Language: Goal

Understand some of the elements of language and how they are ***modeled*** in ***programs*** that manipulate language.

- Syntax rules
- n-grams: character, word, phrases, sentence
  - Tokenization
  - Bag of words model
  - Naïve Bayes
  - Word representation – word2vec, Skip-Gram architecture, CBOW
- Neural Networks – Transformer based architecture
  - Recurrent NN with encoding and decoding stages
  - Attention
  - Positional Encoding

Language Models: Example tasks

- automatic summarization, where the AI is given text as input and it produces a summary of the text as output.
- information extraction, where the AI is given a corpus of text and the AI extracts data as output.
- machine translation, where the AI is given a text in the origin language and it outputs the translation in the target language.
- speech recognition, where the AI is given speech, and it produces the same words in text.
- text classification, where the AI is given text and it needs to classify it as some type of text.
- …

Recall Syntax

- Syntax is the set of rules that define the structure of ***sentences in a language*** by defining the atomic elements of a language and how they can be combined to make a more complex units of meaning.
- E.g. Syntax in English sets forth a specific order for grammatical elements like subjects, verbs, direct and indirect objects, etc. For example, if a sentence has a verb, direct object, and subject, the proper order for a declarative sentence is
subject → verb → direct object.

Recall Semantics

- Semantics is the meaning of words or sentences.
- E.g.
  - In python or basic arithmetic the semantics of "+" when given to numbers is that they are to be added to one another.
  - In English, people disagree all the time about the meaning of words, words can mean different things in different contexts E.g.
    - difficult (in reference to a task)
    - expert (in reference to a person's knowledge)
    - truth
    - tolerance
    - safety
    - my program was well tested ☺
    - ...

What is a language model

- A **language model** is a type of **machine learning** model trained to determine a probability distribution over possible next words. A model tries to predict the next most appropriate word to fill in a blank space in a sentence or phrase, based on the context of the given text.
- For example, in a sentence that sounds like this, *"Jenny dropped by the office for the keys so I gave them to [...],"* a good model will determine that the missed word is likely to be a pronoun. Since the relevant piece of information here is *Jenny*, the most probable pronoun is *she* or *her*.
- The important thing is that the model doesn't focus on grammar, but rather on how words are used in a way that is similar to how people write.

Issues in trying to build language models

- Acceptable vs syntactically incorrect
  Just before nine o'clock Sherlock Holmes stepped briskly into the room.
  Just before Sherlock Holmes nine o'clock stepped briskly the room.
- Ambiguity
  I saw the man on the mountain with a telescope.
  Who has the telescope? Man or Mountain??
- Multiple acceptable ways to express same idea
  Just before nine o'clock Sherlock Holmes stepped briskly into the room.
  A few minutes before nine, Sherlock Holmes walked quickly into the room.
- Nonsense but syntactically correct
  Colorless green ideas sleep furiously.

Ideas to help determine meaning and role of word in the Language (English)?

- How do you determine meaning of sentence? Ideas?
  - Word meaning given position in the sentence
  - Word's role in the sentence, part of speech, subject, object, …
  - Roles of the words around it in sentence, roles and meaning
  - Words related but *distant*
- How are these ideas incorporated into Language Models including Large Language Models

Context-Free Grammars : what are the roles words play (Example of type of syntax representation)

- A grammar is a set of rules for putting strings together and so corresponds to a language.
  - A set of variables (e.g. the set of nouns)
  - A set of terminals (e.g. book)
  - A list of productions
- A context free grammar is a four tuple (V, Σ, S, P) where
  - V is a finite set of variables (e.g. noun phrase, noun, adjective, verb)
  - Σ is a finite set of terminals
  - S is the start state
  - P is a finite set of production of the form $V \rightarrow (V \cup \Sigma)^*$

Note: In a **context-free grammar** all the production rules must be of the form **non-terminal symbol → string of terminals and/or non-terminals (variables)** context sensitive grammars allow a more complex left-hand side e.g. $\alpha A \beta$

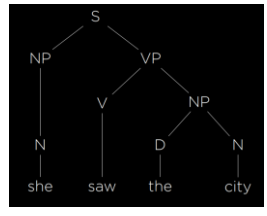
Very simple CFG
As written in nltk (natural language toolkit)

```
S -> NP VP          # Sentence is a noun phrase followed by a verb phrase
NP -> D N | N       # Noun phrase is either a noun or a determiner followed by a noun
VP -> V | V NP      # Verb phrase is either a verb or a verb followed by a noun phrase
D -> "the" | "a“    # Determiner is “the” or “a”
N -> "she" | "city" | "car“   # a noun is …
V -> "saw" | "walked“         # a verb is …
```

Parser: Converts a sentence into how it can be produced through productions. That is, it shows the syntactic structure of the sentence. Usually refers to a part of a compiler.

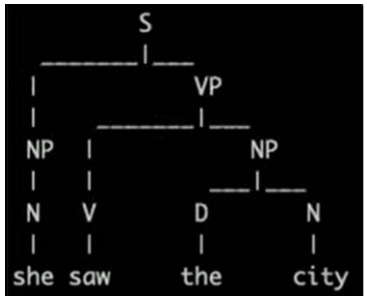
Parsing a sentence using a grammar

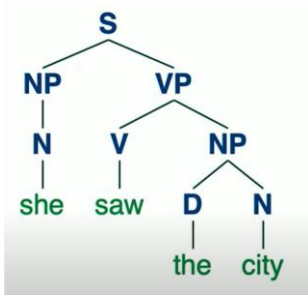
- Sentence: *She saw the city*
  ***Label parts of speech**: N V D N* *individual words are all terminals*
- Why this represents a legal sentence in the grammar
  - A noun phrase (NP) is a group of words that connect to a *noun (N)*. In this grammar it is either a *noun(N) (she)* or *determiner(D) noun(N) (the city)*
  - There is also a verb *saw*
  - A verb phrase is a *(verb(V) noun phrase(NP))* (*saw the city*)
  - Finally, a sentence (*She saw the city)* is a noun phrase (*she*) followed by a verb phrase (*saw the city*)



nltk

- *nltk* (Natural Language Toolkit) is a library that given a grammar will analyze a sentence.
- See the Language in Harvard CS50 video 5:19 to 13:00, especially after 9:30 to 13:00





cfg1

```
import nltk

grammar = nltk.CFG.fromstring("""
   S -> NP VP

   AP -> A | A AP
   NP -> N | D NP | AP NP | N PP
   PP -> P NP
   VP -> V | V NP | V NP PP

   A -> "big" | "blue" | "small" | "dry" | "wide"
   D -> "the" | "a" | "an"
   N -> "she" | "city" | "car" | "street" | "dog" | "binoculars"
   P -> "on" | "over" | "before" | "below" | "with"
   V -> "saw" | "walked"
""")

parser = nltk.ChartParser(grammar)

sentence = input("Sentence: ").split()
try:
    for tree in parser.parse(sentence):
        tree.pretty_print()
except ValueError:
    print("No parse tree possible.")
```

Bag-of-words (BOW) Model

- Bag-of-words is a model that represents text as an unordered collection of words.
  - Ignores syntax
  - Considers only the meanings of the words in the sentence.
- E.g. Categorize by key words – sentiment analysis
  - Positive words – associate with positive reviews
  - Negative words – associate with negative reviews

Bag of Words model: Sentiment Analysis:

Classifying product reviews : *Bag of Words and Naïve Bayes*

- Bag-of-words is a model that represents text as an unordered collection of words. This model ignores syntax and considers only the meanings of the words in the sentence.
- Sentiment Analysis: Classification of reviews, SPAM filters
  (CS 50: Language: Naïve Bayes Section starts at 19:10 - 31:00)
- Base this on Naïve Bayes Rule: Assumes usage of a word is independent of the other words (this is of course incorrect !!! But works in many cases)
- Recall and review incorporating evidence in Bayesian
- Pr (Positive Review | words)

$$= \frac{\Pr(\text{words} \mid \text{Positive Review})\ \Pr(\text{Positive Review})}{\Pr(\text{words})}$$

Finding Words: Tokenization and n-grams

- Tokenization: the task of splitting a sequence of characters into pieces (tokens)
- A contiguous sequence of n items from a sample of text
- Sentence to n-grams (tokens = words)
  - How often have I said to you that when you have eliminated the
    How often have
    often have I
    have I said
    I said to
    said to you
- Use n-grams to predict the next word in a Markov chain. n-gram models assume each word depends only preceding n−1 words: $P(w_i \mid w_1 \ldots w_{i-1})$ =def $P(w_i \mid w_{i-n+1} \ldots w_{i-1})$

2: Markov Models: E.g. Text Generation

- Recall Markov models consist of nodes, the value of each of which has a probability distribution based on a finite number of previous nodes.
- Markov models can be used to generate text.
- Idea:
  - Train a model using text
  - Establish probabilities of n-gram based on the previous n-gram
  - See ***generator.py*** at 17:00 in CS50 Language lecture

Generate.py

```
import markovify  # another python library
import sys

# Read text from file
if len(sys.argv) != 2:
    sys.exit("Usage: python generator.py sample.txt")
with open(sys.argv[1]) as f:
    text = f.read()

# Train model
text_model = markovify.Text(text)

# Generate sentences
print()
for i in range(5):
    print(text_model.make_sentence())
    print()
```


Improving Language Models

- Naive neural models and standard n-gram models have similar shortcomings
  - Models get very large (and sparse) as n increases
  - We can't generalize across similar contexts
  - N-gram Markov (independence) assumptions are too strict
- Better neural language models overcome these by
  - use word embeddings instead of one-hots as input: Instead of representing context words as distinct, discrete symbols they use a dense low-dimensional vector representation where similar words have similar vectors
  - using recurrent nets instead of feed forward nets
  - Instead of a fixed-length (n-gram) context, use recurrent nets to encode variable lengths contexts

Neural Networks in Language Processing:
Example: translation, chatbots

- Translation: Since sentences are not a fixed size, we run into the problem of translating a sequence to another sequence where sizes are not fixed.
- AI chatbot needs to ***understand*** (not same as human understanding) a sequence of words and generate an appropriate sequence as output.
- Need better word meaning representation, relationships between words
- Recurrent neural networks can be designed to handle this.
  - Input is taken into the network, creating a hidden state.
  - Passing a second input into the encoder, along with the first hidden state, produces a new hidden state.

Word Representation: One-Hot Representation

- How to represent ***word meanings*** in a model
- One-Hot Representation: each word is represented with a vector that consists of as many values as we have words.
  - Except for a single value in the vector that is equal to 1, all other values are equal to 0.
  - How we can differentiate words is by which of the values is 1, ending up with a unique vector per word.
- Example, “He wrote a book” can be represented as four vectors:
  - [1, 0, 0, 0] (he) [0, 1, 0, 0] (wrote) [0, 0, 1, 0] (a) [0, 0, 0, 1] (book)
- Problems:
  - *Too many words make the vectors too long*
  - *No clear way to represent “semantics/similarity” of words*
    - » *For _???___ he ate … : e.g. breakfast, lunch, dinner*

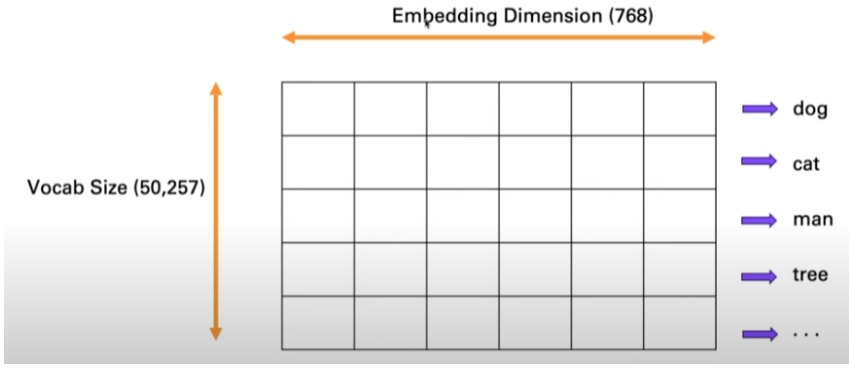
Word Representation: Embeddings

- Idea: *You shall know a word by the company it keeps.* J. R. Firth, 1957
- A word embedding is a function that maps each word type to a single vector in some high dimensional space
- Example: Word2Vec (2013) can be thought of as a classifier of words
- Different architectures for doing this:
  - Skip-Gram Architecture, which is a neural network architecture for predicting context given a target word.
  - CBOW, which is a neural network architecture for predicting single word from given context

Word Embeddings

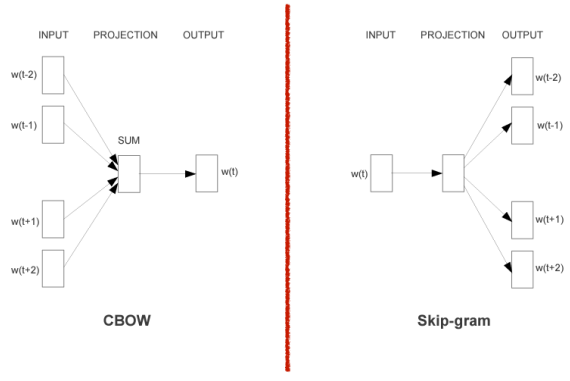
- Desire a representation of words that does not require manual feature engineering, but allows for similarity between related words
- Learned directly from data – a corpus
- word embedding: a vector of numbers representing a word.
  - learned automatically from the data.
  - have additional properties beyond mere proximity for similar words
  - have proven to be a good representation for downstream language tasks (such as question answering or translation or summarization)
  - possible to use generic pretrained vectors
- Pre-trained vector dictionaries include WORD2VEC, GloVe (Global Vectors), and FASTTEXT, which has embeddings for 157 languages.

Embedding from ChatGPT – 2??



Word2Vec architectures

you know a word by the company it keeps

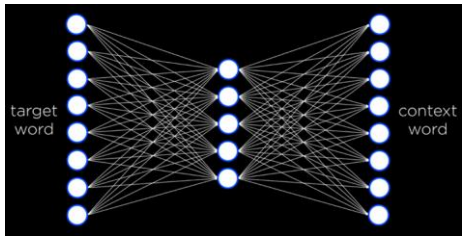


The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. (from /courses.engr.illinois.edu/cs546/)

Word Representation: Embeddings

- Skip-Gram Architecture, which is a neural network architecture for predicting context given a target word.
  - the neural network has an input unit for every target word.
  - A smaller, single hidden layer (e.g. 50 or 100 units, though this number is flexible) will generate values that represent the distributed representations of words.
  - Every unit in this hidden layer is connected to every unit in the input layer.
  - The output layer will generate words that are likely to appear in a similar context as the target words

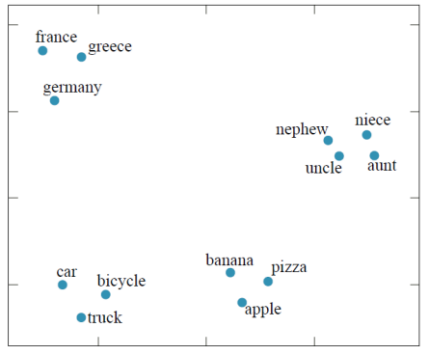
Skip-Gram Architecture



- This neural network turns out to be quite powerful. In the end, of the process, every word ends up being just a vector, or a sequence of numbers.
  - book: [-0.226776 -0.155999 -0.048995 -0.569774 0.053220 0.124401 -0.091108 -0.606255 -0.114630 0.473384 0.061061 0.551323 -0.245151 -0.014248 -0.210003 0.316162 0.340426 0.232053 0.386477 -0.025104 -0.024492 0.342590 0.205586 -0.554390 -0.037832 -0.212766 -0.048781 -0.088652 0.042722 0.000270 …..

Word Embeddings

2-dimensional Principal Component Analysis, PCA



Word embedding vectors computed by the GloVe algorithm trained on 6 billion words of text. 100-dimensional word vectors are projected down onto two dimensions in this visualization. Similar words appear near each other.

Embedding Vectors "*capture concepts*"

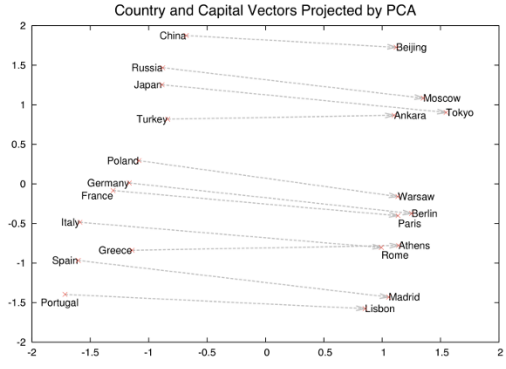


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Vectors "*relationships*"

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Embedding Vectors "*relational meaning*"

Solve *A is to B as C is to [what]? Vector arithmetic*

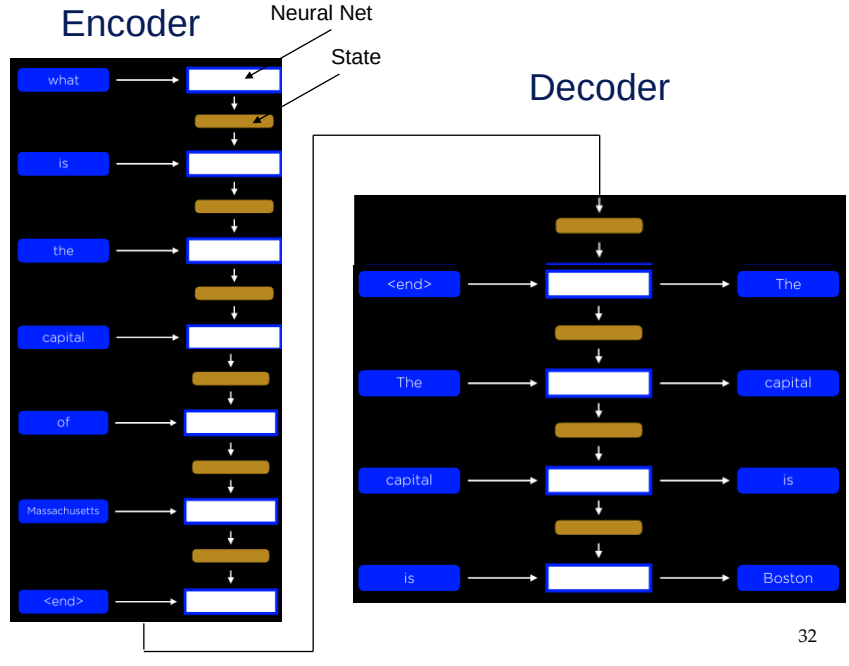
- Use the word embedding vectors for the words A, B, and C, compute the vector D = C + (B–A) and look up the word that is closest to D

A	B	C	D = C + (B − A)	Relationship
Athens	Greece	Oslo	Norway	*Capital*
Astana	Kazakhstan	Harare	Zimbabwe	*Capital*
Angola	kwanza	Iran	rial	*Currency*
copper	Cu	gold	Au	*Atomic Symbol*
Microsoft	Windows	Google	Android	*Operating System*
New York	New York Times	Baltimore	Baltimore Sun	*Newspaper*
Berlusconi	Silvio	Obama	Barack	*First name*
Switzerland	Swiss	Cambodia	Cambodian	*Nationality*
Einstein	scientist	Picasso	painter	*Occupation*
brother	sister	grandson	granddaughter	*Family Relation*
Chicago	Illinois	Stockton	California	*State*
possibly	impossibly	ethical	unethical	*Negative*
mouse	mice	dollar	dollars	*Plural*
easy	easiest	lucky	luckiest	*Superlative*
walking	walked	swimming	swam	*Past tense*

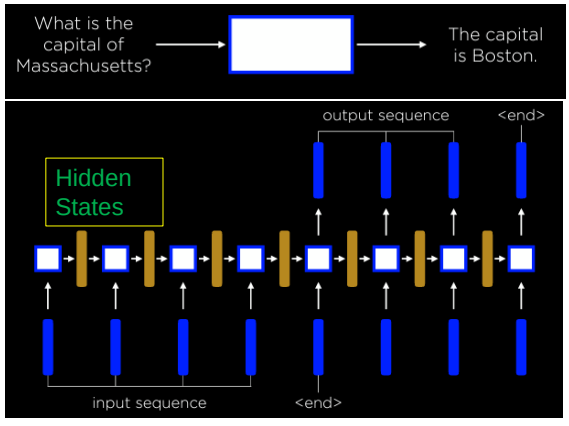
Architecture Neural Network Language Models

- Transformer Based Models: Encoders, Decoders, etc.
  "Attention Is All You Need"

- The encoder stage ideally needs to store all the information from the input stage in one final state.
  1. For large sequences, it's very challenging to store all that information into a single state value.
  2. Some parts of the hidden states in the input sequence are more important than others. How can we know what states (or words) are more important than others?
- Attention
- Transformers



Encoder – Decoder Architecture: Hidden state



Encoder Decoder

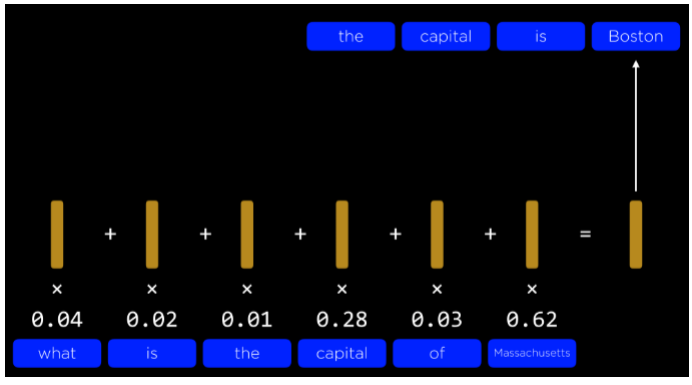
Issues/Problems

1. In the encoder stage all the information from the input stage must be stored in one final state. This gets very large.
   - It would be useful to somehow combine (summarize relevant) information from all the hidden states.
   -
2. Some of the hidden states in the input sequence are more important than others. How can we know what states (or words) are more important than others?
3. What parts of the hidden state are important at different points of the decoding process?!

Attention

- Attention refers to the neural network's ability to decide what values are more important than others.
- In the sentence "What is the capital of Massachusetts?"
  - To generating the final word of the answer, "capital" and "Massachusetts" are the most important to pay attention to.
- By computing an **attention score** a neural network will create a final context vector that the decoder can use to calculate the final word.
- But training a recurrent neural network like this requires sequential training of word after word. This takes a lot of time.
- As large language models grow, they take longer and longer to train. Need a new architecture.
- Difficult to parallelize

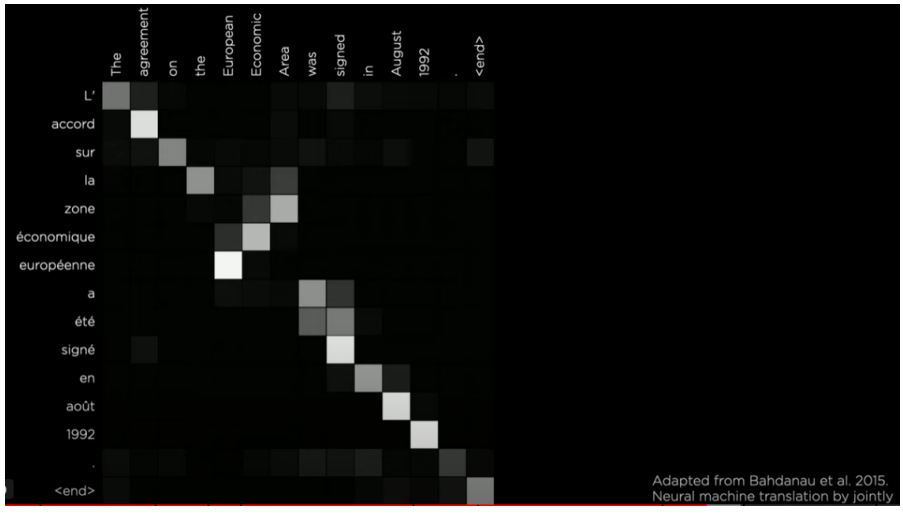
Use of Attention Scores



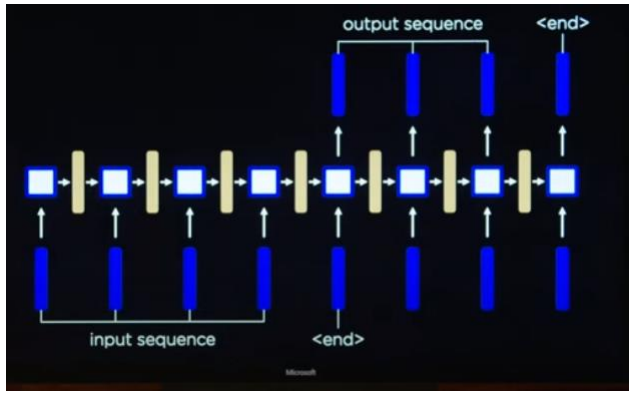
Use of Attention Scores

- Which words to pay attention to.
- During training process the NN learns how to calculate attention scores
- Get a value for each input word determining its importance
  - Each input word is also associated with the hidden state content vector.
  - Take the vectors and compute the weighted average with the attention scores $\rightarrow$ new vector value
  - Represents the hidden states weighted by the attention scores
- This context vector is fed into our decoder to determine what the next word should be, in this case, ***Boston***.

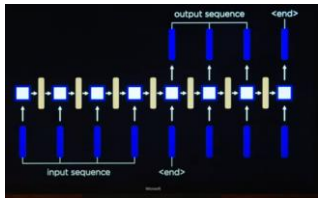
Attention scores from Machine Language Translation

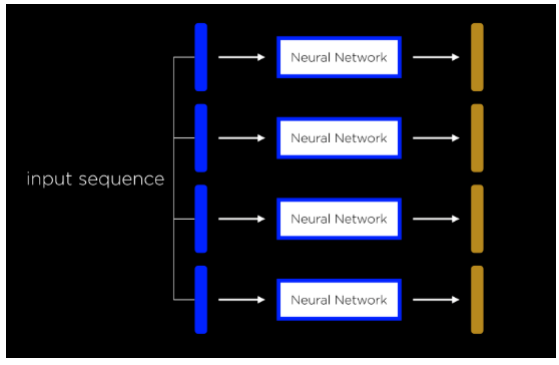


Sequential
Slow - especially as models get larger

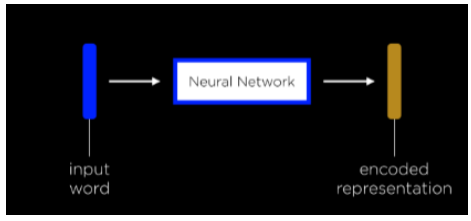


Transformer Architecture: Transformers
Basic Idea: Parallelize

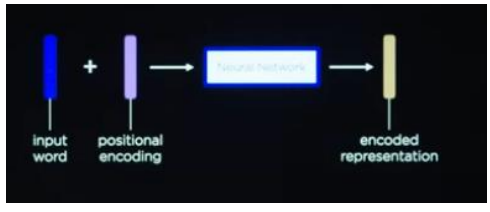




Add Positional Encoding to contain position in the encode representation of the input word

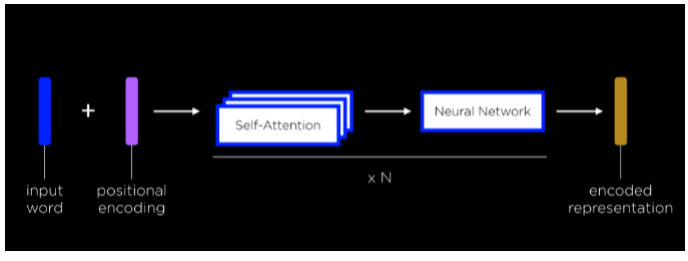


- Each word in parallel
- To keep position to encoding



The words in the output sequence to pay attention to other input words

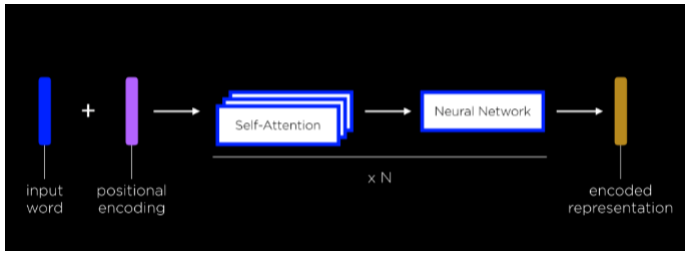
Self Attention: Input word determines what other words to which it needs to pay attention



Multiheaded self attention:

The words in the output sequence to pay attention to other input words

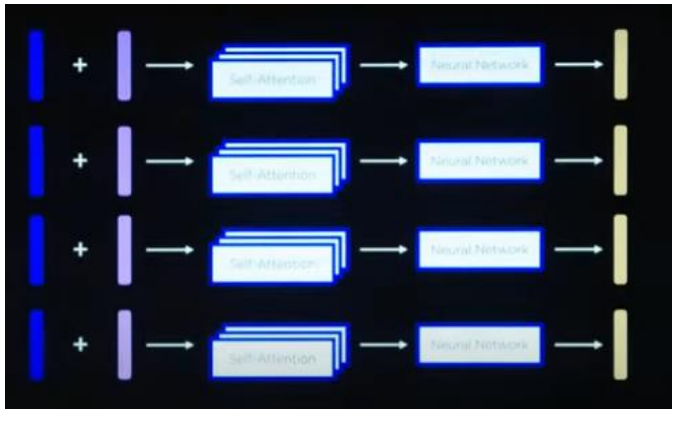
Self Attention: Input word determines what other words to which it needs to pay attention



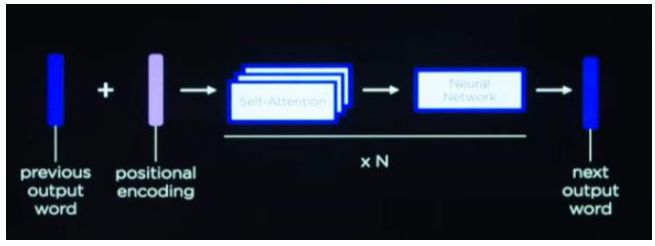
Multiheaded self attention: self attention head might focus on different parts of the input sequence, capturing various aspects or relationships within the data

Repeat for **all** the input words

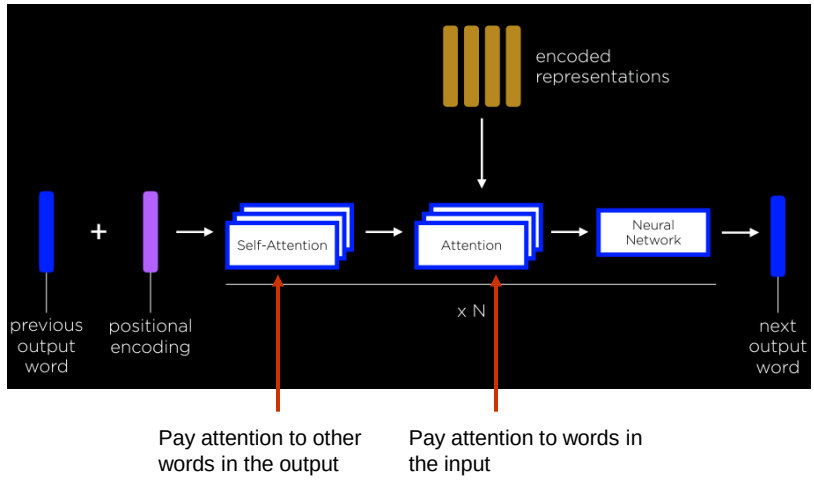
Encoded representations used in decoding



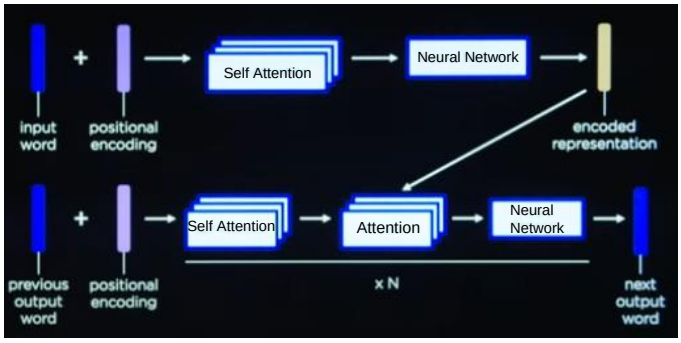
Decoder – similar to encoder but …



Decoder also accepts the encoded representation of the input into attention layers



Full Model: Lots of variation for different variations



Big picture

- Encoded representations that will be useful to when decoding to generate the output sequence.
- Pay attention to both input and output words
- Then the decoder takes the previous output word, pay attention to that encoded input, and then generate the next output word.