



University College Dublin  
Ireland's Global University

# Quantitative Text Analysis

University College Dublin

---

Instructor: Yen-Chieh Liao & Stefan Müller

Week 12 (22 April 2024)

# Outline

1. From Bag-of-Words to the Transformer
2. Transformers
3. API
4. Lab Session

# **From Bag-of-Words to the Transformer**

---

# Bag-of-Words

## Advantages

- Easy to implement
- Human-interpretable results
- Domain adaptation

## Disadvantages

- Dimensionality curse.
- No solution for unseen words.
- Hardly capture semantic relations such as is-a, has-a, synonyms.
- Word order information is ignored.
- Slow for large vocabularies.

# Word Embeddings

- Out of Vocabulary (OOV): word embeddings have a significant limitation in handling words that are not present in their training data.
- Static embeddings don't update with new information or vocabulary unless re-trained, making them less adaptable to evolving language use or new data.
- word embeddings struggle to effectively handle multiple-word expressions and compound words.
- word embeddings cannot adequately represent words with multiple meanings (polysemy) since each word is assigned a single, context-independent vector.

# Transformers

---

# Transformers

**Bidirectional Context:** from both left and right sides of a token within a sentence, unlike directional models which only read text left-to-right or right-to-left.

**Transformer Architecture:** Uses attention mechanisms to weigh the importance of each word in the sentence, no matter its position, which enhances understanding compared to BoW, which treats each word independently.

**Subword Tokenization:** BERT uses WordPiece tokenization, handling unknown words better than BoW by breaking them down into known subwords.

**Fine-Tuning:** Once pre-trained, BERT can be fine-tuned with additional output layers to perform well on a variety of specific tasks much better than non-contextual methods like BoW.

**Position Embeddings:** Includes position information which allows the model to understand ordering and structure in the text, offering significant advantages over BoW, which loses all order information.



**Layer** and **Attention Head** are two key components understanding and processing language.

- **Multi-Head Self-Attention:** allows the model to learn different features of each word in the input sequence from different subspaces. Self-Attention capture complex relationships between words.
- **Feed-Forward Neural Networks:** The output from each position is processed by the same feed-forward neural network, which is shared across all positions.

# Attention Head

- **Diversity:** Each attention head has different parameters, so they can capture features in different representational subspaces. This diversity allows the model to understand the data more comprehensively.
- **Focus:** some heads may focus on capturing syntactic
- **Integrated Information Processing:** By combining the outputs of multiple attention heads, the model can synthesize information from different aspects, forming a richer text representation.

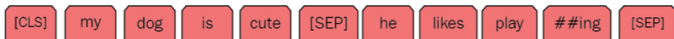
## Tokens in Transformers:

- **[CLS]**: Used for sequence-level tasks like classification. The final hidden state of the [CLS] token is used to represent the entire sequence's sentiment, e.g., *"[CLS] I loved the new Batman movie! [SEP]"*.
- **[SEP]**: Separates sentences or sequences, useful for tasks involving two sentences, e.g., *"[CLS] What year was Batman released? [SEP] Batman was released in 2022. [SEP]"*.
- **[MASK]**: During pre-training, [MASK] is used for the Masked Language Model task, masking words randomly for the model to predict, e.g., *"The quick brown [MASK] jumps over the lazy dog" might mask "fox"*.

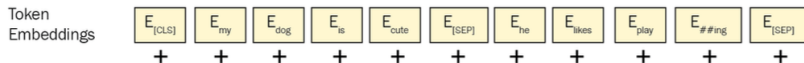
## Input Text

```
text = "[CLS] my dog is cute [SEP] he likes ##ing [SEP]"
```

Input



**Token Embeddings** These are vectors that represent each token in the input. Each token (like “[CLS]”, “my”, “dog”, “is”, “cute”, “[SEP]”, etc.) is converted into a vector called a token embedding. These embeddings capture the semantic meaning of each token.



**Segment Embeddings:** are used to distinguish between two different sentences or segments within the same input. In this case, the tokens belonging to the first sentence are marked with one segment embedding ( $E_A$ ), and the tokens of the second sentence are marked with another ( $E_B$ ).

Segment  
Embeddings



**Position Embeddings:** Since BERT doesn't inherently capture the order of tokens, position embeddings are added to give the model information about the position of each token in the sequence. Each position in the sequence has a corresponding position embedding ( $E_0, E_1, E_2$ , etc.).



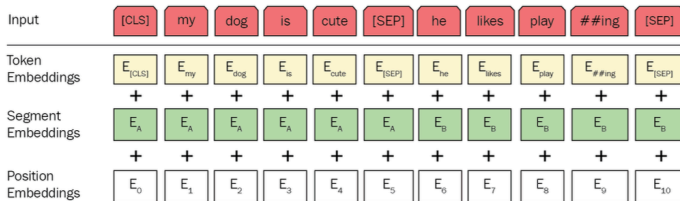


Figure 3.1 – The BERT model

**Source:** Denis Rothman (2021, P. 78)

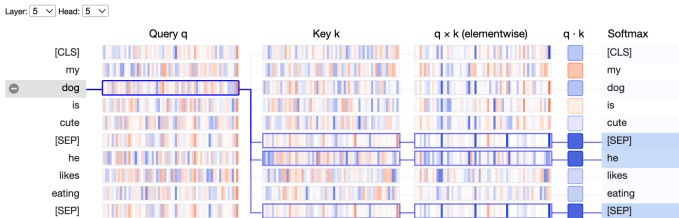


## **Attention Weights:**

- Self-attention weights in Transformer models quantify the relevance of all tokens when assessing a particular one, aiding in the contextual understanding of words.
- This attention pattern can vary across different layers and heads, as each one might learn to focus on different aspects of the input.

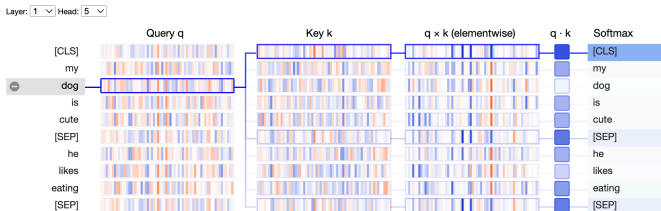
## Dog in Layer 5

If there is a thick line from the word “dog” to “he”, it suggests that within the given head, the model pays more attention to “he” when processing the word “dog”.



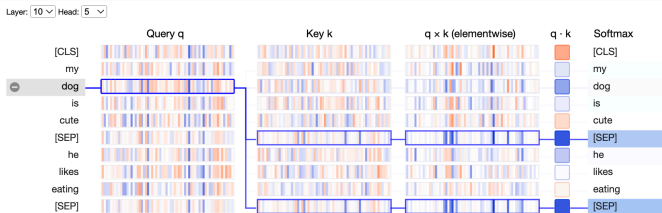
**Code:** `2024_qta_lab_week_12_transformers_supplements.ipynb`

## Dog in Layer 1



**Code:** *2024\_qta\_lab\_week\_12\_transformers\_supplements.ipynb*

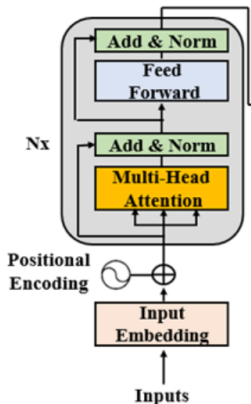
## Dog in Layer 10



**Code:** `2024_qta_lab_week_12_transformers_supplements.ipynb`

# Attention Is All You Need

Primary advantage is the ability to efficiently process sequential data while capturing long-distance dependencies between elements in the sequence (Vaswani, Ashish, et al. 2017).



## In Transformer API

```
print(" input_ids: \n",encoded_input['input_ids'])

## input_ids:
## tensor([[ 101,  1031, 23616,  2015,  1033,  2026,  3899,  2003, 10140,  102,
##           2002,  7777,  1001,  1001, 13749,  102,  102]])

print(" token_type_ids: \n",encoded_input['token_type_ids'])

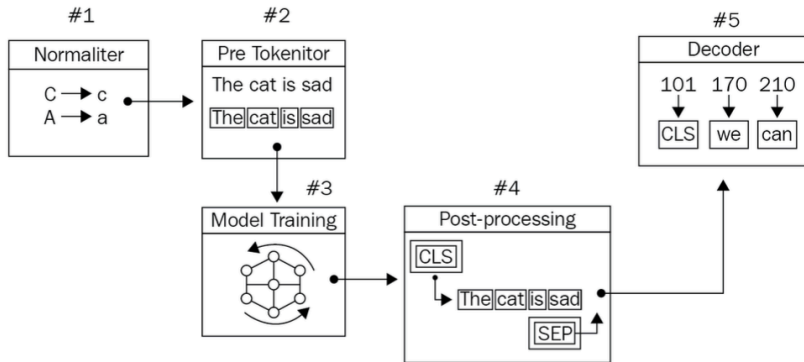
## token_type_ids:
## tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])

print(" attention_mask: \n",encoded_input['attention_mask'])

## attention_mask:
## tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

**Code:** *2024\_qta\_lab\_week\_12\_transformers\_supplements.ipynb*

# Tokenization in Transformers



**Source:** Denis Rothman (2021, P.106)

# Training Model vs. Fine-tuning

Aspect	Fine-tuning	Training from Scratch
<b>Starting Point</b>	Starts with a pre-trained model	Begins with randomly initialized model weights
<b>Data Requirement</b>	Requires less data due to prior learned features	Needs a large dataset to learn from
<b>Time and Resources</b>	Less time and resources needed	More time-consuming and resource-intensive
<b>Use Cases</b>	Suitable for adapting existing models to new tasks	Ideal for novel tasks with no existing models



**API**

---

# HuggingFace vs FLAIR NLP

Feature	Hugging Face	FLAIR
<b>Model Variety</b>	Wide range of pre-trained transformer models	High-quality models for specific tasks like NER & PoS
<b>Ease of Use</b>	User-friendly, great for beginners and experts	User-friendly with a focus on PyTorch users
<b>Integration</b>	Integrates with PyTorch, TensorFlow, and JAX	Built on top of PyTorch
<b>Community Support</b>	Large, with extensive resources and a collaborative hub	Growing, research-oriented community
<b>Use Cases</b>	Broad range of NLP tasks	Specialized tasks where character-level modeling excels

# Hugging Face Playground

distilbert/**distilbert-base-uncased**

like 408

Fill-Mask Transformers PyTorch TensorFlow JAX Rust Safetensors bookcorpus wikipedia English distilbert expert

Inference Endpoints arxiv:1910.01108 License: apache-2.0

Model card Files and versions Community 18

Train Deploy Use in Transformers

Edit model card

DistilBERT base model (uncased)

This model is a distilled version of the [BERT base model](#). It was introduced in [this paper](#). The code for the distillation process can be found [here](#). This model is uncased: it does not make a difference between english and English.

Model description

DistilBERT is a transformers model, smaller and faster than BERT, which was pretrained on the same corpus in a self-supervised fashion, using the BERT base model as a teacher. This means it was pretrained on the raw texts only, with no humans labelling them in any way (which is why it can use lots of publicly available data) with an automatic process to generate inputs and labels from those texts using the BERT base model. More precisely, it was pretrained with three objectives:

- Distillation loss: the model was trained to return the same probabilities as the BERT base model.
- Masked language modeling (MLM): this is part of the original training loss of the BERT base model. When taking a sentence, the model randomly masks 15% of the words in the input then run the entire masked sentence through the model and has to predict the masked words. This is different from traditional recurrent

Downloads last month  
**16,393,049**

Safetensors Model size 67M params Tensor type F32

Inference API

Fill-Mask Examples

Mask token: [MASK]

Paris is the [MASK] of France.

Compute

Computation time on cpu: cached

capital	0.982
birthplace	0.003
northeznmost	0.001
centre	0.001
southeznmost	0.001

JSON Output Maximize

**URL:** <https://huggingface.co/distilbert/distilbert-base-uncased>

# Hugging Face Tutorial

The screenshot shows the Hugging Face website's NLP Course page. The top navigation bar includes links for Models, Datasets, Spaces, Posts, Docs, and Pricing. The left sidebar contains a search bar, a language selector (EN), and a table of contents with sections: 0. SETUP, 1. TRANSFORMER MODELS (selected), 2. USING TRANSFORMERS, 3. FINE-TUNING A PRETRAINED MODEL, and 4. SHARING MODELS AND TOKENIZERS. The main content area is titled 'Introduction' and features a large video player with the text 'Welcome to the Hugging Face Course'. Below the video, a paragraph states: 'This course will teach you about natural language processing (NLP) using libraries from the Hugging Face ecosystem — Transformers, Datasets, Tokenizers, and Accelerate — as well as the Hugging Face Hub.' The right sidebar contains a list of links: Introduction, Welcome to the Course!, What to expect?, Who are we?, FAQ, and Let's Go.

Hugging Face

Search models, datasets, users...

Models Datasets Spaces Posts Docs Pricing

NLP Course

Search documentation

EN 1,009

0. SETUP

1. TRANSFORMER MODELS

Introduction

Natural Language Processing

Transformers, what can they do?

How do Transformers work?

Encoder models

Decoder models

Sequence-to-sequence models

Bias and limitations

Summary

End-of-chapter quiz

2. USING TRANSFORMERS

3. FINE-TUNING A PRETRAINED MODEL

4. SHARING MODELS AND TOKENIZERS

Introduction

Welcome to the Course!

What to expect?

Who are we?

FAQ

Let's Go

Welcome to the Hugging Face course

Watch on YouTube

This course will teach you about natural language processing (NLP) using libraries from the Hugging Face ecosystem — Transformers, Datasets, Tokenizers, and Accelerate — as well as the Hugging Face Hub.

**Source:** <https://huggingface.co/learn/nlp-course/chapter1/1>

# FLAIR NLP (More Focus on NLP Tasks)

 [Flair](#) [Tutorial](#) [Blog](#) [Demo](#)

[GitHub](#)  

## flair

A very simple framework for state-of-the-art Natural Language Processing (NLP)

[Get started](#)



### Easy to Use

State-of-the-art NLP with just a few lines of code!  
Find entities, detect sentiment, and more → [check out our demo!](#)



### Huge Community

With a community of ~200 code contributors, Flair is used in hundreds of companies, over [2,000 open source projects](#), and over 2,000 research papers!



### Open Source and Free

Flair is completely free and open source.

# Lab Session

---

# Lab Session: Files

## FLAIR NLP

- [2024\\_qta\\_lab\\_week\\_12\\_flair.ipynb](#)
- [2024\\_qta\\_lab\\_week\\_12.qmd](#) (flaiR)

## Transformers API (Python)

- [2024\\_qta\\_lab\\_week\\_12\\_transformers.ipynb](#)
- [2024\\_qta\\_lab\\_week\\_12\\_transformers\\_supplements.ipynb](#)

## Others

- [2024\\_qta\\_lab\\_week\\_11.ipynb](#) (*previous lab exercise in Python*)

## Lab Session: Tasks

- We will experiment with and fine-tune a transformer model on annotated environmental policy data.
- We will use the DistilBERT model.



# Clearing FLAIR and Its Cache

```
clear_flair_cache()
```

```
remove.packages("flaiR")
```