* this notebook is for personal study of an exam. naturally, i may skip some topics. i suggest you use it as a north star rather than a rigid checklist.

what is NLP? automating the analysis, generation, and acquisition of human language.

Morphology explores the word structures and their meaningful components.

Studies

Lexemes focuses on normalizing and clarifying word meanings especially with words that have multiple meanings: bank, meaning

Syntax focuses on sentence structures and rules

Semantics turns language to actionable or understandable representations like commands or logic

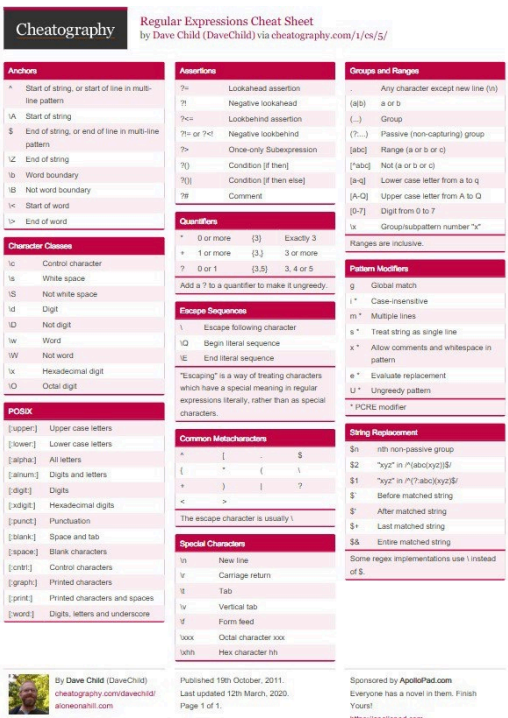
NLP models provide a framework for understanding language from basic text processing to ML

what makes NLP challenging comes down to many factors.

languages involve many layers (sound, grammar, syntax, etc.) which interact with each in many ways. Human language is also ambiguous, words can take multiple meanings and the richness of it allows an idea to be expressed in multiple ways.

Basic Text Processing

regex models strings to describe text patterns



tokenization is to break down text to meaningful units. we have "type" and we have "tokens" that are an instance of a type such as (Cat vs. Cats)

the challenge is with numbers and short/merged forms. isn't, PhD, $63.51

tokenization can be rule-based for faster inference or data-driven for a flexible approach. and tokens are subwords. data-driven uses a large corpus to decide the tokens.

a common tokenizer: **Byte Pair** originally a compression algo, turns most frequent subwords into a new token and comes after a deterministic tokenizer. (BPE used by GPT)

Some basic text operations

normalization: putting words/tokens in standard format. (US vs. U.S) which an example of is Case Folding.

lemmatization: reduce words to their base form.

edit distance

minimum edit distance is the min number of operations to go from text-1 to text-2. Levenstein distance is the same but operations have fixed cost:

- insertion ⇒ 1
- deletion ⇒ 1
- Substitution ⇒ 2

he distance is found using search (dynamic programming)

Language Models

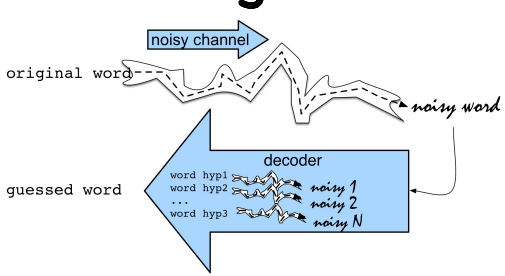
in the topic of error correction and spelling errors there are two types:

Non-word spelling correction for words that are not correct at all (graffe → giraffe)

Solution: find words not in dictionary, suggest words with the least minimum distance prioritized by frequency

Real-word spelling " For words that are in dictionary but used in wrong context (desert vs. dessert)

another solution is **Noisy Channel**



given a misspelled word x, find the correction w such that:

$$\hat{w} = \operatorname{argmax} P(w|x)$$
$$= \operatorname{argmax} \frac{P(x|w) \cdot P(w)}{P(x)}$$

Error	Correction	Transformation: Correct Letter	Error Letter	Position (Letter #)	Type
acress	actress	t	—	2	deletion
acress	cress	—	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	—	s	5	insertion
acress	acres	—	s	4	insertion

Figure B.3 Candidate corrections for the misspelling *acress* and the transformations that would have produced the error (after Kernighan et al. (1990)). "—" represents a null letter.

the noisy channel is a probabilistic model that assumes an observed word is the noisy version of original data. It relies on Baye's rule to find the most likely original word.

The naming can seem strange but it originates from communication systems. "noisy" in our case means misspelled or wrong. "Channel" is the way error rises, which for us is how one misspelled a word. The goal is to model how channel causes noise to mitigate it.

N-Gram Probabilities

Given the last sequences, estimate what word is most common to follow.

$$P(w_i|w_{i-1}) = \frac{Count(w_{i-1}w_i)}{Count(w_{i-1})}$$

What kind of knowledge is captured by this language model?

unigram $P(w_i)$
bigram $P(w_i|w_{i-1})$
⋮

The N-Gram is a simple model that:

- is able to capture syntax (like nouns follow "the")
- has a shallow context window
- unseen N-grams have zero probability
- high order N-grams need much memory & computation
- work purely based on surface patterns

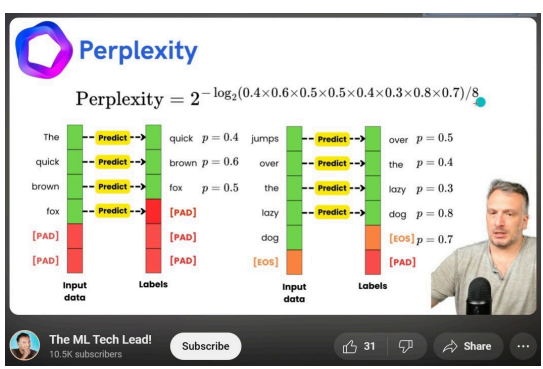
one metric that helps us evaluate a language model is **perplexity**

it evaluates how confident an LM is in a test set

lower perplexity means LM assigns higher probability to correct words

$$Perplexity = 2^{-\frac{1}{T}\sum_{i=1}^{T} \log_2 p(t_i)}$$

T: total number of tokens
$p(t_i)$: probability of ith token in test set



the whole formula is

Average cross-entropy

2

= 1 if model is very confident

= +∞ if it is lame

the origin of the formula goes back to information theory, a perplexity of 10 implies that on average, the model must choose from 10 equally likely options at each step

log is used so that the multiplications of many probabilities doesn't cause numerical underflow (the number would be super small)

Problem: LMs assign 0 to unseen sequences. the solution is **Smoothing** (like add 1 to all frequencies and renormalize)

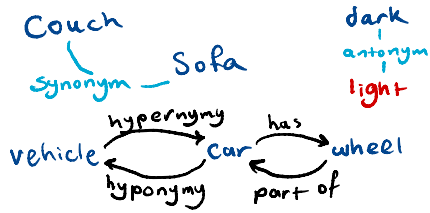
Another solution is backoff to lower order of N-gram, sometimes lowering N helps (better generalization)

Text Classification

there is Naive Bayes and there is logistic Regression which uses defined features (number of positive words, # of "no"s, etc.) won't go into detail

Semantic Vector Embedding

words have many types of association and relatedness with each other.



read — lemma — reading

how to represent a word with these relationships?

one way is to represent a word as a point in an N-dimentional space so a word = vector aka an embedding as it is embedded in this space.

→ enables computing similarity

→ can be learned automatically

embeddings can be derived from Co-occurance matrices: words that appear together often have similar meanings.

(really cool! search it up)

Now raw occurrence counts are not informative on their own → Reweighting

give more weight to more meaningful associations

reduce noise

one method of Reweighting is **tf-idf**

it takes into account both the frequency and the uniqueness of words.

$$TF\text{-}IDF = TF \times IDF$$

TF: how frequent a word t is document d

$$TF(t,d) = \frac{\#\ t \text{ in } d}{\text{total terms in } d}$$

log-scaling can be applied to reduce the effect of high frequencies

$$TF(t,d) = 1 + \log_{10}(Count(t,d))$$

IDF: how unique a word is

$$IDF(t) = \log_{10}\left(\frac{1}{1 + \#\text{documents with } t}\right)$$

the problem with these frequency-based embeddings are:

1. they are too long
2. they are sparse

→ harder for ML models to tune

→ not generalize well (curse of dim)

So it's best to use embeddings that are shorter and less dense

Neural Language Modeling

Simple form: use a feedforward network that predicts next word probability and trains word embeddings.

~ word2vec ~

trains a model to predict how likely two words are to appear close to each other. The goal is to learn good word embeddings with this model.

use two vectors per word:

1. when w is in context
2. when w is an outside word

and average / concat in the end. this helps with optimization.

In the end we throw away the classifier and use the word embeddings.

Properties of embeddings

→ similarity depends on context size during training

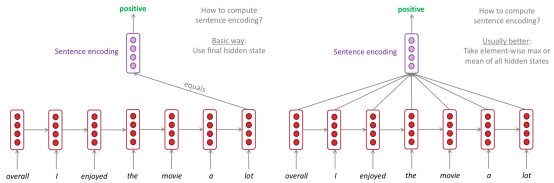
→ words can have first-order co-occurance or second-order (mutual neighbor)

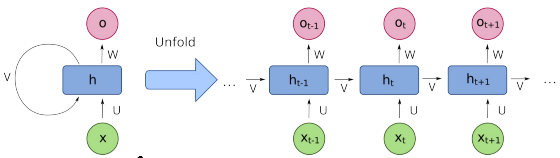
→ they capture relational meanings

Italy − Rome ≃ France − Paris

Recurrent Neural Network

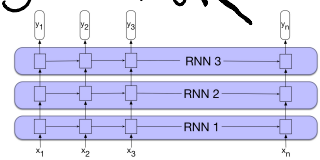
RNNs can also be used for sentence encoding or sentiment analysis by using the hidden state of all or last step.





from wikipedia : RNN (CC)

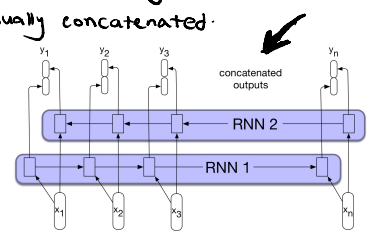
Stacked-RNN is a type of architecture but the cost of training rises rapidly

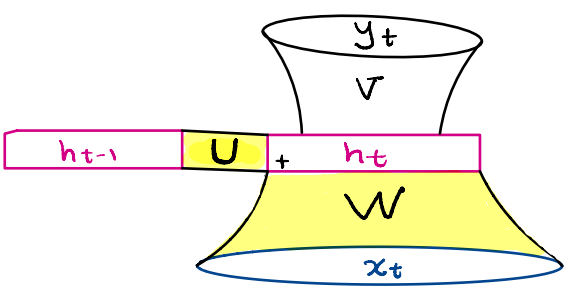


Language modeling is a temporal problem. The problem with N-gram models of fixed window sizes is that the context is limited. Small CW causes degrade in output predictions and big CW causes huge computation.

RNNs can theoretically preserve arbitery long CW size. The hidden state is used from previous steps to process current token.

Bidirectional RNN is to look at a whole sequence (not for generation) and the outputs are usually concatenated.





A transformed hidden state of previous step is added to the transformed input.

$$h_t = g(Uh_{t-1} + Wx_t + b)$$

an activation function

* **h** provides a sort of memory or context
* the context has no fixed size
* model does not scale with the size of the context
* Same weights applied at each step

like an N-gram, RNN can be used for autoregressive generation through repeated sampling.

Start with <s> end with </s>

LSTM Network

LSTM does not guarantee no vanishing gradient but makes it easier to preserve long-term information.

Vanishing Gradient

gradient signals from far away is lost compared to close signals. This can be a serious problem in RNNs and cause it to lose context from further tokens of previous steps.

LSTMs have a hidden state h and a cell state C of the same size. C stores long-term information.

LSTM can write, read, erase information from each cell. Deciding which is controlled by three gates of the same size as h and C.

new cell content: the new content to be written to the cell

$$\tilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$$

Cell State: keep some content from last cell state and write new content.

$$c^{(t)} = f^{(t)} \cdot c^{(t-1)} + i^{(t)} \cdot \tilde{c}^{(t)}$$

hidden State: read some content from the cell state.

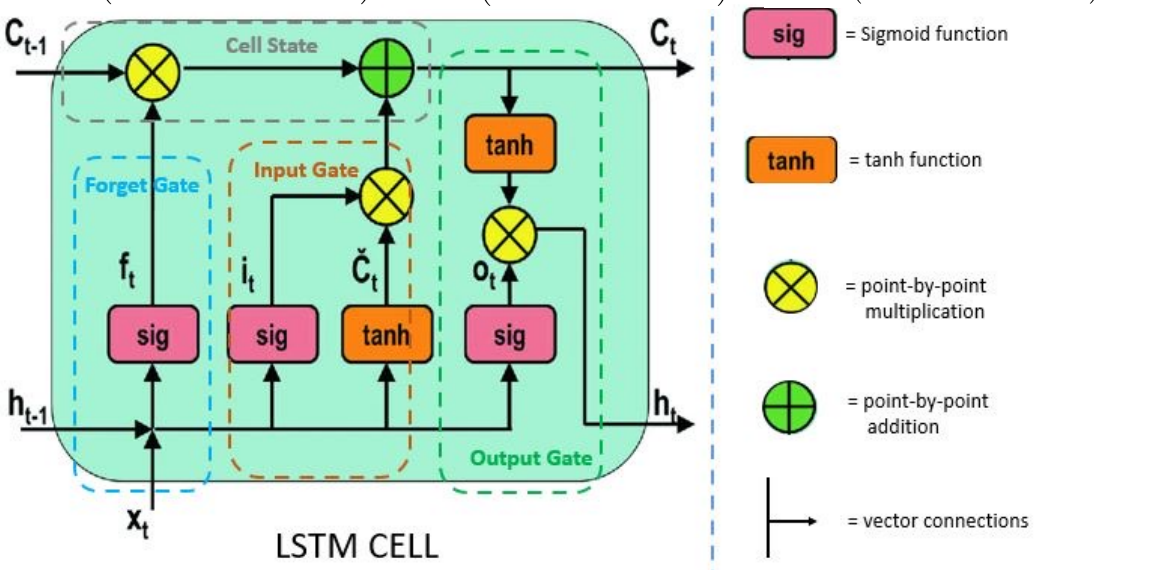
$$h^{(t)} = o^{(t)} \cdot \tanh c^{(t)}$$

forget gate: what is kept or forgotten from previous cell state

Input Gate: what parts of new cell content are written to cell

Output Gate: what parts of cell are outputted to the hidden state

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right) \quad i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right) \quad o^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$$



Gated Recurrent Units (GRU)

a simpler alternative to LSTM

At each step has an input state and a hidden state (no Cell State)

update gate: which parts of hidden state are updated vs. preserved

$$u^{(t)} = \sigma\left(W_u h^{(t-1)} + U_u x^{(t)} + b_u\right)$$

Reset gate: which parts of previous hidden States are are used to compute new content

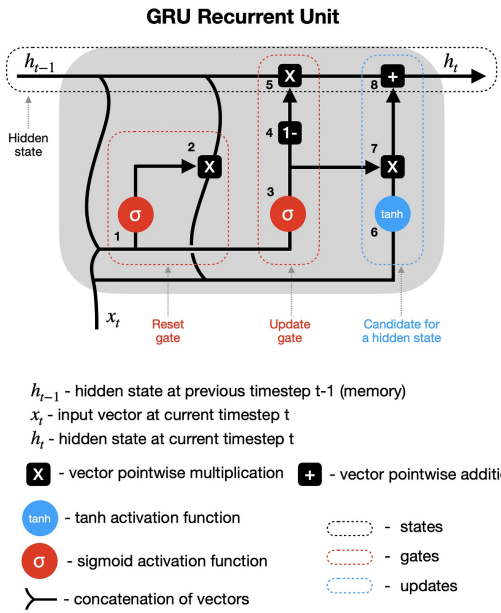
$$r^{(t)} = \sigma\left(W_r h^{(t-1)} + U_r x^{(t)} + b_r\right)$$

New Hidden State Content: reset gate Selects parts of previous hidden state. Use this and the input to compute new content

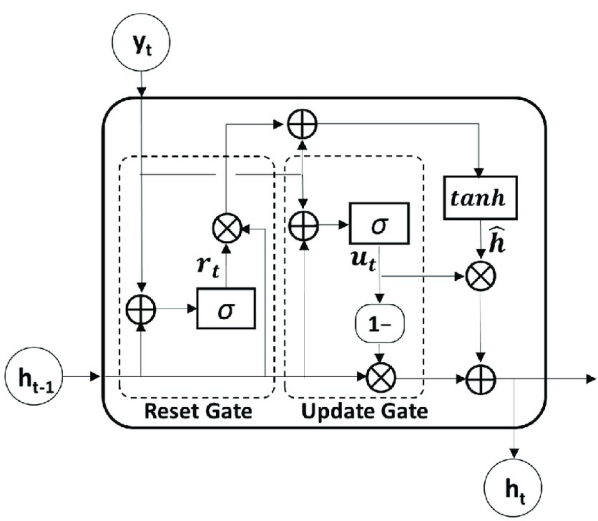
$$\tilde{h}^{(t)} = \tanh\left(W_h(r^{(t)} \cdot h^{(t-1)}) + U_h x^{(t)} + b_h\right)$$

Hidden State: what is kept from previous hidden state (update gate) and what is updated to new content.

$$h^{(t)} = (1 - u^{(t)}) \cdot h^{(t-1)} + u^{(t)} \cdot \tilde{h}^{(t)}$$

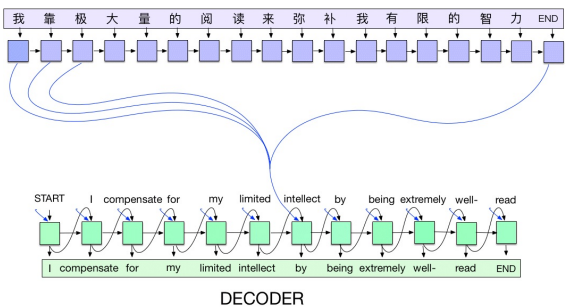


LSTM is powerful but GRU is faster.

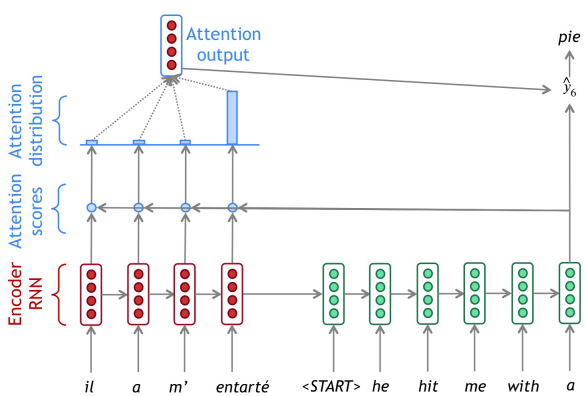


Seq2seq  CNNs for NLP

or also encoder-decoder is a type of network (in this case two RNNs) that take a sequence as input and output a sequence.

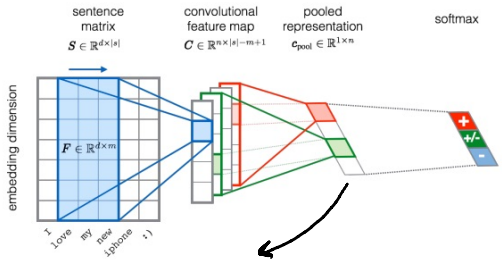


Seq2seq RNN with Attention is another type of network that in each step of the decoderr, it's directly connected to encoder to pay attention to certain parts of encoder.



This attention is so the encoder learns how to give attention to various parts of the input and forget less, it's much more simple than the self-attention mechanism.

the idea is to use 1D convolutions on subsequences of certain length. It also includes max (or average) pooling over time

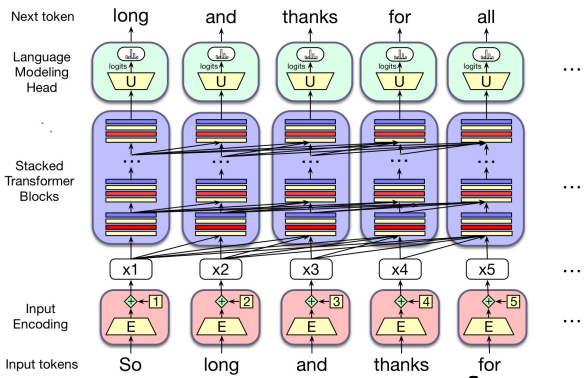


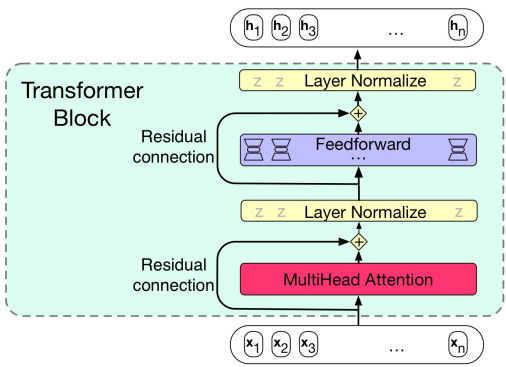
this max-pooling over time goes through the time dimension of each feature and selects the max.

The reason is so that it deals of variable-length sequences, but it is not very plausible and tends to oversimplify the process. On the other hand the CNN networks is parallelized and optimized for GPU. These networks are also more difficult to interpret.

Transformers

the good thing about RNNs is that they have no fixed-length context window (in theory) but suffer from vanishing gradient and slow training. The transformers support parallel processing of input seq.





(from CS224N NLP)

At their heart, use self-attention which obtains a contextualized representation of each token. There are three important roles:

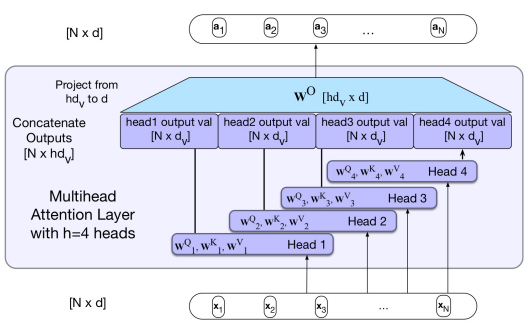
- **query** is the token we're attending to
- **key** other tokens we take into account when attending to query (including the query)
- **value** is what we compute attention scores for, can be anything.

$$q_i = x_i W^Q \quad k_i = x_i W^K \quad v_i = x_i W^V$$

the whole formula can be written as:

$$\text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V$$

we use **multi-head** attention to get context of tokens in multiple ways.



the trns block has other layers as well:

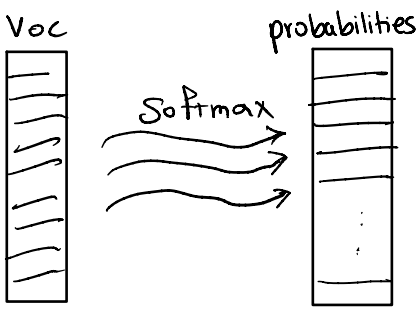
- **Feedforward** layer for more process
- **Residual Connections** to flow the gradient and prevent it from vanishing.
- **layer Norm** to keep the weights in limit and stabilize training.

Large Language Models

Conditional generation is when LLM outputs response based on a given prompt. But many task can be done without a prompt and with word prediction.

Decoding can be **greedy**: select word of highest probability. Deterministic.

Beam Search is another method.



In Beam Search we select the K highest probability tokens. If K=2 we select w_1 and w_2 and feed each to the model again and get two new sets of probs P_1 and P_2 and we multiply each w with its P: w_1P_1 w_2P_2 and we get two vectors.

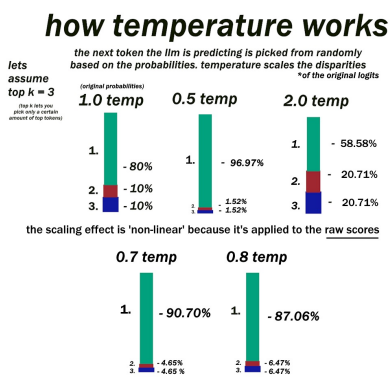
If K=1 then you have greedy search but Beam is a general case that lets you look into potential paths down a search tree.

Sampling is most popular. It is Probabilistic and allows to tune between more random generation or more factual.

Top-K truncates the top k most scored tokens, renormalize, and sample

Top-P select vocabs with at least p probability.

Temperature reshapes the probability distribution by dividing logits by T $y = \text{Softmax}(u/T)$



Skipped Pretraining types of Fine-tuning:

1. Continual Pretraining
2. Parameter Efficient FT (PEFT) update some params and keep the rest frozen
3. Task-specific FT by adding a head to the model (like for classification)
4. Supervised FT which requires a dataset of dedicated prompt/responses

a type of language models is bidirectional encoder like BERT that takes an entire sequence + generated output so far. Best for translation, sentiment classification, etc.

Does not use masked attention.

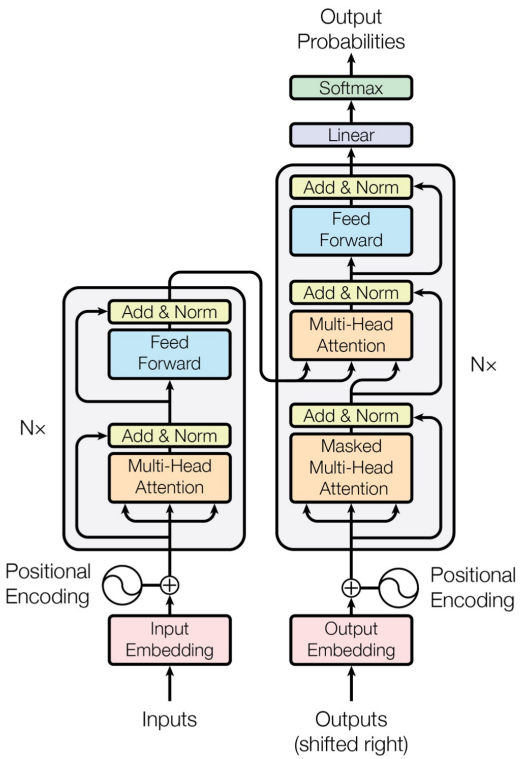
They aren't trained to predict next word but similar to denoising tasks, predicts tokens in the middle of sequence.

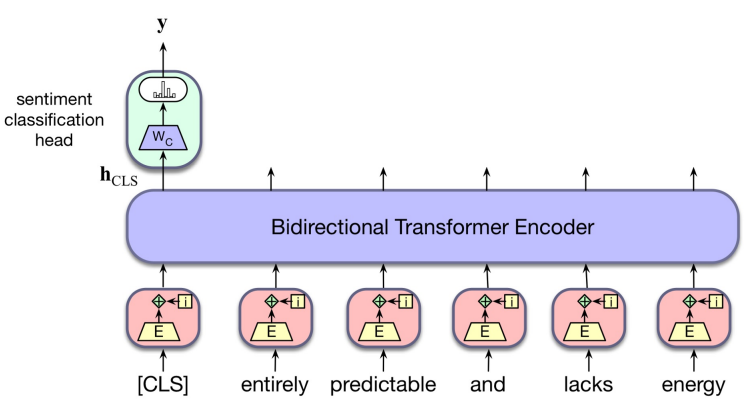
tokenizer fertility is a measure of how many tokens on average are needed for a word. Impacts inference speed of models.

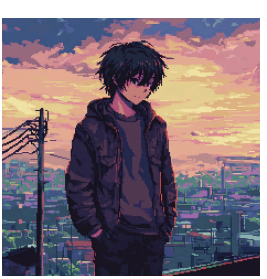
language models can also be used for classification. Encoder-decoder models like BERT can do this with a special [CLS] token. Also last layers can be changed and fine-tuned.

Encoder-decoder

decoders are trained to predict next words encoders are trained to predict masked words in a sequences. These models benefit from bidirectional processing of text.







problem with large LLMs

these models often perform better, high cost, environmental footprint,

Some Methods to Reduce Computation
- Parameter Sharing
- Pruning
- Knowledge Distillation
- Mixed Precision

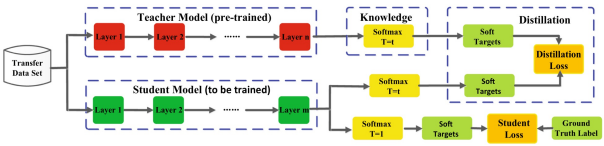


Fig. 5 The specific architecture of the benchmark knowledge distillation (Hinton et al. 2015)

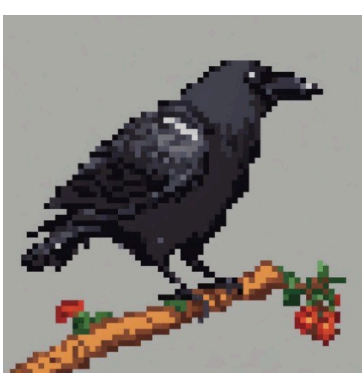
Spectrum of Learning

zero shot — basic prompting | one-shot — single example | few-shot | full fine-tuning

(Read about prompting, Chain-of-thought)

RLHF

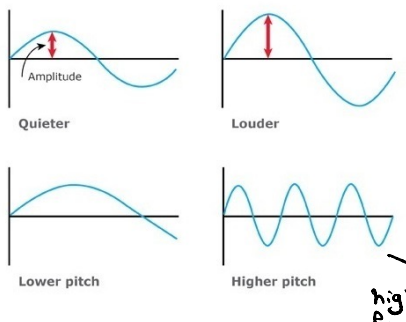
1. pre-train model
2. train a reward model → map preferences into numbers
3. fine-tune based on reward



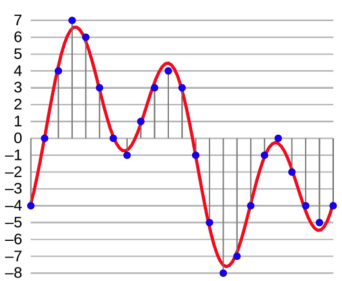
Speech Processing & Multi Modality

Speech processing is a bridge between NLP and signal processing.

Sound is vibration of air molecules. Pitch is the frequency and loudness is the amplitude



Signal is the representation of sound waves. It's analog by nature but with sampling and quantization can be converted into digital form.



In Analog Digital Conversion (ADC) signal is first sampled and the quantized.

(Read about Fourier Transform and its variances)

Audio has many features we can extract:
Pitch, rhythm, stress patterns, zero-crossing of waveforms, length and loudness,

etc...

typical applications of Speech & sound modelling

Speech Recognition — Sound → Text

Text 2 Speech — Text → Sound

Speech Seperation — seperating different sources

Speech Enhancement — increases quality

The typical **ASR architecture** is encoder-decoder sequence to sequence with Transformers or RNN's.

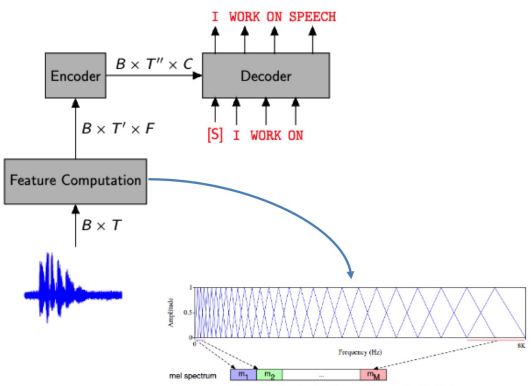


Image taken from Speech and Language Processing, Jurafsky, Martin

The Sound waves are represented as a spectrum or time-frequency that is fed int Encoder to extract patterns which is then handled by the Decoder.

with TTS (tranfer text to audio) the pipeline is again **encoder-decoder** which takes text, the encoder turns it into a latent representation which is given to a decoder to produce a mel-spectogram.

Speech Foundational Models

general-purpose transformer-based models for speech. a challenge with speech is that it can be very variable. SFTs can mitigate this challenge bc:

1. they learned robust representations from huge data
2. they leverage advanced processing techniques

These SFTs can be used as feature extractors or a head can be mounted on them and fine-tuned for specific tasks.