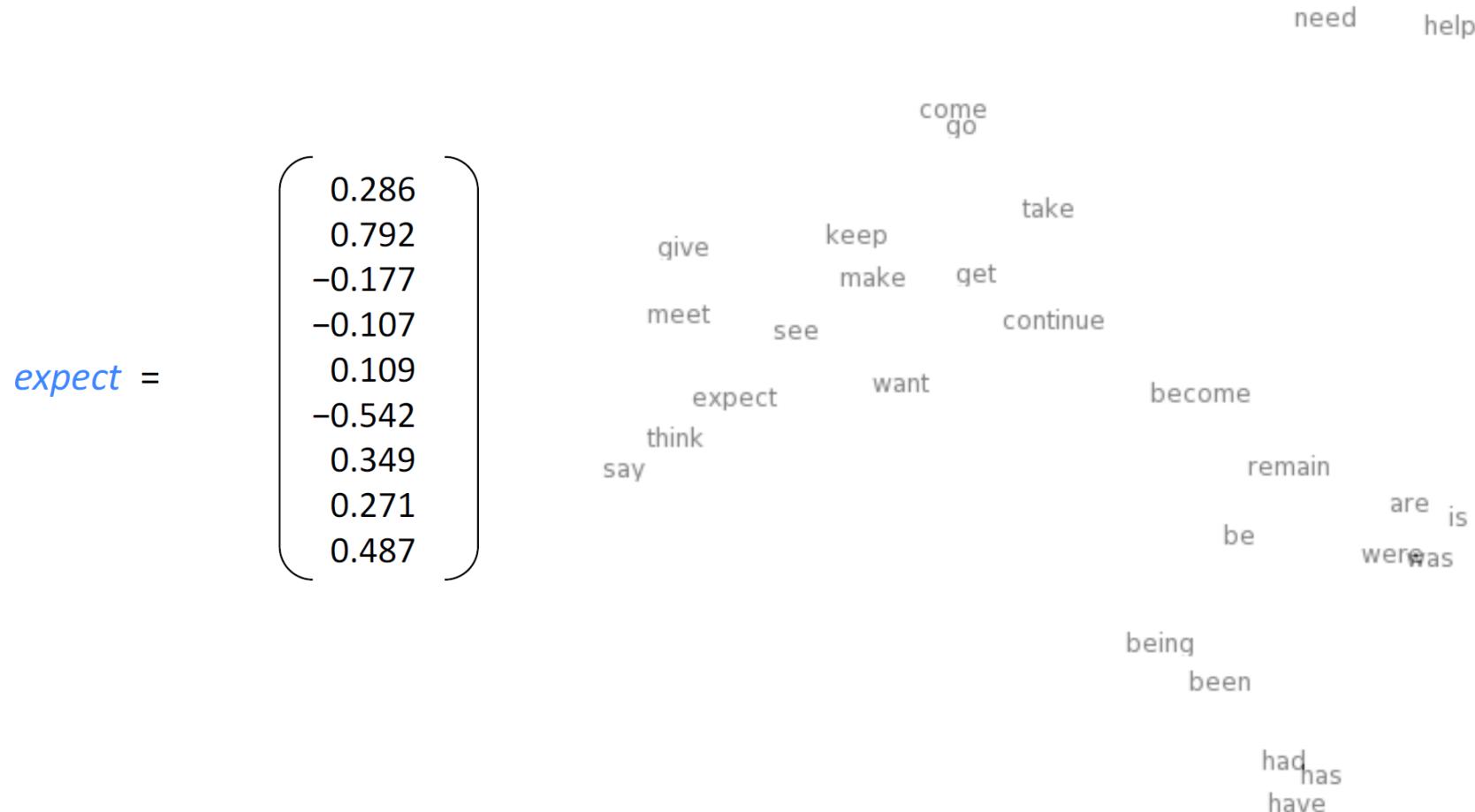
Word meaning is defined in terms of vectors

We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

... those other words also being represented by vectors ... it all gets a bit recursive

$$\text{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Word meaning as a neural word vector – visualization



Word2vec: Overview

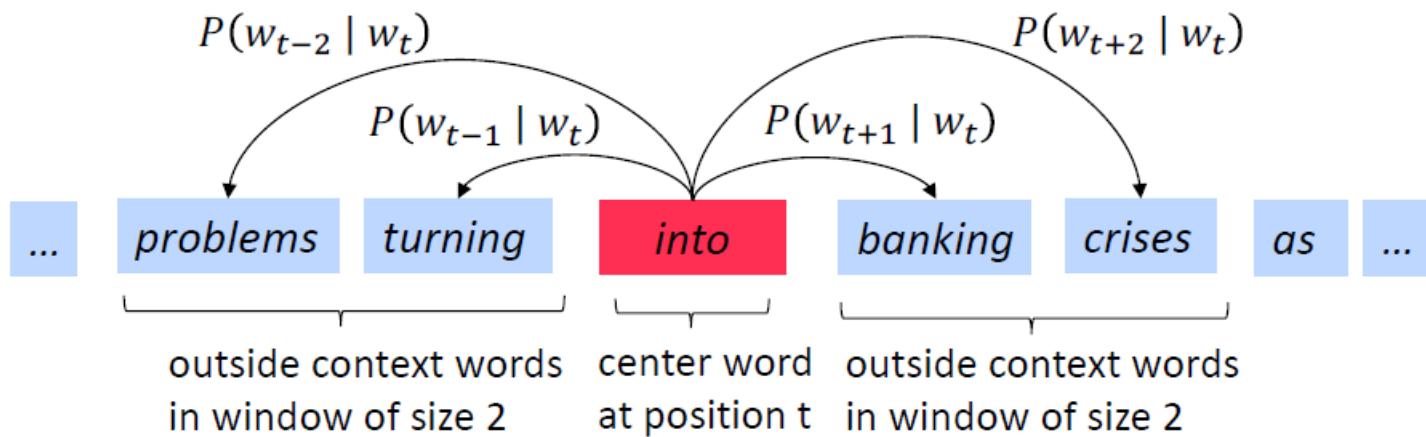
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context (“outside”) words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Main idea of word2vec (Mikolov et al. 2013)

- Example windows and process for computing $P(w_{t+j} | w_t)$



Word2vec: objective function

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m , given center word w_j .

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

θ is all variables
to be optimized

sometimes called *cost* or *loss* function

The **objective function** $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

Minimizing objective function \Leftrightarrow Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

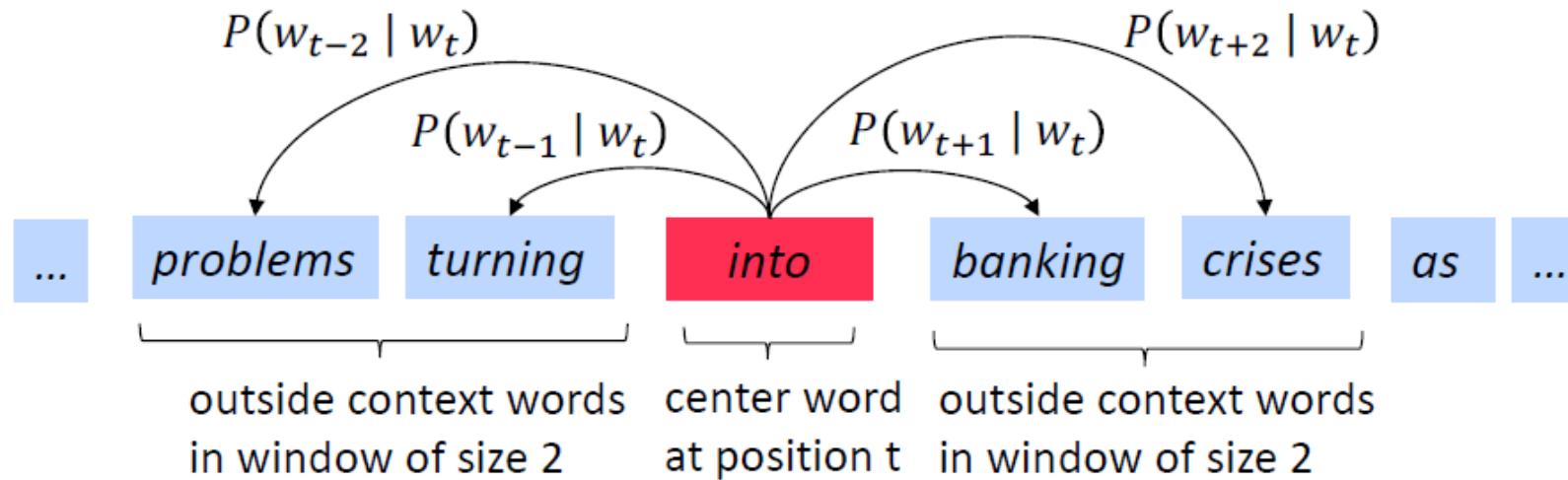
$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

- **Question:** How to calculate $P(w_{t+j} | w_t; \theta)$?
- **Answer:** We will use two vectors per word w :
 - v_w when w is a center word
 - u_w when w is a context word
- Then for a center word c and a context word o :

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Word2Vec - Example

- Example windows and process for computing $P(w_{t+j} | w_t)$



Word Embeddings - Glove

Word Embeddings

Load the pre-trained GloVe 300 dimensional embedding matrix Obtained here: <https://nlp.stanford.edu/projects/glove/>

```
In [2]: 1 embedding_file = 'data/glove.6B.300d.txt'  
2  
3 df_embedding = pd.read_csv(embedding_file, sep=' ', index_col=0,  
4                           header=None, na_values=None,  
5                           keep_default_na=False,  
6                           quoting=csv.QUOTE_NONE)  
7 df_embedding.sort_index(inplace=True)
```

```
In [3]: 1 df_embedding.shape
```

```
Out[3]: (400000, 300)
```

```
In [4]: 1 df_embedding.sample(10)
```

```
Out[4]:
```

	1	2	3	4	5	6	7	8	9	10	...	291	292	293	1
0															
sonar	0.33048	0.116850	-0.411260	0.188480	0.321090	-0.813960	-0.051148	-0.337450	-0.204780	-1.198300	...	-0.858400	0.252970	-0.539270	-0.5950
hearted	-0.20698	-0.404830	-0.267510	-0.591330	-0.064752	0.043845	0.084845	-0.105650	0.011034	-0.076297	...	-0.041760	0.238870	-0.110440	0.2389
dosage	-0.65587	0.827610	-0.055756	0.357710	0.394540	-0.125710	-0.314500	-0.197430	-0.250880	-0.710290	...	-0.332710	0.171840	0.229170	-0.5064
lightyears	-0.22943	-0.682830	-0.280610	0.480470	-0.135190	-0.379870	-0.172250	-0.115170	0.552260	0.839840	...	0.395030	0.395180	0.264700	-0.0008
ragni	-0.13872	0.161070	0.525340	0.042151	-0.436300	-0.484830	0.134350	0.033304	0.360310	0.768940	...	0.460640	0.338860	0.124590	-0.1319
poshteh-ye	-0.56235	0.084654	-0.066012	0.231860	-0.147000	-0.511670	0.572680	0.350820	-0.723060	1.059200	...	0.047565	0.557360	0.006690	0.3710
mdivani	0.36266	-0.045557	-0.034230	0.157770	-0.653320	-0.478830	0.234880	-0.129390	0.160240	0.981830	...	-0.099246	0.012156	0.445630	0.0298
harriers	0.38212	0.665550	0.366030	0.118270	0.145510	0.104760	0.538450	0.458350	0.065454	0.874270	...	-1.532600	0.975060	-0.196270	-0.6112
kolah	-0.25497	0.078224	-0.314990	-0.011261	0.041182	-0.240370	-0.041194	0.519860	-0.778400	0.574290	...	-0.341470	0.626570	0.012646	-0.4497
11-week	0.25130	0.842190	0.396610	0.275760	0.087887	0.737490	0.125450	-0.847930	0.401530	0.813140	...	0.183810	0.695540	-0.114280	0.3364

10 rows × 300 columns

Word Embeddings – Cosine similarity

Cosine Similarity

$$\text{sim}(U, V) = \frac{U^T \cdot V}{\|U\|_2 \|V\|_2} = \cos(\theta)$$



```
In [5]: 1 def cosine_similarity(u, v):
2     cs = np.dot(u, v) / (np.linalg.norm(u, ord=2) * np.linalg.norm(v, ord=2))
3     return cs
```

```
In [6]: 1 cosine_similarity(df_embedding.loc['france'], df_embedding.loc['italy'])
```

```
Out[6]: 0.5643459466769057
```

```
In [7]: 1 cosine_similarity(df_embedding.loc['ball'], df_embedding.loc['crocodile'])
```

```
Out[7]: 0.025038104793071447
```

```
In [8]: 1 cosine_similarity(df_embedding.loc['france'] - df_embedding.loc['paris'],
2                           df_embedding.loc['rome'] - df_embedding.loc['italy'])
```

```
Out[8]: -0.6228158930510513
```

Word Embeddings – Word Analogy

```
In [9]: 1 with open('data/google-10000-english.txt', 'r') as f:  
2     vocab = f.read().splitlines()  
3 df_embedding_small = df_embedding.reindex(vocab)
```

```
In [10]: 1 def word_analogy(a, b, c):  
2     emb_a, emb_b, emb_c = df_embedding_small.loc[a], \  
3                     df_embedding_small.loc[b], \  
4                     df_embedding_small.loc[c]  
5     max_sim = -100  
6     result = None  
7     for w, emb_w in df_embedding_small.iterrows():  
8         if w in [a, b, c]:  
9             continue  
10        sim = cosine_similarity(emb_b - emb_a, emb_w - emb_c)  
11        if sim > max_sim:  
12            max_sim = sim  
13            result = w  
14    return result
```

```
In [11]: 1 test_cases = [  
2     ('paris', 'france', 'london'),  
3     ('paris', 'france', 'rome'),  
4     ('paris', 'france', 'madrid'),  
5     ('paris', 'france', 'berlin'),  
6     ('man', 'king', 'woman'),  
7     ('walk', 'walked', 'go'),  
8     ('man', 'woman', 'boy'),  
9     ('apple', 'juice', 'mother')  
10 ]  
11  
12 for t in test_cases:  
13     r = word_analogy(t[0], t[1], t[2])  
14     print('{} -> {} : {} -> {}'.format(t[0], t[1], t[2], r))
```

```
paris -> france : london -> britain  
paris -> france : rome -> italy  
paris -> france : madrid -> spain  
paris -> france : berlin -> germany  
man -> king : woman -> queen  
walk -> walked : go -> went  
man -> woman : boy -> girl  
apple -> juice : mother -> lemon
```

Word Embeddings – Tensorflow

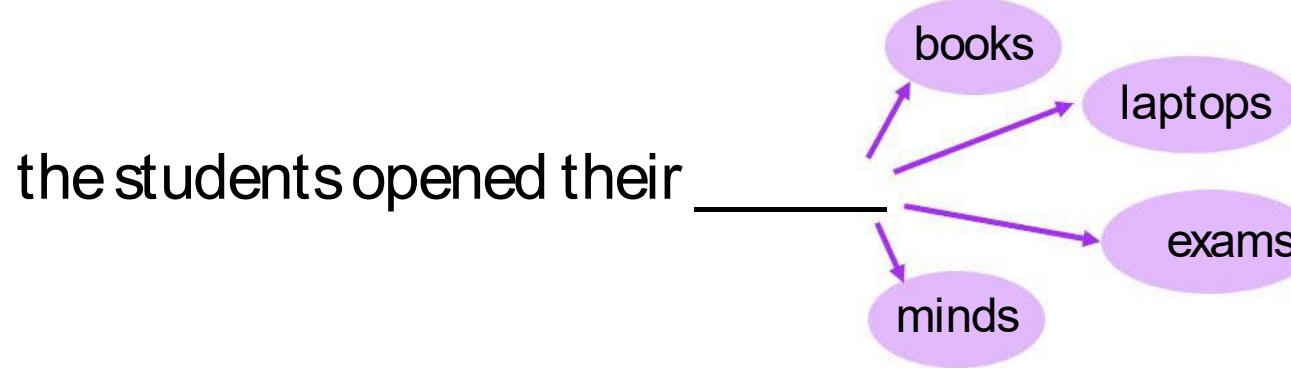
Notebook

https://www.tensorflow.org/text/guide/word_embeddings

Emergent linguistic structure in deep contextual neural word representations

1. Language Modeling

A **Language Model (LM)** predicts a word in a context



For a word sequence $\square^{(\square)}, \square^{(\#)}, \dots, \square^{(\%)}$, it gives the probability of $\square^{(\%&)}$:

$$\cdot (\square^{(\%&)} | \square^{(\%)}, \dots, \square^{(\square)})$$

An LM is a key part of decoding tasks like **speech recognition**, **spelling correction**, and any **NL generation task**, including **machine translation**, **summarization**, and **story generation**

LMs in The Dark Ages: n-gram models

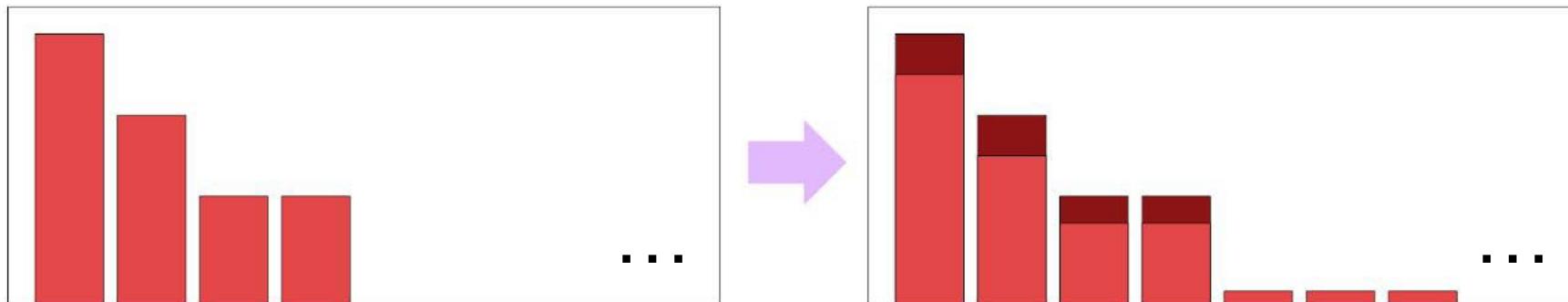
Count how many words follow word sequences; divide to get cond. prob.

Classic **curse of dimensionality** scenario: millions of params

Markov assumption:

$$\Pr(\text{the} | \text{President Trump denied the}) \approx \Pr(\text{the} | \text{denied the})$$

Discounting/Smoothing



Mixture/Backoff

$$\Pr_{78}(\text{the} | \text{President Trump denied the}) \approx : \Pr(\text{the} | \text{President Trump denied the}) + (1 - :) \Pr(\text{the} | \text{denied the})$$

Enlightenment era neural language models (NLMs)

1. Solve curse of dimensionality by sharing of statistical strength via dense, low-dimensionality word vectors $\text{>}_{\square}, \text{>}_{\#}, \dots, \text{>}_{?}$ [Bengio, Ducharme, Vincent & Jauvin JMLR 2003], etc.:

$$\cdot (\square^{(\% \& \)} | \square^{(\% \ , \ \% @ \)}) = \text{softmax}(\text{FFNN}(\text{>}^{(\% \), \ \text{>}^{(\% @ \)}}))$$

2. Solve failure to exploit long contexts via recurrent NNs

First, simple RNNs, soon usually LSTMs [Zaremba et al. 2014]

the same **stump** which had impaled the car of many a guest
in the past thirty years and which **he refused to have removed**

$$\cdot (\square^{(\% \& \)} | \square^{(H \%)}) = \text{LSTM}(h^{(\% \), \ \square^{(\%)}})$$

A RNN Language Model

output distribution

$$\hat{y}^{(t)} = \text{softmax}(\mathbf{U}\mathbf{h}^{(t)} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden states

$$\mathbf{h}^{(t)} = \sigma(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_e \mathbf{e}^{(t)} + \mathbf{b}_1)$$

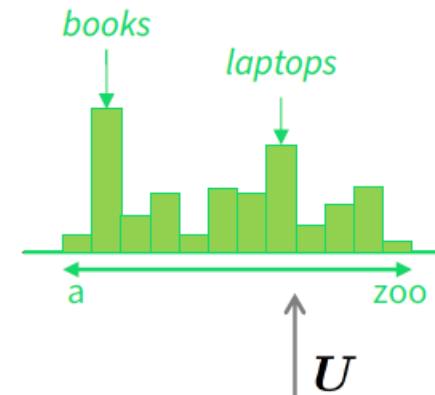
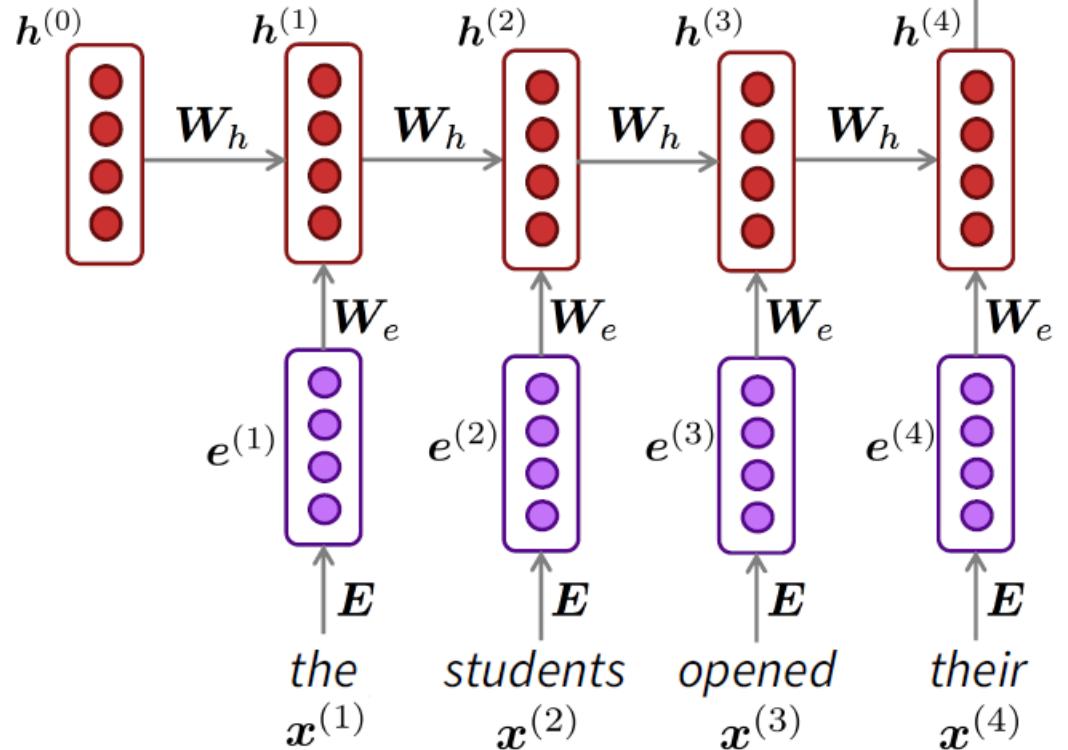
$\mathbf{h}^{(0)}$ is the initial hidden state

word embeddings

$$\mathbf{e}^{(t)} = \mathbf{E}\mathbf{x}^{(t)}$$

words / one-hot vectors

$$\mathbf{x}^{(t)} \in \mathbb{R}^{|V|}$$



Long Short-Term Memory (LSTM)

For LSTM:

Input Gate	$\mathbf{i}_t := \text{sigm}(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1}) ,$
Forget Gate	$\mathbf{f}_t := \text{sigm}(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1}) ,$
Output Gate	$\mathbf{o}_t := \text{sigm}(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1}) ,$
Memory State	$\tilde{\mathbf{c}}_t := \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1}) ,$
Cell State	$\mathbf{c}_t := \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t ,$
	$\mathbf{h}_t := \mathbf{o}_t \odot \tanh(\mathbf{c}_t) .$

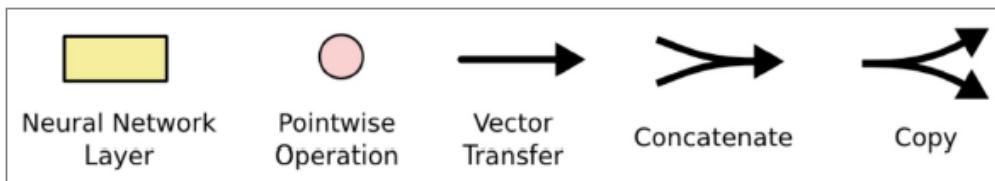
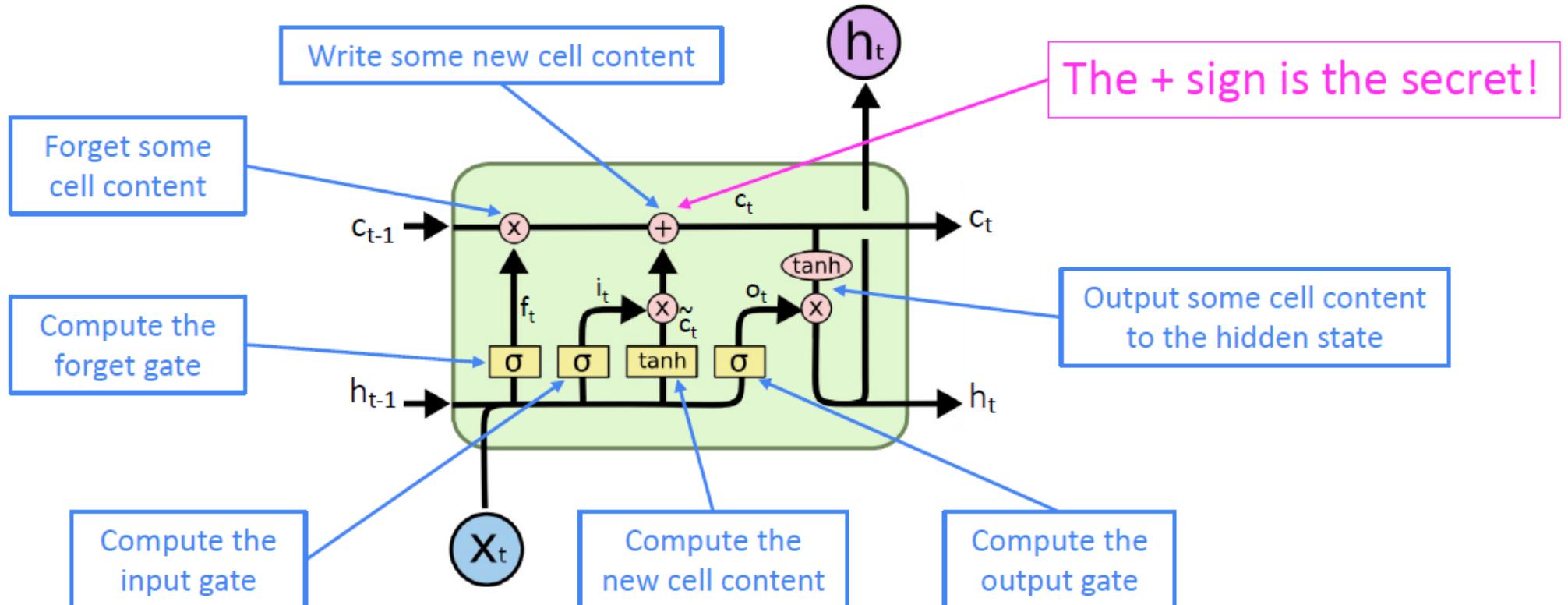
Similarly for RNN, we have

$$\mathbf{h}_t := \tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1}) .$$

Now it is easy to know why LSTM can solve the problems in RNN (Gradients?)

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



Contextual word representations

output distribution

$$\hat{y}^{(t)} = \text{softmax}(\mathbf{U}h^{(t)} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma(\mathbf{W}_h h^{(t-1)} + \mathbf{W}_e e^{(t)} + \mathbf{b}_1)$$

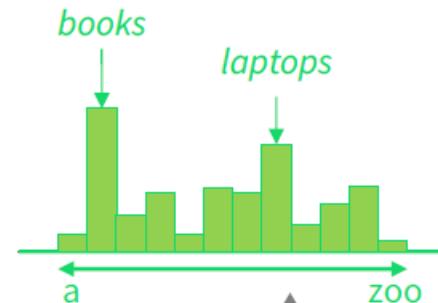
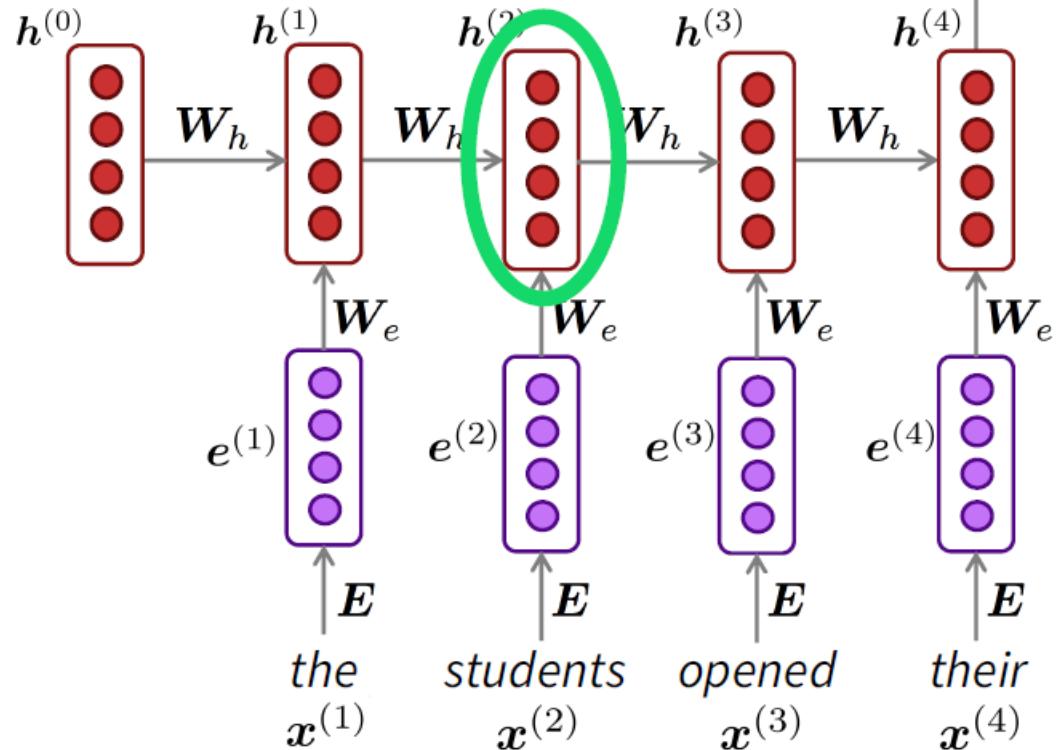
$h^{(0)}$ is the initial hidden state

word embeddings

$$e^{(t)} = \mathbf{E}x^{(t)}$$

words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$



Representing words by their context



- **Distributional semantics:** A word's meaning is given by the words that frequently appear close-by
 - “You shall know a word by the company it keeps” (J. R. Firth 1957: 11)
 - One of the most successful ideas of modern statistical NLP!
- When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of w to build up a representation of w

...government debt problems turning into banking crises as happened in 2009...

...saying that Europe needs unified banking regulation to replace the hodgepodge...

...India has just given its banking system a shot in the arm...



These **context words** will represent banking via a reconstruction loss

2. Let's scale it up!



ELMo	GPT	BERT	GPT-2	XL-Net, ERNIE, Grover RoBERTa, ...
Oct 2017	June 2018	Oct 2018	Feb 2019	
Training:	Training	Training	Training	
800M words	800M words	3.3B words	40B words	
42 GPU days	240 GPU days	256 TPU days ~320–560 GPU days	~2048 TPU v3 days according to a reddit thread	July 2019 on



Google AI



Transformer models

All of these models are Transformer models

ELMo
Oct 2017
Training:
800M words
42 GPU days



GPT
June 2018
Training
800M words
240 GPU days



BERT
Oct 2018
Training
3.3B words
256 TPU days
~320–560 GPU days



GPT-2
Feb 2019
Training
40B words
[~2048 TPU v3 days according to a reddit thread](#)



XL-Net,
ERNIE,
Grover
RoBERTa, ...
July 2019



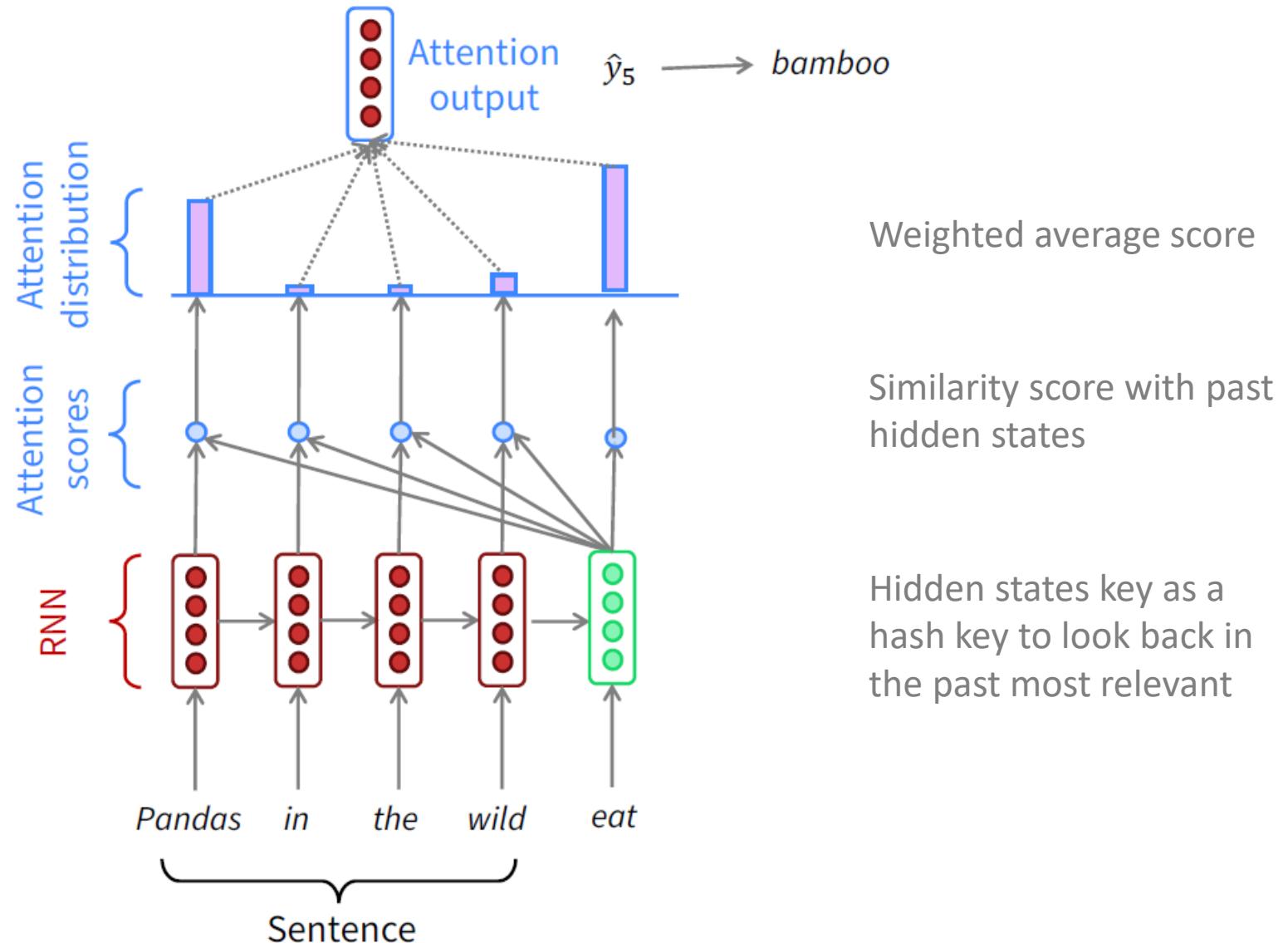
Recurrent models with (self-)attention

$$\mathbf{c}_t = \sum_s \mathbf{a}_t(s) \bar{\mathbf{h}}_s$$

$$\mathbf{a}_t(s) = \frac{e^{\text{score}(s)}}{\sum_{s'} e^{\text{score}(s')}}$$

$$\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_s) = \mathbf{h}_t^\top \mathbf{W}_a \bar{\mathbf{h}}_s$$

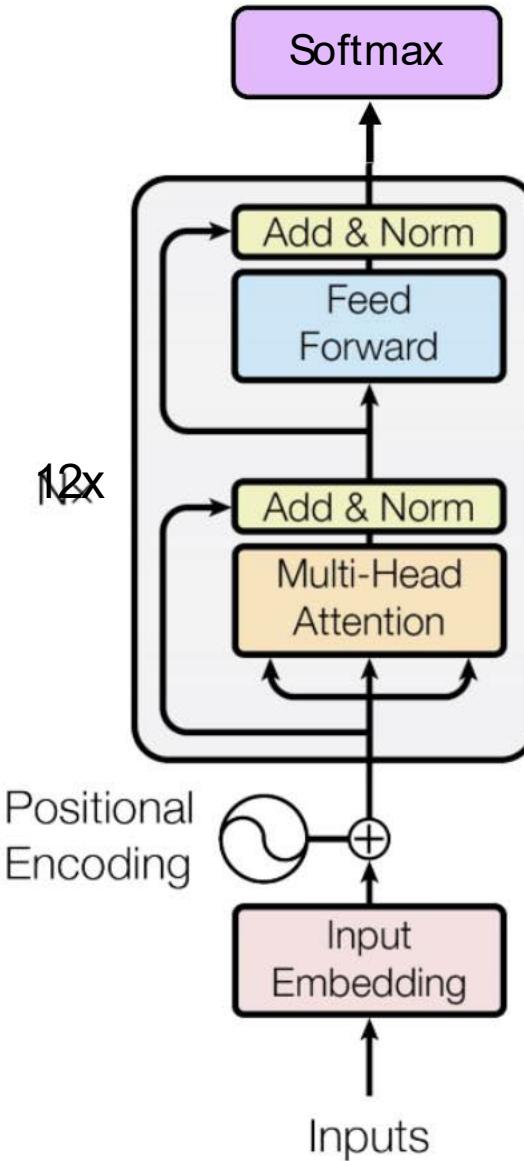
Bilinear attention



Transformer (Vaswani et al. 2017)

<https://arxiv.org/pdf/1706.03762.pdf>

- Non-recurrent sequence model (or sequence-to-sequence model)
- A deep model with a sequence of attention-based transformer blocks
- Depth allows a certain amount of lateral information transfer in understanding sentences, in slightly unclear ways
- Final cost/error function is standard cross-entropy error on top of a softmax classifier



BERT: Devlin, Chang, Lee, Toutanova (2018)



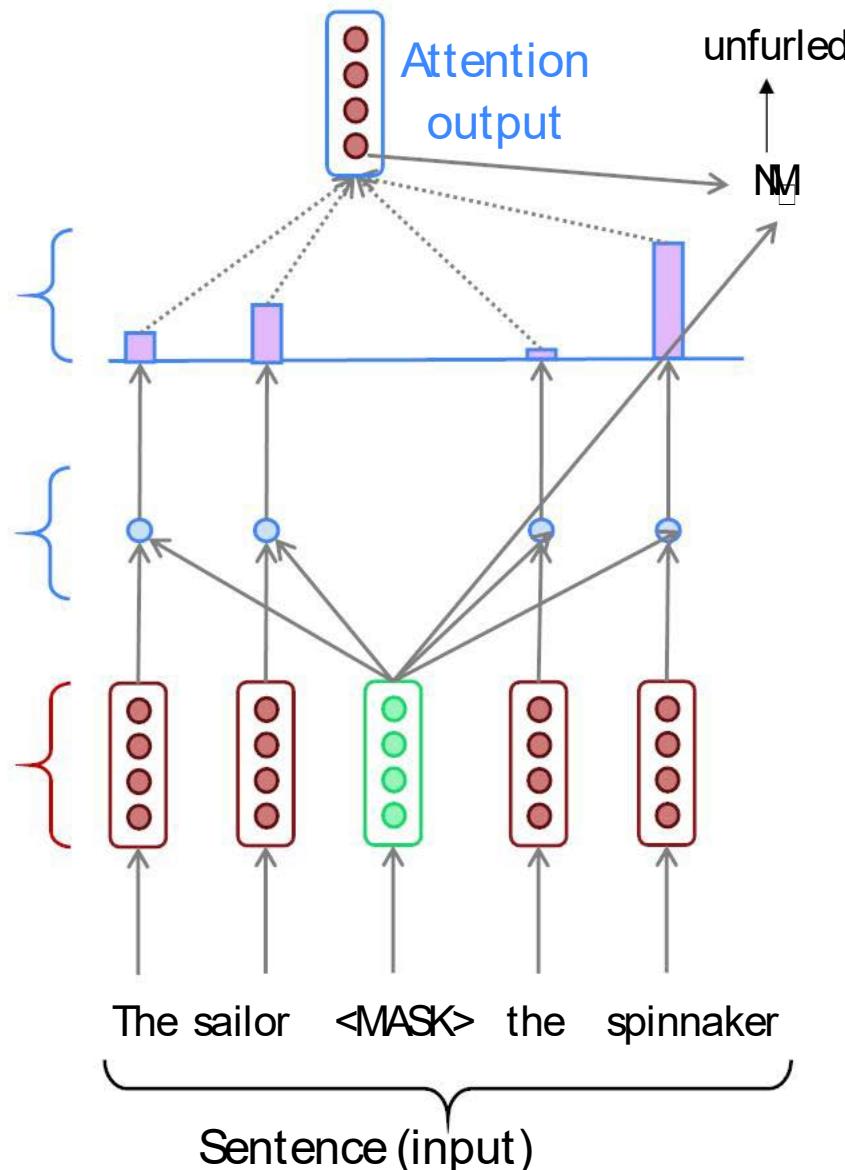
BERT (Bidirectional Encoder Representations from Transformers):
Pre-training of Deep Bidirectional Transformers for Language
Understanding, which is then fine-tuned for a particular task

Pre-training uses a cloze task formulation where 15% of words are masked out and predicted:

store gallon

the man went to the [MASK] to buy a [MASK] of milk

Self-attention in masked sequence model



We get the attention scores P^t for step t

$$\mathbf{e}^t = [\mathbf{s}_t^T \mathbf{h}_1, \dots, \mathbf{s}_t^T \mathbf{h}_N] \in \mathbb{R}^N$$

We take softmax to get the attention (prob.) distribution Q^t for step t

$$\alpha^t = \text{softmax}(\mathbf{e}^t) \in \mathbb{R}^N$$

We use Q^t to take weighted sum of the hidden states to get attention output

$$\mathbf{a}_t = \sum_{i=1}^N \alpha_i^t \mathbf{h}_i \in \mathbb{R}^h$$

Finally we join (sum or concatenate) the attention output \mathbf{R}^t with the decoder hidden state \mathbf{S}_t and proceed in model

Multi-head (self) attention

With simple self-attention: Only one way for a word to interact with others

Solution: Multi-head attention

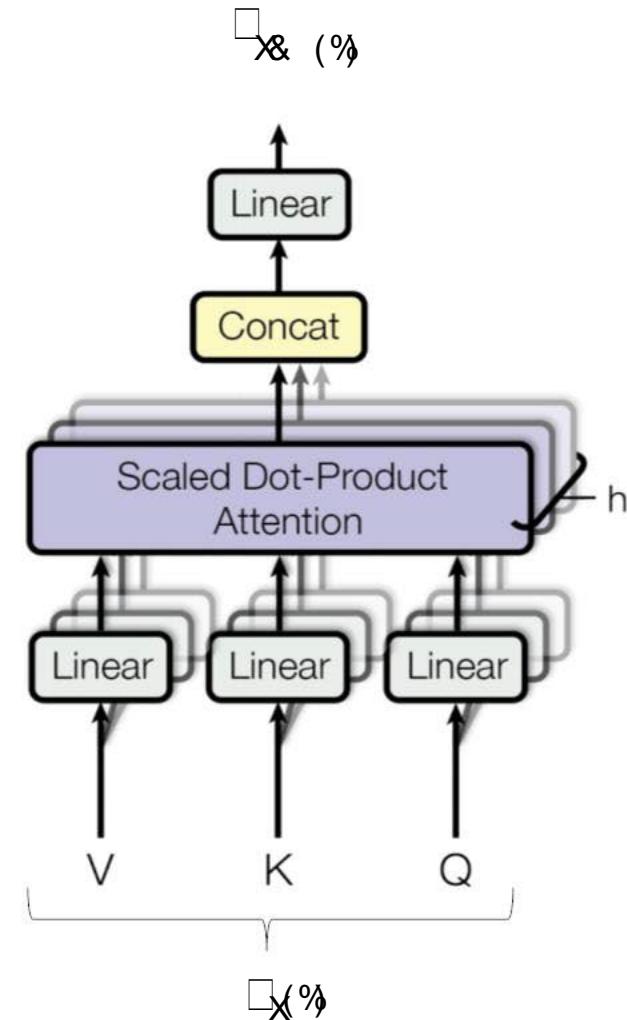
Map input into $h = 12$ many lower dimensional spaces via U_V matrices

Then apply attention, then concatenate outputs and pipe through linear layer

$$\text{Multihead}(\mathbf{x}^{(\%)}) = \text{Concat}(h_{\text{PR}}[\cdot]) \mathbf{U}_V$$

$$h_{\text{PR}}[\cdot] = \text{Attention}(\mathbf{x}^{(\%)} \mathbf{U}_V[-], \mathbf{x}^{(\%)} \mathbf{U}_V[?], \mathbf{x}^{(\%)} \mathbf{U}_V[\cdot])$$

$$\text{So attention is like bilinear: } \mathbf{x}^{(\%)} (\mathbf{U}_V[-] (\mathbf{U}_V[?])^T)^a \mathbf{x}^{(\%)}_b$$

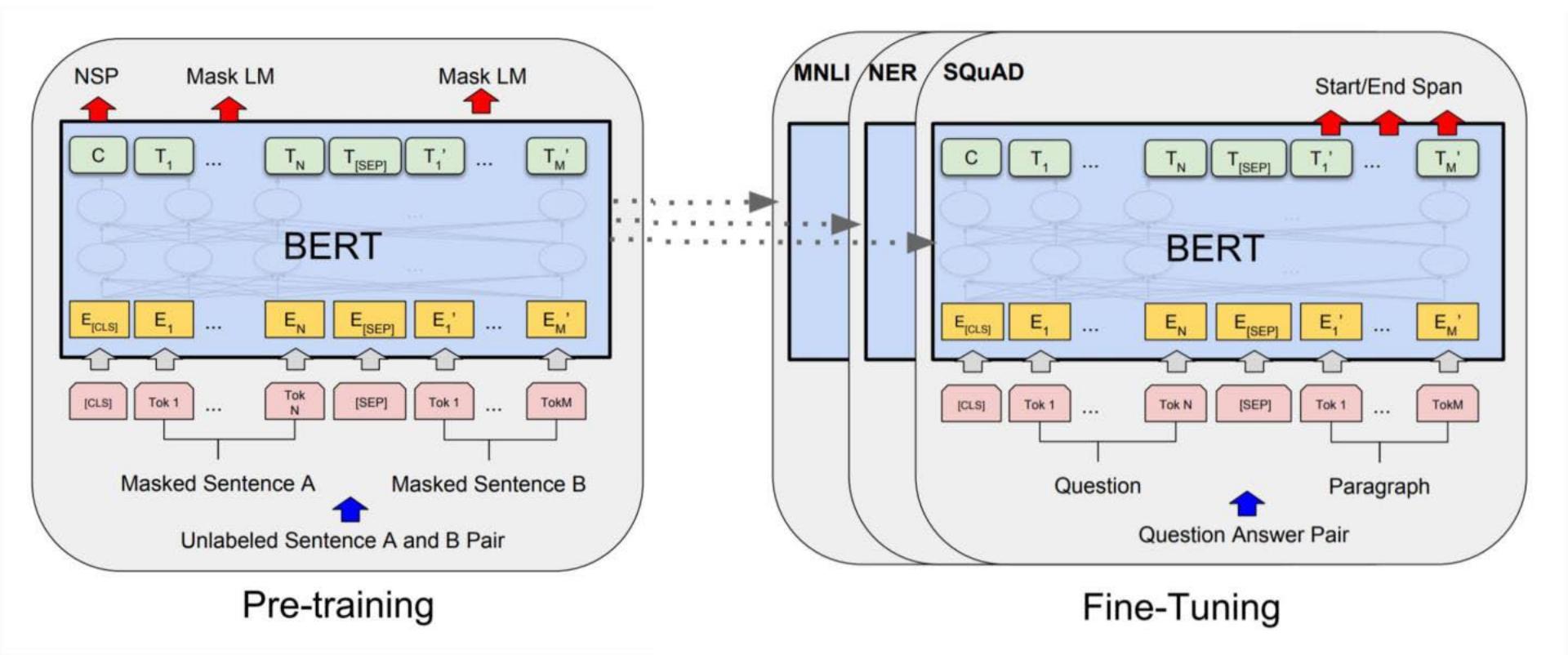


BERT model



Pre-train contextual word vectors in a LM-like way with transformers

Learn a classifier built on the top layer for each task that you fine tune for



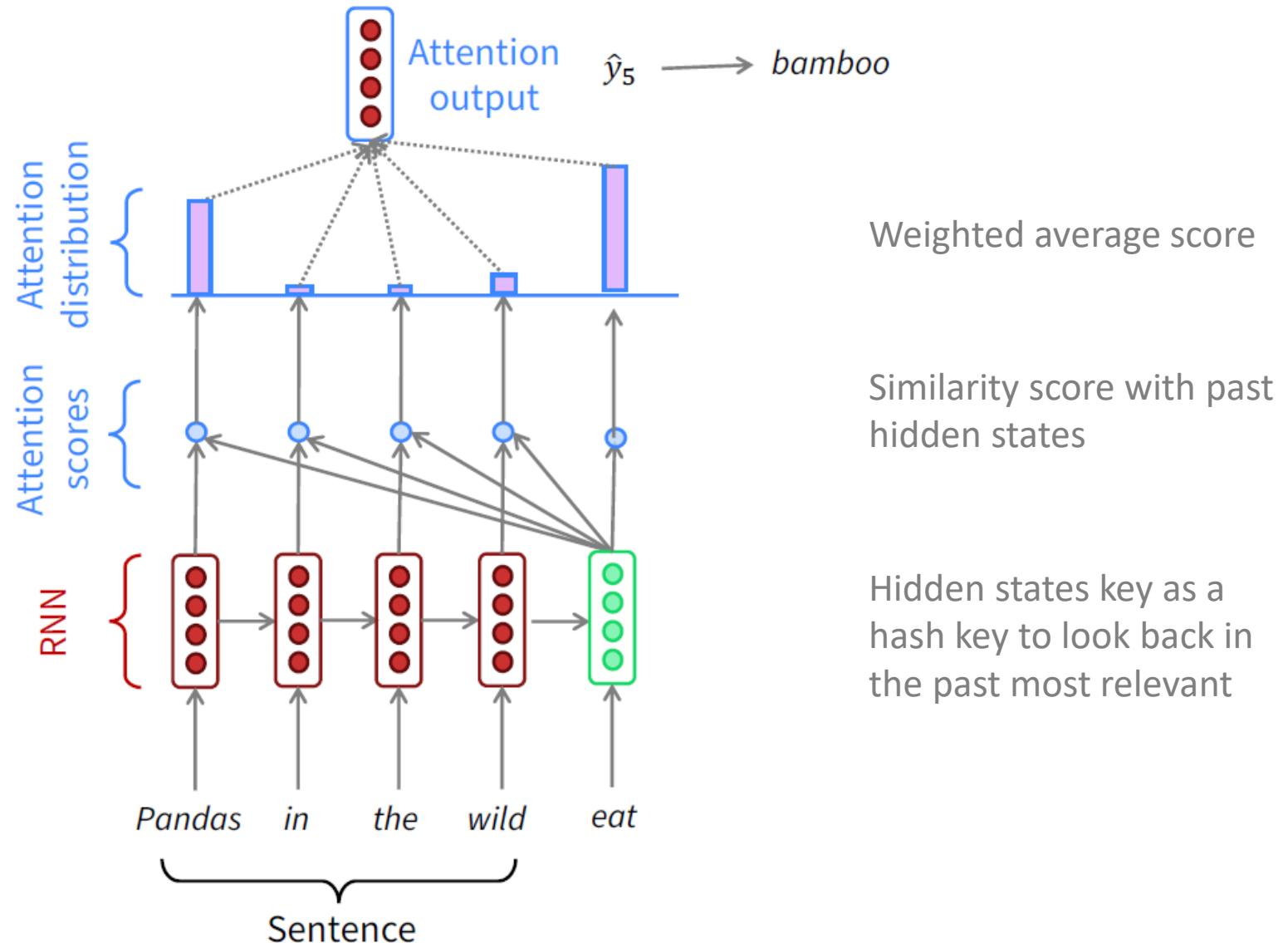
Recurrent models with (self-)attention

$$\mathbf{c}_t = \sum_s \mathbf{a}_t(s) \bar{\mathbf{h}}_s$$

$$\mathbf{a}_t(s) = \frac{e^{\text{score}(s)}}{\sum_{s'} e^{\text{score}(s')}}$$

$$\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_s) = \mathbf{h}_t^\top \mathbf{W}_a \bar{\mathbf{h}}_s$$

Bilinear attention



Attention and Self-Attention

Attention is a mechanism in the neural network that a model can learn to make predictions by selectively attending to a given set of data. The amount of attention is quantified by learned weights and thus the output is usually formed as a weighted average.

Self-attention is a type of attention mechanism where the model makes prediction for one part of a data sample using other parts of the observation about the same sample. Conceptually, it feels quite similar to [non-local means](#). Also note that self-attention is permutation-invariant; in other words, it is an operation on sets.

There are various forms of attention / self-attention, Transformer ([Vaswani et al., 2017](#)) relies on the *scaled dot-product attention*: given a query matrix \mathbf{Q} , a key matrix \mathbf{K} and a value matrix \mathbf{V} , the output is a weighted sum of the value vectors, where the weight assigned to each value slot is determined by the dot-product of the query with the corresponding key:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

And for a query and a key vector $\mathbf{q}_i, \mathbf{k}_j \in \mathbb{R}^d$ (row vectors in query and key matrices), we have a scalar score:

$$a_{ij} = \text{softmax}\left(\frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d_k}}\right) = \frac{\exp(\mathbf{q}_i \mathbf{k}_j^\top)}{\sqrt{d_k} \sum_{r \in S_i} \exp(\mathbf{q}_i \mathbf{k}_r^\top)}$$

where S_i is a collection of key positions for the i -th query to attend to.

Notations

Symbol	Meaning
d	The model size / hidden state dimension / positional encoding size.
h	The number of heads in multi-head attention layer.
L	The segment length of input sequence.
$\mathbf{X} \in \mathbb{R}^{L \times d}$	The input sequence where each element has been mapped into an embedding vector of shape d , same as the model size.
$\mathbf{W}^k \in \mathbb{R}^{d \times d_k}$	The key weight matrix.
$\mathbf{W}^q \in \mathbb{R}^{d \times d_k}$	The query weight matrix.
$\mathbf{W}^v \in \mathbb{R}^{d \times d_v}$	The value weight matrix. Often we have $d_k = d_v = d$.
$\mathbf{W}_i^k, \mathbf{W}_i^q \in \mathbb{R}^{d \times d_k/h}; \mathbf{W}_i^v \in \mathbb{R}^{d \times d_v/h}$	The weight matrices per head.
$\mathbf{W}^o \in \mathbb{R}^{d_v \times d}$	The output weight matrix.
$\mathbf{Q} = \mathbf{X}\mathbf{W}^q \in \mathbb{R}^{L \times d_k}$	The query embedding inputs.
$\mathbf{K} = \mathbf{X}\mathbf{W}^k \in \mathbb{R}^{L \times d_k}$	The key embedding inputs.
$\mathbf{V} = \mathbf{X}\mathbf{W}^v \in \mathbb{R}^{L \times d_v}$	The value embedding inputs.
S_i	A collection of key positions for the i -th query \mathbf{q}_i to attend to.
$\mathbf{A} \in \mathbb{R}^{L \times L}$	The self-attention matrix between a input sequence of lenght L and itself. $\mathbf{A} = \text{softmax}(\mathbf{Q}\mathbf{K}^\top / \sqrt{d_k})$.
$a_{ij} \in \mathbf{A}$	The scalar attention score between query \mathbf{q}_i and key \mathbf{k}_j .
$\mathbf{P} \in \mathbb{R}^{L \times d}$	position encoding matrix, where the i -th row \mathbf{p}_i is the positional encoding for input \mathbf{x}_i .

Positional Encoding

The encoding proposed by the authors is a simple yet genius technique which satisfies all of those criteria. First of all, it isn't a single number. Instead, it's a d -dimensional vector that contains information about a specific position in a sentence. And secondly, this encoding is not integrated into the model itself. Instead, this vector is used to equip each word with information about its position in a sentence. In other words, we enhance the model's input to inject the order of words.

Let t be the desired position in an input sentence, $\vec{p}_t \in \mathbb{R}^d$ be its corresponding encoding, and d be the encoding dimension (where $d \equiv_2 0$) Then $f : \mathbb{N} \rightarrow \mathbb{R}^d$ will be the function that produces the output vector \vec{p}_t and it is defined as follows:

$$\vec{p}_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases}$$

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

Positional Encoding

You can also imagine the positional embedding \vec{p}_t as a vector containing pairs of sines and cosines for each frequency (Note that d is divisible by 2):

$$\vec{p}_t = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \vdots \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}_{d \times 1}$$

BERT Results with IMDB Data Set – Joint work with Tinghao Li

- BERT Base version 110 million parameters
- After 17 minutes training in one GPU core, it achieved 97% accuracy for the classification with F1 score at 96%

GPT-3: Language Models are Few-Shot Learners

Recent work has demonstrated substantial gains on many NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. While typically task-agnostic in architecture, this method still requires task-specific fine-tuning datasets of thousands or tens of thousands of examples. By contrast, humans can generally perform a new language task from only a few examples or from simple instructions – something which current NLP systems still largely struggle to do. Here we show that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even reaching competitiveness with prior state-of-the-art fine-tuning approaches. <https://github.com/openai/gpt-3>

Specifically, we train GPT-3, an autoregressive language model with 175 billion parameters, 10x more than any previous non-sparse language model, and test its performance in the few-shot setting. For all tasks, GPT-3 is applied without any gradient updates or fine-tuning, with tasks and few-shot demonstrations specified purely via text interaction with the model. GPT-3 achieves strong performance on many NLP datasets, including translation, question-answering, and cloze tasks, as well as several tasks that require on-the-fly reasoning or domain adaptation, such as unscrambling words, using a novel word in a sentence, or performing 3-digit arithmetic. At the same time, we also identify some datasets where GPT-3's few-shot learning still struggles, as well as some datasets where GPT-3 faces methodological issues related to training on large web corpora. Finally, we find that GPT-3 can generate samples of news articles which human evaluators have difficulty distinguishing from articles written by humans. We discuss broader societal impacts of this finding and of GPT-3 in general.

GPT-3: Language Models are Few-Shot Learners

Resources

<https://github.com/openai/gpt-3>

<https://gpt3examples.com/>

<https://www.notion.so/API-Developer-Toolkit-49595ed6ffcd413e93ebff10d7e70fe7>

Natural Language Processing

Tokenizing and Building a Model

Loading and Tokenizing

Load data

```
In [12]: 1 news_data = pd.read_csv('data/news_data.csv',
 2                      usecols = ['TEXT', 'SENTIMENT'])
 3 news_data.drop_duplicates(inplace=True)
```

```
In [13]: 1 news_data.tail()
```

Out[13]:

	SENTIMENT	TEXT
611430	0.94	Commonwealth Bank of Australia Upgraded to Buy from Hold by Bell Potter
611431	-0.58	Qbe Insurance Downgraded to Hold from Buy by Bell Potter
611432	0.49	Collective & Roland Team for
611433	-0.58	Mcmillan Shakespeare Downgraded to Neutral from Outperform by Macquarie
611434	0.95	Nearmap Initiated at Outperform by Macquarie

Loading and Tokenizing

Initialize the tokenizer and compress the embedding matrix

Build out a set of all tokens from every `TEXT` record using the full tokenizer (with its vocab initialised from the embedding matrix), then filter the embedding matrix dictionary to only include those tokens that exist in the `TEXT`. Then, re-instantiate the tokenizer using the embedding matrix dictionary. This speeds up training for testing/POC purpose by minimizing the size of the embedding tensor.

```
In [14]: 1 # Load saved tokenizer?
2 if True:
3     with open('model/tokenizer.pkl', 'rb') as f:
4         tokenizer = pickle.load(f)
5     df_embedding = df_embedding.reindex(tokenizer.vocab.keys())
6     df_embedding.sort_index(inplace=True)
7     df_embedding.loc['<START>'] = np.zeros(df_embedding.shape[1])
8     df_embedding.loc['<UNK>'] = df_embedding.mean()
9     df_embedding.loc['<NUMBER>'] = np.ones(df_embedding.shape[1])/2
10    df_embedding.loc['<PAD>'] = np.ones(df_embedding.shape[1])
11
12 else:
13     tokenizer = FullTokenizer(df_embedding.index.values)
14
15 def process(arr):
16     for i in map(tokenizer.tokenize, arr):
17         tokens.update(i)
18     return tokens
19
20 tokens = set()
21 if __name__ == '__main__':
22     p = mp.Pool(processes=mp.cpu_count())
23     split_arr = np.array_split(news_data.TEXT.values, mp.cpu_count())
24     pool_results = p.map(process, split_arr)
25     p.close()
26     p.join()
27
28     all_tokens = set.union(*pool_results)
29
30 all_tokens = set([t for t in all_tokens if not is_number(t)])
31 df_embedding = df_embedding.reindex(all_tokens).dropna()
32 df_embedding.sort_index(inplace=True)
33 df_embedding.loc['<START>'] = np.zeros(df_embedding.shape[1])
34 df_embedding.loc['<UNK>'] = df_embedding.mean()
35 df_embedding.loc['<NUMBER>'] = np.ones(df_embedding.shape[1])/2
36 df_embedding.loc['<PAD>'] = np.ones(df_embedding.shape[1])
37
38 tokenizer = FullTokenizer(df_embedding.index.values)
```

Loading and Tokenizing

Tokenization example

```
In [15]: 1 test_sentence = 'Tokenizer takes a string and breaks it down into tokens.'
2 tokenizer.tokenize(test_sentence)
```

```
Out[15]: ['tokenizer',
'takes',
'a',
'string',
'and',
'breaks',
'it',
'down',
'into',
'tokens',
'.']
```

```
In [16]: 1 tokenizer.convert_tokens_to_ids(tokenizer.tokenize(test_sentence))
```

```
Out[16]: [1241, 56549, 1252, 55330, 3529, 8641, 29211, 17049, 28807, 58228, 19]
```

Visualise tokens in the word embeddings matrix

```
In [17]: 1 df_embedding.iloc[tokenizer.convert_tokens_to_ids(tokenizer.tokenize(test_sentence))]
```

```
Out[17]:
```

	1	2	3	4	5	6	7	8	9	10	...	291	292	293	294
0															
<UNK>	0.056789	-0.022786	-0.002645	0.069256	0.010156	-0.036345	0.032409	-0.064922	-0.041482	0.234542	...	0.020075	0.102891	0.090614	-0.024519
takes	-0.265480	-0.109470	0.179890	-0.077156	-0.001871	-0.027737	-0.180900	-0.145750	0.341820	-1.314900	...	-0.278790	0.128100	-0.213730	-0.148400
a	-0.297120	0.094049	-0.096662	-0.344000	-0.184830	-0.123290	-0.116560	-0.099692	0.172650	-1.638600	...	0.075972	-0.424260	-0.396700	0.326830
string	0.490790	0.373550	-0.108000	-0.181940	-0.406680	-0.214930	-0.200550	-0.469240	-0.120310	-0.924760	...	0.161770	-0.332490	-0.319580	0.347060
and	0.038466	-0.039792	0.082747	-0.389230	-0.214310	0.170200	-0.025657	0.095780	0.238600	-1.634200	...	0.045194	-0.204050	-0.210970	-0.110250
breaks	-0.273560	0.012609	-0.441310	0.410770	0.134850	0.144790	0.190910	-0.257050	0.442460	-0.934800	...	-0.304630	0.051168	-0.794090	0.129680
it	0.033284	-0.040754	-0.048377	0.120170	-0.139150	-0.176940	-0.062908	0.170560	0.200770	-2.428700	...	0.091222	-0.402000	0.154300	0.230990
down	-0.081429	-0.110040	-0.031034	0.604570	0.067606	-0.316090	-0.460590	-0.202730	0.468520	-1.600900	...	0.212930	-0.172900	0.013380	0.450640
into	-0.182340	-0.192090	0.151670	-0.048763	0.283540	-0.025568	-0.398820	-0.014072	0.410290	-1.987000	...	-0.058528	-0.133670	0.159020	0.072970
tokens	-0.671420	-0.013502	0.078301	0.009528	-0.039577	0.361740	0.370120	0.124180	0.449070	0.137810	...	-0.519930	-0.119760	-0.075887	0.342280
.	-0.125590	0.013630	0.103060	-0.101230	0.098128	0.136270	-0.107210	0.236970	0.328700	-1.678500	...	0.060148	-0.156190	-0.119490	0.234450

11 rows × 300 columns



Loading and Tokenizing

Define encoding/decoding methods

The `encode` method takes a string and converts it to IDs using the tokenizer dictionary. `decode` performs the opposite action.

```
In [18]: 1 def encode(string):
2     encoding = ["<START>"] + tokenizer.tokenize(string)
3     encoding = tokenizer.convert_tokens_to_ids(encoding)
4     return encoding
5
6 def decode(ids):
7     tokens = tokenizer.convert_ids_to_tokens(ids)
8     return tokens
```

Process the dataset

We encode `TEXT` using the tokenizer to obtain IDs that correspond to each token's index within the embedding matrix. We use multiprocessing to process large datasets.

```
In [19]: 1 def process(df):
2     res = df.TEXT.apply(encode)
3     return res
4
5 if __name__ == '__main__':
6     p = mp.Pool(processes=mp.cpu_count())
7     split_dfs = np.array_split(news_data, mp.cpu_count())
8     pool_results = p.map(process, split_dfs)
9     p.close()
10    p.join()
11
12    # merging parts processed by different processes
13    dataset = pd.concat(pool_results, axis=0)
```

Preview the dataset:

```
In [20]: 1 dataset.tail()
```

```
Out[20]: 611430    [1240, 12768, 5885, 41528, 5144, 60208, 58166, 9550, 21961, 26614, 9589, 6649, 45148]
611431        [1240, 46376, 28611, 17057, 58166, 26614, 21961, 9550, 9589, 6649, 45148]
611432            [1240, 12567, 5, 49267, 56981, 21423]
611433        [1240, 36652, 52076, 17057, 58166, 40196, 21961, 42449, 9589, 35053]
611434            [1240, 1241, 28416, 4837, 42449, 9589, 35053]
Name: TEXT, dtype: object
```

Loading and Tokenizing

Let's decode the sample:

```
In [21]: 1 dataset.tail().apply(decode)
```

```
Out[21]: 611430    [<START>, commonwealth, bank, of, australia, upgraded, to, buy, from, hold, by, bell, potter]
611431            [<START>, qbe, insurance, downgraded, to, hold, from, buy, by, bell, potter]
611432            [<START>, collective, &, roland, team, for]
611433    [<START>, mcmillan, shakespeare, downgraded, to, neutral, from, outperform, by, macquarie]
611434            [<START>, <UNK>, initiated, at, outperform, by, macquarie]
Name: TEXT, dtype: object
```

Pad all sequences to maxlen (inferred from the dataset) to obtain equal length vectors

```
In [22]: 1 maxlen = dataset.apply(len).max() + 10
2 data = keras.preprocessing.sequence.pad_sequences(dataset.values,
3                                                 value=tokenizer.vocab["<PAD>"],
4                                                 padding='post',
5                                                 maxlen=maxlen)
```

Build The model

```
In [26]: 1 neurons = 128
2 dropout = 0.40
3
4 my_embedding = layers.Embedding(input_dim=df_embedding.shape[0],
5                                 output_dim=df_embedding.shape[1],
6                                 input_length=maxlen,
7                                 trainable=False)
8
9 my_embedding.build((None, ))
10 my_embedding.set_weights([df_embedding.values])
11
12 model = keras.Sequential([
13     my_embedding,
14     layers.LSTM(neurons, return_sequences=True),
15     layers.Dropout(dropout),
16     layers.LSTM(neurons, return_sequences=False),
17     layers.Dropout(dropout),
18     layers.Dense(1, activation='tanh')
19 ])
20
21 optimizer = tf.optimizers.Adam(learning_rate=0.001)
22
23 def r2_keras(y_true, y_pred):
24     SS_res = K.sum(K.square(y_true - y_pred))
25     SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
26     return (1 - SS_res/(SS_tot + K.epsilon()))
27
28 model.compile(loss='mean_squared_error',
29                 optimizer=optimizer,
30                 metrics=[r2_keras])
31 model.summary()
32
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
embedding (Embedding)	(None, 58, 300)	19349400
lstm (LSTM)	(None, 58, 128)	219648
dropout (Dropout)	(None, 58, 128)	0
lstm_1 (LSTM)	(None, 128)	131584
dropout_1 (Dropout)	(None, 128)	0
dense (Dense)	(None, 1)	129
<hr/>		
Total params: 19,700,761		
Trainable params: 351,361		
Non-trainable params: 19,349,400		

FinEAS : Financial Embedding Analysis of Sentiment

Asier Gutiérrez-Fandiño

Miquel Noguer i Alonso

Petter Kolm

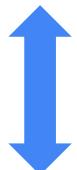
Jordi Armengol-Estabé

<https://arxiv.org/abs/2111.00526>

What is Sentiment

Sentiment is a very difficult task!

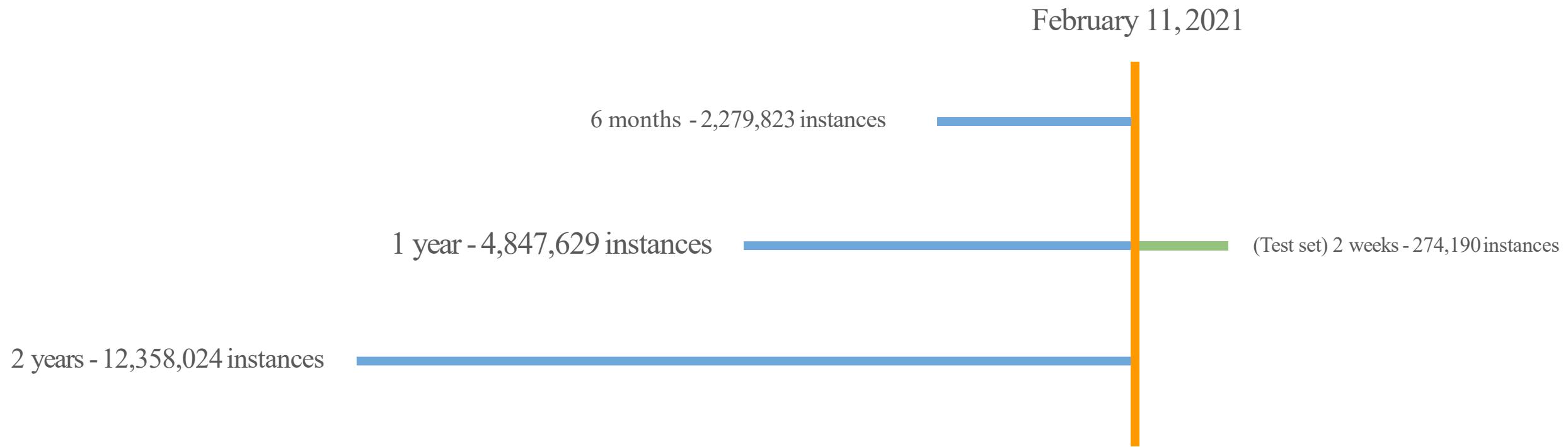
- **Contextual** “I could die for a McFlurry” / “I could die for not eating”
- **References** “I have been waiting for you as much as for the PS4 availability”
 - Knowledge Base?
- **Imbalance** for instance, too much hate about Telcos on Twitter
- **Time dependent** “My neighbour wears a face mask” previous to 2019 could be **confusion**, today might be labeled **asneutral**.
- **Irony** is very difficult to understand.
- **Emojis** can be your enemy 😅
- **Impact** is difficult to measure



Annotation is **ALWAYS** biased

About the dataset I

Data collection



About the dataset II

Low imbalance.

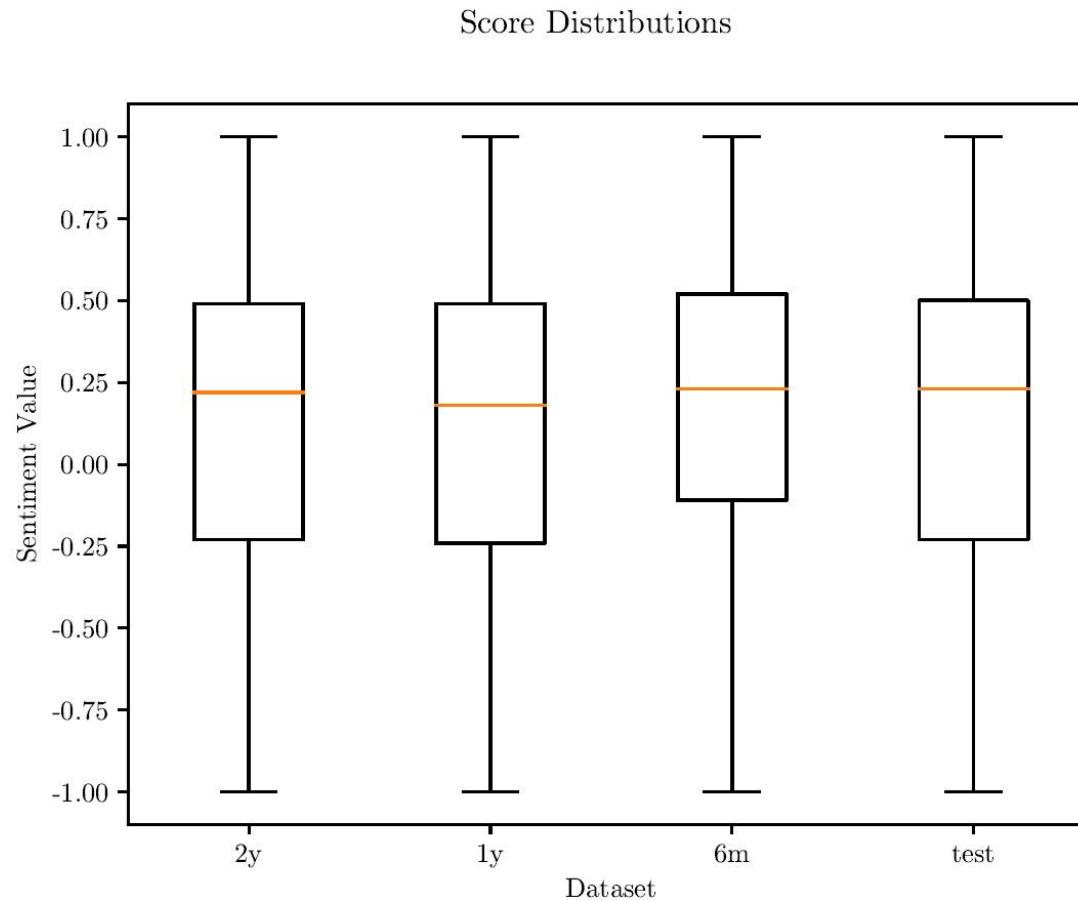
Min and max met.

Nice boxes.

Compare 1y with 6m and 1y with test.

- It is the time where vaccines started to be released.

✓ Good distribution, Good dataset.

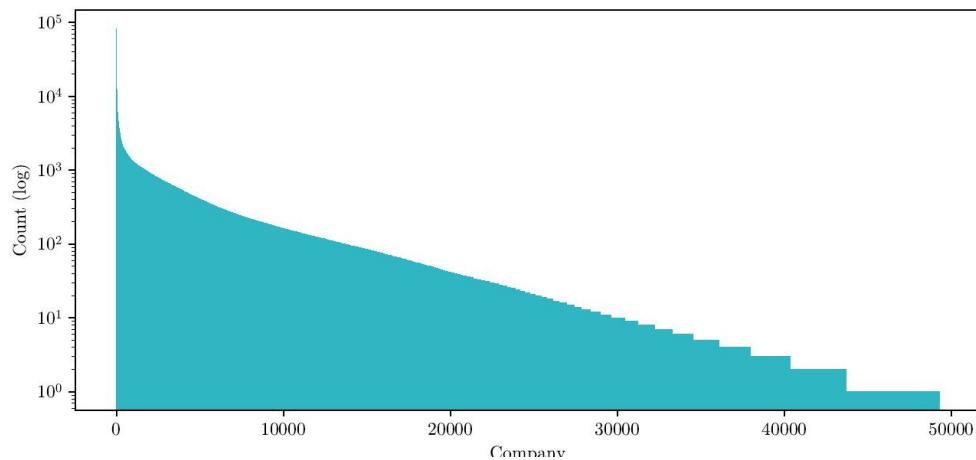


About the dataset III

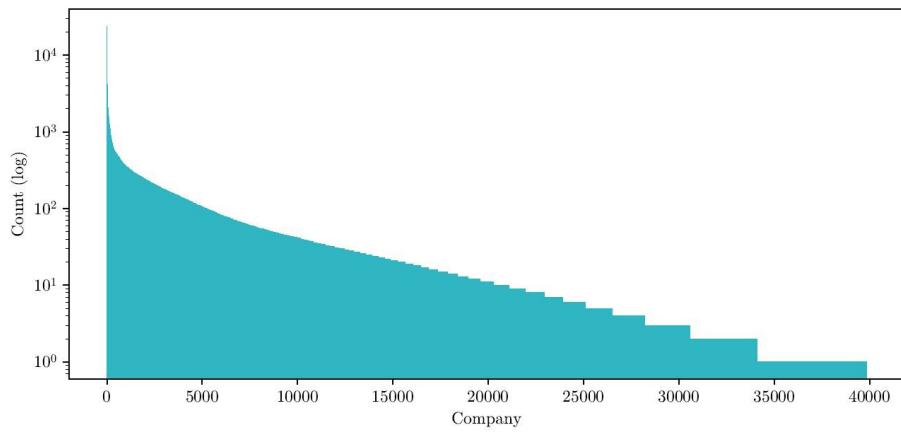
What about the company distribution?

Follows a Power-Law

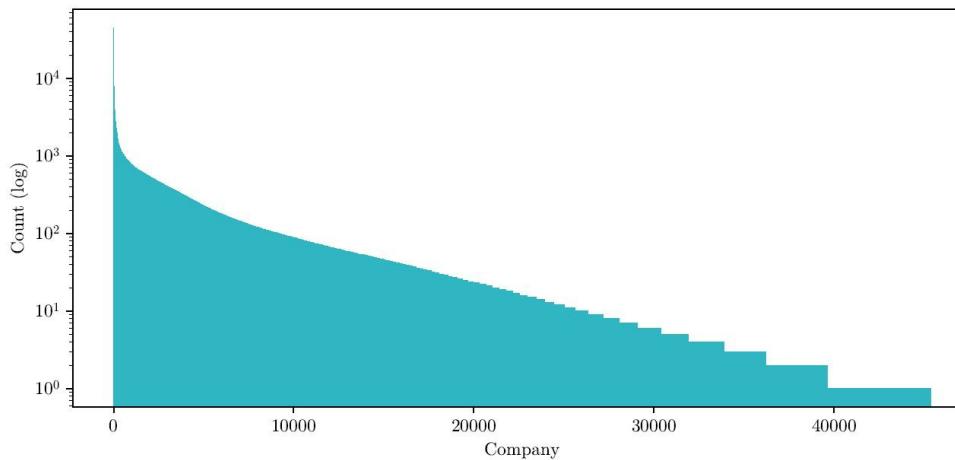
Company count distribution (2y)



Company count distribution (6m)



Company count distribution (1y)



Our proposal: SentenceBERT

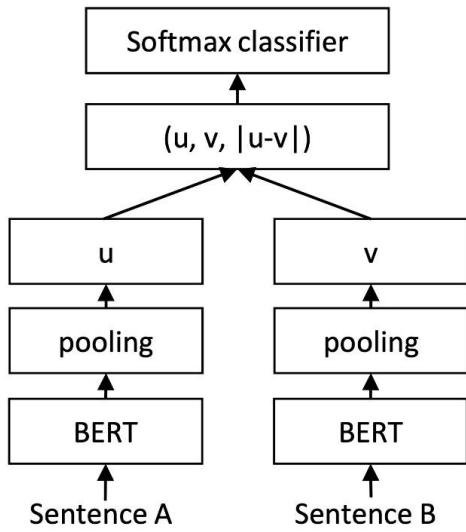


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

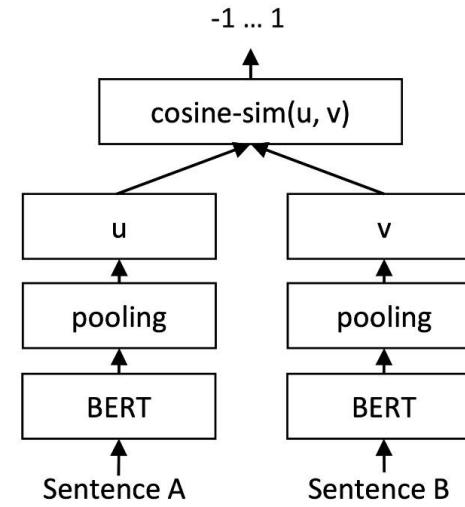


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

First iteration

- **FinEAS:** Use Sentence Bert as feature extractor.



- BERT with same strategy as above.
- Bidirectional LSTMs.

First iteration results

FinEAS yields positive results as it adapts better to the data.

Simple BiLSTM is similar to BERT and is sometimes better.

It would be interesting to know how we can improve adaptation.
Let's unfreeze!

	FinEAS	BERT	BiLSTM
6 months	0.0556	0.2124	0.2108
6 months N2Ws	0.1061	0.2190	0.2194
1 year	0.0654	0.2137	0.2140
1 year N2Ws	0.1058	0.2191	0.2194
2 years	0.0671	0.2087	0.2086
N2Ws	0.1065	0.2188	0.2185

Second iteration and results

Unfreezing...
& competitors
What is FinBERT:
Financial Phrasebank

Dates	FinEAS	FinBERT
6 months	0.0044	0.0050
1 year	0.0036	0.0034
2 years	0.0033	0.0040

Conclusions

- Although RMSE might be small, we have a LARGE test set. The difference is therefore LARGE.
- The more historical data the better.
- Model lacks capacity when training.
 - Recall: larger models are better models but...
 - ...recall: hungry models (😭).
- We know that the model perform well for the next two weeks but there might be changes on the market.

Future work



- Look for open repositories.
- Annotation?
- A production ready architecture.
 - Based on case based reasoning?
- Further exploit other variables that are present in the dataset (e.g. relevance).
- Word analysis.
- Company based sentiment.