

# Τεχνητή Νοημοσύνη

*17η διάλεξη (2025-26)*

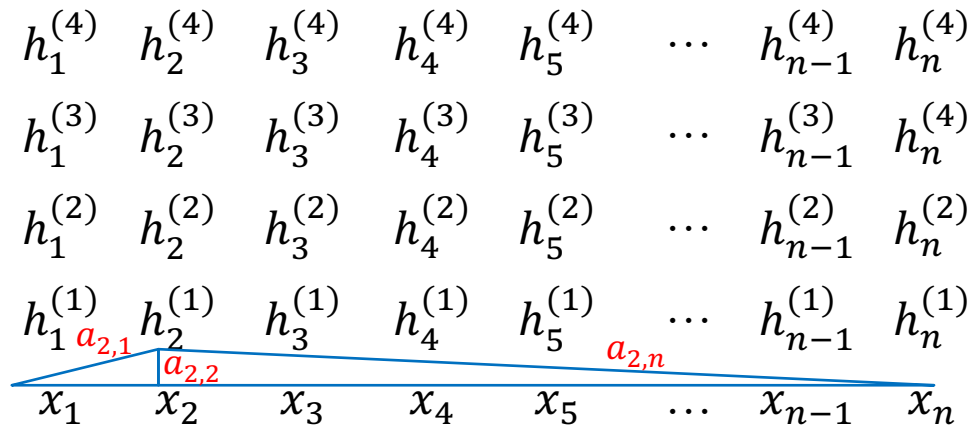
Ίων Ανδρουτσόπουλος

<http://www.aueb.gr/users/ion/>

# Contents

- Transformer encoders, BERT.
- Decoder-only Transformers, Large Language Models (LLMs), GPT-x.
- Prompting, supervised fine-tuning.
- Adding vision to LLMs (VLMs), LLaVA.

# Transformers for token classification



$$h_i^{(1)} = \text{MLP}^{(1)} \left( \sum_{r=1}^n a_{i,r}^{(1)} x_r \right) \in \mathbb{R}^m$$

All the layers produce  $m$ -dimensional (revised) word embeddings

4<sup>th</sup> attention layer

3<sup>rd</sup> attention layer

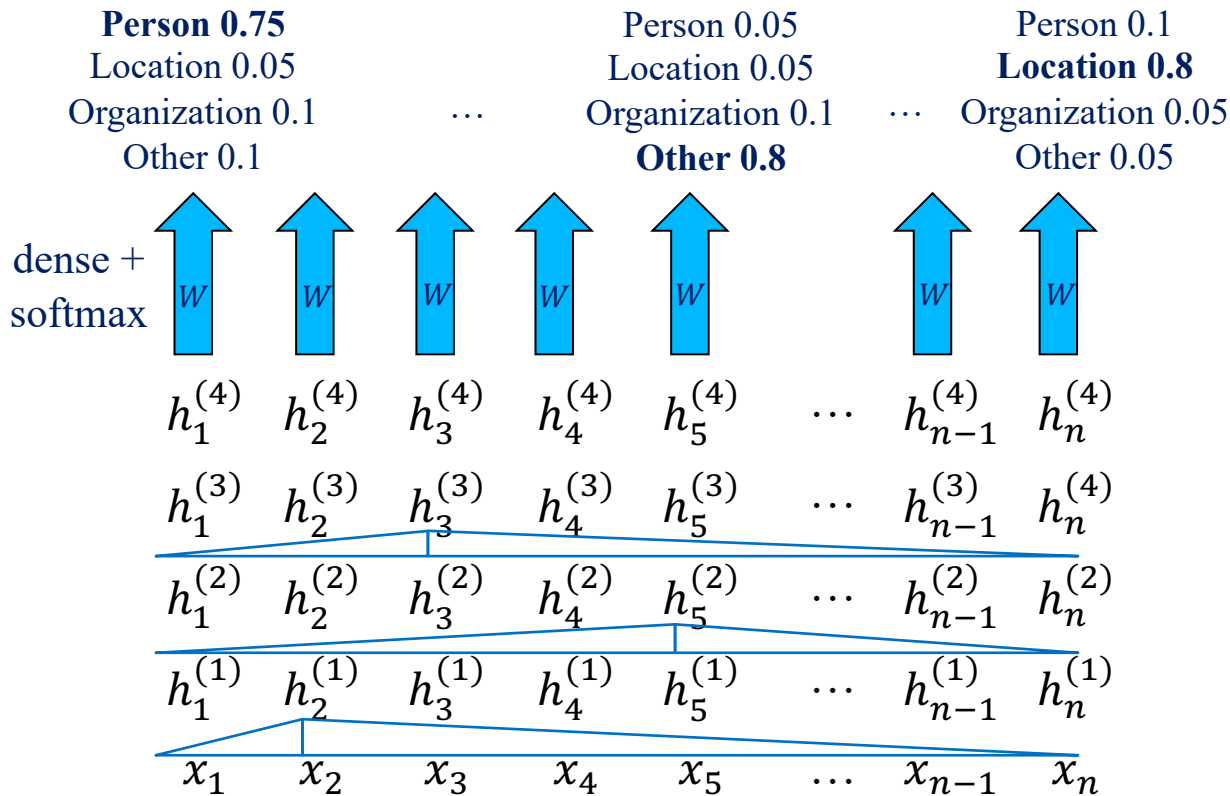
2<sup>nd</sup> attention layer

1<sup>st</sup> attention layer

Initial  $m$ -dimensional word embeddings

To produce the **revised embedding for the  $i$ -th word** of a text, we **sum all the original embeddings** of the words of the text, but **weighted by attention scores**. We then pass the sum through an **MLP** (or “FFNN”).

# Transformers for token classification



$$h_i^{(1)} = \text{MLP}^{(1)} \left( \sum_{r=1}^n a_{i,r}^{(1)} x_r \right) \in \mathbb{R}^m$$

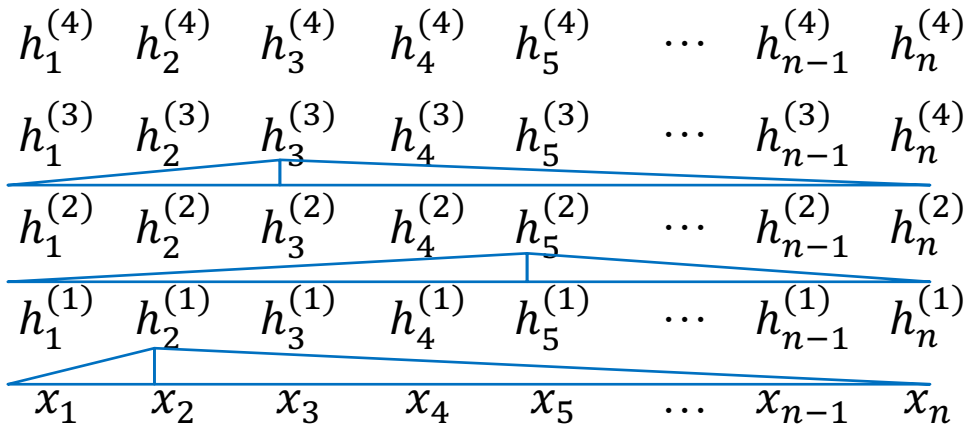
$$h_i^{(j)} = \text{MLP}^{(j)} \left( \sum_{r=1}^n a_{i,r}^{(j)} h_r^{(j-1)} \right) \in \mathbb{R}^m$$

# Transformers for text classification

$$h^{max} = \left\langle \max(h_{*,1}^{(4)}), \max(h_{*,2}^{(4)}), \dots, \max(h_{*,m}^{(4)}) \right\rangle^T \in \mathbb{R}^m$$

↑ **global max (or average) pooling** (max or average of each dimension)

Vector representing the entire text. We pass it through a **dense layer and softmax (or MLP)** to obtain a **probability per class**.



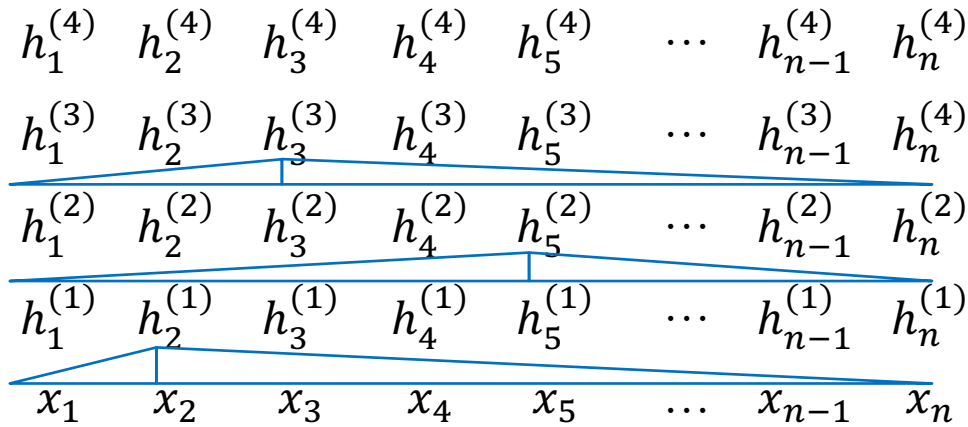
**Compare to the correct predictions** to obtain a **loss** (cross-entropy) and **backpropagate to adjust the weights of the entire net**, including the initial word (token) embeddings.

$$h_i^{(1)} = \text{MLP}^{(1)} \left( \sum_{r=1}^n a_{i,r}^{(1)} x_r \right) \in \mathbb{R}^m$$

$$h_i^{(j)} = \text{MLP}^{(j)} \left( \sum_{r=1}^n a_{i,r}^{(j)} h_r^{(j-1)} \right) \in \mathbb{R}^m$$

Without the **MLP** (or at least a dense layer), each dimension of  $h_i^{(j)}$  would only depend on the corresponding dimensions of the  $h_r^{(j-1)}$  vectors.

# Query-Key-Value self-attention



All the vectors of a layer can be computed in **parallel**.

Each layer is an “**encoder block**”. It has its own  $W^{Q,(j)}$ ,  $W^{K,(j)}$ ,  $W^{V,(j)}$  matrices, called an “**attention head**”.

$W^{Q,(j)}$ ,  $W^{K,(j)}$ ,  $W^{V,(j)}$  are randomly initialized and learned via backpropagation.

$$h_i^{(1)} = \text{MLP}^{(1)} \left( \sum_{r=1}^n a_{i,r}^{(1)} v_r^{(1)} \right) =$$

value vector for position  $r$

$$= \text{MLP}^{(1)} \left( \sum_{r=1}^n \text{softmax} \left( q_i^{(1)T} k_r^{(1)} \right) v_r^{(1)} \right) \in \mathbb{R}^{m \times 1}$$

query vector for position  $i$   
key vector for position  $r$

$$q_i^{(1)} = W^{Q,(1)} x_i$$

$$k_r^{(1)} = W^{K,(1)} x_r$$

$$v_r^{(1)} = W^{V,(1)} x_r$$

$$h_i^{(j)} = \text{MLP}^{(j)} \left( \sum_{r=1}^n a_{i,r}^{(j)} v_r^{(j)} \right) =$$

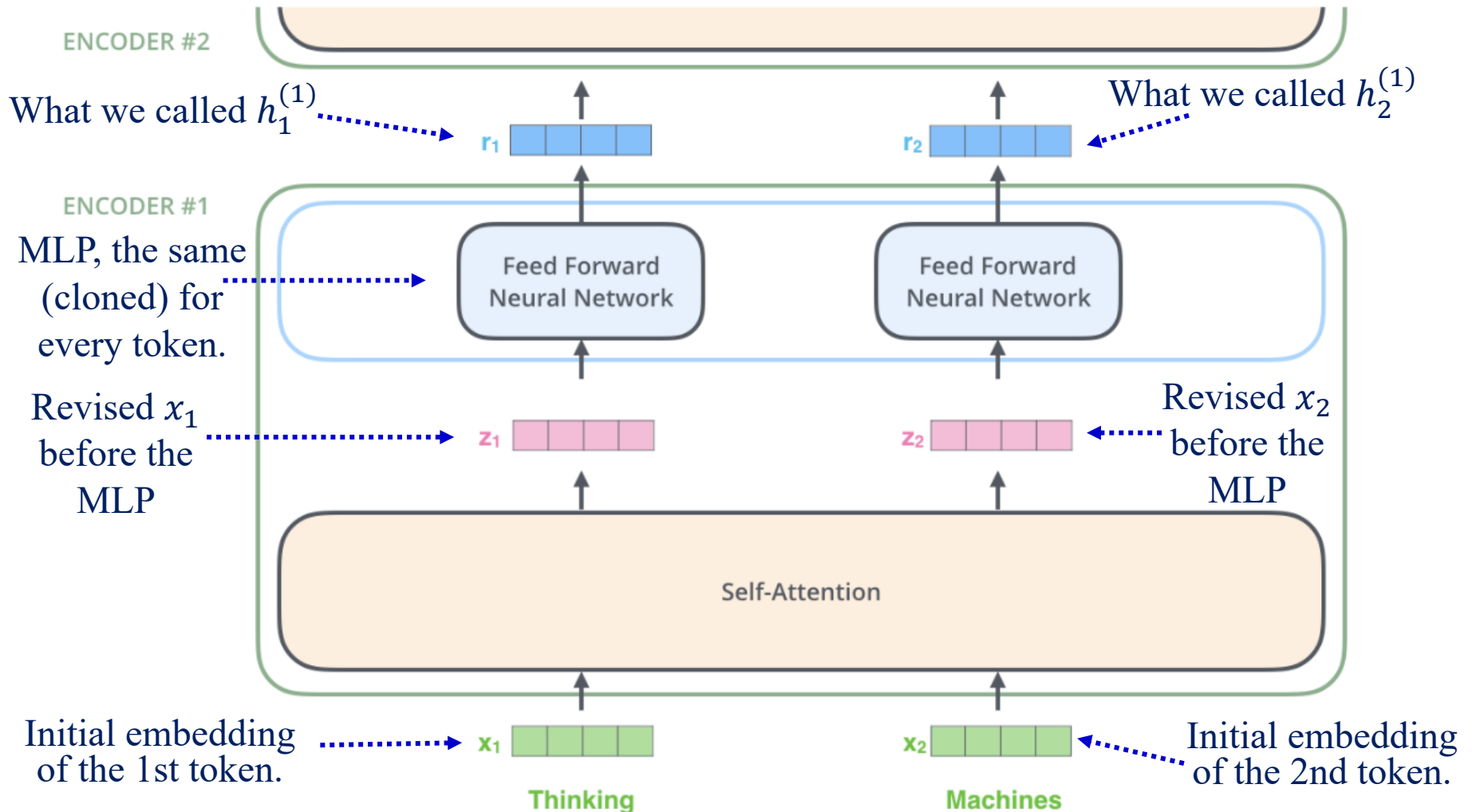
$$= \text{MLP}^{(j)} \left( \sum_{r=1}^n \text{softmax} \left( q_i^{(j)T} k_r^{(j)} \right) v_r^{(j)} \right) \in \mathbb{R}^{m \times 1}$$

$$q_i^{(j)} = W^{Q,(j)} h_i^{(j-1)}$$

$$k_r^{(j)} = W^{K,(j)} h_r^{(j-1)}$$

$$v_r^{(j)} = W^{V,(j)} h_r^{(j-1)}$$

# Stacking Transformer Encoders



Figures from J. Alammari's "The Illustrated Transformer" (<https://jalammari.github.io/illustrated-transformer/>). Transformers paper: Vaswani et al., "Attention is All You Need", 2017 (<https://arxiv.org/abs/1706.03762>).

# Query-Key-Value attention via matrices

$$X \times W^Q = Q$$

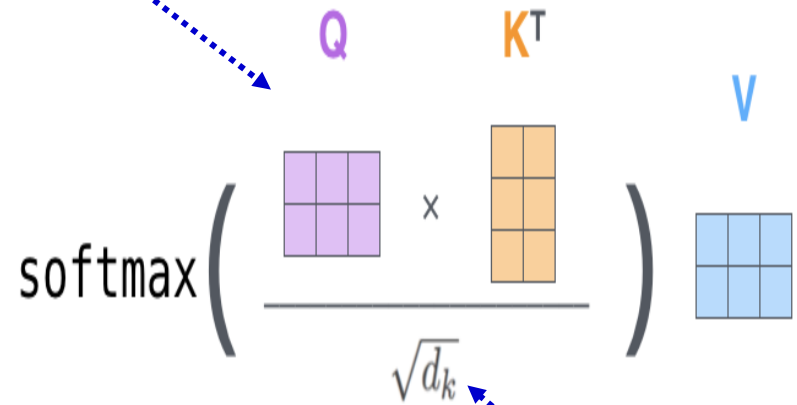

$$X \times W^K = K$$


$$X \times W^V = V$$

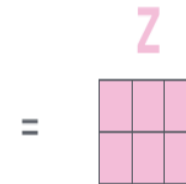

Here vectors are shown as rows, hence they need to be **left-multiplied** with the matrices.

We compute **all the  $q_i$  vectors** of a layer (block) using a **single matrix multiplication**. Similarly for all the  $k_r$  and  $v_r$  vectors of a layer.

We also compute **all the attention scores  $a_{i,r}$**  of a layer and all the revised vectors (before the MLP) using **matrix multiplications**.

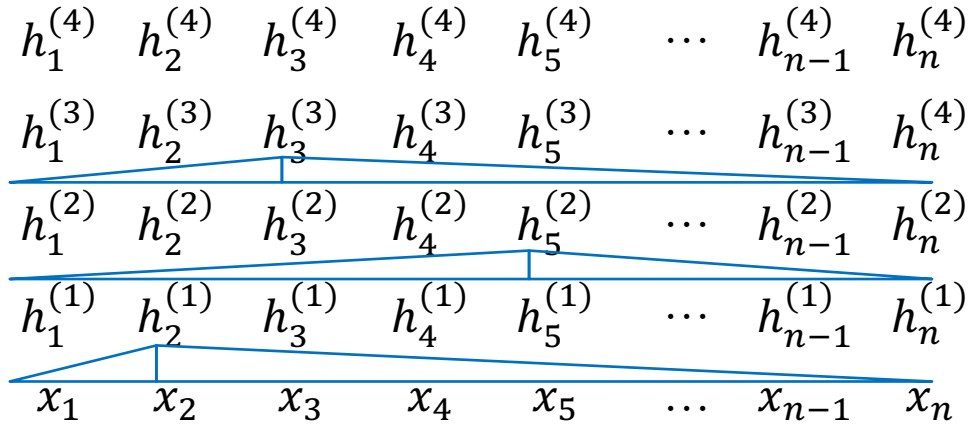
$$\text{softmax} \left( \frac{Q \times K^T}{\sqrt{d_k}} \right) \times V$$


$d_k$  is the dimensionality of the  $k_r$  and  $v_r$  vectors.

$$= Z$$


Figures from J. Alammari's "The Illustrated Transformer"  
(<https://jalammari.github.io/illustrated-transformer/>). Transformers paper: Vaswani et al.,  
"Attention is All You Need", 2017 (<https://arxiv.org/abs/1706.03762>).

# Adding residual connections



$$h_i^{(1)} = \text{MLP}^{(1)} \left( \sum_{r=1}^n a_{i,r}^{(1)} v_r^{(1)} \right) + x_i =$$

$$q_i^{(1)} = W^{Q,(1)} x_i$$

$$k_r^{(1)} = W^{K,(1)} x_r$$

$$v_r^{(1)} = W^{V,(1)} x_r$$

$$= \text{MLP}^{(1)} \left( \sum_{r=1}^n \text{softmax} \left( q_i^{(1)T} k_r^{(1)} \right) v_r^{(1)} \right) + x_i$$

$$h_i^{(j)} = \text{MLP}^{(j)} \left( \sum_{r=1}^n a_{i,r}^{(j)} v_r^{(j)} \right) + h_i^{(j-1)} =$$

$$q_i^{(j)} = W^{Q,(j)} h_i^{(j-1)}$$

$$k_r^{(j)} = W^{K,(j)} h_r^{(j-1)}$$

$$v_r^{(j)} = W^{V,(j)} h_r^{(j-1)}$$

$$= \text{MLP}^{(j)} \left( \sum_{r=1}^n \text{softmax} \left( q_i^{(j)T} k_r^{(j)} \right) v_r^{(j)} \right) + h_i^{(j-1)}$$

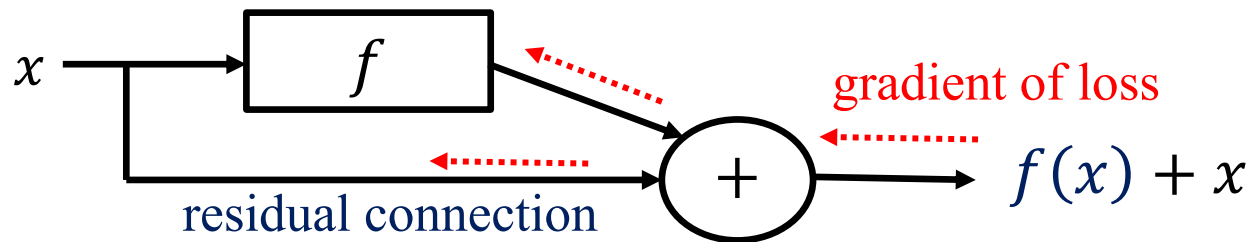
# Residual connections

- Given a **block** (part of a neural network) that would **normally compute**:

$$y = f(x)$$

where  $x, y$  are vectors, we **add its input to its output**.

$$y = f(x) + x$$



- During **backpropagation**, the **addition** gate just **copies the gradient** from its output to its inputs (lecture 13).
- If the **gradient vanishes** in the  **$f$  block**, it will reach the previous blocks (that produced  $x$ ) via the residual.

# Residual connections – continued

- Residual connections **allow stacking** more layers/blocks **without vanishing gradients**.
- They also give the network the **option not to use  $f$**  (by learning weights that always produce  $f(x) = 0$ ), but still **pass on to following blocks the information  $x$**  that a previous block computed.
- **Residuals** were first used (and are still used) in **CNNs**, but they are now also used in **RNNs** and **Transformers**.

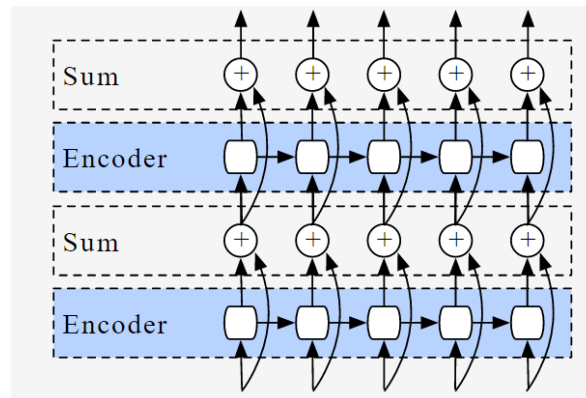


Image from Stephen Merity's  
[http://smerity.com/articles/2016/google\\_nmt\\_arch.html](http://smerity.com/articles/2016/google_nmt_arch.html)

# Multiple attention heads

Because of the softmax, **each attention head**  $W, K, V$  mostly considers only one token. So, let's use **multiple attention heads**.

1) This is our input sentence\*

2) We embed each word\*

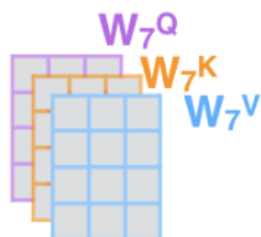
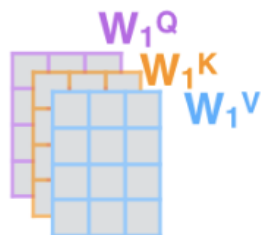
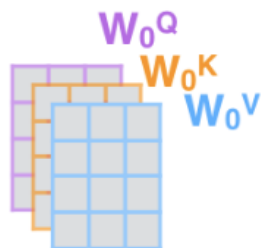
Thinking  
Machines



\* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



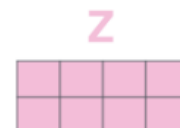
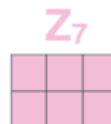
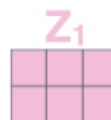
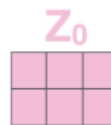
3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices



4) Calculate attention using the resulting  $Q/K/V$  matrices



5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer



$W^O$  is useful even if the concatenated  $Z_0, \dots, Z_7$  already have the right dimensions, to allow **combinations of features from different attention heads**.

Figures from J. Alammari's "The Illustrated Transformer"  
(<https://jalammari.github.io/illustrated-transformer/>). Transformers paper: Vaswani et al.,  
"Attention is All You Need", 2017 (<https://arxiv.org/abs/1706.03762>).

# Positional encodings



**Positional encodings** needed to capture the **word order/positions**.

- **Without them**, Transformers are **unaware of word order**.
- In the simplest case, we use **position embeddings**.
  - Embedding of **position 1**, embedding of **position 2** etc.
  - Initialized randomly, **learned** during backpropagation.
- **Sinusoid functions** used to produce them in the **original paper**.
- **RoPE** positional embeddings are **very popular**.
  - See as optional material <https://huggingface.co/blog/RDTvlokkip/when-ai-finally-learns-where-it-is>, <https://arxiv.org/abs/2104.09864>.

Figures from J. Alammam's "The Illustrated Transformer" (<https://jalammar.github.io/illustrated-transformer/>).  
Transformers paper: Vaswani et al., "Attention is All You Need", 2017 (<https://arxiv.org/abs/1706.03762>).

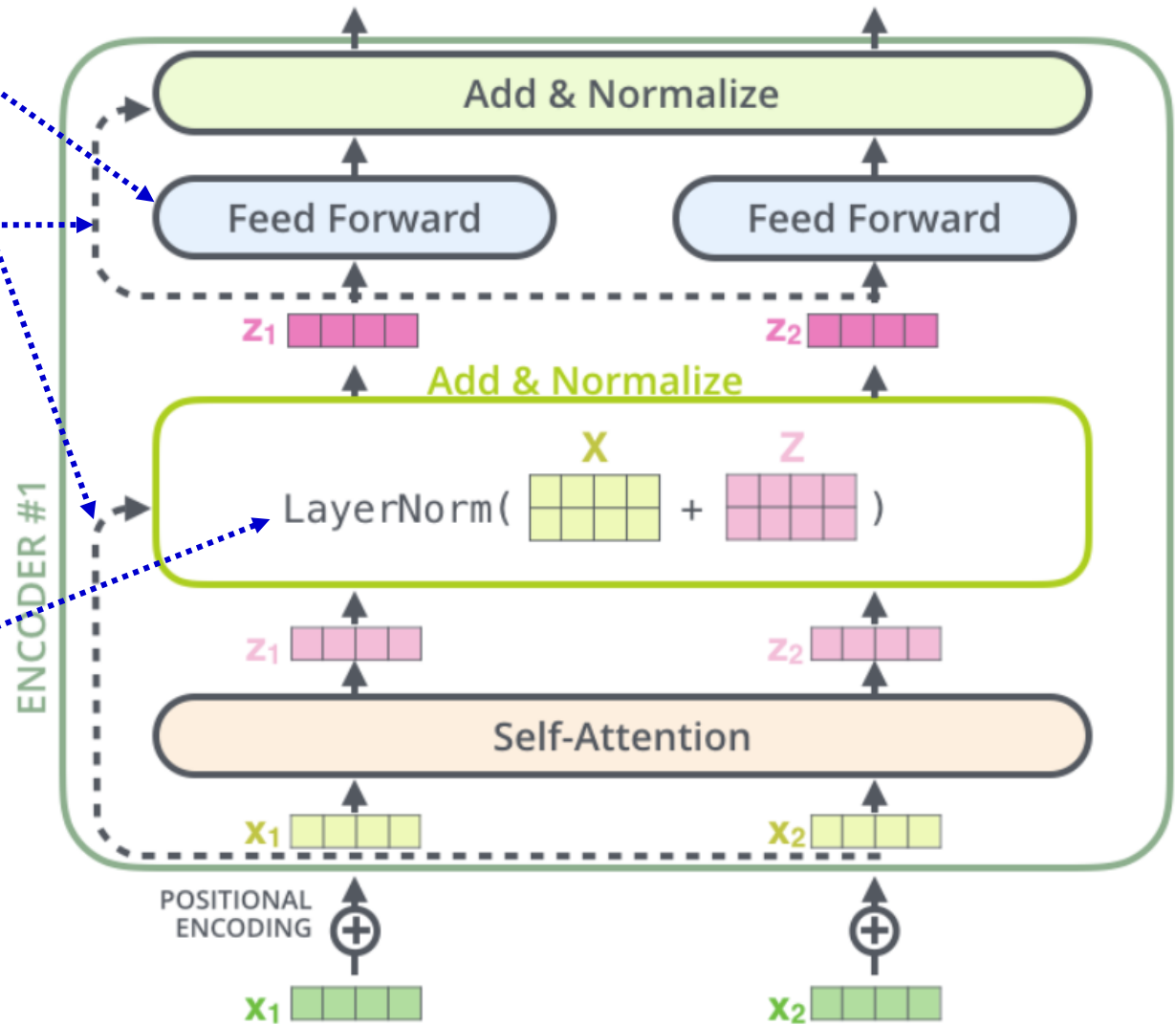
# Complete Transformer encoder block

“Feed Forward”: the **same MLP** at all word positions

“Add”: **residual connections**

**Layer Normalization** (Lecture 14). Here, we subtract from each cell  $(X + Z)_{r,c}$  of  $(X+Z)$  the mean  $\mu_r$  of its row, divide by the std. dev  $\sigma_r$  of the row, and multiply by a learned column-specific parameter  $g_c$ .

**Dropout** applied to the output of the self-attention and feed forward sublayers (before adding the residual and normalizing), inside the feed forward net, and after adding positional embeddings.



Figures from J. Alammari's "The Illustrated Transformer" (<https://jalammari.github.io/illustrated-transformer/>). Transformers paper: Vaswani et al., "Attention is All You Need", 2017 (<https://arxiv.org/abs/1706.03762>).

# BERT – Pretraining to predict masked words

Use the output of the masked word's position to predict the masked word

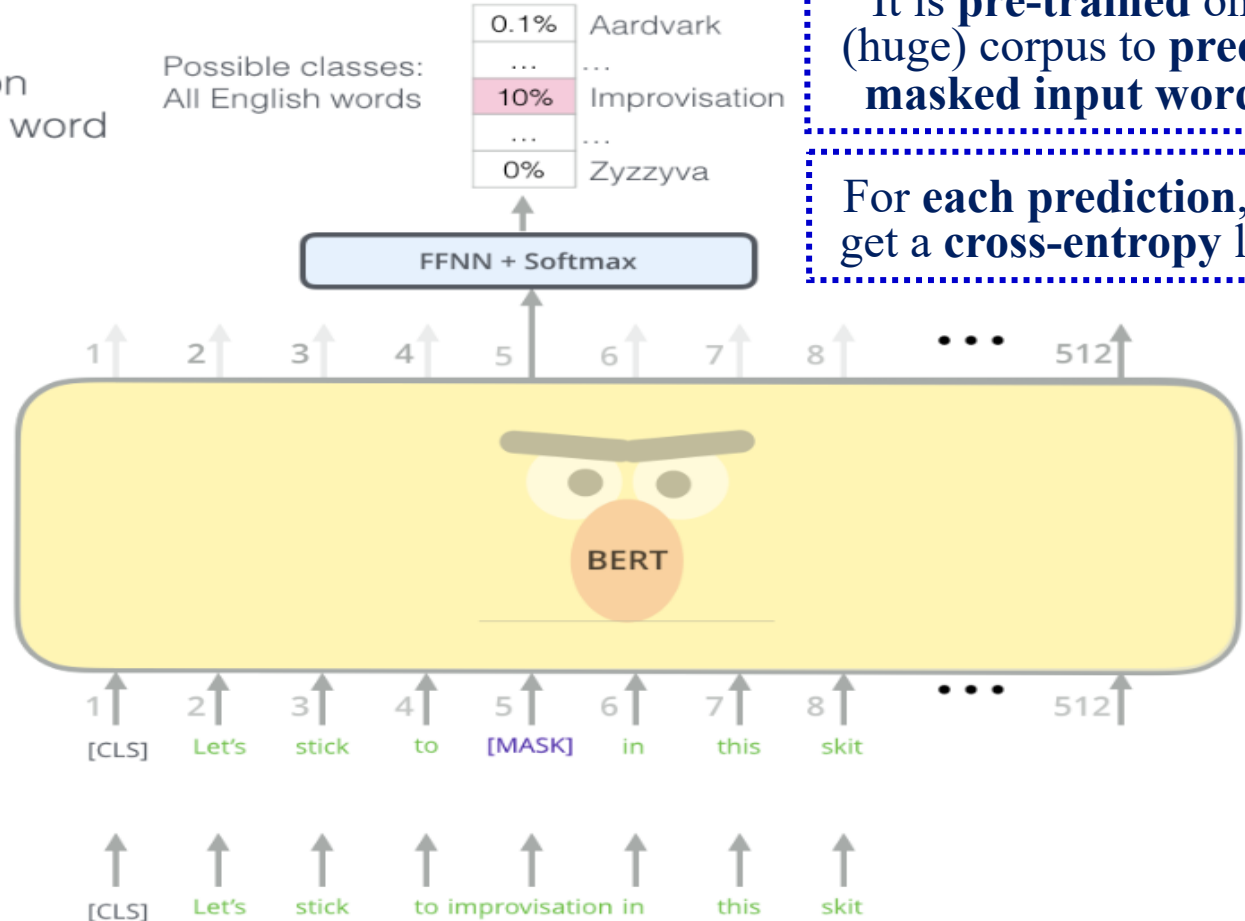
BERT uses **stacked Transformer encoders** (instead of RNNs) to turn each **sequence of input embeddings** to a **sequence of context aware embeddings**.

Possible classes:  
All English words

0.1%	Aardvark
...	...
10%	Improvisation
...	...
0%	Zyzyva

It is **pre-trained** on a (huge) corpus to **predict masked input words**.

For each prediction, we get a **cross-entropy loss**.



Randomly mask  
15% of tokens

Input

BERT's clever language modeling task masks 15% of words in the input and asks the model to predict the missing word.

Figures from J. Alammari's "The Illustrated BERT, ELMo, and co." (<http://jalammari.github.io/illustrated-bert/>). BERT paper: Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 2018 (<https://arxiv.org/abs/1810.04805>).

# BERT – Pretraining to predict the next sentence

Predict likelihood that sentence B belongs after sentence A

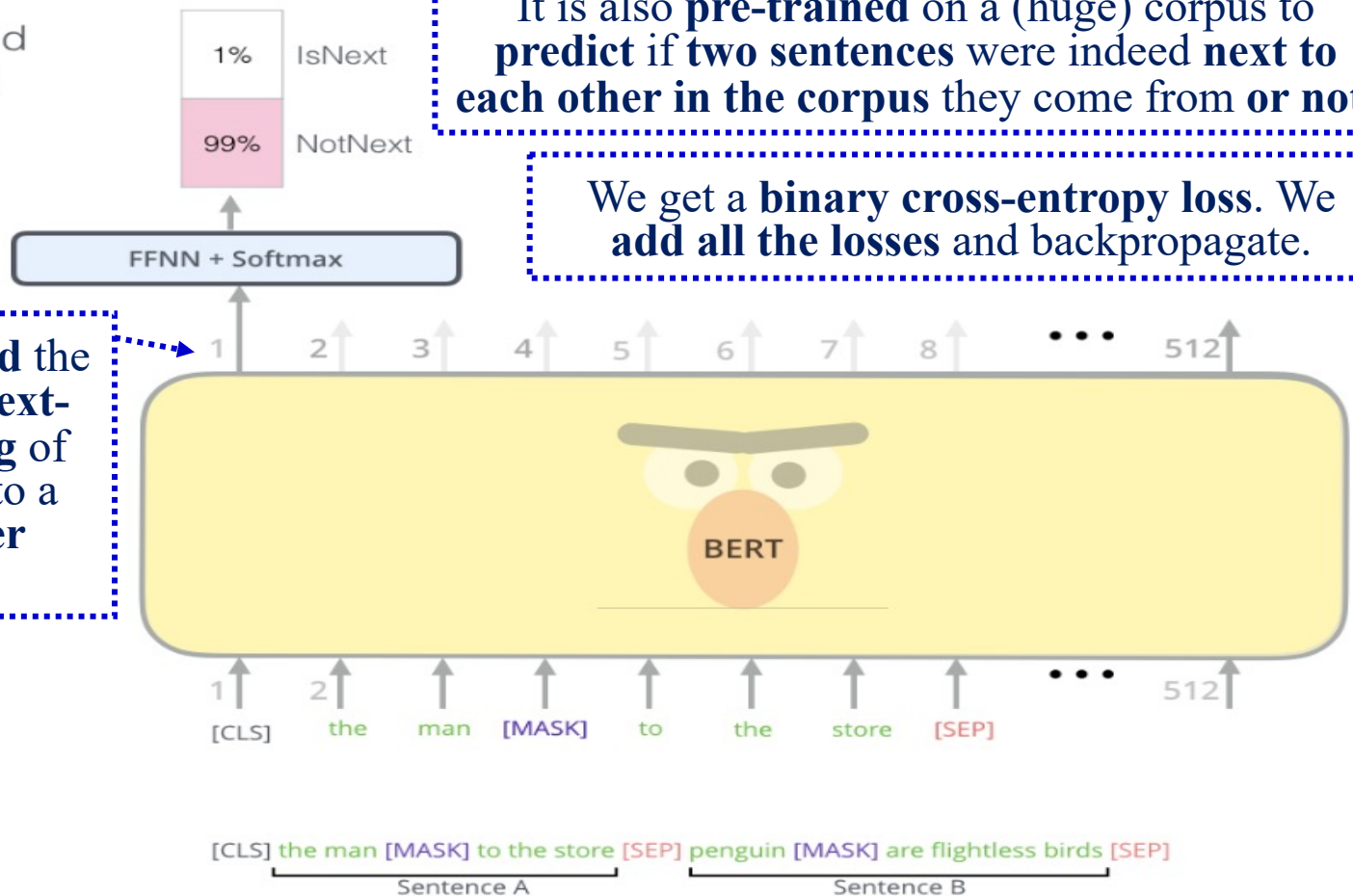
It is also **pre-trained** on a (huge) corpus to **predict if two sentences were indeed next to each other in the corpus they come from or not.**

We get a **binary cross-entropy loss**. We **add all the losses** and backpropagate.

In this case, we **feed** the top-layer **context-aware embedding** of the [CLS] token to a **binary classifier (MLP)**.

Tokenized Input

Input



The second task BERT is pre-trained on is a two-sentence classification task. The tokenization is oversimplified in this graphic as BERT actually uses WordPieces as tokens rather than words --- so some words are broken down into smaller chunks.

Figures from J. Alammari's "The Illustrated BERT, ELMo, and co." (<http://jalammari.github.io/illustrated-bert/>). BERT paper: Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 2018 (<https://arxiv.org/abs/1810.04805>).

# BERT – Fine-tuning for sentence classification

We feed the **context-aware top-layer embedding** of the [CLS] token of each **sentence** to a **task-specific classifier** (e.g., MLP or single dense layer) that classifies the sentence (e.g., **Positive, Neutral, Negative** etc.).

$$\vec{o} = \text{softmax}(W \cdot \vec{h}_{[CLS]} + \vec{b})$$

**Cross-entropy** and backprop.

Starting from the **pre-trained BERT**, we **jointly train BERT (further)** and the **task-specific classifier** on (possibly few) **task-specific training examples** (e.g., tweets + opinion labels).

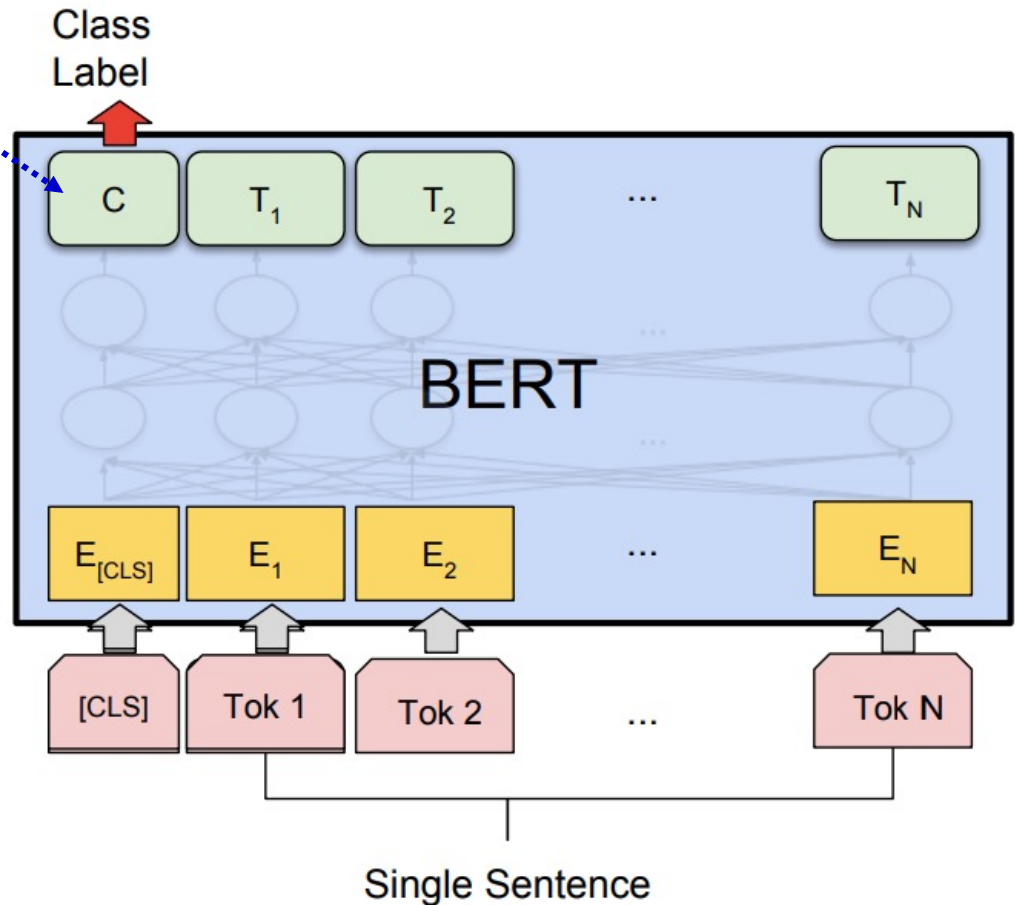


Figure from Devlin et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, 2018 (<https://arxiv.org/abs/1810.04805>).

# BERT – Fine-tuning for token classification

We feed the **context-aware top-layer embedding** of each word of the sentence to a **classifier** (e.g., MLP or single dense layer) that classifies them as **B-Per** (1<sup>st</sup> word of person name), **I-Per** (non-first word of person name), **B-Org** (1<sup>st</sup> word of organization name), **I-Org**, ..., **Other**.

$$\vec{o}_i = \text{softmax}(W \cdot \vec{h}_i + \vec{b})$$

**Cross entropy per position  $i$ .**  
Sum them and backprop.

Starting from the **pre-trained BERT**, we **jointly train BERT (further)** and the **task-specific classifier** on (possibly few) **task-specific training examples**.

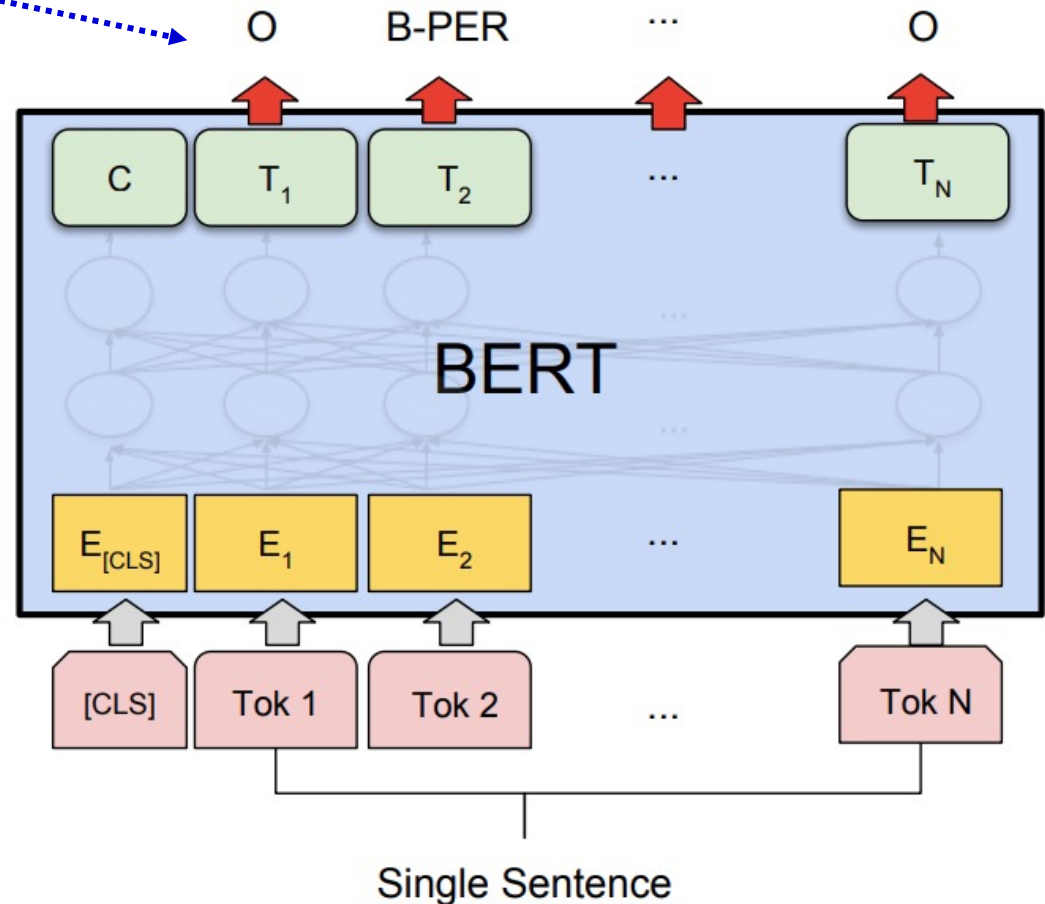


Figure from Devlin et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, 2018 (<https://arxiv.org/abs/1810.04805>).

# BERT – Fine-tuning for textual entailment

We feed the **context-aware top-layer embedding** of the [CLS] token of each **sentence pair** to a **task-specific classifier** (e.g., MLP or single dense layer) that classifies the pair as **Entailment (E)**, **Contradiction (C)**, **Neutral (N)**.  
E.g., “Mary plays in the garden” entails “Mary is in the garden”, but contradicts “Mary is asleep”, and is neutral to “John is in the garden”).

$$\vec{o} = \text{softmax}(W \cdot \vec{h}_{[CLS]} + \vec{b})$$

**Cross-entropy** and backprop.

Starting from the **pre-trained BERT**, we **jointly train BERT (further)** and the **task-specific classifier** on (possibly few) **task-specific training examples**.

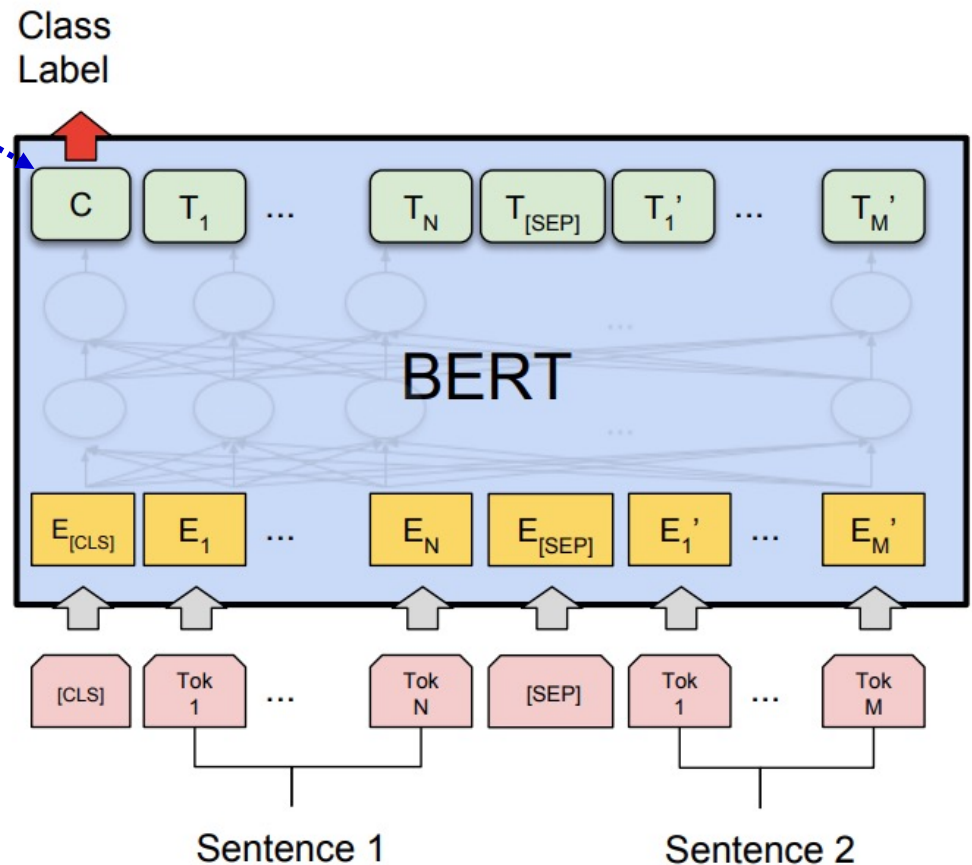


Figure from Devlin et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, 2018 (<https://arxiv.org/abs/1810.04805>).

# Machine Reading Comprehension (MRC)

Paragraph

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called “showers”.

Question

What causes precipitation to fall?

**gravity**

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?

**graupel**

Where do water droplets collide with ice crystals to form precipitation?

**within a cloud**

For each paragraph and question, we need to **classify the tokens of the paragraph** as being **parts of the answer or not**.

Figure from P. Rajpurkar et al., “SQuAD: 100,000+ Questions for Machine Comprehension of Text.”, EMNLP 2016 (<https://aclweb.org/anthology/D16-1264>).

# BERT – Fine-tuning for MRC

We feed the **context-aware top-layer embedding** of each word of the paragraph to a **classifier** (e.g., MLP or single dense layer) that classifies them as **Beginning**, **Inside**, or **Outside** of the answer.

$$\vec{o}_i = \text{softmax}(W \cdot \vec{h}_i + \vec{b})$$

**Cross entropy per paragraph position  $i$ .** Sum them and **backprop.**

Starting from the **pre-trained BERT**, we **jointly train BERT (further)** and the **task-specific classifier** on (possibly few) **task-specific training examples**.

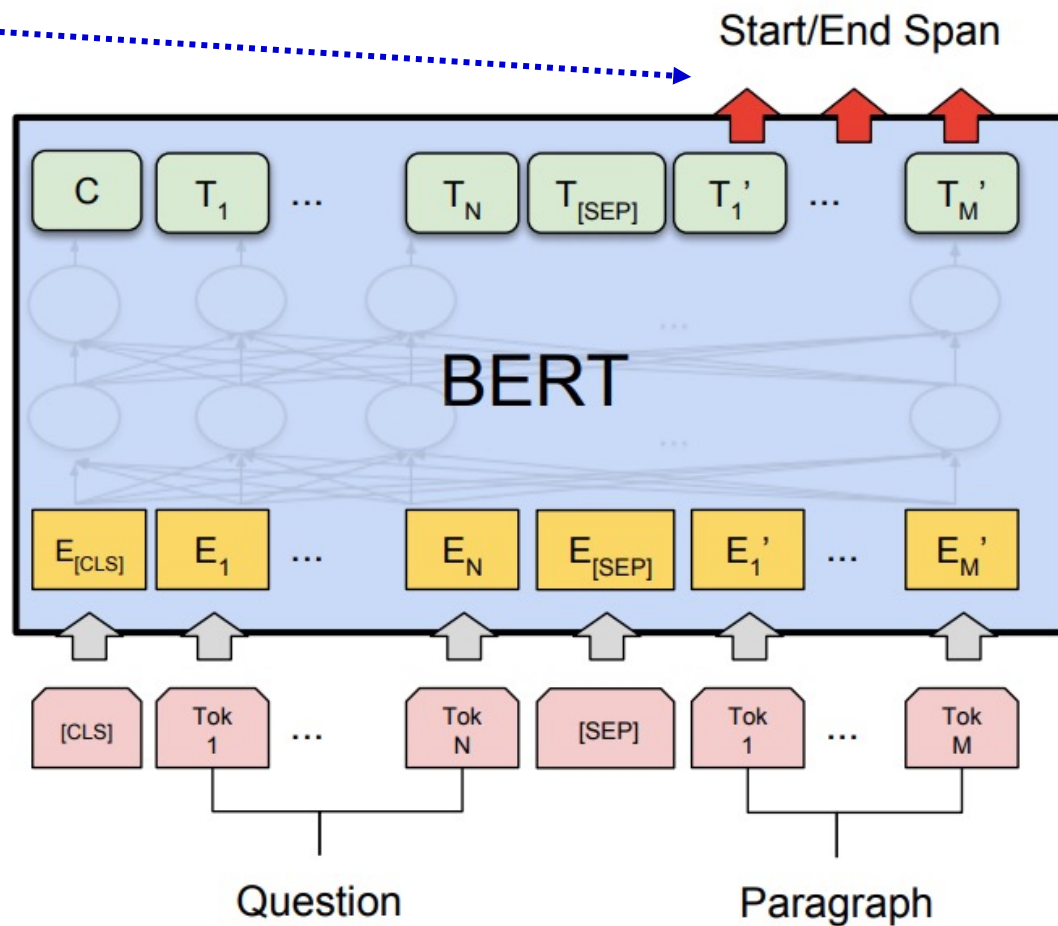
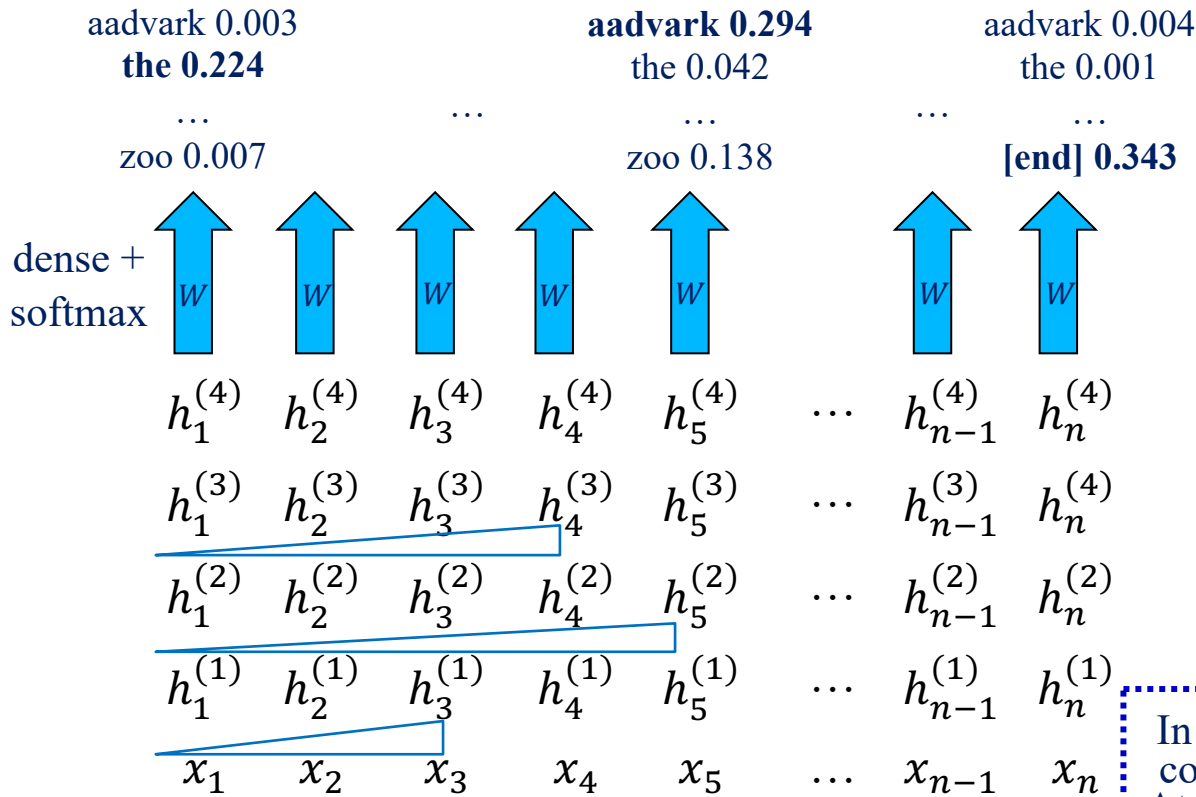


Figure from Devlin et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, 2018 (<https://arxiv.org/abs/1810.04805>).

# Transformer-based LLMs

“Decoder-only”  
Transformers



For each word position, we get a **probability distribution** showing how probable it is for **each word of the vocabulary** to be the **next word**. The first word of the sequence is a pseudo-word [start].

The LLM is **trained to predict the next word** using millions of sequences from the Web. For each **prediction**, we get a **cross-entropy loss** and adjust the weights.

In **training**, all vectors of a layer can be computed in **parallel**, as with encoders. At **test time**, we generate **word by word**.

When **revising the embedding  $x_i$**  at position  $i$ , we **only look at the words at positions  $\leq i$** , because the LLM has to **predict the word at position  $i + 1$**  based on the previous words **only**. Similarly, when revising  $h_i^{(j)}$ .

$$h_i^{(1)} = \text{MLP}^{(1)} \left( \sum_{r=1}^i a_{i,r}^{(1)} x_r \right) + x_i$$

$$h_i^{(j)} = \text{MLP}^{(j)} \left( \sum_{r=1}^i a_{i,r}^{(j)} h_r^{(j-1)} \right) + h_i^{(j-1)}$$

Multiple heads in practice, layer norm, dropout.

# Νευρωνικά γλωσσικά μοντέλα

- Ένα εκπαιδευμένο ΓΜ μπορεί επίσης να συνεχίσει ένα ημιτελές κείμενο (π.χ. ημιτελή πρόταση ή διάλογο).
- Διαλέγοντας κάθε φορά την πιο πιθανή επόμενη λέξη (ή μια από τις πιο πιθανές ή beam search).
- Μια πιθανή συνέχιση ενός ερωτήματος είναι η σωστή απάντηση.
  - Χρειάζεται συνήθως και πρόσθετη εκπαίδευση με ερωτήματα-απαντήσεις.

Default (GPT-3.5)



I teach a 12 week long AI course. Can you suggest what I should cover per week?



Designing a 12-week AI course is a great endeavor. Below is a week-by-week breakdown of topics and concepts you could cover. Keep in mind that the course can be adjusted based on the prior knowledge of your students and the depth you want to go into each topic. Here's a general outline:

## Week 1: Introduction to AI

- What is AI? History and milestones.
- Types of AI: Narrow (Weak) vs. General (Strong) AI.
- AI in popular culture and misconceptions.

## Week 2: Machine Learning Fundamentals

- Introduction to Machine Learning (ML).
- Supervised, Unsupervised, and Reinforcement Learning.
- Data preprocessing and feature engineering.

## Week 3: Supervised Learning

- Linear Regression.
- Logistic Regression.

Παράδειγμα χρήσης του **ChatGPT** (<https://chat.openai.com/>). Το ChatGPT και παρόμοια LLMs χρησιμοποιούν **Transformers**, αντί για RNNs.

# Prompt engineering in GPT-3

## Prompt

I am a highly intelligent question answering bot. If you ask me a question that is rooted in truth, I will give you the answer. If you ask me a question that is nonsense, trickery, or has no clear answer, I will respond with "Unknown".

Q: What is human life expectancy in the United States?  
A: Human life expectancy in the United States is 78 years.

Q: Who was president of the United States in 1955?  
A: Dwight D. Eisenhower was president of the United States in 1955.

Q: Which party did he belong to?  
A: He belonged to the Republican Party.

Q: What is the square root of banana?  
A: Unknown

Q: How does a telescope work?  
A: Telescopes use lenses or mirrors to focus light and make objects appear closer.

Q: Where were the 1992 Olympics held?  
A: The 1992 Olympics were held in Barcelona, Spain.

Q: How many squigs are in a bonk?  
A: Unknown

Q: Where is the Valley of Kings?  
A:

## Sample response

The Valley of Kings is located in Luxor, Egypt.

- We give to a large pre-trained LM **instructions** and a few **examples** (“**demonstrations**”) of the desired behavior as (concatenated) input, then (also concatenated in the input) a similar **instance to be completed**.
  - We can also say what **kind of agent** (e.g., intelligent, polite) the system is, how to **format the answer** etc.
- **No fine-tuning** involved!
  - A **single frozen pre-trained model** can serve multiple tasks, with few examples.

GPT-3 paper:

<https://papers.nips.cc/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf>

GPT-3 examples from:

<https://beta.openai.com/examples/default-qa>

For more ideas on **how to write prompts**, see <https://arxiv.org/abs/2406.06608>.

# Supervised fine-tuning on human responses

- **Just with prompting**, without any fine-tuning, **large LMs** (LLMs, e.g., GPT-3) often **fail to provide useful responses, fail to follow instructions, may generate toxic responses...**
  - Q: What is the capital of Greece? A: Why the %%\$\$ do you care?
- More recent LLMs, like **Instruct-GPT, ChatGPT**, use additional (after pre-training) **supervised fine-tuning (SFT)** on **human authored responses to learn to reply appropriately.**
  - Having **pre-trained the LLM to predict next words**, now **further train it** (with **cross-entropy per word**) to **respond to requests as humans did.**
  - **Back to pre-train then fine-tune**, but without task-specific fine-tuning...

---

**Prompt:**

Serendipity means the occurrence and development of events by chance in a happy or beneficial way. Use the word in a sentence.

---

**Labeler demonstration**

Running into Margaret and being introduced to Tom was a fortunate stroke of serendipity.

---

# Reinforcement learning from human feedback

- **Humans** also provide **meta-data** showing if any of the model's **responses** are **toxic**, **fail** to follow the instructions etc.
- **Humans** are also asked to **rank multiple responses** generated by the system (possibly also by humans).
- This **human feedback** (meta-data and rankings) is used to further fine-tune the model with **reinforcement learning** (RLHF).
- **SFT and RLHF both help** generate more useful responses.

**Output A**

summary1

**Rating (1 = worst, 7 = best)**

1 2 3 4 5 6 7

---

*Fails to follow the correct instruction / task ?*  Yes  No

*Inappropriate for customer assistant ?*  Yes  No

*Contains sexual content*  Yes  No

*Contains violent content*  Yes  No

*Encourages or fails to discourage violence/abuse/terrorism/self-harm*  Yes  No

*Denigrates a protected class*  Yes  No

*Gives harmful advice ?*  Yes  No

*Expresses moral judgment*  Yes  No

**Direct Preference Optimization (DPO)** is a popular alternative to conventional RLHF.

<https://arxiv.org/abs/2305.18290>

<https://huggingface.co/blog/pref-tuning>

# Chain-of-thought prompting

## Standard Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The answer is 27. ❌

## Chain-of-Thought Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

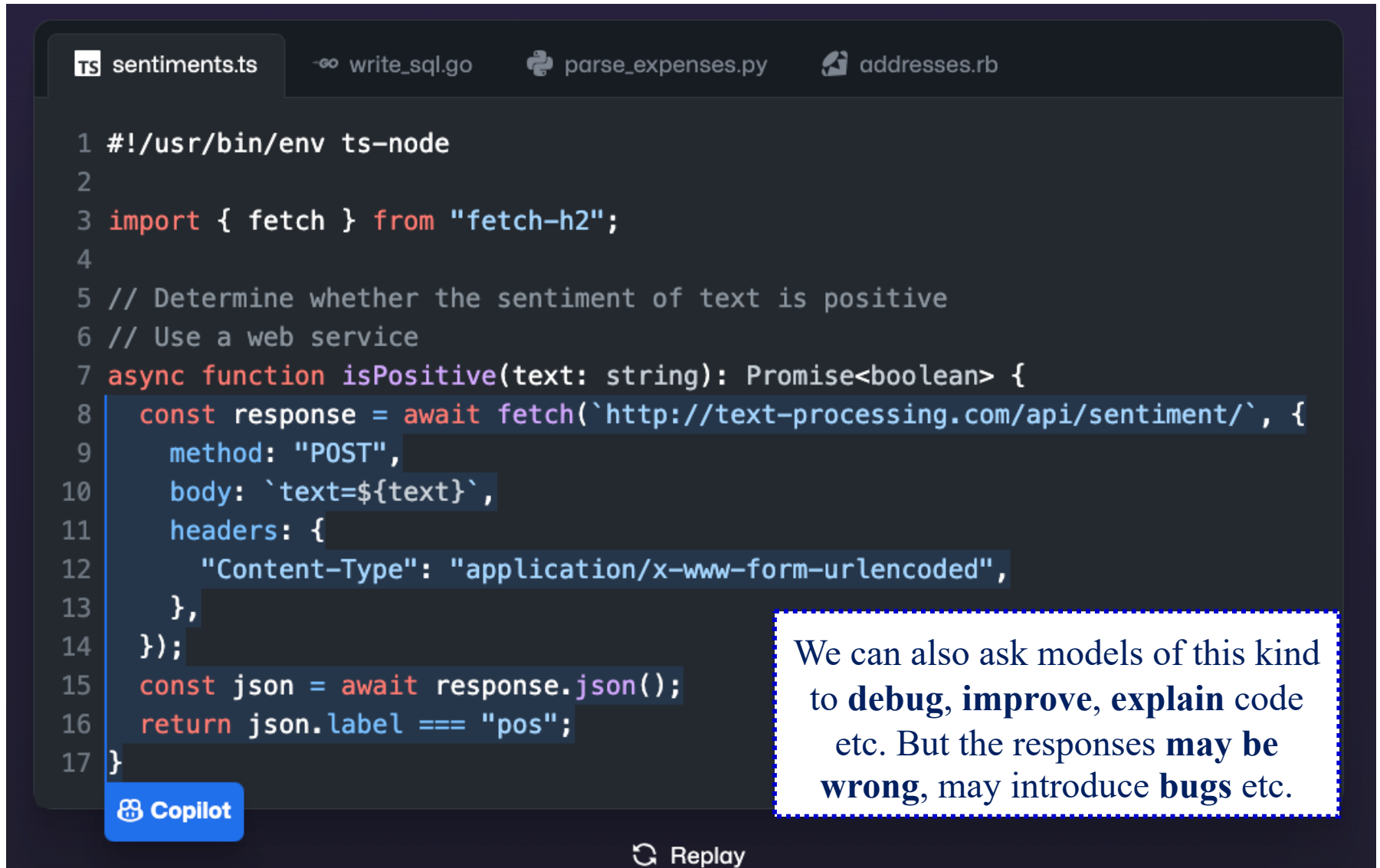
### Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had  $23 - 20 = 3$ . They bought 6 more apples, so they have  $3 + 6 = 9$ . The answer is 9. ✅

- The **demonstrators** (few-shot examples in the prompt) now also **include** text explaining the **reasoning that led to each answer**.
  - We prompt the model to **generate both the answer and its reasoning**.
  - **Performance often improved and we also get some explanation (?)**.

Figure from Wei et al. (2022), “Chain-of-thought prompting elicits reasoning in large language models”, NeurIPS 2022 (<https://arxiv.org/abs/2201.11903>).

# Generating code completions



The image shows a code editor interface with several tabs at the top: 'sentiments.ts' (active), 'write\_sql.go', 'parse\_expenses.py', and 'addresses.rb'. The main editor area displays TypeScript code for a function named 'isPositive'. The code is as follows:

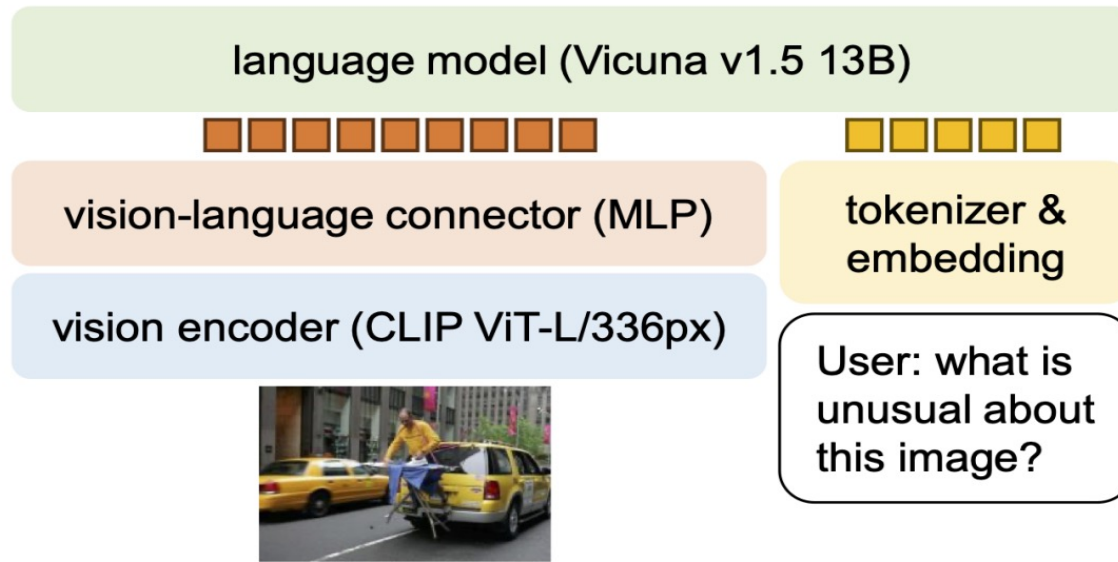
```
1 #!/usr/bin/env ts-node
2
3 import { fetch } from "fetch-h2";
4
5 // Determine whether the sentiment of text is positive
6 // Use a web service
7 async function isPositive(text: string): Promise<boolean> {
8   const response = await fetch(`http://text-processing.com/api/sentiment/`, {
9     method: "POST",
10    body: `text=${text}`,
11    headers: {
12      "Content-Type": "application/x-www-form-urlencoded",
13    },
14  });
15  const json = await response.json();
16  return json.label === "pos";
17 }
```

A blue Copilot icon is visible in the bottom left corner of the editor. A callout box on the right side of the editor contains the following text:

We can also ask models of this kind to **debug, improve, explain** code etc. But the responses **may be wrong**, may introduce **bugs** etc.

At the bottom center of the editor, there is a 'Replay' button with a circular arrow icon.

# Adding vision to LLMs: LLaVA-1.5

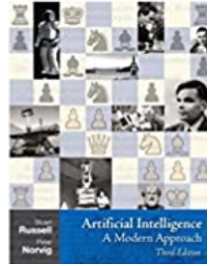


- An **image encoder** (e.g., CNN, here ViT) produces **image embeddings**. One embedding from the **channels** of each “**pixel**” of the **last max-pooling layer**.
- An **MLP** projects them to the **space** of the **token embeddings**.
- The **LLM** is **fed** with both **image and token embeddings** (user question), word-by-word **generates a textual response**.

Figure from Liu et al. (2024), “Improved Baselines with Visual Instruction Tuning”, CVPR 2024 (<https://arxiv.org/abs/2310.03744>).

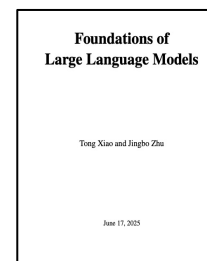
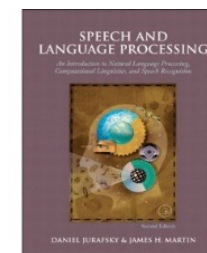
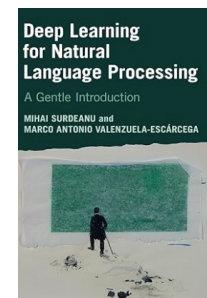
# Βιβλιογραφία

- Russel & Norvig (4<sup>η</sup> έκδοση, ελληνική μετάφραση), ενότητες 24.4, 24.5, 24.6.
- Βλαχάβας κ.ά.: Δεν υπάρχουν σχετικές ενότητες.



# Recommended reading

- M. Surdeanu and M.A. Valenzuela-Escarcega, *Deep Learning for Natural Language Processing: A Gentle Introduction*, Cambridge Univ. Press, 2024.
  - Chapters 12–13. See <https://clulab.org/gentlenlp/text.html>
  - Also available at AUEB's library.
- Jurafsky and Martin's, *Speech and Language Processing* is being revised (3<sup>rd</sup> edition) to include DL methods.
  - <http://web.stanford.edu/~jurafsky/slp3/>
- T. Xiao and J. Zhu, *Foundations of Large Language Models*, preprint.
  - <https://arxiv.org/abs/2501.09223>
  - Chapter 4 covers SFT and RL alignment methods.



# Recommended reading – continued

- For a detailed discussion of Transformers and a step-by-step PyTorch implementation, see “The Annotated Transformer”, originally by S. Rush, updated by A. Huang et al. (2022).
  - <http://nlp.seas.harvard.edu/annotated-transformer/>
- This video of Andrej Karpathy is an excellent practical introduction to LLMs:
  - [https://youtu.be/zjkBMFhNj\\_g?feature=shared](https://youtu.be/zjkBMFhNj_g?feature=shared)