Natural Language Processing in Finance

May 2023

Artificial Intelligence Finance Institute

NYU Tandon

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, QI 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-Estapé

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

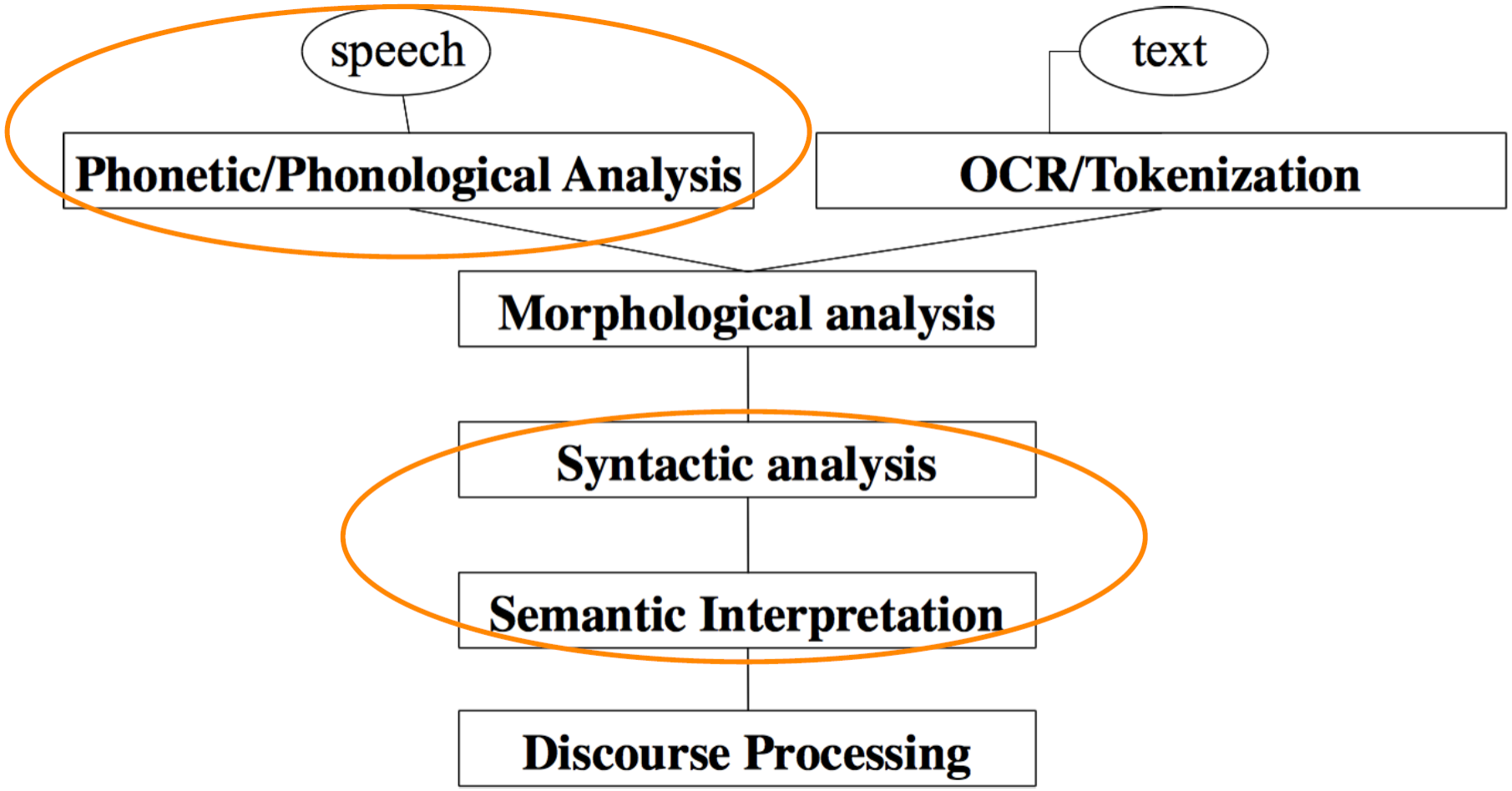
Applications range from simple to complex:

- 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

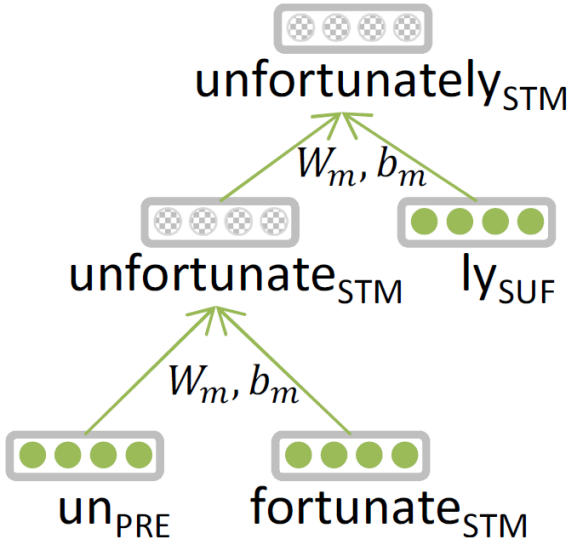


Representations of NLP Levels: Morphology

- Traditional: Words are made of morphemes

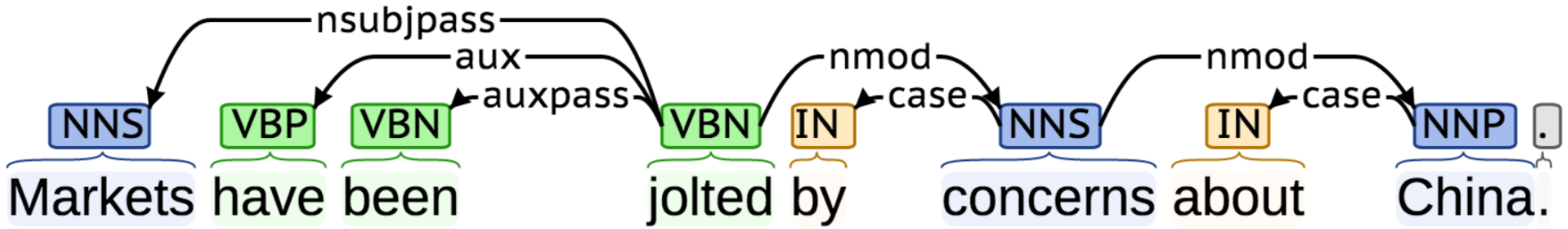
prefix	stem	suffix
un	interest	ed

- 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



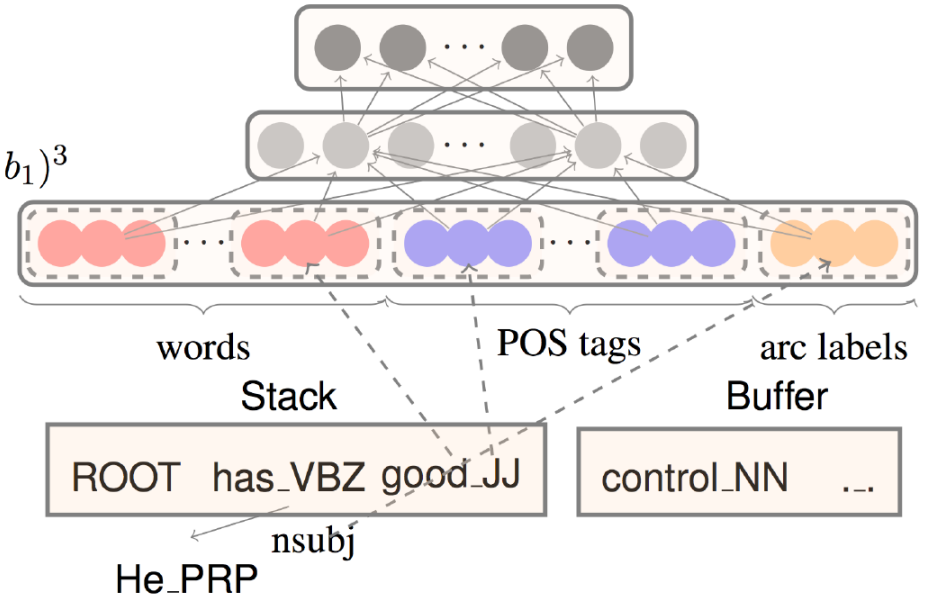
Softmax layer:

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

Hidden layer:

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

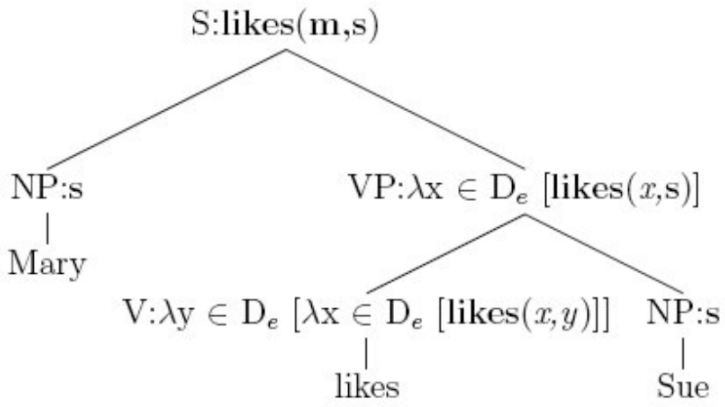
Input layer: $[x^w, x^t, x^l]$

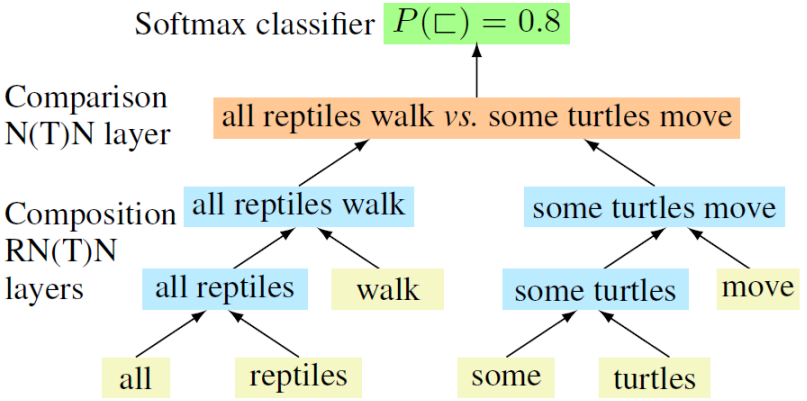


Configuration

Representations of NLP Levels: Semantics

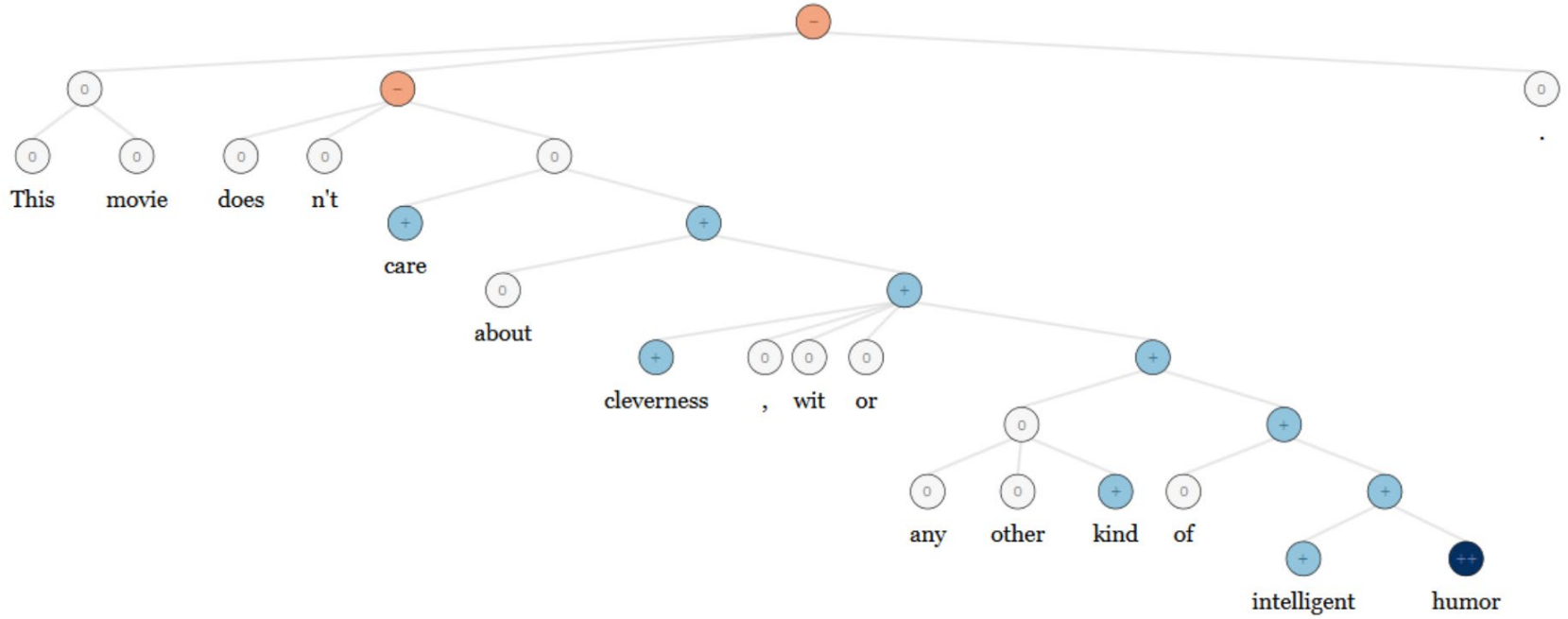
- Traditional: Lambda calculus
  - Carefully engineered functions
  - Take as inputs specific other functions
  - No notion of similarity or fuzziness of language
- DL:
  - Every word and every phrase and every logical expression is a vector
  - a neural network combines two vectors into one vector
  - 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 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 & GATECH	MT-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	9

NLP Tools

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

https://huggingface.co/

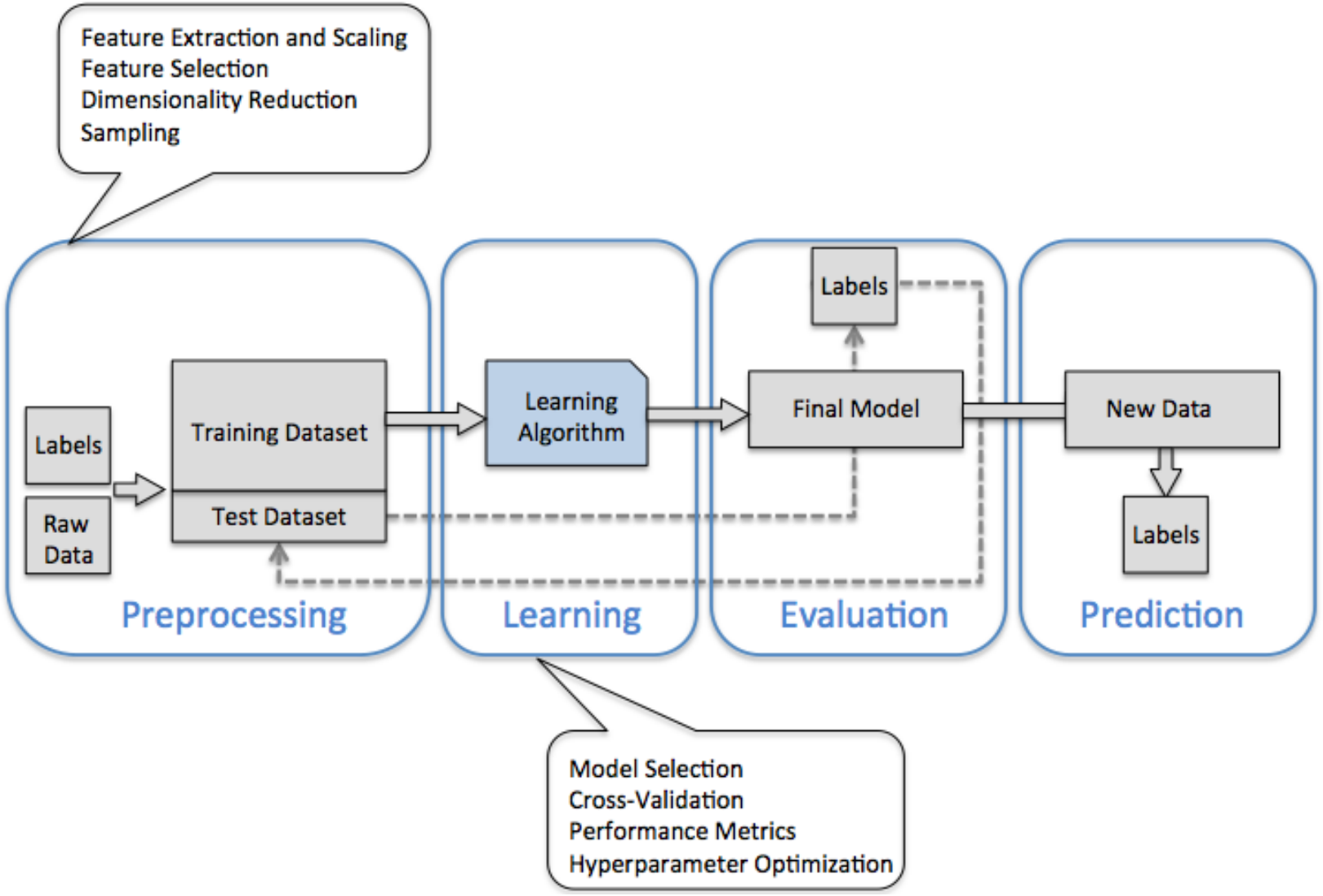
https://www.tensorflow.org/tutorials

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

Natural Language Processing in Finance

Essentials

Machine Learning Modeling



WORKFLOW SUMMARY OUTLINE

1 - BIG DATA EXTRACTION

1. ESG / SDG Taxonomies
2. Text Data in 60 Languages from 50,000+ sources
3. Geo-referenced Data from 150+ Countries with a focus on North America, EMEA, APAC and Emerging Markets.
4. Company Annual and Sustainability Reports
5. News and Blogs
6. Social Media and Twitter
7. Wide range of curated Websites
8. SEC filings
9. Industry-specific, Independent and NGO reports
10. Government & Multilateral Data

2 - GLOBAL AI NLP MODELS

1. Language Detection and Translation
2. Topic Modeling
3. Entity Recognition
4. Text Pre-Processing
    1. Tokenization
    2. Remove Stop Words
    3. Lemmatization
    4. Normalization
5. Domain-specific Dictionary generation & curation
6. Feature Extraction – Frequency Based
    1. TF-IDF
    2. Bag of Words
7. Parsing
    - Part of Speech Tagging
    - Syntactic Parser
    - Phrase Tagging
8. Deep Learning Models
    1. Word Embeddings
    2. Recurrent Neural Networks
    3. Convolutional Neural Networks
    4. Transformers

3 - OUTPUT & ANALYTICS

1. Daily ESG / SDG Scores
    1. Positive and Negative News Tones
    2. Z-scoring by SDG
    3. Short-Term and Long-Term Indicators
    4. Uses Volume/Dispersion
    5. Sparsity analysis
    6. ESG/SDG Matrix
    7. Country, Sector and Company-specific Indicators
2. Outlier Analysis
3. Word Clouds
4. Text Summarization
5. Dashboards

Historical Evolution Sentiment Analysis

Financial Phrase Bank[

LEXICON METHODS	EMBEDDINGS	DEEP LEARNING
Single dimension	Multiple dimensions	Multiple dimensions
Single layer	Single layer	Multiple layers
61% Accuracy*	**78% Accuracy***	**97% Accuracy***

*On financial phrases with 100% Agreement [1]

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 miss spellings 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 frequences 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.

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

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:

- $f_{t,d}$ / (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:

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

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 connoted 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 Connoted Words - they arrive at their sentiment word lists containing unambiguous and singularly connoted 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

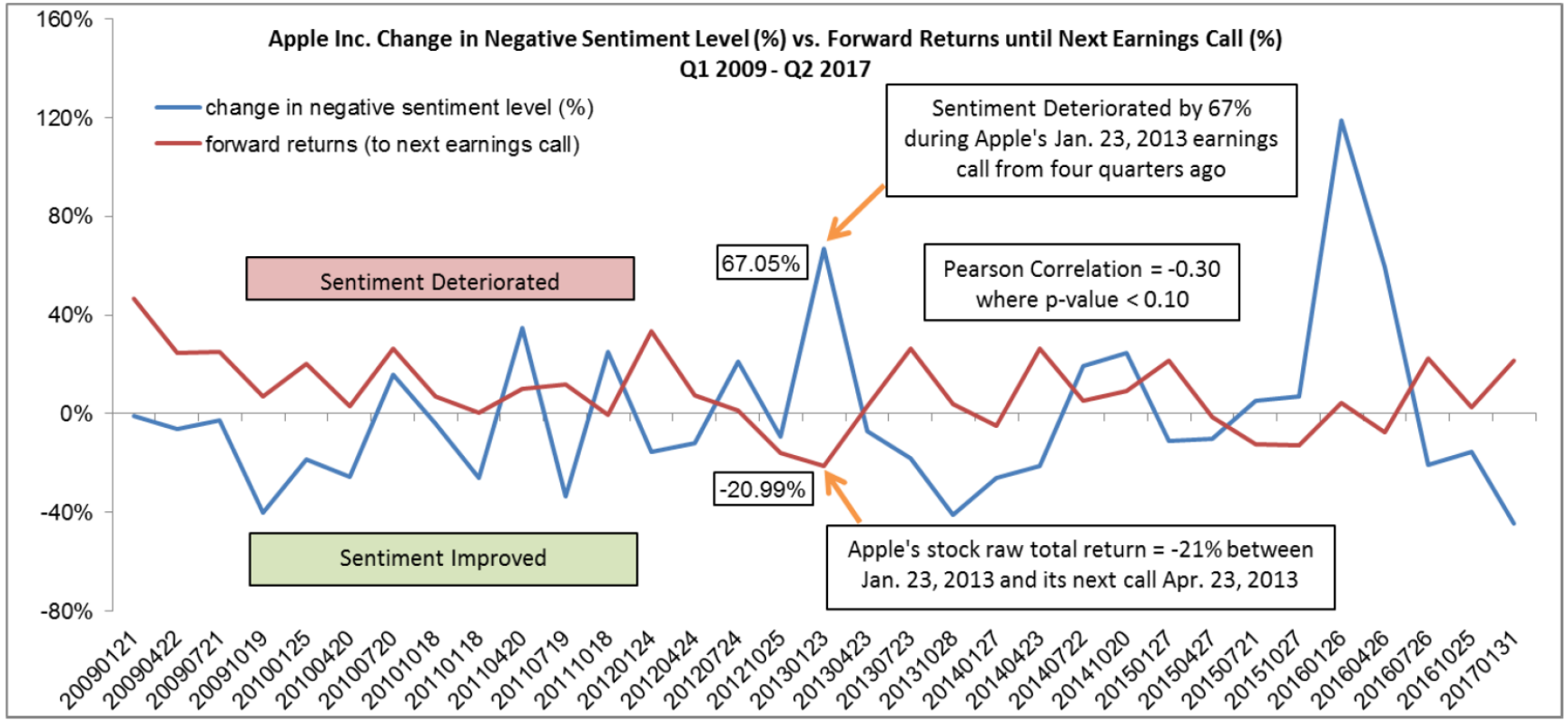
The first use case is to capture the historical relationship of Apple Inc.'s sentiment level changes from its earnings calls and its forward returns until the next call (Exhibit next slide). The changes in sentiment are defined as quarter-over-quarter (QoQ) changes from four quarters ago (to account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

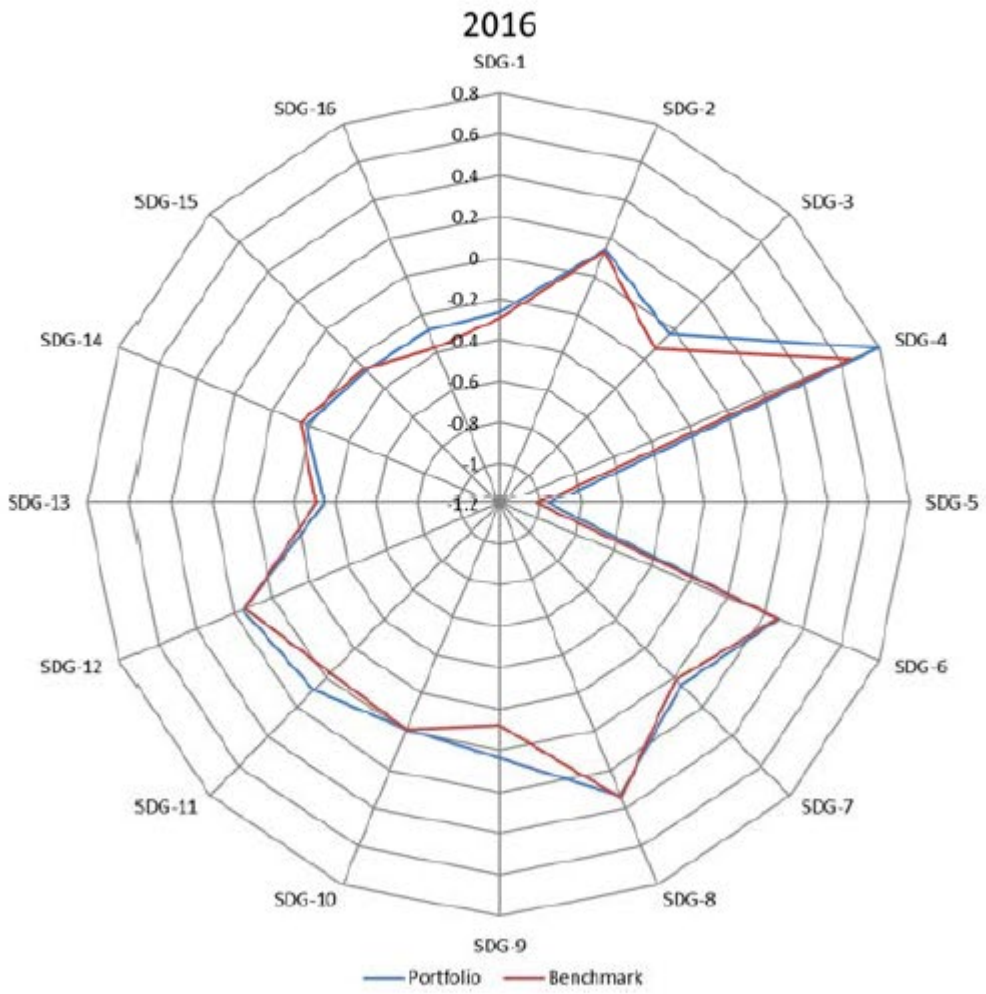
This is partly due to the perceived conflict of ESG investing with an asset manager's fiduciary duty and partly due to low-quality ESG data despite the near ubiquity of sustainability reports. We analyze the relationship between alpha generation and ESG metrics, and measure the impact companies have 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.'

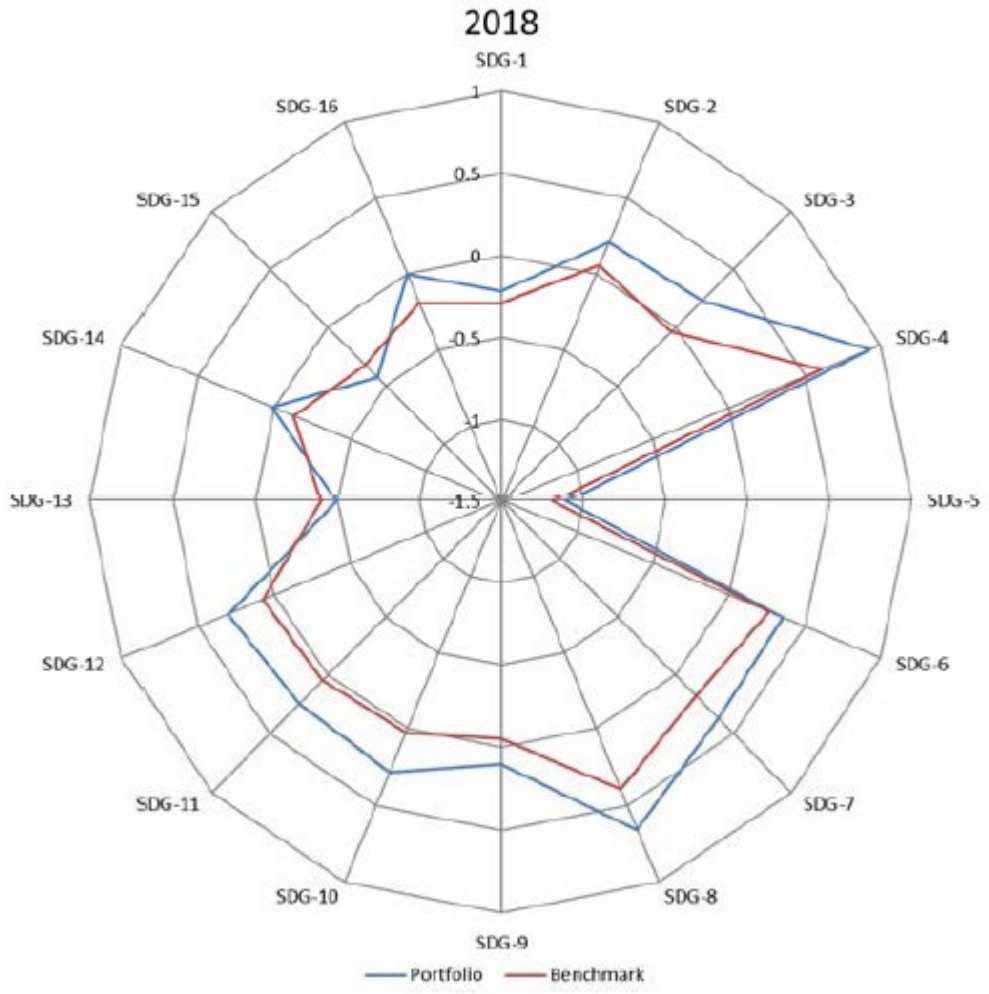
We show that an ESG momentum portfolio both outperforms the MSCI US index and has a relatively better SDG footprint than that of the index. Third, we establish a positive contemporaneous connection between the portfolio's ESG ratings momentum and its SDG footprint. Thus, a positive linkage exists between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

$$Sim\_Cosine = \frac{D_1^{TF} \cdot D_2^{TF}}{||D_1^{TF}|| \times ||D_2^{TF}||}$$

D_A: We expect demand to increase.

D_B: We expect worldwide demand to increase.

Step 1: Define our Texts

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

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

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

Step 3: Turn our texts into vectors

$$Sim\_Cosine = \frac{D_1^{TF} \cdot D_2^{TF}}{||D_1^{TF}|| \times ||D_2^{TF}||}$$

$$= \frac{(1 \times 1 + 1 \times 1 + 0 \times 1 + 1 \times 1 + 1 \times 1 + 1 \times 1)}{(\sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2}) \times (\sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2})}$$

$$= 0.91$$

Step 4: Follow the formula to calculate cosine similarity!

Quick intro to cluster demeaning

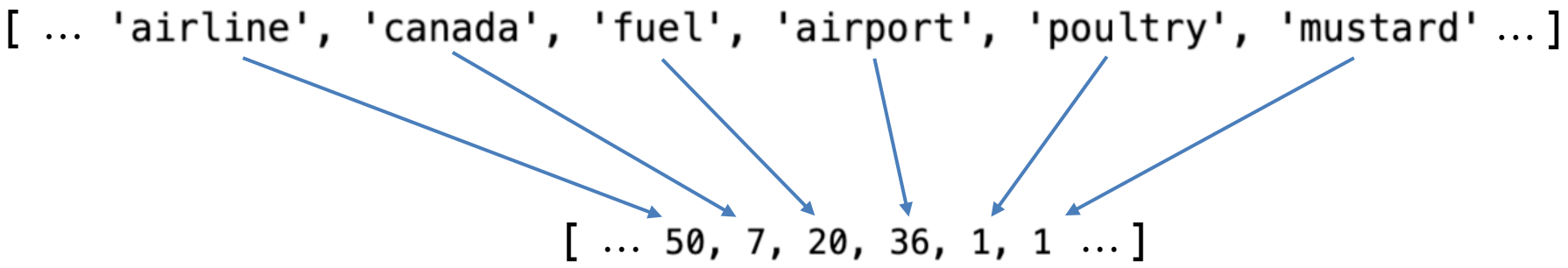
- Since not all industries have the same business practices, the returns and signals for companies tend to group together
- Want to find these clusters and control for them via cluster demeaning
- Can do so using Item 1: Business by embedding these descriptions into a vector and calculating pairwise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

Quick intro to cluster demeaning

Superset of all words used (cleaned)

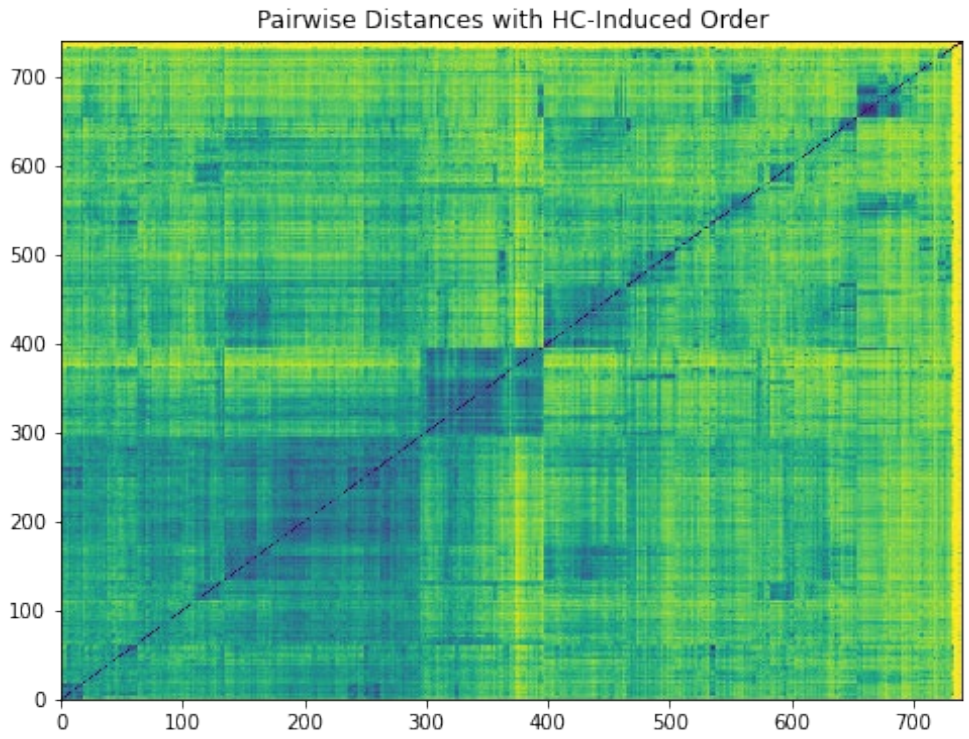
Create a vector containing the frequency of each word in superset within the business description

Do this for each business description, for each applicable date



filingDate							
2021-07-29	[1, 1, 1, 57, 50, 7, 20, 36, 1, 1, 1, 7, 7, 6,...	[0, 0, 1, 0, 0, 0, 7, 0, 1, 1, 0, 2, 2, 0, 1, ...	[0, 0, 0, 0, 0, 0, 4, 0, 0, 1, 0, 0, 0, 0, 0, ...	[1, 1, 1, 0, 0, 0, 3, 0, 2, 1, 0, 12, 0, 0, 1,...	[1, 1, 1, 0, 0, 0, 4, 0, 1, 1, 1, 1, 1, 0, 0, ...	[0, 0, 2, 0, 0, 2, 13, 0, 2, 2, 0, 15, 3, 5, 2...	[0, 1, 0, 0, 0, 31, 12, 0, 1, 2, 3, 7, 1, 1, 7...

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

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

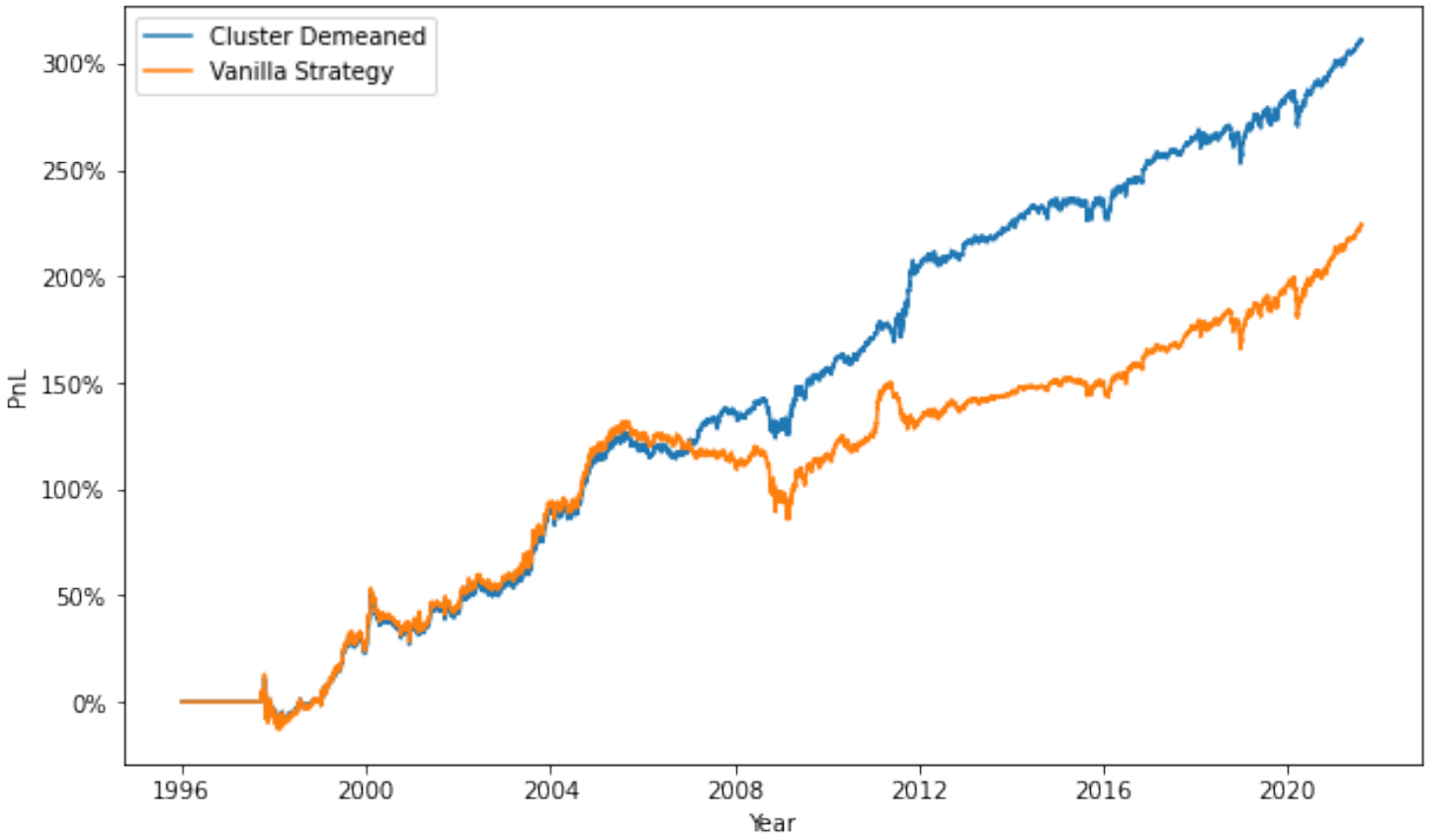
Step 5: Multiply by 2 and subtract 1

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

$$\sum_i p_i = 0$$

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

Constructing a portfolio using cosine similarity



	De-Meaned	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

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

Word meaning as a neural word vector – visualization

expect =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

Word2vec: Overview

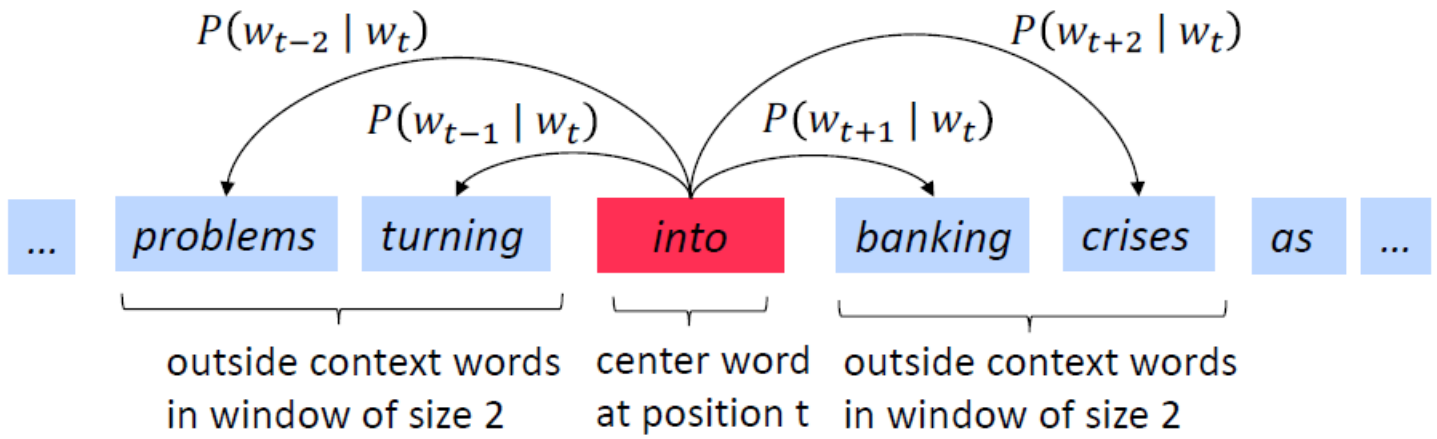
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

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 $P(w_{t+j} \mid w_t)$



Word2vec: objective function

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

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

θ is all variables to be optimized

sometimes called *cost* or *loss* function

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

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

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

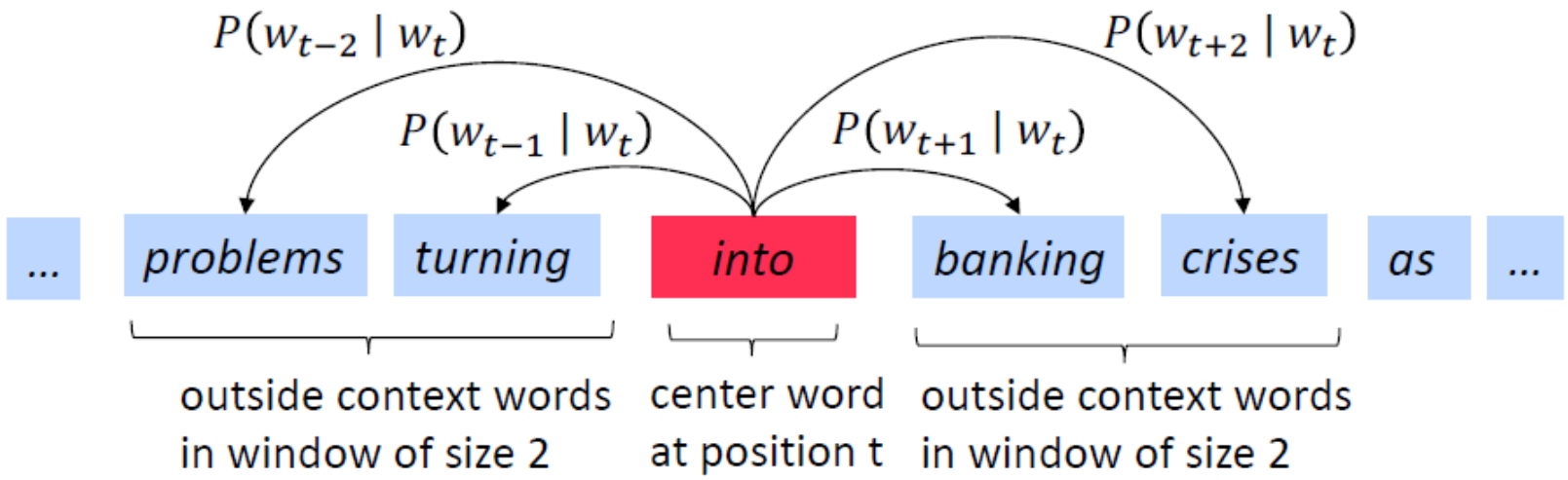
$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- **Question:** How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?
- **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 $P(w_{t+j} \mid w_t)$



Word Embeddings - Glove

Word Embeddings

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

In [2]:
```
embedding_file = 'data/glove.6B.300d.txt'

df_embedding = pd.read_csv(embedding_file, sep=' ', index_col=0,
                           header=None, na_values=None,
                           keep_default_na=False,
                           quoting=csv.QUOTE_NONE)
df_embedding.sort_index(inplace=True)
```

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

Out[3]: (400000, 300)

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

Out[4]:

0	1	2	3	4	5	6	7	8	9	10	...	291	292	293	2
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	-0.595(
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	0.238
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	-0.506
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	-0.000
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	-0.131
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	0.371(
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	0.029
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	-0.611
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	-0.449
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	0.336

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

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

Out[6]: 0.5643459466769057

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

Out[7]: 0.025038104793071447

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

```

Out[8]: -0.6228158930510513

Word Embeddings – Word Analogy

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

In [10]:
```
def word_analogy(a, b, c):
    emb_a, emb_b, emb_c = df_embedding_small.loc[a], \
                          df_embedding_small.loc[b], \
                          df_embedding_small.loc[c]
    max_sim = -100
    result = None
    for w, emb_w in df_embedding_small.iterrows():
        if w in [a, b, c]:
            continue
        sim = cosine_similarity(emb_b - emb_a, emb_w - emb_c)
        if sim > max_sim:
            max_sim = sim
            result = w
    return result
```

In [11]:
```
test_cases = [
    ('paris', 'france', 'london'),
    ('paris', 'france', 'rome'),
    ('paris', 'france', 'madrid'),
    ('paris', 'france', 'berlin'),
    ('man', 'king', 'woman'),
    ('walk', 'walked', 'go'),
    ('man', 'woman', 'boy'),
    ('apple', 'juice', 'mother')
]

for t in test_cases:
    r = word_analogy(t[0], t[1], t[2])
    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

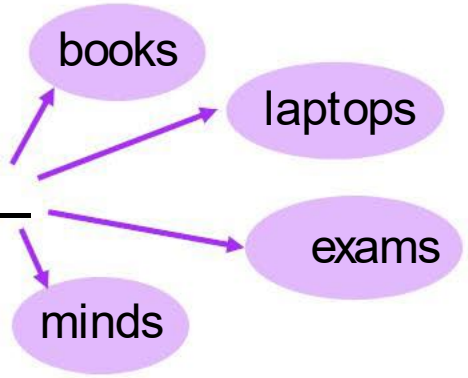
https://www.tensorflow.org/text/guide/word_embeddings

Emergent linguistic structure in deep contextual neural word representations

1. Language Modeling

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

the students opened their _____



For a word sequence $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, it gives the probability of $x^{(t+1)}$:

$$P(x^{(t+1)} | x^{(t)}, \ldots, x^{(1)})$$

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

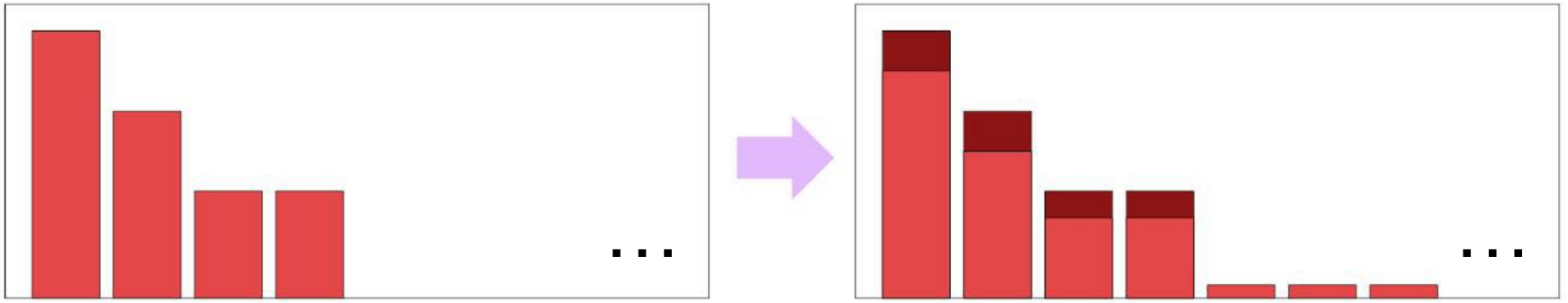
Count how often words follow word sequences; divide to get cond. prob.

Classic curse of dimensionality scenario: zillions of params

Markov assumption:

$$P(\text{the} \mid \text{President Trump denied the}) \approx P(\text{the} \mid \text{denied the})$$

Discounting/Smoothing



Mixture/Backoff

$$P_{KN}(w_i \mid w_{i-1}, w_{i-2}) \approx \lambda P(w_i \mid w_{i-1}, w_{i-2}) + (1-\lambda) P(w_i \mid w_{i-1})$$

Enlightenment era neural language models (NLMs)

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

$$' \left(□^{(\%\&\ )} \middle| □^{(\%)}, □^{(\%@\ )}\right) = \text{softmax}(\text{FFNN}(>^{(\%)}, >^{(\%@\ )}))$$

2. Solve failure to exploit long contexts via recurrent NN or Transf

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

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

$$' \left(□^{(\%\&\ )} \middle| □^{(H\%)}\right) = \text{LSTM}(h^{(\%)}, □^{(\%)})$$

A RNN Language Model

output distribution

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

hidden states

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

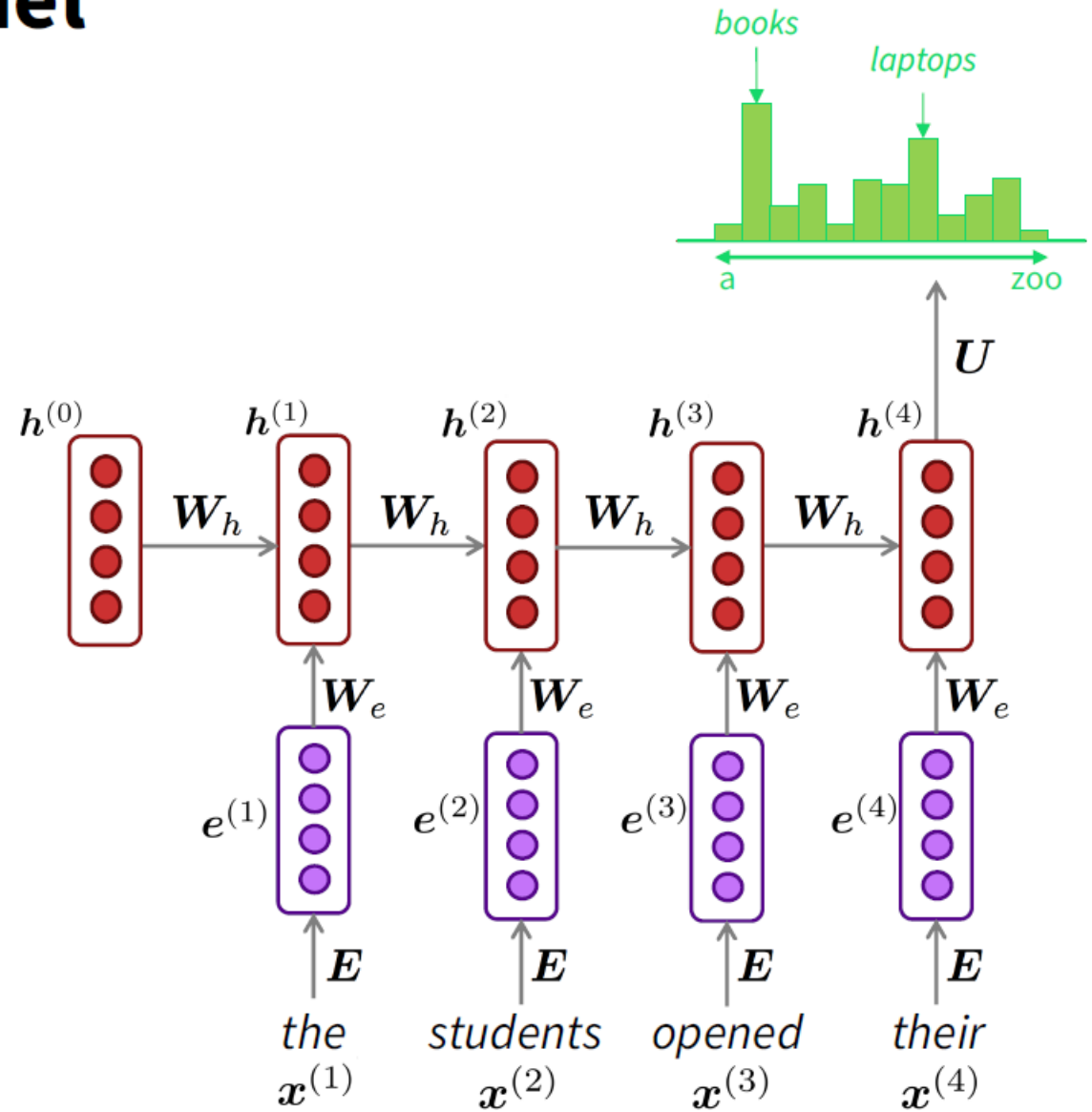
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

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

words / one-hot vectors

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



Long Short-Term Memory (LSTM)

For LSTM:

Input Gate

Forget Gate

Output Gate

Memory State

Cell State

$$
\begin{aligned}
\boldsymbol{i}_t &:= \operatorname{sigm}(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1})\,, \\
\boldsymbol{f}_t &:= \operatorname{sigm}(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1})\,, \\
\boldsymbol{o}_t &:= \operatorname{sigm}(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1})\,, \\
\tilde{\boldsymbol{c}}_t &:= \tanh(\boldsymbol{W}_{xc}\boldsymbol{x}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1})\,, \\
\boldsymbol{c}_t &:= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t\,, \\
\boldsymbol{h}_t &:= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)\,.
\end{aligned}
$$

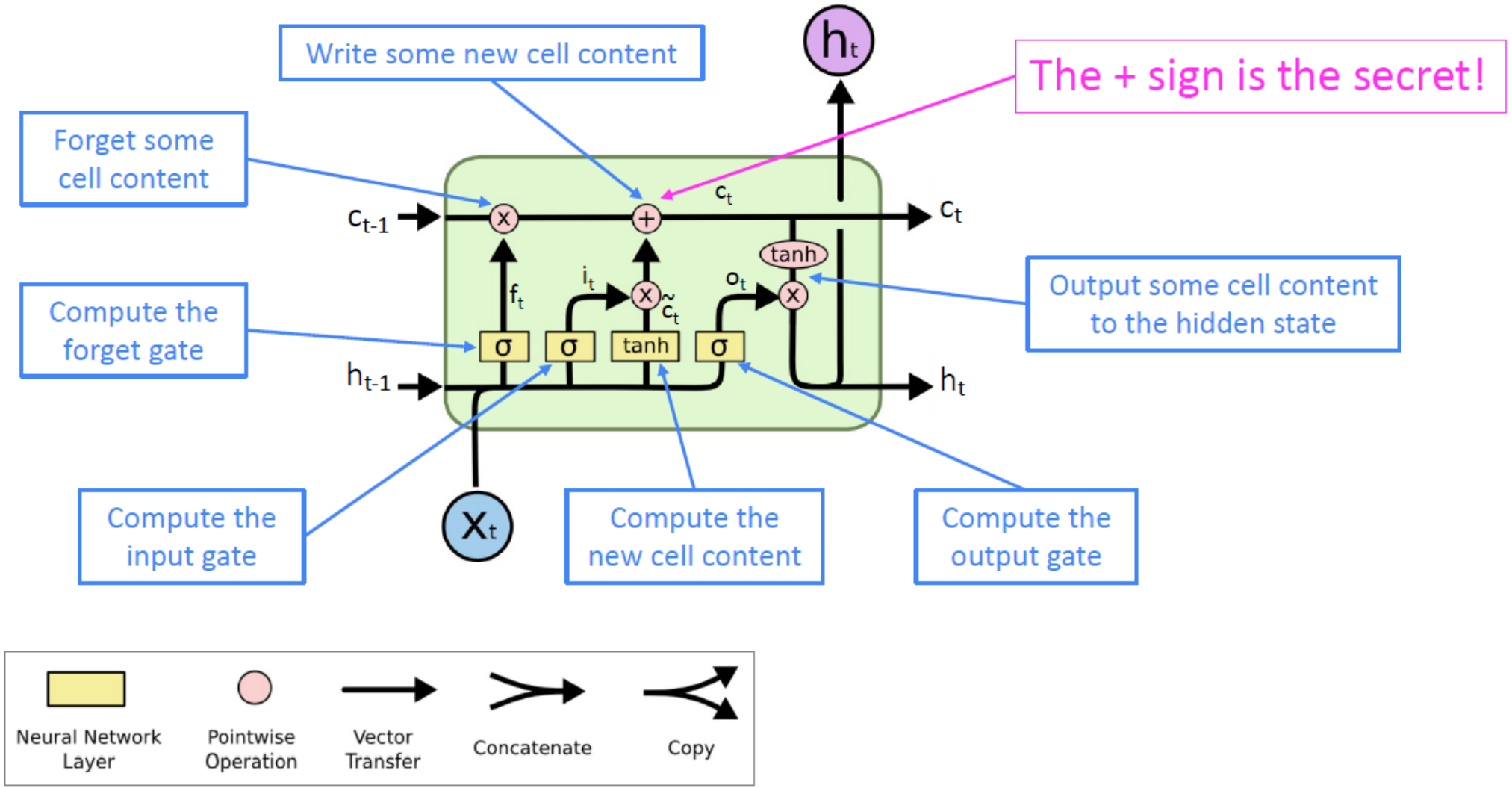
Similarly for RNN, we have

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

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

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

hidden states

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

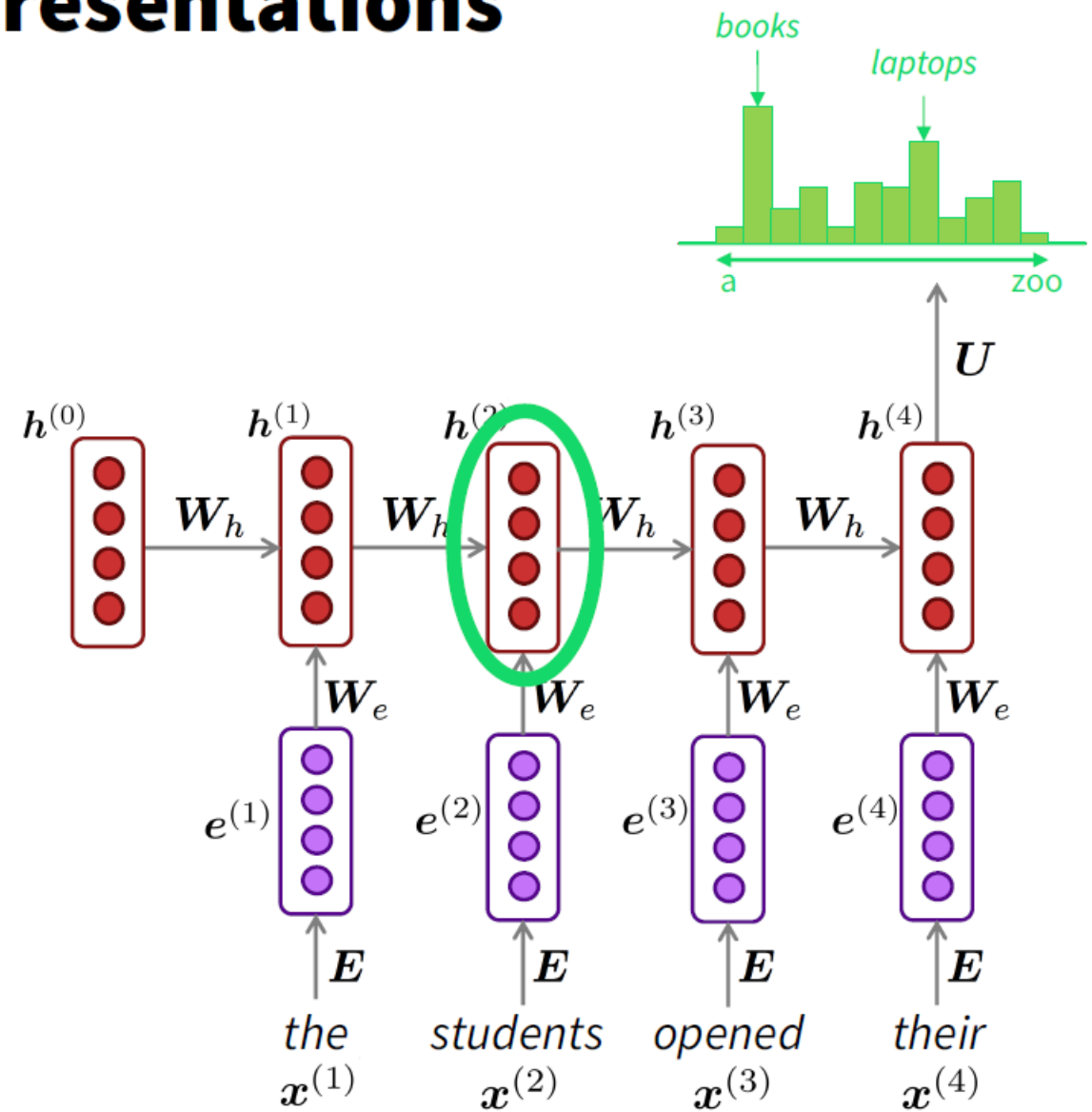
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

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

words / one-hot vectors

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



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
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 on

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

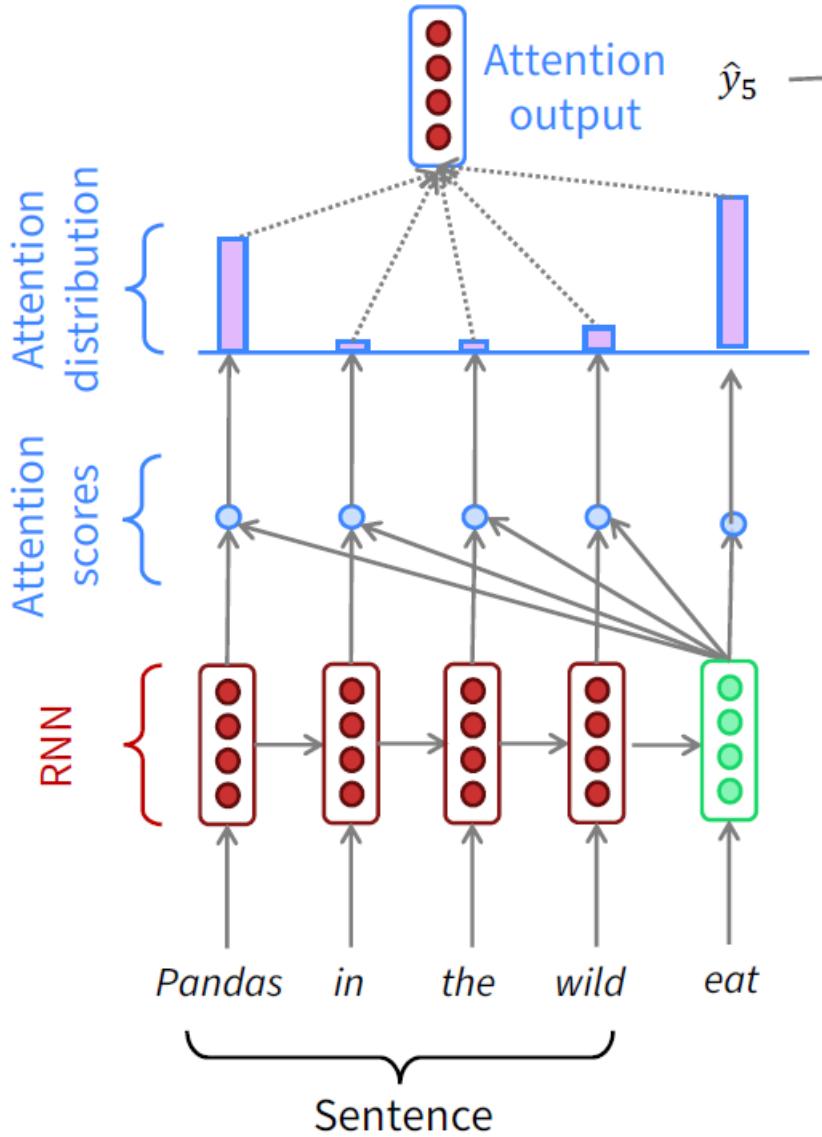
Recurrent models with (self-)attention

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

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

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

Bilinear attention



Weighted average score

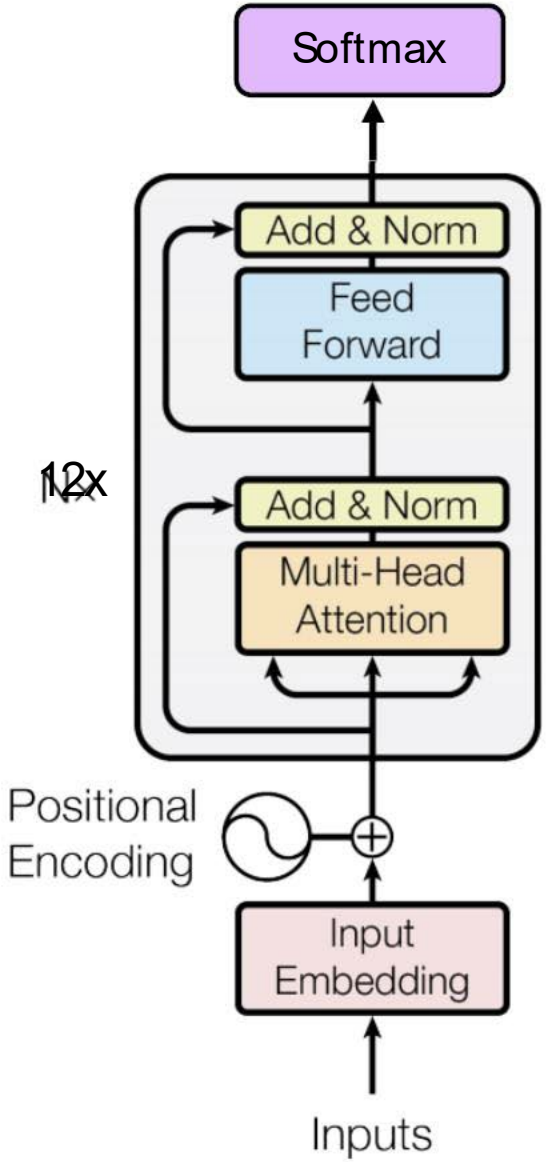
Similarity score with past hidden states

Hidden states key as a hash key to look back in the past most relevant

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

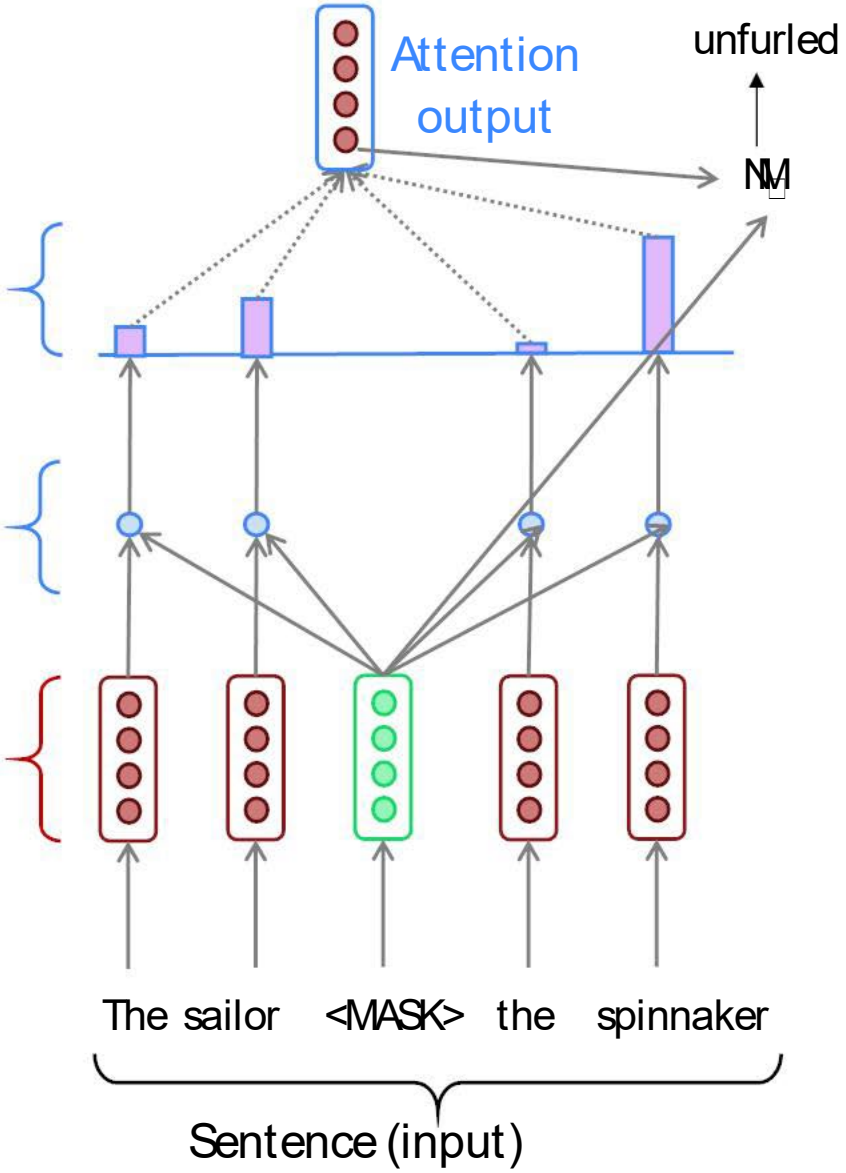
Pre-training uses a cloze task formulation where 15% of words are masked out and predicted:

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 $P^{\%}$ for step t

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

We take softmax to get the attention (prob.) distribution $Q^{\%}$ for step t

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

We use $Q^{\%}$ to take weighted sum of the hidden states to get attention output

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

Finally we join (sum or concatenate) the attention output $R^{\%}$ with the decoder hidden state $S_{\%}$ 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

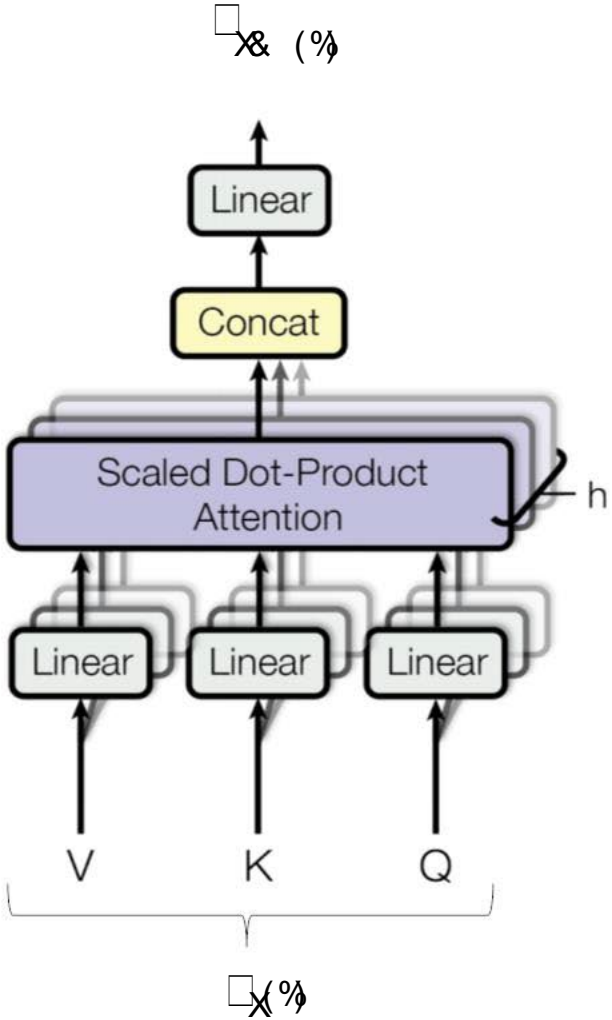
Map input into $h = 12$ many lower dimensional spaces via U_V matrices

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

$$\text{Multihead}(X^{(i)}) = \text{Concat}(head_1, \ldots, head_h)U^O$$

$$head_i = \text{Attention}(XU_i^Q, XU_i^K, XU_i^V)$$

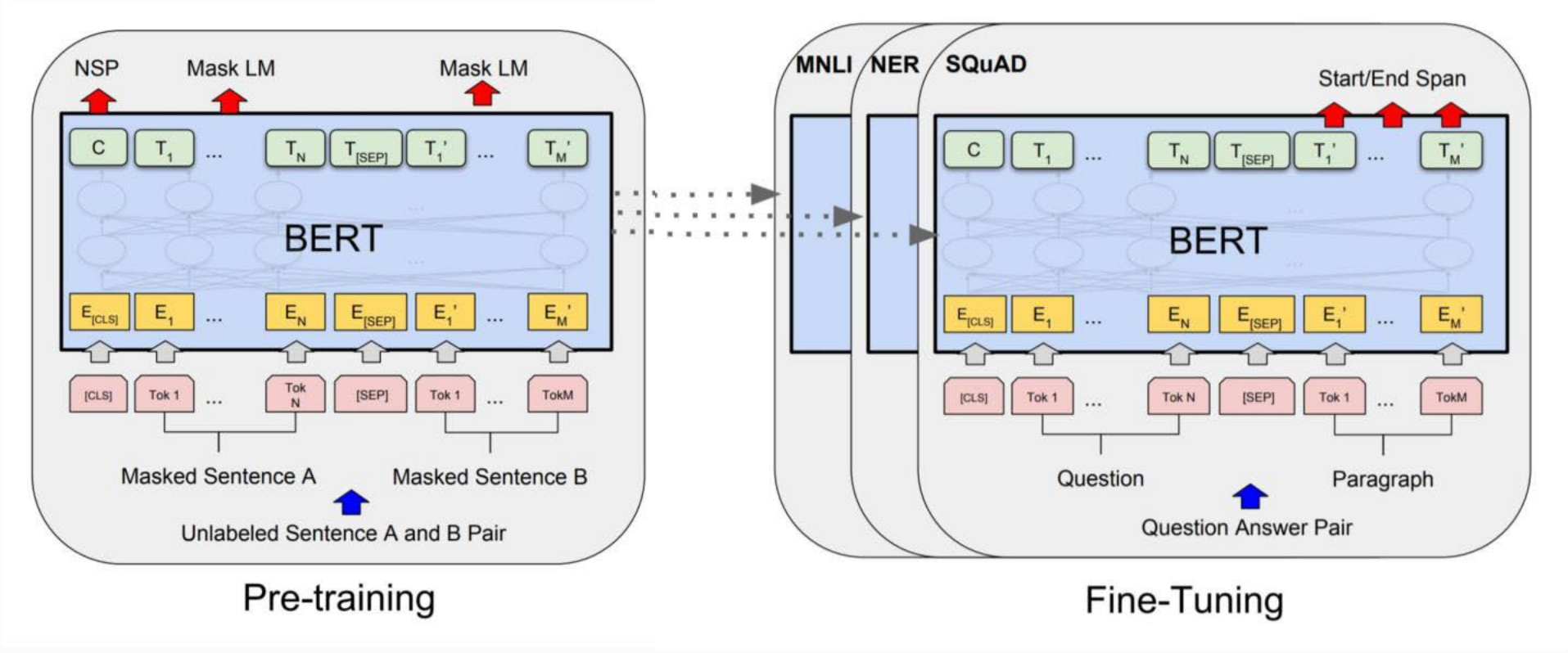
So attention is like bilinear: $X^{(i)}(U_i^Q(U_i^K)^a)X^{(b)}$



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



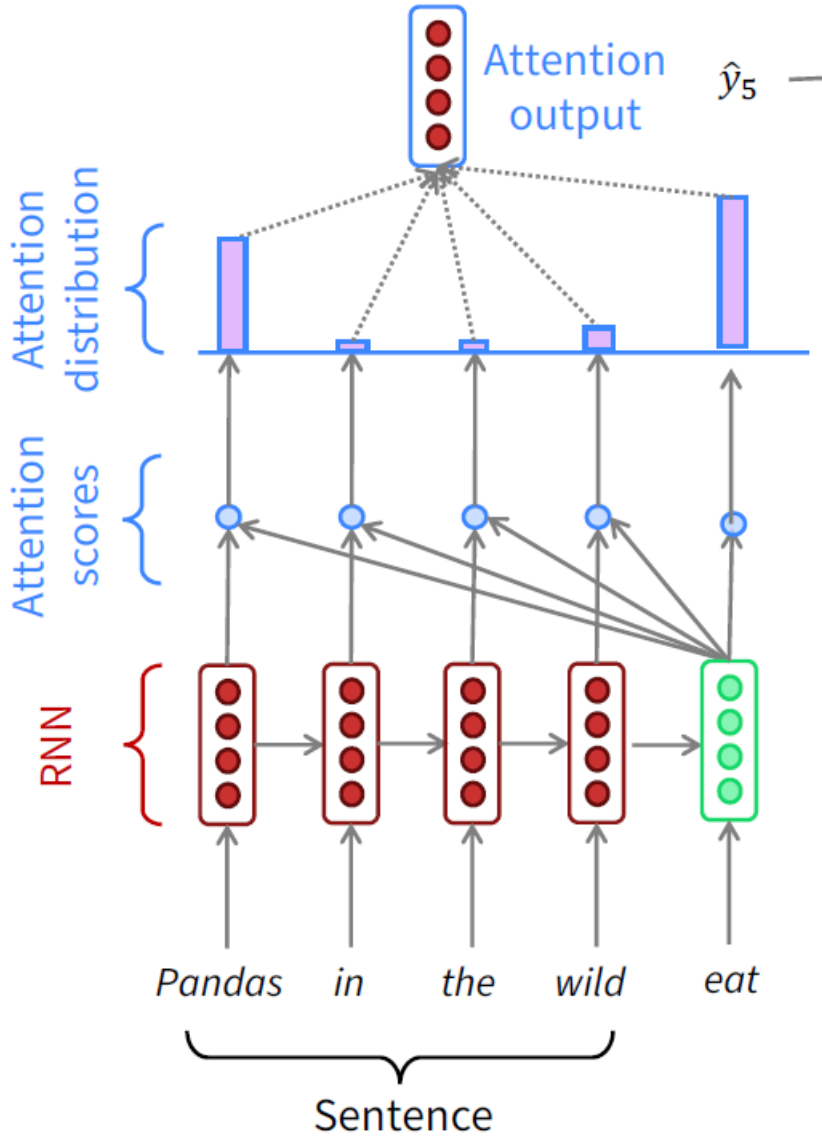
Recurrent models with (self-)attention

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

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

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

Bilinear attention



Weighted average score

Similarity score with past hidden states

Hidden states key as a hash key to look back in the past most relevant

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:

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

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:

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

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. $\mathbf{A} = \mathbf{softmax}(\mathbf{Q}\mathbf{K}^\top / \sqrt{d_k})$.
$a_{ij} \in \mathbf{A}$	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, $\overrightarrow{p_t} \in \mathbb{R}^d$ be its corresponding encoding, and d be the encoding dimension (where $d \equiv_2 0$) Then $f : \mathbb{N} \to \mathbb{R}^d$ will be the function that produces the output vector $\overrightarrow{p_t}$ and it is defined as follows:

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

where

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

Positional Encoding

You can also imagine the positional embedding $\overrightarrow{p_t}$ as a vector containing pairs of sines and cosines for each frequency (Note that d is divisble by 2):

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

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

In [13]:
```
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]:

```
# load saved tokenizer?
if True:
    with open('model/tokenizer.pkl', 'rb') as f:
        tokenizer = pickle.load(f)
    df_embedding = df_embedding.reindex(tokenizer.vocab.keys())
    df_embedding.sort_index(inplace=True)
    df_embedding.loc['<START>'] = np.zeros(df_embedding.shape[1])
    df_embedding.loc['<UNK>'] = df_embedding.mean()
    df_embedding.loc['<NUMBER>'] = np.ones(df_embedding.shape[1])/2
    df_embedding.loc['<PAD>'] = np.ones(df_embedding.shape[1])

else:
    tokenizer = FullTokenizer(df_embedding.index.values)

    def process(arr):
        for i in map(tokenizer.tokenize, arr):
            tokens.update(i)
        return tokens

    tokens = set()
    if __name__ == '__main__':
        p = mp.Pool(processes=mp.cpu_count())
        split_arr = np.array_split(news_data.TEXT.values, mp.cpu_count())
        pool_results = p.map(process, split_arr)
        p.close()
        p.join()

        all_tokens = set.union(*pool_results)

    all_tokens = set([t for t in all_tokens if not is_number(t)])
    df_embedding = df_embedding.reindex(all_tokens).dropna()
    df_embedding.sort_index(inplace=True)
    df_embedding.loc['<START>'] = np.zeros(df_embedding.shape[1])
    df_embedding.loc['<UNK>'] = df_embedding.mean()
    df_embedding.loc['<NUMBER>'] = np.ones(df_embedding.shape[1])/2
    df_embedding.loc['<PAD>'] = np.ones(df_embedding.shape[1])

tokenizer = FullTokenizer(df_embedding.index.values)
```

Loading and Tokenizing

Tokenization example

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

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

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

Out[17]:

0	1	2	3	4	5	6	7	8	9	10	...	291	292	293	294
<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	-0.02451
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	-0.14840
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	0.32683
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	0.34706
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	-0.11025
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	0.12968
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	0.23099
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	0.45064
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	0.07297
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	0.34228
.	-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	0.23445

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]:
```
def encode(string):
    encoding = ["<START>"] + tokenizer.tokenize(string)
    encoding = tokenizer.convert_tokens_to_ids(encoding)
    return encoding

def decode(ids):
    tokens = tokenizer.convert_ids_to_tokens(ids)
    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]:
```
def process(df):
    res = df.TEXT.apply(encode)
    return res

if __name__ == '__main__':
    p = mp.Pool(processes=mp.cpu_count())
    split_dfs = np.array_split(news_data, mp.cpu_count())
    pool_results = p.map(process, split_dfs)
    p.close()
    p.join()

    # merging parts processed by different processes
    dataset = pd.concat(pool_results, axis=0)
```

Preview the dataset:

In [20]:
```
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]:

```
neurons = 128
dropout = 0.40

my_embedding = layers.Embedding(input_dim=df_embedding.shape[0],
                                output_dim=df_embedding.shape[1],
                                input_length=maxlen,
                                trainable=False)

my_embedding.build((None, ))
my_embedding.set_weights([df_embedding.values])

model = keras.Sequential([
    my_embedding,
    layers.LSTM(neurons, return_sequences=True),
    layers.Dropout(dropout),
    layers.LSTM(neurons, return_sequences=False),
    layers.Dropout(dropout),
    layers.Dense(1, activation='tanh')
])

optimizer = tf.optimizers.Adam(learning_rate=0.001)

def r2_keras(y_true, y_pred):
    SS_res = K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return (1 - SS_res/(SS_tot + K.epsilon()))

model.compile(loss='mean_squared_error',
                    optimizer=optimizer,
              metrics=[r2_keras])
model.summary()

```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
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
=================================================================
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 iAlonso

Petter Kolm

Jordi Armengol-Estapé

https://arxiv.org/abs/2111.00526

What is Sentiment

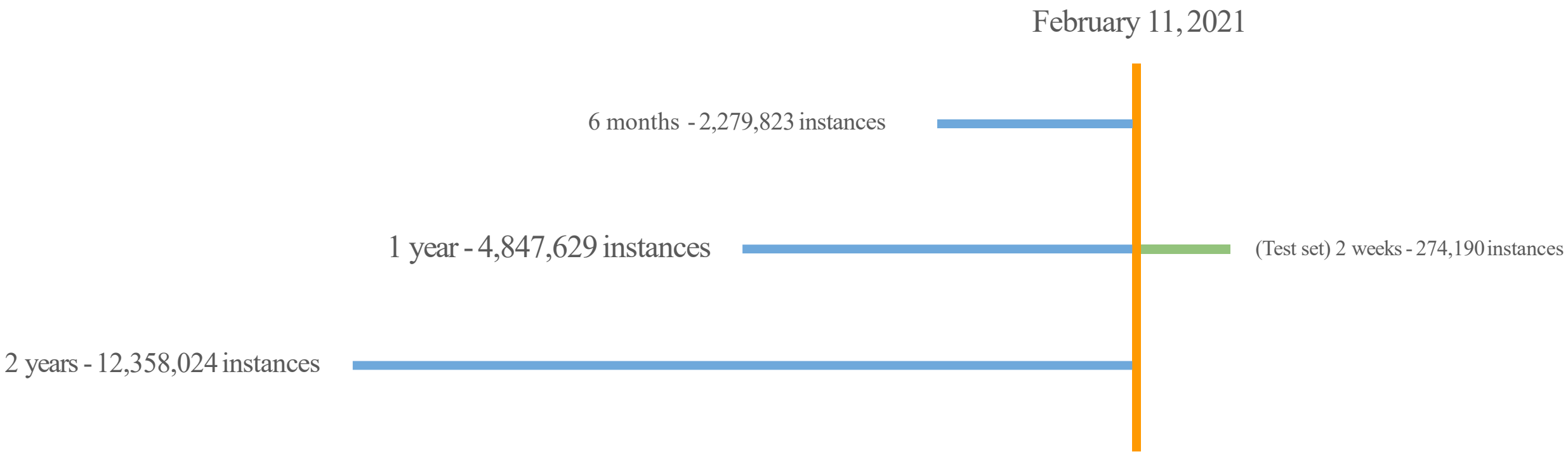
Sentiment is a very difficult task!

- **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 as neutral.
- **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.

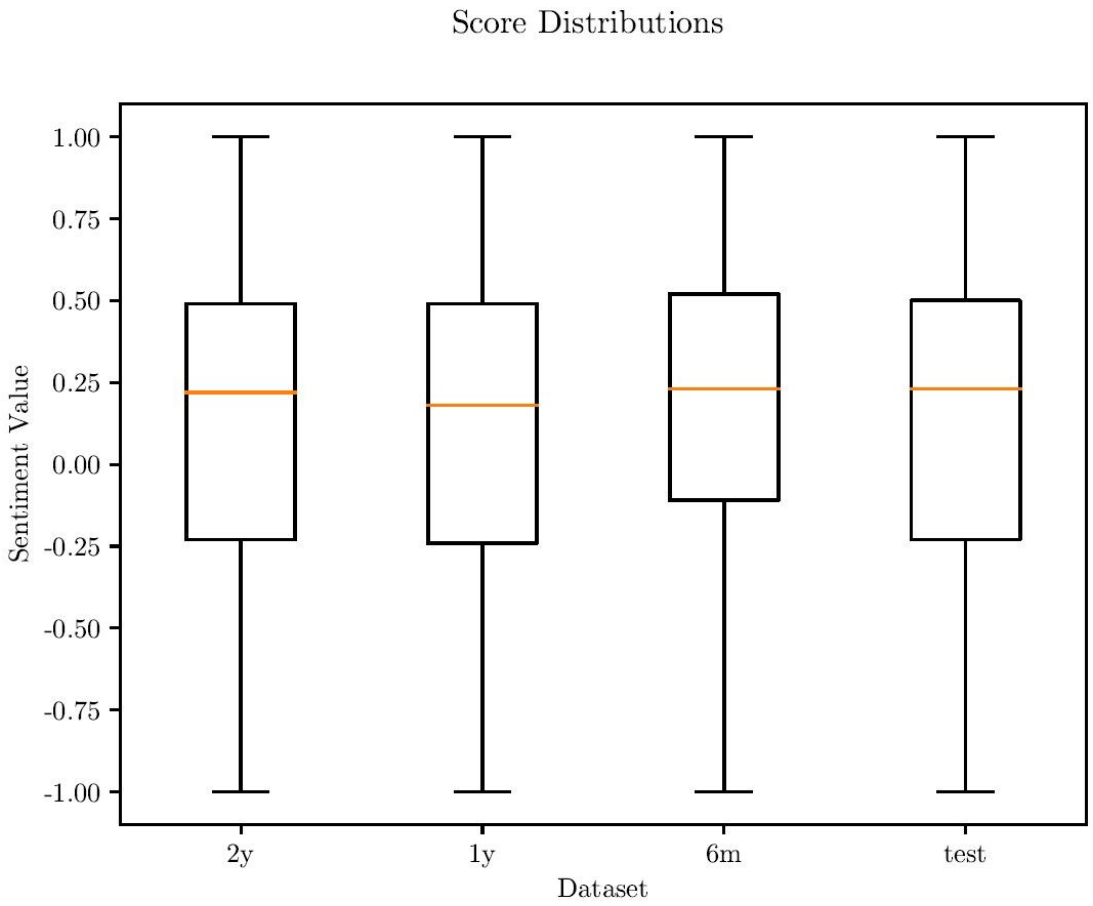
Min and maxmet.

Nice boxes.

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

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

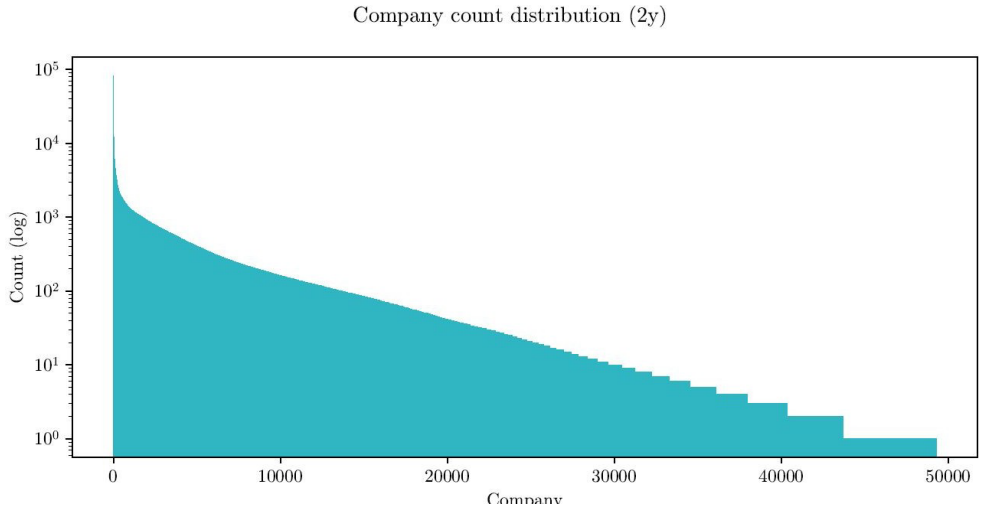
✔ Good distribution, Good dataset.

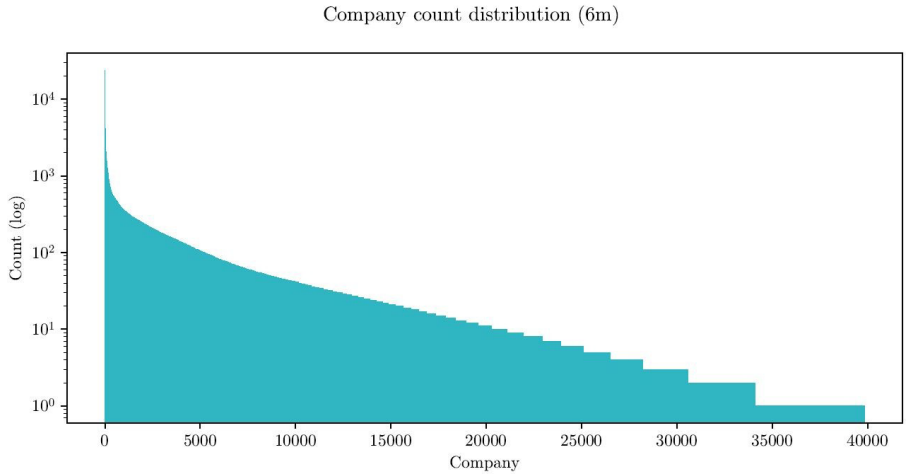


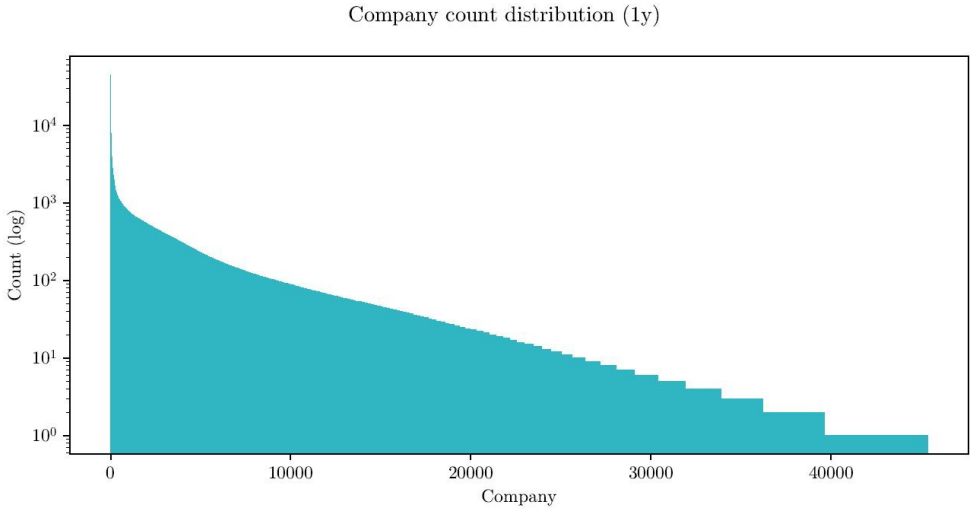
About the dataset III

What aboutthe company distribution?

Follows a Power-Law







Our proposal: SentenceBERT

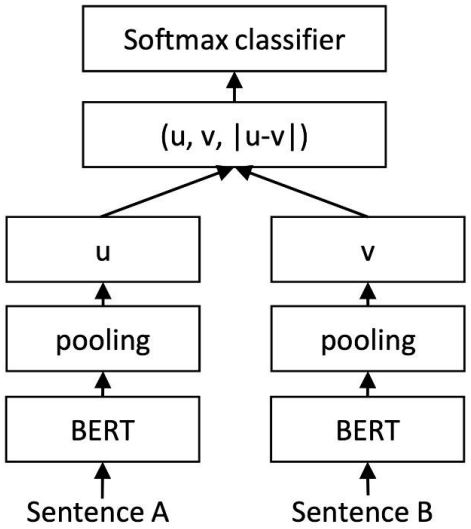


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

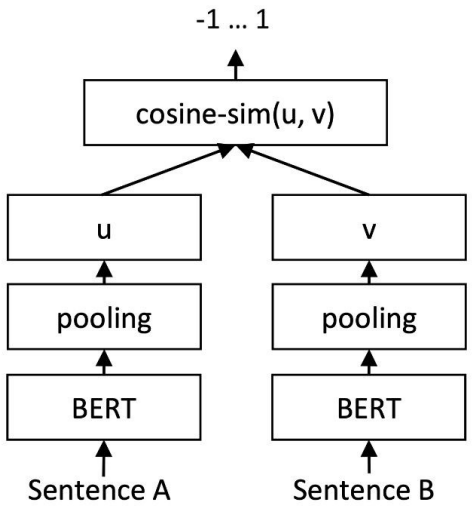


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

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 modelsbut...
  - ...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.

Natural Language Processing

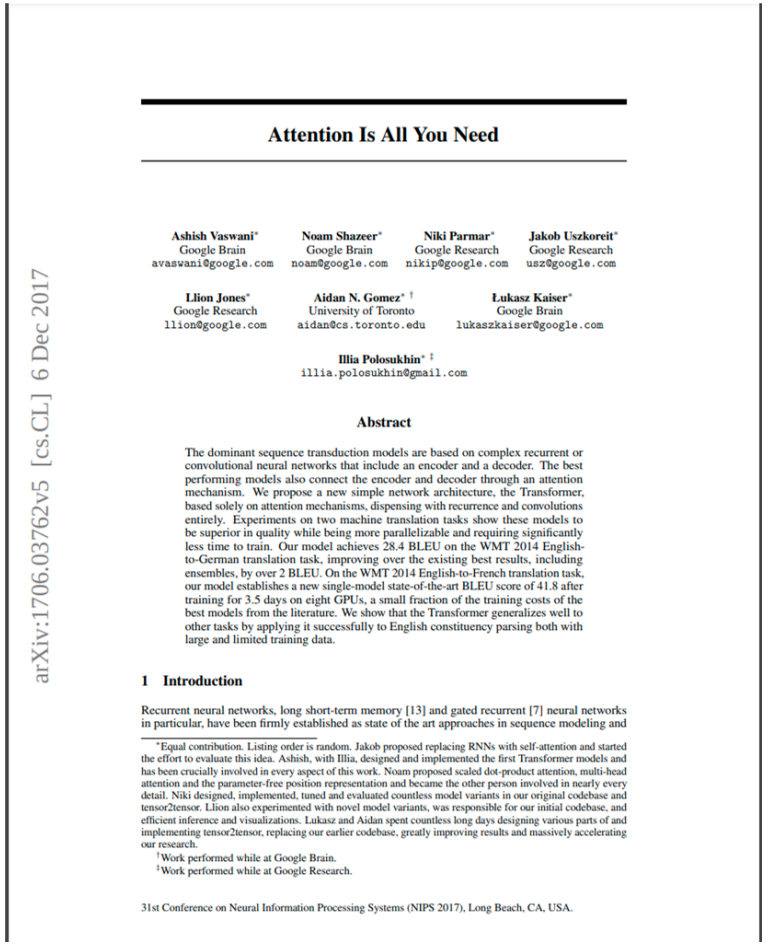
Transformers

What are Transformers?

Attention Is All You Need

Google Brain and Google Research
Cited by 66302 as of feb 2023

https://arxiv.org/abs/1706.03762



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

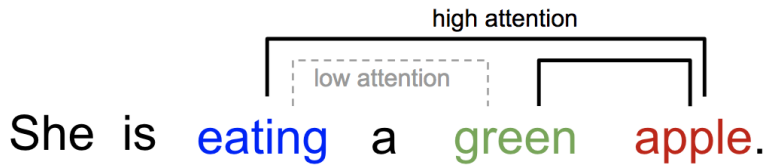
†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

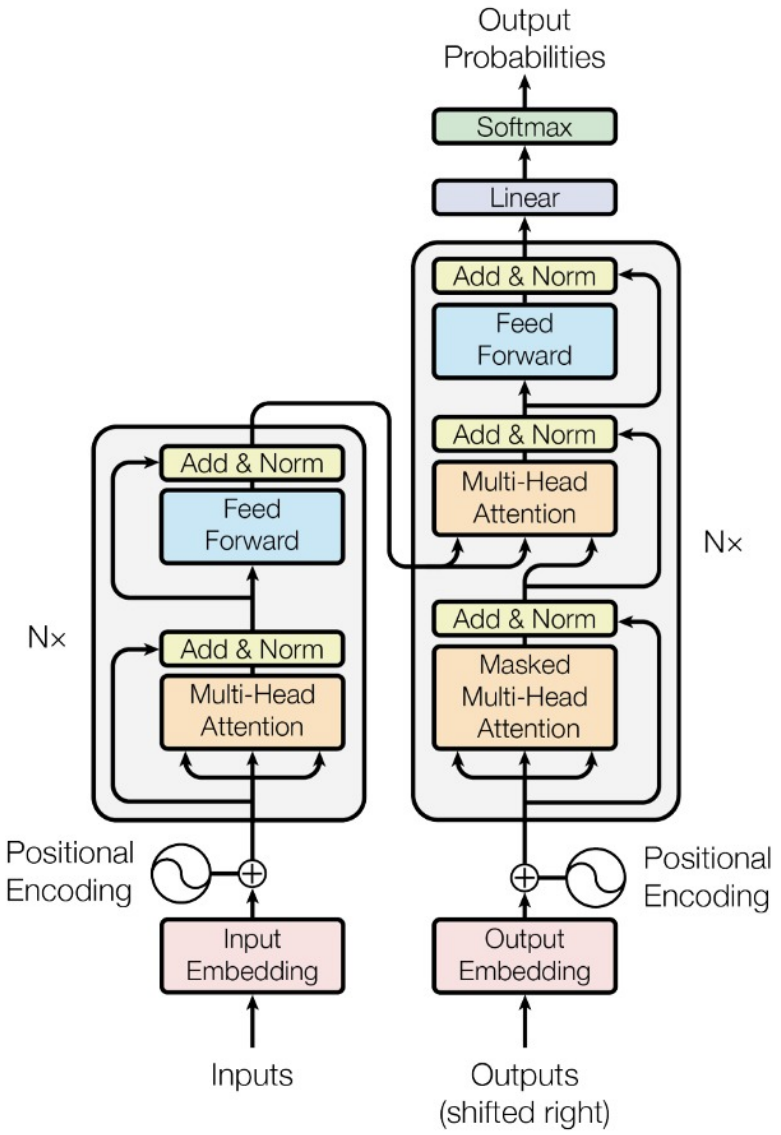
What is Attention? And … Why?

- Attention allows to draw connections between any parts of a sequence.
- RNNs have limited memory mostly. With Transformers, it is possible to keep long-range dependencies that have the same likelihood of being taken into account as any other short-range dependencies.
- Think of Chat GPT
- Relates different positions of a sequence to compute its representation.
- Generate an Encoding for sequences.
- Useful for Translation, Summarization,



The Taxonomy of the Transformer

- Encoder - Decoder Structure
- Positional Encoding
  (**sin** for even and **cos** for odd positions)
- Multi-Head Attention Layer
- Masking
- Residual Connections
- Feed Forward and Normalization Layers



Attention and Self Attention

- Given a query matrix **Q**, a key matrix **K** and a value matrix **V**, the output is the weighted sum of the value vectors.

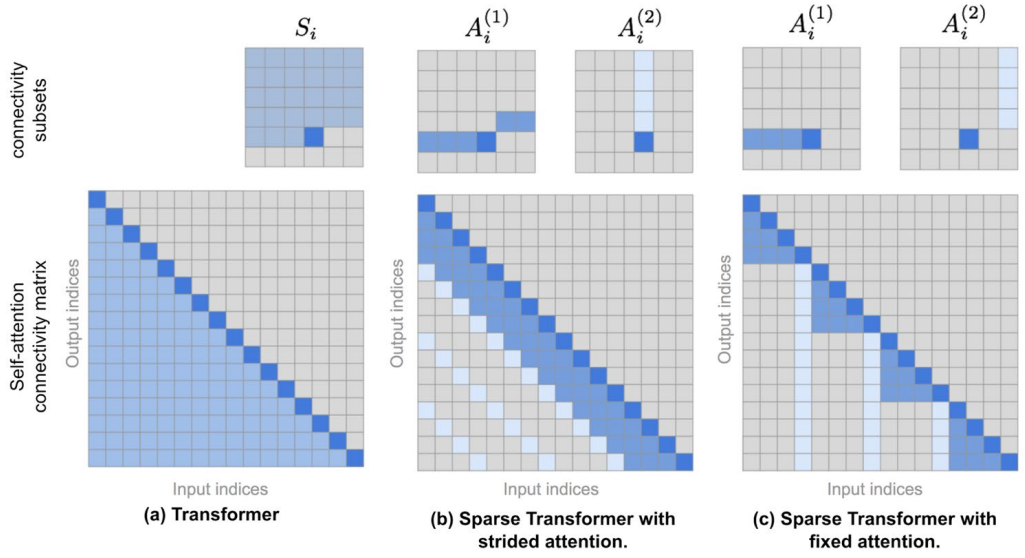
$$\text{attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

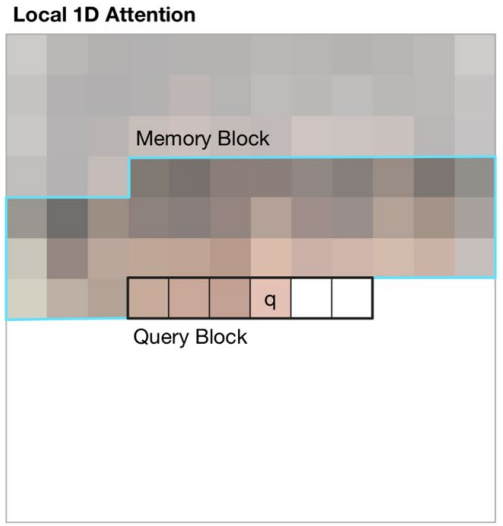
- Weights are assigned depending on the dot-product of the query with the key.

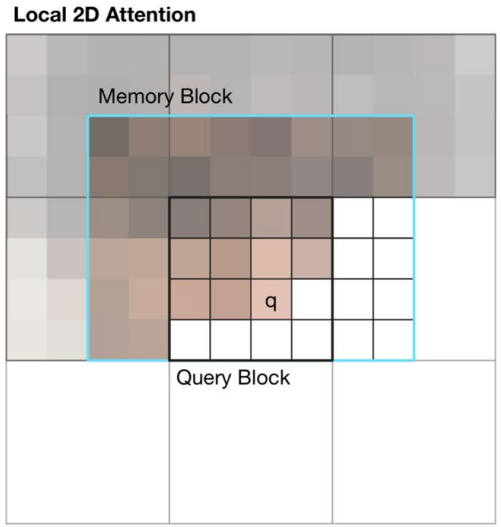
- For query and key vectors $\mathbf{q}_i, \mathbf{k}_j \in \mathbb{R}^d$ we can compute the scalar score.

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

Attention Patterns



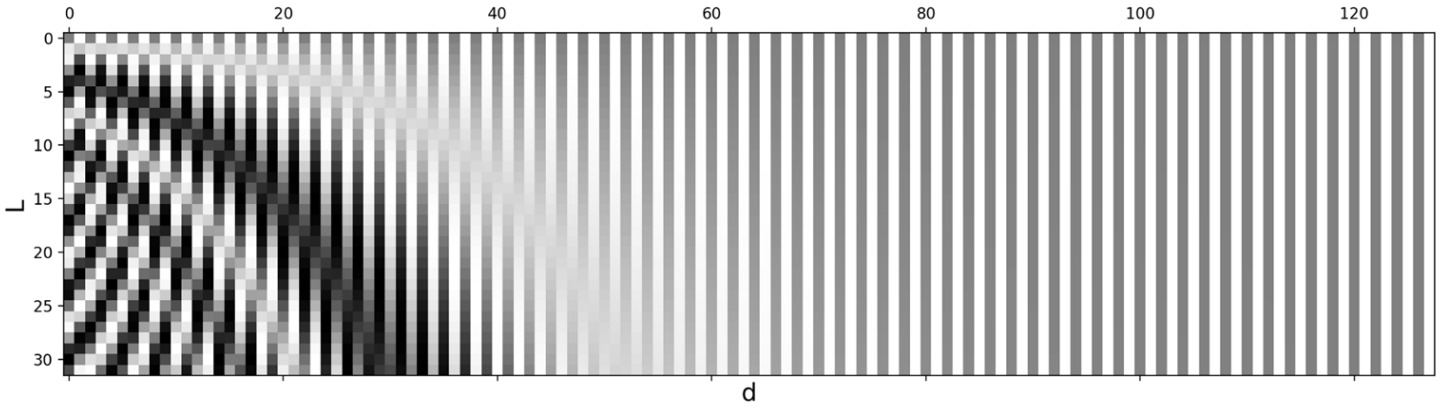




Positional Encoding

- Since self-attention is permutation invariant, important to properly encode positions to provide order information to the model.
- Same dimension as input embedding
- Sinusoidal Positional Encoding

$$PE(i,\delta) = \begin{cases} \sin\left(\frac{1}{10000^{2\delta'/d}}\right) & if\,\delta = 2\delta' \\ \cos\left(\frac{1}{10000^{2\delta'/d}}\right) & if\,\delta = 2\delta' + 1 \end{cases}$$



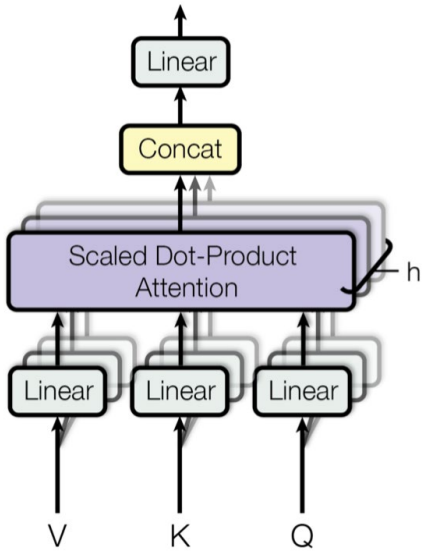
Multi-Head Attention

Key component in the Transformer architecture.

The multi-head mechanism splits the inputs into smaller chunks and then computes scaled dot-product attention in parallel.

$$\text{MultiHeadAttn}(\mathbf{X}_q, \mathbf{X}_k, \mathbf{X}_v) = [\text{head}_1; \ldots; \text{head}_h]\mathbf{W}^o$$

$$\text{where head}_i = \text{Attention}(\mathbf{X}_q\mathbf{W}_i^q, \mathbf{X}_k\mathbf{W}_i^k, \mathbf{X}_v\mathbf{W}_i^v)$$



Then, attention outputs are concatenated and linearly transformed into output dimensions.

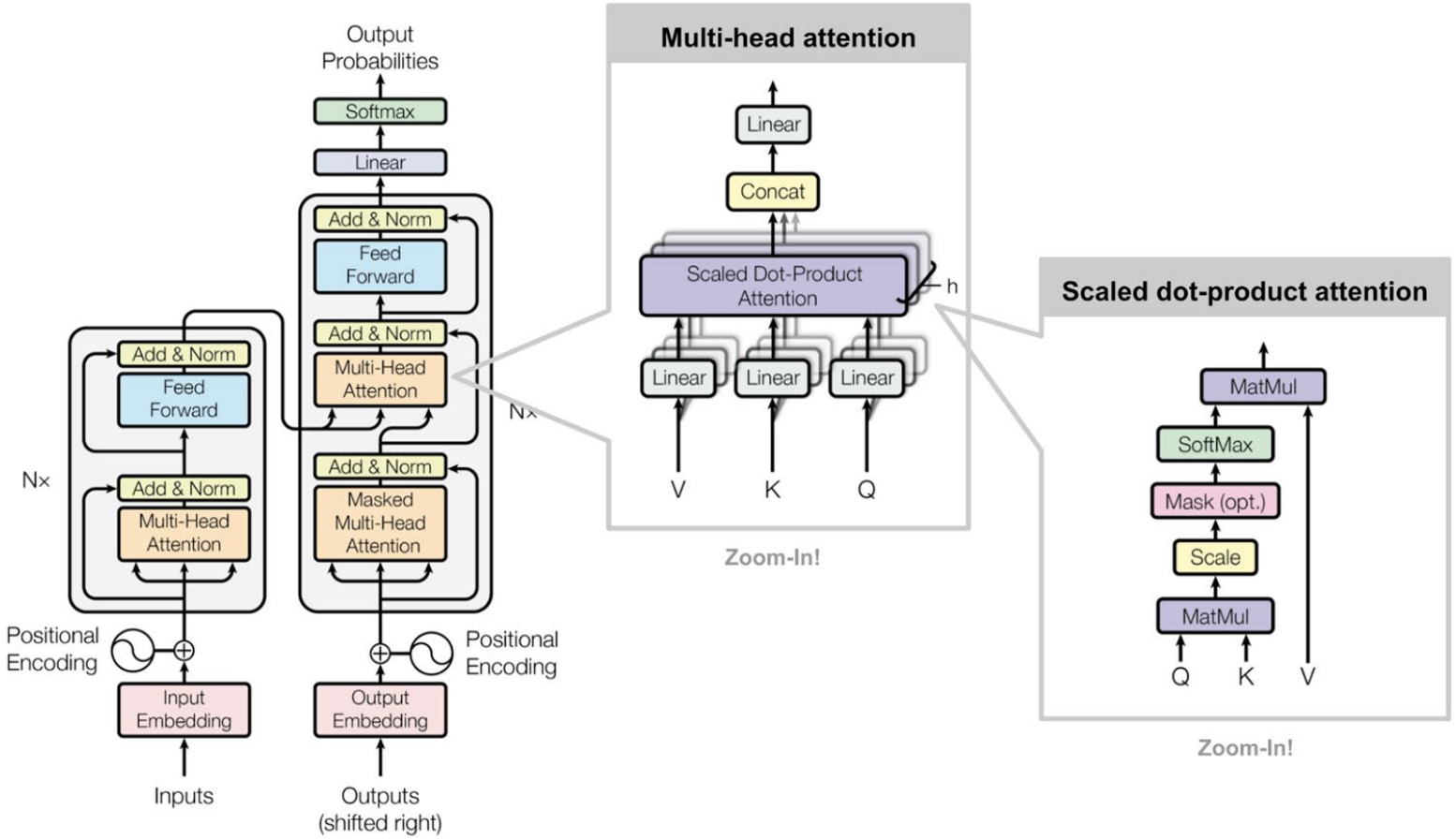
$$\mathbf{W}_i^q, \mathbf{W}_i^k \in \mathbb{R}^{d \times d_k/h} \; \mathbf{W}_i^v \in \mathbb{R}^{d \times d_v/h}$$

Note that
Are the weight matrices to map input embeddings into query, key and value matrices.

$$\mathbf{W}^o \in \mathbb{R}^{d_v \times d}$$

Represents the output linear transformation

Queries, Keys and Values



Blockwise Attention

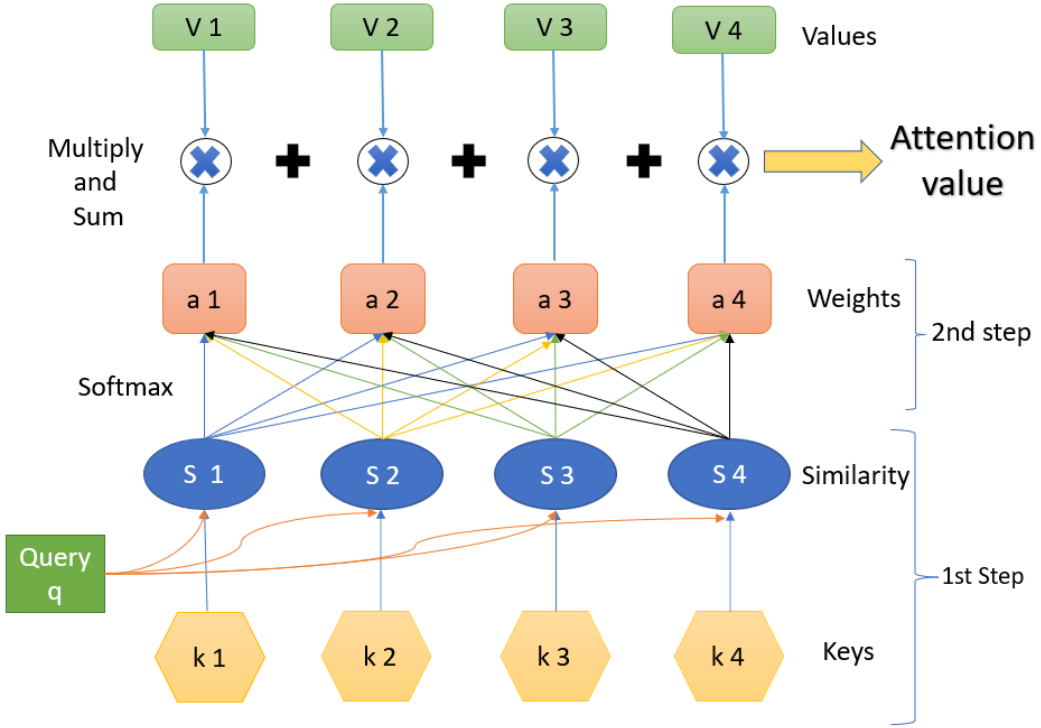
- The idea of the product with M is Masking. This way, words that appear after in the sequence, are completely ignored by the model.
- It is used in the decoder part of the Transformer.
- The idea of using a softmax simply leads us to have a probability and the sqrt factor enables easier convergence. Note that *d* is the positional encoding size / hidden state dimension.

$$\text{attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{M}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} \odot \mathbf{M}\right)$$

$$(\mathbf{A} \odot \mathbf{M})_{ij} = \begin{cases} A_{ij} & if M_{ij} = 1 \\ -\infty & if M_{ij} = 0 \end{cases}$$

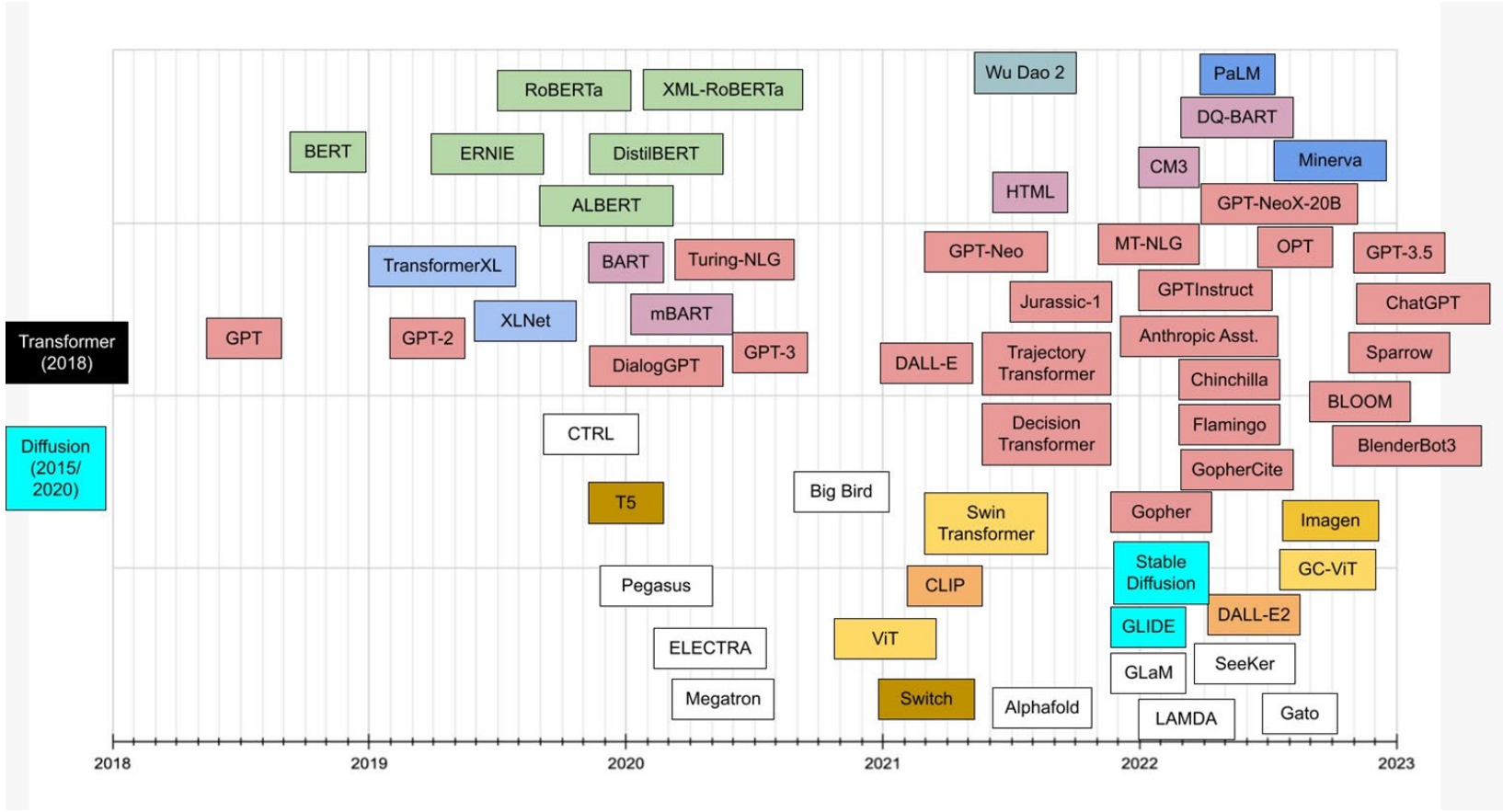
$$\text{where } M_{ij} = \begin{cases} 1 & if \pi\left(\lfloor \frac{(i-1)n}{L} + 1 \rfloor\right) = \lfloor \frac{(i-1)n}{L} + 1 \rfloor \\ 0 & \text{otherwise} \end{cases}$$

Understanding QKV

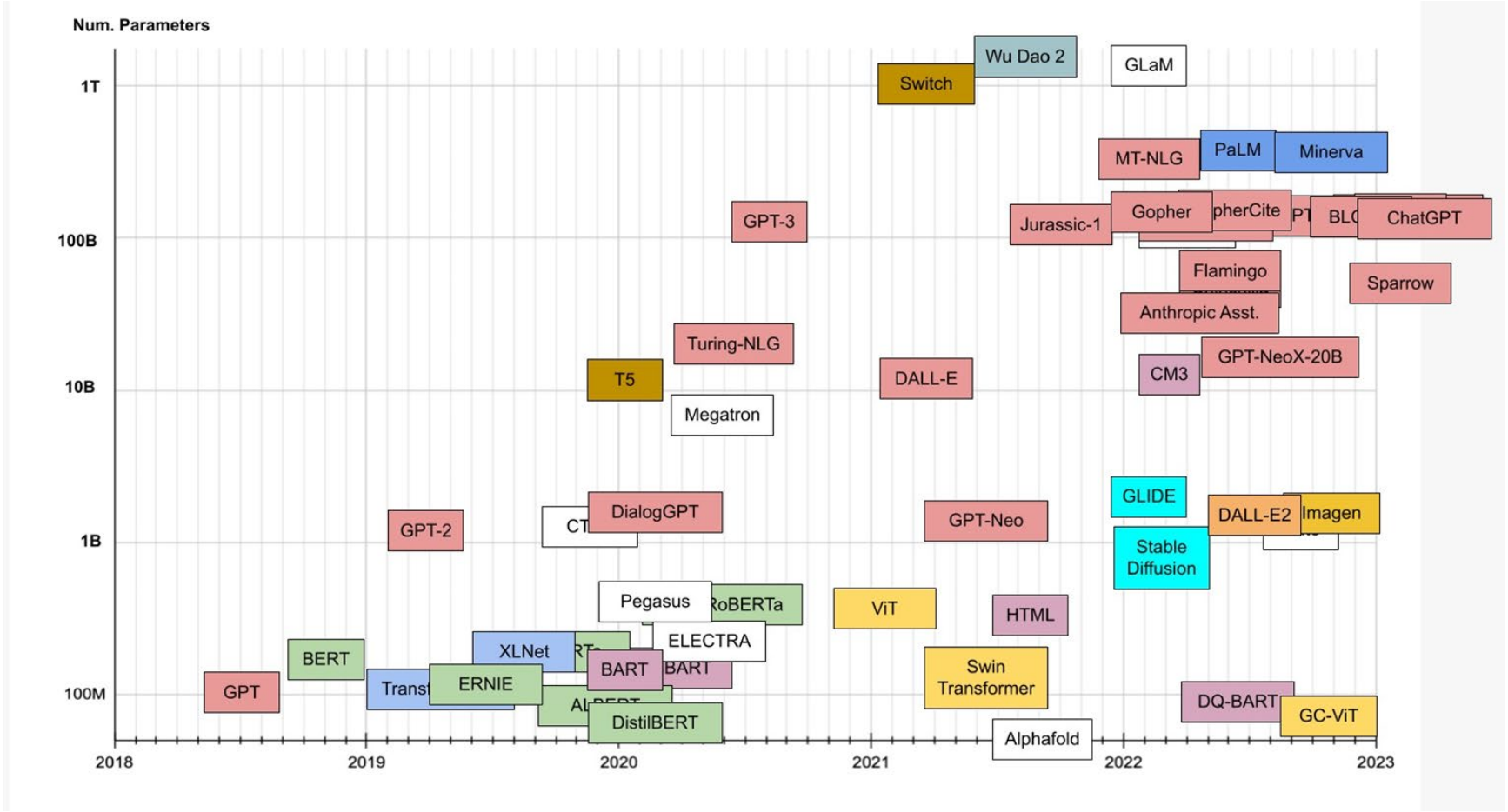


https://miro.medium.com/v2/resize:fit:1100/format:webp/1*Rv_pntt-N2WL7LMbIptHxQ.png

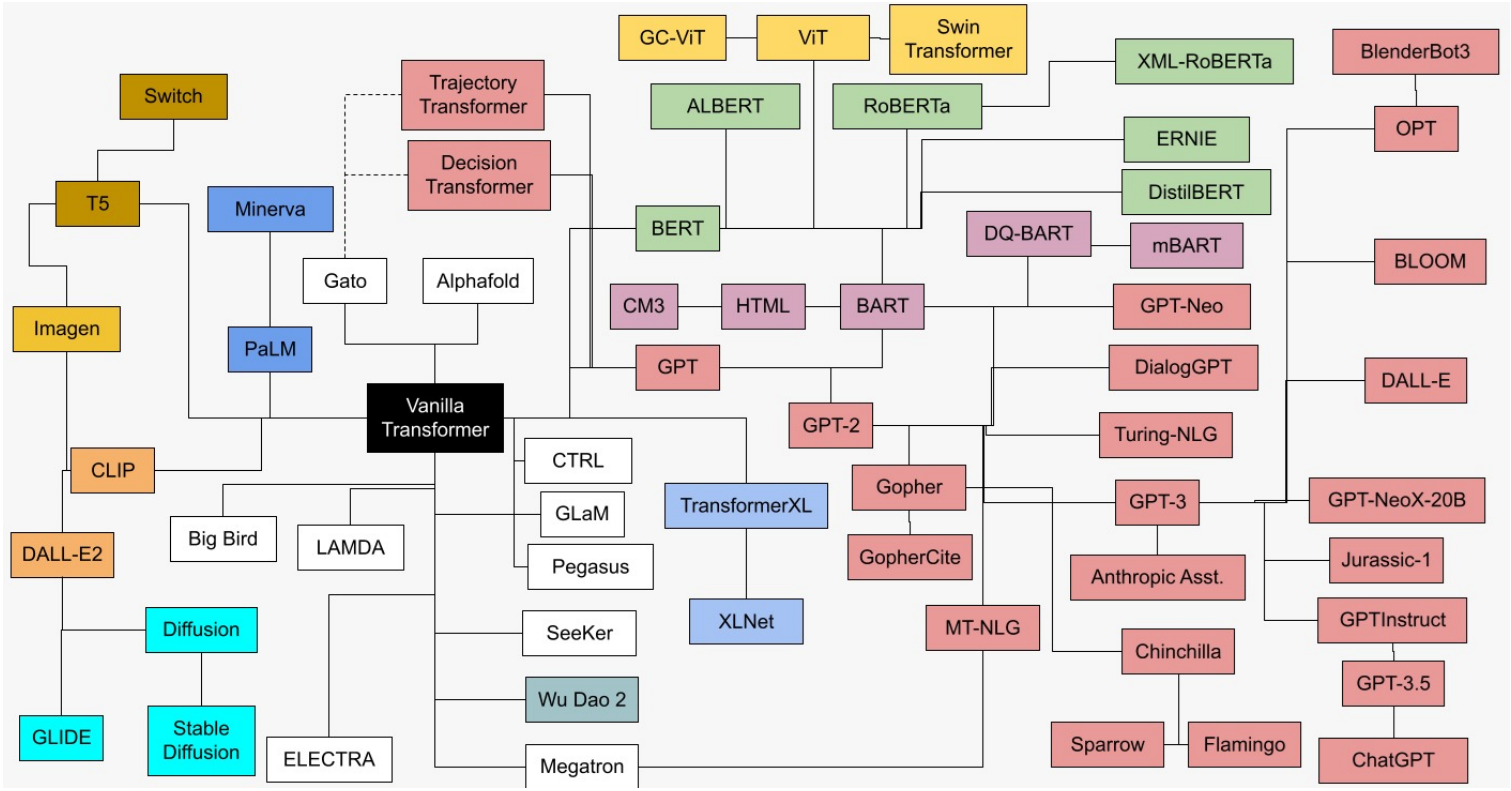
Evolution of Transformers



Number of Parameters comparison



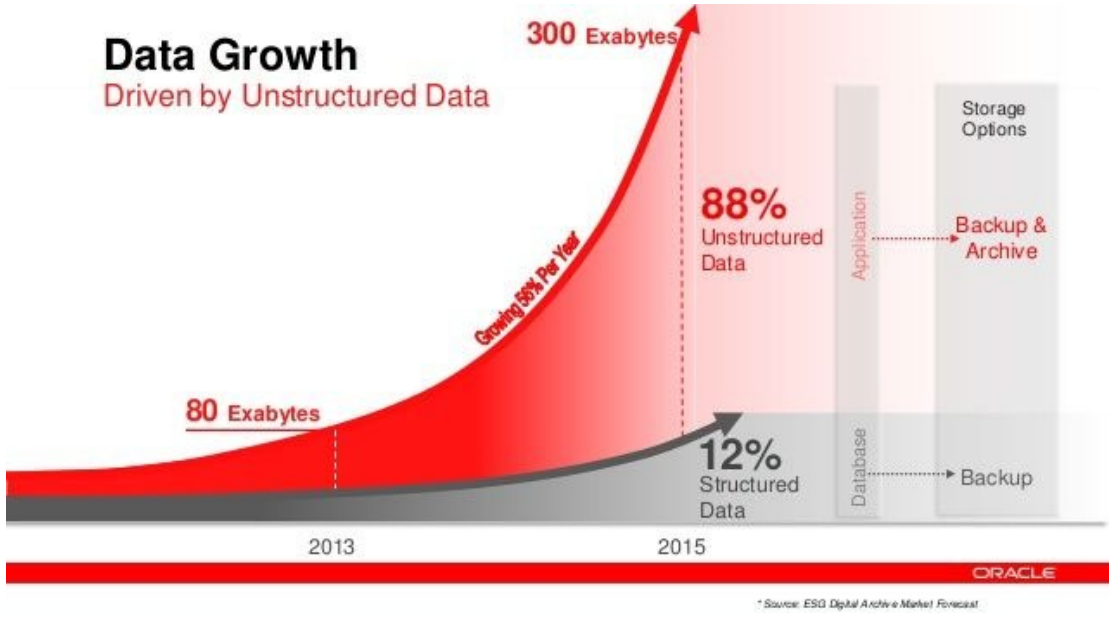
Visualization of some transformer architectures

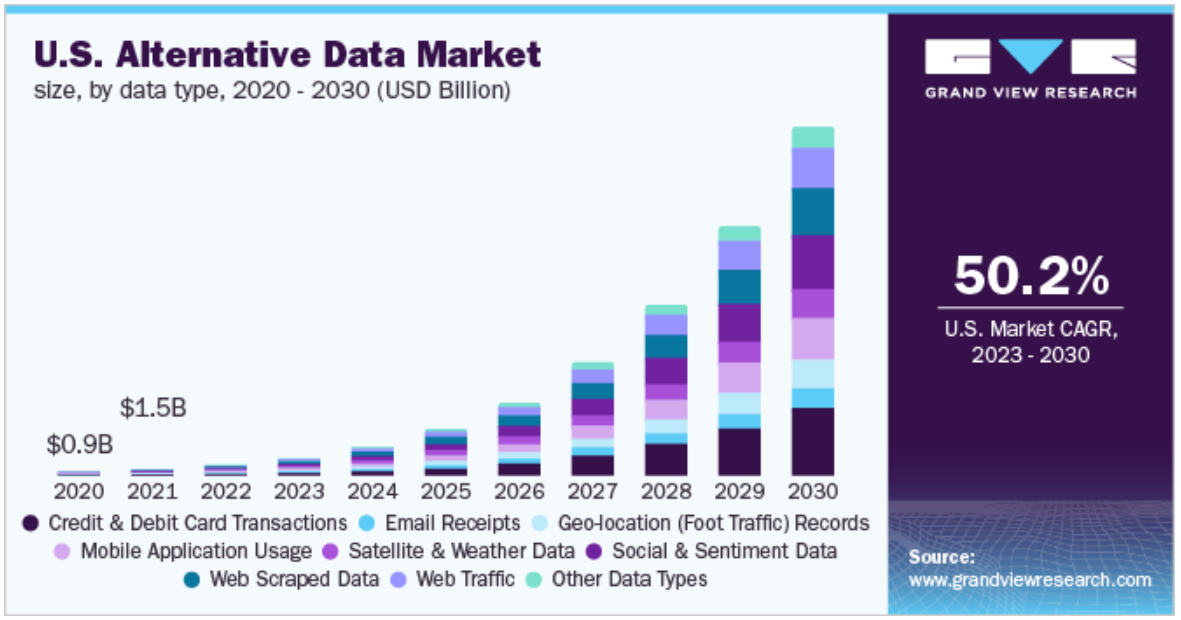


What has all this to do with Finance?

AIFI
ARTIFICIAL INTELLIGENCE IN FINANCE INSTITUTE

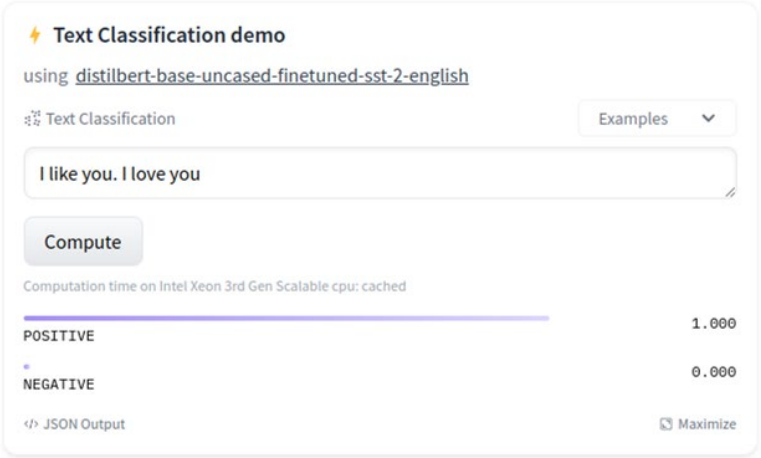
- Alternative Data is Growing Exponentially, might as well take advantage of it

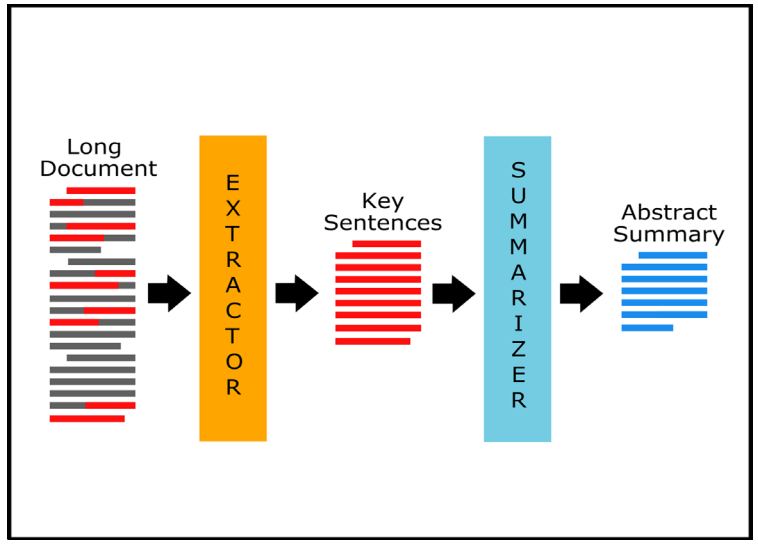




What has all this to do with Finance?

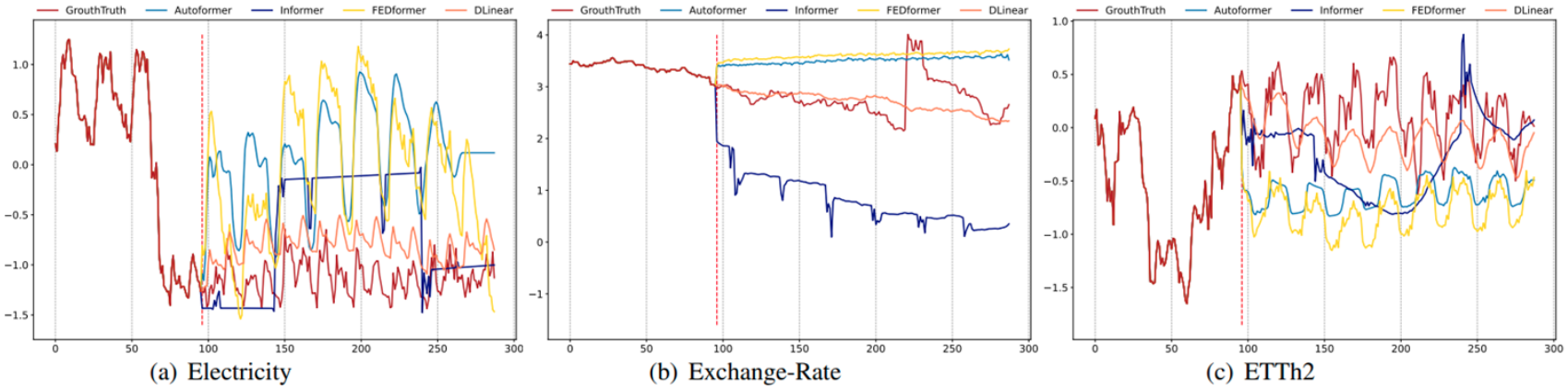
- Sentiment Analysis
- New Alphas
- Can be used for Factor Models
- Summarizing Financial Statements





Transformers for Time Series

- In late 2022 a new approach to time series was developed using Transformers.
- For now, is just a proposal, not yet competing with current models.
- Problems of using Transformers for time Series:
  - We want to extract the temporal relations in an ordered set of continuous points
  - Tokens preserve some ordering information, but *the nature of the permutation-invariant self-attention mechanism inevitably results in temporal information loss.*
  - Rest of the study of Transformers for TS: https://arxiv.org/pdf/2205.13504.pdf



(a) Electricity (b) Exchange-Rate (c) ETTh2

Hugging Face 🤗

Main page of Hugging Face
https://huggingface.co/

Transformer courses
https://huggingface.co/course/chapter1/1

Bonus: Deep RL intro course

https://huggingface.co/deep-rl-course/unit0/introduction?fw=pt

SpaCy

Main page of Hugging Face
https://spacy.io/

NLP Advanced Course
https://course.spacy.io/en/

Resources

AIFI
ARTIFICIAL INTELLIGENCE IN FINANCE INSTITUTE

Github
https://github.com/xamat/TransformerCatalog

https://lilianweng.github.io/posts/2023-01-27-the-transformer-family-v2/

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.pinterest.es%2Fpin%2F675891856551687253%2F&psig=AOvVaw2-Aq2NTubMf3sXR81y0jYo&ust=1677231528569000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCJjTzb2sq_0CFQAAAAAdAAAAABB5

https://www.grandviewresearch.com/industry-analysis/alternative-data-market

https://web.stanford.edu/class/cs224n/reports/final_summaries/summary019.html

Reinforcement Learning from Human Feedback

Reinforcement Learning from Human Feedback

The alignment problem

- The next-token output matches the statistical trend the model has observed
- It may not be aligned to user's intention
- Solution: Use human labelers to provide feedback to align the model to the intention of the prompt
  - Human labelers guiding AI to sound plausible to other humans

Human labeller's prompt distribution

Table 1: Distribution of use case categories from our API prompt dataset.

Use-case	(%)
Generation	45.6%
Open QA	12.4%
Brainstorming	11.2%
Chat	8.4%
Rewrite	6.6%
Summarization	4.2%
Classification	3.5%
Other	3.5%
Closed QA	2.6%
Extract	1.9%

Table 2: Illustrative prompts from our API prompt dataset. These are fictional examples inspired by real usage—see more examples in Appendix A.2.1.

Use-case	Prompt
Brainstorming	List five ideas for how to regain enthusiasm for my career
Generation	Write a short story where a bear goes to the beach, makes friends with a seal, and then returns home.
Rewrite	This is the summary of a Broadway play: """ {summary} """ This is the outline of the commercial for that play: """

Reinforcement Learning from Human Feedback

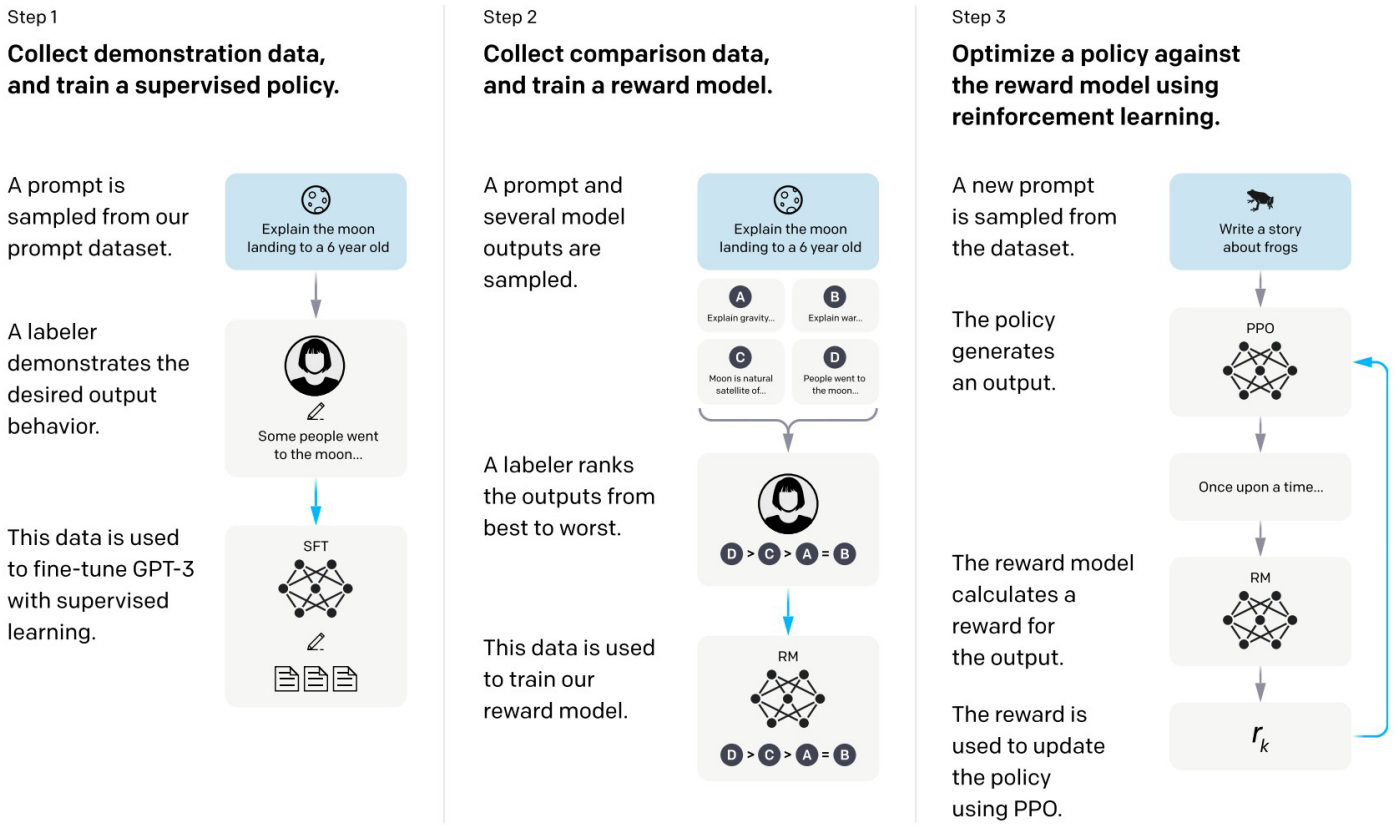
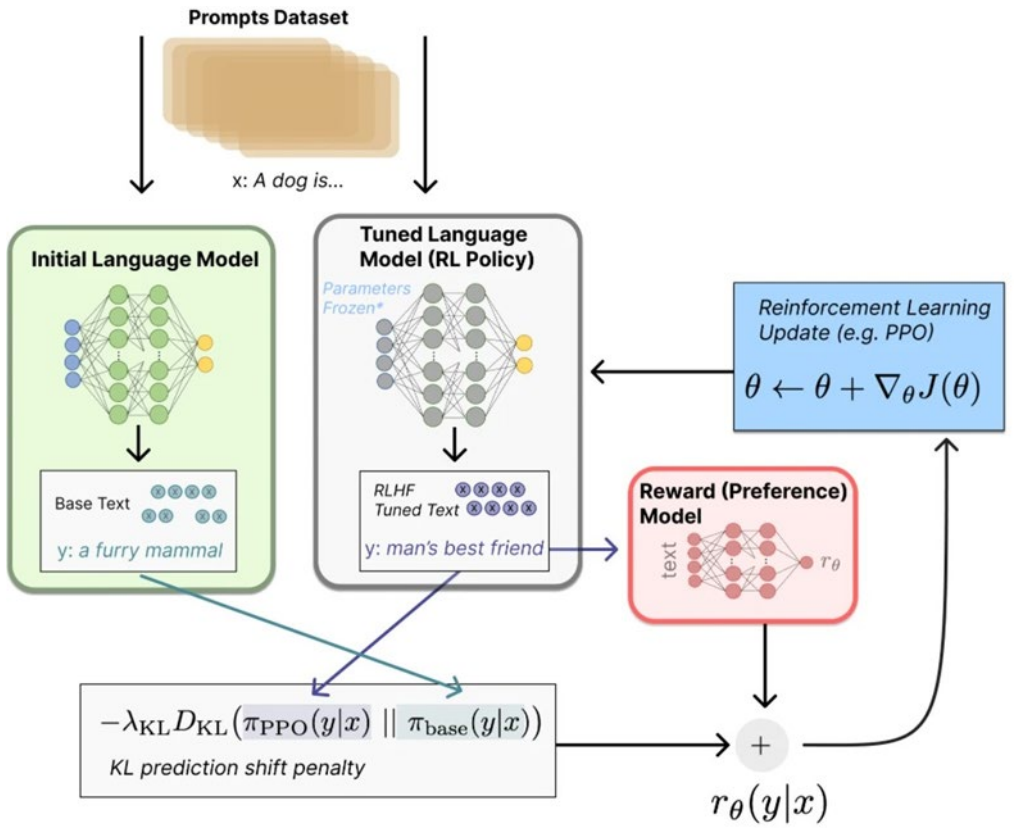


Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

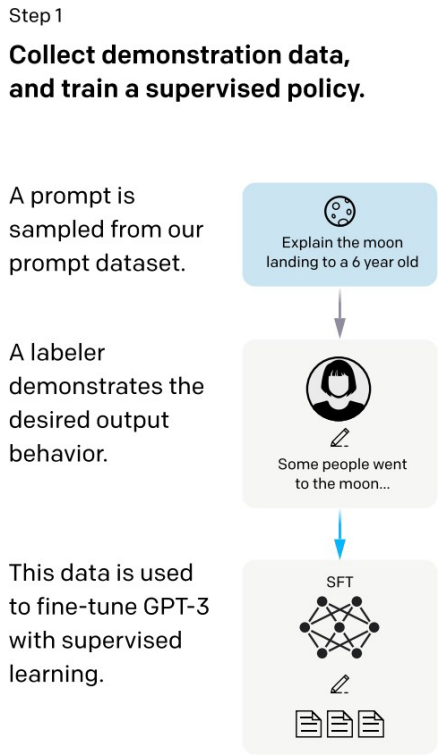
Reinforcement Learning from Human Feedback



Reinforcement Learning from Human Feedback

Step 1: Supervised Fine-Tuning

- Use pre-trained GPT-3 model
- Fine-tune GPT-3 on labeler demonstrations using *Supervised Learning*
- Trained model is called **SFT Model**



Step 2: Reward Modelling

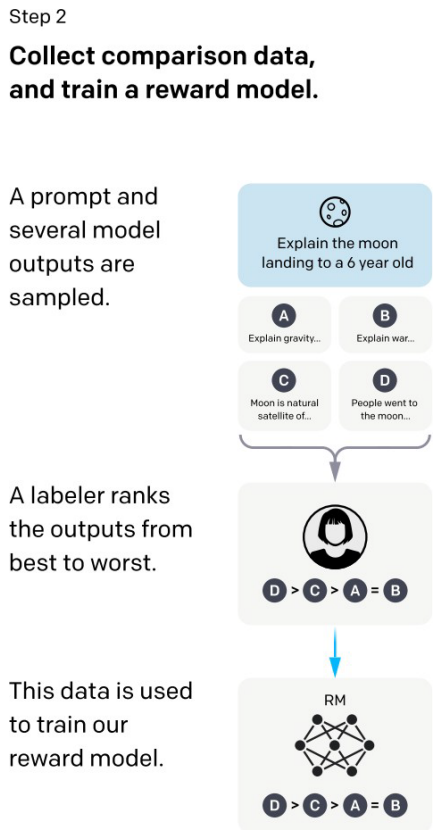
- From the **SFT model**, generate a scalar reward
- Trained to conform to relative ranking of arbitrary pairs of responses
- Why not hinge loss?

Specifically, the loss function for the reward model is:

Should be higher than

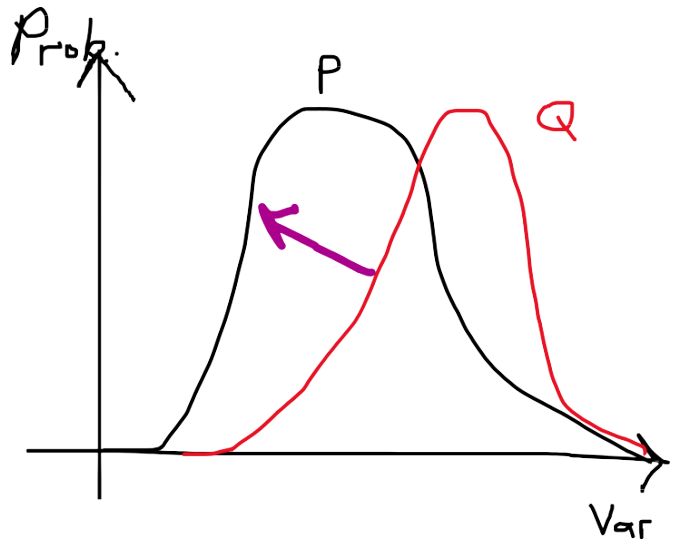
$$\text{loss}(\theta) = -\frac{1}{\binom{K}{2}} E_{(x,y_w,y_l)\sim D}\left[\log\left(\sigma\left(r_\theta(x,y_w) - r_\theta(x,y_l)\right)\right)\right] \quad (1)$$

where $r_\theta(x,y)$ is the scalar output of the reward model for prompt x and completion y with parameters θ, y_w is the preferred completion out of the pair of y_w and y_l, and D is the dataset of human comparisons.



Primer: KL Divergence

- 2 distributions: *P* and *Q*
- Minimal loss occurs when *Q* = *P* (Jenson's Inequality)
  - See Sect 3.1 Gibb's Inequality from https://www.cs.cmu.edu/~venkatg/teaching/ITCS-spr2013/notes/lect-jan22.pdf
- Training on this KL loss makes the distribution of Q become closer to P



$$D_{\mathrm{KL}}(P \parallel Q) = -\sum_{x \in \mathcal{X}} P(x) \log\left(\frac{Q(x)}{P(x)}\right)$$

Reinforcement Learning from Human Feedback

Step 3: Reinforcement Learning

- Fine-tune SFT model on environment using PPO
- Generalize learnt rewards to arbitrary prompts

- State: Prompt
- Action: Response by RL model
- Reward: Generated by PPO objective function

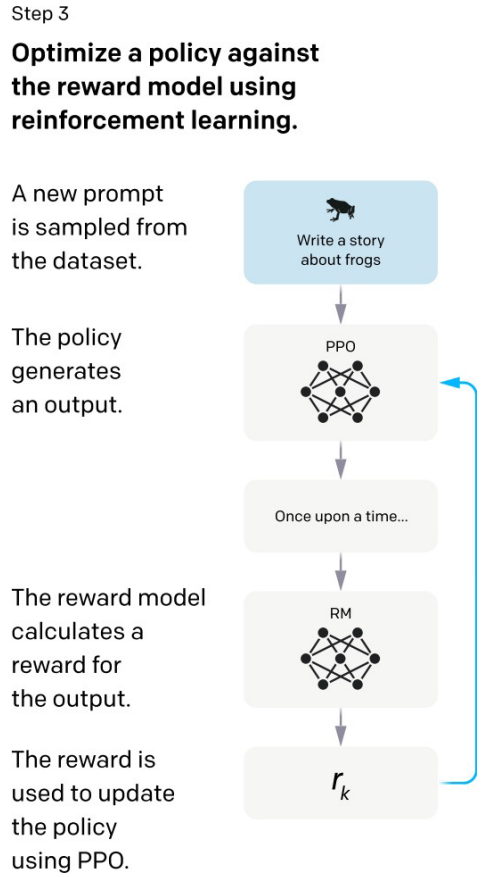
- Why not cross-entropy loss rather than KL divergence loss?

Reward model — KL loss: Conform to SFT model

$$\text{objective}(\phi) = E_{(x,y)\sim D_{\pi_\phi^{\text{RL}}}}\left[r_\theta(x,y) - \beta \log\left(\pi_\phi^{\text{RL}}(y \mid x)/\pi^{\text{SFT}}(y \mid x)\right)\right] +$$

$$\gamma E_{x\sim D_{\text{pretrain}}}\left[\log(\pi_\phi^{\text{RL}}(x))\right]$$

Cross-entropy loss: Conform to pre-training gradients



Data Used

- Relatively small amounts of data ~30k prompts used for Reward Modelling
- Likely able to generalize to majority of the prompts from a small subset

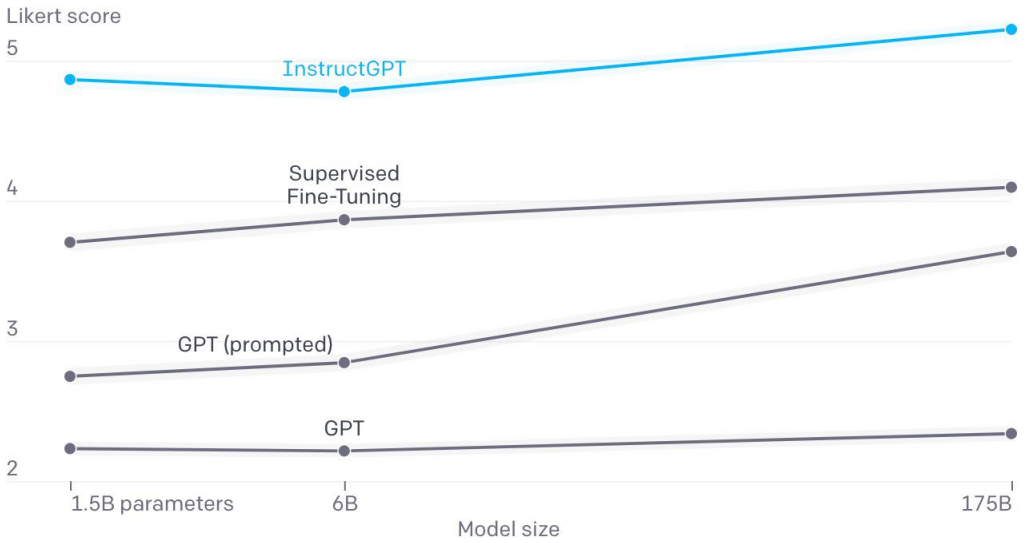
Table 6: Dataset sizes, in terms of number of prompts.

SFT Data			RM Data			PPO Data		
split	source	size	split	source	size	split	source	size
train	labeler	11,295	train	labeler	6,623	train	customer	31,144
train	customer	1,430	train	customer	26,584	valid	customer	16,185
valid	labeler	1,550	valid	labeler	3,488			
valid	customer	103	valid	customer	14,399			

Reinforcement Learning from Human Feedback

Results of Alignment

- High Likert Score for RLHF model
- Higher perceived quality for RLHF model-generated output
- Though increasing model size did not improve it by much



Quality ratings of model outputs on a 1–7 scale (y-axis), for various model sizes (x-axis), on prompts submitted to InstructGPT models on our API. InstructGPT outputs are given much higher scores by our labelers than outputs from GPT-3 with a few-shot prompt and without, as well as models fine-tuned with supervised learning. We find similar results for prompts submitted to GPT-3 models on the API.

Reinforcement Learning from Human Feedback

Results of Alignment

- Very few human inputs (only 40 labellers!) needed for alignment across various types of text prompts
  - Note the number of human labellers for ChatGPT is undisclosed and could possibly be much larger than this
- Models generalize to preference of “held-out” labelers which did not produce any training data

Moderation API

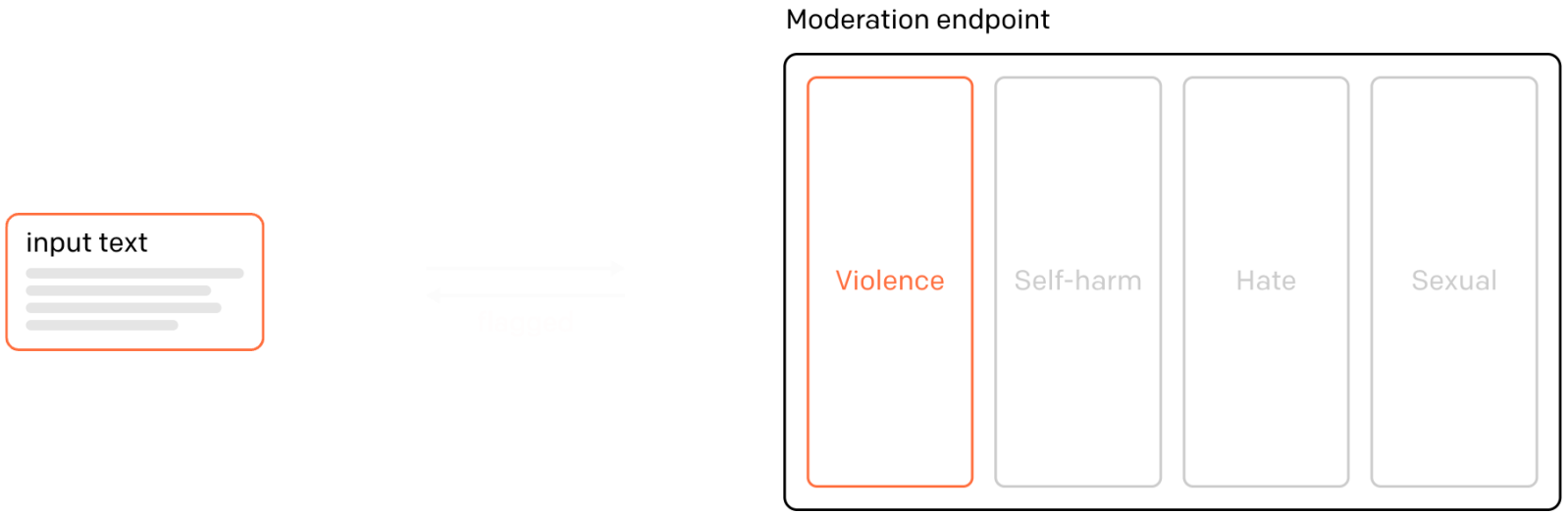
Rule-based filtering to prevent sensitive output

Rule-based versus Neural Networks

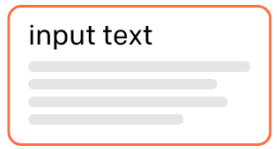
- Neural networks are hard to configure the desired outputs, as it is still largely one black box
  - We cannot simply just ask the model not to do something, or modify some parameter to stop it from doing something

- Apply some form of moderation over ChatGPT outputs in order to prevent unwanted content

If flagged, generate some filler text



If flagged, generate some filler text



C

How do I bully John Doe?

It is not appropriate or ethical to bully anyone. Bullying is a form of abusive behavior that can cause severe harm to the person being bullied, both emotionally and physically. It is important to treat all individuals with kindness, respect, and empathy. If you have a conflict or issue with someone, it is best to try to address it in a calm and respectful manner and seek help from a trusted authority figure if necessary.

Discussion ChatGPT

Strengths of ChatGPT

- Good for combining human knowledge in novel ways
  - Creative
  - Adaptable to small changes in input prompt
- Great for coding and summarization
- Good for doing homework requiring regurgitation of concepts
- Can be a fun AI to chat with on any topic

Weaknesses of ChatGPT

- No fact-checker
- No reasoning module
- Confident even when incorrect
- Can be ambiguous and hedge multiple options even when there is one clear option
- Not good at math

Questions to Ponder (Technical)

- Why not use a Hinge Loss for the Reward Model?
- Why not do away with PPO, and instead use the Reward Model as a CLIP-like objective to select the output responses?
- Should we just generate the next token only? Why not generate the next n tokens?
- Is ChatGPT really generalizing to out-of-sample prompts? Or is the web data so large that almost anything you can think of is within-sample?

Questions to Ponder (Ethical/General)

- Should ChatGPT be used for niche areas, e.g. law?
- Should ChatGPT be banned for students, or should students be able to use ChatGPT as a resource just like a web?
- How effective will ChatGPT be for web search?
- Is ChatGPT sentient/conscious? (I think there is a simple answer, but somehow this is frequently asked)