Mobiz notes for my learning

Understanding the Langchain documentation
Language models course
Deep learning course

- Base LLM: Predicts next word
- Instruction Tuned LLM: Tries to follow instructions
- RLHF: learning with human feedback
  Specify output structures
  Summary, name, output, JSON

- Giving out instructions for chatGPT, extremely clear text inputs
  Eg: Words, sentence or character limit

- Iterative Prompt development
  Idea, analysis, experiment, error analysis

- Usage Examples: Understanding Sentiment, Understanding emotion from text, Extracting a list of topics

- LLMs + custom data

Connecting large language models

- Varying the GPT temperature

- Knowledge necessary for learning: Probability statistics, linear algebra

- Real life use: Covid or no covid Classify a certain X-ray as covid, other as no covid

---

Deep learning is a powerful technique used in artificial intelligence (AI) that enables computers to learn and make decisions, somewhat similar to how our brain works. Here's a simplified explanation of how it works:

- Neurons and Neural Networks:

Deep learning is inspired by the human brain, where neurons process information. Artificial Neural Networks (ANN) are used in deep learning to mimic this process.

- Layers of Neurons:

A deep learning model consists of layers of interconnected neurons, arranged like a tower (input layer at the bottom, output layer on top).
These layers are also called hidden layers because their functioning is not directly visible to us.

- Input and Output:

The input layer receives data (e.g., images, text, audio) that the model will analyze.
The output layer produces the results or predictions (e.g., image recognition, language translation).

- Weights and Biases:

Each connection between neurons has a weight that determines its importance in the decision-making process.
Neurons also have biases, which help them make decisions more accurately.

- Learning from Data:

Deep learning requires a massive amount of data for training the model.
During training, the model adjusts its weights and biases by comparing its predictions with the actual outcomes.

- Loss Function:

A loss function measures the difference between the predicted results and the actual results.
The goal is to minimize this difference to improve the model's accuracy.

- Backpropagation:

It calculates how much each weight and bias contributed to the error and adjusts them accordingly.

- Activation Function(s):

An activation function introduces non-linearity to the model.
It decides whether a neuron should be activated (fire) or not based on the incoming data.

- Deep Learning vs. Shallow Learning:

Shallow learning models have only a few hidden layers, limiting their ability to learn complex patterns.
Deep learning models, with multiple hidden layers, can learn intricate patterns, making them more powerful.

- Convolutional Neural Networks (CNNs) for Images:

CNNs are specialized deep learning models for image-related tasks.
They use filters to detect features (e.g., edges, textures) and learn hierarchical representations.

- Recurrent Neural Networks (RNNs) for Sequences:

RNNs are used for sequence-related tasks, such as language processing and time series prediction.

They have loops to store and utilize information from previous inputs.

Transfer Learning:
Transfer learning allows using pre-trained models for new tasks, saving time and resources.
Models trained on vast datasets (e.g., ImageNet) have learned general features useful for various tasks.

GPUs for Speed:
Training deep learning models is computationally intensive.
Graphics Processing Units (GPUs) accelerate calculations, making training faster.

Applications of Deep Learning:
Deep learning is used in various fields, like computer vision, natural language processing, speech recognition, and autonomous vehicles.
It powers virtual assistants (e.g., Siri, Alexa) and recommendation systems (e.g., Netflix, Spotify).
With its ability to handle large datasets and learn intricate patterns, it has revolutionized numerous industries and continues to shape the future of technology.

Models:
LLMs
Large Language Models (LLMs) are the first type of models we cover. These models take a text string as input, and return a text string as output.
Chat Models
Chat Models are the second type of models we cover. These models are usually backed by a language model, but their APIs are more structured. Specifically, these models take a list of Chat Messages as input, and return a Chat Message.
Text Embedding Models
The third type of models we cover are text embedding models. These models take text as input and return a list of floats.

Prompt: What's passed to the underlying model
Chains: LLMChain, which combines a PromptTemplate, a Model, and Guardrails to take user input, format it accordingly, pass it to the model and get a response, and then validate and fix (if necessary) the model output.
Memory: Memory is the concept of storing and retrieving data in the process of a conversation. There are two main methods:
1. Based on input, fetch any relevant pieces of data
2. Based on the input and output, update state accordingly

Indexes: Indexes refer to ways to structure documents so that LLMs can best interact with them
Agents & Tools:
Tools
How language models interact with other resources.

Agents
The language model that drives decision making.

Langchain:
It connects to the AI models you want to use, links them to outside sources

Agent:
LangChain provides agents that have access to a suite of tools. Depending on the user's input, an agent can decide which tools to call.
All of this means that LangChain makes it possible for developers to build remarkably powerful applications by combining LLMs with other sources of computation and knowledge.

Embeddings: kind of like a wrapper for other modules

VectorStores:
- most common ways to store and search over unstructured data
- Retrieve embedding vectors that are most similar

Open AI API: gives access to the open ai features in code

- Query: has to answer the query along with the resources and references
  (No need to upload files from the web interface for the web api bot.)

- Evaluation: Evaluating generative models is challenging with traditional metrics.
  LangChain helps by providing prompts and chains to assess models using LLMs themselves.

- Modular and Flexible Approach:
  LangChain's modular approach allows developers to choose the appropriate model for their use case and leverage the provided components to build applications
  It empowers developers with components like prompt templates, output parsers, indexes, retrievers, chat message history, and agents.

- Creating Custom Solutions:
  By understanding core concepts and components, developers can tailor custom solutions to meet specific needs.
  LangChain's adaptability and ease of use make it an invaluable tool for unlocking the full potential of language models.

- Accessing the API:

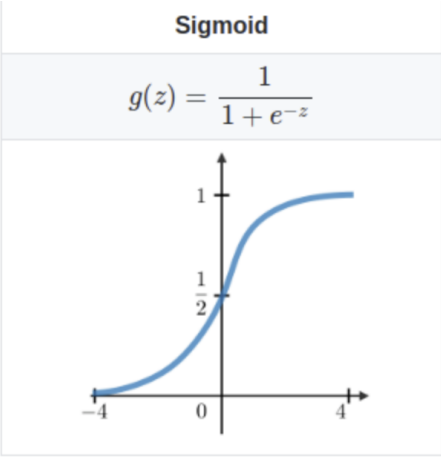
Developers can access the OpenAI API by making HTTP requests to their servers.
These requests contain data, such as text, on which the API performs language-related tasks.

Understanding the artificial neural network(ATT), An information processing unit inspired by biological circuits. 3 main boxes of category Input -> hidden units -> Output units. The total number of layers, level of nodes between the inputs and outputs, the number of neurons per architecture define the arch.

- Singlelayer (SLP) and Multilayer perceptron (MLP).

Neurons memories consist of a vector of weights. Calculation is done by multiplying the sum of the input vectors each value by the corresponding elements of the weight vector. Displayed output is the input of the activation function.
SLP is a feedforward network based on a threshold transfer function. Can only classify linearly separable cases with a binary target.

Activation functions make the decision making of the neural network. Calculates the net output of the neural node. Also called a binary step function.
1 when passing the threshold limit, 0 when not.
A neuron's activation function dictates whether it should be turned off or on. Eg, Sigmoid (1 / (e^-z))



Algorithm initial weights on the SLP are assigned randomly as there is no prior knowledge. SLP sums all the weighted output, if the answer is above the threshold 1, else 0.

Andrew NG how images are manipulative:
Why do we normalize images?
Normalize: Rearranging the pixel values so they fall within a specific range or distribution.
prevent issues like vanishing gradients and better neumaricalstability,
Linear Regression:
Imagine you're trying to predict house prices based on their size. You'll learn how to draw the best-fit line that helps you make accurate predictions.
Think of cost function as a way to measure how good or bad your predictions are, and gradient descent as a method to make your predictions even better over time.
Multiple Variables:

Imagine predicting house prices not just from size, but also from the number of bedrooms and location. You'll learn how to handle multiple factors in your predictions.
Normalizing features is like putting everything on the same scale, making it easier for the learning process.
Logistic Regression and Regularization:

Consider sorting emails into spam and not-spam categories. Logistic regression helps you with this classification task.
Regularization is like keeping a watchful eye on your model to prevent it from overcomplicating things and making silly mistakes.
Neural Networks and Deep Learning:

Think of artificial neural networks as computer-brain mimics that help you model complex relationships in data. It's like learning how to imitate human thought processes.
Visualize these networks making predictions, adjusting weights to get better results, and all of this happening automatically.
Improving Neural Networks:

Imagine a "toolbox" to make your neural network smarter—better initialization, using "brakes" like regularization, and optimizing the way it learns.
Think of it like training a dog: teach it well, prevent it from getting too excited, and reward it when it does a good job.
Building Better Projects:

Consider building a machine learning project like baking a cake—careful planning, finding the right ingredients (data), and checking the final product (evaluating errors).
Support Vector Machines (SVM):

Imagine having a dataset where you need to draw a line to separate two groups. SVM helps you find the best line, even if it's a curved or wiggly one.
Clustering and Dimensionality Reduction:

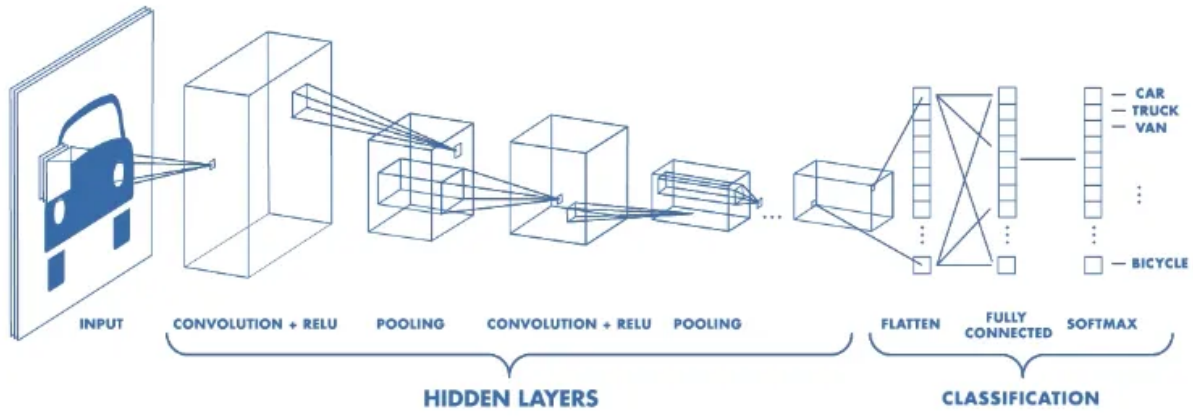
Imagine sorting different species of animals. Clustering groups them based on similarities, and dimensionality reduction helps you simplify data without losing too much information.
Anomaly Detection and Recommenders:

Picture spotting rare events, like a manufacturing defect. Anomaly detection is like finding the needle in a haystack.
Think of recommendation systems as personal shoppers for data—they know your preferences and suggest items accordingly.
Large Scale Learning:

Consider managing a huge amount of data—it's like handling a mountain. Techniques like stochastic gradient descent help you climb this mountain step by step.
Application: Photo OCR:

Imagine teaching a computer to "read" text in images, like converting images of text into editable documents.

Convolution Neural Networks:
Explanation picked from above: Specialized deep learning models for image-related tasks. They use filters to detect features (e.g., edges, textures) and learn hierarchical representations.
Specializes in a grid like topology (e.g. images)
The layers are arranged in such a way so that they detect simpler patterns first (lines, curves, etc.) and more complex patterns (faces, objects, etc.) further along.



A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

Convolutional Layer:

- Carries the main portion of the network's computational load.
- This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field.

- The kernel is spatially smaller than an image but is more in-depth(?). This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels. (depth is expressed by number of feature maps produced by the layer. Aka number of channels in the output image)
- A set of filters (small matrixes), that slide over the input image. The elemet wise multiplication produces a single output value. Filter is moved and the proces is repeated again.

Convolution + Activation

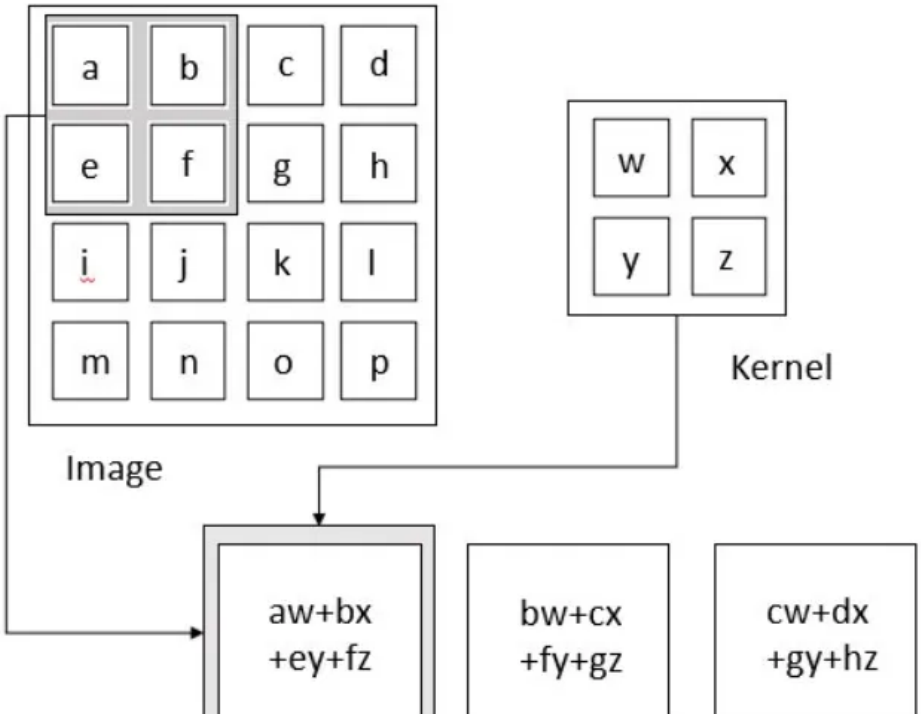
Pooling Layer

- Sjnd

If we have an input of size W x W x D and Dout number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Formula for Convolution Layer
This will yield an output volume of size Wout x Wout x Dout.



- Pooling Layer

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

YOLO:

YOLO (You Only Look Once) is an object detection algorithm that aims to detect and localize objects within images or video frames. The algorithm divides the input image into a grid and predicts bounding boxes and class probabilities for objects within each grid cell.
Can be trained on single GPUs and also nvidia divers with CUDA support

How does YOLO work, The YOLO v8 Algorithm:
https://blog.roboflow.com/whats-new-in-yolov8/
Also uses convolution and max pooling methods for image analysis. Key difference between between a Image recognition and Image Localization
Recognition does not localize the location of what it recognizes. Localization can represent the location of the specified obj in rectangle or etc.

Step by step process:
Label your images using any 3rd party tool you want, in my case LabelImg to draw rectangles around the flowers and specify the size and color. Video labeling is also a viable option but I chose to dissect the video into frames and label from several angles.

* Make sure the file is saved as an YOLO file (txt) which is the same name as the image you inserted, and has the coordinates of where the specific classes are and the size of the class for the image. (There are online tools to translate into other file formats.)

Divide your data into training data and test/ validation data. The ratio can be specified yourself but the training data should have much more files compared to the validation files.

Install YOLO v8 on the device you want to train, pip install ultralynitics.
Also make sure to use CPU when on mac M1, GPU on most other machines. Install CUDA for machines with Nvidia drivers and cuda support.

Manually create a YAML file with the correct directories. Root directory must be added if you want to run in google colab using google drive files. Installation location of ultralynitics must also be specified in this case.

Train, specify epochs as necessary. Also possible to use google drive and google colab but file size after installation of ultralytics is very large so not suggested. (Also relatively slow CPU speed.)

Test, validate if your model works.

PYPI Library:
Langchain PYPI ideas and brainstorming for ideas

Numpy and pandas already seem to have libraries going or them, so focusing on alternative large model tools like Dusk could be the mve

Dask
Dask is a flexible library for parallel computing in Python.

Dask is composed of two parts:

Dynamic task scheduling optimized for computation. This is similar to Airflow, Luigi, Celery, or Make, but optimized for interactive computational workloads.

“Big Data” collections like parallel arrays, dataframes, and lists that extend common interfaces like NumPy, Pandas, or Python iterators to larger-than-memory or distributed environments. These parallel collections run on top of dynamic task schedulers.

What would be the purpose of the model? :

1. Large-Scale Data Cleaning and Preprocessing
2. Exploratory Data Analysis (EDA)
3. Feature Engineering
4. Aggregations and Grouping
5. Parallelizing Custom Functions
6. Time Series Analysis
7. Parallel Image and Signal Processing
8. Graph Processing
9. Distributed Computing
10. Interactive Data Visualization
11. Incremental Machine Learning
12. Hybrid Workflows

How can I use Langchain?

Ask a question -> provide code necessary (this would be worse than just inputting to chatGPT)
Optimization

.query

Example code:
==========
TEMPLATE = """
You are working with a pandas dataframe in Python. The name of the dataframe is `df`.
The dataframe has the following columns: {df_columns}.

You should execute code as commanded to either provide information to answer the question or to
do the transformations required.

You should not assign any variables; you should return a one-liner in Pandas.

This is your objective: {query}

Go!

```python
print(df.head())
```
```output
{df_head}
```
```python"""
==========

---

Understanding Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed for working with sequences of data. Here's a beginner-friendly overview of RNNs:
```

RNNs:

- RNNs are neural networks designed to process sequences of data, where each step depends on the previous step.
- They are used in various applications, including natural language processing, speech recognition, and time series analysis.

1. Sequence Processing:

RNNs are particularly good at handling sequences like sentences in language, where each word depends on the words that came before it.

2. Key Components of RNNs:
   - Hidden State: The RNN has a hidden state that stores information about the previous steps in the sequence.
   - Weights: RNNs have weights that are adjusted during training to capture patterns in the data.

3. Understanding the Hidden State:

- The hidden state is like a memory that holds information from previous time steps.
- It helps the network understand context and dependencies in the sequence.

4. Forward and Backward Pass:

- RNNs perform a forward pass to make predictions.
- They use backpropagation to update their weights and learn from data.

5. Vanishing Gradient Problem:

- One challenge with RNNs is the vanishing gradient problem, which can make it difficult for the network to capture long-range dependencies.

6. Applications of RNNs:

- RNNs are used in:
  - Natural language processing for tasks like language modeling and text generation.
  - Speech recognition to convert spoken words into text.
  - Time series forecasting for predicting future values based on historical data.

7. Limitations:

- RNNs struggle with very long sequences because of the vanishing gradient problem.
- They can be computationally expensive to train.

8. Alternatives to RNNs:

- In some cases, newer architectures like LSTMs and Transformers have been developed to address the limitations of traditional RNNs.

9. Resources for Learning:

- Online courses and tutorials are great resources to learn more about RNNs and their applications.

RNNs are a fundamental building block in the field of deep learning, and they are a great starting point for understanding sequential data processing.

Title: Understanding Transformers - A Beginner's Guide

Transformers are a groundbreaking neural network architecture that has revolutionized natural language processing and many other fields. Let's explore the basics of Transformers:

1. Introduction to Transformers:
   - Transformers are a type of neural network architecture introduced in a paper titled "Attention is All You Need."
   - They've become the foundation for many state-of-the-art models, including BERT, GPT-3, and more.

2. Self-Attention Mechanism:
   - The core innovation of Transformers is the self-attention mechanism.
   - Self-attention allows the model to weigh the importance of different parts of the input sequence when making predictions.

3. Key Components of Transformers:
   - Encoder-Decoder Structure: Transformers often consist of an encoder to process input data and a decoder to generate output data.
   - Multi-Head Attention: Transformers use multiple self-attention heads to capture different relationships in the data.
   - Positional Encoding: Transformers incorporate positional information into the model, enabling it to handle sequences.

4. Language Modeling:
   - Transformers excel in language modeling tasks because they can capture long-range dependencies in text.

5. Training Transformers:
   - Transformers are trained using large datasets and vast computational resources.
   - Pre-trained models are fine-tuned for specific tasks, reducing the need for extensive training on task-specific data.

6. Applications of Transformers:
   - Transformers are used in:
     - Machine translation, enabling seamless translation between languages.

- Text summarization, extracting key information from documents.
- Question-answering systems, providing detailed responses to user queries.
- Chatbots and virtual assistants, facilitating human-like interactions.

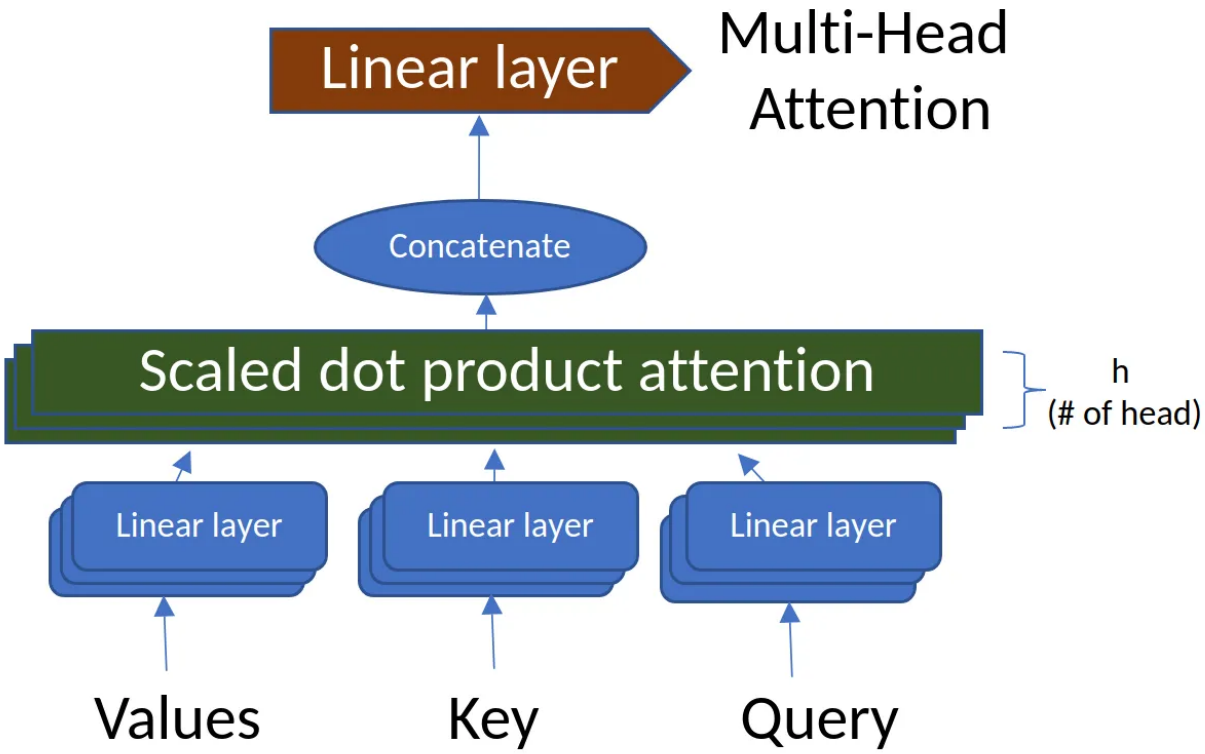
7. Limitations:
- Training and fine-tuning Transformers can be resource-intensive.
- Transformers may require large amounts of data to achieve peak performance.

8. Future Developments:
- Transformers continue to evolve, with new architectures and models emerging regularly.

9. Resources for Learning:
- Online courses, blog posts, and research papers are valuable resources for diving deeper into Transformers.

Transformers have reshaped the field of natural language processing and have found applications across various domains, making them a key technology to explore in the world of deep learning.

Basic blocks:
Multi-Head Attention



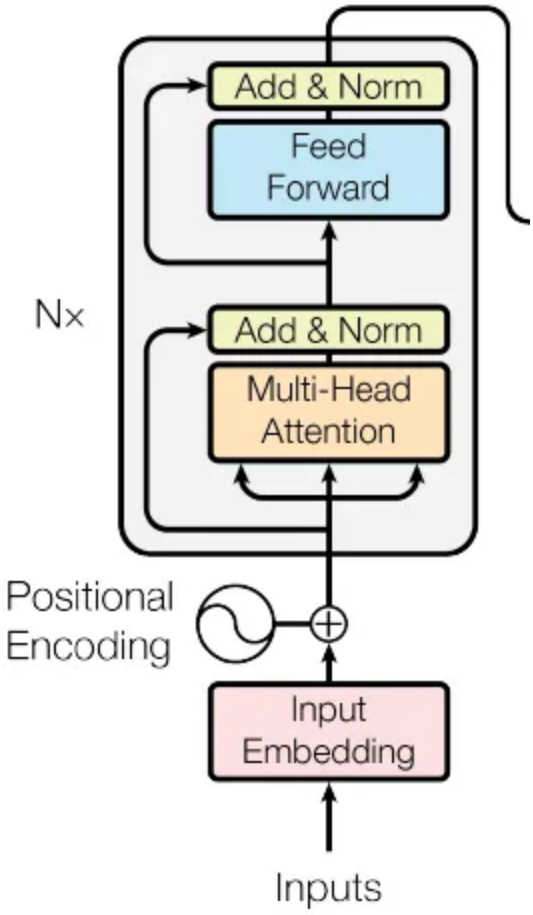
The Multi-Head Attention mechanism computes the attention between each pair of positions in a sequence. It consists of multiple “attention heads” that capture different aspects of the input sequence.

(Positional) FeedForward Networks
Initializes with two linear transformation layers and a ReLU activation function. The forward method applies these transformations and activation function sequentially to compute the output. This process enables the model to consider the position of input elements while making predictions.

Positional Encoding

Positional Encoding is used to inject the position information of each token in the input sequence. It uses sine and cosine functions of different frequencies to generate the positional encoding.

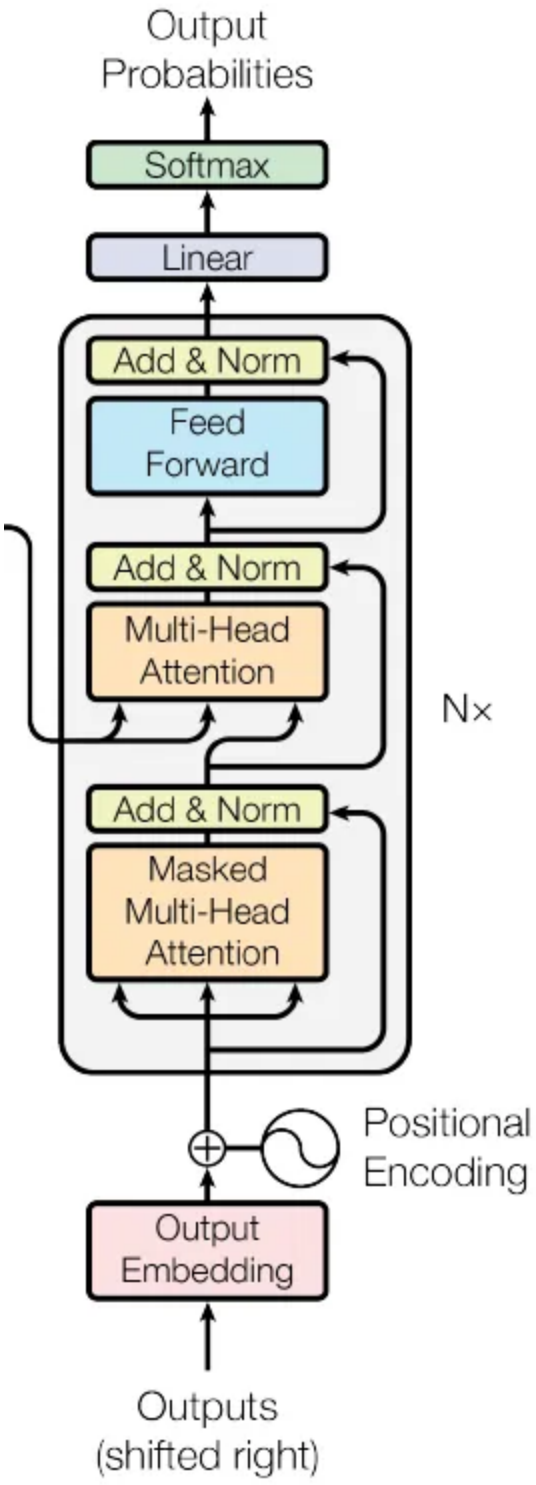
Encoder Layer:



consists of a Multi-Head Attention layer, a Position-wise Feed-Forward layer, and two Layer Normalization layers

MultiHeadAttention module, a PositionWiseFeedForward module, two layer normalization modules, and a dropout layer. The forward methods computes the encoder layer output by applying self-attention, adding the attention output to the input tensor, and normalizing the result. Then, it computes the position-wise feed-forward output, combines it with the normalized self-attention output, and normalizes the final result before returning the processed tensor.

Decoder Layer



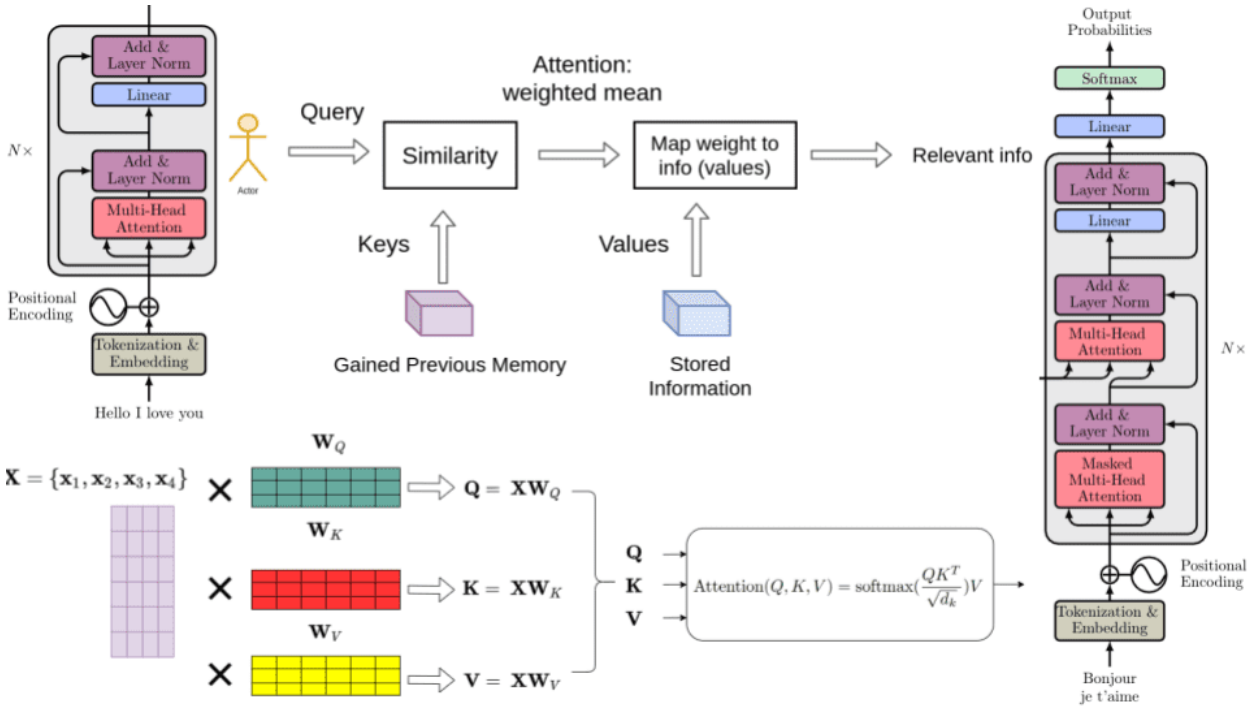
consists of two Multi-Head Attention layers, a Position-wise Feed-Forward layer, and three Layer Normalization layers

Calculate the masked self-attention output and add it to the input tensor, followed by dropout and layer normalization.
Compute the cross-attention output between the decoder and encoder outputs, and add it to the normalized masked self-attention output, followed by dropout and layer normalization.
Calculate the position-wise feed-forward output and combine it with the normalized cross-attention output, followed by dropout and layer normalization.
Return the processed tensor.

Encoder + Decoder layers = Transformer model

Generate source and target masks using the generate_mask method.
Compute source and target embeddings, and apply positional encoding and dropout.
Process the source sequence through encoder layers, updating the enc_output tensor.
Process the target sequence through decoder layers, using enc_output and masks, and updating the dec_output tensor.
Apply the linear projection layer to the decoder output, obtaining output logits.

https://towardsdatascience.com/build-your-own-transformer-from-scratch-using-pytorch-84c850470dcb



Employing usage of an appropriate dataset and loss function

Basic Steps:

Defining the basic building blocks - Multi-head Attention, Position-Wise Feed-Forward Networks, Positional Encoding

Building the Encoder block

Building the Decoder block

Transformer Implementation Overview

Transformers are a fundamental innovation in deep learning, known for their versatility and applications across various domains, from natural language processing to computer vision. In this brief note, we'll provide an overview of key concepts from the "Attention Is All You Need" paper and how to build a simple Transformer model in PyTorch.

Key Concepts

Attention Mechanisms

- Transformers are centered around attention mechanisms, which enable the modeling of dependencies in sequences without considering their distance.
- They employ a specific type of attention called multi-head attention, which is crucial to the model's operation.

Scaled Dot-Product Attention

- Scaled dot-product attention computes the importance of elements in a sequence.
- It uses queries (Q), keys (K), and values (V) matrices to calculate attention scores.
- The scores are then normalized with a softmax function to ensure they sum to one.
- Finally, the attention scores are applied to the value matrix (V) through matrix multiplication.

Multi-Head Attention

- Multi-head attention consists of several identical attention heads, each with its linear layers and scaled dot-product attention.
- These heads allow the model to attend to different parts of the input sequence independently.
- Increasing the number of attention heads enhances the model's capacity to capture information from various sequence positions.

Positional Encoding

- Positional encoding is vital because multi-head attention doesn't consider the sequence's relative positions.
- It adds positional information to the input so that the model understands the order of data points in sequences.

- Sinusoidal encoding is often used due to its periodicity, enabling the model to handle longer sequences effectively.

Building the Transformer

The Transformer model consists of an encoder and a decoder.

Encoder

- The encoder processes the input sequence and generates a memory vector.
- It comprises multiple layers, each containing self-attention and feedforward networks.
- Layer normalization and dropout are applied for regularization.

Decoder

- The decoder takes the target sequence and incorporates information from the encoder memory.
- It also includes self-attention layers, multi-head attention layers, and feedforward networks.
- Like the encoder, it utilizes layer normalization and dropout.

Transformer Class

- To create the full Transformer model, combine the encoder and decoder.
- Pass data through them in the correct order to get predictions.

Conclusion

This simplified overview provides essential insights into the Transformer architecture and how to implement it. Transformers are becoming increasingly important in deep learning, not only in natural language processing but also in computer vision. We can expect more state-of-the-art models to adopt attention-based mechanisms, leading to advancements in various tasks like object detection, image segmentation, and image generation.