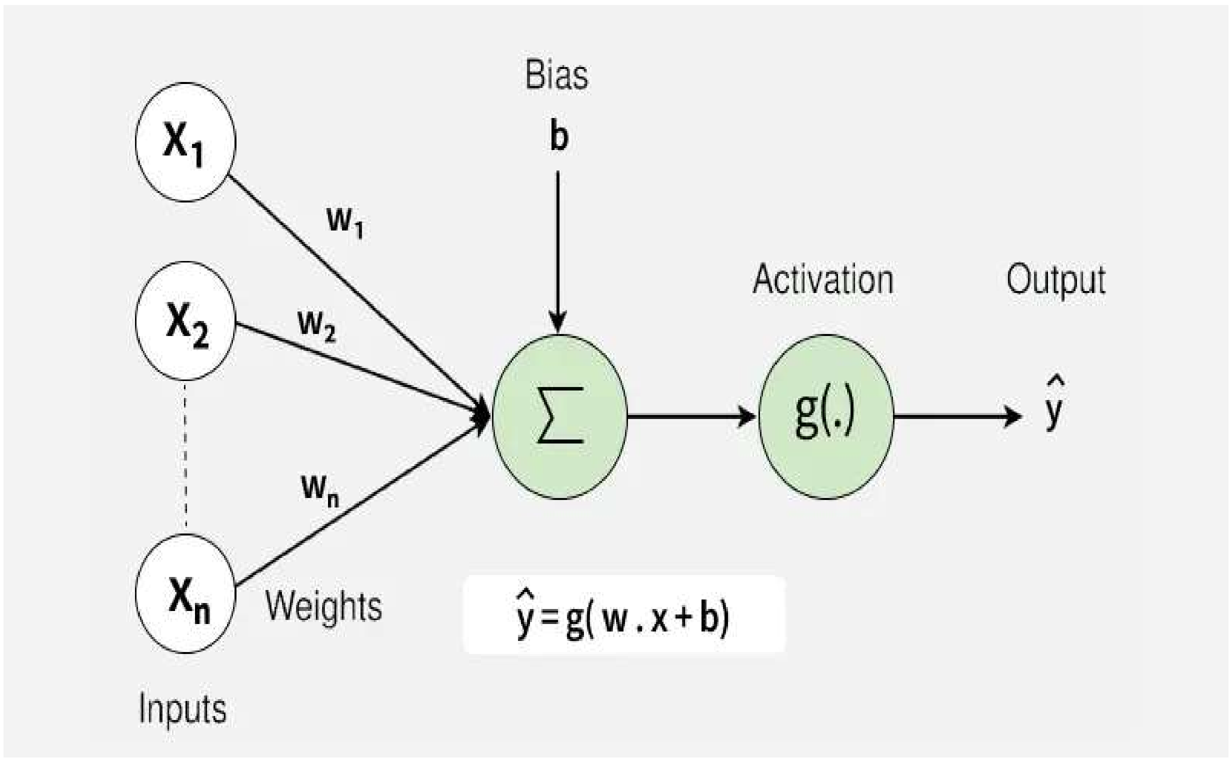
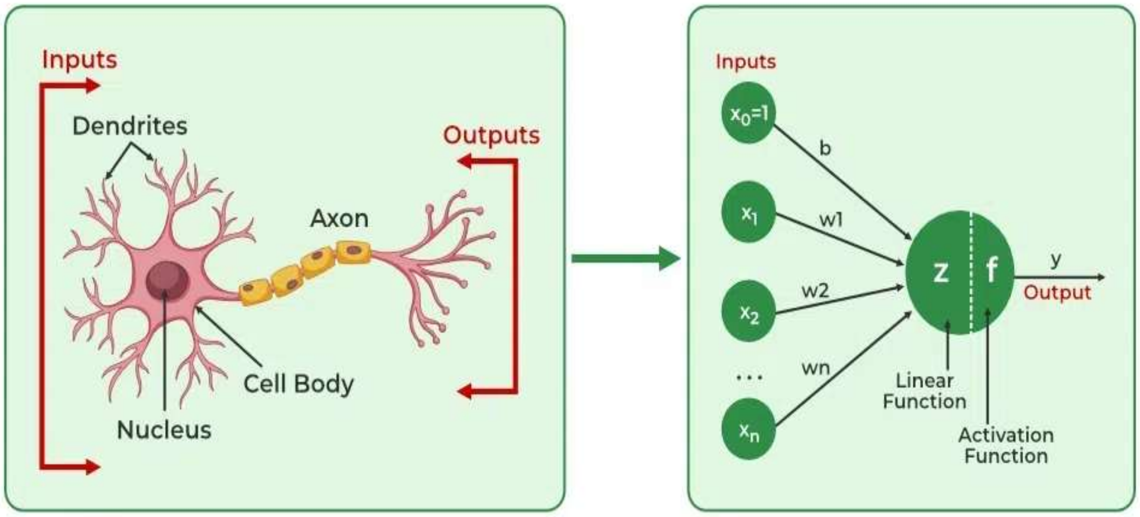
Neural Network

Neural networks are machine learning models that mimic the complex functions of the human brain. These models consist of interconnected nodes or neurons that process data, learn patterns and enable tasks such as pattern recognition and decision-making.

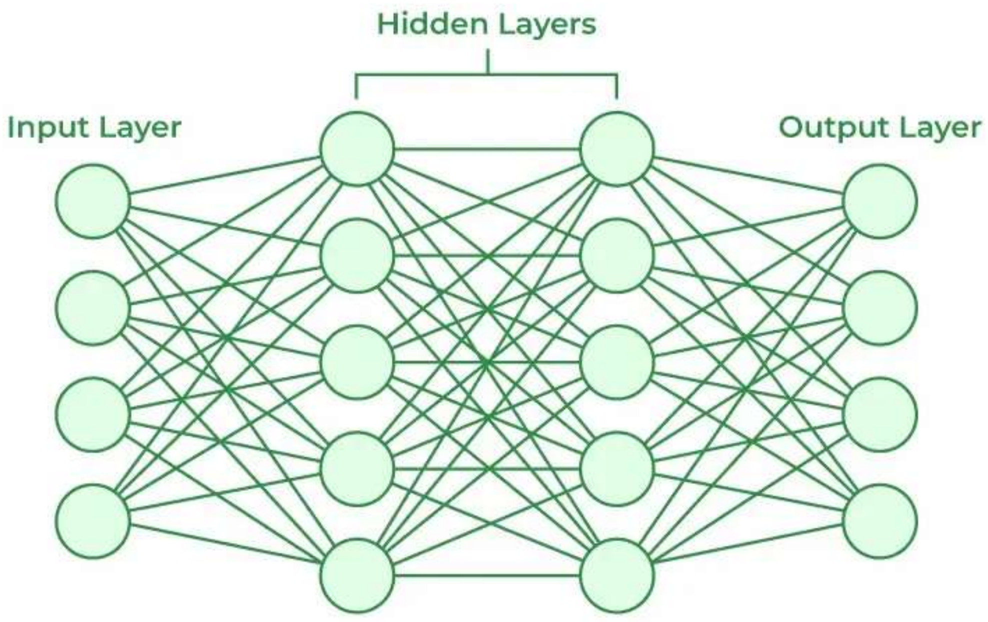


Autoregressive model, Basics of Transformer Architecture, choosing a single token from probability distribution, parallel token processing ***SL. No-9***


Inputs
Dendrites
Outputs
Axon
Cell Body
Nucleus
Inputs
$x_0=1$
b
x_1
w1
x_2
w2
...
wn
x_n
z
f
y
Output
Linear Function
Activation Function

Layers in Neural Network Architecture

1. **Input Layer:** This is where the network receives its input data. Each input neuron in the layer corresponds to a feature in the input data.
2. **Hidden Layers:** These layers perform most of the computational heavy lifting. A neural network can have one or multiple hidden layers. Each layer consists of units (neurons) that transform the inputs into something that the output layer can use.
3. **Output Layer:** The final layer produces the output of the model. The format of these outputs varies depending on the specific task like classification, regression.



Working of Neural Networks

1. Forward Propagation

When data is input into the network, it passes through the network in the forward direction, from the input layer through the hidden layers to the output layer. This process is known as forward propagation. Here’s what happens during this phase:

1. Linear Transformation: Each neuron in a layer receives inputs which are multiplied by the weights associated with the connections. These products are summed together and a bias is added to the sum. This can be represented mathematically as:

$$z = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

where

- w represents the weights
- x represents the inputs
- b is the bias

2. Activation: The result of the linear transformation (denoted as z) is then passed through an activation function. The activation function is crucial because it introduces non-linearity into the system, enabling the network to learn more complex patterns. Popular activation functions include ReLU, sigmoid and tanh.

2. Backpropagation

After forward propagation, the network evaluates its performance using a loss function which measures the difference between the actual output and the predicted output. The goal of training is to minimize this loss. This is where backpropagation comes into play:

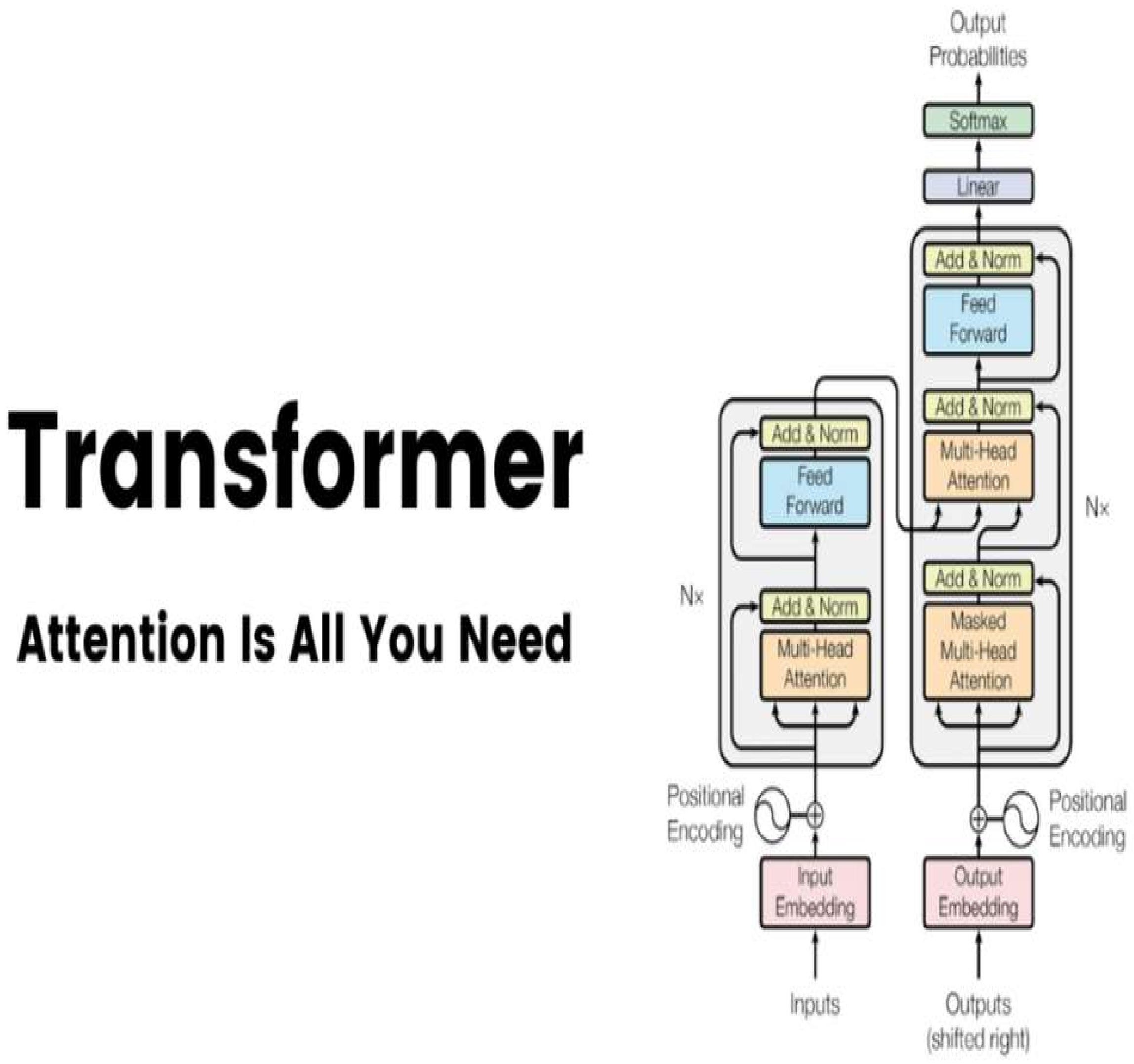
- **Loss Calculation:** The network calculates the loss which provides a measure of error in the predictions. The loss function could vary; common choices are mean squared error for regression tasks or cross-entropy loss for classification.
- **Gradient Calculation:** The network computes the gradients of the loss function with respect to each weight and bias in the network.
- **Weight Update:** Once the gradients are calculated, the weights and biases are updated using an optimization algorithm like stochastic gradient descent (SGD). The weights are adjusted in the opposite direction of the gradient to minimize the loss.

3. Iteration

This process of forward propagation, loss calculation, backpropagation and weight update is repeated for many iterations over the dataset. Over time, this iterative process reduces the loss and the network's predictions become more accurate.

Through these steps, neural networks can adapt their parameters to better approximate the relationships in the data, thereby improving their performance on tasks such as classification, regression or any other predictive modelling.

Deep Dive into the Transformer Architecture:



Transformer Architecture represents a paradigm shift in how sequential data is processed for natural language understanding and generation. At its core, the Transformer model abandons the conventional reliance on recurrent or convolutional layers, instead leveraging a novel mechanism known as self-attention to process data in parallel.

The Encoder-Decoder Structure

Central to the Transformer's design is its encoder-decoder structure, consisting of stacks of encoder and decoder layers. Each encoder layer performs two primary functions: self-attention and position-wise feed-forward neural networks. The encoder's role is to process the input sequence and map it into a continuous representation that holds both the semantic and syntactic information of the input.

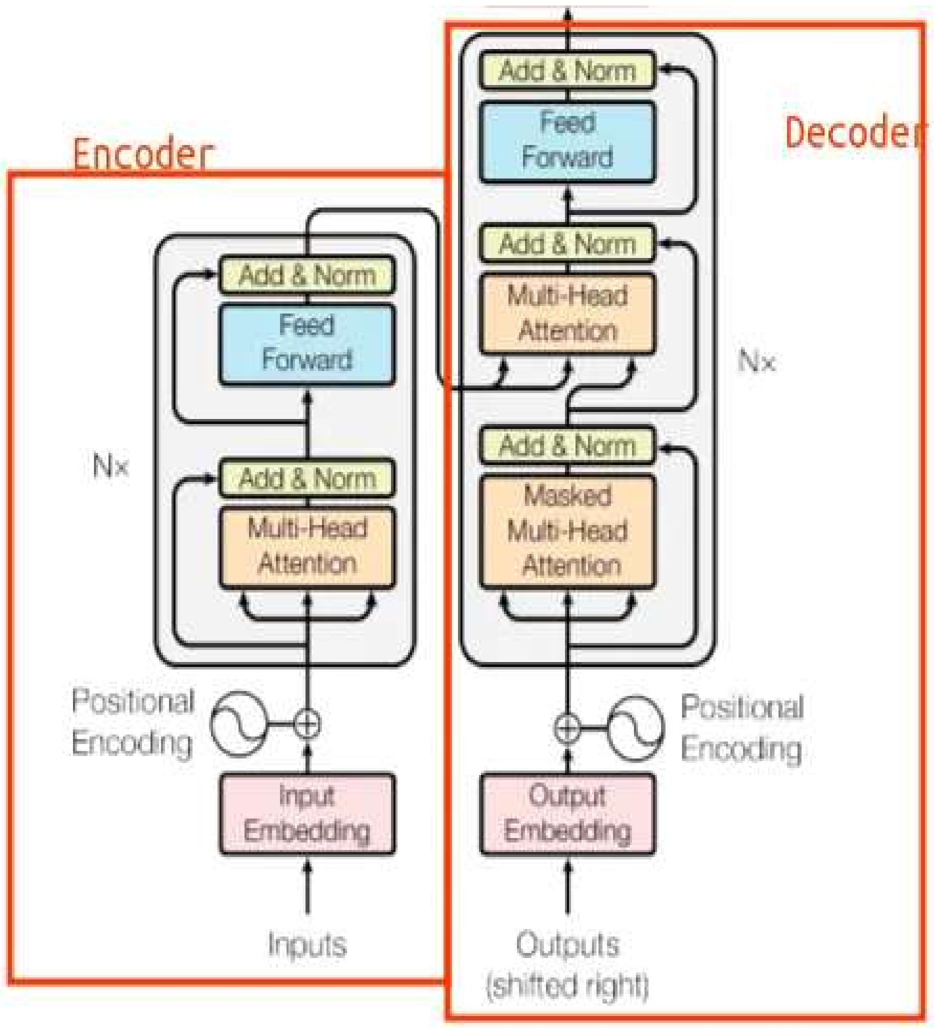
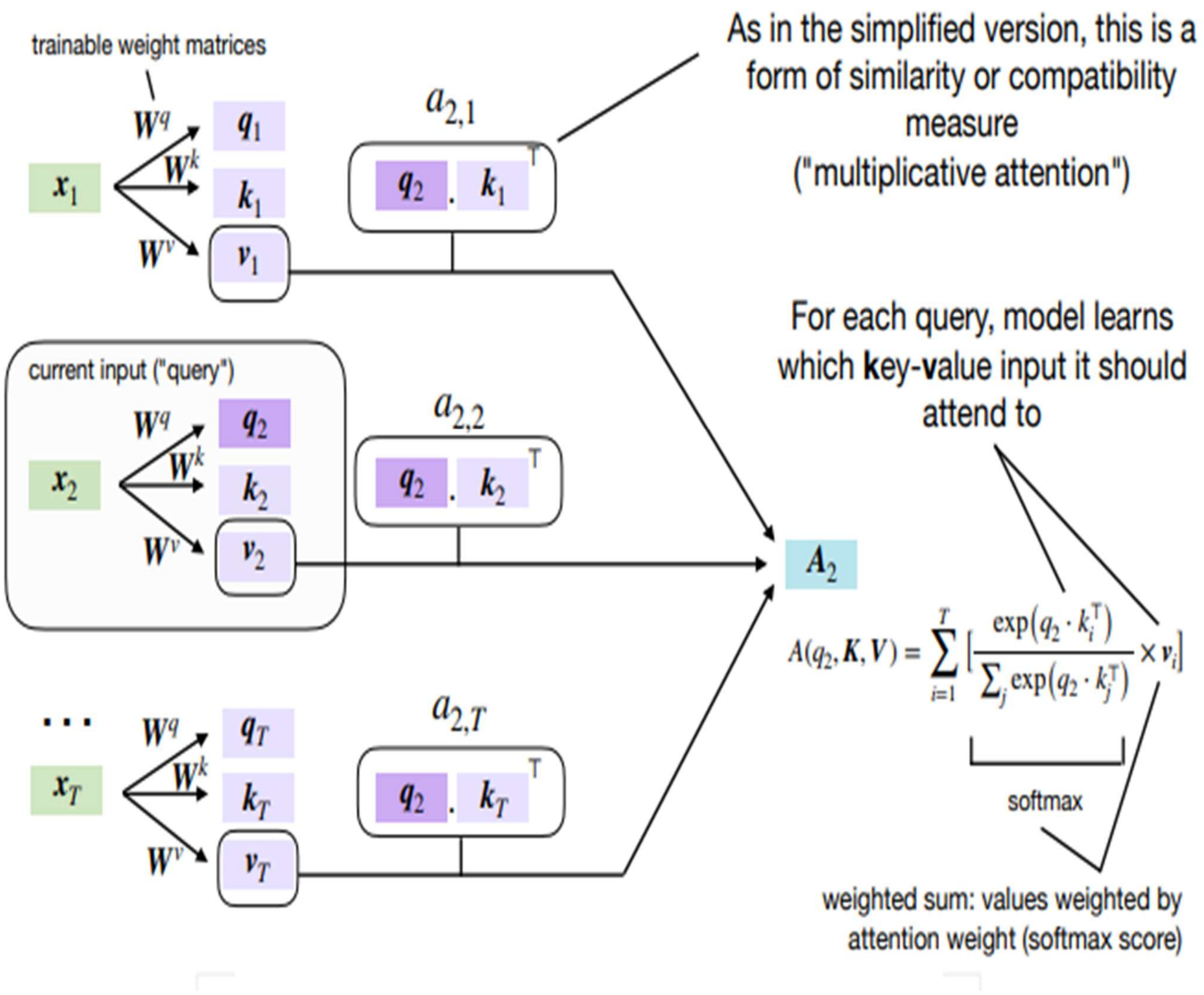


Fig. **The Encoder-Decoder Structure**

Self-Attention Mechanism

The self-attention mechanism is the heart of the Transformer, enabling the model to weigh the importance of different words within the input sequence relative to each other. Unlike traditional attention mechanisms that operate in a query-key-value paradigm, self-attention applies this concept within the input sequence itself, allowing each position to attend to all positions and thus capturing the intricate dependencies regardless of their distance in the sequence. This mechanism is crucial for understanding the context and meaning of words in sentences, enhancing the model's language processing capabilities.

Self-Attention Mechanism



trainable weight matrices
W^q
W^k
W^v
x_1
q_1
k_1
v_1
$a_{2,1}$
$q_2 . k_1^\top$
As in the simplified version, this is a form of similarity or compatibility measure ("multiplicative attention")
current input ("query")
x_2
q_2
k_2
v_2
$a_{2,2}$
$q_2 . k_2^\top$
For each query, model learns which **k**ey-**v**alue input it should attend to
A_2
...
x_T
q_T
k_T
v_T
$a_{2,T}$
$q_2 . k_T^\top$
softmax
weighted sum: values weighted by attention weight (softmax score)

$$A(q_2, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{T} \Big[\underbrace{\frac{\exp(q_2 \cdot k_i^\top)}{\sum_j \exp(q_2 \cdot k_j^\top)}}_{\text{softmax}} \times \boldsymbol{v}_i \Big]$$

Real-world Applications and Impact of the Transformer Architecture

The Transformer architecture has been foundational in driving advancements across a wide spectrum of NLP tasks. Its ability to process sequential data in parallel and capture long-range dependencies has led to significant improvements in both understanding and generating natural language.

Machine Translation

The initial application of the Transformer model demonstrated its superiority in machine translation tasks. By efficiently handling sequences and understanding the context of entire sentences, the Transformer has achieved state-of-the-art performance in translating between languages, reducing training times and improving accuracy.

Text Summarization

Transformers have revolutionized text summarization by enabling models to generate concise and relevant summaries of lengthy texts. This is achieved through their ability to understand the overall context and identify the most significant parts of the source material, a task that previous models struggled with.

Question Answering

In question-answering systems, the Transformer's ability to understand and process natural language queries has led to more accurate and context-aware responses. By analysing the relationships and significance of words within both the query and the source documents, these models can provide precise answers to a wide range of questions.

Sentiment Analysis

The application of Transformer models in sentiment analysis has allowed for more nuanced and context-sensitive interpretations of text. Their deep understanding of language enables them to discern subtle nuances in sentiment, leading to more accurate classification of texts according to the sentiments expressed.

Language Model Pre-training

Perhaps the most transformative application of the Transformer has been in the development of large pre-trained language models like BERT and GPT. These models, built upon the Transformer architecture, have set new benchmarks in a multitude of NLP tasks by leveraging vast amounts of text data to learn rich representations of language.

Example: ***Problem***

Consider the neural network in Figure 2.11. The network takes two input variables, x_1 and x_2, outputs two continuous variables, y_1 and y_2, and utilises the Sigmoid activation function at each hidden unit. At the current training checkpoint, the weights have the following values: $w_{11}^{(1)} = 0.1$, $w_{21}^{(1)} = 0.2$, $w_{12}^{(1)} = 0.3$, $w_{22}^{(1)} = 0.4$, $w_{11}^{(2)} = 0.5$, $w_{21}^{(2)} = 0.7$, $w_{12}^{(2)} = 0.6$, $w_{22}^{(2)} = 0.8$. The bias terms are $b_1 = 0.25$ and $b_2 = 0.35$.

Given a new training input vector $\mathbf{x} = (x_1, x_2) = (0.1, 0.5)$ and the expected output $\mathbf{t} = (t_1, t_2) = (0.05, 0.95)$, let us calculate the update for using $w_{11}^{(2)}$ stochastic gradient descent and $\eta = 0.1$.

We will first forward propagate through the neural network to store values of hidden units and predicted outputs.

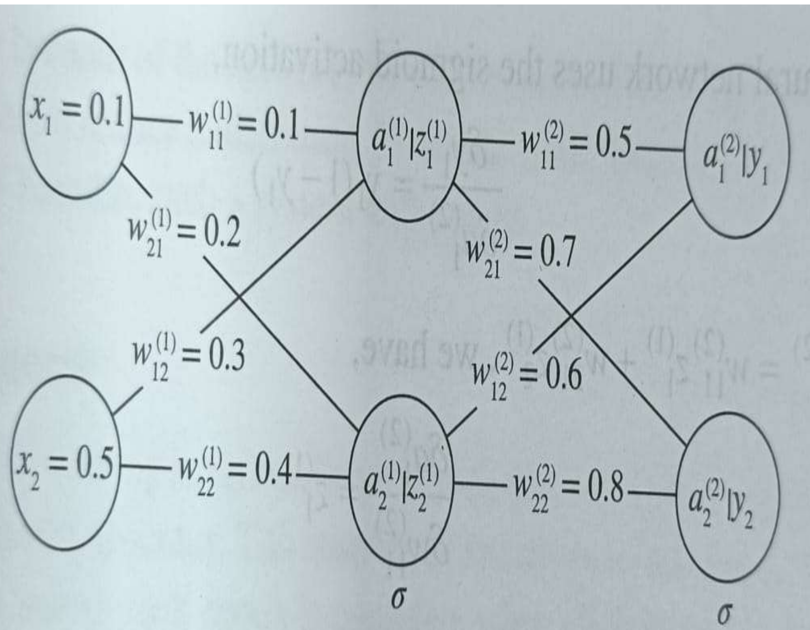


FIGURE 2.11 The neural network architecture for Example 2.5.

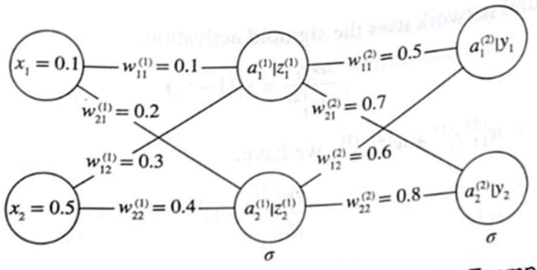


FIGURE 2.11 The neural network architecture for Example 2.5.

Forward Propagation:

$$a_1^{(1)} = 0.1 \cdot 0.1 + 0.5 \cdot 0.3 + 0.25 = 0.410$$

$$z_1^{(1)} = \sigma(a_1^{(1)}) = \frac{1}{1+e^{-0.41}} = 0.601$$

$$a_2^{(1)} = 0.1 \cdot 0.2 + 0.4 \cdot 0.5 + 0.25 = 0.470$$

$$z_2^{(1)} = \sigma(a_2^{(1)}) = \frac{1}{1+e^{-0.47}} = 0.615$$

$$a_1^{(2)} = 0.5 \times 0.601 + 0.6 \times 0.615 + 0.35 = 1.020$$

$$a_2^{(2)} = 0.7 \times 0.601 + 0.8 \times 0.615 + 0.35 = 1.263$$

$$y_1 = \sigma(a_1^{(2)}) = 0.735$$

$$y_2 = \sigma(a_2^{(2)}) = 0.780$$

We will now calculate the error contribution due to this new training input vector.

Computing Total Error:

$$E = \frac{1}{2}(t_1 - y_1)^2 + \frac{1}{2}(t_2 - y_2)^2$$

$$E = \frac{1}{2}(0.735 - 0.05)^2 + \frac{1}{2}(0.780 - 0.950)^2 = 0.234 + 0.014 = 0.248$$

Backward Propagation: We will use the chain rule to calculate the partial derivative of total error E with respect to $w_{11}^{(2)}$.

$$\frac{\delta E}{\delta w_{11}^{(2)}} = \frac{\delta E}{\delta y_1} \cdot \frac{\delta y_1}{\delta a_1^{(2)}} \cdot \frac{\delta a_1^{(2)}}{\delta w_{11}^{(2)}}$$

Differentiating E with respect to y_1,

$$\frac{\delta E}{\delta y_1} = -1 \cdot (t_1 - y_1)$$

As the neural network uses the sigmoid activation,

$$\frac{\delta y_1}{\delta a_1^{(2)}} = y_1(1-y_1)$$

Since $a_1^{(2)} = w_{11}^{(2)} z_1^{(1)} + w_{21}^{(2)} z_2^{(1)}$, we have,

$$\frac{\delta a_1^{(2)}}{\delta w_{11}^{(2)}} = z_1^{(1)}$$

Therefore,

$$\frac{\delta E}{\delta w_{11}^{(2)}} = (y_1 - t_1) \cdot y_1(1-y_1) \cdot z_1^{(1)} = 0.685 \times 0.735 \times 0.265 \times 0.601 = 0.08018$$

The updated weight $w_{11}^{(2)}$ can be obtained using the following equation (assuming $\eta = 0.1$),

$$(w_{11}^{(2)})_{\text{new}} = w_{11}^{(2)} - 0.1 \times 0.08018 = 0.5 - 0.1 \times 0.08018 = 0.49198$$

Parallel token processing

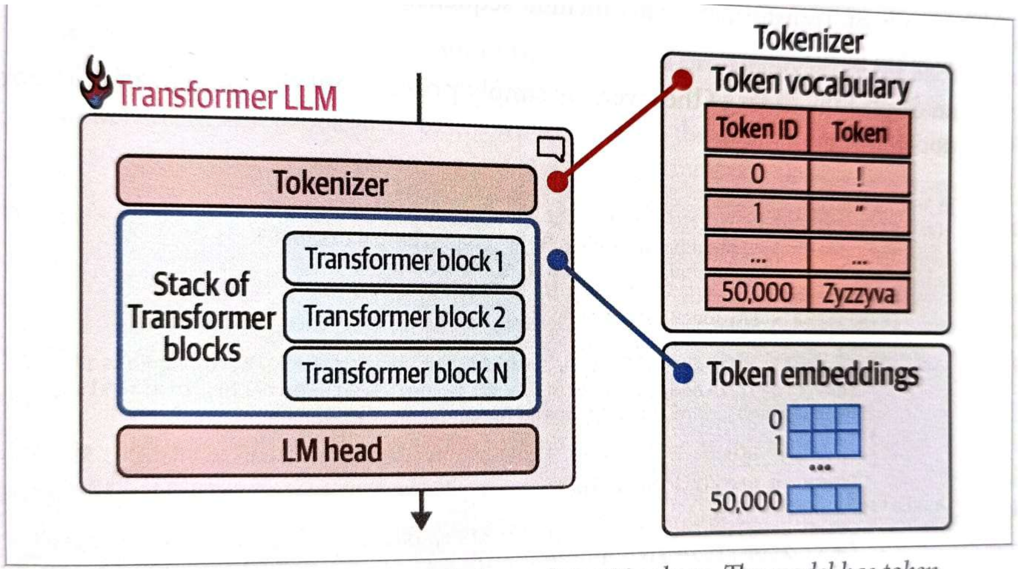


Figure 3-5. The tokenizer has a vocabulary of 50,000 tokens. The model has token embeddings associated with those embeddings.

Parallel Token Processing and Context Size

One of the most compelling features of Transformers is that they lend themselves better to parallel computing than previous neural network architectures in language processing. In text generation, we get a first glance at this when looking at how each token is processed. We know from the previous chapter that the tokenizer will break down the text into tokens. Each of these input tokens then flows through its own computation path (that's a good first intuition, at least). We can see these individual processing tracks or streams in Figure 3-8.

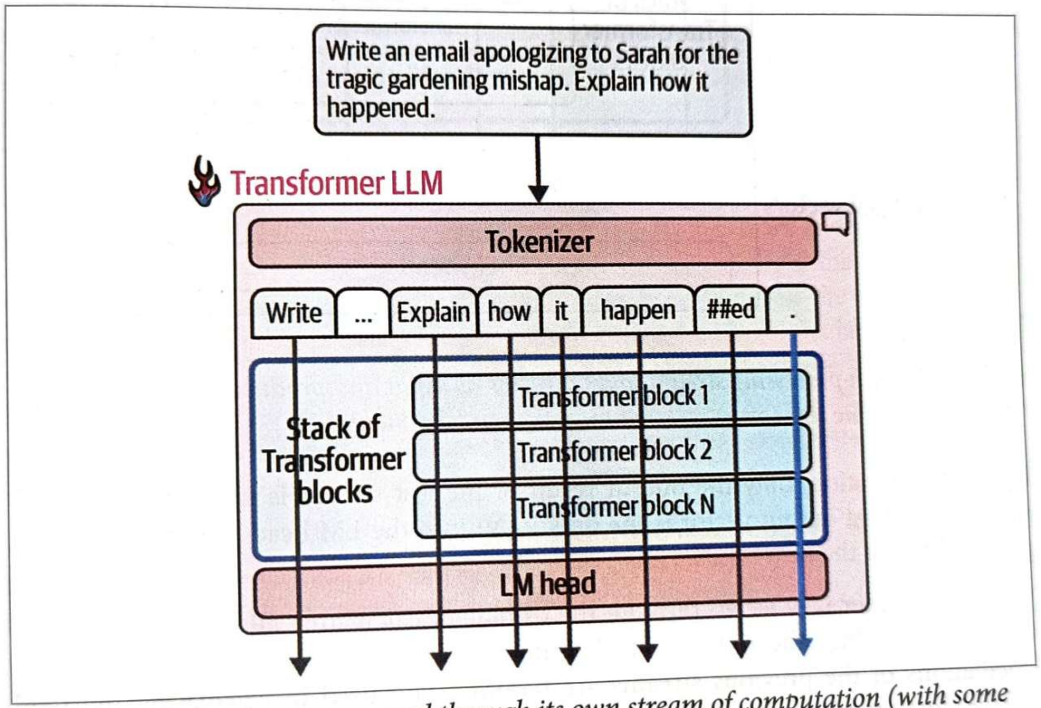

Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.
Transformer LLM
Tokenizer
Write
...
Explain
how
it
happen
##ed
.
Stack of Transformer blocks
Transformer block 1
Transformer block 2
Transformer block N
LM head

Figure 3-8. Each token is processed through its own stream of computation (with some interaction between them in attention steps, as we'll later see).