# CoNFET: An English Sentence To Emojis Translation Algorithm
# or
# 📄🗜️➡️😆

Alex Day, Chris Mankos,
Dr. Soo Kim, and Dr. Jody Strausser

Clarion University of Pennsylvania

# Overview

- Introduction
  - Motivation
  - Related work
- Algorithm Overview
  - Sentence Compositions
  - Emoji Mapping
  - Translation Generation
  - Translation Scoring
  - Improvements
- Results
- Conclusion & Future Work

# Motivation

- **Is it possible to generate representative emoji sentences intelligently?**
  - Not a complete one-to-one mapping of words to emojis
  - Capture ‘essence’ of several words
- Why is this important?
  - Pictograms are internationally recognizable
  - Possibility to improve the current computational “emoji understanding”
- Some examples of what we would like:
  - My dog can run so fast → 🐕 🏃 💨
  - I’m thinking that this computer has a virus → 🤔 🖥️ 🦠

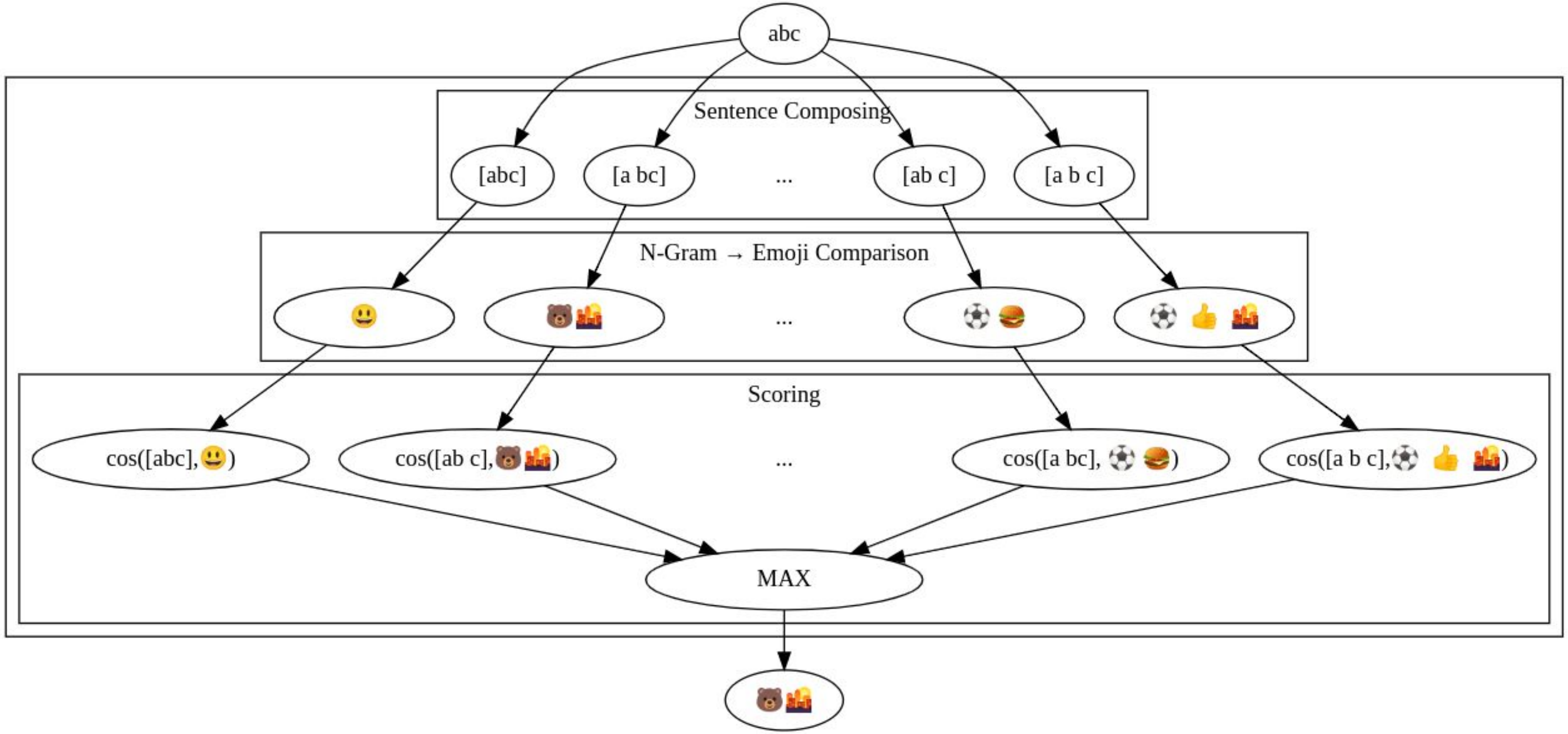
# Related Work

- Embeddings
  - Word2Vec
  - Sent2Vec
  - Emoji2Vec
- Direct word → emoji mappings
  - https://decodeemoji.com
  - https://meowni.ca/emoji-translate/
- Emoji Dick
  - Translation of Moby Dick into Emojis
- Similar works
  - An Approach for Text-to-Emoji Translation

# Composition of N-Grams for Emoji Translation (CoNET)

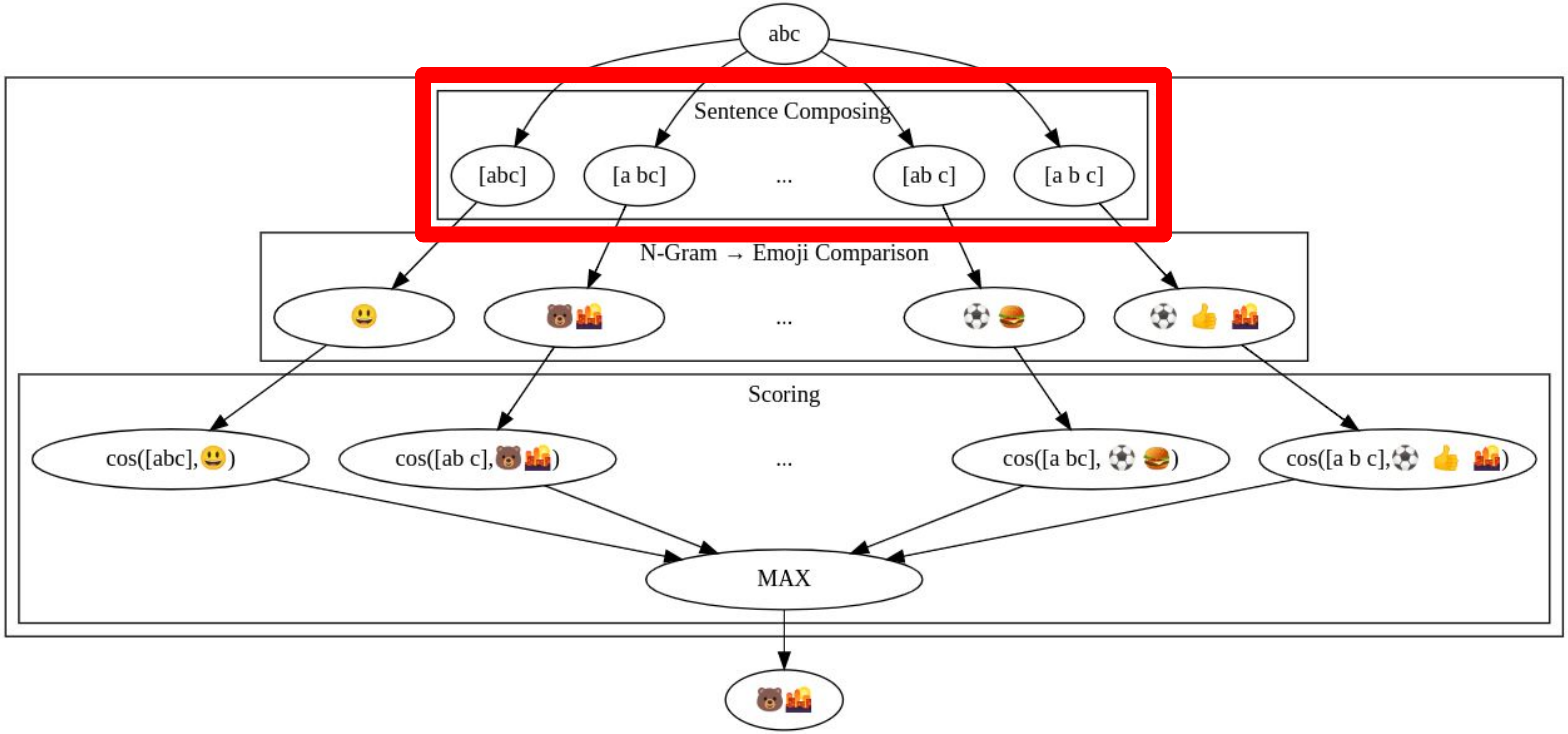
- CoNET is a combination of machine learning and natural language processing (NLP) techniques that can produce a series of emojis when given a variable length input sentence
- Algorithm is split into separate parts
  1. Sentence compositions
  2. N-Gram → emoji comparison
  3. Translation scoring
  4. Summary generation

# High-Level Algorithm Architecture




Combination of N-Grams for Emoji Translation

# High-Level Algorithm Architecture




Combination of N-Grams for Emoji Translation

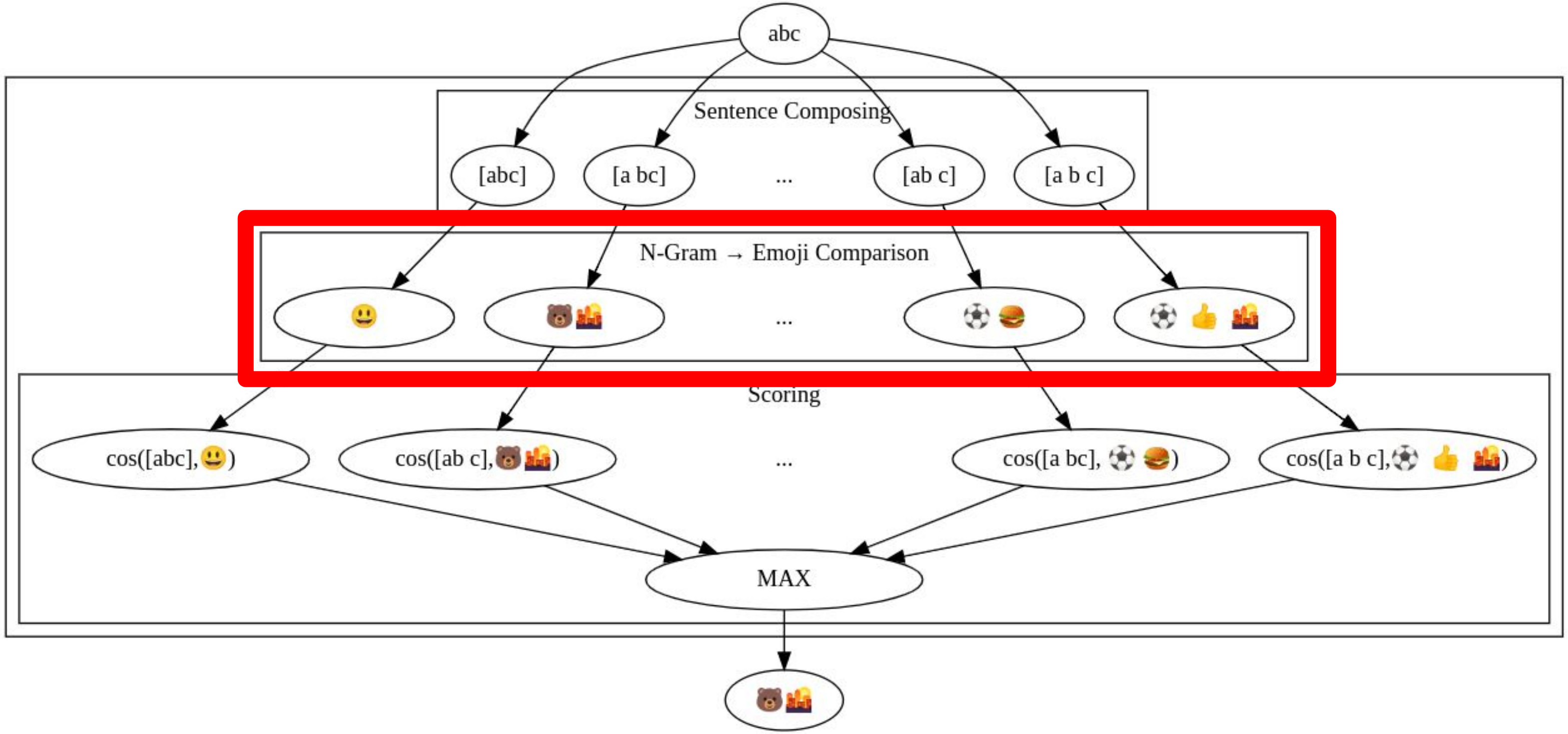
# Sentence Compositions

- An <u>n-gram</u> is a variable length sequence of contiguous words, normally in the context of a larger phrase or sentence.
    - A sentence can be represented by a sequence of n-grams
    - **<u>Ex:</u>** The sentence “The dog bit me very hard” has the n-grams:
        - “The dog bit”, “me”, “very hard”
- We will refer to a sequence of n-grams as the **n-gram sequence**, and an individual n-gram in the sequence as an **n-gram**
- The simplest way to partition a sentence is to do so exhaustively
    - **<u>Ex:</u>** For the sentence “The dog bit me very hard” we check all sequences of n-grams:

$$\boxed{\texttt{The}}\ \boxed{\texttt{dog}}\ \boxed{\texttt{bit}}\ \boxed{\texttt{me}}\ \boxed{\texttt{very hard}}$$

- Assumption: there must exist some optimal n-gram sequence that generates the best summary

# High-Level Algorithm Architecture



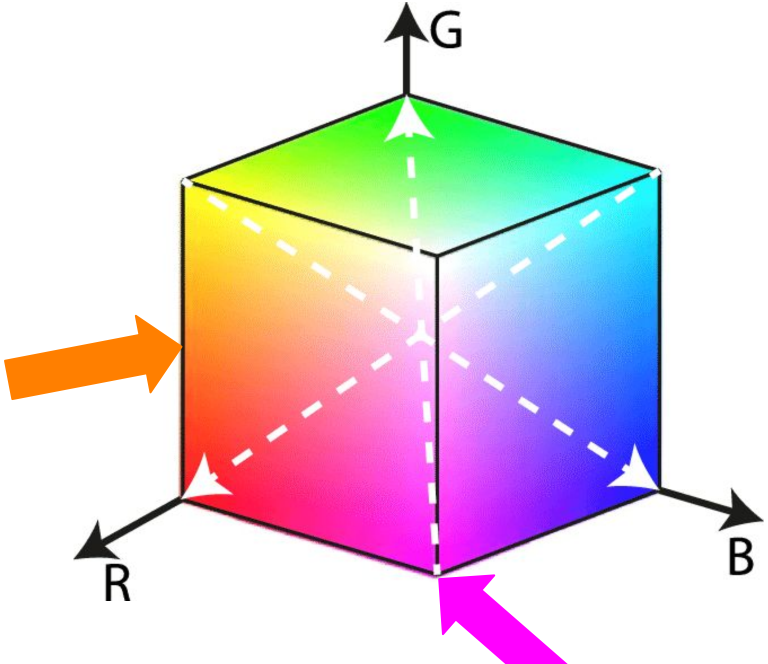

Combination of N-Grams for Emoji Translation

# N-Gram→ Emoji Comparison

- Need a way to translate an n-gram (eg. “the dog”) to an emoji (eg. 🐕) that is **not** just a one-to-one mapping from word to emoji
- What is an emoji?
  - Just a mapping over a description:
    - 🐕 → Dog, Puppy, Beagle
    - 🌎→Earth, Home, North and South America

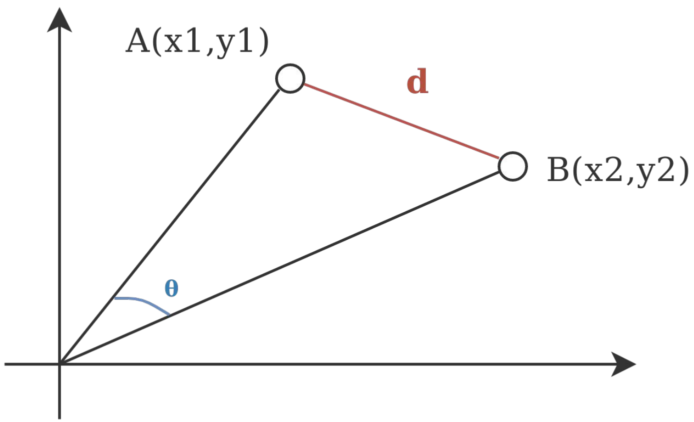
# Embeddings Explained using Colors

- Colors can be mapped into three-dimensional space using a vector of 3 numbers representing red, green, and blue
- Colors as specific points
  - Purple → [1, 0, 1]
  - Orange → [1, 0.5, 0]
- Adding colors together
  - Red + Blue = Purple
  - [1, 0, 0] + [0, 0, 1] = [1, 0, 1]



# Embeddings Explained using Colors Continued...

- We can also use this idea to compare points in the color space
- Similarity Metrics
  - Euclidean Distance is the line distance between two points (d)
  - Cosine Difference is 1 - the cosine of the measure of the angle from the origin between two points (Θ)
- Determining color difference
  - cos(Purple, Orange) = 0.04
  - cos(Red, Blue) = 1
- Closest Vectors (Vectors with lowest difference)
  - closest(Red) = {Scarlet, Lipstick Red, ...}



# N-Gram→ Emoji Comparison Continued...

- Sent2Vec
  - Translates a sentence, phrase, or word into a 700-long vector with elements from -1 - 1
  - Embeds the semantic meaning of the sentence into a machine readable and representative format

```
sent1 = "I have completed the homework"
s2v.embed_sentence(sent1)
```

```
[-1.48401037e-01  2.86369592e-01 -3.84726822e-02 -2.70754427e-01
 -2.85093516e-01  2.10888043e-01 -1.31084099e-01 -1.06972098e-01
 -7.71741331e-01 -2.98576862e-01  5.25636554e-01  5.20731509e-03
 -1.16157994e-01  5.41290402e-01 -1.70695543e-01 -2.00217500e-01
 -2.27179080e-01 -6.52487054e-02  6.75283015e-01 -2.49963105e-02
  3.56599867e-01 -1.71164840e-01  3.64689410e-01  2.17651650e-02
  1.07673749e-01  3.45551342e-01  2.88191617e-01  3.98644842e-02
  1.64047748e-01  1.60539448e-01 -2.62602031e-01  5.31935453e-01
  2.14340508e-01 -4.34283257e-01  1.40293509e-01  4.07240801e-02
  4.79401052e-01  1.69155195e-01 -1.64602101e-02 -7.75851190e-01
 -4.03030366e-01  2.18428180e-01  3.06886464e-01 -2.29964495e-01
```

...

# N-Gram→ Emoji Comparison Continued...

- Sent2Vec
  - Translates a sentence, phrase, or word into a 700-long vector with elements from -1 - 1
  - Embeds the semantic meaning of the sentence into a machine readable and representative format

```
sent1 = "I have completed the homework"
sent2 = "He did finish the homework"
cosine(s2v.embed_sentence(sent1), s2v.embed_sentence(sent2))
```

```
0.63223
```

```
sent1 = "I have completed the homework"
sent2 = "The quick brown fox jumped over the lazy dog"
cosine(s2v.embed_sentence(sent1), s2v.embed_sentence(sent2))
```

```
0.16078
```

# N-Gram→ Emoji Comparison Continued...

- To find the closest emoji to a vector the following function is executed:

**Input:** Sentence

$Input\ Embedding \leftarrow Embed(Sentence)$

**for** $Emoji,\ Description$ *in* $Dataset$ **do**

$\quad Emoji\ Embedding \leftarrow Embed(Description)$

$\quad Similarity \leftarrow cosine(Emoji\ Embedding, Input\ Embedding)$

**end**

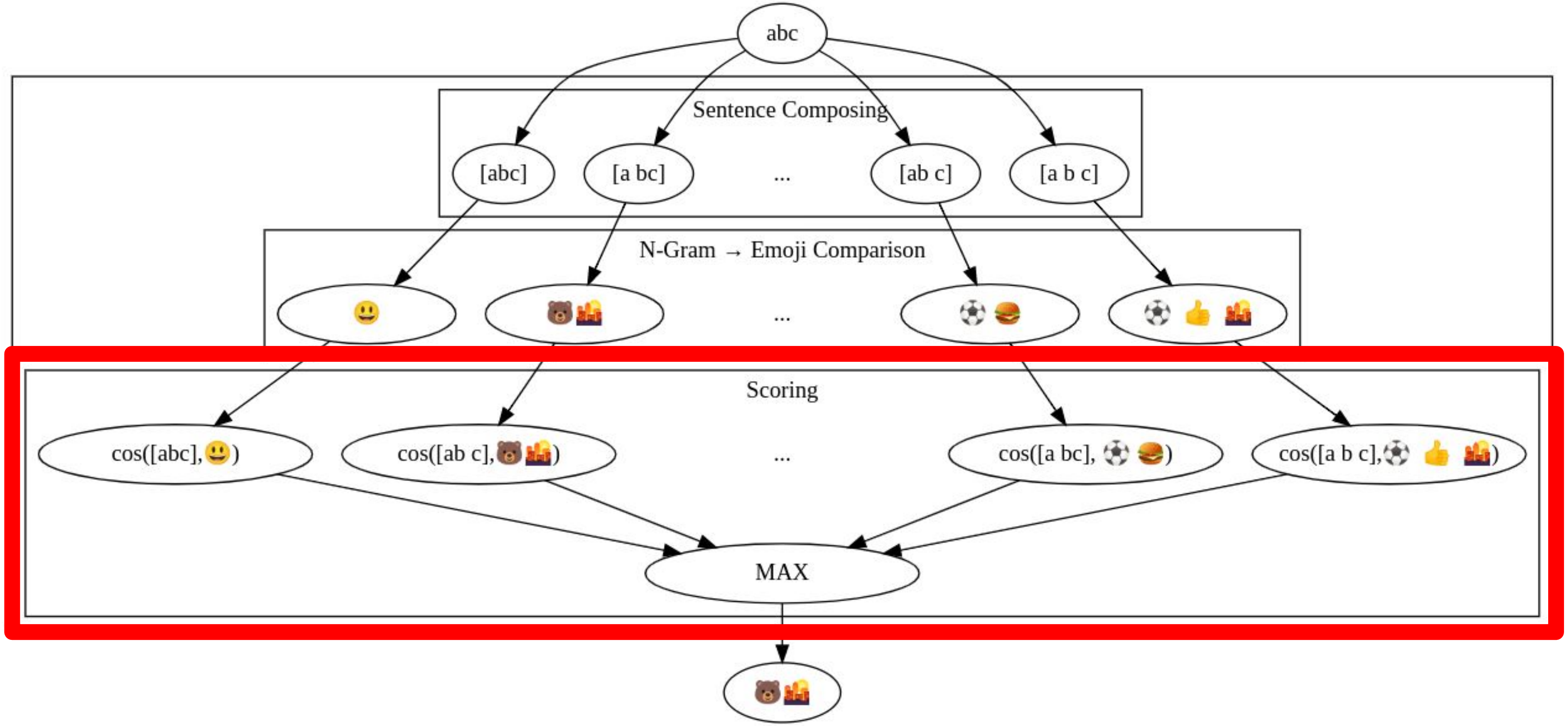
**return** $Emoji, Similarity, Description$ with highest $Similarity$

- We can now query our dataset like this:

$$ClosestEmoji(\text{considerate})$$

$>>>$ (💛, 0.508 . . . , respectful)

# High-Level Algorithm Architecture




Combination of N-Grams for Emoji Translation

# Translation Scoring

- We score a sentence based on the sum of its parts. Meaning that the sentence's score as a whole is an average of the cosine similarity of the n-gram → emoji pairs that make up that summary.
- 🐩🎽💨
    - N-grams → "the dog" "runs" "fast"
    - Emoji-grams → "dog" "run" "fast"
    - Cosine Similarity → 0.96, 1.0, 1.0
    - Average Cosine Similarity → 0.9844
- 💭💾🐛
    - N-grams → "i think that this" "computer" "has a virus"
    - Emoji-grams → "think" "computer" "virus"
    - Cosine Similarity → 0.52, 1.0, 0.79
    - Average Cosine Similarity → 0.77

# Summary Generation

**Input:** English Sentence, S

$Sequences \leftarrow Exhaustive(S)$

**for** *N-Gram Sequence* ***in*** *Sequences* **do**

  **for** *N-Gram* ***in*** *N-Gram Sequence* **do**

    Add closest emoji to summary

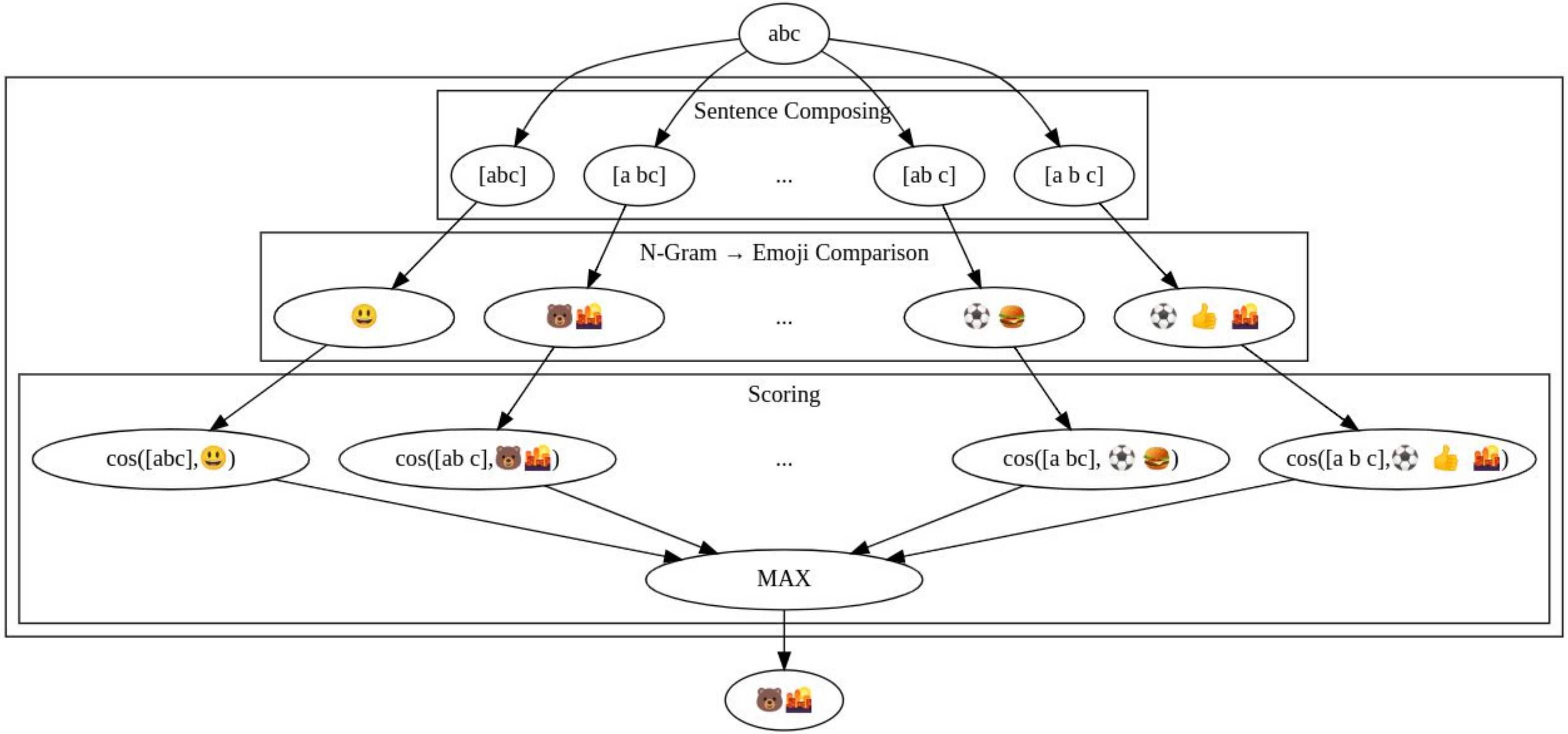
  **end**

  Score summary

**end**

**return** Summary with highest score

# Summary Generation




Combination of N-Grams for Emoji Translation

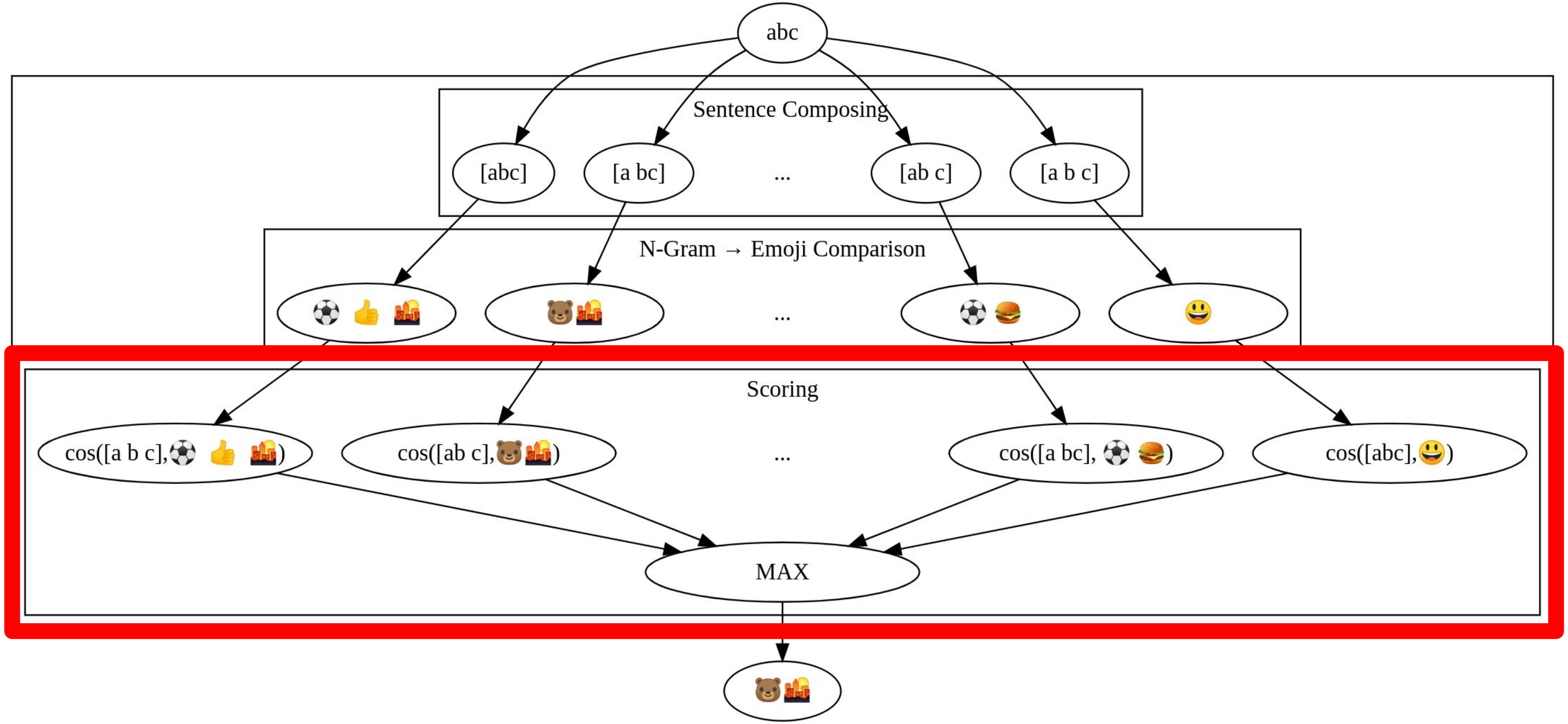
# Drawbacks with this implementation

- Computationally expensive
  - Every word doubles the number of compositions to look at
  - $2^{n-1}$ total compositions to consider, where n is the sentence length
- Words are variably impactful
  - ‘The’, ‘a’, ‘but’, and other similar words
- Little to no context
  - 🐶🌭🐶
    - “A dog eats a treat while another dog watches.”
    - “A dog steals the food of the dog.”
    - “A dog shares his snack with a dog.”

# Addressing the issues

- Computationally expensive
  - Don’t look at every possible combination
  - The natural parts of speech offer an intuitive approach to grouping a sentence
- Variable importance of words
  - Maybe if we could weigh the relative importance of words, we could more fairly score…?
  - Inspiration from the automatic text summarization techniques that we considered years ago
- Context
  - More important for human translators
  - Sentiment analysis - the feeling of a sentence - as an initial attempt

# High-Level Algorithm Architecture

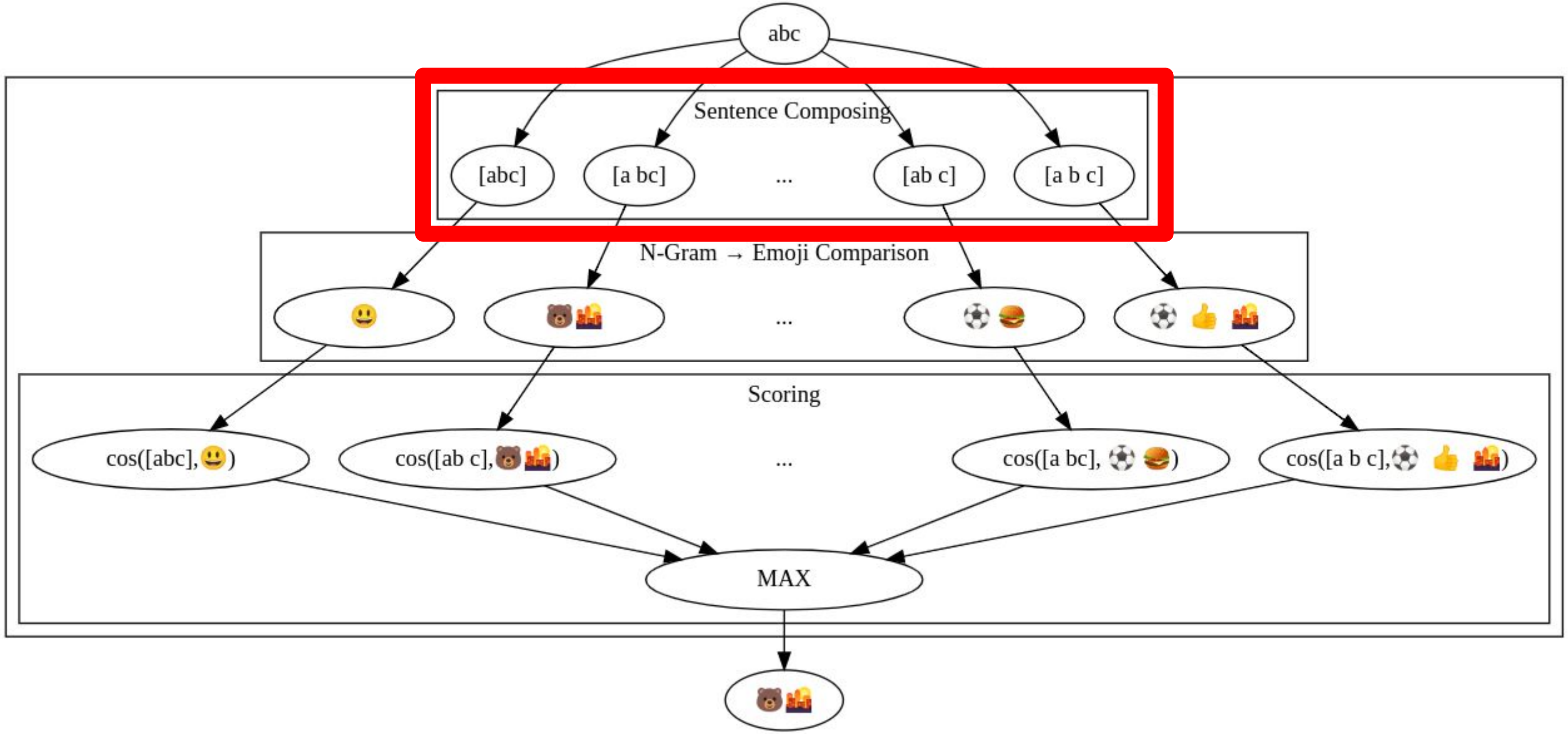



Combination of N-Grams for Emoji Translation

# Tf-idf to high level considerations go here

Made a mistake and need to fix it. Fixing other end first before I write.

# Algorithm Improvements

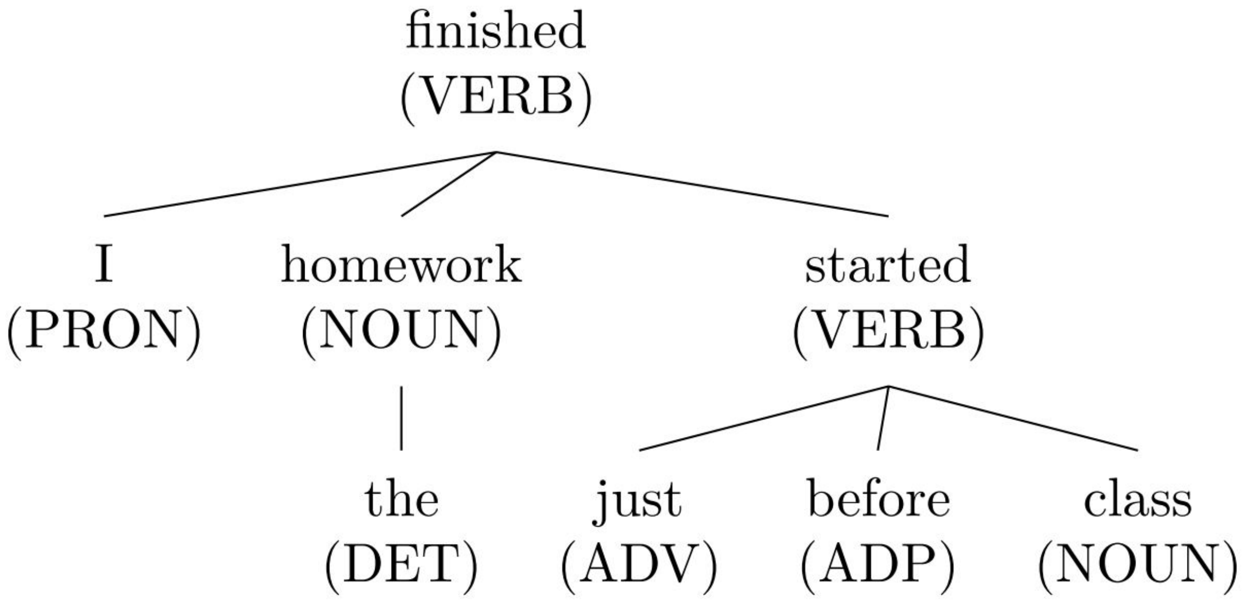



Combination of N-Grams for Emoji Translation

# Dependency Tree Segment Generation

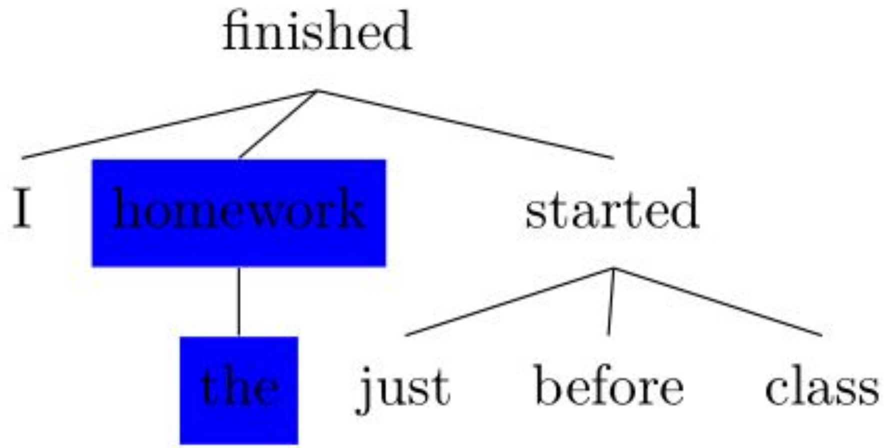
- Exhaustive splitting is
  - Naive
  - Computationally expensive
  - Suffers from reward hacking
- Sentences can be turned into a tree-like structure based on the syntactic dependencies.
- Given that tree we can collapse it down to produce the n-grams from all the remaining nodes:
  1. If there is a node-to-node relationship with only one child we combine them
  2. If there are two or more leafs on the same level we combine them

# Dependency Trees

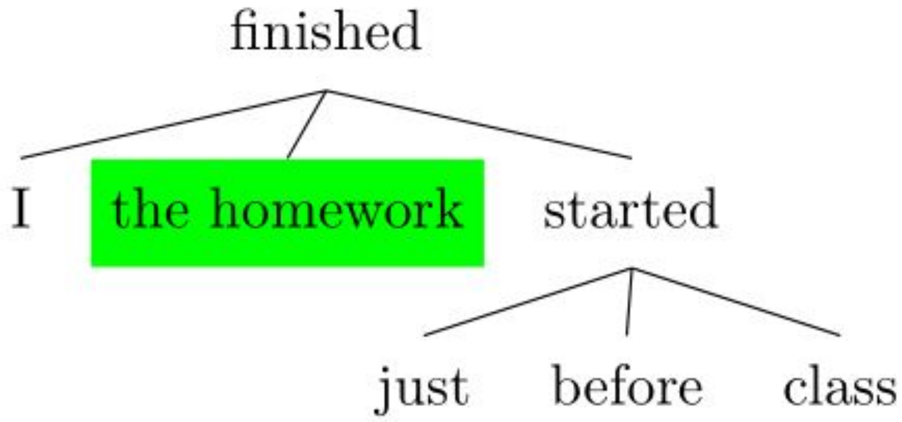



"I finished the homework just before the class started."

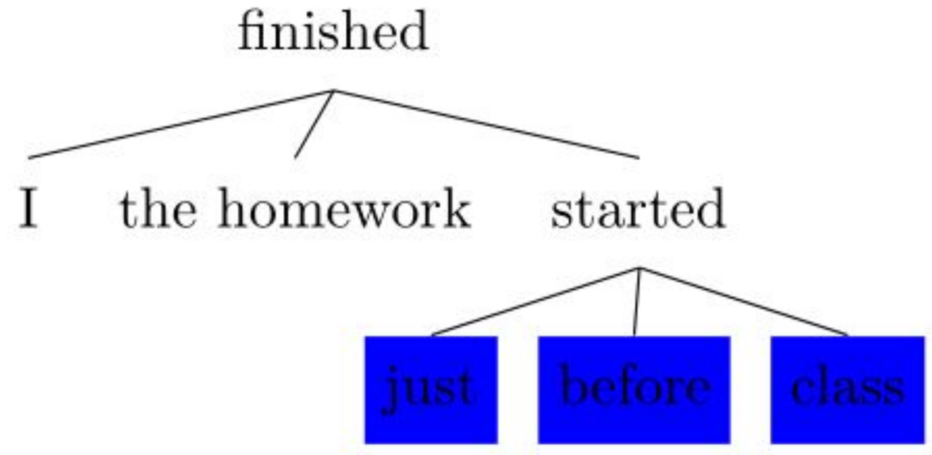
# Child Collapse



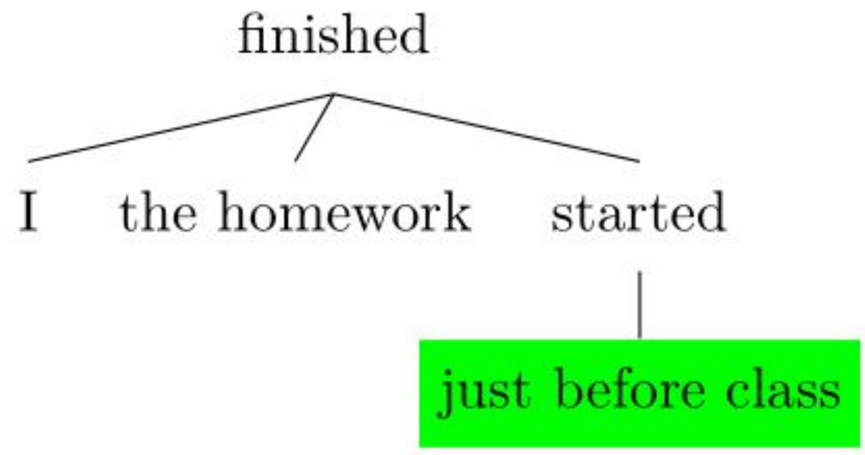
# Child Collapse Continued...



# Neighbor Collapse...



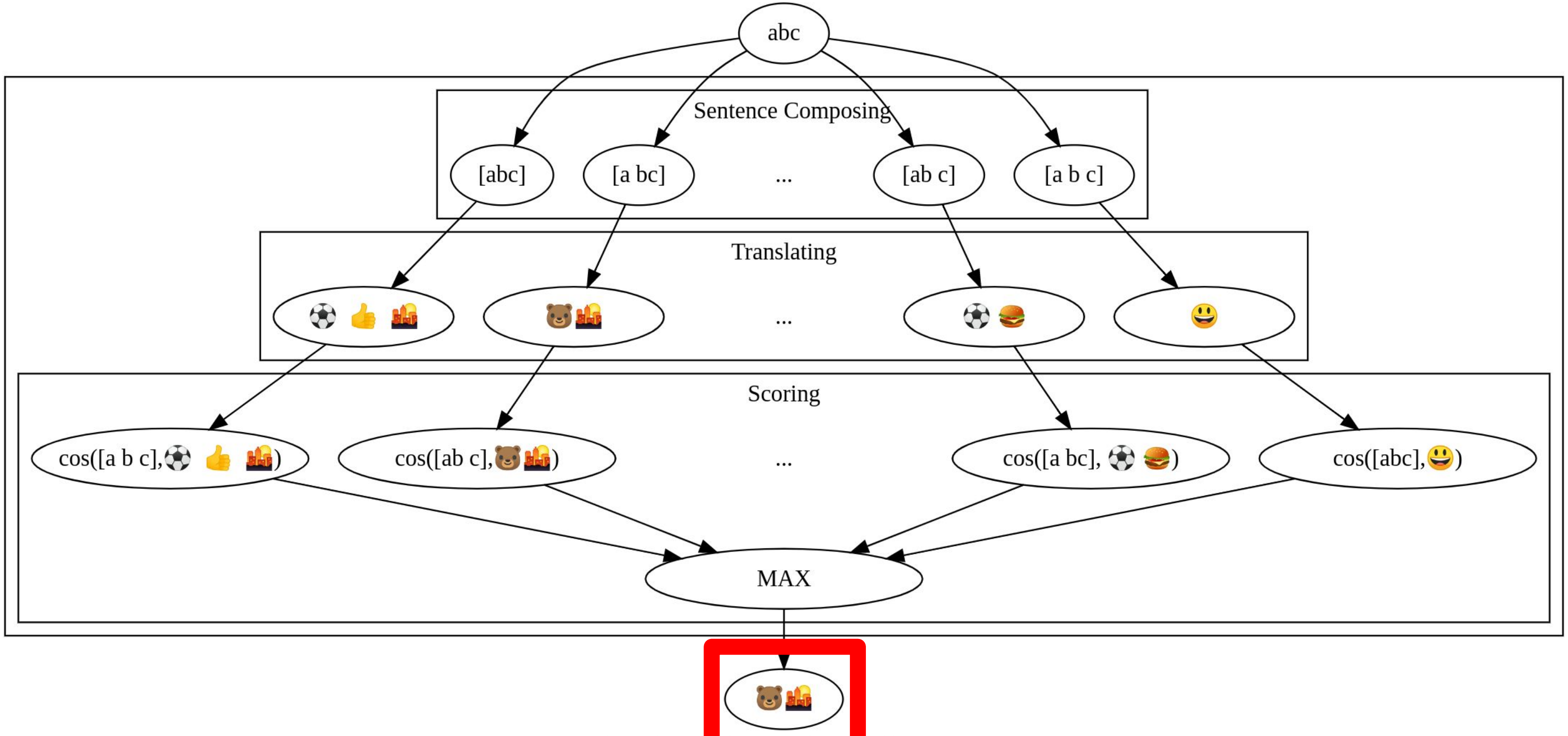
# Neighbor Collapse Continued...



# Dependency Tree Segment Generation Continued...

| | |
|---|---|
| **Sentence 1. The student drew a snowflake on the chalk board** | |
| Exhaustive | The student drew / a / snowflake / on / the chalk board |
| Tree Collapse | The student / drew / a snowflake / the chalk / on board |
| **Sentence 2. I finished the homework just before class started** | |
| Exhaustive | I / finished the homework just before class started |
| Tree Collapse | I / finished / the homework / just before class / started |
| **Sentence 3. Can you calculate the number of giraffes that have ever existed?** | |
| Exhaustive | can / you calculate the number of giraffe that have ever existed |
| Tree Collapse | can you / calculate / number / that have ever / of giraffes existed |

# High-Level Algorithm Architecture




Combination of N-Grams for Emoji Translation

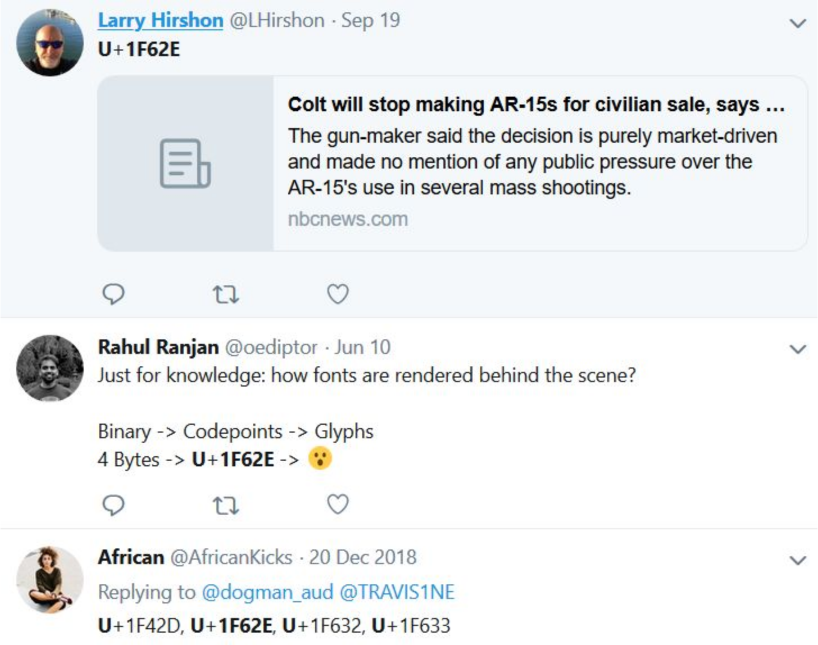
# Sentiment

- Initially, just a table
  - Score a sentence using TextBlob
  - Pull an emoji from a table
- Iteration 2:
  - Use twitter’s API to find tweets containing each emoji
  - Run each tweet through TextBlob
  - Find the mean and standard deviation of each emoji
  - Run the target sentence through TextBlob
  - Find the nearest emoji

| | (Negative) | Polarity | (Positive) |
|---|---|---|---|
| Subjectivity | -1 to -0.33 | -0.33 to 0.33 | 0.33 to 1 |
| (Opinion) 1 to 0.66 | 😡 | 😐 | 😂 |
| 0.66 to 0.33 | 😟 | 😶 | 🙂 |
| (Fact) 0.33 to 0 | 👎 | ⚖️ | 👍 |

# Issues

- Dataset
  - Size (Twitter API rate limit)
  - Query structure issues
    - Searching for “U+1F62E” (😮)
    - Not always in tweet
  - Can only search so far back
    - 😷 results in many Hong-Kong related tweets



# Issues Continued

- Preprocessing is a different beast
  - “Current mood: ”
  - Spelling, punctuation


```
>>> test()
that was tight
Sentiment(polarity=-0.17857142857142858, subjectivity=0.2857142857142857)
>>>
=============== RESTART: C:\Users\Chris\Desktop\NaiveSentiment.py =======
>>> test()
that was tighttttt
Sentiment(polarity=0.0, subjectivity=0.0)
```

# TextBlob conclusion

- More data
- Better preprocessing
- Maybe categorical data analysis combined with with a better dataset (Kaggle)

```python
for keys,values in tweetDict.items():
    print(keys, values)
```

```
['Polarity mean: 0.051', 'Polarity StDev: 0.276', 'Subjectivity mean: 0.294', 'Subjectivity StDev: 0.292']
['Polarity mean: -0.049', 'Polarity StDev: 0.302', 'Subjectivity mean: 0.293', 'Subjectivity StDev: 0.319']
['Polarity mean: 0.008', 'Polarity StDev: 0.146', 'Subjectivity mean: 0.15', 'Subjectivity StDev: 0.202']
['Polarity mean: 0.129', 'Polarity StDev: 0.409', 'Subjectivity mean: 0.456', 'Subjectivity StDev: 0.425']
['Polarity mean: 0.081', 'Polarity StDev: 0.199', 'Subjectivity mean: 0.191', 'Subjectivity StDev: 0.216']
['Polarity mean: 0.011', 'Polarity StDev: 0.304', 'Subjectivity mean: 0.382', 'Subjectivity StDev: 0.311']
['Polarity mean: 0.16', 'Polarity StDev: 0.215', 'Subjectivity mean: 0.308', 'Subjectivity StDev: 0.4']
['Polarity mean: 0.265', 'Polarity StDev: 0.432', 'Subjectivity mean: 0.43', 'Subjectivity StDev: 0.28']
['Polarity mean: 0.064', 'Polarity StDev: 0.161', 'Subjectivity mean: 0.332', 'Subjectivity StDev: 0.285']
['Polarity mean: 0.125', 'Polarity StDev: 0.2', 'Subjectivity mean: 0.187', 'Subjectivity StDev: 0.313']
['Polarity mean: 0.387', 'Polarity StDev: 0.48', 'Subjectivity mean: 0.565', 'Subjectivity StDev: 0.317']
['Polarity mean: 0.212', 'Polarity StDev: 0.331', 'Subjectivity mean: 0.246', 'Subjectivity StDev: 0.255']
['Polarity mean: 0.08', 'Polarity StDev: 0.271', 'Subjectivity mean: 0.33', 'Subjectivity StDev: 0.349']
['Polarity mean: 0.345', 'Polarity StDev: 0.224', 'Subjectivity mean: 0.564', 'Subjectivity StDev: 0.329']
['Polarity mean: -0.004', 'Polarity StDev: 0.3', 'Subjectivity mean: 0.238', 'Subjectivity StDev: 0.286']
['Polarity mean: 0.115', 'Polarity StDev: 0.54', 'Subjectivity mean: 0.428', 'Subjectivity StDev: 0.405']
['Polarity mean: 0.08', 'Polarity StDev: 0.133', 'Subjectivity mean: 0.14', 'Subjectivity StDev: 0.297']
```

# Testing

- No easy quantitative way to score translations
  - BLEU score requires large dataset
- Human in the loop testing
  - Large variance in small sample size of responses
  - Large variance in what our emoji sentences can represent
- Stuck with judging translations ourselves

# Results (Exhaustive Splitting)

| Input Sentence | Output Emojis | Score |
|---|---|---|
| The dog runs fast | 🐩🎽💨 | 0.984 |
| The child was in love with the cat | 👶😚🐾 | 0.824 |
| They are playing christmas music from the bell tower | 🎴🎄🎻⏰🏰 | 0.893 |
| I think that this computer has a virus | 💭💾🐛 | 0.769 |
| I have to wear my headphones to run in the race | 🎩🎧🎽🏁 | 0.960 |
| The company Apple makes both cell phones and computers | 🍏📱💾 | 0.903 |

# Results (Smart Dependency Tree Splitting)

| Input Sentence | Output Emojis | Score |
|---|---|---|
| The dog runs fast | 🎽🐩 | 0.663 |
| The child was in love with the cat | 👶🚯🐾 | 0.629 |
| They are playing christmas music from the bell tower | 🚯🎴🎄🏰 | 0.706 |
| I think that this computer has a virus | 👤💭💯💾🐛 | 0.822 |
| I have to wear my headphones to run in the race | 👤🔬🚯🎩🎧🎽🏁 | 0.668 |
| The company Apple makes both cell phones and computers | 🍏🚯📞📱 | 0.590 |

# Conclusion

- We presented a novel method of translation from English to emoji
- Translates some sentences well, but needs improvement in other areas
- A decent (as far as we can tell) first approach to this problem

# Future Work

- **Improved dataset**
  - The dataset is the main influence on the “readability” of the generated summaries.
  - The dataset we have is aimed at word vectorization rather than sentence vectorization
  - A larger dataset could utilize deep learning techniques
- Each n-gram is currently independent of every other n-gram in the sequence
  - By checking before and ahead and using that to influence the decision it may lead to better results. This is a proven technique used by Recurrent Neural Networks.
- Improve Testing Metrics
  - Translate emojis back into a sentence and calculate distance from the input sentence
- One n-gram can have multiple emojis all with the same similarity. We need some way of determining the closest “closest” emoji.
  - Maybe by considering the emoji’s other keywords as well
  - Consider part-of-speech tagging emoji descriptions

# Demo

## emoji.alexday.me

# Questions?

# TF-IDF

- A way to score the relative importance of a word (term) in a given sentence (document)
- Two parts
  - TF -> Term Frequency: How often a word shows up in the document
    - Motivation: The more often a word shows up, the more important it is to the document
    - Number of times a word shows up in the document
  - IDF -> Inverse Document Frequency: How often a word shows up in all of the documents
    - Motivation: The more things a word shows up in, the less important it is to a single document
    - log(|corpus|/term)
- TF-IDF(term, document, corpus) = TF(term, document) x IDF(term, corpus)

# Estimating n-grams with TF-IDF

- TF-IDF typically considers *individual* words
- Sometimes trained on bigrams or trigrams, but not often much more than that
  - "New York" is common and meaningful, but 2 arbitrary words often aren't
- If we can have a score for arbitrary n-grams, then we can consider weighted averages
  - Gensim is terrific for handling larger datasets, but we cannot consider anything other than unigrams without jumping through some hoops
  - Estimate n-grams out of the constituent unigrams

$$\frac{\sum_{i=1}^{n} w_i x_i}{\sum_{j=1}^{n} w_j}$$

# Steps

- Find TF-IDF scores of each term in the document to translate
- When looking for the score of a larger n-gram, approximate it
  - Individual document frequencies can be multiplied together to get an estimation of how often the words occur together
  - Can estimate how often the words occur in the given order by using the input sentence as an estimation for average sentence length
- Score each composition by multiplying the tf-idf weight of each n-gram with the uncertainty score calculated by cosine difference

```
[['the', 0.087], ['runs', 0.341], ['fast', 0.268], ['dog', 0.304]]
[['the dog', 0.429], ['runs', 0.316], ['fast', 0.255]]
```

# High level considerations

- Consider a bigram
  - Pool of ‘words’ to select from is much larger - would expect frequency of [AB] to be smaller overall
    - Document frequency should be way down
  - Bigrams should score better
- Hugely variable based on training corpus
  - Some were sentences and others were full blown books
  - Assumptions made on word frequencies
  - “The dog runs fast”
    - I looked at datasets of SMS spam/ham, Jeopardy! questions, Canadian Parliament transcriptions, and they heavily impacted the weights.