Data Mining

Language Mining

https://data-mining.github.io/winter-2026/

CS 453/553 – Winter 2026
Yu Wang, Ph.D.
Assistant Professor
Computer Science
University of Oregon
Slide Courtesy: Zhiting Hu from UCSD

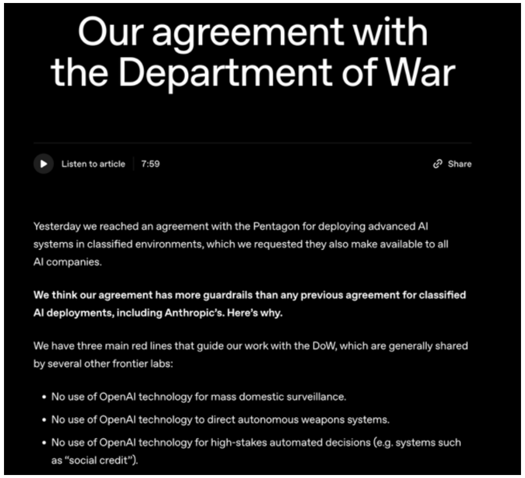
Where we are?

	03/02/2026 00:00 Monday	Presentation 6	**Group** ◦ Group 11: 7-7:15 pm ◦ Group 12: 7:15-7:30 pm ◦ Zoom
Lecture	03/02/2026 Monday	Language Mining	**Course Materials:** ◦ Slides
	03/04/2026 00:00 Wednesday	Presentation 6	**Group** ◦ Group 13: 7-7:15 pm ◦ Group 14: 7:15-7:30 pm ◦ Zoom
Lecture	03/04/2026 Wednesday	Language Mining 2	**Course Materials:** ◦ Slides
	03/09/2026 00:00 Monday	Presentation 6	**Group** ◦ Group 15: 7-7:15 pm ◦ Group 16: 7:15-7:30 pm ◦ Zoom
Lecture	03/09/2026 Monday	Review Future	**Course Materials:** ◦ Slides
Exam	**03/11/2026 01:20 Wednesday**	**Quzz 2**	Topics: ◦ Lecture 9 - Lecture 16 ◦ Closed Book
Due	**03/20/2026 23:59 Friday**	**Project Report Due**	

Language Mining

Final Week

Fun Facts



Our agreement with the Department of War

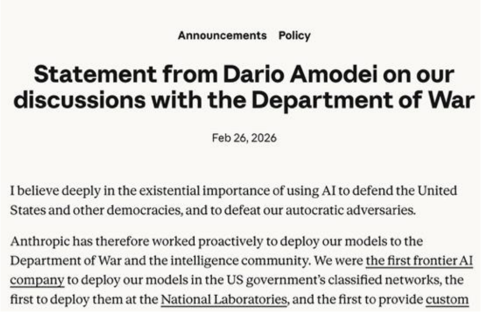
Listen to article 7:59 Share

Yesterday we reached an agreement with the Pentagon for deploying advanced AI systems in classified environments, which we requested they also make available to all AI companies.

We think our agreement has more guardrails than any previous agreement for classified AI deployments, including Anthropic's. Here's why.

We have three main red lines that guide our work with the DoW, which are generally shared by several other frontier labs:

- No use of OpenAI technology for mass domestic surveillance.
- No use of OpenAI technology to direct autonomous weapons systems.
- No use of OpenAI technology for high-stakes automated decisions (e.g. systems such as "social credit").



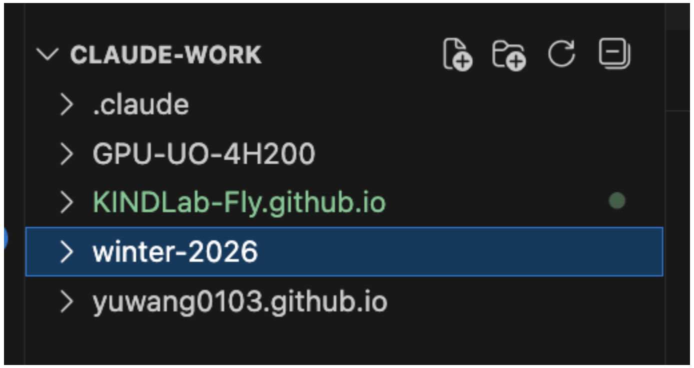
Announcements Policy

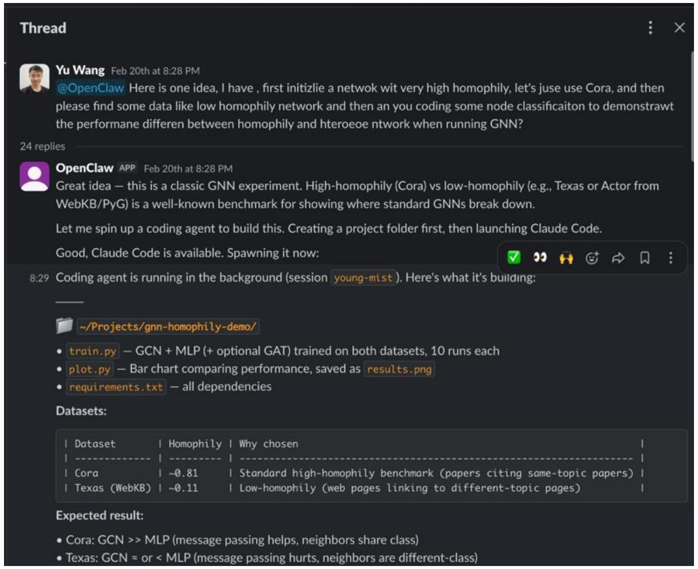
Statement from Dario Amodei on our discussions with the Department of War

Feb 26, 2026

I believe deeply in the existential importance of using AI to defend the United States and other democracies, and to defeat our autocratic adversaries.

Anthropic has therefore worked proactively to deploy our models to the Department of War and the intelligence community. We were the first frontier AI company to deploy our models in the US government's classified networks, the first to deploy them at the National Laboratories, and the first to provide custom





Thread

Yu Wang Feb 20th at 8:28 PM
@OpenClaw Here is one idea, I have , first initizlie a netwok wit very high homophily, let's juse use Cora, and then please find some data like low homophily network and then an you coding some node classificaiton to demonstrawt the performane differen between homophily and hteroeoe ntwork when running GNN?

24 replies

OpenClaw APP Feb 20th at 8:28 PM
Great idea — this is a classic GNN experiment. High-homophily (Cora) vs low-homophily (e.g., Texas or Actor from WebKB/PyG) is a well-known benchmark for showing where standard GNNs break down.

Let me spin up a coding agent to build this. Creating a project folder first, then launching Claude Code.

Good, Claude Code is available. Spawning it now:

8:29 Coding agent is running in the background (session `young-mist`). Here's what it's building:

`~/Projects/gnn-homophily-demo/`

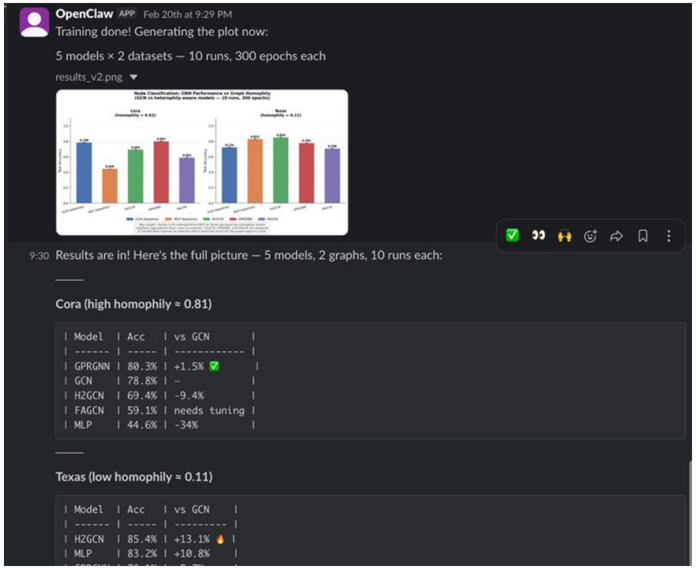
- `train.py` — GCN + MLP (+ optional GAT) trained on both datasets, 10 runs each
- `plot.py` — Bar chart comparing performance, saved as `results.png`
- `requirements.txt` — all dependencies

Datasets:

Dataset	Homophily	Why chosen
Cora	~0.81	Standard high-homophily benchmark (papers citing same-topic papers)
Texas (WebKB)	~0.11	Low-homophily (web pages linking to different-topic pages)

Expected result:

- Cora: GCN >> MLP (message passing helps, neighbors share class)
- Texas: GCN ≈ or < MLP (message passing hurts, neighbors are different-class)



OpenClaw APP Feb 20th at 9:29 PM
Training done! Generating the plot now:

5 models × 2 datasets — 10 runs, 300 epochs each

results_v2.png

9:30 Results are in! Here's the full picture — 5 models, 2 graphs, 10 runs each:

Cora (high homophily ≈ 0.81)

Model	Acc	vs GCN
GPRGNN	80.3%	+1.5% ✅
GCN	78.8%	–
H2GCN	69.4%	-9.4%
FAGCN	59.1%	needs tuning
MLP	44.6%	-34%

Texas (low homophily ≈ 0.11)

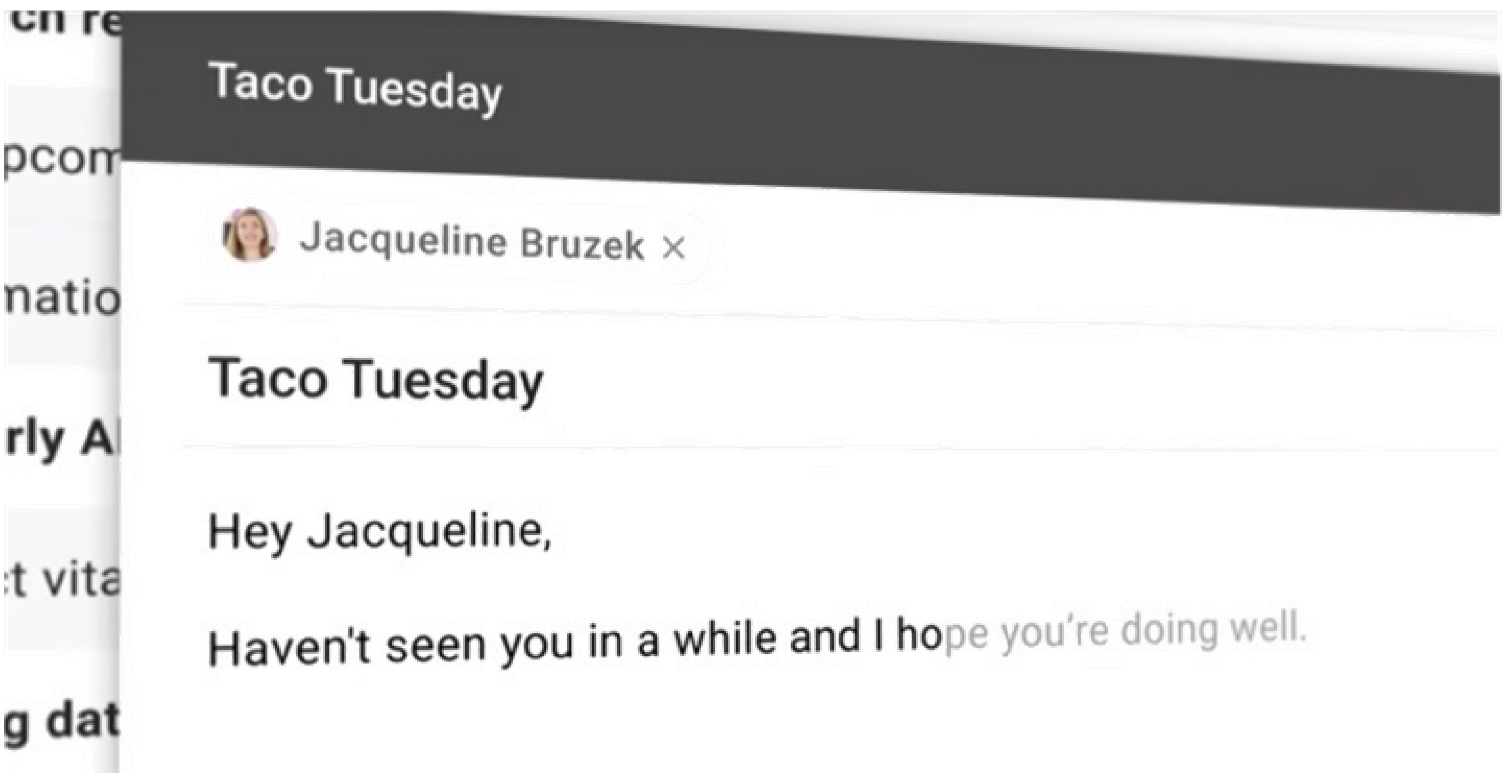
Model	Acc	vs GCN
H2GCN	85.4%	+13.1% 🔥
MLP	83.2%	+10.8%

Summary so far

- language models utilities
  - Generation, evaluation of fluency, few-shot prediction (GPT3), …
- N-gram language models
  - Unigram LM
  - N-gram LM
- Neural language models:
  - Embedding: one-hot vectors -> embedding vectors
  - Neural networks

Motivations of Language Models

- Generation



Email auto-completion

Motivations of Language Models

- Generation

Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article: **After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination.**
The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.

Figure 3.14: The GPT-3 generated news article that humans had the greatest difficulty distinguishing from a human written article (accuracy: 12%).

Motivations of Language Models

- Evaluation of language fluency
  - Ex:

Successful speech recognition requires generating a word sequence that is:

- ▶ Faithful to the acoustic input
- ▶ Fluent

If we're mapping acoustics $\boldsymbol{a}$ to word sequences $\boldsymbol{w}$, then:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, \text{Faithfulness}(\boldsymbol{w}; \boldsymbol{a}) + \text{Fluency}(\boldsymbol{w})$$

Language models can provide a "fluency" score.

Motivations of Language Models

- Evaluation of language fluency
  - Ex: Other tasks that have text (or speech) as output:
    - translation from one language to another
    - conversational systems
    - document summarization
    - image captioning
    - optical character recognition
    - spelling and grammar correction

    If we're mapping inputs $\boldsymbol{i}$ to word sequences $\boldsymbol{w}$, then:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, \text{Faithfulness}(\boldsymbol{w}; \boldsymbol{i}) + \text{Fluency}(\boldsymbol{w})$$

    Language models can provide a "fluency" score.

Motivations of Language Models

- Few-shot prediction (e.g., GPT3)

Prompts in Red. GPT3 responses in Blue

Q. How do you draw a bicycle?
A. You draw a bicycle with two circles connected by a line.

Q. How do you draw a ball?
A. You draw a ball with one circle.

Q. How do you draw a house?
A. You draw a house with a trapezium on top of a square.

Q. How do you draw a rabbit?
A. You draw a rabbit with four circles: one large circle for the head, a small one for the nose, a large circle for the body, and two tiny circles for the feet.

Q. How do you draw a snail?
A. You draw a snail with six circles: one large circle for the shell, two small circles for the antennae, one large circle for the head, one small circle for the tail, and a small circle for the mouth.

Notations

- $\mathcal{V}$ is a finite set of (discrete) symbols (words or characters); $V = |\mathcal{V}|$
- $\mathcal{V}^*$ is the (infinite) set of sequences of symbols from $\mathcal{V}$
- In language modeling, we imagine a sequence of random variables $X_1, X_2, \ldots$ that continues until some X_n takes the value "$\bigcirc$" (a special end-of-sequence symbol).
- $\mathcal{V}^\dagger$ is the (infinite) set of sequences of $\mathcal{V}$ symbols, with a single $\bigcirc$, which is at the end.

The Language Modeling Problem

- Input: training data $\boldsymbol{x} = (x_1, x_2, \dots, x_N)$ in $\mathcal{V}^\dagger$
  - (assuming one instance $\boldsymbol{x}$ for simplicity of notations)
- Output: p: $\mathcal{V}^\dagger \rightarrow \mathbb{R}$
- Think of p as a measure of plausibility

Probabilistic Language Model

- We let p be a probability distribution, which means that

$$\forall \boldsymbol{x} \in \mathcal{V}^{\dagger}, p(\boldsymbol{x}) \geq 0$$
$$\sum_{\boldsymbol{x} \in \mathcal{V}^{\dagger}} p(\boldsymbol{x}) = 1$$

- Advantages:
  - Interpretability
  - We can apply the maximum likelihood principle to build a language model from data

Decomposing using the Chain Rule

$$
p(\boldsymbol{X} = \boldsymbol{x}) = \begin{pmatrix} p(X_1 = x_1) \\ \cdot\, p(X_2 = x_2 \mid X_1 = x_1) \\ \cdot\, p(X_3 = x_3 \mid \boldsymbol{X}_{1:2} = \boldsymbol{x}_{1:2}) \\ \vdots \\ \cdot\, p(X_N = \bigcirc \mid \boldsymbol{X}_{1:N-1} = \boldsymbol{x}_{1:N-1}) \end{pmatrix}
$$

$$
= \prod_{i=1}^{N} p(X_i = x_i \mid \boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1})
$$

Example:

Predict each word based on the "history"

$\boldsymbol{x}$ = *(I, like, this, movie, …)*

$$
p(\boldsymbol{x}) = \cdots\, p_\theta(like \mid I)\, p_\theta(this \mid I, like) \cdots
$$

Summary so far

- language models utilities
  - Generation, evaluation of fluency, few-shot prediction (GPT3), …
- N-gram language models
  - Unigram LM
  - N-gram LM
- Neural language models:
  - Embedding: one-hot vectors -> embedding vectors
  - Neural networks

Unigram Model: Empty History

$$p(\boldsymbol{X} = \boldsymbol{x}) = \prod_{i=1}^{N} p(X_i = x_i \mid \boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1})$$

Multinomial distribution

$$\overset{\text{assumption}}{=} \prod_{i=1}^{N} p(X_i = x_i; \boldsymbol{\theta}) = \prod_{i=1}^{N} \theta_{x_i}$$

Maximum likelihood estimate: for every $v \in \mathcal{V}$,

$$\theta_v^* = \frac{\sum_{i=1}^{N} \mathbf{1}\{x_i = v\}}{N}$$

$$= \frac{\text{count}_{\boldsymbol{x}}(v)}{N}$$

Language Mining – N-gram

Example

The probability of

Presidents tell lies .

is:

$$p(X_1 = \text{Presidents}) \cdot p(X_2 = \text{tell}) \cdot p(X_3 = \text{lies}) \cdot p(X_4 = .) \cdot p(X_5 = \bigcirc)$$

In unigram model notation:

$$\theta_{\text{Presidents}} \cdot \theta_{\text{tell}} \cdot \theta_{\text{lies}} \cdot \theta_{.} \cdot \theta_{\bigcirc}$$

Using the maximum likelihood estimate for $\boldsymbol{\theta}$, we could calculate:

$$\frac{\text{count}_{\boldsymbol{x}}(\text{Presidents})}{N} \cdot \frac{\text{count}_{\boldsymbol{x}}(\text{tell})}{N} \cdots \frac{\text{count}_{\boldsymbol{x}}(\bigcirc)}{N}$$

Unigram Models: Assessment

Pros:

- Easy to understand
- Cheap
- Good enough for information retrieval (maybe)

Cons:

- Fixed, known vocabulary assumption
- "Bag of words" assumption is linguistically inaccurate
  - $p(\text{the the the the}) \gg p(\text{I want ice cream})$

n-gram Models

$$p(\boldsymbol{X} = \boldsymbol{x}) = \prod_{i=1}^{N} p(X_i = x_i \mid \boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1})$$

$$\overset{\text{assumption}}{=} \prod_{i=1}^{N} p(X_i = x_i \mid X_{i-\mathsf{n}+1:i-1} = \boldsymbol{x}_{i-\mathsf{n}+1:i-1}; \boldsymbol{\theta})$$

$$= \prod_{i=1}^{N} \theta_{x_i \mid \boldsymbol{x}_{i-\mathsf{n}+1:i-1}}$$

$(\mathsf{n}-1)$th-order Markov assumption $\equiv$ n-gram model

- Unigram model is the $\mathsf{n} = 1$ case
- For a long time, trigram models ($\mathsf{n} = 3$) were widely used
- 5-gram models ($\mathsf{n} = 5$) were common in MT for a time

n-gram Models

- Maximum likelihood estimate for the n-gram model's probability of v given a $(n-1)$-length history $\boldsymbol{h}$

$$
\begin{aligned}
\theta_{v|\boldsymbol{h}} &= p(X_i = v \mid \boldsymbol{X}_{i-\mathsf{n}+1:i-1} = \boldsymbol{h}) \\
&= \frac{p(X_i = v, \boldsymbol{X}_{i-\mathsf{n}+1:i-1} = \boldsymbol{h})}{p(\boldsymbol{X}_{i-\mathsf{n}+1:i-1} = \boldsymbol{h})} \\
&= \frac{\mathrm{count}_{\boldsymbol{x}}(\boldsymbol{h}v)}{N} \bigg/ \frac{\mathrm{count}_{\boldsymbol{x}}(\boldsymbol{h})}{N} \\
&= \frac{\mathrm{count}_{\boldsymbol{x}}(\boldsymbol{h}v)}{\mathrm{count}_{\boldsymbol{x}}(\boldsymbol{h})}
\end{aligned}
$$

Choosing n is a Balancing Act

If n is too small, your model can't learn very much about language.

As n gets larger:

- The number of parameters grows with $O(V^n)$.
- Most n-grams will never be observed, so you'll have lots of zero probability n-grams. This is an example of **data sparsity**.
- Your model depends increasingly on the training data; you need (lots) more data to learn to generalize well.

This is a beautiful illustration of the bias-variance tradeoff.

Other "tricks"

- Smoothing

  The game: prevent $\theta_{v|\boldsymbol{h}} = 0$ for any v and $\boldsymbol{h}$, while keeping $\sum_{\boldsymbol{x}} p(\boldsymbol{x}) = 1$ so that perplexity stays meaningful.

  - Simple method: add $\lambda > 0$ to every count (including counts of zero) before normalizing (the textbook calls this "Lidstone" smoothing)

- Dealing with Out-of-Vocabulary Terms
  - Define a special OOV or "unknown" symbol **unk**. Transform some (or all) rare words in the training data to **unk**.
  - Build a language model at the character level.
  - Some new methods use data-driven, deterministic tokenization schemes that segment some words into smaller parts to reduce the effective vocabulary size (Sennrich et al., 2016; Wu et al., 2016).

n-gram Models: Assessment

Pros:

- Easy to understand
- Cheap (with modern hardware; Lin and Dyer, 2010)
- Fine in some applications and when training data is scarce

Cons:

- Fixed, known vocabulary assumption
- Markov assumption is linguistically inaccurate
  - (But not as bad as unigram models!)
- Data sparseness problem

Summary so far

- language models utilities
  - Generation, evaluation of fluency, few-shot prediction (GPT3), …
- N-gram language models
  - Unigram LM
  - N-gram LM
- Neural language models:
  - Embedding: one-hot vectors -> embedding vectors
  - Neural networks

Neural Language Models

Instead of a lookup for a word and fixed-length history ($\theta_{v|\boldsymbol{h}}$), define a vector function:

$$p(X_i \mid \boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1}) = \mathbf{NN}(\mathbf{enc}(\boldsymbol{x}_{1:i-1}); \boldsymbol{\theta})$$

where $\boldsymbol{\theta}$ do the work of *encoding* the history and *transforming* it into a distribution over the next word.
The transformation is described as a composed series of simple transformations or "layers."

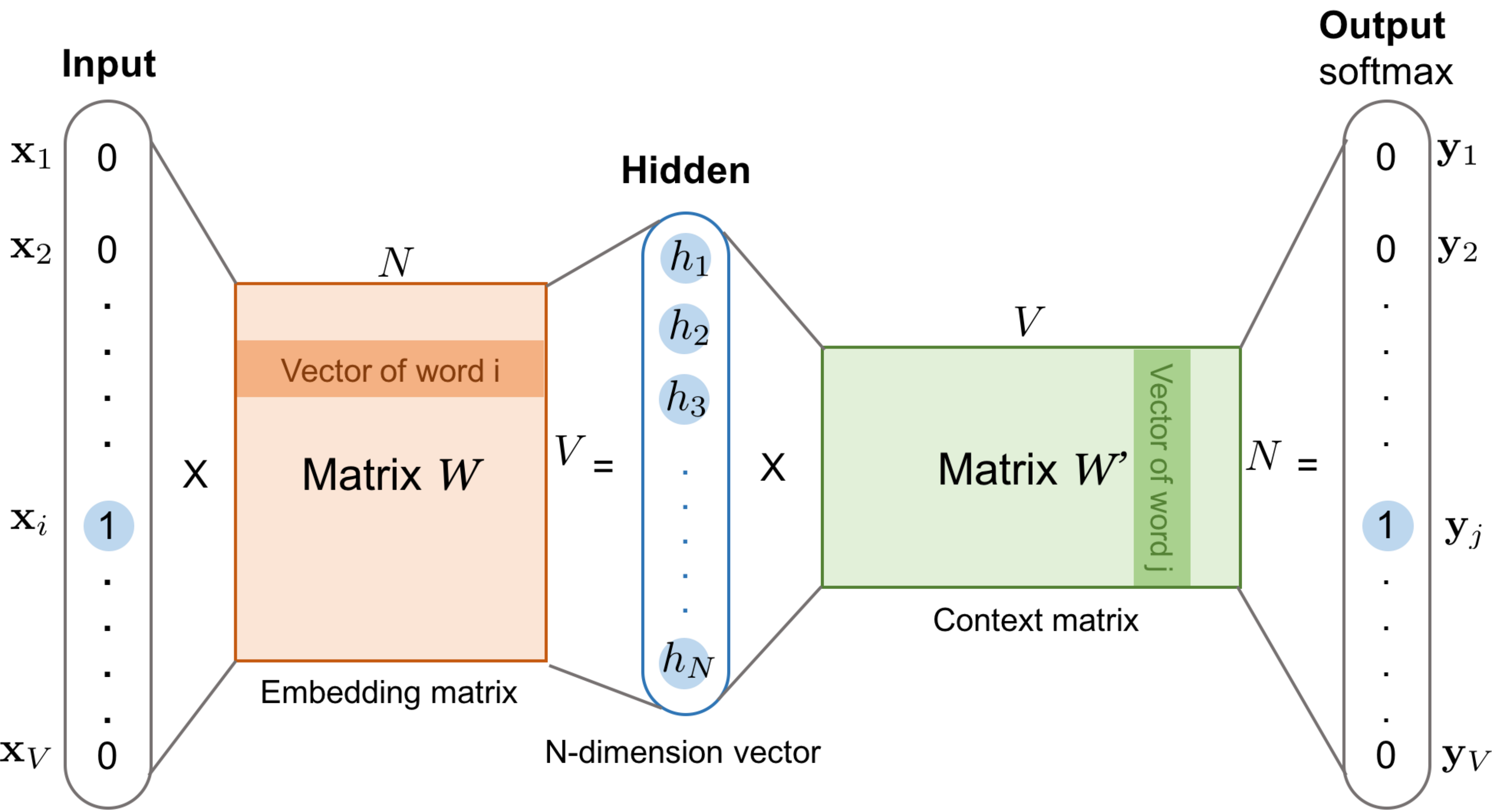
Neural Language Models

Instead of a lookup for a word and fixed-length history ($\theta_{v|\boldsymbol{h}}$), define a vector function:

$$p(X_i \mid \boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1}) = \mathbf{NN}(\mathbf{enc}(\boldsymbol{x}_{1:i-1}); \boldsymbol{\theta})$$

where $\boldsymbol{\theta}$ do the work of *encoding* the history and *transforming* it into a distribution over the next word.
The transformation is described as a composed series of simple transformations or "layers."

- We first map word histories $\boldsymbol{h}$ to vectors/matrices
- We interpret the output as $p(X_i \mid \boldsymbol{X}_{1:i-1} = \boldsymbol{h})$


Input
Output
softmax
Hidden
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_i$
$\mathbf{x}_V$
0
0
1
0
X
N
Vector of word i
Matrix W
Embedding matrix
V =
h_1
h_2
h_3
h_N
N-dimension vector
X
V
Matrix W'
Vector of word j
Context matrix
N =
0
0
1
0
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_j$
$\mathbf{y}_V$

Two Key Components

- "Embedding" words as vectors
- Layering to increase capacity (i.e., the set of distributions that can be represented).

"One Hot" Vectors

Let $\mathbf{e}_i \in \mathbb{R}^V$ be the ith column of the identity matrix $\mathbf{I}$.

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}; \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}; \quad \ldots; \quad \mathbf{e}_V = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$\mathbf{e}_i$ is the "one hot" vector for the ith word in $\mathcal{V}$.

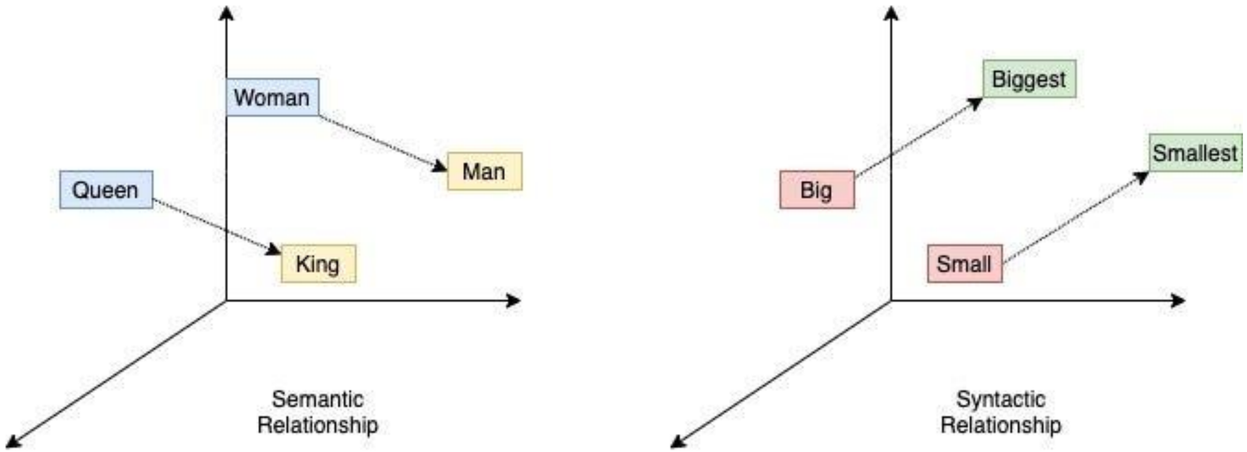
A neural language model starts by "looking up" each word by multiplying its one hot vector by a matrix $\underset{V \times d}{\mathbf{M}}$; $\mathbf{e}_v^\top \mathbf{M} = \mathbf{m}_v$, the "embedding" of v.

$\mathbf{M}$ becomes part of the parameters ($\boldsymbol{\theta}$).

Sequences of Word Vectors

Given a word sequence $\langle v_1, v_2, \ldots, v_k \rangle$, we transform it into a sequence of word vectors,

$$\mathbf{m}_{v_1}, \mathbf{m}_{v_2}, \ldots, \mathbf{m}_{v_k}$$



Adding Layers

- Neural networks are built by composing functions, a mix of
  - Affine, $\boldsymbol{v}' = \boldsymbol{W}\boldsymbol{v} + \boldsymbol{b}$ (note that the dimensionality of $\boldsymbol{v}$ and $\boldsymbol{v}'$ might be different)
  - Nonlinearity, e.g.,
    - rectified linear ("relu") units $v_i' = \max(0, v_i)$
    - elementwise hyperbolic tangent $v_i' = \tanh(v_i) = \frac{e^{v_i} - e^{-v_i}}{e^{v_i} + e^{-v_i}}$
    - softmax $v_i' = \exp\{ v_i \} / \sum_j \exp\{ v_j \}$
  - More complex components (composed of the above operations):
    - Convolutional layers
    - Recurrent NNs
    - Attention

Summary so far

- language models utilities
  - Generation, evaluation of fluency, few-shot prediction (GPT3), …
- N-gram language models
  - Unigram LM
  - N-gram LM
- Neural language models:
  - Embedding: one-hot vectors -> embedding vectors
  - Neural networks

Outline

- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing gradients
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT

Outline

- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing gradients
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT

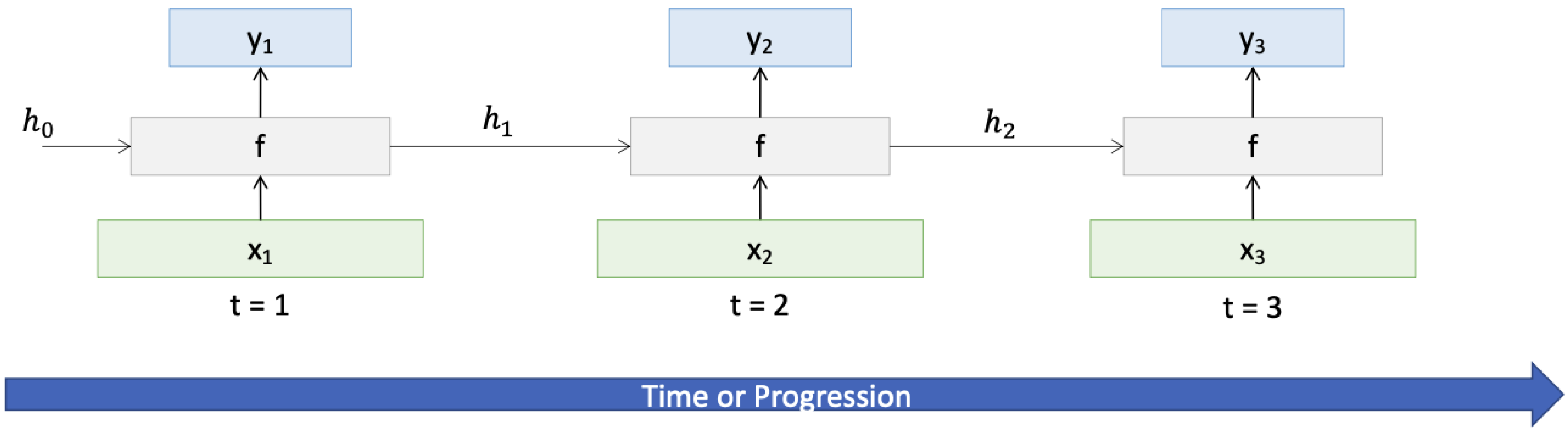
Language Mining – RNN

The teacher told the student that he was brilliant.

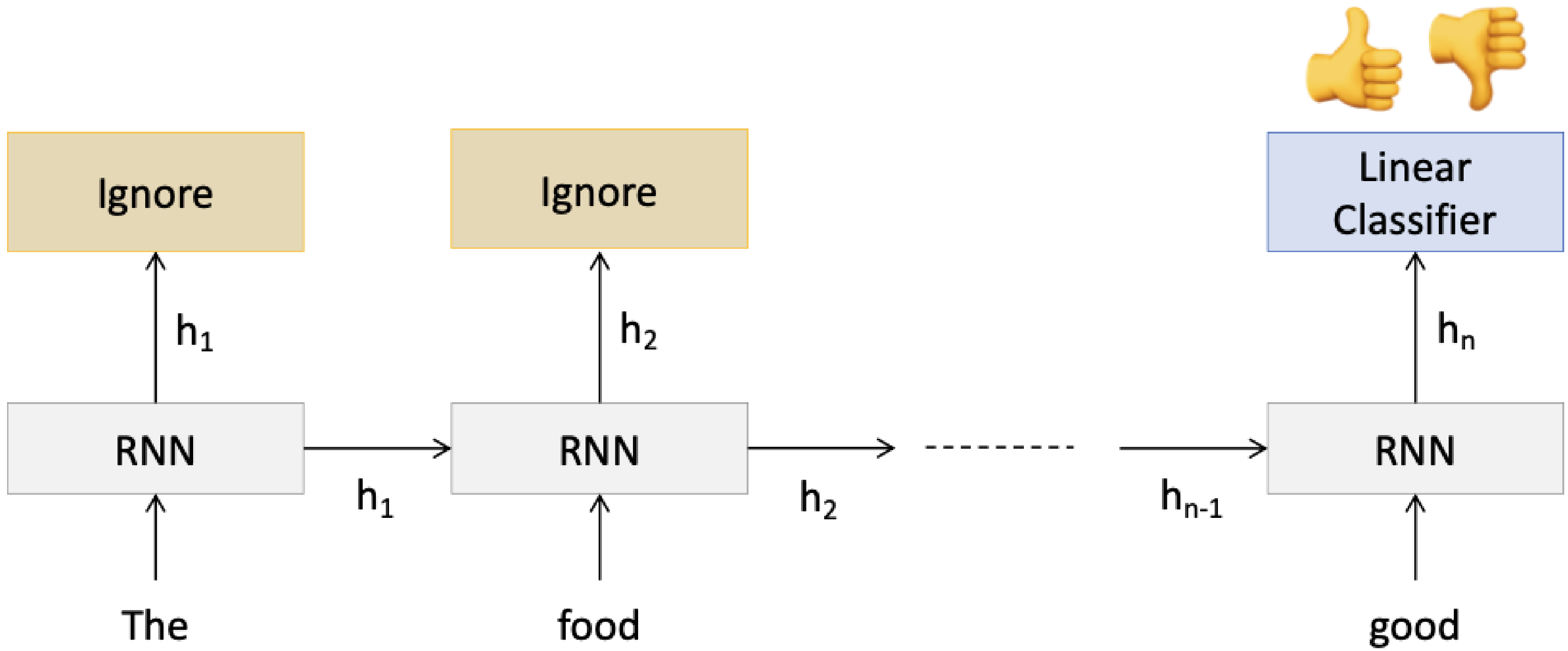
The student told the teacher that he was brilliant.

You read this sentence from left to right and understand the sentence

What if we have sequences of variable lengths, like sentences or videos that we would like to analyze?

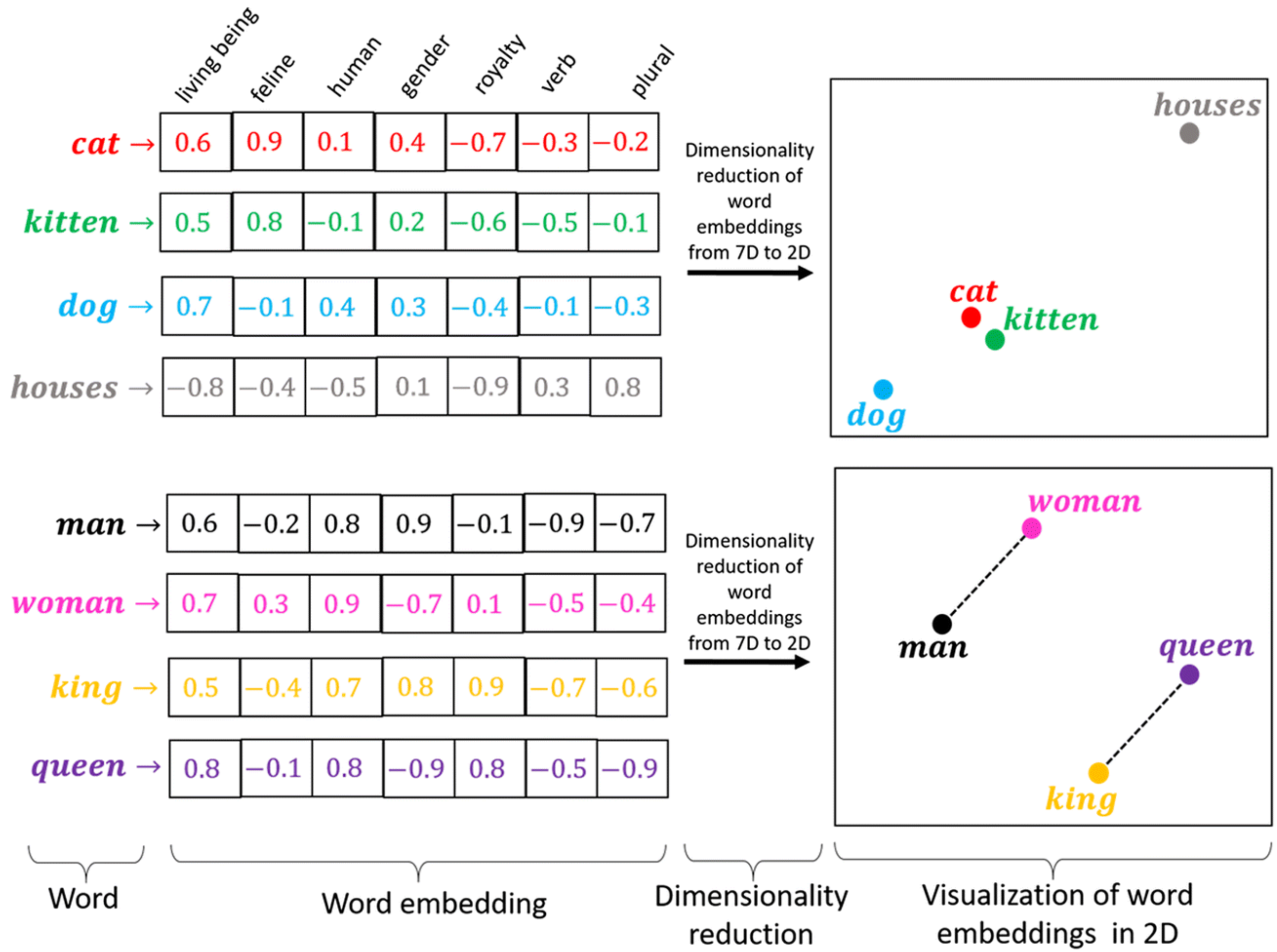


Language Mining – RNN

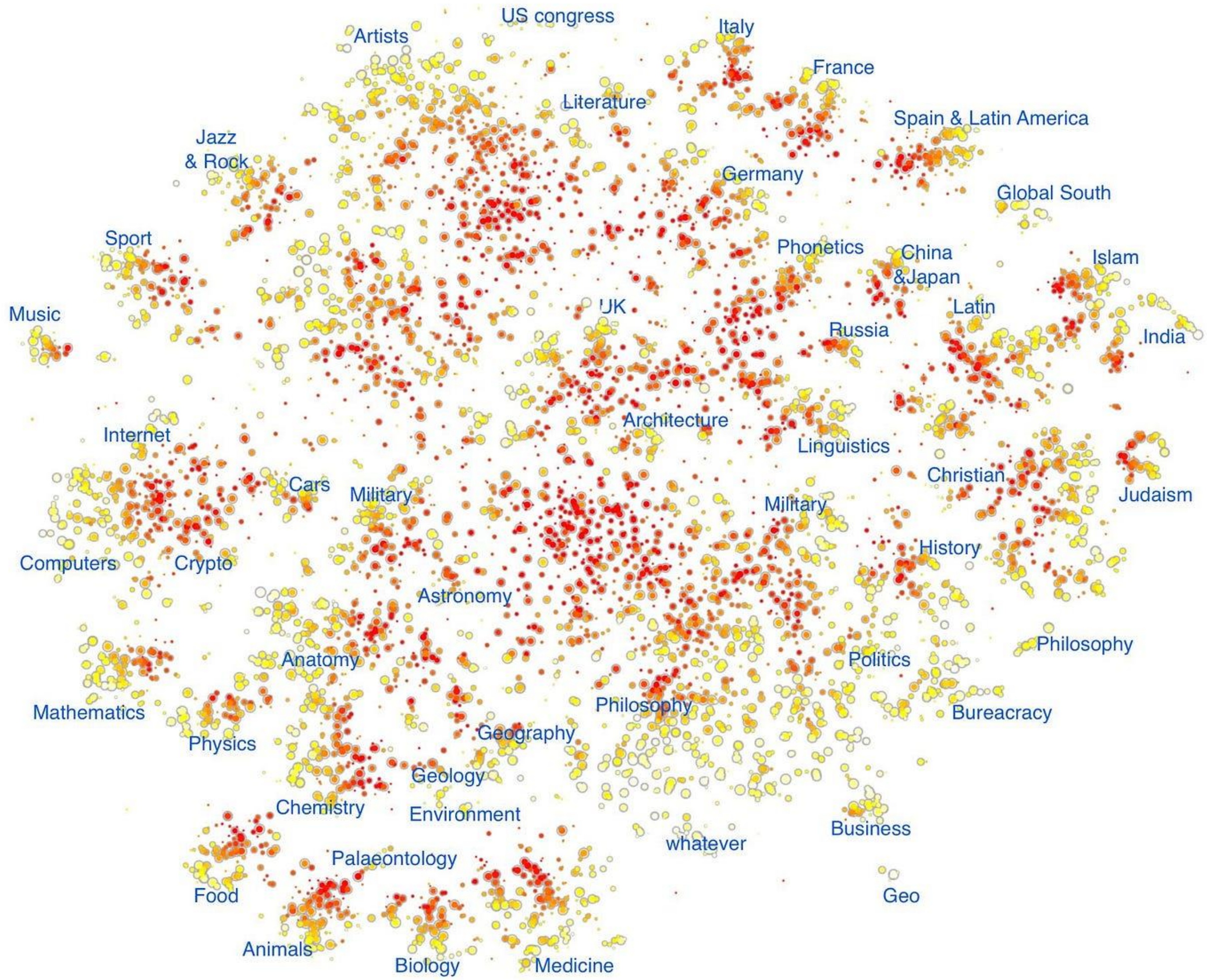


RNN for sentiment analysis/next token prediction

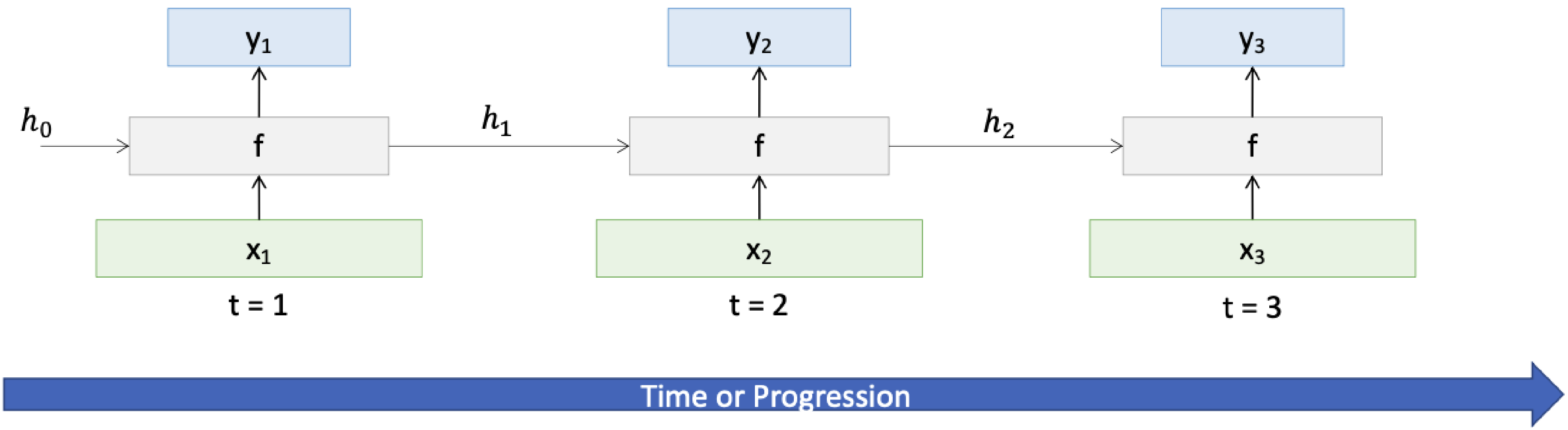
Language Mining – RNN



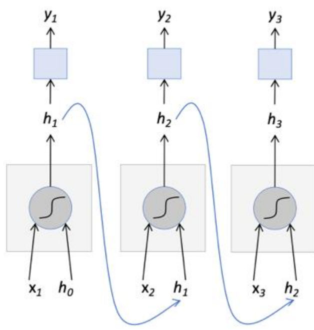
Language Mining – RNN



What if we have sequences of variable lengths, like sentences or videos that we would like to analyze?

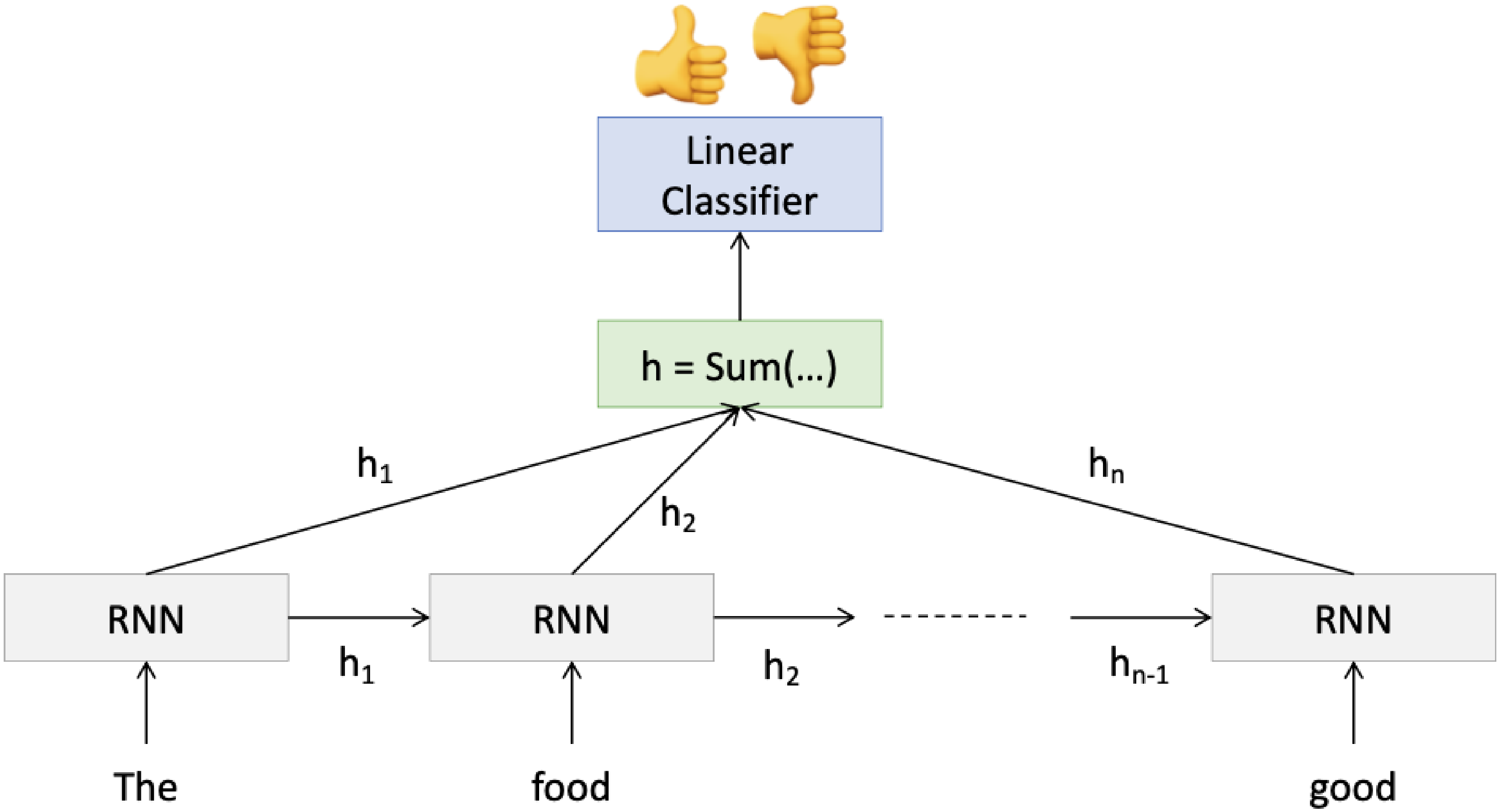


How would you setup the model (token embedding, MLP layer)



- $h_t = \sigma(W^T \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix})$
- $y_t = F(h_t)$

Language Mining – RNN



RNN for sentiment analysis

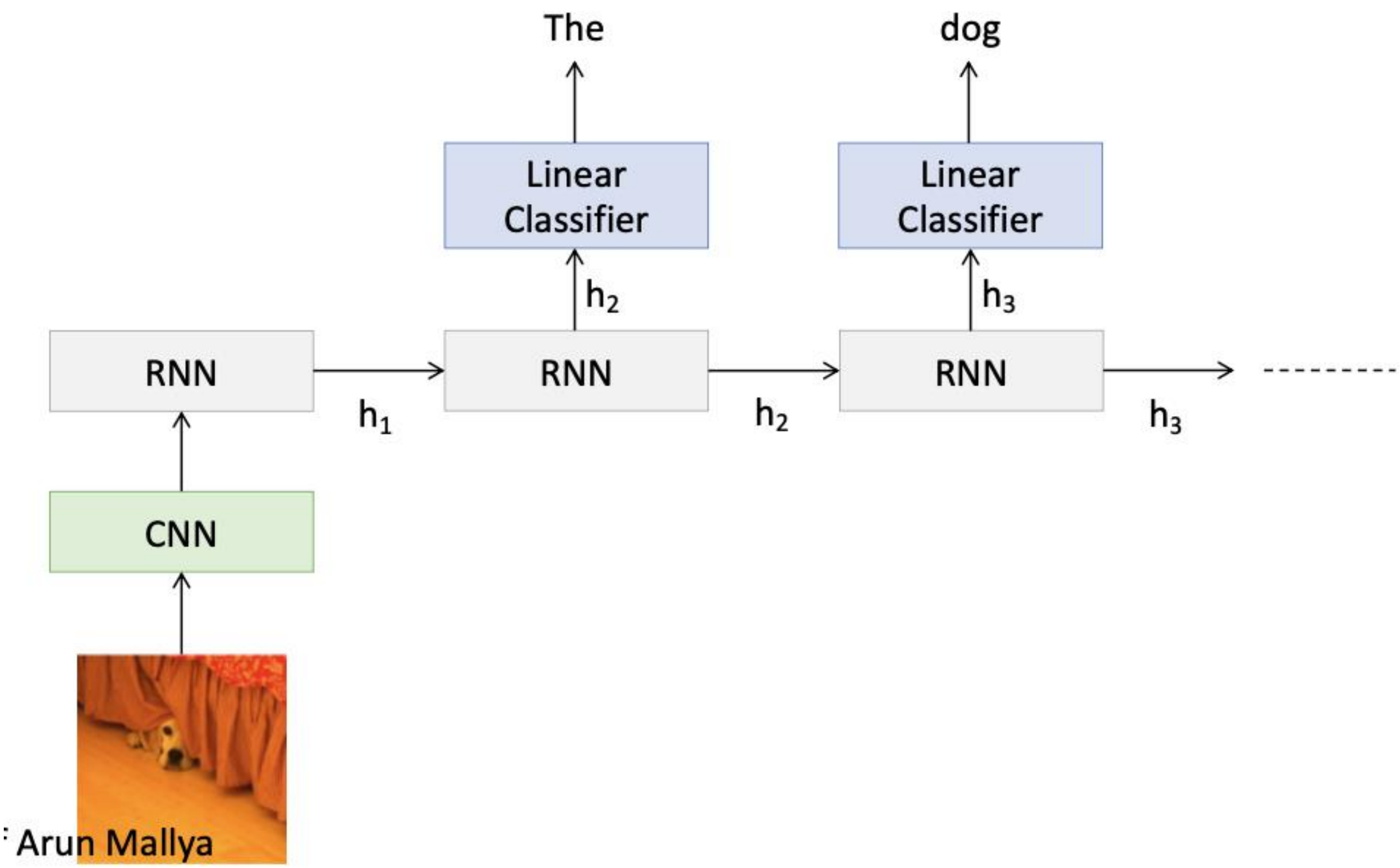
Image Captioning

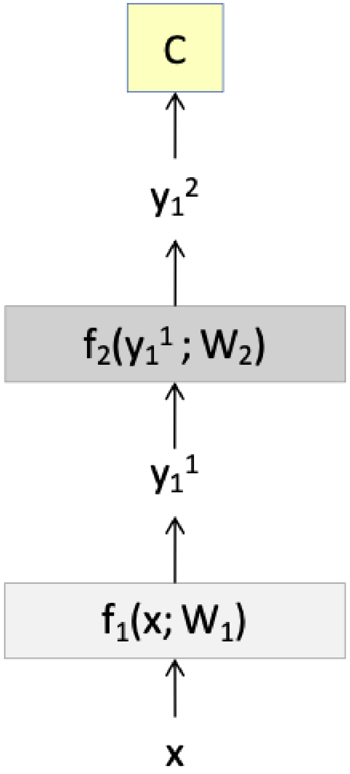
- Given an image, produce a sentence describing its contents

- Inputs: Image feature (from a CNN)
- Outputs: Multiple words (let's consider one sentence)


: The dog is hiding

Image Captioning





$$y_1 = f_1(x; W_1)$$

$$y_2 = f_2(y_1; W_2)$$

$$C = \text{Loss}(y_2, y_{GT})$$

$$\text{Find } \frac{\partial C}{\partial W_1}, \frac{\partial C}{\partial W_2}$$

$$\frac{\partial C}{\partial W_2} = \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial W_2}\right)$$

$$\frac{\partial C}{\partial W_1} = \left(\frac{\partial C}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

$$= \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

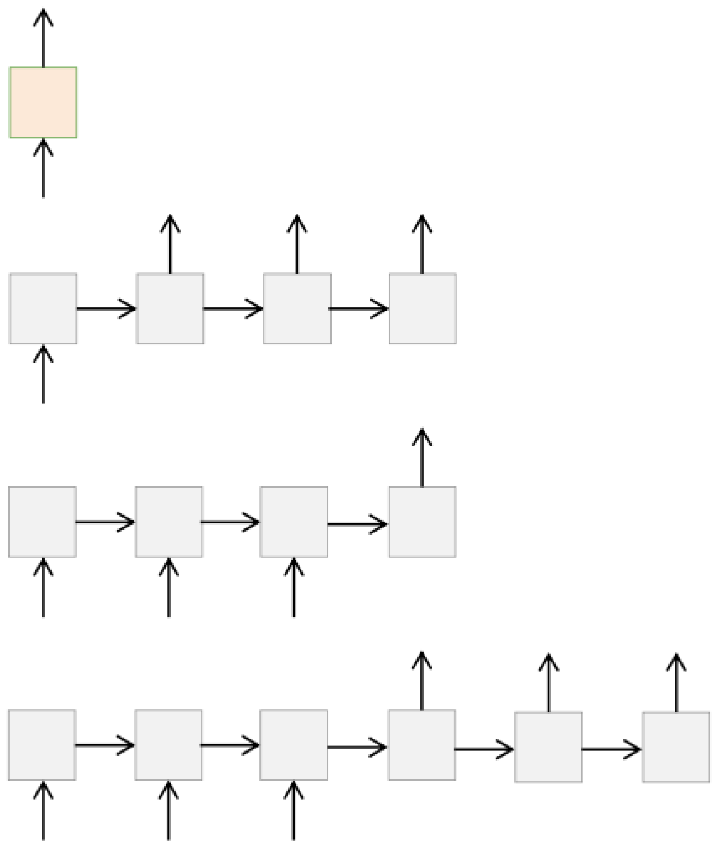
Language Mining – RNN

$$h_t = f(Wh_{t-1} + b)$$

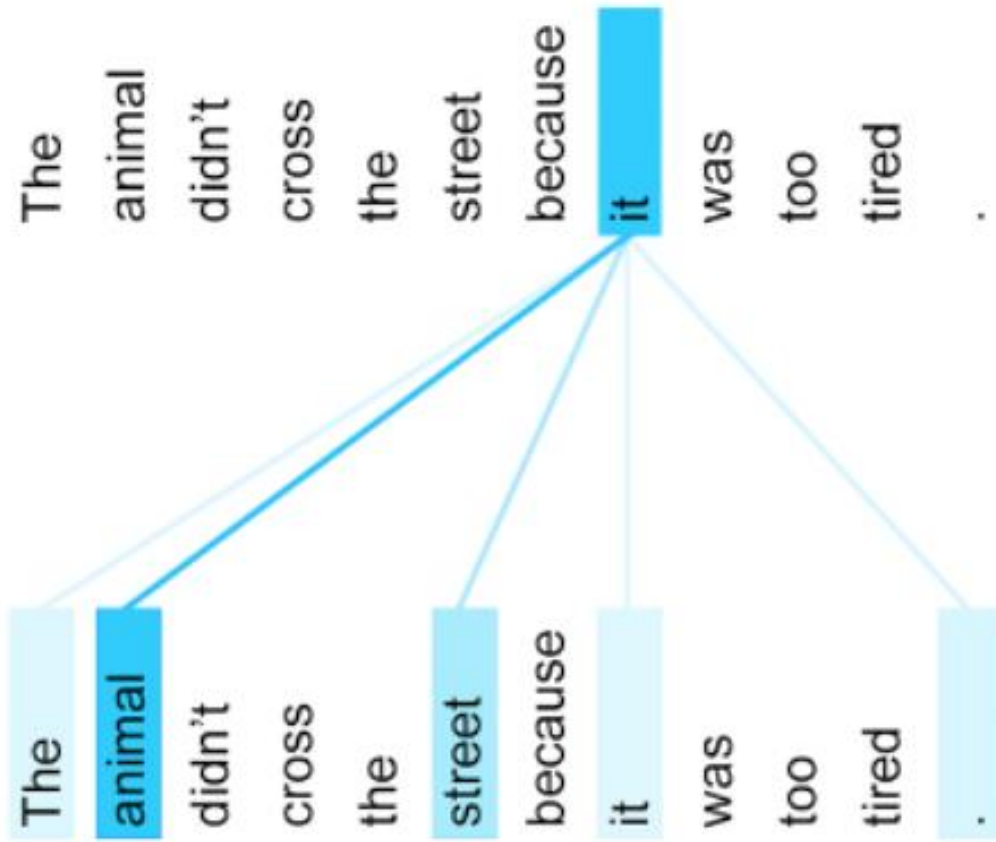
$$\|\frac{\partial L}{\partial h_0}\| \approx \sigma_{\max}^{T} \|\frac{\partial L}{\partial h_T}\|$$

- Largest singular value > 1 → **Exploding gradients**
  - Slight error in the late time steps causes drastic updates in the early time steps → Unstable learning
- Largest singular value < 1 → **Vanishing gradients**
  - Gradients passed to the early time steps is close to 0. → Uninformed correction

Input – Output Scenarios

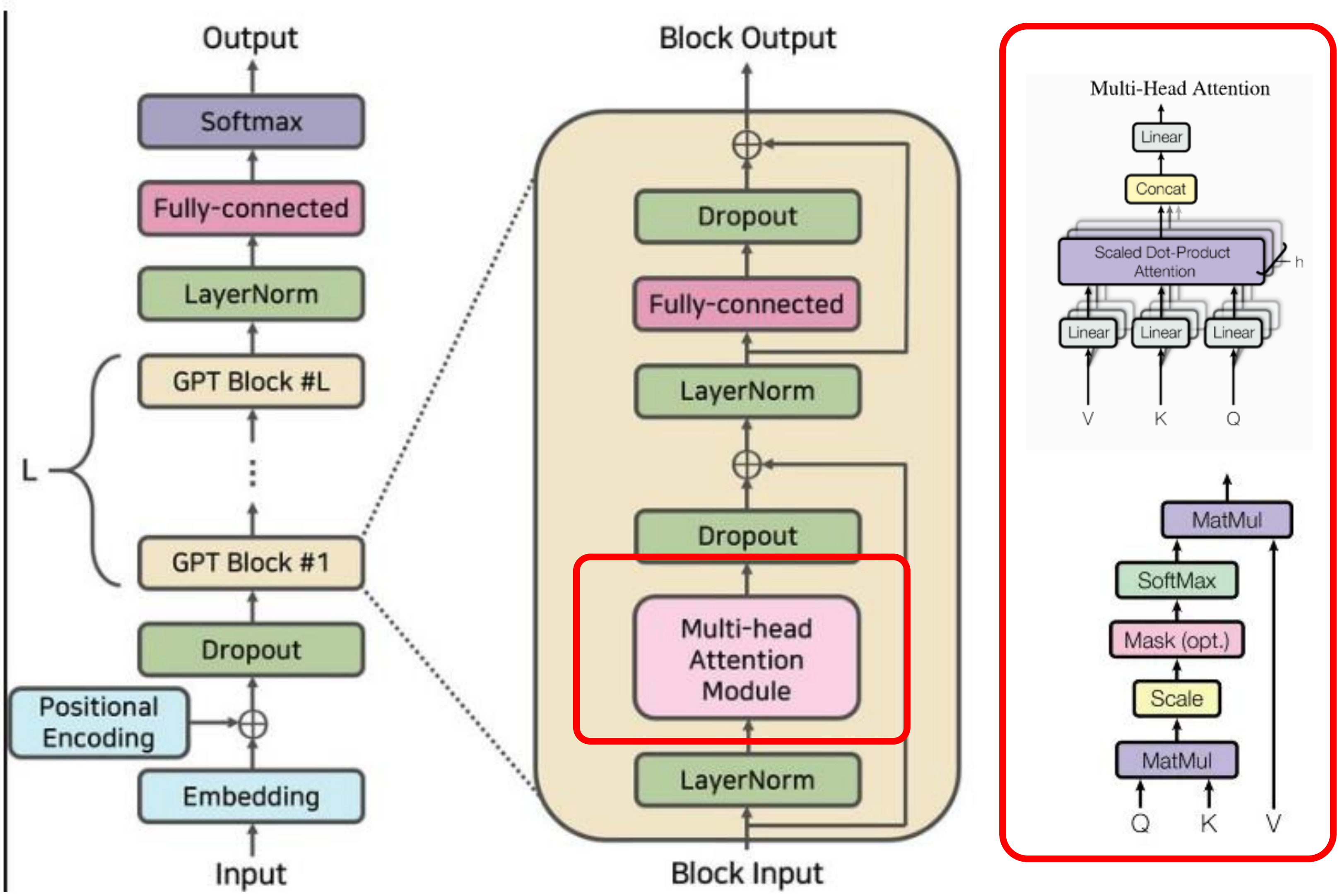


Scenario	Example
Single - Single	Feed-forward Network
Single - Multiple	Image Captioning
Multiple - Single	Sentiment Classification
Multiple - Multiple	Translation



Key Value Query

Language Mining – Transformer - Attention



Language Mining – Transformer - Attention

Think of YouTube.

First you enter your **Query** *in the search bar. Then your* **Query** *is compared against a set of* **Keys** *(in this case, video titles, tags and descriptions etc. within the YouTube database). After this, YouTube proceeds to retrieve the videos that best match your* **Query***. These video results are referred to as* **Values***.*

Key Value Query

I would like to taste the local food in France.

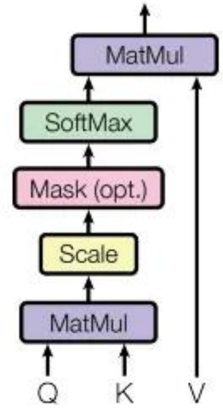
I would like to taste the local food in France.

$$Q = XW_Q \qquad K = XW_K \qquad V = XW_V$$

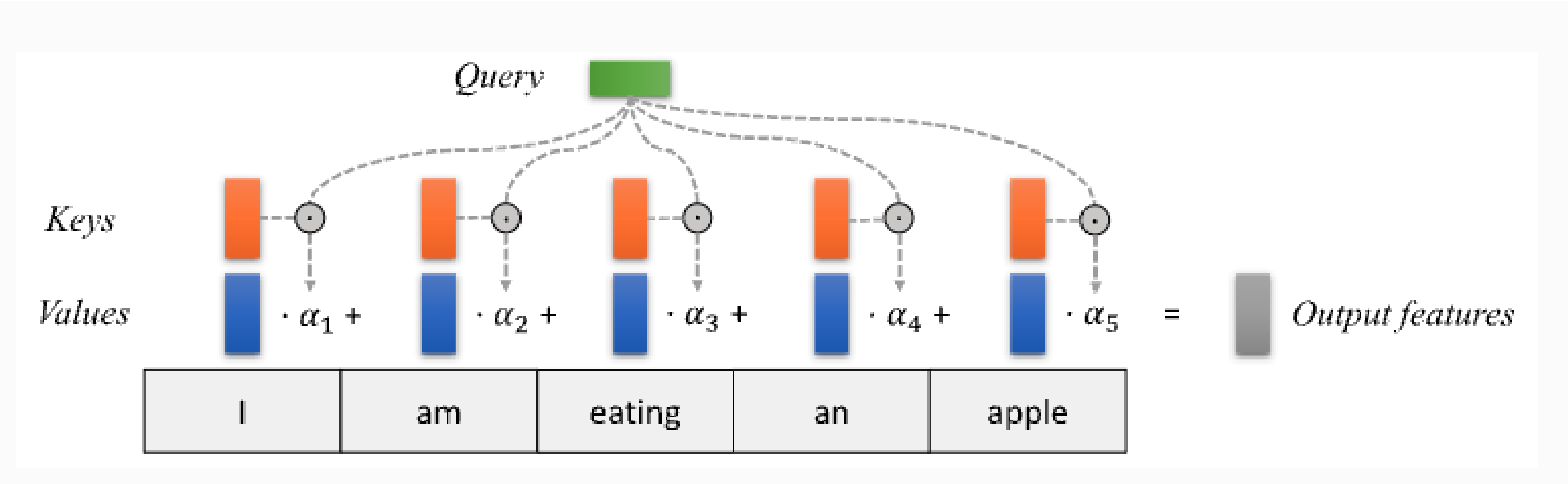
$Q \cdot K = QK^T =$

	I	would	like	to	taste	the	local	food	in	France
I	h									
would		h								
like			h							
to				h						
taste					h			h		
the						h				
local							h			h
food					h			h		
in									h	
France							h			h

$$softmax\left(\frac{Q_i K^T}{\sqrt{d_k}}\right) = \frac{e^{\left(\frac{Q_i K^T}{\sqrt{d_k}}\right)}}{\sum_{j=1}^{d_k} e^{\left(\frac{Q_j K^T}{\sqrt{d_k}}\right)}}$$



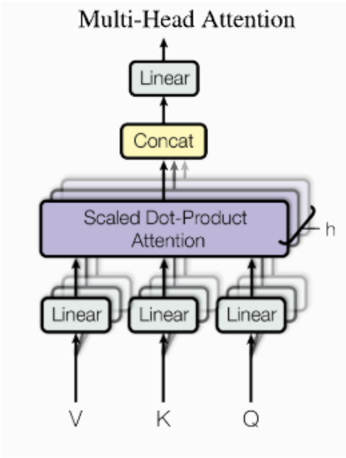
Language Mining – Transformer - Attention



$$\alpha_i = \frac{\exp\left(f_{attn}\left(\text{key}_i, \text{query}\right)\right)}{\sum_j \exp\left(f_{attn}\left(\text{key}_j, \text{query}\right)\right)}, \quad \text{out} = \sum_i \alpha_i \cdot \text{value}_i$$

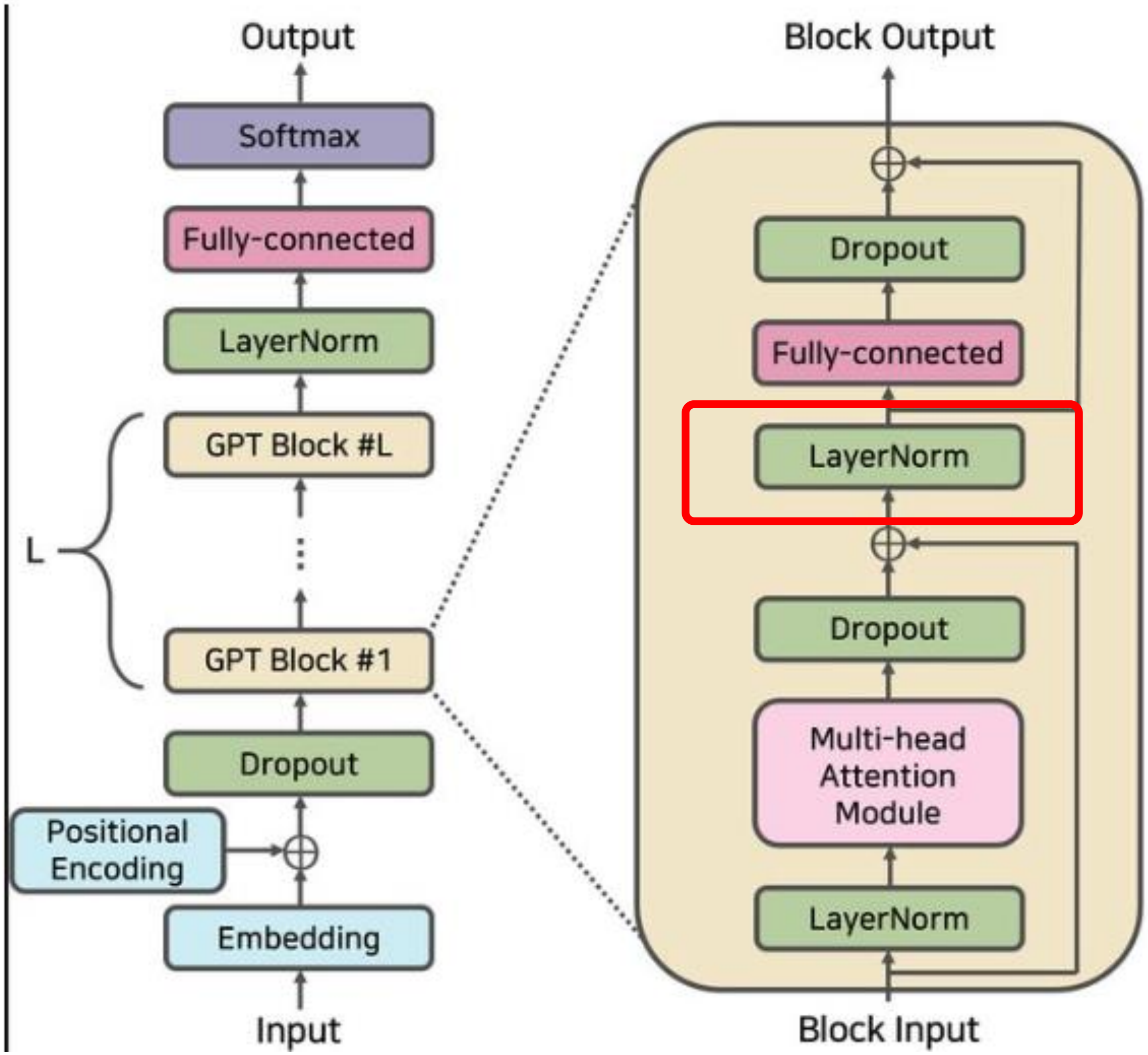
The scaled dot product attention allows a network to attend over a sequence. However, often there are multiple different aspects a sequence element wants to attend to, and a single weighted average is not a good option for it. This is why we extend the attention mechanisms to multiple heads

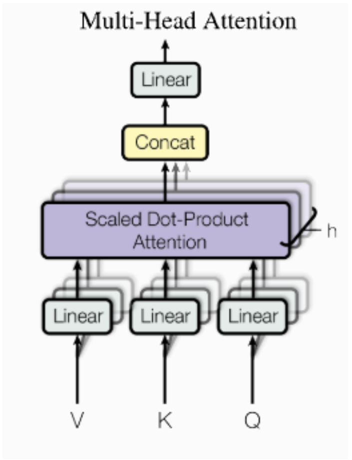
$$\begin{aligned}\text{Multihead}(Q,K,V) &= \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O\\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

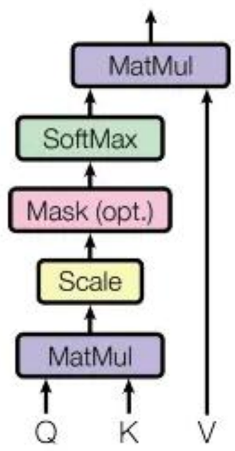


$W_{1\ldots h}^{Q} \in \mathbb{R}^{D\times d_k}$, $W_{1\ldots h}^{K} \in \mathbb{R}^{D\times d_k}$, $W_{1\ldots h}^{V} \in \mathbb{R}^{D\times d_v}$, and $W^{O} \in \mathbb{R}^{h\cdot d_v\times d_{out}}$

Language Mining – Transformer – LayerNorm

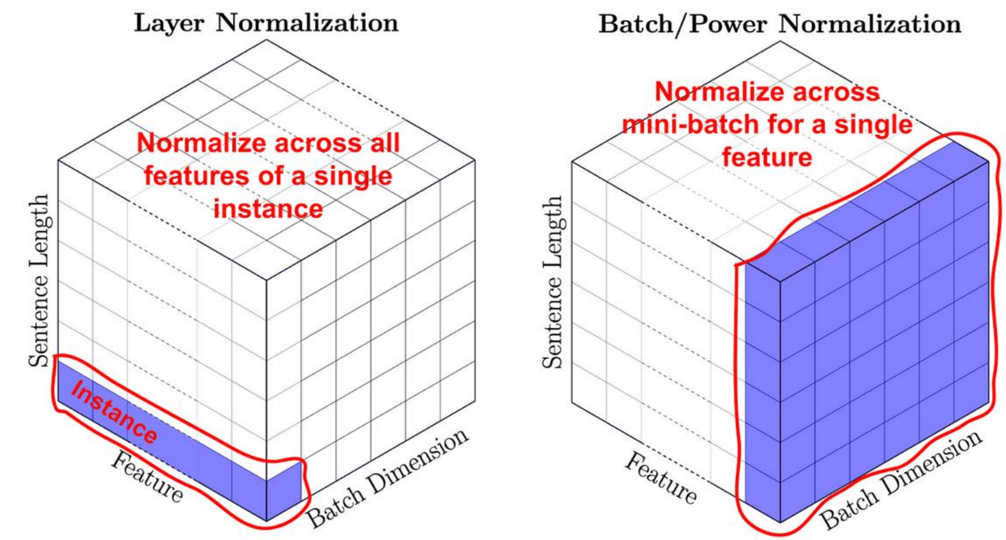






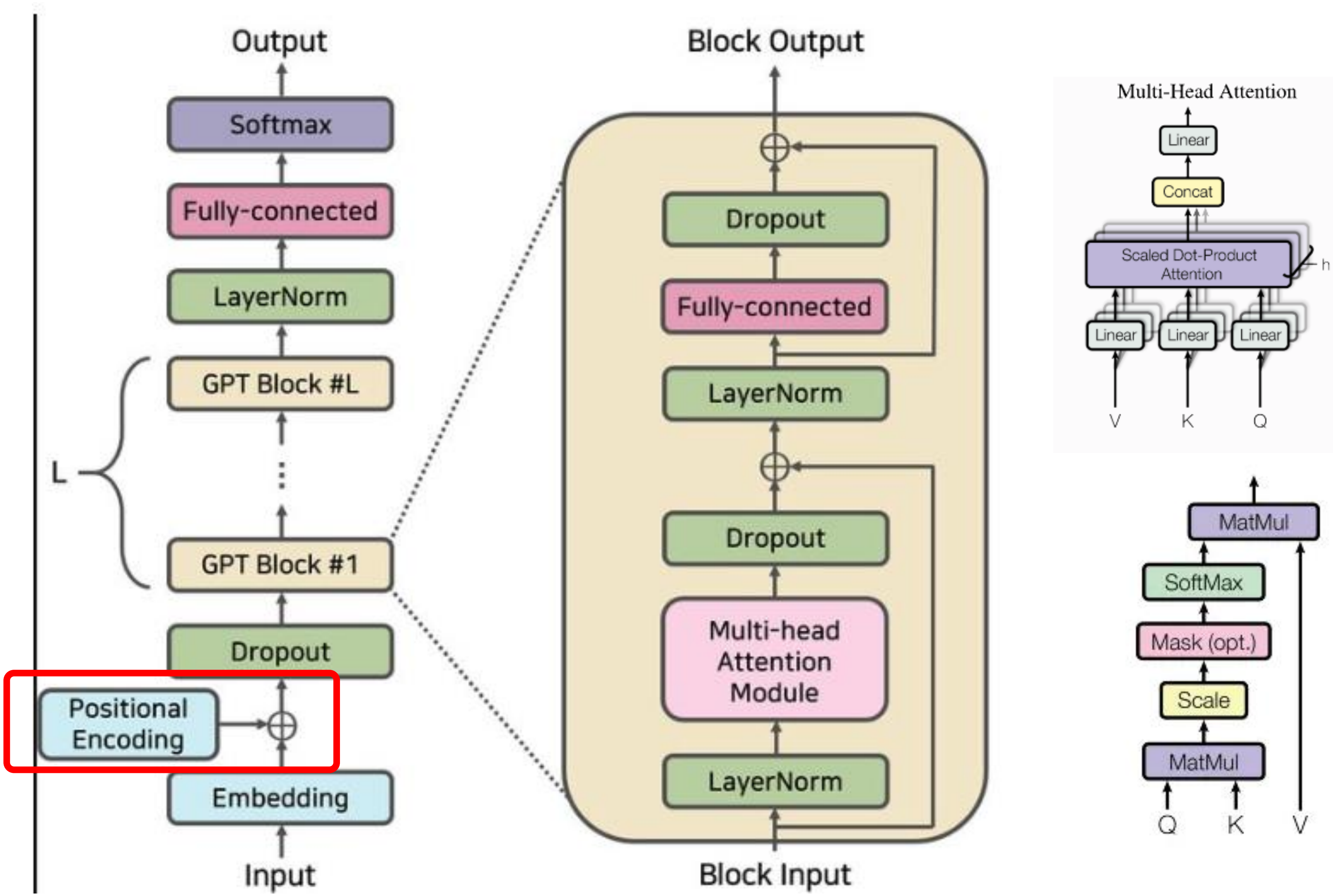
Language Mining – Transformer – LayerNorm

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$
$$x = \mathrm{LayerNorm}(x + \mathrm{FFN}(x))$$



$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

Language Mining – Transformer – Positional Encoding



The teacher told the student that he was brilliant.

The student told the teacher that he was brilliant.

Two sentences using exactly the same tokens but end up with very different meaning

Language Mining – Transformer – Positional Encoding

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

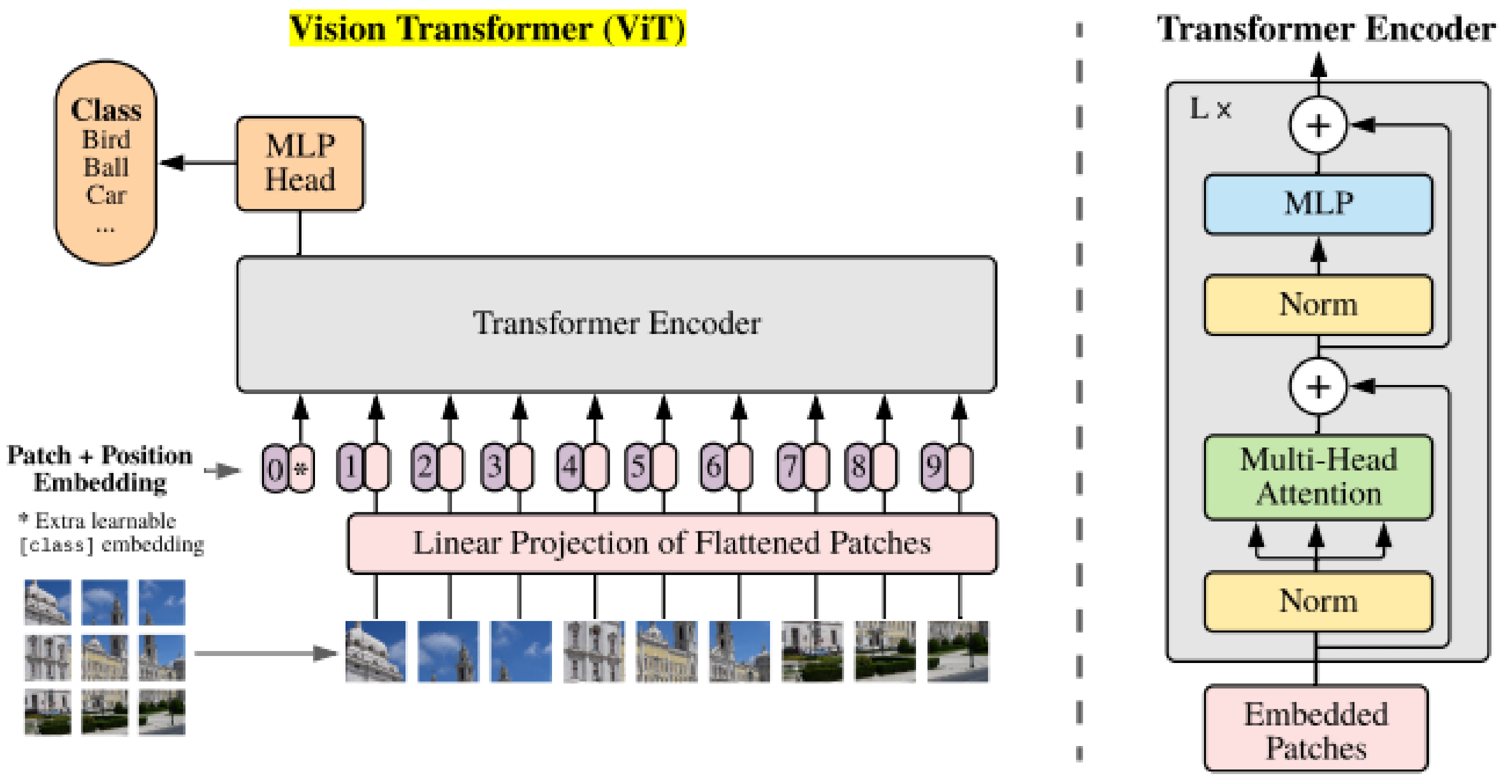
k: Position of an object in the input sequence,

d: Dimension of the output embedding space

Positional Encoding Matrix with d=4, n=100

Sequence	Index of token, k	i=0	i=0	i=1	i=1
I	0	$P_{00}=\sin(0)$ = 0	$P_{01}=\cos(0)$ = 1	$P_{02}=\sin(0)$ = 0	$P_{03}=\cos(0)$ = 1
am	1	$P_{10}=\sin(1/1)$ = 0.84	$P_{11}=\cos(1/1)$ = 0.54	$P_{12}=\sin(1/10)$ = 0.10	$P_{13}=\cos(1/10)$ = 1.0
a	2	$P_{20}=\sin(2/1)$ = 0.91	$P_{21}=\cos(2/1)$ = -0.42	$P_{22}=\sin(2/10)$ = 0.20	$P_{23}=\cos(2/10)$ = 0.98
Robot	3	$P_{30}=\sin(3/1)$ = 0.14	$P_{31}=\cos(3/1)$ = -0.99	$P_{32}=\sin(3/10)$ = 0.30	$P_{33}=\cos(3/10)$ = 0.96

Language Mining – Vision Transformer

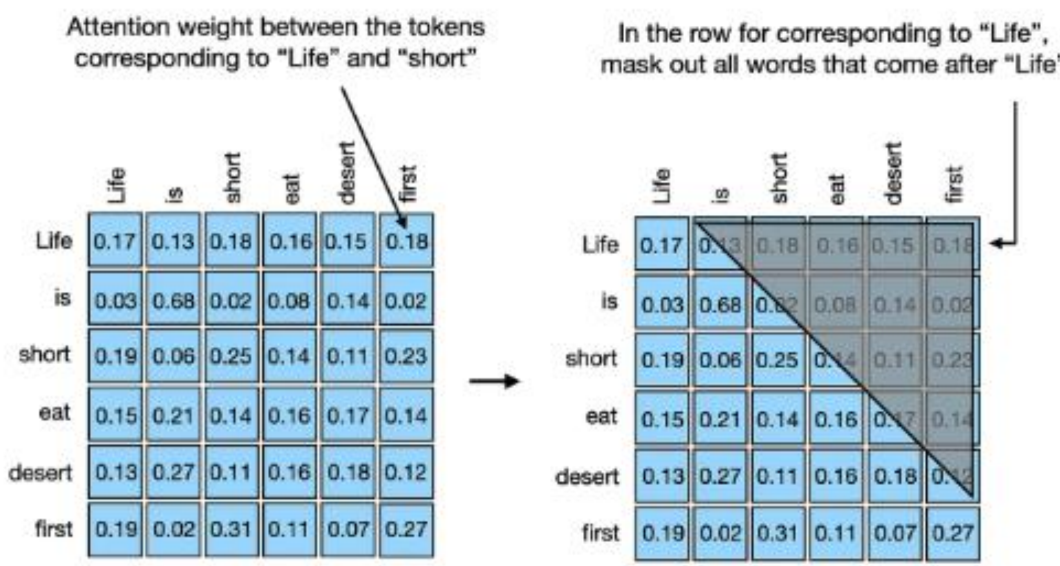


I would like to **taste** the **local food** in **France**.

$$Q = XW_Q \qquad K = XW_K \qquad V = XW_V$$

Self-Attention

Causal-Attention



	Life	is	short	eat	desert	first
Life	0.17	0.13	0.18	0.16	0.15	0.18
is	0.03	0.68	0.02	0.08	0.14	0.02
short	0.19	0.06	0.25	0.14	0.11	0.23
eat	0.15	0.21	0.14	0.16	0.17	0.14
desert	0.13	0.27	0.11	0.16	0.18	0.12
first	0.19	0.02	0.31	0.11	0.07	0.27

	Life	is	short	eat	desert	first
Life	0.17	0.13	0.18	0.16	0.15	0.18
is	0.03	0.68	0.02	0.08	0.14	0.02
short	0.19	0.06	0.25	0.14	0.11	0.23
eat	0.15	0.21	0.14	0.16	0.17	0.14
desert	0.13	0.27	0.11	0.16	0.18	0.12
first	0.19	0.02	0.31	0.11	0.07	0.27

Language Mining – Transformer – Different Attention

Cross-attention

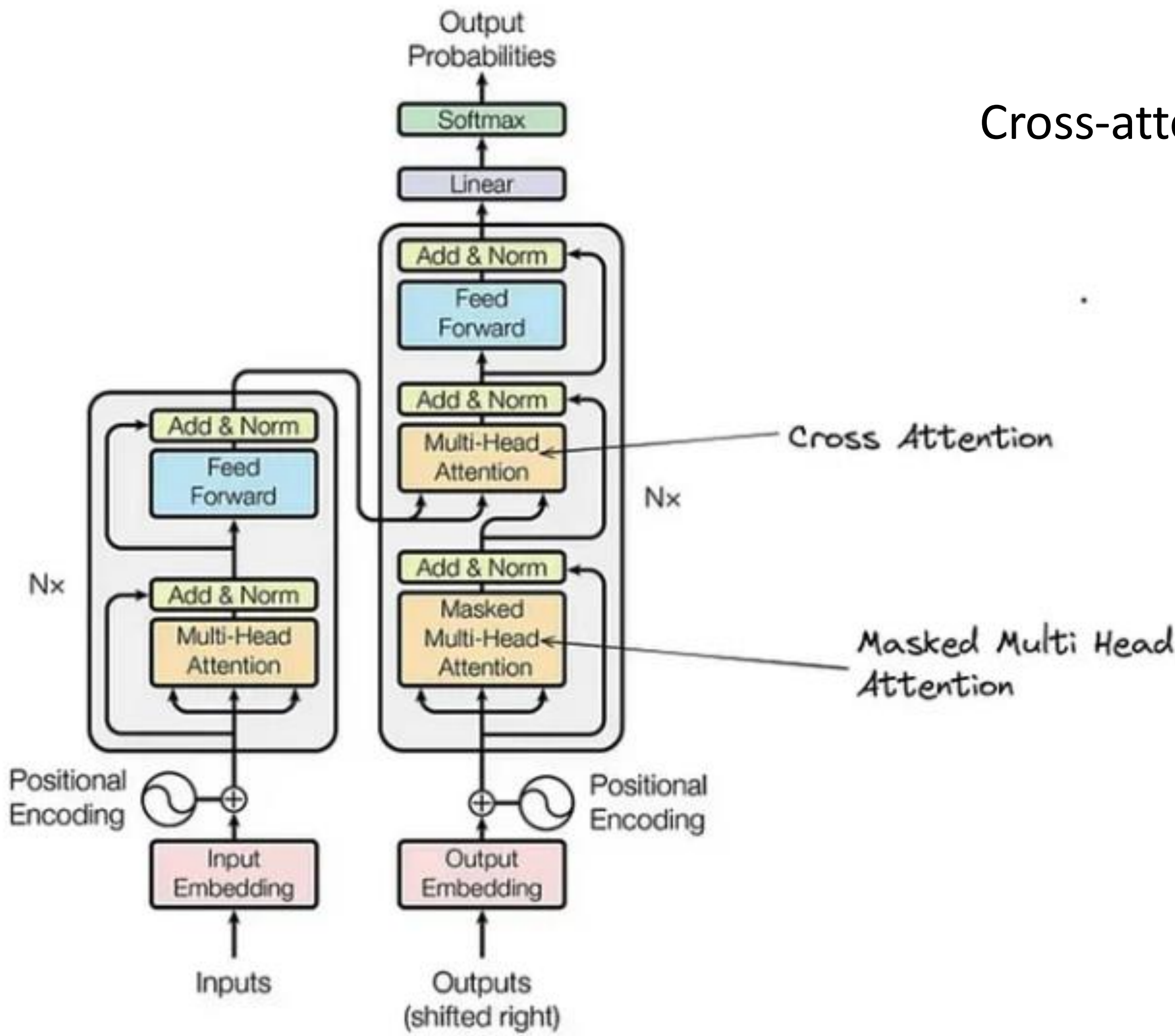


Figure 1: The Transformer - model architecture.

Language Mining – Transformer – Different Attention

Multi-Attention

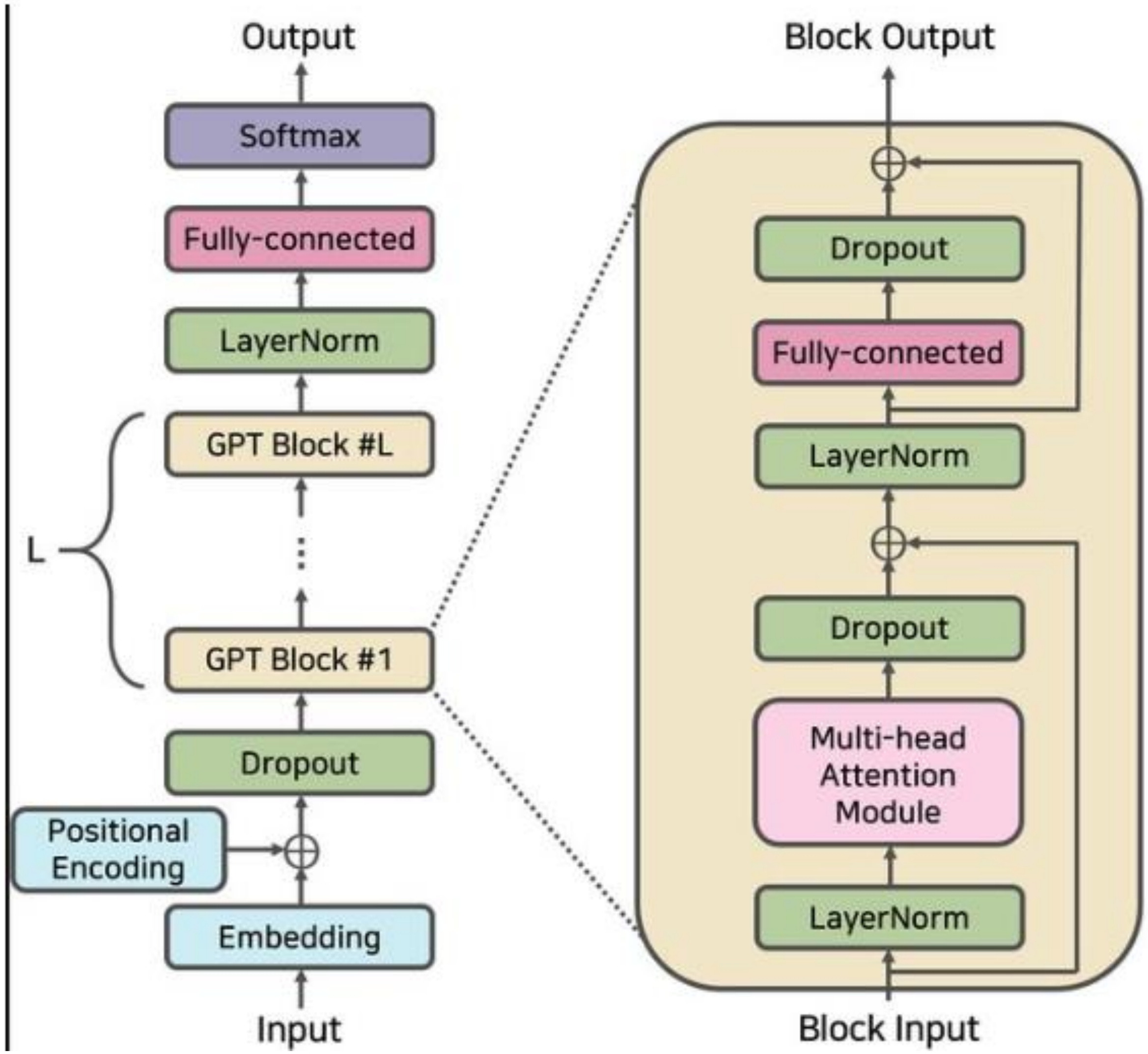
Self-Attention

Causal-Attention

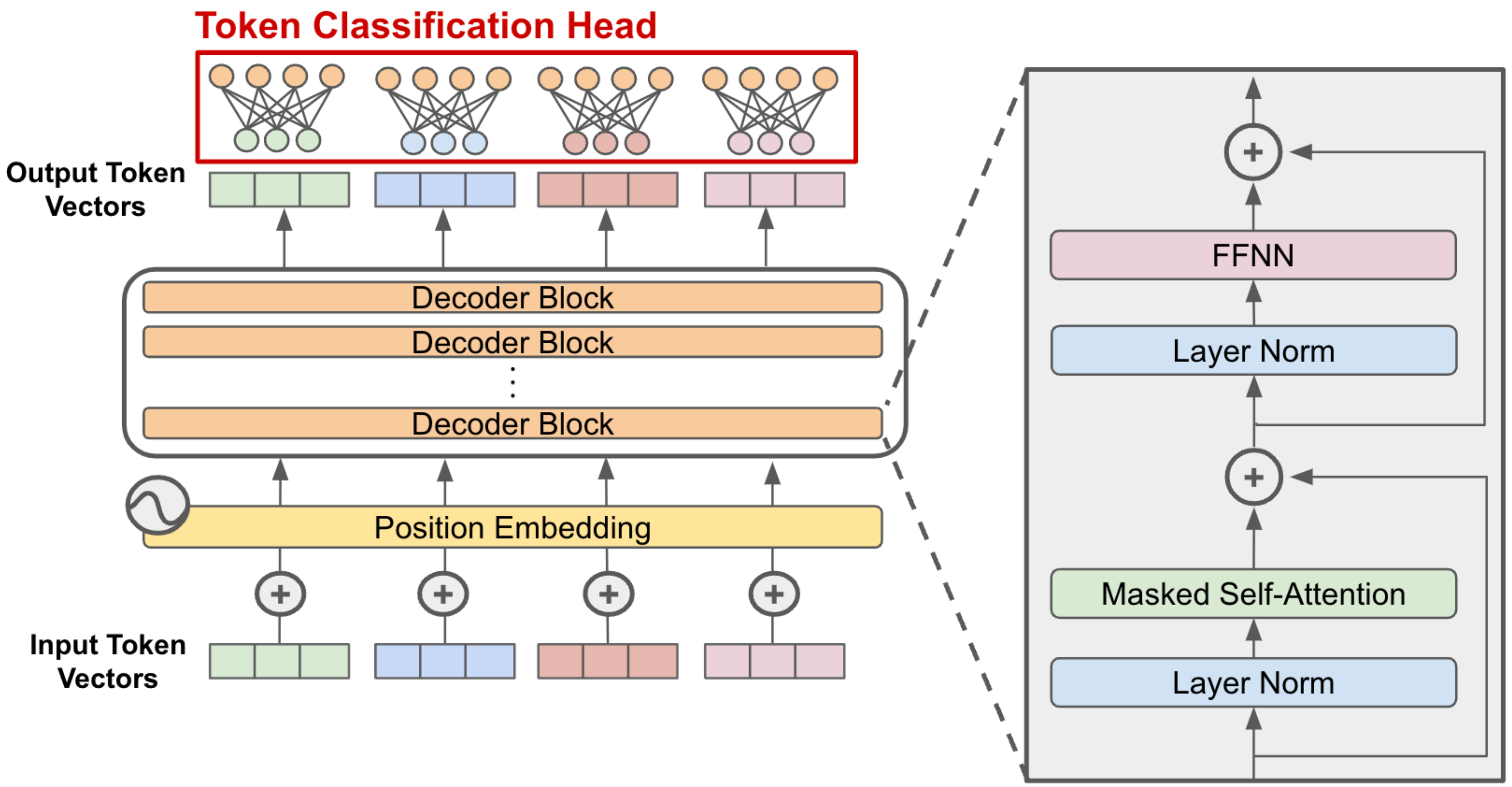
Cross-Attention

Sentiment Analysis	Next-token Prediction	Translation

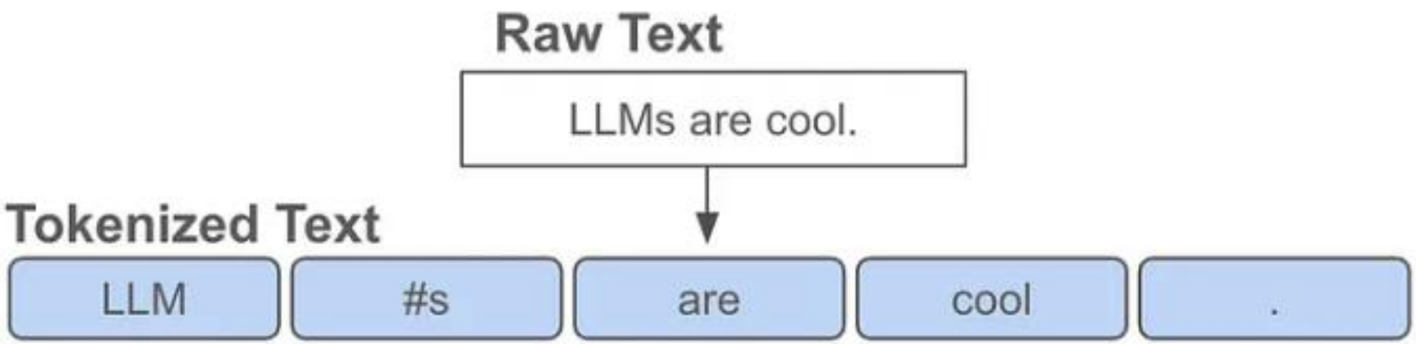
Language Mining – LLM

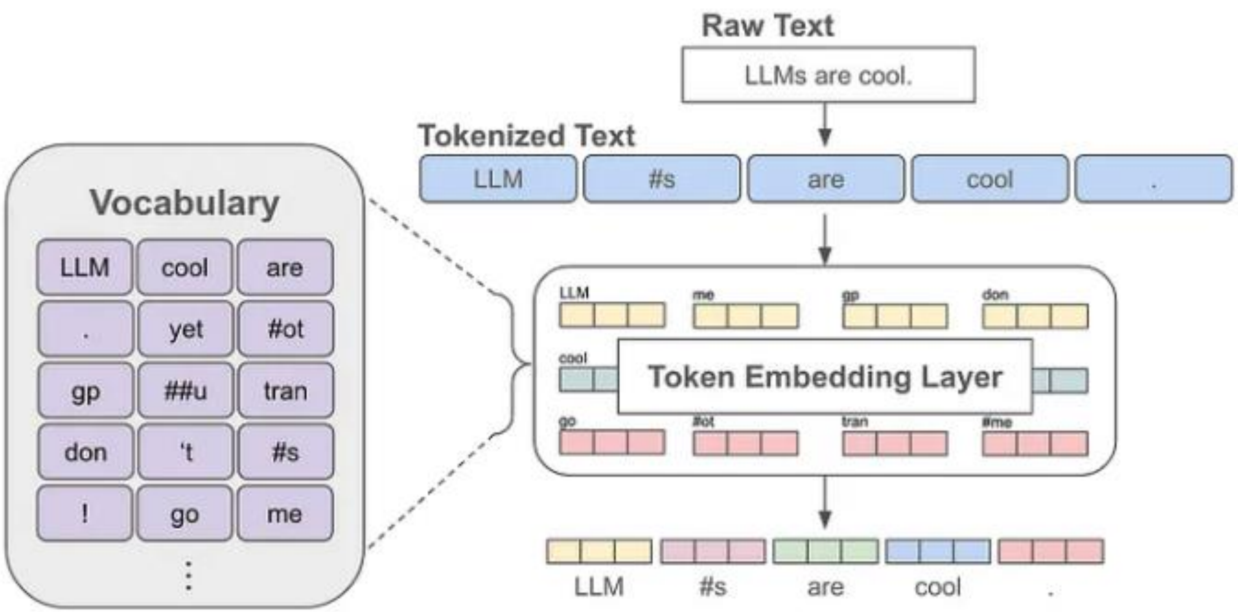


Language Mining – LLM

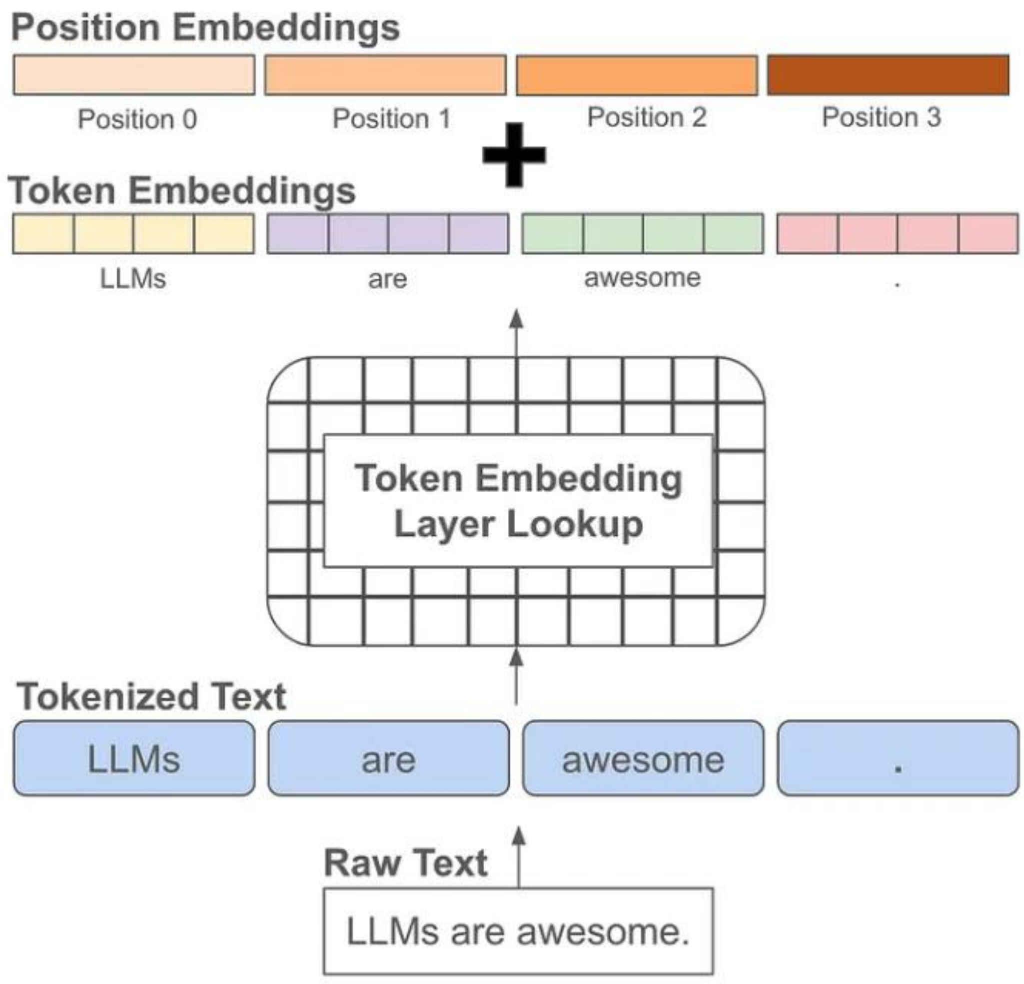


Language Mining – LLM



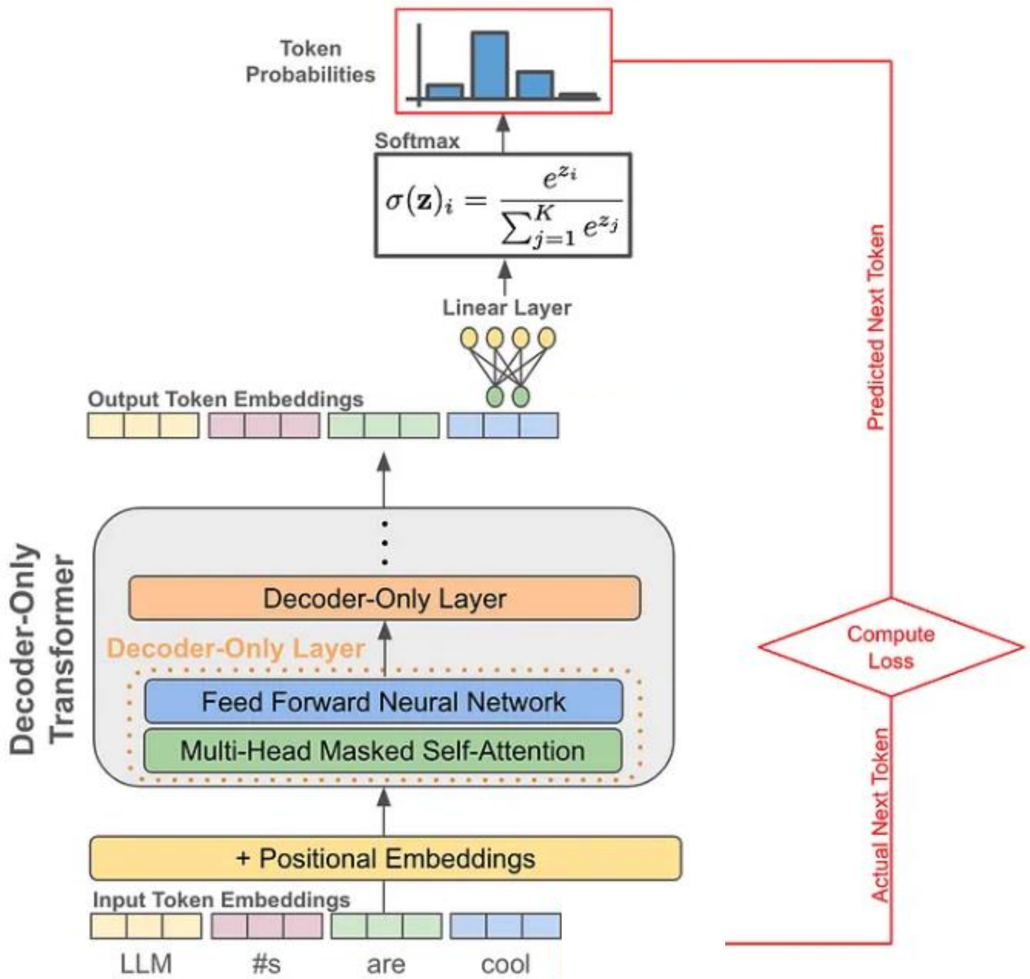


Language Mining – LLM



Position Embeddings With In a Language Model

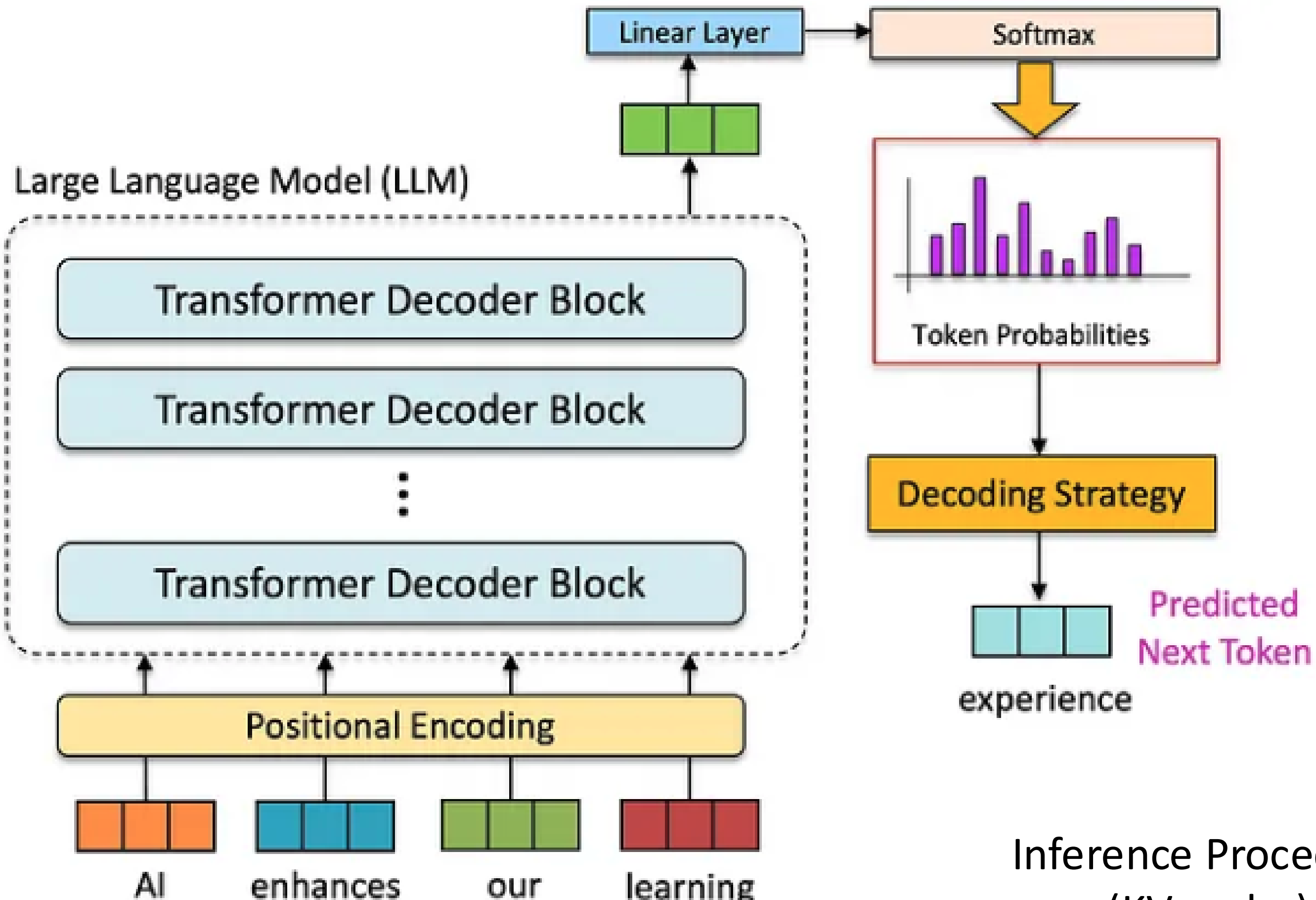
Language Mining – LLM



Training Procedure

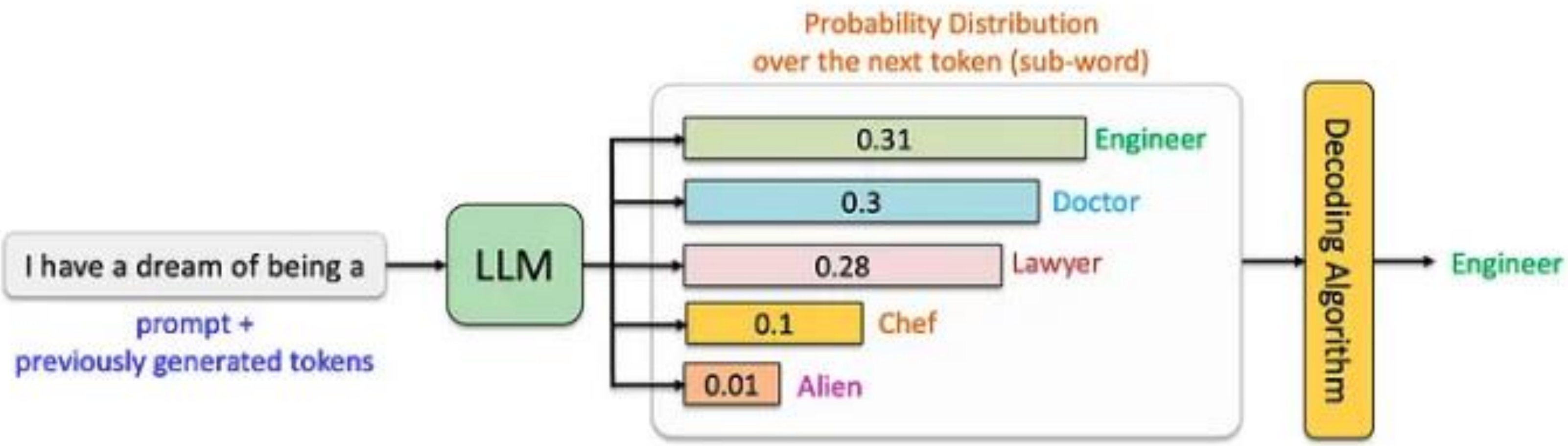
$$\mathcal{U} = \{u_1, u_2, \ldots, u_N\} \qquad \underbrace{\mathcal{L}(\mathcal{U})}_{\text{Language model loss over the full text corpus}} = \sum_{i=1}^{N} \log \big(\underbrace{\mathbb{P}(u_i | u_{i-k}, \ldots, u_{i-1}, \Theta)}_{\text{Conditional probability of i-th token given k preceding tokens and model parameters } \theta}\big)$$

Language Mining – LLM



Inference Procedure
(KV cache)

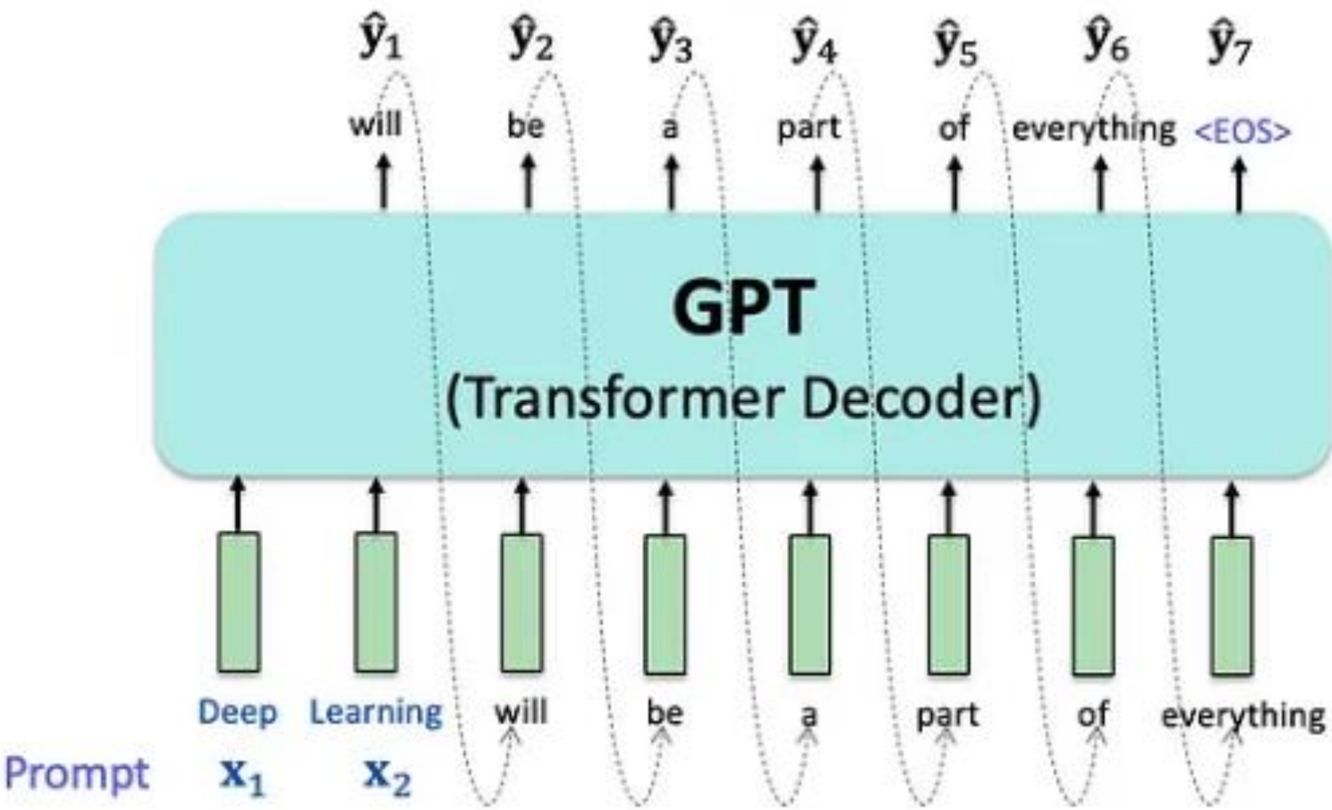
Language Mining – LLM



Autoregressive Nature

$$P(\hat{\mathbf{y}}_i | \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m, \hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_{i-1})$$

Language Mining – LLM

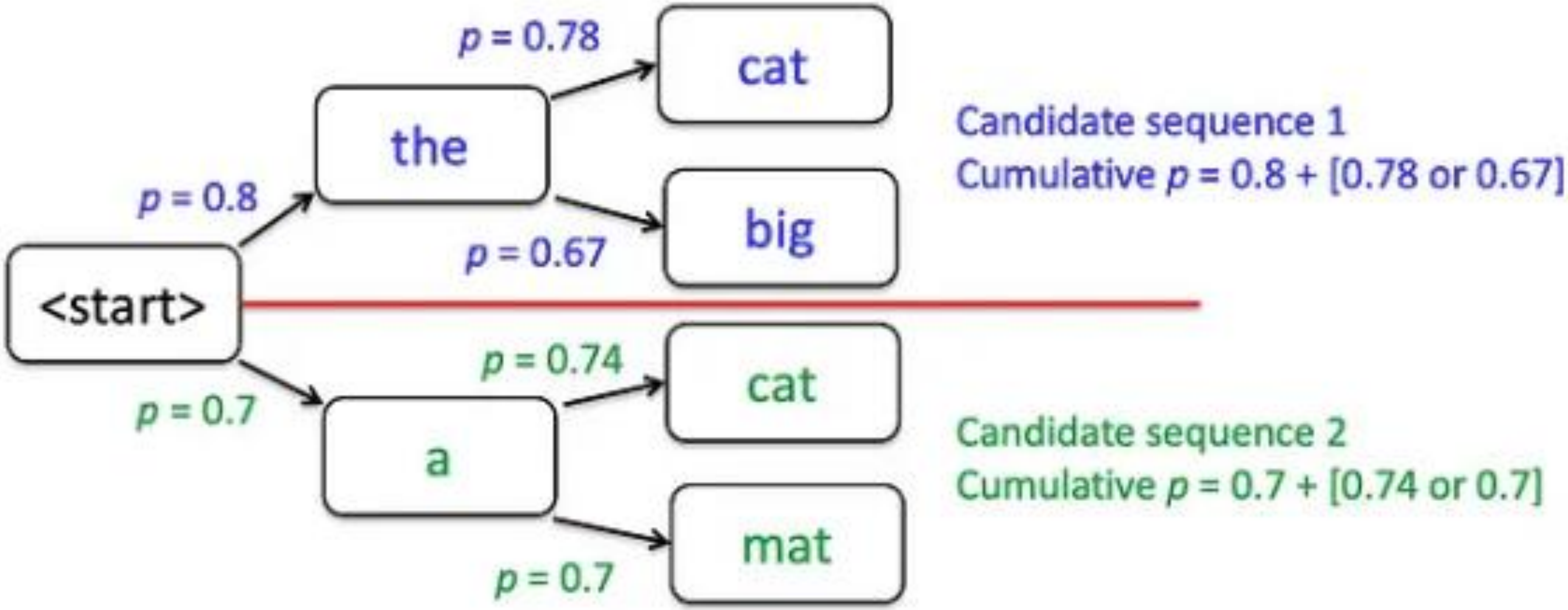


$$\hat{\mathbf{y}}_i = \arg\max_{\hat{\mathbf{y}}} P(\hat{\mathbf{y}}|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m, \hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_{i-1})$$

Greedy Decoding

Greedy decoding is computationally efficient and easy to implement.
It does not explore alternative paths that might lead to more globally optimal sequences.

Language Mining – LLM



https://medium.com/@lmpo/mastering-llms-a-guide-to-decoding-algorithms-c90a48fd167b

Question Time!