அத்தியாயம் 3: வார்த்தைகளின் உலகம் - சொல்லுக்குச் சொல் புரிதல்

மொழியைச் சுத்தம் செய்து, அதன் இலக்கணப் பாத்திரங்களை அடையாளம் கண்ட பிறகு, செயற்கை நுண்ணறிவு (AI) தனது அடுத்த சவாலைச் சந்திக்கிறது: ஒரு முழு வாக்கியத்தின் அல்லது பத்தியின் அர்த்தத்தை எண்களாக மாற்றி, கணினிக்குப் புரிய வைப்பது எப்படி?

இந்த மாபெரும் கேள்விக்கான விடை, ஒரு சிக்கலான புதிய கோட்பாட்டிலிருந்து வரவில்லை. மாறாக, 1950-களிலேயே Zellig Harris போன்ற மொழியியல் ஆய்வாளர்களால் முன்மொழியப்பட்டது. இது மொழியியல் மற்றும் தகவல் மீட்புத் துறைகளில் பயன்படுத்தப்பட்ட, ஒரு மிக எளிமையான ஆனால் புரட்சிகரமான யோசனையிலிருந்து வந்தது. அந்த யோசனை இதுதான்: "ஒரு வாக்கியத்தின் அர்த்தத்தைப் புரிந்துகொள்ள, அதன் சிக்கலான இலக்கணத்தையும், வார்த்தைகளின் வரிசையையும் தற்காலிகமாக மறந்துவிட்டு, அதில் உள்ள வார்த்தைகளை மட்டும் கணக்கில் எடுத்துக்கொண்டால் என்ன?"

இந்த அணுகுமுறைதான் **பேக் ஆஃப் வேர்ட்ஸ் (Bag of Words - BoW)**.

இந்த எளிய தந்திரத்தின் மூலம், சிக்கலான மனித மொழி, கணினிகளுக்குப் புரியும் எளிய எண்களின் வரிசையாக (Numerical Vector) மாறுகிறது. இதுவே, கணினிகள் உரையைப் புரிந்துகொள்ள எடுத்த முதல் மற்றும் மிக முக்கியமான படியாகும்.

3.1. பேக் ஆஃப் வேர்ட்ஸ்: வார்த்தைகளின் மந்திரப் பை

இதன் பெயர் குறிப்பிடுவதைப் போலவே, ஒரு வாக்கியத்தில் உள்ள அத்தனை வார்த்தைகளையும் எடுத்து, ஒரு பையில் (Bag) போட்டுக் குலுக்குவதாகக் கற்பனை செய்துகொள்ளுங்கள். அந்தப் பைக்குள், வார்த்தைகளின் வரிசை, அவற்றுக்கு இடையேயான தொடர்பு என அனைத்தும் கலைந்துவிடும். ஆனால், முடிவில் அந்தப் பையிடம் ஒரு முக்கியமான தகவல் மட்டும் மிஞ்சியிருக்கும்: **எந்த வார்த்தை, எத்தனை முறை இடம்பெற்றுள்ளது** என்பதுதான் அது.

இந்த மந்திரப் பை எப்படி வேலை செய்கிறது?

1. **சொல் அகராதியை உருவாக்குதல் (Vocabulary Creation):** முதலில், நம்மிடம் உள்ள எல்லா வாக்கியங்களிலும் இருக்கும் தனித்துவமான வார்த்தைகளை எல்லாம் சேகரித்து, அந்த மந்திரப் பையின் "அகராதியை" உருவாக்குகிறோம்.
    - **வாக்கியம் 1:** "I love programming."
    - **வாக்கியம் 2:** "Programming is fun."

    இந்த இரண்டு வாக்கியங்களிலிருந்து Vocabulary உருவாக்கும் போது:

   a. **ஒவ்வொரு வாக்கியத்தையும் சொற்களாகப் பிரிக்கிறோம் (Tokenization):**
      - "I", "love", "programming" (முதல் வாக்கியம்)
      - "Programming", "is", "fun" (இரண்டாம் வாக்கியம்)

   b. **தனித்துவமான சொற்களை மட்டும் எடுத்துக்கொள்கிறோம்:**
      - "I", "love", "programming", "is", "fun"

   c. **Vocabulary உருவாகிறது:**

```
Vocabulary = ["I", "love", "programming", "is", "fun"]
```

> *குறிப்பு:*
>
> - *Vocabulary-ல் உள்ள சொற்களின் வரிசை முக்கியமில்லை, ஆனால் ஒவ்வொரு வாக்கியத்தையும் இதே வரிசையில் குறியாக்கம் செய்ய வேண்டும்.*
> - *பொதுவாக, stop words ("I", "is" போன்றவை) நீக்கப்படலாம், ஆனால் இங்கே எளிமைக்காக வைத்துள்ளோம்.*

2. **எண்ணிக்கையைக் கணக்கிடுதல் (Frequency Calculation):** இப்போது, ஒவ்வொரு வாக்கியத்தையும் எடுத்து, அதில் அகராதியில் உள்ள வார்த்தைகள் எத்தனை முறை வந்துள்ளன என்று கணக்கிடுகிறோம்.

   Vocabulary: ["I", "love", "programming", "is", "fun"]

   - **வாக்கியம் 1:** "I love programming."
     - "I" → 1 (அகராதியில் உள்ளது)
     - "love" → 1 (அகராதியில் உள்ளது)
     - "programming" → 1 (அகராதியில் உள்ளது)
     - "is" → 0 (அகராதியில் இல்லை)
     - "fun" → 0 (அகராதியில் இல்லை)

   இதன் வெக்டர்: ["I", "love", "programming", "is", "fun"] → [1, 1, 1, 0, 0]

   - **வாக்கியம் 2:** "Programming is fun."
     - "I" → 0 (அகராதியில் இல்லை)
     - "love" → 0 (அகராதியில் இல்லை)
     - "programming" → 1 (அகராதியில் உள்ளது)

- "is" → 1 (அகராதியில் உள்ளது)
- "fun" → 1 (அகராதியில் உள்ளது)

இதன் வெக்டர்: ["I", "love", "programming", "is", "fun"] → [0, 0, 1, 1, 1]

> *குறிப்பு:*
>
> - *ஒரு சொல் ஒரு வாக்கியத்தில் பல முறை வந்தால், அந்த எண்ணிக்கை வெக்டரில் காட்டப்படும்.*
> - *எடுத்துக்காட்டு: "Programming is fun. I love programming."*
>
>   *["I", "love", "programming", "is", "fun"] → [1, 1, 2, 1, 1] ("programming" 2 முறை வந்ததால், அதன் மதிப்பு 2).*

இந்த எண்களின் வரிசையைப் பார்த்ததும், கணினிக்கு அந்த வாக்கியத்தின் ஒரு கணிதரீதியான புரிதல் கிடைத்துவிடுகிறது.

BoW: எளிமையின் வலிமையும், அறியாமையின் பலவீனமும்

BoW முறையின் மிகப்பெரிய பலம் அதன் எளிமை. இலக்கணத்தின் சிக்கலான விதிகளைப் புறக்கணித்து, வார்த்தைகளின் எண்ணிக்கையை மட்டும் வைத்து, அது மொழியை கணிதத்திற்குள் கொண்டுவருகிறது. ஆனால், அதன் மிகப்பெரிய பலவீனமும் இதுதான். அது **சூழலை (Context)** முற்றிலும் மறந்துவிடுகிறது.

> *"The child makes the dog happy" மற்றும் "The dog makes the child happy"*

இந்த இரண்டு வாக்கியங்களின் அர்த்தமும் முற்றிலும் வேறு. ஆனால், BoW-ன் மந்திரப் பையைப் பொறுத்தவரை, இரண்டு வாக்கியங்களிலும் ஒரே வார்த்தைகள், ஒரே எண்ணிக்கையில் இருப்பதால், இரண்டும் ஒன்றுதான்! இதுவே அதன் அறியாமையின் விளைவு. இந்தக் குறைபாடு இருந்தாலும், பல நிஜ உலகப் பயன்பாடுகளில் BoW ஒரு சக்திவாய்ந்த கருவியாக விளங்குகிறது:

ஸ்பேம் மின்னஞ்சல் கண்டறிதல் (Spam Detection)

ஒரு மின்னஞ்சலில் "Free," "Offer," "Win" போன்ற வார்த்தைகள் எத்தனை முறை வருகின்றன என்பதைக் கணக்கிட்டு, அது ஸ்பேம் ஆ இல்லையா என்பதை எளிதாகக் கண்டறிகிறது.

உதாரணம்:

- **மின்னஞ்சல் 1:** `"Congratulations! You win a free prize. Click this exclusive offer now!"`
    - *பகுப்பாய்வு:* இதில் `win`, `free`, `prize`, `offer` போன்ற ஸ்பேம் வார்த்தைகள் அதிக எண்ணிக்கையில் உள்ளன. எனவே, BoW மாதிரி இதற்கு **அதிக ஸ்பேம் மதிப்பெண்** கொடுத்து, அதை உங்கள் 'Junk' ஃபோல்டருக்கு அனுப்பிவிடும்.
- **மின்னஞ்சல் 2:** `"Hi, the report for our quarterly meeting is attached. Please review."`
    - *பகுப்பாய்வு:* இதில் ஸ்பேம் வார்த்தைகள் எதுவும் இல்லை. எனவே, இதற்கு **குறைந்த ஸ்பேம் மதிப்பெண்** கொடுத்து, உங்கள் 'Inbox'-ல் பாதுகாப்பாக வைக்கும்.

உணர்வுப் பகுப்பாய்வு (Sentiment Analysis)

ஒரு விமர்சனத்தில் "love," "amazing" போன்ற நேர்மறை வார்த்தைகள் அதிகமாக உள்ளதா, அல்லது "terrible," "worst" போன்ற எதிர்மறை வார்த்தைகள் அதிகமாக உள்ளதா என்பதைக் கொண்டு, அந்த விமர்சனத்தின் உணர்வைக் கணிக்கிறது.

உதாரணம்:

- **விமர்சனம் 1:** `"I love this new phone. The camera is amazing and the performance is great!"`
    - *பகுப்பாய்வு:* `love` (+1), `amazing` (+1), `great` (+1) போன்ற வார்த்தைகளால், இதன் மொத்த உணர்வு மதிப்பெண் **+3 (நேர்மறை)** என வகைப்படுத்தப்படும்.
- **விமர்சனம் 2:** `"This is a terrible product. The battery life is the worst."`
    - *பகுப்பாய்வு:* `terrible` (-1), `worst` (-1) போன்ற வார்த்தைகளால், இதன் மொத்த உணர்வு மதிப்பெண் **-2 (எதிர்மறை)** என வகைப்படுத்தப்படும்.

தகவல் மீட்பு (Information Retrieval)

கூகுள் போன்ற தேடுபொறிகள், நீங்கள் தேடும் வார்த்தைகள் எந்தெந்தப் பக்கங்களில் அதிகமாக வருகின்றன என்பதைக் கண்டறிந்து, மிகவும் பொருத்தமான பக்கங்களை முதலில் காட்டுகின்றன.

உதாரணம்:

- **உங்கள் தேடல்:** `"easy python programming tutorial"`
- **பக்கம் A:** "An introduction to Python programming... This easy tutorial..." (`python`: 2, `programming`: 1, `easy`: 1, `tutorial`: 1 -> **மொத்த பொருத்தம்: 5**)
- **பக்கம் B:** "A history of the Python snake. A brief tutorial on snakes." (`python`: 1, `tutorial`: 1 -> **மொத்த பொருத்தம்: 2**)
- *பகுப்பாய்வு:* உங்கள் தேடல் வார்த்தைகள் **பக்கம் A**-ல் அதிக முறை வருவதால், தேடுபொறி அதை உங்களுக்கு முதல் தேர்வாகக் காட்டும்.

BoW, செயற்கை நுண்ணறிவு மொழியைப் புரிந்துகொள்ள எடுத்த ஒரு மாபெரும் முதல் படி என்பதில் சந்தேகமில்லை. அது, சிக்கலான மனித மொழியை, கணினிகளுக்குப் புரியும் எளிய எண்களாக மாற்றி, ஒரு புதிய பாதையைத் திறந்துவிட்டது.

ஆனால், அந்த எளிமைக்கு ஒரு பெரிய விலை கொடுக்கப்பட்டது: **அர்த்தம் (Meaning)**.

BoW-ன் உலகில், "நாய் மனிதனைக் கடித்தது" என்பதும், "மனிதன் நாயைக் கடித்தான்" என்பதும் ஒன்றுதான். ஏனெனில், அது வார்த்தைகளைக் கணக்கிடுமே தவிர, அவற்றின் வரிசையையோ, அவற்றுக்கு இடையிலான உறவையோ புரிந்துகொள்ளாது. மேலும், அது "the" போன்ற பயனற்ற வார்த்தைகளைப் போலவே, "awesome" போன்ற முக்கியமான வார்த்தைகளையும் வெறும் எண்ணிக்கையாகவே பார்த்தது.

> *ஒரு வார்த்தையின் உண்மையான முக்கியத்துவத்தை எப்படி அளவிடுவது? வார்த்தைகளுக்கு இடையிலான ஆழமான தொடர்புகளை எப்படிப் புரிந்துகொள்வது?*

இந்தக் கேள்விகள்தான், AI-யை வார்த்தைகளைக் கணக்கிடும் நிலையிலிருந்து, அவற்றின் அர்த்தத்தையும், முக்கியத்துவத்தையும் எடைபோடும் அடுத்தகட்டப் பயணத்திற்கு அழைத்துச் சென்றன.

3.2. எம்பெடிங்ஸ்: வார்த்தைகளுக்கு ஓர் ஆன்மாவைக் கொடுத்தல்

பேக் ஆஃப் வேர்ட்ஸ் (BoW), மொழியை எண்களாக மாற்றும் முதல் படியை நமக்குக் காட்டியது. ஆனால், அது வார்த்தைகளின் ஆன்மாவை, அதாவது அதன் ஆழமான அர்த்தத்தையும், மற்ற வார்த்தைகளுடன் அது கொண்டுள்ள உறவையும் புரிந்துகொள்ளத் தவறியது.

"நாய் மனிதனைக் கடித்தது" என்பதற்கும், "மனிதன் நாயைக் கடித்தது" என்பதற்கும் உள்ள வித்தியாசத்தை அதனால் உணர முடியவில்லை. அது வார்த்தைகளின் ஆன்மாவை, அதாவது அதன் **அர்த்தத்தை (Meaning)** இழந்தது.

"நாய்," "பூனை," "அரசன்," "ராணி" - இந்த வார்த்தைகள் அனைத்தும் BoW-க்கு வெறும் தனித்தனி அடையாளங்கள். ஆனால், நமக்கோ "நாயும்" "பூனையும்" விலங்குகள் என்றும், "அரசனும்" "ராணியும்" ஆளுமை மிக்க மனிதர்கள் என்றும், அவர்களுக்கிடையே ஒரு தொடர்பு உள்ளது என்றும் தெரியும்.

இந்த மாபெரும் சவாலைச் சமாளிக்க, AI தனது அடுத்த புரட்சிகரமான படியை எடுத்து வைத்தது. அதுதான் **எம்பெடிங் (Embedding)**. BoW என்பது வார்த்தைகளை ஒரு பையில் அள்ளிக் கொட்டுவது என்றால், எம்பெடிங் என்பது வார்த்தைகளுக்கென ஒரு பிரம்மாண்டமான **பிரபஞ்சத்தை (Universe)** அல்லது ஒரு வரைபடத்தை உருவாக்குவது.

Embedding-கள் என்பது ஒரு வகையான **தரவு பிரதிநிதித்துவம் (Data Representation)**. இது உரை (text), படங்கள் (images), ஒலி (audio) போன்ற தரவுகளை எண்களாக மாற்றி, கணினிகள் புரிந்து கொள்ளும் வகையில் காட்டுகிறது. Embedding-கள் முக்கியமாக **Natural Language Processing (NLP)**-ல் பயன்படுத்தப்படுகிறது, அங்கு வார்த்தைகள், வாக்கியங்கள் அல்லது பத்திகள் எண்களாக மாற்றப்படுகின்றன (Numerical Vector). இது வெறும் எண்ணிக்கை அல்ல; இது வார்த்தையின் பல பரிமாண அடையாளம் (Multi-dimensional Identity) அல்லது அதன் டிஜிட்டல் டி.என்.ஏ (Digital DNA).

வார்த்தைப் பிரபஞ்சத்தின் வரைபடம்

இந்த வரைபடத்தில், ஒவ்வொரு வார்த்தைக்கும் ஒரு தனித்துவமான இடம், ஒரு குறிப்பிட்ட ஆயத்தொலைவு (Co-ordinate) உண்டு. அந்த ஆயத்தொலைவுதான், அந்த வார்த்தையின் **எம்பெடிங் வெக்டர் (Embedding Vector)** - அதாவது, நூற்றுக்கணக்கான எண்களைக் கொண்ட ஒரு வரிசை.

இந்த வரைபடத்தின் மந்திர விதி இதுதான்: **ஒரே மாதிரியான அர்த்தம் கொண்ட வார்த்தைகள், வரைபடத்தில் அருகருகே அமைந்திருக்கும்.**

- "அரசன்" (King), "ராணி" (Queen), "இளவரசர்" (Prince) போன்ற வார்த்தைகள், "அரச குடும்பம்" என்ற பகுதியில் ஒன்றாக இருக்கும்.
- "மகிழ்ச்சி" (Happy), "ஆனந்தம்" (Joyful), "பேரானந்தம்" (Ecstatic) ஆகியவை, "நேர்மறை உணர்வுகள்" என்ற பகுதியில் நெருங்கிய நண்பர்களாக இருக்கும்.

ஒரு வெக்டர் என்ன சொல்கிறது?

"அரசன்" (King) என்ற வார்த்தையின் வெக்டரான `[0.95, -0.12, 0.87, ...]` என்பதில் உள்ள ஒவ்வொரு எண்ணும், அந்த வார்த்தைப் பிரபஞ்சத்தில் அதன் இடத்தைக் குறிக்கிறது.

- **முதல் எண் (0.95):** இது "அரசப் பண்பு" (Royalty) என்ற அச்சில், "அரசன்" எவ்வளவு தூரத்தில் இருக்கிறார் என்பதைக் குறிக்கலாம். மதிப்பு அதிகமாக இருப்பதால், தொடர்பு வலுவாக இருக்கிறது.
- **இரண்டாம் எண் (-0.12):** இது "பாலினம்" (Gender) என்ற அச்சைக் குறிக்கலாம். எதிர்மறை மதிப்பு, அது ஆண்பாலைச் சார்ந்தது என்பதைக் காட்டுகிறது. ஒருவேளை, "ராணி" (Queen) என்ற வார்த்தைக்கு இதே மதிப்பு `+0.15` என நேர்மறையாக இருக்கலாம்.
- **மூன்றாம் எண் (0.87):** இது "அதிகாரம்" (Authority) என்ற அச்சைக் குறிக்கலாம். "அரசன்" என்ற சொல்லுக்கு அதிகாரம் உண்டு என்பதை இந்த உயர் மதிப்பு காட்டுகிறது. இதுவே, "வேலைக்காரன்" (Servant) என்ற சொல்லுக்கு `-0.90` என இருக்கலாம்.

இந்த வரைபடம் மிகவும் துல்லியமானது, "அரசன்" என்ற புள்ளியிலிருந்து "மனிதன்" (Man) என்ற புள்ளியை கழித்து, "பெண்" (Woman) என்ற புள்ளியைக் கூட்டினால், நாம் கிட்டத்தட்ட "ராணி" (Queen) என்ற புள்ளியை அடைந்துவிடலாம்! (`King - Man + Woman ≈ Queen`). இந்த வரைபடத்தில், ஒவ்வொரு வார்த்தைக்கும் ஒரு தனித்துவமான இடம், ஒரு குறிப்பிட்ட ஆயத்தொலைவு உண்டு. அந்த ஆயத்தொலைவுதான், அந்த வார்த்தையின் எம்பெடிங் வெக்டர் (Embedding Vector).

இதுதான் எம்பெடிங்கின் சக்தி. அது வார்த்தைகளுக்கு இடையிலான சிக்கலான உறவுகளைக் கணித ரீதியாகப் புரிந்துகொள்கிறது. ஒவ்வொரு வார்த்தைக்கும், நியூரல் நெட்வொர்க்குகள் புரிந்துகொள்ளும் மொழியில், நூற்றுக்கணக்கான பரிமாணங்களைக் கொண்ட ஒரு தொடர்ச்சியான வெக்டரை (continuous vector) அது முகவரியாக வழங்குகிறது. இது, ஒரு சொல்லின் அர்த்தத்தையும் (semantic features), அதன் இலக்கணப் பண்புகளையும் (syntactic features) உள்ளடக்கியது. மேலும், ஒரு சொல்லின் ஆன்மா, ஒரே எண்ணில் அடங்காமல், அதன் நூற்றுக்கணக்கான பரிமாணங்களில் ஒரு பரவிக் காணப்படும் பிரதிநிதித்துவமாக (distributed representation) மிளிர்கிறது. GPT-3 போன்ற மாபெரும் மொழி மாதிரிகளில் (LLMs), ஒவ்வொரு வார்த்தைக்கும் **768 பரிமாணங்கள் (dimensions)** உள்ளன. அதாவது, ஒவ்வொரு வார்த்தையின் இடத்தையும் 768 வெவ்வேறு அச்சுகளைக் கொண்டு மிக நுணுக்கமாக இந்த வரைபடம் வரையறுக்கிறது

இந்த எம்பெடிங் தொழில்நுட்பம்தான், இன்று நாம் காணும் அனைத்து நவீன டீப் லேர்னிங் மாதிரிகளின் அடித்தளமாகும். இது தனிப்பட்ட வார்த்தைகளுக்கு மட்டுமல்ல, முழுமையான வாக்கியங்களுக்கும், ஏன், பெரிய ஆவணங்களுக்குமே உருவாக்கப்படுகிறது. இயந்திர மொழிபெயர்ப்பு முதல் உணர்ச்சிப் பகுப்பாய்வு வரை, எண்ணற்ற பயன்பாடுகளுக்கு இதுவே அடிப்படையாக அமைகிறது.

இரண்டு உலகங்களின் வேறுபாடு

இந்த இடத்தில், நாம் ஒரு மாபெரும் மாற்றத்தைக் காண்கிறோம். "பேக் ஆஃப் வேர்ட்ஸ்" (BoW) என்ற எளிய கணக்காளருக்கும், "எம்பெடிங்" என்ற பிரபஞ்ச வரைபடக் கலைஞருக்கும் உள்ள வேறுபாடு இதுதான்:

BoW, வார்த்தைகளை வெறும் எண்ணிக்கையாகப் பார்த்தது; எம்பெடிங், அவற்றின் உறவுகளையும், அர்த்தத்தையும் புரிந்துகொண்டது. BoW, ஒரு பையில் உள்ள தனித்தனிப் பொருட்களைப் போல வார்த்தைகளைப் பார்த்தது; எம்பெடிங், ஒரு பிரபஞ்சத்தில் உள்ள நட்சத்திரங்களைப் போல, அவற்றின் ஆயத்தொலைவுகளையும், அவற்றுக்கு இடையிலான ஈர்ப்பு விசையையும் கண்டறிந்தது. BoW, சூழலை முற்றிலும் புறக்கணித்தது; எம்பெடிங், சூழலையே தனது மிகப்பெரிய பலமாகக் கொண்டது. இதனால்தான், BoW எளிய வகைப்படுத்தல் பணிகளோடு நின்றுவிட, எம்பெடிங் இன்று நாம் காணும் அனைத்து நவீன AI மொழி மாதிரிகளுக்கும் ஆன்மாவாக விளங்குகிறது.

சுருக்கமாக, இந்த மாற்றம் AI-யின் பார்வையை முற்றிலுமாக மாற்றியது.

> *"பேக் ஆஃப் வேர்ட்ஸ்" (BoW), கணினிக்கு வார்த்தைகள் என்ற எழுத்துக்களை எழுத்துக்கூட்டிக் கணக்கிடக் கற்றுக்கொடுத்தது. அது ஒரு கணக்காளரின் பார்வை.*

> *ஆனால் "எம்பெடிங்", அந்த வார்த்தைகளுக்கு இடையிலான ஓசை, நயம், மற்றும் ஆழமான உறவுகளை உணரக் கற்றுக்கொடுத்தது. அது ஒரு கவிஞனின் பார்வை.*

வார்த்தைகளைக் கணக்கிடும் இயந்திரமாக இருந்த AI, வார்த்தைகளின் ஆன்மாவை உணரும் ஒரு கலைஞனாகப் பரிணாமம் அடைந்த தருணம் இது. இந்த ஒற்றைப் பாய்ச்சல்தான், இன்று நாம் காணும் அனைத்து நவீன மொழி மாதிரிகளின் அடித்தளமும், ஆன்மாவும் ஆகும்.

3.3. Word2Vec: வார்த்தைகளின் உறவை வரையும் கலை

"பேக் ஆஃப் வேர்ட்ஸ்" (BoW), வார்த்தைகளைக் கணக்கிட்டதே தவிர, அவற்றின் ஆழமான அர்த்தத்தையோ, 'அரசன்' மற்றும் 'ராணி' போன்ற சொற்களுக்கு இடையிலான நுட்பமான தொடர்பையோ புரிந்துகொள்ளும் திறனற்றதாக இருந்தது. இந்த மாபெரும் சவாலைச் சமாளிக்கவே, வார்த்தைகளை அவற்றின் உறவுகளின் அடிப்படையில் ஒரு பிரபஞ்சத்தில் நிலைநிறுத்தும் 'எம்பெடிங்' என்ற தத்துவம் பிறந்தது.

எம்பெடிங் என்பது வார்த்தைகளுக்கு ஒரு பிரபஞ்சத்தை உருவாக்கும் தத்துவம் என்றால், அந்தப் பிரபஞ்சத்தை முதன்முதலில் துல்லியமாக வரைந்த வரைபடக் கலைஞன்தான் **Word2Vec**.

2013-ல், கூகுளில் பணியாற்றிய தாமஸ் மிக்கோலோவ் (Tomas Mikolov) மற்றும் அவரது குழுவினர், இந்த புரட்சிகரமானதொழில்நுட்பத்தை உலகுக்கு அறிமுகப்படுத்தினர். இது, வார்த்தைகளுக்கு இடையிலான ஆழமான, மறைந்திருக்கும் உறவுகளைக் கண்டறியும் ஒரு நேர்த்தியான முறையாகும். Word2Vec, இந்த உறவுகளைக் கண்டறிய இரண்டு வெவ்வேறு, ஆனால் மிக புத்திசாலித்தனமான வழிகளைக் கையாள்கிறது.

வழி 1: CBOW - துப்புத் துலக்கும் துப்பறிவாளன்

முதல் முறையான **CBOW (Continuous Bag of Words)**, ஒரு புதிரை விடுவிக்கும் அனுபவமிக்க துப்பறிவாளனைப் போலச் செயல்படுகிறது. அது, ஒரு வாக்கியத்தில் உள்ள சுற்றியுள்ள வார்த்தைகளை (சூழலை) ஆதாரமாக எடுத்துக்கொண்டு, மையத்தில் காணாமல் போன வார்த்தை என்னவாக இருக்கும் என்று யூகிக்கும்.

- **சூழல்:** `"மூன்று நாட்களாக மழை ____ பெய்து கொண்டிருக்கிறது."`
- **CBOW-ன் யூகம்:** சுற்றியுள்ள வார்த்தைகளைப் பார்த்து, இடத்தில் "**தொடர்ந்து**" என்ற வார்த்தையே மிகச் சரியாகப் பொருந்தும் என்று அது கணிக்கும்.

இந்த முறை, சிறிய தரவுத் தொகுப்புகளில் மிக வேகமாகவும், திறமையாகவும் செயல்படும்.

வழி 2: Skip-Gram - கற்பனை வளம் மிக்க கதைசொல்லி

இரண்டாவது முறையான **Skip-Gram**, இதற்கு நேர்மாறாக, ஒரு கற்பனை வளம் மிக்க கதைசொல்லியைப் போலச் செயல்படுகிறது. நீங்கள் ஒரு மைய வார்த்தையை (கருவை) அதனிடம் கொடுத்தால், அதைச் சுற்றி என்னென்ன வார்த்தைகள் (கதை) வர வாய்ப்புள்ளது என்று அது யூகிக்கும்.

- **மைய வார்த்தை:** `"கணினி"`
- **Skip-Gram-ன் யூகம்:** அதிலிருந்து, "**மின்னணு,**" "**தொழில்நுட்பம்,**" "**இணையம்,**" "**மென்பொருள்**" போன்ற தொடர்புடைய வார்த்தைகளின் ஒரு கூட்டத்தை அது உருவாக்கும்.

இந்த முறை, பெரிய தரவுத் தொகுப்புகளில், குறிப்பாக அரிதான வார்த்தைகளின் நுணுக்கமான அர்த்தத்தைப் புரிந்துகொள்வதில் சிறப்பாகச் செயல்படுகிறது.

எம்பெடிங்குகளின் உண்மையான மாயாஜாலம், அவை வார்த்தைகளின் உறவுகளை எளிய கணிதச் சமன்பாடுகளாக மாற்றுவதில்தான் உள்ளது. இதன் மிக பிரபலமான உதாரணம் இதுதான்:

`அரசன் − ஆண் + பெண் ≈ அரசி`

இந்தச் சமன்பாடு, வெறும் வார்த்தைகளை வைத்து விளையாடும் தந்திரம் அல்ல. இது, எம்பெடிங்குகள் மொழியின் ஆழமான கட்டமைப்பைப் புரிந்துகொண்டதற்கான அசைக்க முடியாத சாட்சி.

`vector("அரசன்") - vector("ஆண்") + vector("பெண்") ≈ vector("அரசி")`

1. **ஒவ்வொரு சொல்லும் ஒரு எண் வரிசை (Vector):**
    - "அரசன்" = [0.8, 0.2, 0.7,...]
    - "ஆண்" = [0.9, 0.1, 0.6,...]
    - "பெண்" = [0.1, 0.9, 0.6,...]
2. **கணித செயல்பாடு:**

    ```
    [அரசன் வெக்டர்] - [ஆண் வெக்டர்] + [பெண் வெக்டர்]
    = [0.8-0.9+0.1, 0.2-0.1+0.9, 0.7-0.6+0.6,...]
    ≈ [0.0, 1.0, 0.7,...] (இது "அரசி" வெக்டருக்கு அருகில்)
    ```

3. **ஏன் இது வேலை செய்கிறது?**
    - எம்பெடிங்ஸ் **பாலினம்** (Gender) போன்ற பண்புகளை தனித்த பரிமாணங்களில் சேமிக்கிறது
    - "அரசன்" - "ஆண்" = "அரச தன்மை" (Royalty without gender)
    - "அரச தன்மை" + "பெண்" = "அரசி"
4. **செயல்பாட்டு விளக்கம்:**

    ```
    அரசன்: [அரச தன்மை + ஆண்மை]
    - ஆண்: [ஆண்மை]
    + பெண்: [பெண்மை]
    ----------------------------
    அரசி: [அரச தன்மை + பெண்மை]
    ```

இது எப்படி வேலை செய்கிறது?

Word2Vec, "அரசப் பண்பு" (Royalty), "பாலினம்" (Gender) போன்ற நுட்பமான கருத்துக்களைத் தனித்தனி பரிமாணங்களில் (dimensions) சேமிக்கிறது.

1. `vector("அரசன்")` என்பதில், **[அரசப் பண்பு + ஆண் பாலினம்]** ஆகிய இரண்டின் கூறுகளும் கலந்திருக்கும்.
2. அதிலிருந்து `vector("ஆண்")` என்பதைக் கழிக்கும்போது, **[ஆண் பாலினம்]** என்ற கூறு நீக்கப்பட்டு, **[அரசப் பண்பு]** மட்டும் எஞ்சியிருக்கும்.
3. இந்தத் தனித்த **[அரசப் பண்புடன்]**, `vector("பெண்")` என்பதைச் சேர்க்கும்போது, அது **[அரசப் பண்பு + பெண் பாலினம்]** என்று மாறுகிறது.

அதுதான் **"அரசி"**.

இப்படிப்பட்ட மாயாஜால உறவுகளைக் கண்டறிய, Word2Vec, அதன் CBOW மற்றும் Skip-Gram முறைகள் மூலம், கணினிகள் மொழியின் அர்த்தத்தை வெறுமனே மனப்பாடம் செய்யாமல், அதன் உள்ளார்ந்த உறவுகளைப் புரிந்துகொள்ளும் ஒரு புதிய சகாப்தத்தைத் தொடங்கி வைத்தது.

Word2Vec: நிஜ உலகின் தாக்கம்

Word2Vec என்ற வார்த்தைப் பிரபஞ்சத்தை வரையும் வரைபடக் கலைஞன், ஆய்வகங்களை விட்டு வெளியேறி, நிஜ உலகில் ஒரு மாபெரும் தாக்கத்தை ஏற்படுத்தினான்.

தேடுபொறிகளில் (Search Engines): அது தேடுபொறிகளுக்கு ஒரு புதிய ஞானத்தைக் கொடுத்தது. நீங்கள் "apple" என்று தேடும்போது, அது நீங்கள் உண்ணும் **பழத்தை** தேடுகிறீர்களா, அல்லது பயன்படுத்தும் **தொழில்நுட்ப நிறுவனத்தை** தேடுகிறீர்களா என்பதை, உங்கள் தேடலின் சூழலைக் கொண்டு அது புரிந்துகொண்டது. இந்த வகையான semantic precision, தேடல் முடிவுகளின் பொருத்தத்தையும் பயனருக்கு கிடைக்கும் தகவலின் தரத்தையும் அதிகரிக்கிறது. இது தேடல் முடிவுகளை முன்பை விட மிகத் துல்லியமாக்கியது.

மொழிபெயர்ப்பில் (Machine Translation): அது ஒரு பிரபஞ்ச மொழிபெயர்ப்பாளனாகச் செயல்பட்டது. ஒரு மொழியில் உள்ள வார்த்தையின் ஆன்மாவை (embedding) உணர்ந்து, மற்றொரு மொழியில் அதே ஆன்மாவைக் கொண்ட வார்த்தையைக் கண்டறிந்தது. இதனால், மொழிபெயர்ப்புகள் வெறும் வார்த்தை மாற்றமாக இல்லாமல், உணர்வுப்பூர்வமாகவும், இயல்பாகவும் మారின.

பரிந்துரை அமைப்புகளில் (Recommendation Systems): அமேசானில், "இதை வாங்கியவர்கள், அதையும் வாங்கினார்கள்" என்றோ, நெட்ஃபிளிக்ஸில், "உங்களுக்குப் பிடித்தமான மற்றத் தொடர்கள்" என்றோ வரும் பரிந்துரைகளின் பின்னணியில் இதன் தர்க்கமே உள்ளது. ஒரு திரைப்படத்தின் "உலகத்தை" உங்களுக்குப் பிடித்திருந்தால், அதே போன்ற உலகத்தைக் கொண்ட மற்றொரு திரைப்படத்தையும் அது உங்களுக்குப் பரிந்துரைக்கும்.

ஒரு தலைசிறந்த படைப்பின் சில குறைகள்

ஒவ்வொரு மாபெரும் படைப்பிற்கும் சில வரம்புகள் இருப்பது போல, Word2Vec-க்கும் சில குறைகள் இருந்தன.

1. **அறிமுகமில்லாத வார்த்தைகள் (Out-of-Vocabulary):** Word2Vec, ஒரு கடுமையான விருந்தினர் பட்டியல் கொண்ட ஒரு விருந்தைப் போன்றது. அது தனது பயிற்சியின் போது பார்க்காத ஒரு புதிய வார்த்தை வந்தால், அதை முற்றிலுமாகப் புறக்கணித்துவிடும்.
2. **சொல் உருவமைப்பை உணராமை (Morphology):** அது "ஓடு" (run), "ஓட்டப்பந்தய வீரர்" (runner), "ஓடிக்கொண்டிருத்தல்" (running) ஆகிய மூன்றையும் வெவ்வேறு அந்நியர்களாகவே பார்த்தது. அவை ஒரே குடும்பத்தைச் சேர்ந்தவை என்பதை அதனால் உணர முடியவில்லை.
3. **தரவுப் பசி (Data Requirement):** ஒரு சிறந்த கலைஞனைப் போல, தனது தலைசிறந்த படைப்பை உருவாக்க, அதற்கு ஒரு பரந்த பட்டறிவு (மிகப்பெரிய தரவுத்தளம்) தேவைப்பட்டது. இது, வளங்கள் குறைவாக உள்ள மொழிகளுக்கு ஒரு தடையாக அமைந்தது.

Word2Vec-ல் சில குறிப்பிடத்தக்க வரம்புகள் உள்ளன. முதன்மையானதாக தனது பயிற்சியின் போது பார்க்காத ஒரு புதிய வார்த்தை (Out-of-Vocabulary) வந்தால், அதை முற்றிலுமாகப் புறக்கணித்துவிடும். அந்தப் புதிய வார்த்தைக்கு அதன் உலகில் இடமில்லை. இந்த ‘Out-of-Vocabulary’ (OOV) பிரச்சனை, குறிப்பாக பன்மொழி மற்றும் வளர்ந்து வரும் மொழிகளில், ஒரு முக்கிய தடையாக இருக்கிறது.

இரண்டாவது, Word2Vec ஒவ்வொரு வார்த்தையையும் ஒரு தனி அலகாக கருதுகிறது. அது "ஓடு" (run), "ஓட்டப்பந்தய வீரர்" (runner), மற்றும் "ஓடிக்கொண்டிருத்தல்" (running) ஆகிய மூன்றையும் வெவ்வேறு அந்நியர்களாகவே பார்த்தது. அவை அனைத்தும் "ஓடு" என்ற ஒரே வேர்ச்சொல்லின் குடும்பத்தைச் சேர்ந்தவை என்பதை அதனால் உணர முடியவில்லை.

எனவே “run”, “runner”, “running” போன்ற வார்த்தைகளுக்கிடையிலான morphological தொடர்புகளைப் புரிந்துகொள்ள முடியாது. இது subword-level தகவல்களை (பிரிக்கக்கூடிய வேர்ச்சொல் அமைப்புகள்) புறக்கணிக்கிறது. இந்த நுணுக்கத்தைப் புரிந்துகொள்ள, **FastText** போன்ற அடுத்த தலைமுறை மாதிரிகள் தேவைப்பட்டன.

மூன்றாவது, புதிய அல்லது குறைவாக பேசப்படும் மொழிகளில் Word2Vec மாதிரியை பயிற்சி செய்ய பெரிய மற்றும் தரமான corpus தேவைப்படும். இது resource-poor மொழிகளில் இதை பயனுள்ளதாக உருவாக்க கடினமாக்குகிறது.

இந்த எல்லைகள் இருந்தபோதிலும், Word2Vec இன்னும் பல NLP பணிகளுக்கு ஒரு அடித்தளமான பங்கு வகிக்கிறது. அதன் எளிமையும், வேகமான பயிற்சித்திறனும், பயனுள்ள semantic representations-ஐ உருவாக்கும் திறனும், இதனை இன்றும் பரவலாகப் பயன்படுத்தத்தக்கதாக்குகின்றன.

Python-ல் Word2Vec மாதிரியை பயிற்சி செய்தல்:

```
from gensim.models import Word2Vec

# உதாரண வாக்கியங்கள்
sentences = [
    ["I", "love", "machine", "learning"],
    ["AI", "is", "fascinating"],
    ["I", "study", "NLP"]
]

# Word2Vec மாதிரியை பயிற்சி செய்தல்
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1,
workers=4)

# "king" என்ற வார்த்தையின் embedding-ஐ பெறுதல்
king_vector = model.wv['king']
print("ராஜா-ன் embedding:", king_vector)

# "king" என்ற வார்த்தைக்கு ஒத்த வார்த்தைகளை கண்டறிதல்
similar_words = model.wv.most_similar('king')
print("ராஜா-க்கு ஒத்த வார்த்தைகள்:", similar_words)
```

Gensim-ஐப் பயன்படுத்தி **Word2Vec** மாடல் ஒவ்வொரு வார்த்தைக்கும் ஒரு எம்பெடிங் (vector representation) உருவாக்குகிறது. மாடல் இந்த எம்பெடிங்களைக் கொண்டு வார்த்தைகளுக்கு இடையிலான தொடர்பை (semantic similarity) கண்டறிகிறது.

model.wv.most_similar('king') என்ற அழைப்பின் மூலம், **“king”** என்ற வார்த்தைக்கு ஆக்க பக்கத்தில் உள்ள தொடர்புடைய வார்த்தைகளை (similar words) கண்டறிகிறது.

```
[('ruler', 0.25290292501449585), ('queen', 0.1703130453824997),
('are', 0.15026721358299255),
('fascinating', 0.13887712359428406),
('NLP', 0.10853718221187592),
('love', 0.03476576507091522),
('and', 0.016124976798892021),
('I', 0.004502517171204090),
('study', -0.005892974324524403),
('is', -0.02770209312438965)]
```

('ruler', 0.2529): “ruler” என்பது **king**-க்கு மிக அண்மையான பொருள் கொண்ட வார்த்தை என மாடல் கருதுகிறது. இந்த similarity score **0.2529** என்பது **king** மற்றும் **ruler** வார்த்தைகளின் வெக்டர் இடையிலான cosine similarity அடிப்படையில் கணக்கிடப்பட்டது.

('queen', 0.1703): “queen” என்பது **king**-க்கான தொடர்புடைய மற்றொரு வார்த்தையாக இருக்கிறது. இது “ruler”-க்கு அடுத்தடுத்த பொருளுடையது, ஆனால் குறைவான ஒத்தபாடுடையது (**0.1703**).

('are', 0.1503): இது எம்பெடிங் டேட்டாவிலிருந்து, **are** என்ற வார்த்தை “king”-க்குப் பக்கத்தில் தோன்றியதால் ஒத்தவாறு தெரிகிறது.

('fascinating', 0.1389): இதுவும் **king**-க்கு தொடர்புடையதாக கண்டறியப்பட்டது, ஆனால் semantic தொடர்பு (பொருள் தொடர்பு) இல்லாமல், கற்றல் தரவின் அடிப்படையில் மட்டும் தோன்றுகிறது.

('NLP', 0.1085): **NLP** போன்ற வார்த்தை காரணமாக, மாதிரி மிகச்சிறிய data-வில் இருந்து இதுபோன்ற தவறான தொடர்புகளை வெளிப்படுத்துகிறது.

('love', 0.0347) மற்றும் பிற வார்த்தைகள்: இது மிகவும் குறைந்த ஒத்தபாடுடையது, எனவே **king**-க்கு இது பொருத்தமற்றது எனவும் கூறலாம்.

இங்கே பயிற்சி செய்த தரவுத்தொகுப்பு (sentences) மிகச் சிறியது, மற்றும் “king” வார்த்தையை சரியான “context”-இல் அதிகமில்லை. ஆகவே, பல வார்த்தைகள் அவற்றின் உண்மையான semantic தொடர்புகளைச் சரியாக பிரதிபலிக்க முடியவில்லை.

தீர்வு: பெரிய மற்றும் பலதரப்பட்ட dataset-ல் மாடலை train செய்ய வேண்டும். “king” போன்ற வார்த்தைகள் ஏற்ற semantic context-இல் (e.g., royalty, leadership) அடிக்கடி தோன்ற வேண்டும்.

முடிவில், Word2Vec ஒரு சக்திவாய்ந்த கருவி என்பதை விட, அது ஒரு மாபெரும் திருப்புமுனை.

அதன் இரண்டு புத்திசாலித்தனமான முறைகளான CBOW (புதிரை விடுவிக்கும் துப்பறிவாளன்) மற்றும் Skip-Gram (கற்பனை வளம் மிக்க கதைசொல்லி) மூலம், அது வார்த்தைகளுக்கு இடையிலான கண்ணுக்குத் தெரியாத உறவுகளை, கணிதத்தின் தெளிவான மொழிக்குக் கொண்டுவந்தது. இது, இயந்திர கற்றல் மாதிரிகளின் துல்லியத்தை மேம்படுத்தியது மட்டுமல்லாமல், கணினிகள் மொழியின் அர்த்தத்தைப் புரிந்துகொள்ள முடியும் என்பதற்கான முதல் உறுதியான சான்றாக அமைந்தது.

Word2Vec வரைந்த அந்த முதல் வார்த்தைப் பிரபஞ்சத்தின் வரைபடத்தின் மீதுதான், இன்றைய நவீன NLP மாளிகை கட்டப்பட்டுள்ளது.

3.4. Sequence-to-Sequence (Seq2Seq): Seq2Seq மாடல்கள் - ஒரு அறிமுகம்

வார்த்தைகளின் அகராதியை செயற்கை நுண்ணறிவு முழுமையாகக் கற்றுக்கொண்டது. ஆனால், வார்த்தைகள் வெறும் கருவிகளே. மனிதர்களாகிய நாம் வார்த்தைகளால் சிந்திப்பதில்லை; எண்ணங்களால் சிந்திக்கிறோம்.

ஒரு முழுமையான சிந்தனையின் சாரத்தை, அதன் உணர்வை, அதன் நோக்கத்தை, வார்த்தைகளின் கோர்வையிலிருந்து ஒரு இயந்திரம் எப்படிப் பிரித்தெடுக்கும்? அதைவிட முக்கியமாக, அந்தச் சாரத்தைப் புரிந்துகொண்டு, அதற்கு பதிலளிக்கும் வகையில் ஒரு புதிய சிந்தனையை எப்படி உருவாக்கும்?

இந்தச் சவாலை, அதாவது எண்ணங்களைக் கையாளும் கலையை, AI ஒரு புத்திசாலித்தனமான கூட்டணியின் மூலம் சாத்தியமாக்கியது. அதுதான் **என்கோடர் (Encoder)** மற்றும் **டிகோடர் (Decoder)** என்ற இரண்டு நிபுணர்களின் கூட்டணி.

இதை, இரண்டு மொழிபெயர்ப்பாளர்களைக் கொண்ட ஒரு குழுவாகக் கற்பனை செய்யுங்கள்:

1. **என்கோடர் - ஆழமாகக் கவனிக்கும் நிபுணர்:** 🧠 முதல் நிபுணரான என்கோடர், ஒரு மொழியில் (உதாரணமாக, தமிழ்) உள்ள ஒரு முழு வாக்கியத்தையும் கவனமாகக் கேட்கிறது. அது வார்த்தைகளை மட்டுமல்ல, அதன் இலக்கணம், சூழல், மற்றும் உள்ளார்ந்த அர்த்தம் என அனைத்தையும் உள்வாங்கிக்கொண்டு, அந்த முழு சிந்தனையையும், மொழி இல்லாத ஒரு தூய எண்ணமாக, அதாவது எண்களைக் கொண்ட ஒரு "**சிந்தனை வெக்டராக" (Thought Vector)** சுருக்குகிறது.
2. **டிகோடர் - அழகாக வெளிப்படுத்தும் நிபுணர்:** ✍️ இரண்டாவது நிபுணரான டிகோடர், என்கோடர் உருவாக்கிய அந்த ஒற்றைச் "சிந்தனை வெக்டரை" எடுத்துக்கொள்கிறது. அந்த மையக் கருத்தை அடிப்படையாகக் கொண்டு, வார்த்தைக்கு வார்த்தை கோர்த்து, இலக்கணப்படி சரியான, அர்த்தமுள்ள ஒரு புதிய வாக்கியத்தை உருவாக்குகிறது.

இந்த என்கோடர்-டிகோடர் கூட்டணியே, இன்று நாம் வியக்கும் பல தொழில்நுட்பங்களின் இதயமாகச் செயல்படுகிறது. ஒரு மொழியில் உள்ளதை மற்றொரு மொழிக்கு நொடியில் மொழிபெயர்க்கும் இயந்திர மொழிபெயர்ப்பு (Machine Translation), ஒரு நீண்ட கட்டுரையைச் சில வரிகளில் சுருக்கும் உரை சுருக்கம் (Text Summarization), நீங்கள் கேட்கும் கேள்விகளுக்குப் பதிலளிக்கும் அரட்டைப் பெட்டிகள் (Chatbots) என அனைத்தின் பின்னணியிலும் இந்த இரு நிபுணர்களின் உரையாடலே நிகழ்கிறது.

வாருங்கள், இந்த என்கோடர் மற்றும் டிகோடர் எப்படி இணைந்து செயல்பட்டு, மனித மொழிக்கும் இயந்திரத்தின் தர்க்கத்திற்கும் இடையே ஒரு பாலத்தை அமைக்கின்றன என்பதை விரிவாகப் பார்க்கலாம்.

என்கோடர் (Encoder)

1. டோக்கனைசேஷன் (Tokenization): வார்த்தைகளை சிறு பகுதிகளாகப் பிரித்தல்

என்கோடர் என்ற ஆழமாகக் கவனிக்கும் நிபுணர், ஒரு தமிழ் வாக்கியத்தை எப்படிப் புரிந்துகொள்கிறது? அது, வாக்கியத்தை ஒரு முழுமையான ஓவியமாகப் பார்க்காது. மாறாக, ஒரு மொழியியல் தொல்பொருள் ஆய்வாளனைப் (Linguistic Archaeologist) போல, அந்த வாக்கியத்தின் ஒவ்வொரு நுணுக்கத்தையும் ஆராய்கிறது. உதாரணமாக, "**அகர முதல எழுத்தெல்லாம்**" என்ற திருக்குறளின் முதல் வரியை அது எதிர்கொள்ளும்போது, அதை அப்படியே எடுத்துக்கொள்வதில்லை.

மாறாக, அதன் அடிப்படைக் கூறுகளாக, அதாவது டோக்கன்களாகப் பிரிக்கிறது:

["அ", "க", "ர", " ", "மு", "த", "ல", " ", "எ", "ழு", "த்", "தெ", "ல்", "லா", "ம்"]

ஒவ்வொரு எழுத்தும், ஒவ்வொரு இடைவெளியும் கூட ஒரு தனித்தனி டோக்கனாக மாற்றப்படுகிறது. இந்த முதல் படியின் மூலமே, கவித்துவமான தமிழ் மொழி, கணினியின் தர்க்க உலகிற்குள் நுழையத் தயாராகிறது.

2. சொல்லகராதி (Vocabulary) உருவாக்குதல்: ஒவ்வொரு டோக்கனுக்கும் தனித்துவமான எண்

அந்த மொழியியல் தொல்பொருள் ஆய்வாளர் (என்கோடர்), வாக்கியத்தின் கூறுகளைத் தனித்தனியாகப் பிரித்தெடுத்த பிறகு, அடுத்த கட்டம் அவற்றுக்கு ஒரு ரகசிய அடையாளத்தைக் கொடுப்பது.

இது ஒரு **ரகசியக் குறியீட்டுப் புத்தகத்தை (Codebook)** அல்லது ஒரு சொல் அகராதியை (Vocabulary) உருவாக்குவதைப் போன்றது. அந்த அகராதியில் உள்ள ஒவ்வொரு தனித்துவமான டோக்கனுக்கும், ஒரு பிரத்யேக எண் அடையாளமாக வழங்கப்படுகிறது.

உதாரணமாக, அந்தக் குறியீட்டுப் புத்தகத்தில்:

உதாரணமாக:

- "அ" → 31
- "க" → 43
- "ர" → 59
- " " → 2
- "மு" → 57
- "த" → 69
- "ல" → 52
- "எ" → 61
- "ழு" → 37
- "த்" → 63
- "தெ" → 77
- "ல்" → 71
- "லா" → 66
- "ம்" → 88

இந்த எண் மாற்றுதலின் மூலம், கவித்துவமான தமிழ் வார்த்தைகள், இப்போது கணினியின் தர்க்க மொழிக்கு (Language of Logic) முழுமையாக மாற்றப்பட்டுவிட்டன. ஒவ்வொரு டோக்கனும் இப்போது ஒரு தனித்துவமான எண்ணாக, அடுத்தகட்ட செயலாக்கத்திற்குத் தயாராக உள்ளது.

3. எம்பெடிங் (Embedding): எண்களை வெக்டர்களாக மாற்றுதல்

அந்தக் குறியீட்டுப் புத்தகத்தில் உள்ள ஒவ்வொரு எண்ணும், ஒரு ஆழமான அர்த்தத்தைக் கொண்ட ஒரு திறவுகோல். அடுத்ததாக, **எம்பெடிங் (Embedding)** என்ற செயல்முறை மூலம், ஒவ்வொரு எண்ணும் ஒரு தனித்துவமான, பல பரிமாணங்களைக் கொண்ட வெக்டராக (Vector) மாற்றப்படுகிறது.

இது, ஒவ்வொரு டோக்கனுக்கும் ஒரு அடையாள அட்டை (ID Number) கொடுப்பதிலிருந்து, அதைப் பற்றிய ஒரு முழுமையான, விரிவான சுயவிவரத்தை (Profile) உருவாக்குவதைப் போன்றது.

உதாரணமாக, **"அகர முதல"** என்ற வார்த்தைகளின் எண் வடிவம் `[31, 43, 59, 2, 57, 69, 52]` என்று பார்த்தோம். இப்போது, எம்பெடிங் செயல்முறை ஒவ்வொரு எண்ணுக்கும் அதன் ஆழமான சுயவிவரத்தை வழங்குகிறது:

எண் அடையாளம் (ID)	வெக்டர் சுயவிவரம் (VECTOR PROFILE)
31 ("அ")	`[0.1, -0.5, 0.8, ...]`
43 ("க")	`[0.2, 0.3, -0.1, ...]`
59 ("ர")	`[-0.4, 0.6, 0.2, ...]`

இந்த எண்களின் நீண்ட வரிசைகள் (வெக்டர்கள்) வெறும் எண்கள் அல்ல; அவை ஒவ்வொரு டோக்கனின் ஆன்மா, அதன் குணம், மற்றும் மற்ற டோக்கன்களுடன் உள்ள உறவின் கணித வெளிப்பாடு. இந்த வெக்டர்களின் மூலமே, கணினி ஒரு டோக்கனின் நுணுக்கமான அர்த்தத்தைப் புரிந்துகொண்டு, தனது அடுத்தகட்டப் பயணத்தைத் தொடங்குகிறது.

4. ஹிட்டன் ஸ்டேட்கள் (Hidden States): சூழல் தகவலை நினைவில் கொள்ளுதல்

கணினி, இப்போது ஒவ்வொரு டோக்கனின் ஆழமான சுயவிவரத்தையும் (Embedding Vector) பெற்றுவிட்டது. ஆனால், தனித்தனி சுயவிவரங்கள் ஒரு கதையைச் சொல்லாது. அந்த டோக்கன்கள் ஒன்றிணைந்து உருவாக்கும் முழுமையான வாக்கியத்தின் ஓட்டத்தையும், அதன் திரண்ட அர்த்தத்தையும் கணினி எப்படிப் புரிந்துகொள்ளும்?

இங்கேதான், **ரிக்கரண்ட் நியூரல் நெட்வொர்க் (RNN)** என்ற தொடர்முறை நரம்பியல் வலைப்பின்னல், ஒரு கதைசொல்லியைப் போல உள்ளே நுழைகிறது.

RNN, ஒரு வாக்கியத்தில் உள்ள வெக்டர்களை ஒரே நேரத்தில் பார்ப்பதில்லை. மாறாக, ஒரு மனிதன் படிப்பது போல, ஒவ்வொன்றும் டோக்கனாக, வரிசைமாறாமல் படிக்கிறது. ஒவ்வொரு டோக்கனைப் படிக்கும்போதும், அது தனது நினைவகத்தை (memory) புதுப்பித்துக்கொண்டே செல்கிறது. இந்தத் தொடர்ந்து மாறிவரும் நினைவகத்திற்குத்தான் **ஹிட்டன் ஸ்டேட் (Hidden State)** என்று பெயர்.

உதாரணமாக, **"அகர முதல"** என்ற வரியை RNN படிக்கும்போது:

1. முதலில், அது **"அ"**-வைப் படிக்கிறது. அதன் நினைவகம் (Hidden State 1), இப்போது "அ"-வின் சாரத்தை வைத்திருக்கிறது.
2. அடுத்து, **"க"**-வைப் படிக்கிறது. அது "அ"-வை மறக்காமல், அதன் நினைவோடு "க"-வையும் இணைத்து, "அக" என்பதன் பொருளைப் பிரதிபலிக்கும் ஒரு புதிய, செறிவூட்டப்பட்ட நினைவகத்தை (Hidden State 2) உருவாக்குகிறது.
3. பிறகு, **"ர"**-வைப் படிக்கிறது. "அக"-வின் நினைவோடு "ர"-வையும் இணைத்து, "அகர" என்ற ஆழமான புரிதலை (Hidden State 3) அடைகிறது.

> - *"அ" → Hidden State 1 (ஆரம்ப நிலை)*
> - *"க" → Hidden State 2 (Hidden State 1 + "க") → அக*
> - *"ர" → Hidden State 3 (Hidden State 2 + "ர") → அகர*
> - *" " → Hidden State 4 (Hidden State 3 + Space) →அகர_*
> - *"மு" → Hidden State 5 (Hidden State 4 + "மு") → அகர _ மு*
> - *"த" → Hidden State 6 (Hidden State 5 + "த") → அகர _ முத*
> - *"ல" → Hidden State 7 (Hidden State 6 + "ல") →அகர _ முதல*

இப்படியே ஒவ்வொரு டோக்கனாகத் தொடர்ந்து, இறுதி டோக்கனான "ல"-வைப் படித்தவுடன், அந்த இறுதி ஹிட்டன் ஸ்டேட் (Hidden State 7), வெறும் "ல"-வின் அர்த்தத்தை மட்டும் கொண்டிருக்காது. மாறாக, அது **"அகர முதல"** என்ற முழுமையான சொற்றொடரின் ஒட்டுமொத்த ஆன்மாவையும், சூழலையும், சாரத்தையும் தனக்குள் அடக்கிய ஒரு ஒற்றை, சக்திவாய்ந்த வெக்டராக மாறியிருக்கும். இந்த இறுதி ஹிட்டன் ஸ்டேட்தான், நாம் முன்பு குறிப்பிட்ட **"சிந்தனை வெக்டர்" (Thought Vector)**.

5. கான்டெக்ஸ்ட் வெக்டர் (Context Vector): ஒட்டுமொத்த தகவலின் சுருக்கம்

என்கோடர் என்ற கதைசொல்லி, இப்போது ஒரு வாக்கியத்தை முழுமையாகப் படித்து முடித்துவிட்டது. அதன் இறுதி நினைவகம் (final hidden state), இப்போது ஒரு சாதாரண நினைவகம் அல்ல. அது, அந்த முழு வாக்கியத்தின் ஆன்மாவையும், சாரத்தையும் தனக்குள் அடக்கிய ஒரு ஒற்றை, சக்திவாய்ந்த வெக்டராக மாறியுள்ளது.

இந்த ஒற்றை வெக்டருக்குத்தான் **கான்டெக்ஸ்ட் வெக்டர் (Context Vector)** என்று பெயர்.

இது, ஒரு தொடர் ஓட்டப் பந்தயத்தில் (Relay Race), முதல் ஓட்டக்காரர் தனது முழு சக்தியையும் திரட்டி, அடுத்த ஓட்டக்காரரிடம் கொடுக்கும் ஒரு சிறிய கோலைப் (Baton) போன்றது. அந்த ஒற்றைக் கோலில், முதல் ஓட்டத்தின் வேகம், தாளம், மற்றும் வெற்றி பெறுவதற்கான உத்தி என அனைத்தும் அடங்கியிருக்கும். உதாரணமாக "அகர முதல" என்ற வாக்கியத்தின் கான்டெக்ஸ்ட் வெக்டர், Hidden State 7 ஆக இருக்கும். இது, முழு வாக்கியத்தின் அர்த்தத்தையும், சூழலையும் சுருக்கமாக கொண்டிருக்கும். இந்த கான்டெக்ஸ்ட் வெக்டர், டிகோடருக்கு முக்கியமான தகவலாகும்.

என்கோடரின் வேலை இங்கே முடிந்தது. இப்போது, அந்த "சிந்தனைக் கோலை" (Context Vector), டிகோடர் என்ற அடுத்த நிபுணரிடம் அது கொடுக்கிறது. வாருங்கள், அந்தச் சிந்தனையை டிகோடர் எப்படி மீண்டும் மொழியாக மாற்றுகிறது என்பதை விரிவாகப் பார்க்கலாம்.

டிகோடர் (Decoder) - எண்களை மீண்டும் சங்கத்தமிழ் பாடலாக மாற்றுதல்

டிகோடர் என்பது என்கோடரால் உருவாக்கப்பட்ட கான்டெக்ஸ்ட் வெக்டரைப் பயன்படுத்தி, மீண்டும் சங்கத்தமிழ் பாடல் வரிகளை உருவாக்கும் செயல்முறையாகும். இதுவும் பல படிகளைக் கொண்டது.

1. இனிஷியல் ஸ்டேட் (Initial State): தொடக்க நிலை

தொடர் ஓட்டப் பந்தயத்தில், இரண்டாவது ஓட்டக்காரரான **டிகோடர்**, என்கோடர் கொடுத்த அந்தச் சக்திவாய்ந்த "சிந்தனைக் கோலை" (கான்டெக்ஸ்ட் வெக்டர்) வாங்கிக்கொள்கிறது.

அந்தக் கோல்தான், டிகோடரின் தொடக்கப் புள்ளி. அதன் முதல் சிந்தனை, அதன் **ஆரம்ப நிலை (Initial State)** என்பது, வேறு எதுவும் இல்லை - என்கோடர் புரிந்துகொண்ட அந்த முழுமையான வாக்கியத்தின் சாராம்சம்தான்.

உதாரணமாக, "அகர முதல" என்ற வாக்கியத்தின் பயணத்தில், என்கோடரின் இறுதி நினைவகம் (Hidden State 7) தான், டிகோடரின் முதல் சிந்தனையாக மாறுகிறது. இந்த ஆரம்பத் தகவலை வைத்துக்கொண்டுதான், டிகோடர் தனது புதிய வாக்கியத்தை உருவாக்கும் பயணத்தைத் தொடங்குகிறது.

2. டோக்கன் ஜெனரேஷன் (Token Generation): வார்த்தைகளை உருவாக்குதல்

அந்தச் "சிந்தனைக் கோலை" (Context Vector) கையில் ஏந்திய டிகோடர், இப்போது ஒரு புதிய வாக்கியத்தை உருவாக்கும் தனது கலைப் பயணத்தைத் தொடங்குகிறது.

அது, முழு வாக்கியத்தையும் ஒரே நேரத்தில் உருவாக்குவதில்லை. மாறாக, ஒரு சிற்பி உளியை வைத்து, கல்லைச் செதுக்குவது போல, வார்த்தைக்கு வார்த்தை பார்த்துப் பார்த்து உருவாக்குகிறது.

முதல் வார்த்தை: முதலில், அது அந்த ஒட்டுமொத்த சிந்தனையை (Context Vector) மட்டும் பார்த்து, "இந்தக் கருத்தைத் தொடங்க மிகவும் பொருத்தமான முதல் வார்த்தை எது?" என்று கேட்கிறது. அதன் நியூரல் நெட்வொர்க், அகராதியில் உள்ள எல்லா வார்த்தைகளின் நிகழ்தகவையும் (Probability) கணக்கிட்டு, "அ" (எண் 31) என்பதற்கு அதிக வாய்ப்பிருப்பதாகச் சொல்லும்.

அடுத்த வார்த்தை: இப்போது, அது தனது முதல் வார்த்தையான "அ" மற்றும் அந்த ஒட்டுமொத்த சிந்தனை ஆகிய இரண்டையும் மனதில் வைத்து, "அ-வைத் தொடர்ந்து வரக்கூடிய அடுத்த சரியான வார்த்தை எது?" என்று கேட்கிறது. நிகழ்தகவின் அடிப்படையில், அது "க" (எண் 43) என்பதைத் தேர்ந்தெடுக்கும்.

இப்படியே, ஒவ்வொரு படியிலும், தான் முன்பு சொன்ன வார்த்தையையும், தனது மையக் கருத்தையும் அடிப்படையாகக் கொண்டு, அடுத்தடுத்த டோக்கன்களை அது ஒரு சங்கிலி போல உருவாக்கிக் கொண்டே செல்கிறது: `[31, 43, 59, 2, ...]`

3. அகராதியைப் பயன்படுத்தி மீண்டும் உரைக்கு மாற்றுதல்

டோக்கன் உருவாக்கம் எவ்வாறு நிகழ்கிறது?

```
itos = {2: ' ', 31: 'அ', 43: 'க', 59: 'ர', 57: 'மு', 69: 'த', 52: 'ல', 61: 'எ', 37: 'ழு', 63: 'த்', 77: 'ெ', 71: 'ல்', 66: 'ா'}
```

டிகோடர், இப்போது `[31, 43, 59, 2, ...]` என்ற எண் சங்கிலியை உருவாக்கிவிட்டது. ஆனால், இந்த எண்கள் கணினிக்கு மட்டுமே புரியும் ஒரு ரகசிய மொழி. இந்த ரகசிய மொழியை, மீண்டும் நமக்குப் புரியும் தமிழ் மொழியாக மாற்றுவது எப்படி?

இங்கேதான், நாம் முன்னர் பயன்படுத்திய **"ரகசியக் குறியீட்டுப் புத்தகம்" (Codebook)** மீண்டும் 役にவருகிறது. ஆனால், இந்த முறை அதைத் தலைகீழாகப் பயன்படுத்துகிறோம். அதாவது, ஒவ்வொரு எண்ணுக்கும் உரிய எழுத்து என்ன என்பதைத் தேடுகிறோம் (`itos` - integer to string).

டிகோடர், தான் உருவாக்கிய எண் சங்கிலியை எடுத்துக்கொள்கிறது: `[31, 43, 59, 2, 57, 69, 52]`

1. அது முதலில் **31** என்ற எண்ணைப் பார்க்கிறது. குறியீட்டுப் புத்தகம் சொல்கிறது: "இதன் அர்த்தம் '**அ**'."
2. அடுத்து **43**-ஐப் பார்க்கிறது. புத்தகம் சொல்கிறது: "இதன் அர்த்தம் '**க**'."
3. பிறகு **59**-ஐப் பார்க்கிறது. அதன் அர்த்தம் '**ர**'.

இப்படியே ஒவ்வொரு எண்ணாக அதன் தமிழ் எழுத்தைக் கண்டறிந்து, இறுதியாக அவற்றை ஒன்றாக இணைக்கிறது.

அந்த ரகசிய எண் சங்கிலி, இப்போது அனைவருக்கும் புரியும் வகையில் ஒரு அழகான தமிழ் சொற்றொடராக உருமாறுகிறது: **"அகர முதல"**.

இப்படித்தான், ஒரு சிந்தனை, எண்களாக மாறி, மீண்டும் ஒரு புதிய சிந்தனையாக, மொழியாக மறுபிறவி எடுக்கிறது. இந்த என்கோடர்-டிகோடர் கூட்டணியின் மூலமே, இயந்திரங்கள் நம்முடன் உரையாடுகின்றன.

இப்போது, நமக்குக் கொடுக்கப்பட்ட எண்களின் பட்டியலை எடுத்துக்கொள்வோம். பட்டியலில் உள்ள ஒவ்வொரு எண்ணையும் எடுத்துக்கொண்டு, அகராதியில் அதற்குரிய எழுத்தைப் பார்ப்போம். இந்த எழுத்துக்களை ஒன்றாக இணைத்தால், நமக்கு டெக்ஸ்ட் கிடைக்கும்.

4. டிகோடரின் செயல்பாடு

டிகோடர் எப்படி ஒரு சீரான, அர்த்தமுள்ள வாக்கியத்தை உருவாக்குகிறது? அதன் ரகசியம், அதன் **ஆட்டோரிக்ரஸிவ் (Autoregressive)** இயல்பில் உள்ளது. ஆட்டோரிக்ரஸிவ் என்பது, **"தன்னிலிருந்தே உருவாக்குவது"** என்று பொருள். ஒரு கவிஞர், ஒரு வரியை எழுதிய பிறகு, அடுத்த வரியை அந்த வரியின் ஓட்டத்திலிருந்தே உருவாக்குவதைப் போன்றது இது.

1. முதலில், டிகோடர் "**அ**" என்ற டோக்கனை உருவாக்குகிறது.

2. அடுத்த டோக்கனை உருவாக்க, அது தான் இப்போது உருவாக்கிய "**அ**"-வையே மீண்டும் உள்ளீடாக எடுத்துக்கொள்கிறது. 'அ'-வைத் தொடர்ந்து என்ன வர வேண்டும் என்று யோசித்து, "**க**"-வை உருவாக்குகிறது.
3. அடுத்து, அது "**அக**" என்ற தொடரை உள்ளீடாக எடுத்துக்கொண்டு, "**ர**"-வை உருவாக்குகிறது.

இந்தத் தொடர்ச்சியான, சங்கிலித் தொடர் போன்ற செயல்முறை, வாக்கியம் முடியும் வரை நிகழ்கிறது. ஒவ்வொரு புதிய வார்த்தையும், அதன் முந்தைய வார்த்தைகளின் தோளில் நின்று பிறப்பதால், டிகோடர் உருவாக்கும் வாக்கியம் இலக்கணப் பிழையின்றி, ஒரு சீரான ஓட்டத்துடன் அமைகிறது.

5. டிகோடரின் முடிவு

டிகோடர், ஒரு கவிஞனைப் போல, வார்த்தைக்கு வார்த்தை கோர்த்து, தனது புதிய வாக்கியத்தை உருவாக்கிக் கொண்டே செல்கிறது. அது எப்போது நிறுத்தும்?

அது "EOS" (End of Sequence) என்ற ஒரு சிறப்பு டோக்கனை உருவாக்கும் வரை, அதன் வார்த்தை உருவாக்கம் தொடரும். இந்த 'EOS' டோக்கன்தான், "என் சிந்தனை முழுமை பெற்றுவிட்டது, முற்றுப்புள்ளி" என்று அந்த இயந்திரம் வைக்கும் இறுதி முத்திரை. அந்த இறுதி டோக்கனுடன், டிகோடரின் கலைப் பயணம் நிறைவடைகிறது.

என்கோடர்-டிகோடர் என்ற கூட்டணியின் குறைபாடுகள்

என்கோடர்-டிகோடர் என்ற கூட்டணி, ஒரு சிந்தனையை மொழியிலிருந்து எண்ணாகவும், மீண்டும் எண்ணிலிருந்து மொழியாகவும் மாற்றும் ஒரு மாயாஜாலத்தைச் செய்தது. ஆனால், அந்த மாயாஜாலத்திற்கு ஒரு வரம்பு இருந்தது. ஒரு முக்கியமான, உள்ளார்ந்த குறைபாடு.

அந்தக் குறைபாட்டைப் புரிந்துகொள்ள, என்கோடரை ஒரு நீண்ட திரைப்படத்தைப் பார்க்கும் ஒருவராகக் கற்பனை செய்யுங்கள். படம் முடிந்ததும், அந்த மூன்று மணி நேரத் திரைப்படத்தின் கதை, திருப்பங்கள், உணர்வுகள் மற்றும் அத்தனை கதாபாத்திரங்களின் நுணுக்கங்கள் என அனைத்தையும், ஒரே ஒரு வாக்கியத்தில் சுருக்கி, ஒரு சிறிய துண்டுச் சீட்டில் எழுதி, டிகோடர் என்ற அடுத்தவரிடம் கொடுக்க வேண்டும்.

இது சாத்தியமா?

ஒரு சிறிய குறும்படமாக (சிறிய வாக்கியம்) இருந்தால், ஒருவேளை சாத்தியமாகலாம். ஆனால், ஒரு நீண்ட காவியத் திரைப்படமாக (நீண்ட வாக்கியம்) இருந்தால், எண்ணற்ற முக்கியமான தகவல்கள் அந்த ஒற்றைத் துண்டுச் சீட்டில் நிச்சயம் தொலைந்து போகும்.

இதுதான் **Seq2Seq** கட்டமைப்பின் மிகப்பெரிய பலவீனம். அது, ஒரு நீண்ட வாக்கியத்தின் முழு ஆன்மாவையும், ஒரே ஒரு கான்டெக்ஸ்ட் வெக்டருக்குள் (அந்தத் துண்டுச் சீட்டுக்குள்) சுருக்க முயற்சித்தது. இந்தத் **தகவல் சுருக்கத்தின் இடர்ப்பாடு (Information Bottleneck)**, நீண்ட வாக்கியங்களைக் கையாளும்போது, மாதிரியின் செயல்திறனைக் கடுமையாகப் பாதித்தது. மேலும், RNN-களின் தொடர்முறை இயல்பால், ஒரு நீண்ட வாக்கியத்தின் தொடக்கத்தில் உள்ள தகவல்கள், இறுதிக்கு வரும்போது அதன் நினைவிலிருந்து மங்கிவிடும் (Vanishing Gradient) பிரச்சனையும் இருந்தது.

தீர்வு: கவனத்தின் சக்தி (The Power of Attention)

இந்தக் குறைபாடுகளைத் தகர்த்தெறிய, ஒரு புதிய, புரட்சிகரமான கட்டமைப்பு பிறந்தது. அதுதான் **டிரான்ஸ்ஃபார்மர் (Transformer)**.

டிரான்ஸ்ஃபார்மர், அந்தத் "துண்டுச் சீட்டு" முறையை முற்றிலுமாக ஒழித்தது. அதற்குப் பதிலாக, அது டிகோடருக்கு ஒரு புதிய, மாயாஜால சக்தியைக் கொடுத்தது. அதுதான் **கவனத்தின் நுட்பம் (Attention Mechanism)**.

இதை இப்படிப் புரிந்துகொள்ளலாம்:

பழைய டிகோடர், கண்களை மூடிக்கொண்டு, என்கோடர் சொன்ன அந்த ஒற்றை வாக்கியத்தை (கான்டெக்ஸ்ட் வெக்டர்) மட்டும் நினைவில் வைத்துக்கொண்டு மொழிபெயர்க்க முயற்சித்தது. ஆனால், டிரான்ஸ்ஃபார்மரின் புதிய டிகோடர், ஒரு திறமையான மொழிபெயர்ப்பாளனைப் போல, மூல வாக்கியத்தை முழுமையாகத் தன் முன் வைத்துக்கொண்டு செயல்படுகிறது. அது, ஒவ்வொரு புதிய வார்த்தையை உருவாக்கும்போதும், மூல வாக்கியத்தின் எல்லா வார்த்தைகளையும் ஒருமுறை திரும்பிப் பார்த்து, "இந்த நேரத்தில், எந்த வார்த்தை மிக முக்கியம்?" என்று **கவனம்** செலுத்துகிறது.

"நான் மழை பெய்தால் வரமாட்டேன்" என்று மொழிபெயர்க்கும்போது, "வரமாட்டேன்" என்ற சொல்லை உருவாக்கும் முன், அது "மழை" என்ற மூல வார்த்தையின் மீது தனது முழு கவனத்தையும் வைக்கும்.

இந்த "கவனம்" என்ற ஒற்றைப் புரட்சிகரமான யோசனை, தகவல் சுருக்கப் பிரச்சனையையும், நினைவிழப்புப் பிரச்சனையையும் ஒரே நேரத்தில் தீர்த்தது. இது, AI மாதிரிகள் மிக நீண்ட, சிக்கலான வாக்கியங்களையும், மிகத் துல்லியமாகக் கையாள வழிவகுத்து, செயற்கை நுண்ணறிவின் அடுத்த சகாப்தத்திற்கு அடித்தளமிட்டது.