Warwick Business School

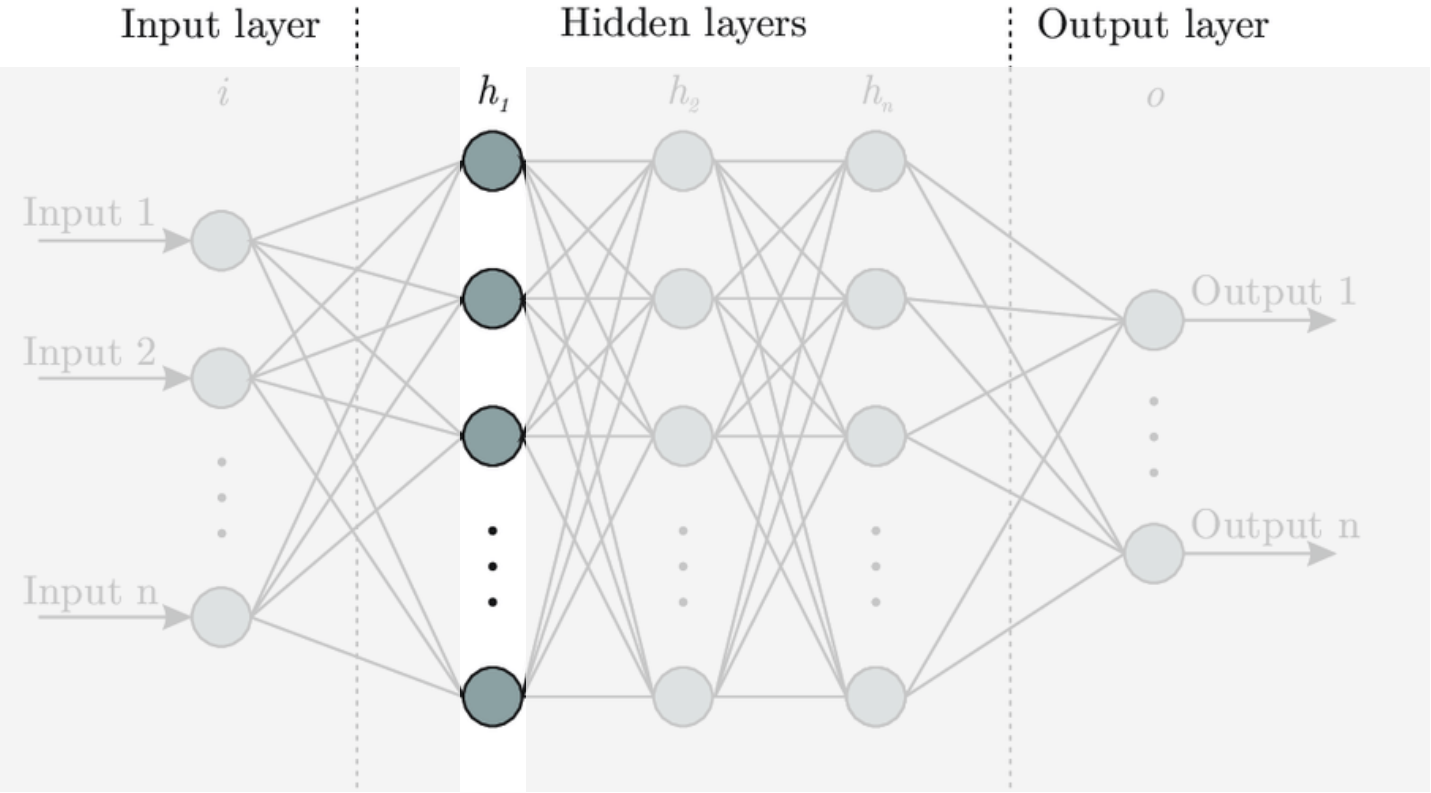
Data Science & Generative AI

Dr Michael Mortenson

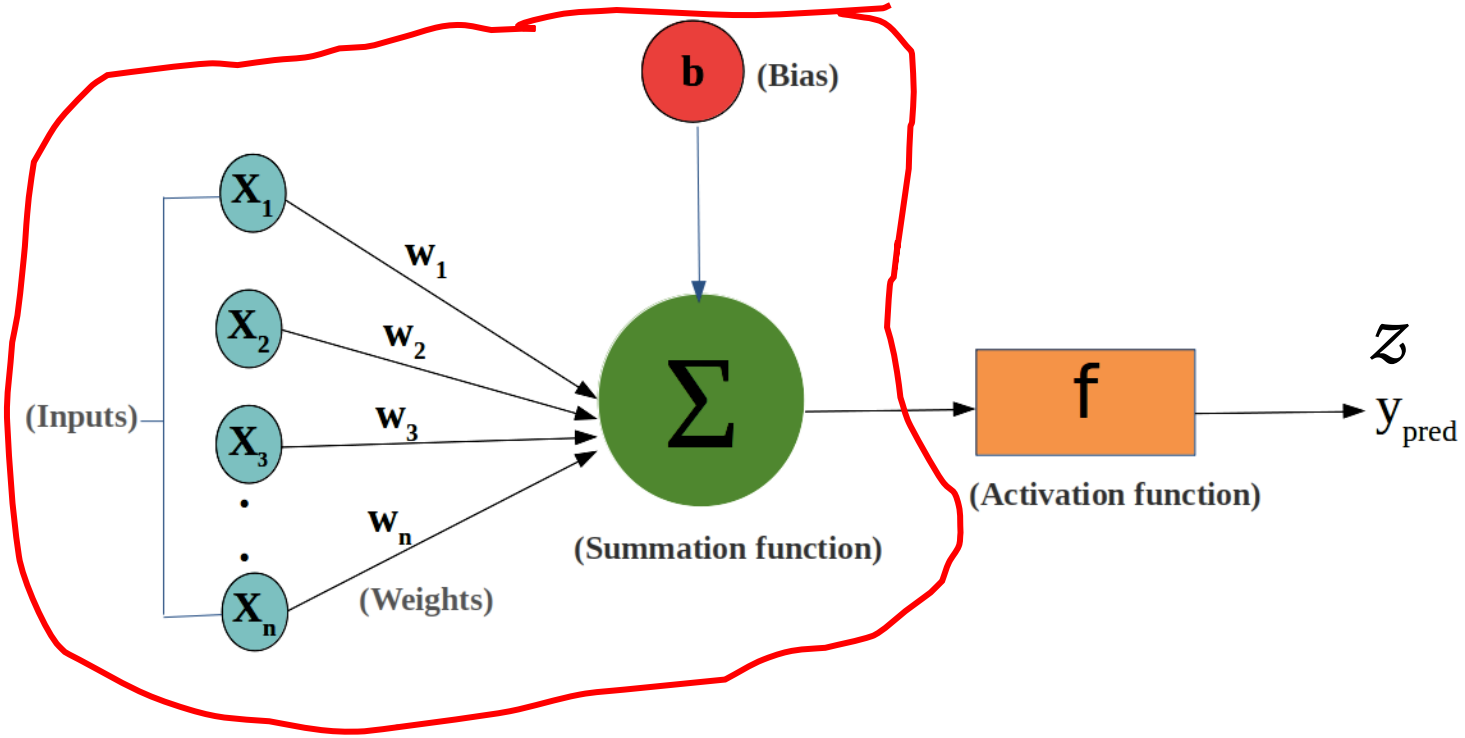
Associate Professor (Reader)
michael.mortenson@wbs.ac.uk

Session 7: Transformers – Self Attention & Other Layers

1.1 The Story So Far: A linear layer of neurons (l_i)



1.1 The Story So Far: A linear layer of neurons (l_i)



1.2 The Layer Cake

1.3 The Transformers Takeover

- ~~Many~~ most AI products are based on transformers including:
  - ✓ ChatGPT
  - ✓ DeepSeek
  - ✓ Gemini
  - ✓ Claude
  - ✓ Copilot
  - ✓ Llama
  - ✓ BERT (etc.)
  - ✓ ViT
  - ✓ Alexa
  - ✓ Grammerly
  - ✓ YouTube Recommendations
  - ✓ Bing
  - ✓ Google Translate
  - ✓ Mistral
  - ✓ Character.AI
  - ✓ Notion.ai
  - ✓ Copy.ai
  - ✓ GitHub Co-pilot
  - ✓ Whisper
  - ✓ DALL-E
  - ✓ Stable Diffusion
  - ✓ Adobe Firefly
  - ✓ IBM Watson
  - ✓ Salesforce Einstein
  - ✓ SAP Joule
  - ✓ Runway ML
  - ✓ Sora
  - ✓ BloombergGPT
  - ✓ Duolingo Max
  - ✓ Tesla FSD
  - ✓ Cohere AI
  - ✓ Khanmigo

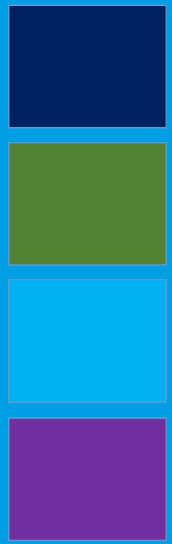
Session Aims

Introduction

Working with “Unstructured” Data

Neural Network Layer Types

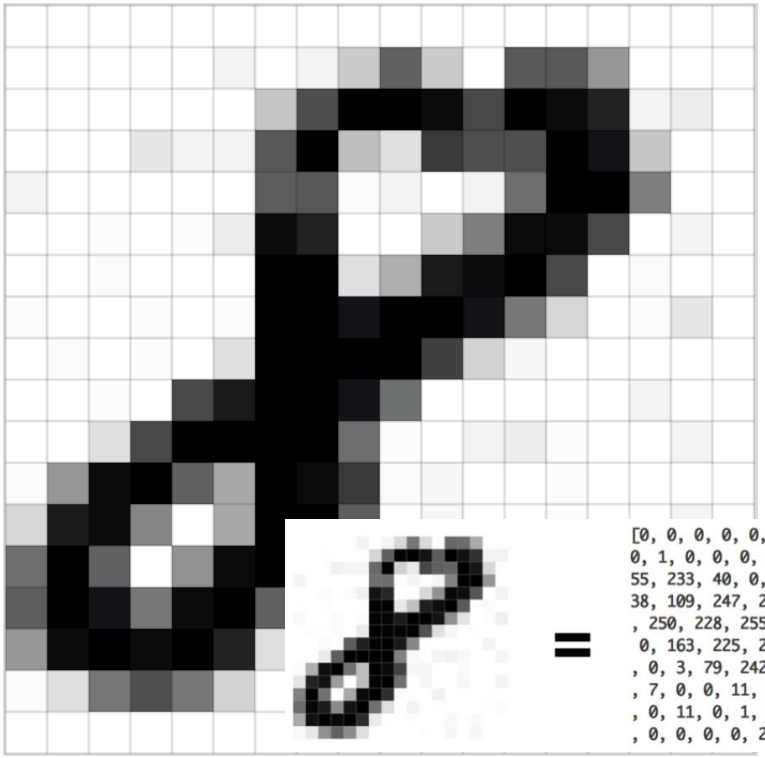
Transformers and Self-Attention



2.1 Deep Learning & Data

Structured data	Semi-structured data	Unstructured data
Databases	XML / JSON data Email Web pages	Audio Video Image data Natural language Documents

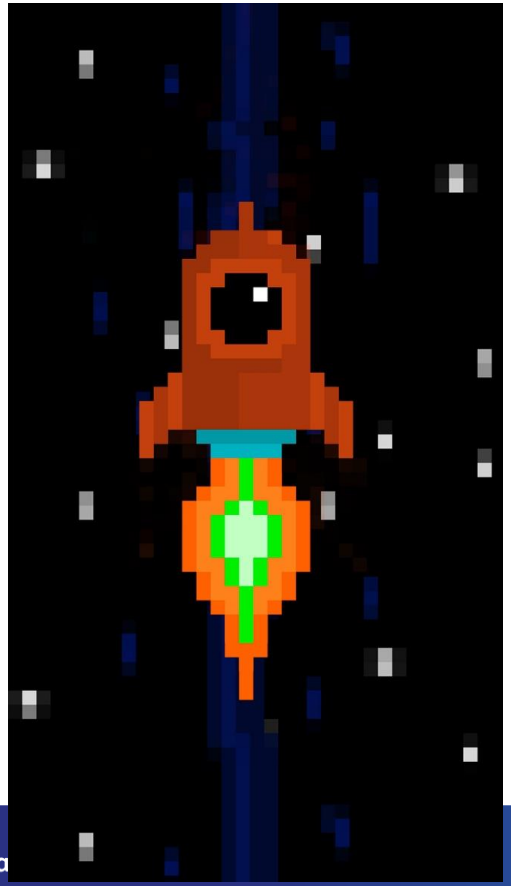
2.2 Image Data

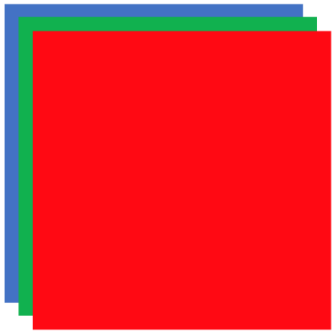


=

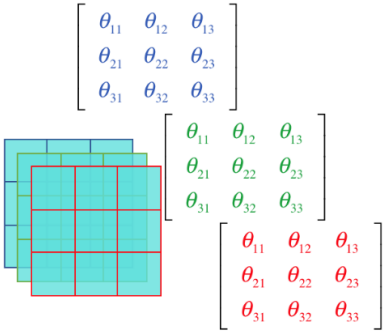
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0,
0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2
55, 233, 40, 0, 0, 10, 0, 0, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 107, 0, 0, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0,
38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252
, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 0, 4,
0, 163, 225, 251, 255, 229, 120, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 10, 14, 6, 0, 0, 9
, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0
, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 0, 145, 248, 228, 116, 235, 255, 141, 34
, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32
, 0, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

2.2 Image Data





Color image



Parameterized filter

INPUT LAYER (RGB IMAGE)

1. # of pixels width
2. # of pixels height
3. # of channels (i.e. colours)

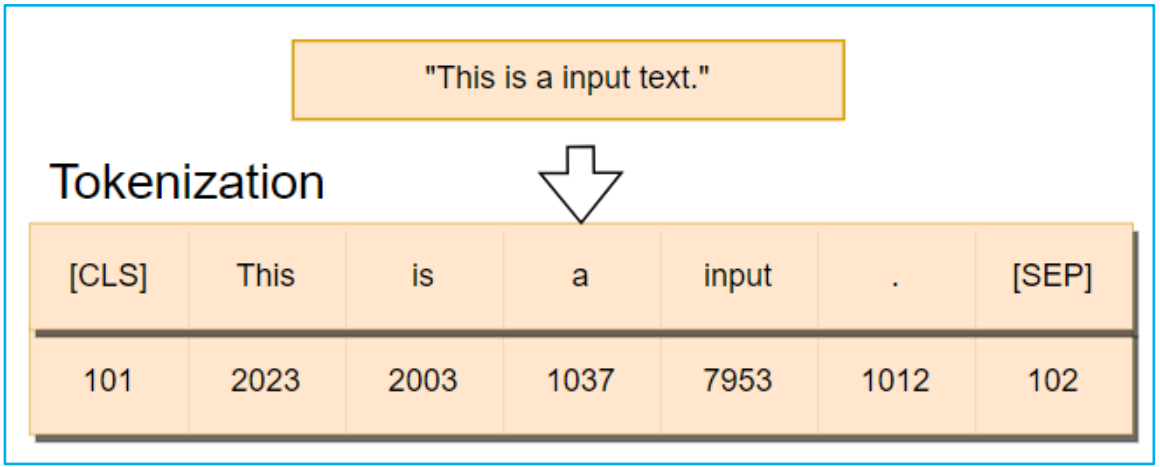
2095 x 3725 x 3

2.3 Text Data (Transformers Approach)

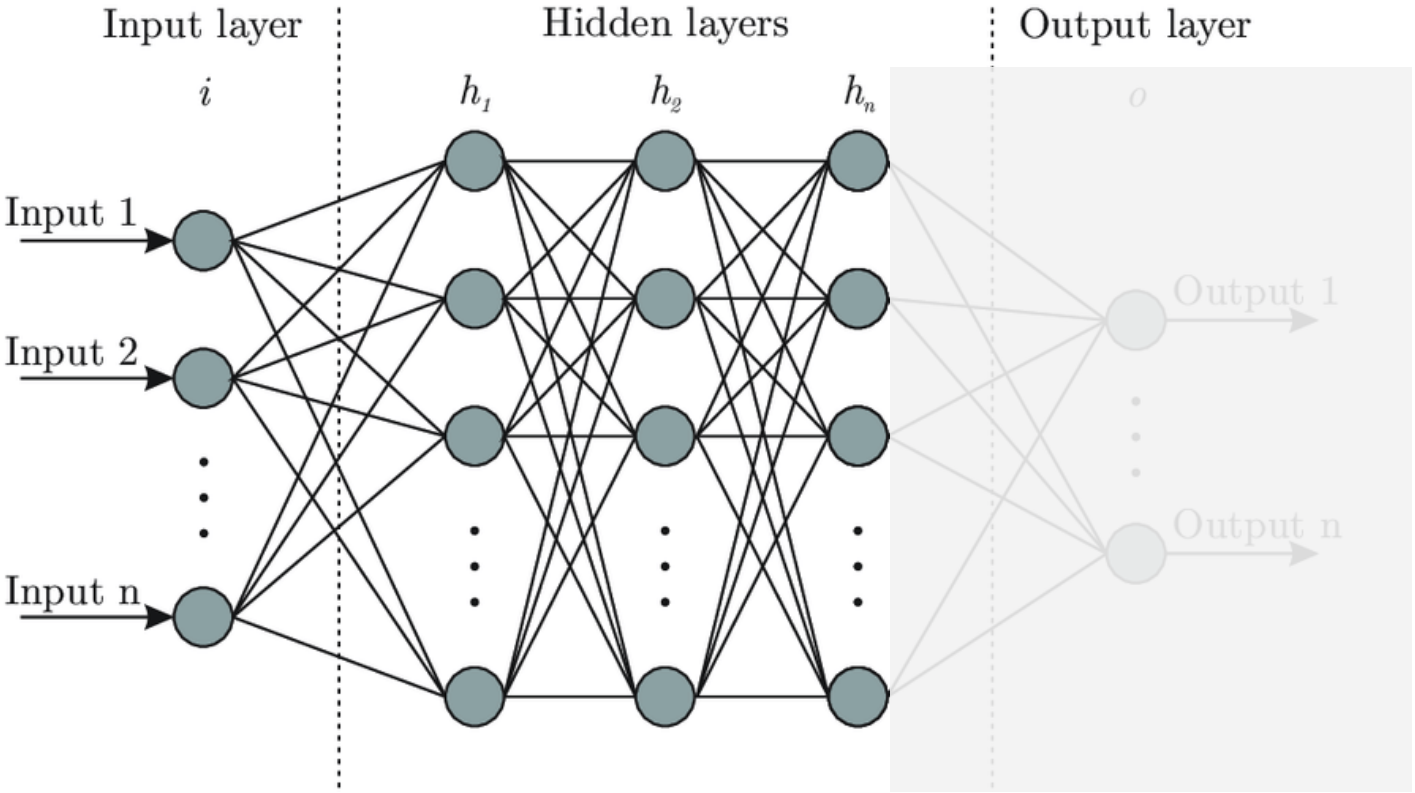
- **Input**: "Please complete the module evaluation"
- **Normalisation & splitting**: "[CLS], please, complete, the, module, evaluation, [SEP]"
- **Tokenisation (numeric)**: "123, 456, 789, 1011, 1112, 1213, 1314, 1415"
- **Embeddings**: "[0.78, 1.23, …, 7.89], [6.54, -1.23, …, 3.21], …, [-1.76, 1.23, … 0.99]"
- **Positions**: [0, 1, 2, 3, 4, 5, 6]
- **Positional encoding**: [[0.11, 5.67, …, 9.87], [5.56, -3.21, …, 1.23], …, [-6.71, 3.21, … 9.90]]
- **Transformed input**: Embeddings $\odot$ Positional Encoding

2.4 Text Data (Tokenisation)

- **Word tokenisation** – split the text by words (common in English).
- **Character tokenisation** – split the text by characters (useful for languages that use more compound words or tasks like spelling correction).
- **Sub-word tokenisation** – split by words or subwords. E.g. “chat” is one token but “chatbot” becomes 2x tokens (“chat” and “bot”). Basically the Goldilocks option. Most GPT-type models use this.



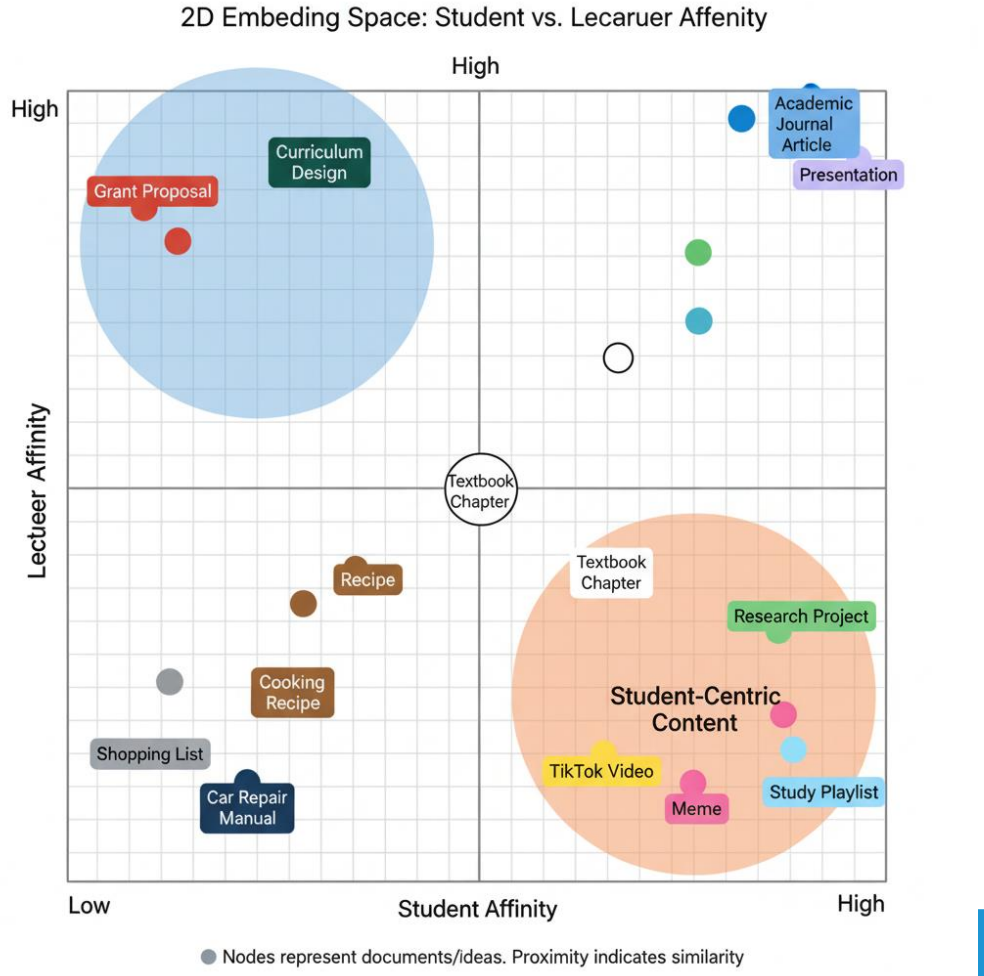
2.5 Text Data (Embeddings)



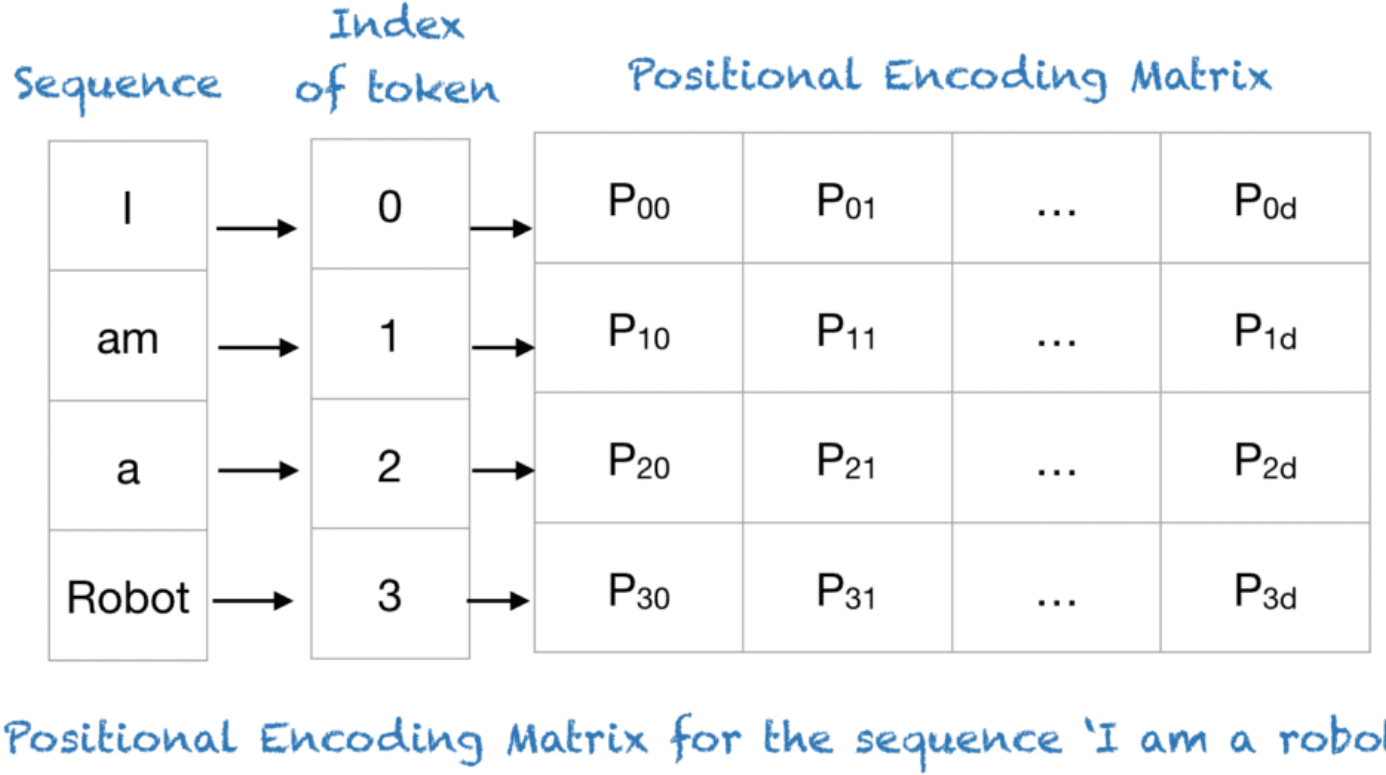
2.5 Text Data (Embeddings)

- An embedding/encoder model is a network with no output prediction:
  - The output is therefore a numerical vector of size 1 x n^L (the number of neurons in the last layer). Effectively we have transformed the data from input space (in the case of transformers each input is an arbitrary ID value) into a feature engineered vector of values.
  - As we know, each neuron has a different specialisation, which in text is a different “topic”. Every word is therefore given a score for how aligned that word is to the particular
  - This is like performing Principle Component Analysis (PCA), which you may have seen elsewhere, except the output space is larger rather than smaller, and we have used the full power of deep learning (non-linearity, many parameters) to find highly representative, semantic spaces …

2.5 Text Data

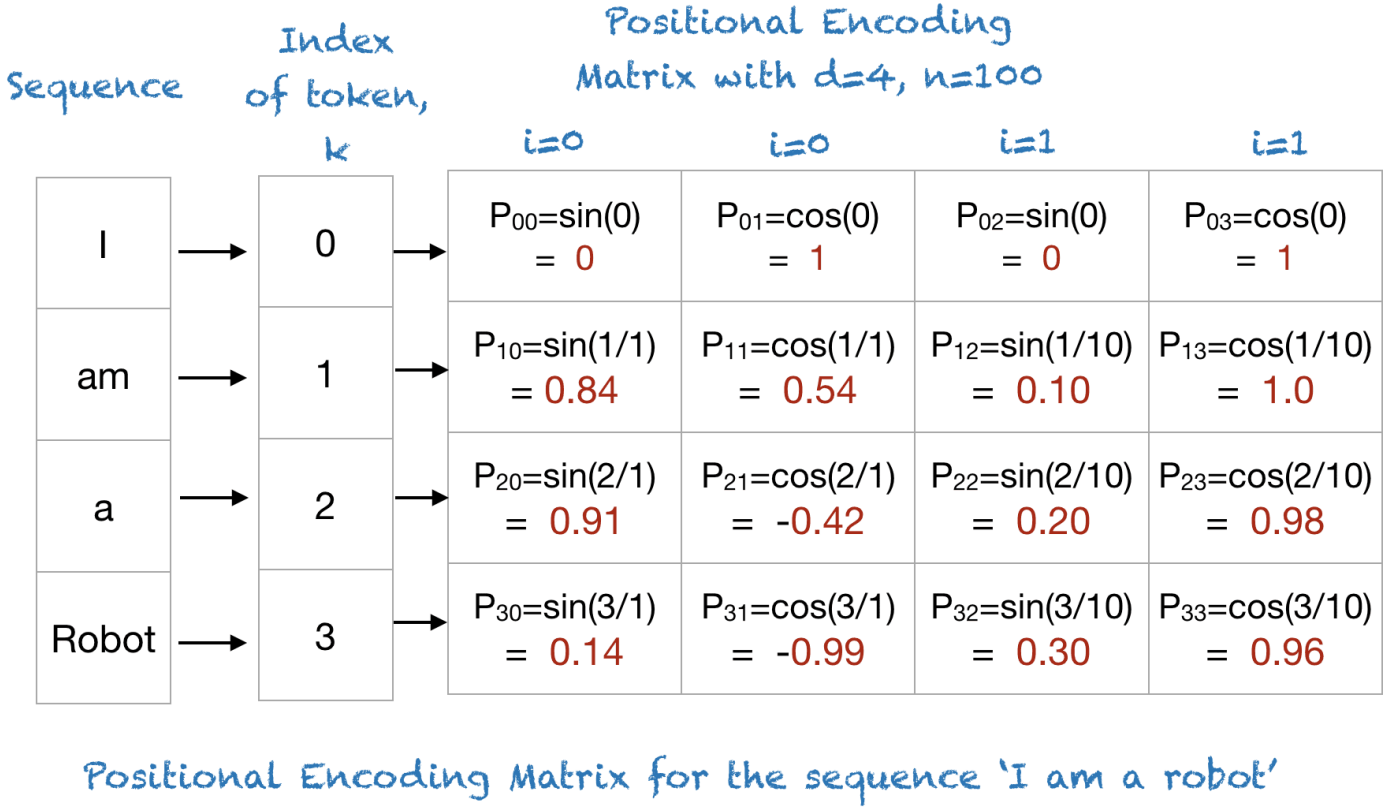

2D Embeding Space: Student vs. Lecaruer Affenity
High
High
Lectueer Affinity
Academic Journal Article
Presentation
Curriculum Design
Grant Proposal
Textbook Chapter
Recipe
Textbook Chapter
Research Project
Cooking Recipe
Student-Centric Content
Shopping List
TikTok Video
Study Playlist
Car Repair Manual
Meme
Low
Student Affinity
High
Nodes represent documents/ideas. Proximity indicates similarity

2.6 Text Data (Positional Encoding)



Positional Encoding Matrix for the sequence 'I am a robot'

2.6 Text Data (Positional Encoding)



Positional Encoding Matrix for the sequence 'I am a robot'

2.7 Text Data (All Together)

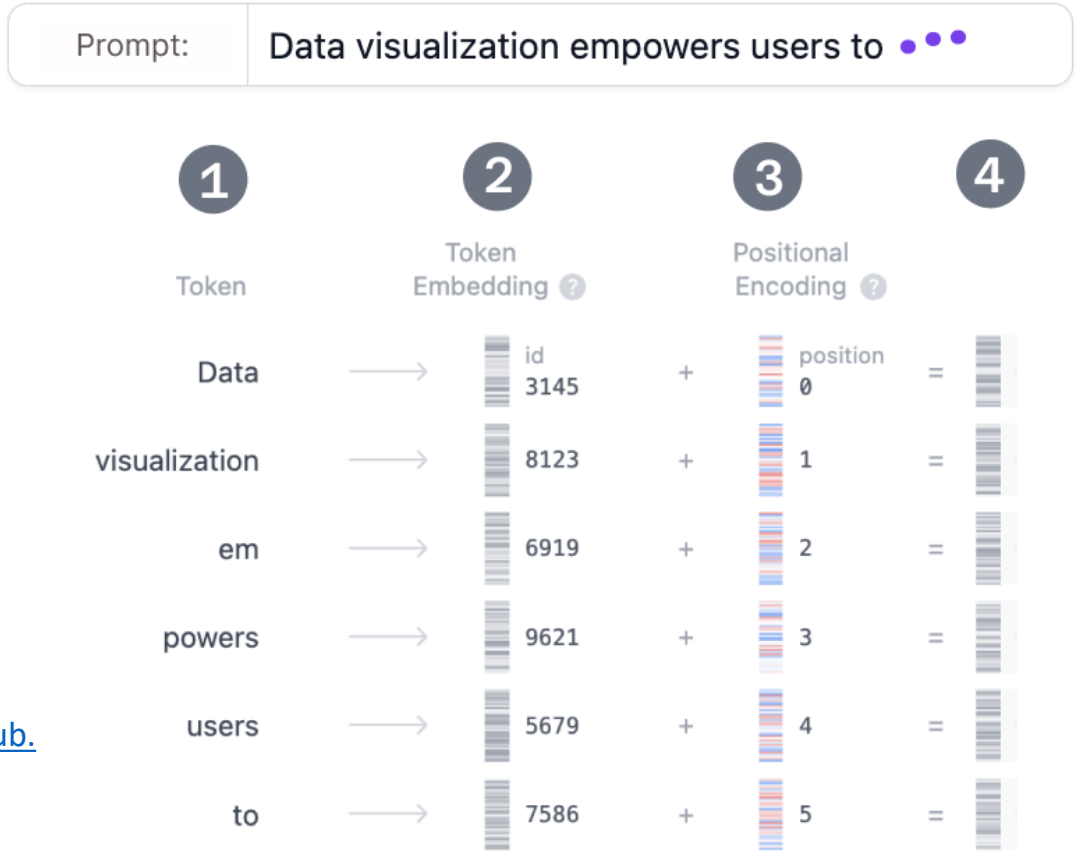


Image Credit: https://poloclub.github.io/transformer-explainer/

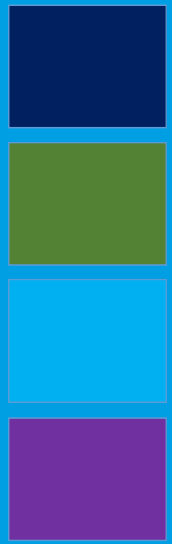
Session Aims

Introduction

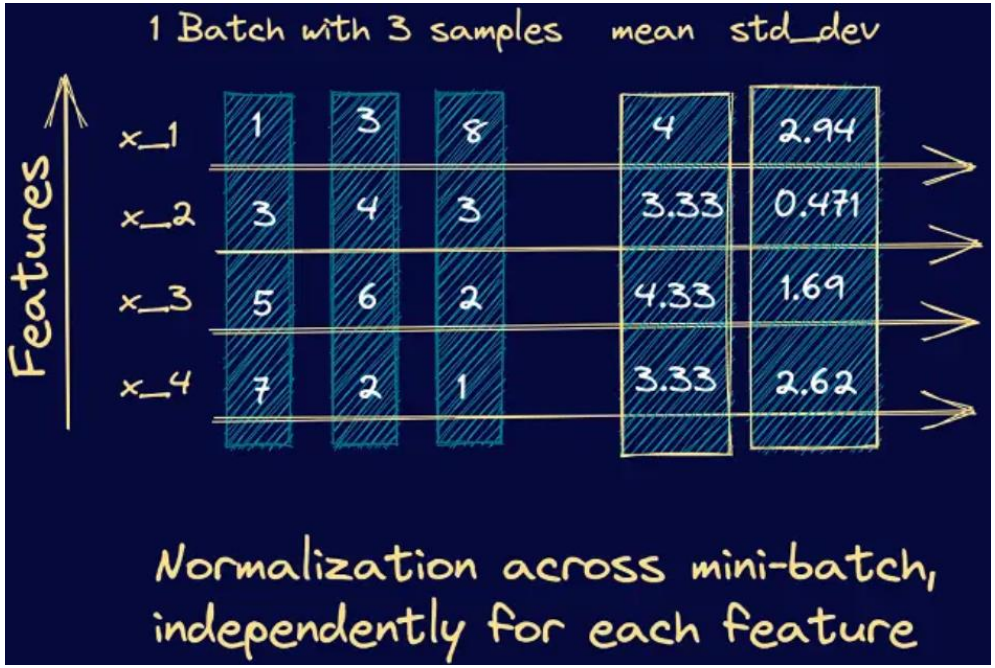
Working with “Unstructured” Data

Neural Network Layer Types

Transformers and Self-Attention



3.1 Normalisation Layers (Batch Normalisation)



$$Z = \frac{x - \mu_b}{\sigma_b}$$

where:

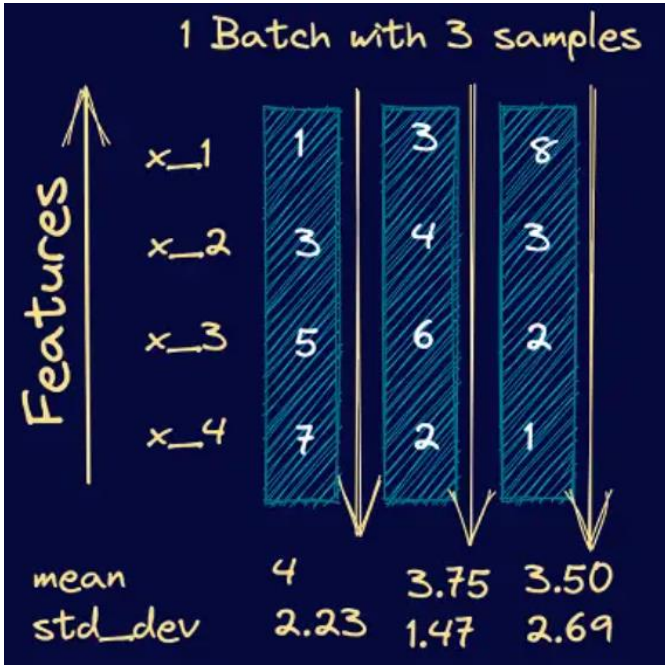
x is the output of the previous layer;

μ_b is the mean of the batch;

σ_b is the standard deviation of the batch.

Image Credit: https://www.pinecone.io/learn/batch-layer-normalization/

3.1 Normalisation Layers (Layer Normalisation)



$$Z = \frac{x - \mu_l}{\sigma_l}$$

where:

x is each output of the previous layer;

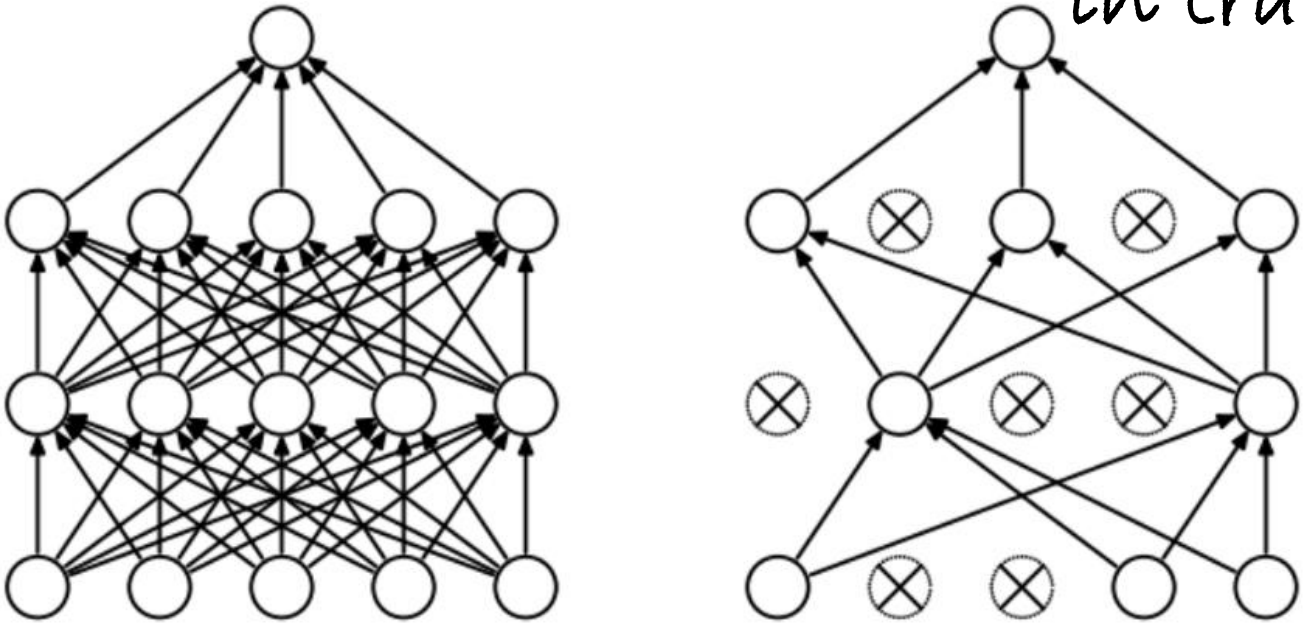
μ_l is the mean of the input sequence;

σ_l is the standard deviation of the input sequence.

Image Credit: https://www.pinecone.io/learn/batch-layer-normalization/

3.2 Dropout Layers

Only do this in training!

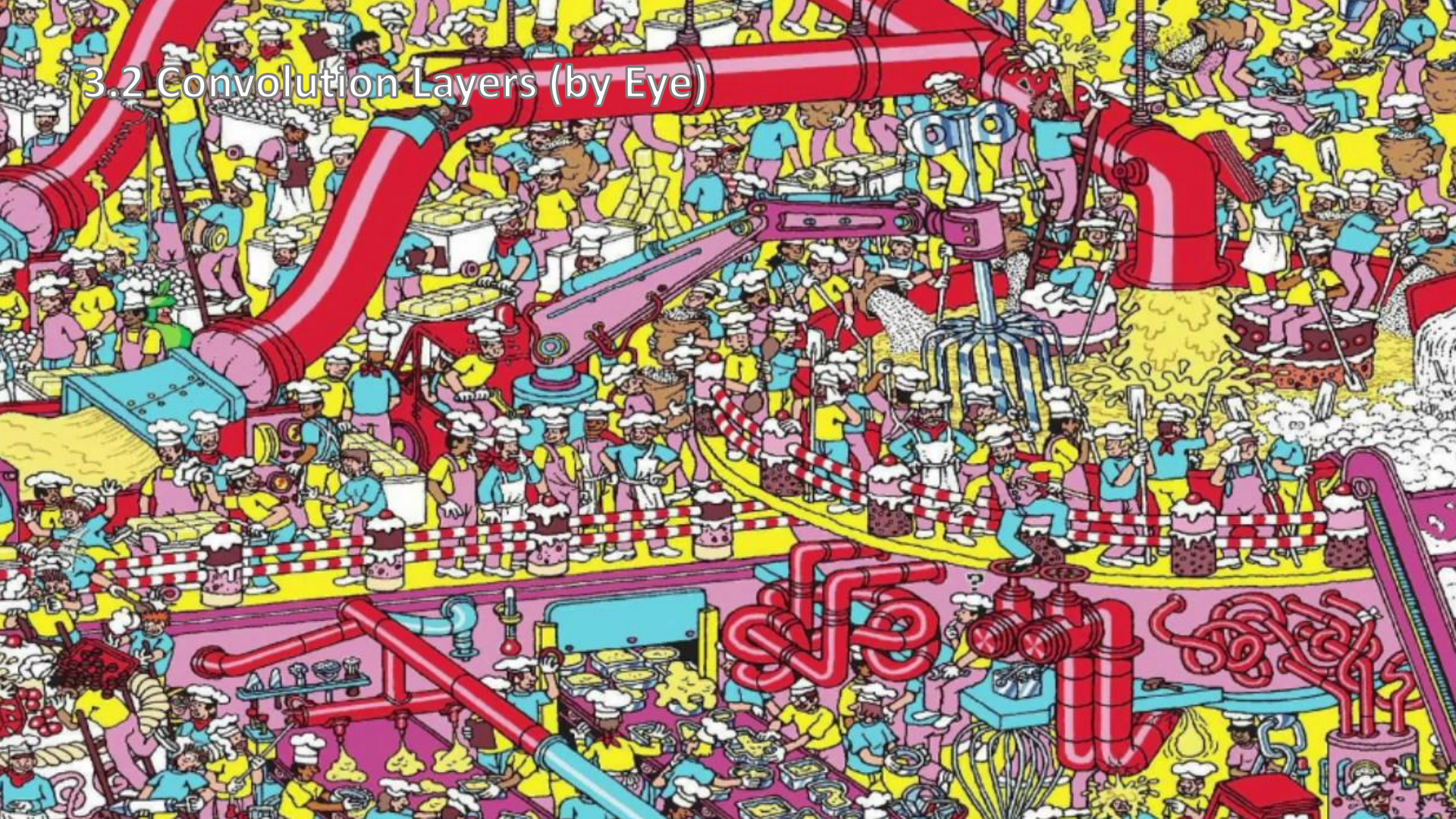


(a) Standard Neural Net

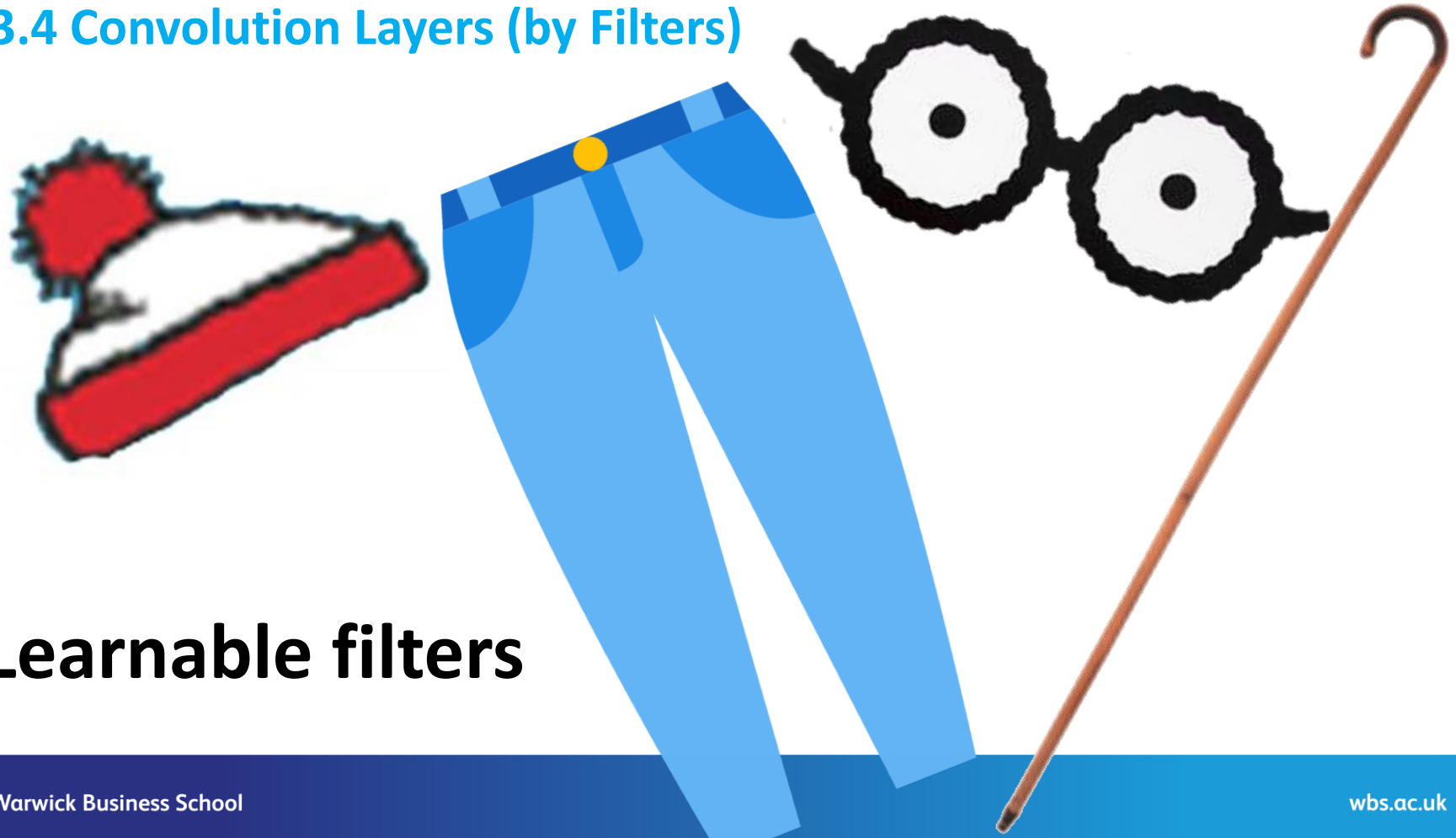
(b) After applying dropout.

3.3 Convolution Layers (by Eye)

3.2 Convolution Layers (by Eye)



3.4 Convolution Layers (by Filters)

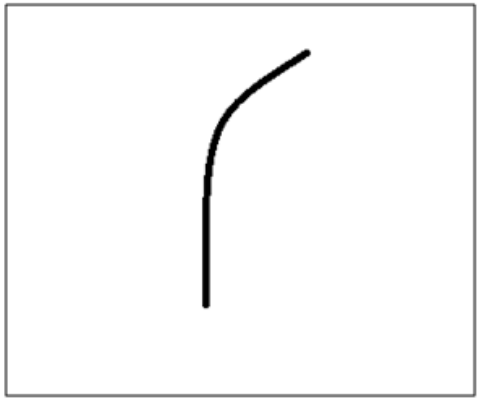


Learnable filters

3.4 Convolution Layers (by Filters)

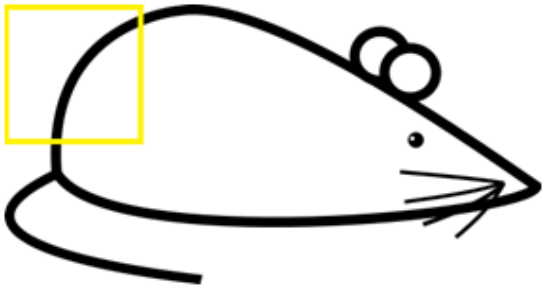
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Original image



Visualization of the filter on the image

Neilsen MA (2015). *Neural Networks and Deep Learning.* Determination Press.

3.4 Convolution Layers (by Filters)

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

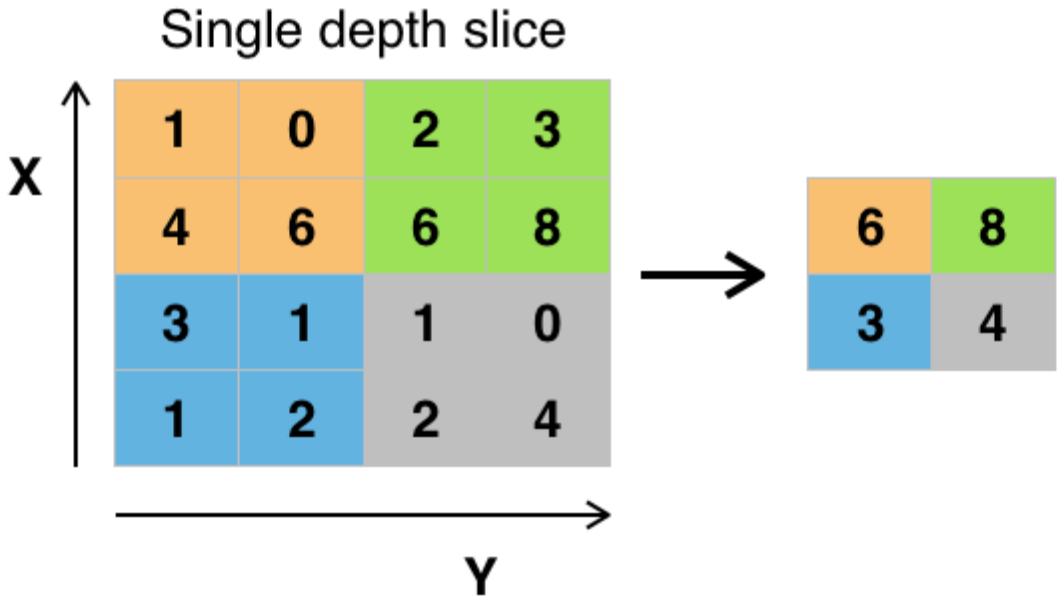
Multiplication and Summation = 0

Neilsen MA (2015). *Neural Networks and Deep Learning.* Determination Press.

3.5 Pooling Layers

- One problem with convolutions is that they make the size of the data much bigger. We go from 1-3 channels of data to n channels of data (where n is the number of kernels/filters used).
- We want to use lots of filters but we don't want to explode the feature space of the data.
- Pooling allows us to reduce the space after convolutions.

3.5 Pooling Layers



Example of Maxpool with a 2x2 filter and a stride of 2

Neilsen MA (2015). *Neural Networks and Deep Learning.* Determination Press.

3.5 Pooling Layers

- Maxpool (select only the max value from the filter size):
  - Basically treats filter activation as more of a binary-type operation … did the original filter get triggered at some point in the scan?
  - Focuses only on what you got right – the number of times the filter is activated. Good for sharp edges and details.
- Meanpool (calculate the average across the filter size):
  - Will find positive activations of the original filter, but also negative activations. How similar is the space to the filter? Good for overall similarity but smooths out the image.

3.6 Recurrent Layers/Units

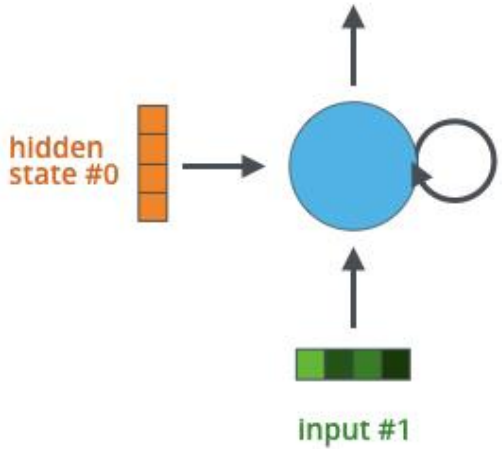
3.6 Recurrent Layers/Units

- For text problems, we want to “remember” previous inputs to help us understand the context of the next inputs. In a forecasting problem, we want to see previous values (e.g. prices) to predict the next.
- Across the full input ($x_1 \rightarrow x_N$), each time a recurrent unit process an input (x_t) we update the hidden state (memory) to reflect new information from the new input.
- Effectively this hidden state is the memory of the network, the context required to interpret each input. It is a numerical representation (a vector) of aspects of the problem. The vector will be different for every dataset / problem, and means that the influence of the hidden state means two RNNs have different “meaning” for the same input.

Recurrent Neural Network

Time step #1:

An RNN takes two input vectors:



Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

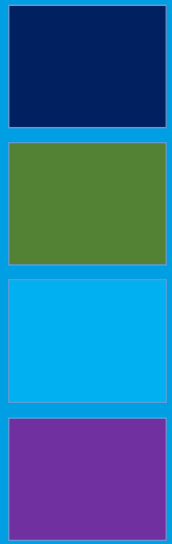
Session Aims

Introduction

Working with “Unstructured” Data

Neural Network Layer Types

Transformers and Self-Attention



4.1 Transformer Inputs

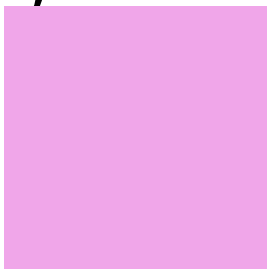
- **Input**: "Please complete the module review"
- **Normalisation & splitting**: "[CLS], please, complete, the, module, review, [SEP]"
- **Tokenisation (numeric)**: "123, 456, 789, 1011, 1112, 1213, 1314, 1415"
- **Embeddings**: "[0.78, 1.23, …, 7.89], [6.54, -1.23, …, 3.21], …, [-1.76, 1.23, … 0.99]"
- **Positions**: [0, 1, 2, 3, 4, 5, 6]
- **Positional encoding**: [[0.11, 5.67, …, 9.87], [5.56, -3.21, …, 1.23], …, [-6.71, 3.21, … 9.90]]
- **Transformed input**: Embeddings ⊙ Positional Encoding

4.2 Embeddings Are NOT All You Need

- However, this alone is not enough. In language, context is king!

4.3 Self-Attention Layers

the scary, blue monster hid under the child's pink bed



4.3 Self-Attention Layers

Query: *I am a noun looking for an adjective to add context to me*

the scary, blue monster hid under the child's pink bed

4.3 Self-Attention Layers

Key: *We are adjectives. We like to add context to nouns.*

the scary, blue monster hid under the child's pink bed

4.3 Self-Attention Layers

Semantically close

Positionally close

the scary, blue monster hid under the child's pink bed

Further away

4.3 Self-Attention Layers

Value: *How should I change the values of the embedding?*

the scary, blue monster hid under the child's pink bed

4.4 Multi-headed Self Attention

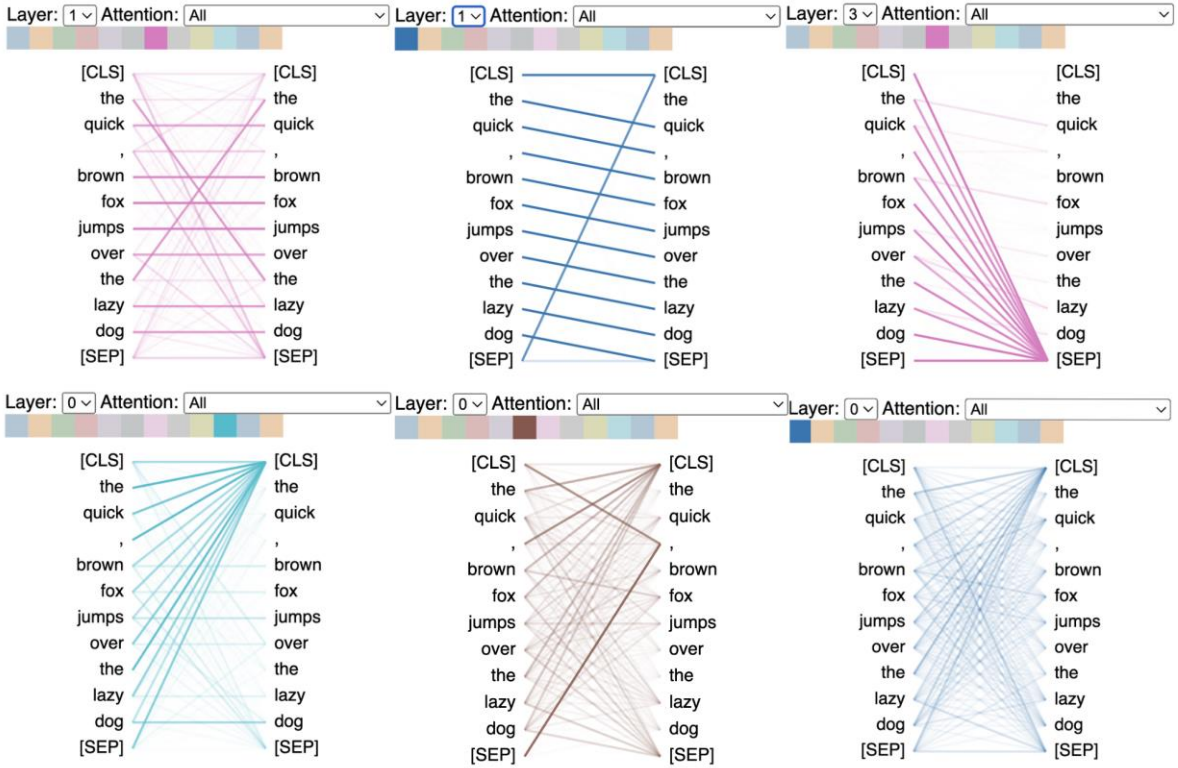


Image Credit: https://www.comet.com/site/blog/explainable-ai-for-transformers/

4.5 Full Transformer Architecture (DeepSeek)

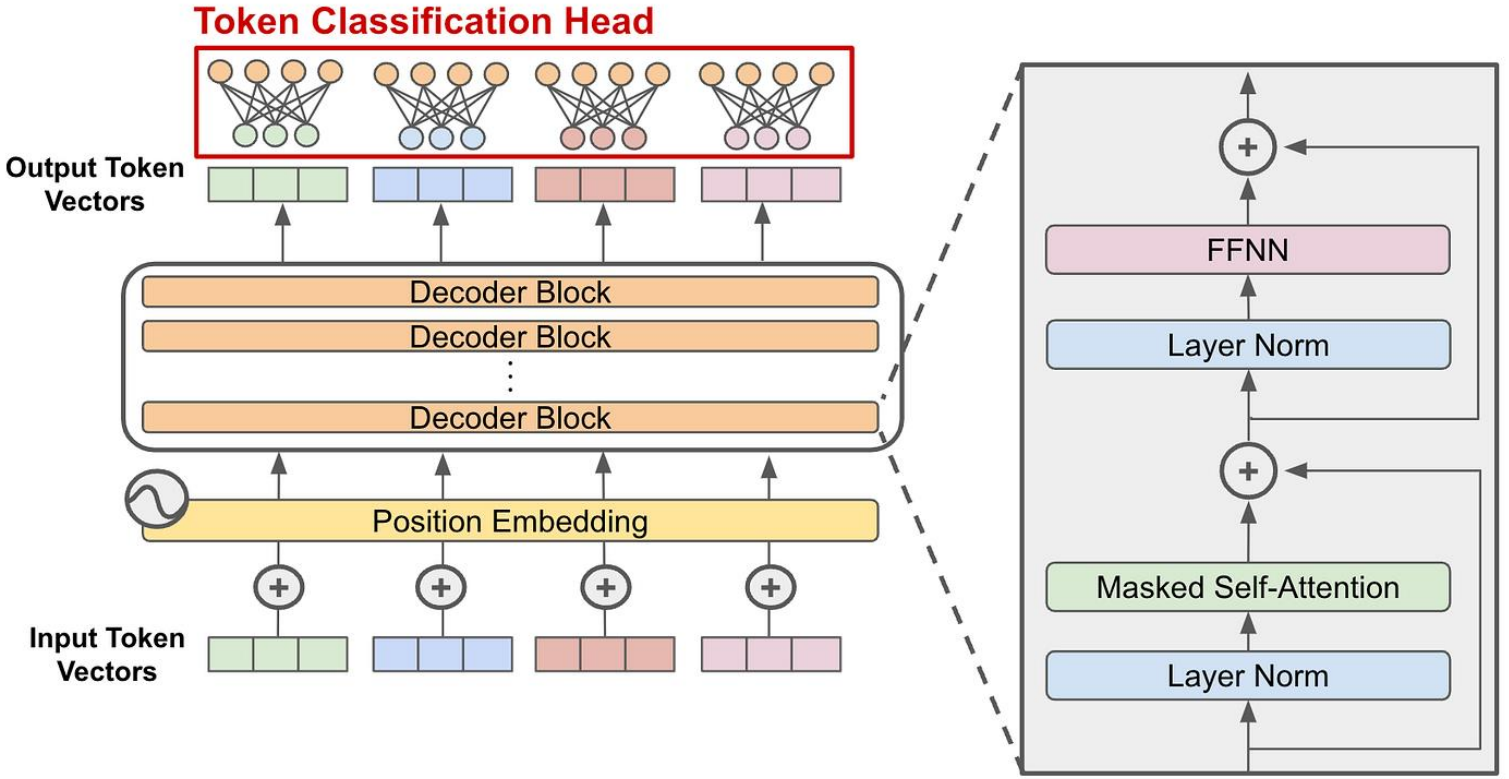
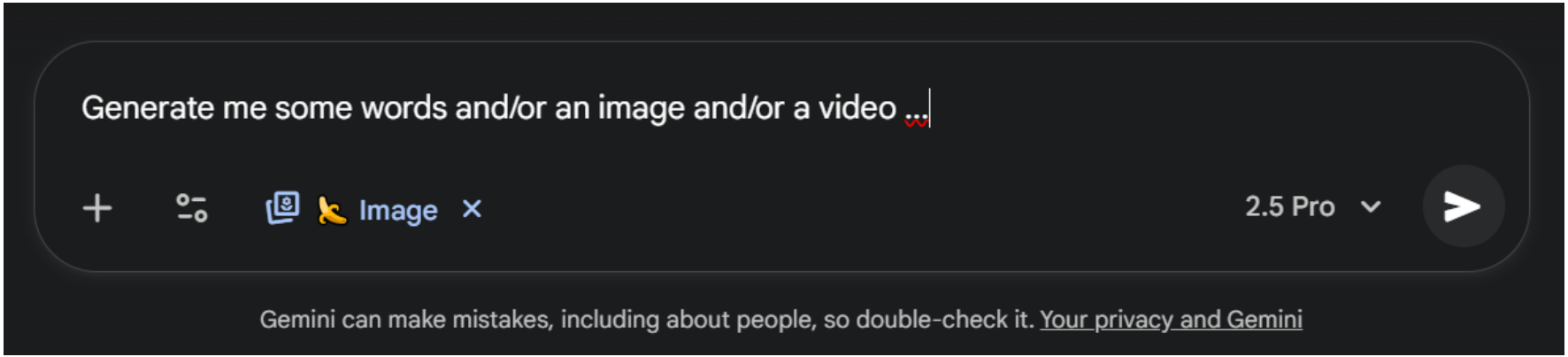


Image Credit: https://medium.com/@tanalpha-aditya/deepseek-coder-when-the-llm-meets-programming-better-than-gpt-3-5-054cf85e3493

4.6 Next Week


Generate me some words and/or an image and/or a video ...
Image
2.5 Pro
Gemini can make mistakes, including about people, so double-check it. Your privacy and Gemini

Further Reading

- 3Blue1Brown (2024). *Transformers (how LLMs work) explained visually | DL5*. https://www.youtube.com/watch?v=wjZofJX0v4M.
- Alammar J and Grootendorst M (2024). *Hands-On Large Language Models*. Sebastopol, CA: O'Reily.
- Alammar J (n.d.). *The Illustrated Transformer.* https://jalammar.github.io/illustrated-transformer/.
- **Goodfellow I, Bengio Y and Courville A (2016). *Deep Learning*. The MIT Press: Cambridge, MA.**
- Karpathy A (2023). *Intro to Large Language Models*. https://www.youtube.com/watch?v=zjkBMFhNj_g.
- Pajankar A and Joshi A (2022). *Hands-on machine learning with Python: implement neural network solutions with Scikit-learn and PyTorch.* Apress: Berkeley, CA.
- Vaswani A *et al.* (2017). Attention is all you need. In: *31st International Conference on Neural Information Processing Systems (NIPS '17)*. Red Hook, NY, USA, 6000–6010.