TEXT STYLE TRANSFER

Project Report

Project X

at

Community of Coders, Veermata Jijabai Technological Institute

September 2023

ACKNOWLEDGEMENT

Firstly, we would like to thank our mentor, ***Labib Asari*** for his commendable role as our mentor throughout this project. His timely guidance, expertise in the domain and insightful feedback were crucial in shaping our project.

We are also grateful to VJTI's COC for continuing the tradition of their seniors and providing this golden opportunity that made our project possible.

Druhi Phutane
druhiphutane@gmail.com

Warren Jacinto
warrenjacinto@gmail.com

Yashvi Gala
yashvigala0904@gmail.com

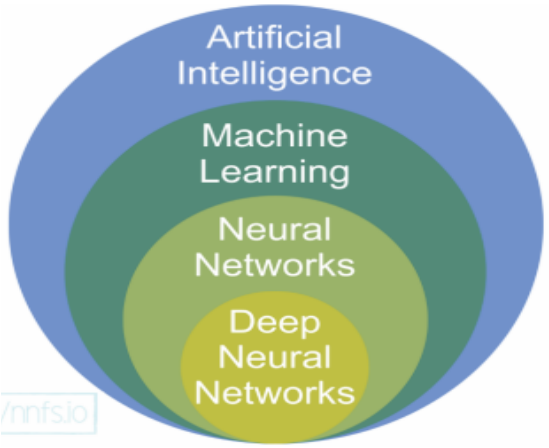
TABLE OF CONTENTS

OVERVIEW

Neural networks, inspired by the human brain, have emerged as the cornerstone of modern machine learning and deep learning. They are composed of layers of interconnected nodes, or neurons, that work collaboratively to learn intricate patterns and relationships within data. Over time, neural networks have evolved from simple feedforward architectures to complex, versatile models capable of understanding and generating human language.

The Transformer architecture, introduced in the seminal paper "Attention Is All You Need," marks a pivotal moment in natural language processing (NLP). Unlike its predecessors, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), the Transformer model relies on self-attention mechanisms to capture contextual information from input sequences more effectively. This innovation eradicates the need for sequential processing, resulting in faster training and improved performance.

Text style transfer is a captivating application of deep learning and NLP, where the Transformer architecture plays a pivotal role. At its core, text style transfer involves the transformation of text from one style or tone to another while preserving its underlying content. It allows us to, for example, convert formal language into a casual conversational style, or vice versa, without altering the meaning.



Linear Algebra

1. Python Lists and Arrays

A Python list is defined by comma-separated objects contained in brackets. A linear array, also called a 1-dimensional array, is the simplest example of an array, and in plain Python, this would be a list.

```
list_of_lists = [[4,2,3],
                 [5,1]]
```

A list of lists is homologous if each list along a dimension is identically long, and this must be true for each dimension. In the case of the list shown above, it's a 2-dimensional list.

The first dimension's length is the number of sublists in the total list (2). The second dimension is the length of each of those sublists (3, then 2). In the above example, when reading across the "row" dimension (also called the second dimension), the first list is 3 elements long, and the second list is 2 elements long — this is not homologous and, therefore, cannot be an array.

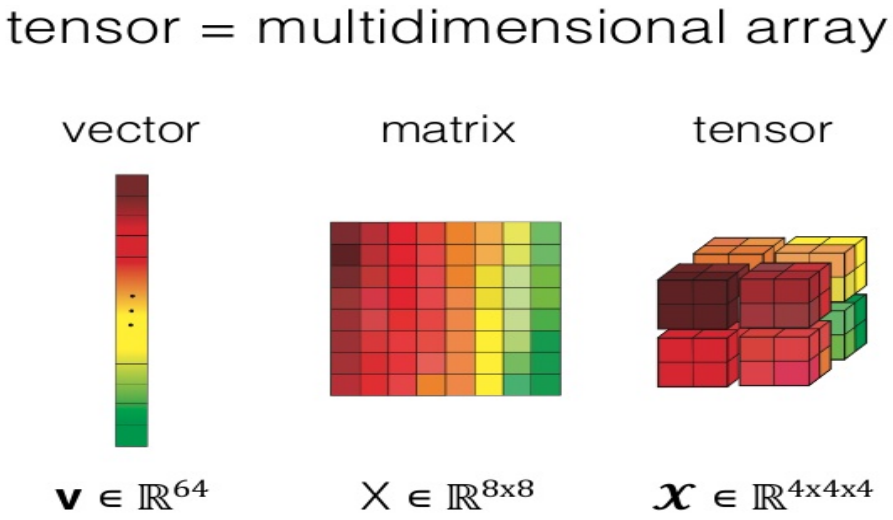
2. Matrix

A **matrix** is pretty simple. It's a rectangular array. It has columns and rows. It is two dimensional. So a matrix can be an array (a 2D array).If m and n are positive integers, that is m, n $\in \mathbb{N}$ then the m×n matrix contains m*n numbers, with m rows and n columns.T

```
list_matrix_array = [[4,2],
                     [5,1],
                     [8,2]]
```

3.Tensor

A tensor object is an object that can be represented as an array. Tensors are a generalisation of scalars (0D tensors), vectors (1D tensors), and matrices (2D tensors) to higher dimensions.



4.Vector

In computer science and mathematics, a vector refers to an ordered collection of elements, often numbers, arranged in a specific sequence. Vectors are used to represent various data structures and concepts, and they can be one-dimensional (1D), two-dimensional (2D), three-dimensional (3D), or higher-dimensional, depending on the context.

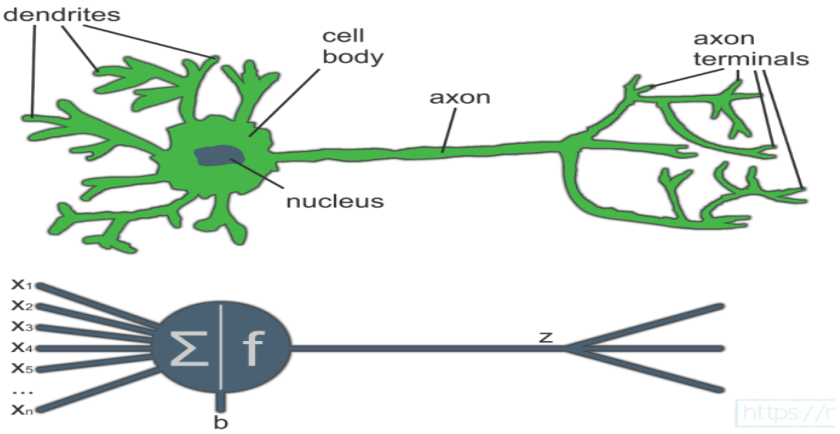
5.Scalar Multiplication/Dot Product

Scalars are single numbers and are an example of a 0th-order tensor. The notation $x \in \mathbb{R}$ states that x is a scalar belonging to a set of real-values numbers, $\mathbb{R}$. Dot product is used to calculate the cosine of the angle between two vectors and find the projection of one vector onto another.

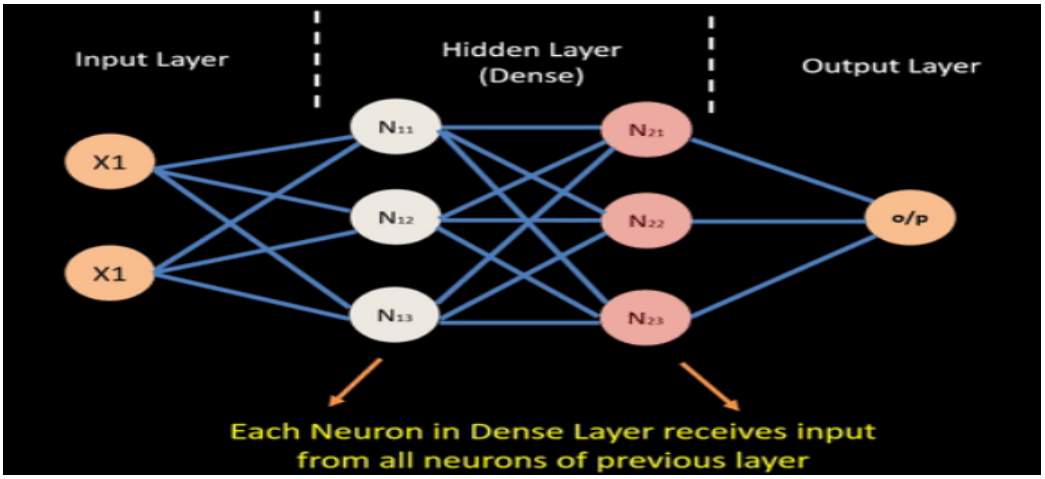
```
a = [1, 2, 3]
b = [2, 3, 4]

dot_product = a[0]*b[0] + a[1]*b[1] + a[2]*b[2]
>>> 20
```

Neural Networks and Deep Learning

"Artificial" neural networks are inspired by the organic brain, translated to the computer. It's not a perfect comparison, but there are neurons, activations, and lots of interconnectivity, even if the underlying processes are quite different. A single neuron by itself is relatively useless, but, when combined with hundreds or thousands (or many more) of other neurons, the interconnectivity produces relationships and results that frequently outperform any other machine learning method.



Dense layers consist of interconnected neurons. In a dense layer, each neuron of a given layer is connected to every neuron of the next layer. Each connection between neurons has a weight associated with it, which is a trainable factor of how much of this input to use, and this weight gets multiplied by the input value. Once all of the inputs·weights flow into our neuron, they are summed, and a bias, another trainable parameter, is added. The purpose of the bias is to offset the output positively or negatively, which can further help us map more real-world types of dynamic data.

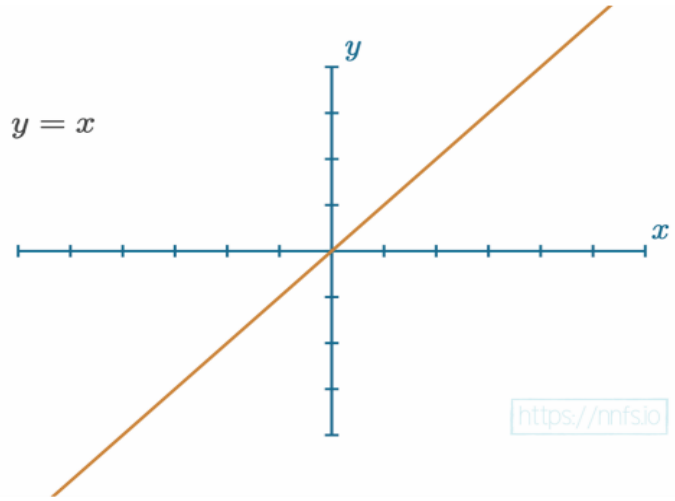


Activation Functions

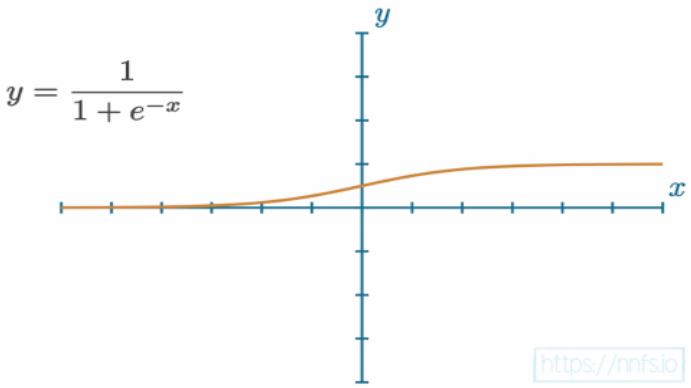
In general, your neural network will have two types of activation functions. The first will be the activation function used in hidden layers, and the second will be used in the output layer. Usually, the activation function used for hidden neurons will be the same for all of them, but it doesn't have to.

Activation Functions introduce non-linearity.

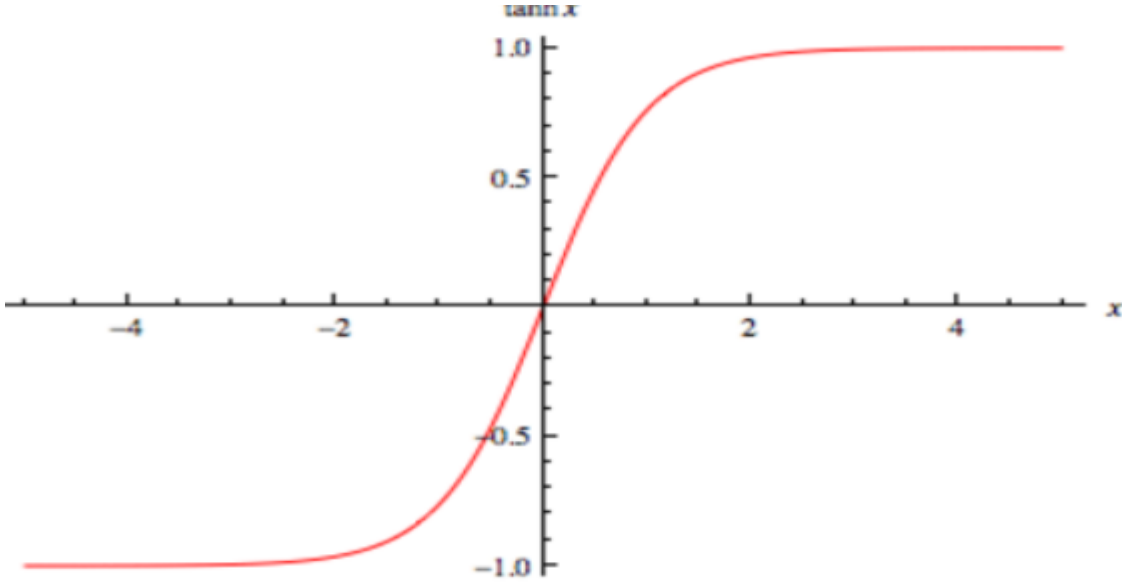
1. *Linear Activation Function*–A linear function is simply the equation of a line. It will appear as a straight line when graphed, where y=x and the output value equals the input.



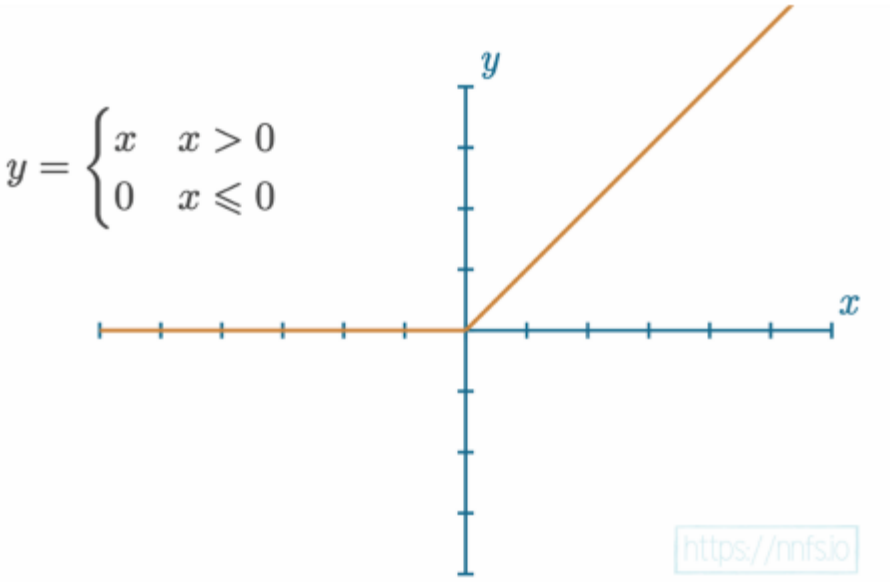
2. *Sigmoid Function*— This function returns a value in the range of 0 for negative infinity, through 0.5 for the input of 0, and to 1 for positive infinity. The output from the Sigmoid function, being in the range of 0 to 1, also works better with neural networks — especially compared to the range of the negative to the positive infinity — and adds nonlinearity.



3. Tanh Activation Function– The tanh function maps the input to a range between -1 and 1, providing a zero-centred output. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

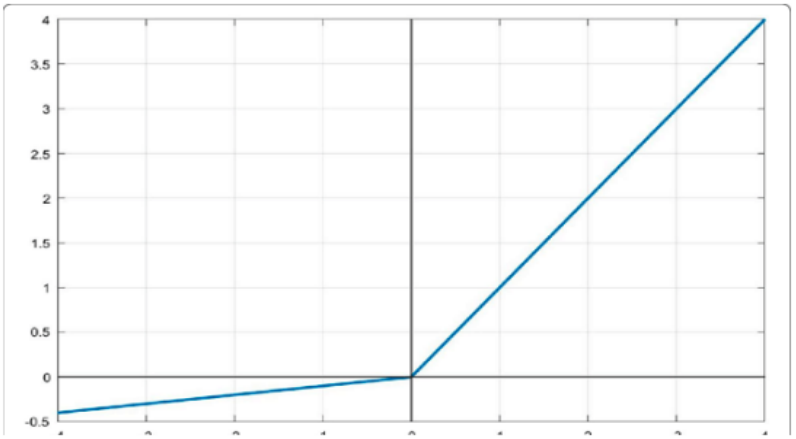


4. Relu Activation Function–The rectified linear activation function is simpler than the sigmoid. It's quite literally y=x, clipped at 0 from the negative side. If x is less than or equal to 0, then y is 0 — otherwise, y is equal to x



5. Leaky Relu Activation Function– Leaky ReLU is a variant of ReLU that allows a small gradient for negative inputs, preventing "dying ReLU" problems in training.

max(0.01*x,x)



Loss Calculation

Backpropagation is a supervised learning technique that adjusts the model's parameters (weights and biases) to minimise the error between predicted and actual output.
Forward Pass: The input data is passed through the neural network to make predictions.

Categorical cross-entropy is explicitly used to compare a “ground-truth” probability (y or “targets”) and some predicted distribution (y-hat or “predictions”). It is also one of the most commonly used loss functions with a softmax activation on the output layer.

Backward Pass : The gradient of the loss with respect to the model's parameters is computed through the chain rule. This gradient represents how much each parameter should be adjusted to minimise the loss.

Optimization is the process of finding the best set of model parameters that minimises the loss function. Common optimization techniques include Adam, RMSprop etc.

Building a Neural Network From Scratch To Make A Cat/Non-Cat Classifier

We constructed a cat v/s non cat binary classifier by using a deep neural network using Python and NumPy. The neural network architecture is defined with multiple layers, activation functions, He initialization, and backpropagation. The primary goal is to classify images into two categories: "cat" and "non-cat."

- He initialization is applied to set initial weights.
- Activation Functions: The neural network utilises two activation functions:
    1. Sigmoid
    2. Relu
- linear_forward(): perform forward propagation for one layer
- linear_activation_forward(): applies activation function to the output of forward propagation.
- L_model_forward() produces forward propagation results for the entire neural network.
- compute_cost function performs loss computation.
- L_model_backward(): orchestrates the entire backward pass.
- update_parameters function adjusts the weights and biases based on computed gradients and learning rate.

Image data for training and testing is loaded and preprocessed.

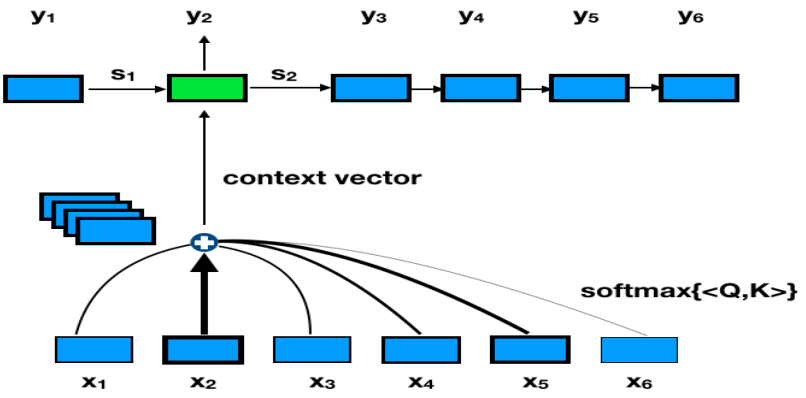
- Images are normalised to have values between 0 and 1.
- L_layer_model(): it orchestrates the training process. It iteratively performs forward and backward passes, updating model parameters.

The test accuracy achieved by the neural network on the provided image dataset is approximately 74.00%.

ATTENTION MECHANISM AND TRANSFORMER ARCHITECTURE

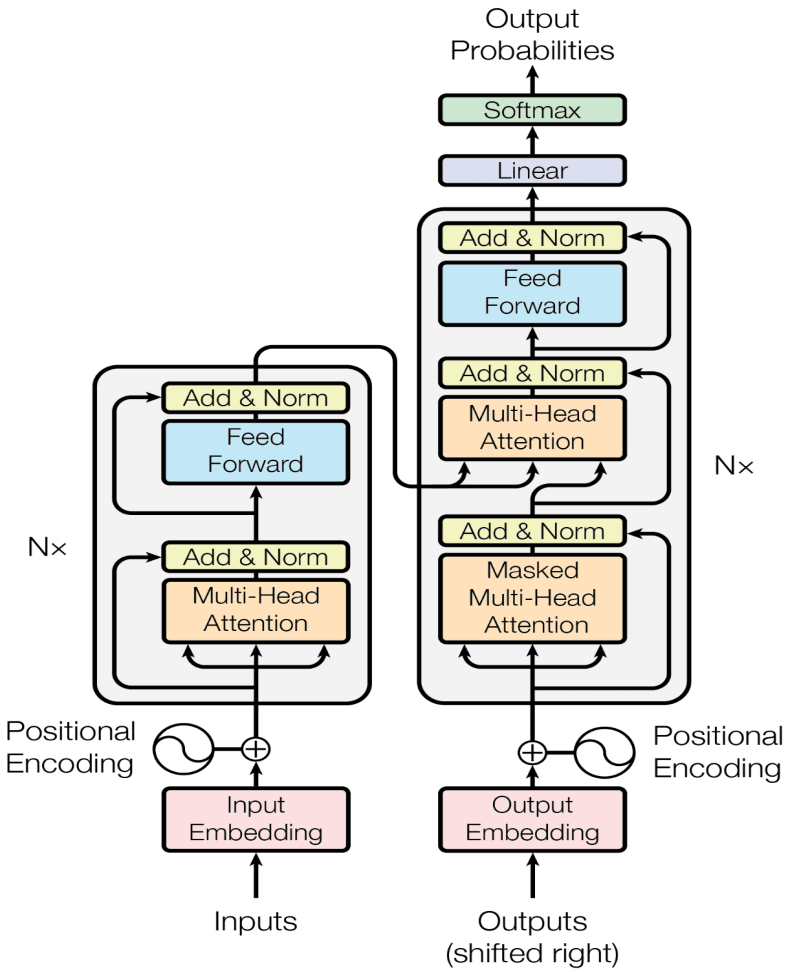
The attention mechanism allows models to focus on different parts of input data when making predictions or generating output, which is especially valuable in tasks involving sequences, such as language understanding, translation, and text generation. The core idea of the attention mechanism is to compute weighted combinations of elements in an input sequence, with the weights determined dynamically based on the importance of each element to the current prediction or context. These weights, often referred to as "attention scores," indicate how much attention or focus should be given to each element in the input sequence. In essence, the attention mechanism assigns a relevance score to each input element, and these scores are used to compute a weighted sum, creating a context vector for the model to work with.

	SELF-ATTENTION MECHANISM	MULTI-HEAD ATTENTION MECHANISM
DEFINITION	Self-Attention refers to the ability of the transformer model to attend to different parts of the input sequence when making prediction.	Multi-Head attention is a module for attention mechanism which runs through an attention mechanism several time in parallel.
OPERATES ON	self-attention operates within a single sequence Weighs words within the same sequence against each other to capture context and dependencies.	multi-attention operates between multiple sequences. Weighs words from different sequences to capture relationships between them, commonly used in tasks involving multiple sequences, like translation.

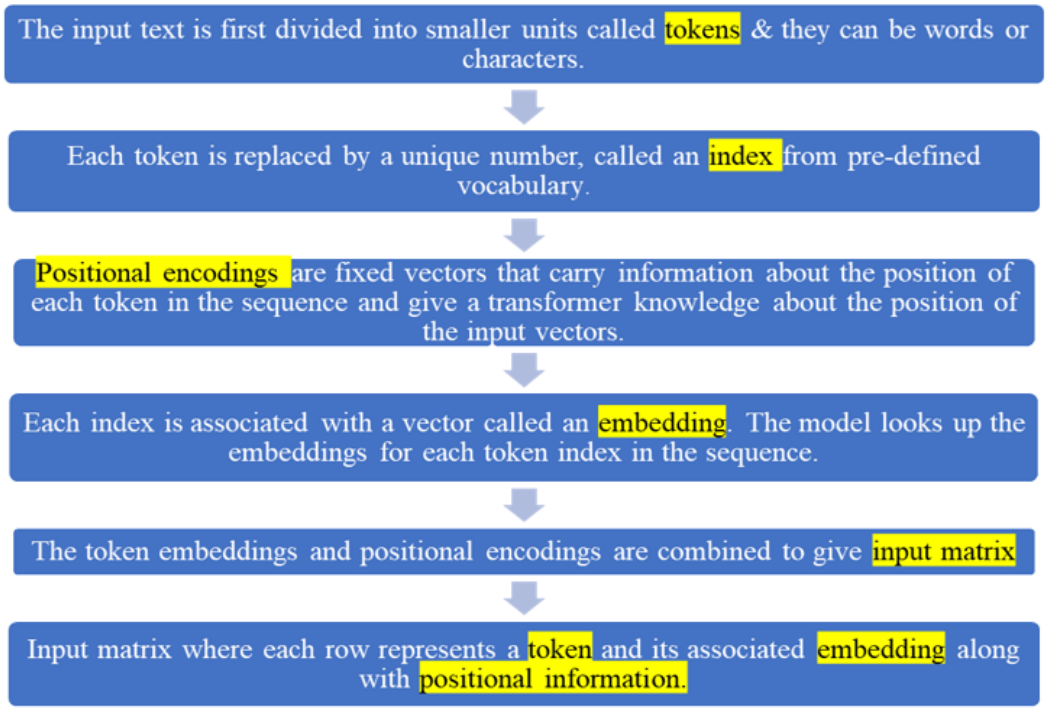


Why are Transformers significant?

- Transformers excel at modelling sequential data, such as natural language.
- Unlike the recurrent neural networks (RNNs), Transformers are parallelizable. This makes them efficient on hardware like GPUs and TPUs. The main reason is that Transformers replace recurrence with attention, and computations can happen simultaneously.
- Unlike RNNs or convolutional neural networks (CNNs) , Transformers are able to capture distant or long-range contexts and dependencies in the data between distant positions in the input or output sequences. Attention allows each location to have access to the entire input at each layer, while in RNNs and CNNs, the information needs to pass through many processing steps to move a long distance, which makes it harder to learn.
- Transformers make no assumptions about the temporal/spatial relationships across the data.



Preparation of Data to be fed to encoder:



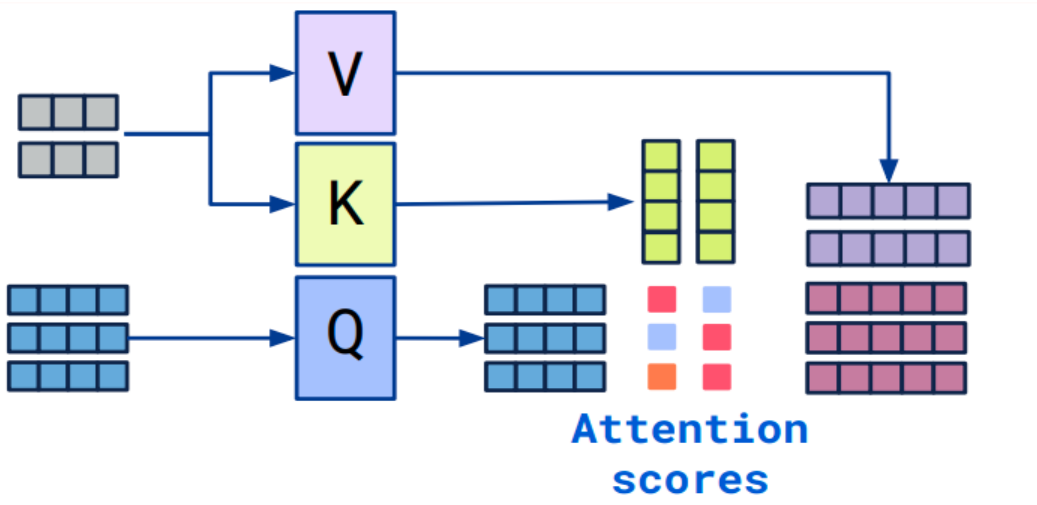
Transformers use multiple attention mechanisms which helps them capture context well:

1. BaseAttention: The `BaseAttention` class serves as a base class for different types of attention mechanisms in the Transformer. It contains three sub-layers:
MultiHeadAttention (self.mha): This is a multi-head self-attention mechanism that allows the model to weigh the importance of different parts of the input sequence when processing a particular token.
LayerNormalization (self.layernorm): This layer normalises the output of the self-attention mechanism to stabilise the training process.
Add (self.add): This layer adds the output of the multi-head attention to the input, helping the model capture residual information.

2. CrossAttention (inherits from BaseAttention): Cross-attention, also known as inter-attention, is a mechanism where tokens in one sequence (e.g., the query sequence) attend to tokens in another sequence (e.g., the context or key sequence). Cross-attention is often used in tasks where the model needs to relate information between two different sets of data, such as machine translation or question-answering. It uses the multi-head attention mechanism to compute the weighted sum of the context sequence (`key` and `value`) based on the query sequence `x`.

3. GlobalSelfAttention (inherits from BaseAttention): The GlobalSelfAttention layer is a self-attention mechanism where the model attends to the entire input sequence, not just a local context. This is different from the standard self-attention where the model only attends to nearby tokens. It calculates attention scores between every token in the sequence, allowing the model to capture global dependencies.

4. CausalSelfAttention(inherits from BaseAttention): The CausalSelfAttention layer is used for autoregressive tasks, where tokens can only attend to previous tokens but not to future tokens. It applies self-attention with a causal (masked) attention mask, ensuring that each token can only attend to previous tokens.



Encoder: The encoder processes the input embeddings (containing token and positional information) by iteratively applying self-attention and feedforward layers. The self-attention mechanism allows the model to capture dependencies between different tokens within the same sequence, and the feedforward layers apply non-linear transformations. The output of the encoder is a sequence of context-aware representations, capturing both local and global dependencies within the input sequence.

Decoder: Like the encoder, the decoder consists of multiple layers. In addition to the self-attention mechanism and position-wise feedforward neural network, the decoder incorporates cross-attention layers. The decoder processes the target sequence, generating token-by-token predictions. At each step, it predicts the next token based on the previously generated tokens and the context provided by the encoder's output.

TRANSLATION FROM SHAKESPEARE TO MODERN ENGLISH

We constructed a transformer using tensorflow and used **4 layers**, set the **embedding dimensions as 256**, used **8 attention heads** and set the **dimensions of the feedforward network as 512.**

```
sentence = 'None but the king?'
ground_truth = 'Only the king?'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:          : None but the king?
Prediction      : nothing but the king ?
Ground truth    : Only the king?
```

```
sentence = ' A sample to the youngest, to the more mature  A glass that feated them, and to the graver  A child that guided dotards; '
ground_truth = 'He was an example to the youngest, to the full-grown a model for their own behavior, and seemed to serious observers like a child leading old people.'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:          :  A sample to the youngest, to the more mature  A glass that feated them, and to the graver  A child that guided dotards;
Prediction      : a sam - faced to the youngest daughter , to the more appropriate mirror that gave birth to a child who slapseed a child .
Ground truth    : He was an example to the youngest, to the full-grown a model for their own behavior, and seemed to serious observers like a child leading old people.
```

```
sentence = 'And as for thee my king, i pray that thou may be victorious in thy attempts'
ground_truth = ""

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:         : And as for thee my king, i pray that thou may be victorious in thy attempts
Prediction     : and as for you , my king , i beg that you can be victorious in your attempt
Ground truth   :
```

```
sentence = 'For you, Posthumus,  So soon as I can win the offended king,  I will be known your advocate: '
ground_truth = 'As for you, Posthumus, as soon as I can calm the upset king, I will speak in your defense.'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:         : For you, Posthumus,  So soon as I can win the offended king,  I will be known your advocate:
Prediction     : you , posthumus , as soon as i can win the offend king , i will be known as your servant .
Ground truth   : As for you, Posthumus, as soon as I can calm the upset king, I will speak in your defense.
```

On simply **increasing the dimensions of the word embeddings from256 to 512** we observed better results:

```
sentence = 'None but the king?'
ground_truth = 'Only the king?'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:         : None but the king?
Prediction     : only the king ?
Ground truth   : Only the king?
```

```
sentence = ' A sample to the youngest, to the more mature  A glass that feated them, and to the graver  A child that guided d
ground_truth = 'He was an example to the youngest, to the full-grown a model for their own behavior, and seemed to serious ob

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:         :  A sample to the youngest, to the more mature  A glass that feated them, and to the graver  A child that guided dotards;
Prediction     : he is an example for the youngest , to the full - grown a model of their own behavior ; and to the graver grown child leading old pe
ople .
Ground truth   : He was an example to the youngest, to the full-grown a model for their own behavior, and seemed to serious observers like a child le
ading old people.
```

```
sentence = 'And as for thee my king, i pray that thou may be victorious in thy attempts'
ground_truth = ""

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:         : And as for thee my king, i pray that thou may be victorious in thy attempts
Prediction     : and as for you , my king , i pray that you can be victorious in your attempts
Ground truth   :
```

```
sentence = 'For you, Posthumus,  So soon as I can win the offended king,  I will be known your advocate: '
ground_truth = 'As for you, Posthumus, as soon as I can calm the upset king, I will speak in your defense.'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:         : For you, Posthumus,  So soon as I can win the offended king,  I will be known your advocate:
Prediction     : as for you , posthumus , as soon as i can win the offend king , i will be known to you .
Ground truth   : As for you, Posthumus, as soon as I can calm the upset king, I will speak in your defense.
```

STYLE TRANSFORMER

For all of the sentences in a single dataset D_i , they share some specific characteristic (e.g. they are all the positive reviews for a specific product), and we refer to this shared characteristic as the style of these sentences. In other words, a style is defined by the distribution of a dataset. Suppose we have K different datasets D_i, then we can define K different styles, and we denote each style by the symbol s^i. . The goal of style transfer is that: given an arbitrary natural language sentence x and a desired style $\hat{s}$, rewrite this sentence to a new one $\hat{x}$ and preserve the information in original sentence x as much as possible.

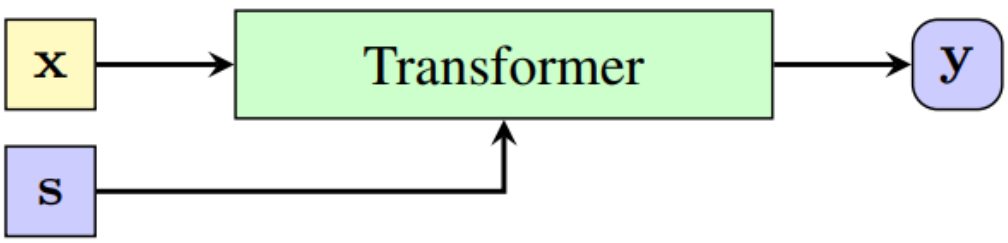
Explicitly, for an input sentence x = (x_1, x_2, x_3 ...,x⊠), the Transformer encoder Enc(x; θE) maps inputs to a sequence of continuous representations z = (z_1,z_2 ...,z⊠). And the Transformer decoder Dec(z; θD) estimates the conditional probability for the output sentence y = (y_1, y_2, ...,z⊠) by auto-regressively factoring it as:

$$p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{m} p_\theta(y_t|\mathbf{z}, y_1, \ldots, y_{t-1})$$

At each time step t, the probability of the next token is computed by a softmax classifier:

$$p_\theta(y_t|\mathbf{z}, y_1, \ldots, y_{t-1}) = \mathrm{softmax}(\mathbf{o}_t)$$

where o⊠ is a logit vector outputted by a decoder network. To enable style control in the standard Transformer framework, we add an extra style embedding as input to the Transformer encoder Enc(x, s; θE).



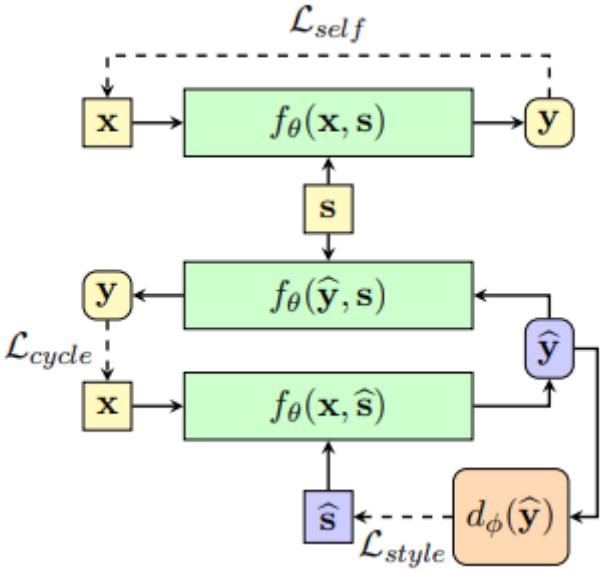
Suppose we use x and s to denote the sentence and its style from the dataset D. Because of the absence of the parallel corpora, we can't directly obtain the supervision for the case $f_\theta(x,\hat{s})$ where $s \neq \hat{s}$. Therefore, we introduce a discriminator network to learn this supervision from the non parallel corpora.

The intuition behind the training of the discriminator is based on the assumption below: As we mentioned above, we only have the supervision for the case $f_\theta(x, s)$. In this case, because the input sentence x and chosen style s both come from the same dataset D, one of the optimum solutions, in this case, is to reproduce the input sentence. Thus, we can train our network to reconstruct the input in this case.

In the case of $f_\theta(x, s)$ where $s \neq \hat{s}$, we construct supervision in two ways.

1) For the content preservation, we train the network to reconstruct original input sentence x when we feed transferred sentence $\hat{y} = f_\theta(x,\hat{s})$ to the Style Transformer network with the original style label s.

2) For the style controlling, we train a discriminator network to assist the Style Transformer network to better control the style of the generated sentence



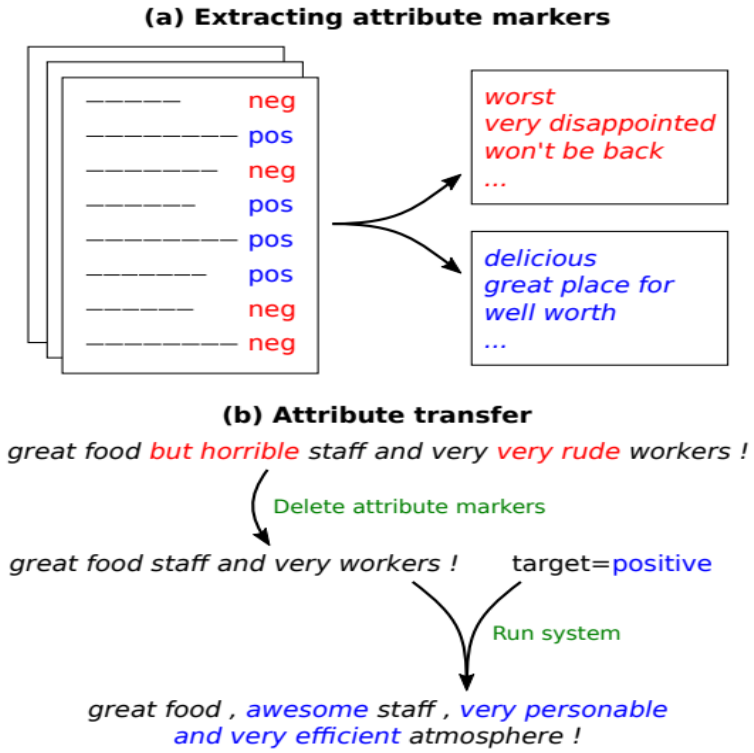
Conditional Discriminator: In a setting similar to Conditional GANs (Mirza and Osindero, 2014), discriminator makes decision conditions on an input style. Explicitly, a sentence x and a proposal style s are fed into discriminator $d_\varphi(x, s)$, and the discriminator is asked to answer whether the input sentence has the corresponding style. In the discriminator training stage, the real sentence from datasets x, and the reconstructed sentence $y = f_\theta(x, s)$ are labelled as positive, and the transferred sentences $\hat{y} = f_\theta(x,\hat{s})$ where $s \neq \hat{s}$, are labelled as negative. In the Style Transformer network training stage, the network f_θ is trained to maximise the probability of positive when feeding $f_\theta(x,\hat{s})$ and $\hat{s}$ to the discriminator.

Multi-class Discriminator: Different from the previous one, in this case, only one sentence is fed into discriminator $d_\varphi(x)$, and the discriminator aims to answer the style of this sentence. More concretely, the discriminator is a classifier with K + 1 classes. The first K classes represent K different styles, and the last class stands for the generated data from $f_\theta(x,\hat{s})$, which is also often referred to as a fake sample. In the discriminator training stage, we label the real sentences x and reconstructed sentences $y = f_\theta(x, s)$ to the label of the corresponding style. And for the transferred sentence $yb = f_\theta(x,\hat{s})$ where $s \neq \hat{s}$, is labelled as the class 0. In the Style Transformer network learning stage, we train the network $f_\theta(x,\hat{s})$ to maximise the probability of the class which stands for style $\hat{s}$.

DELETE RETRIEVE AND GENERATE

Attribute transfer can often be accomplished by changing a few attribute markers— words or phrases in the sentence that are indicative of a particular attribute—while leaving the rest of the sentence largely unchanged.

First, from an unaligned corpora of positive and negative sentences, attribute markers are identified by finding phrases that occur much more often within sentences of one attribute than the other (e.g., “worst” and “very disappointed” are negative markers). Second, given a sentence, we delete any negative markers in it, and regard the remaining words as its content. Third, we retrieve a sentence with similar content from the positive corpus.

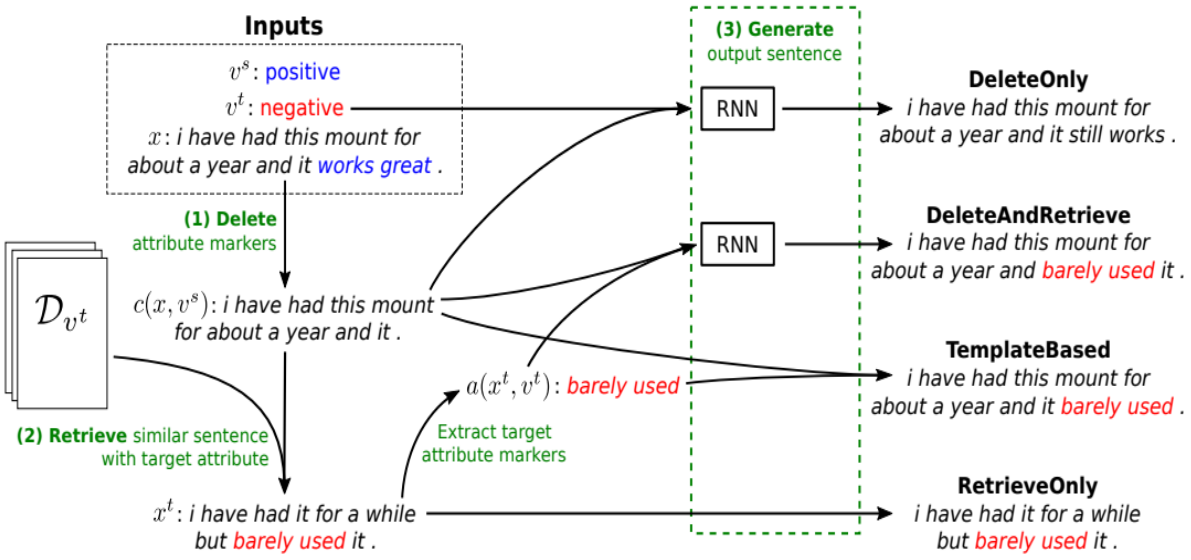


Here, the neural system extracts content words in the same way as the baseline, then generates the final output with an RNN decoder that conditions on the extracted content and the target attribute. This approach

has significant benefits at training time, compared to adversarial networks: having already separated content and attribute, we simply train our neural model to reconstruct sentences in the training data as an auto-encoder.

We assume access to a corpus of labelled sentences $D = \{(x_1, v_1), \ldots, (x_n, v_n)\}$, where x_i is a sentence and $v_i \in V$, the set of possible attributes (e.g., for sentiment, V = {"positive", "negative"}). We define $D_v = \{x : (x, v) \in D\}$, the set of sentences in the corpus with attribute v. Crucially, we do not assume access to a parallel corpus that pairs sentences with different attributes and the same content. Our goal is to learn a model that takes as input (x, v_{g}) where x is a sentence exhibiting source (original) attribute v_{src}, and v_{g} is the target attribute, and outputs a sentence y that retains the content of x while exhibiting v_{g} .

There are four systems: two baselines (RETRIEVE ONLY and TEMPLATE BASED) and two neural models (DELETE ONLY and DELETE AND RETRIEVE)



RETRIEVE ONLY: returns the retrieved sentence x_{tgt} verbatim. This is guaranteed to produce a grammatical sentence with the target attribute, but its content might not be similar to x.

TEMPLATE BASED: replaces the attribute markers deleted from the source sentence $a(x, v_{src})$ with those of the target sentence $a(x_{tgt}, v_{tgt})$. 2 This strategy relies on the assumption that if two attribute markers appear in

similar contexts , they are roughly syntactically exchangeable. For example, “love” and “don’t like” appear in similar contexts (e.g., “i love this place.” and “i don’t like this place.”), and exchanging them is syntactically valid. However, this naive swapping of attribute markers can result in ungrammatical outputs.

DELETE ONLY: first embeds the content $c(x, v_{src})$ into a vector using an RNN. It then concatenates the final hidden state with a learned embedding for v_{tgt}, and feeds this into an RNN decoder to generate y. The decoder attempts to produce words indicative of the source content and target attribute, while remaining fluent.

DELETE AND RETRIEVE: is similar to DELETE ONLY, but uses the attribute markers of the retrieved sentence x_{tgt} rather than the target attribute v_{tgt}. Like DELETEONLY, it encodes $c(x, v_{src})$ with an RNN. It then encodes the sequence of attribute markers $a(x_{tgt}, v_{tgt})$ with another RNN. The RNN decoder uses the concatenation of this vector and the content embedding to generate y.

DELETE AND RETRIEVE combines the advantages of TEMPLATE BASED and DELETE ONLY. Unlike TEMPLATE BASED, DELETE AND RETRIEVE can pick a better place to insert the given attribute markers, and can add or remove function words to ensure grammaticality. Compared to DELETE ONLY, DELETE AND RETRIEVE has a stronger inductive bias towards using target attribute markers that are likely to fit in the current context.

STYLE TRANSFER FROM NON-PARALLEL TEXT BY CROSS ALIGNMENT

This method assumes a generative process for creating data (sentences) involving latent style variables (y), latent content variables (z), and the data points (x). This is a common approach in generative modelling, where data is assumed to be generated from hidden variables.

There are two observed datasets, x_1 and x_2, with the same content distribution but different styles, y_1 and y_2. These datasets consist of samples drawn from different style distributions, $p(x_1|y_1)$ and $p(x_2|y_2)$. The main problem is to estimate the style transfer functions between these two datasets, namely $p(x_1|x_2; y_1, y_2)$ and $p(x_2|x_1; y_1, y_2)$. The challenge is that you only observe the marginal distributions of x_1 and x_2, not the joint distribution.

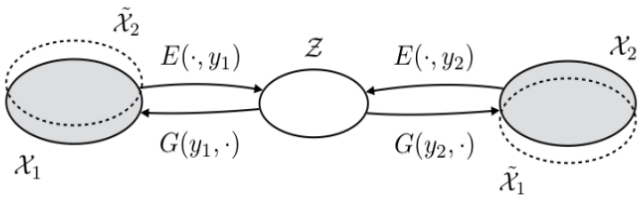


Figure 1: An overview of the proposed cross-alignment method. $\mathcal{X}_1$ and $\mathcal{X}_2$ are two sentence domains with different styles y_1 and y_2, and $\mathcal{Z}$ is the shared latent content space. Encoder E maps a sentence to its content representation, and generator G generates the sentence back when combining with the original style. When combining with a different style, transferred $\tilde{\mathcal{X}}_1$ is aligned with $\mathcal{X}_2$ and $\tilde{\mathcal{X}}_2$ is aligned with $\mathcal{X}_1$ at the distributional level.

Proposition 1:
This proposition states that the estimation problem is feasible if and only if different styles (y, y_0) lead to different distributions $p(x|y)$ and $p(x|y_0)$. In other words, the style variable should uniquely influence the data, making the style transfer task well-defined.

Example 1 (Gaussian): If the content distribution (z) is too simple (e.g., a centred isotropic Gaussian), it becomes impossible to recover the effect of certain style transformations (e.g., rotation). However, if the content distribution is more complex, such as a Gaussian mixture, the style transformations can be uniquely determined.

Example 2 (Word Substitution): This example involves a language model (z) and vocabulary style (y) for words. The problem is akin to word alignment or translation. It demonstrates that the complexity of z influences the feasibility of style transfer. If z is complex, the transfer problem becomes more tractable.

Method: The text outlines a method for learning style transfer functions using an auto-encoder framework. An encoder (E) infers content (z) from a sentence (x) and style (y), and a generator (G) generates a sentence from a style and content. Variants of the auto-encoder are discussed, including aligned auto-encoder and cross-aligned auto-encoder, each with different approaches to aligning the latent spaces and content distributions of the two styles.

Adversarial Training: The models incorporate adversarial training, where discriminators (D1 and D2) are introduced to distinguish between real and generated data. This adversarial training helps align the distributions and improve the quality of style transfer.

Approximation Techniques: To handle the discrete nature of language data, the text mentions the use of approximation techniques, including the use of softmax distributions for word sampling and Professor-Forcing, which matches the sequences of hidden states during training.

To achieve cross-alignment, the model employs a cross-alignment loss function that encourages the encoders to produce similar latent representations for input text from both domains. This is particularly important as the encoders are responsible for extracting the content information from the input text, which should remain consistent across different styles.

The cross-alignment loss is typically formulated using a distance metric, such as the L2 norm or Kullback-Leibler divergence, to measure the

discrepancy between the latent representations produced by the encoders for input text from different domains. By minimising this loss, the encoders are incentivized to learn latent representations that are more aligned, thereby preserving the underlying content while allowing for stylistic variations.

The cross-alignment mechanism can be implemented in various ways, but a common approach involves adding a cross-alignment layer to the encoders. This layer takes as input the latent representations produced by both encoders and computes the cross-alignment loss. The loss is then backpropagated through the network to update the encoder parameters, guiding them towards producing more aligned latent representations.

RESULTS

Input	Cross Aligned	Delete, Retrieve and Generate	Style Former
the decor was seriously lacking	the decor was exceptional	the decor was seriously amazing!	the decor was seriously beautiful.
i love the food ... however service here is horrible.	i love the food and salads here service outstanding !	i love the food ... however service here is very good and reasonably priced	i love the food ... however service here is friendly .
chicken fried chicken was dry but the green chili macaroni and cheese was good	chicken fried chicken was delicious and the chicken salad and cheese chicken was good	chicken fried is great but the green chili is good but the macaroni and cheese is good .	chicken fried chicken was efficient but the green chili macaroni and cheese was good .
this was by far the rudest person i spoke with .	this was by far the nicest person i dealt with	this was by far the rudest person i spoke with the best service .	this was by far the cheerful person i spoke with .
i love this place , the service is always great	i know this place , the food is just a horrible !	i did not like the homework of lasagna , not like it , .	i avoid this place , the service is disgusting bad !
this place has been making great sushi and sashimi for years .	this place has been to great pizza and better for two years .	i ca n't believe how this place has been making sashimi for years .	this place has been making bad sushi and sashimi for years .
the chicken chimi i had was absolutely fantastic !	the chicken burrito i had was absolutely cold !	the chicken chimi i had was crappy .	the chicken chimi i had was absolutely crap !

MAJOR PROBLEMS FACED

>Lack Of GPU Access:

Without any proper GPUs like A100 to train on, we only had our local GPUs, which severely limited our ability to experiment and train with different architectures.

We mostly relied on Google Colab, which is extremely restrictive with the packages it can work with and with the usage limits.

Frequently, we would leave models to train and come back to find that the runtime had been disconnected and lost, or it hit a usage limit and was automatically deleted.

>Dependencies:

While referring to existing implementations, we observed that most of the repositories weren't not well maintained and relied on obscure, hard to find, deprecated packages.

This in conjunction with the above mentioned point, meant countless hours of dependency resolving, trying to get all the packages onto the runtime and run smoothly.

In some cases, it was downright impossible as Colab does not permit changing python versions.

>Lack of parallel datasets:

Text Style Transfer is still a newly emerging field and majority of the datasets are non parallel, scraped from popular websites.

This makes it hard to train models on this data, and also necessitates a lot of preprocessing along with overcomplicating the architecture, thereby increasing training times.

Future Prospects

Working on this project has been an amazing experience and has kindled our passion and interest to explore the field of AI/ML. We definitely plan on expanding our knowledge, even after the completion of this project as this is truly one of the most demanding and extraordinary fields today that has great future scope.

We think that this project will help in the following ways:

1. Understand how unexplored the field of text style transfer is because there exist few models or even research papers that use a parallel dataset and transformer architecture.
2. Provide information about various implementations for future use.

In terms of future scope, we believe our project could be carried further by:

1. Leveraging the power of LLMs and their vast training data, to generate transferred sentences that are on par with human speech.
2. Designing better implementations possibly using RL.

REFERENCES

1) 3B1B Neural network playlist:
https://www.youtube.com/watch?v=kjBOesZCoqc&list=PL0-GT3co4r2y2YErbmuJw2L5tW4Ew2O5B

2) Coursera course(Deep learning specialisation):
Course 1:
https://www.coursera.org/learn/neural-networks-deep-learning?specialization=deep-learning
Course 2:
https://www.coursera.org/learn/deep-neural-network?specialization=deep-learning
Course 5:
https://www.coursera.org/learn/nlp-sequence-models?specialization=deep-learning

3) Coursera course youtube playlist:
https://youtube.com/playlist?list=PLpFsSf5Dm-pd5d3rjNtIXUHT-v7bdaEIe

4)GAN's Playlist:
https://www.youtube.com/playlist?list=PLdxQ7SoCLQAMGgQAIAcyRevM8VvygT pCu

5)Tensorflow:
https://www.tensorflow.org/learn

6) Attention Mechanism Paper by Vaswani et al
https://arxiv.org/pdf/1706.03762.pdf

7) Transformer Based Articles
https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

https://medium.com/mlearning-ai/understanding-the-transformer-model-a-breakdown-of-attention-is-all-you-need-450cffdebf22

8) Style Transformer
https://arxiv.org/pdf/1905.05621.pdf

9) Delete, Retrieve and Generate
https://aclanthology.org/N18-1169.pdf

10) Style Transfer From Non Parallel Text by Cross Alignment
https://arxiv.org/pdf/1705.09655.pdf