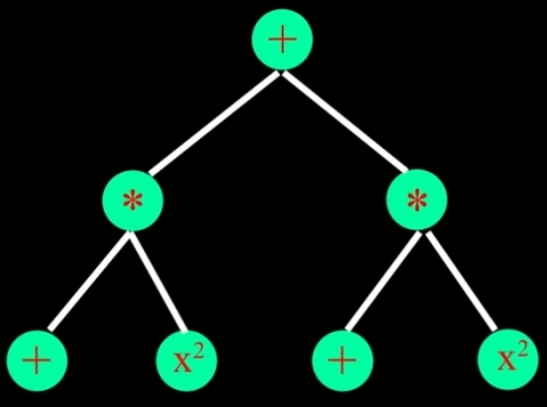
GRAPHS & AUTO-GRAD PACKAGE

Understanding Computational Graph:-

Provides a structured way to representing mathematical expression and operations in the form of a graph, where nodes represent the Operations or variables, & edges represent the dependencies between those operations.

$$((a+b)*b^2) + ((c-d)*d^2)$$



Dynamic vs Static Graph

Dynamic Graph:- Graph is build on the go.

In an interpreted language. (like dynamic graph) code is run line-by-line, offering flexibility to change the code structure on the fly. This Highlights the trade-off between execution efficiency & flexibility in model development.

Static Graph:- The graph of operations is defined and compiled before any actual values are processed.

In Complied language (like Static graph), the entire code is complied before execution, allowing for optimizations but requires whole structure to be defined upfront.

Advantages of Computational Graph

1. Simplified representation:-

Provide a simplified & a clear & a specified representation of complex mathematical operations & the relationship between them.

2. Facilitates Automatic differentiation:-

C.Graph enables automatic differentiation, which is crucial for training Neural Network. By tracing the graph from output back to inputs, framework can automatically compute gradients using the chain rule, simplifying the process of optimizing model parameters.

3. Optimization Opportunities:-

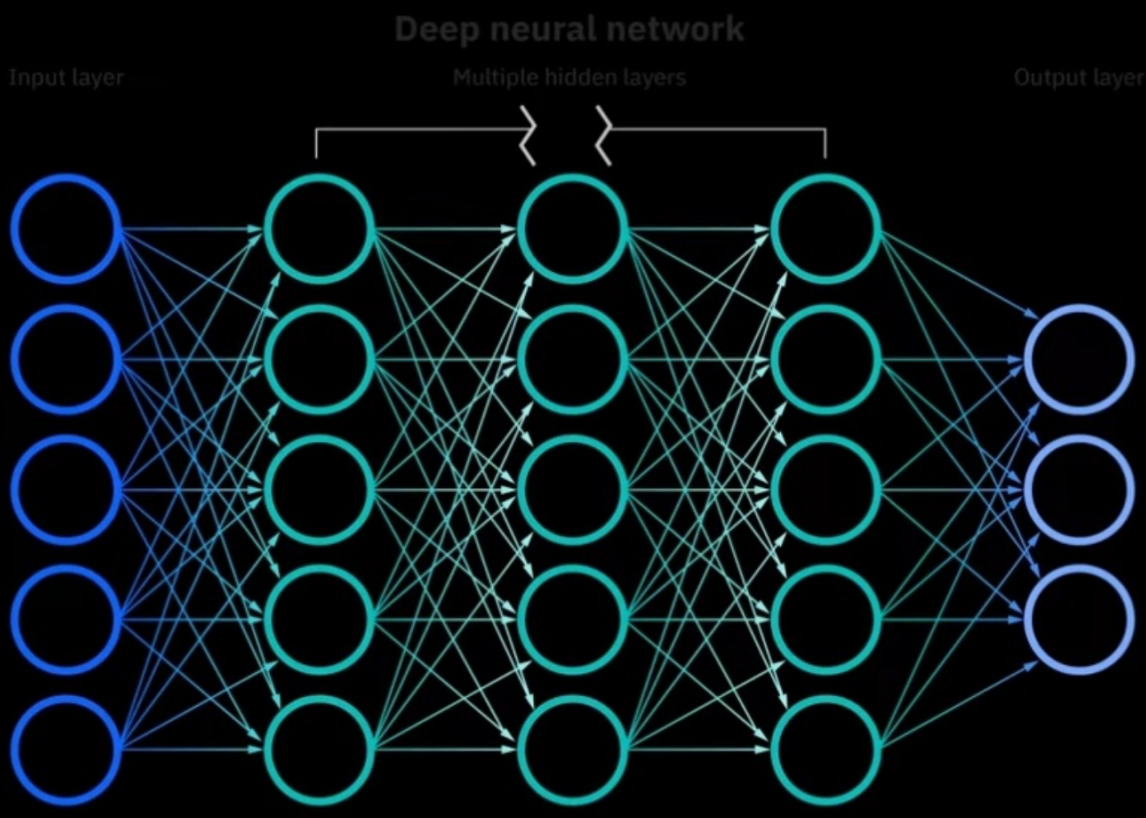
C.Graph allows for various optimization at both the operation level and the graph level. Ex. Operation that are Independent can be executed in parallel and contain sub-expressions that are computed multiple times can be

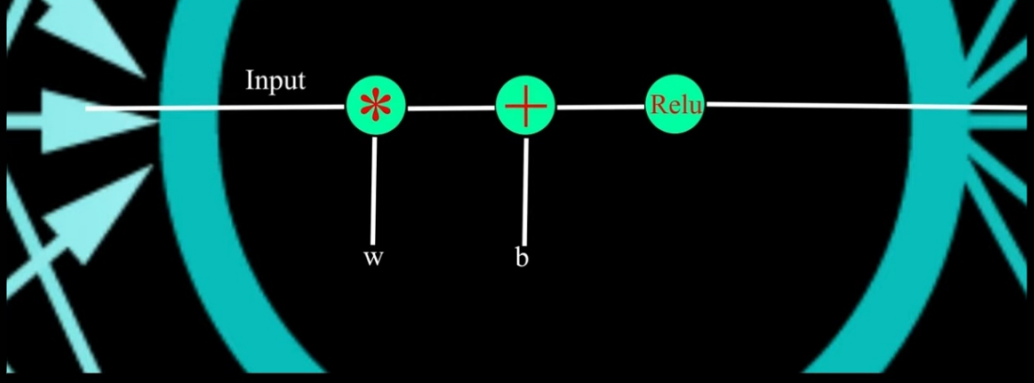
identified & optimized to avoid redundant computations.

4. Efficient Execution on Diverse Hardware:-

C.Graphs enables efficient mapping of operations to diverse Hardware accelerators like GPUs & TPUs
Frameworks can use information from the graph to better Schedule operations & manage memory,
leading to efficient execution.

..





Automatic Differentiation

C.Graph structure is Helpful in automatic differentiation because it clearly defines the Sequence of operations performed during the forward pass of a computation.

When the Backward pass is initiated for gradient computation, the graph helps in efficiently applying the chain rule to propagate Gradient Backwards through the network is this Backward traversal through the graph allows the calculation of Partial derivatives of the operation to propagate Backward.

Auto-Grad Package

The 'auto-grad' package in Py Touch is a fundamental component designed to automate the computation of Backward passes in neural network i.e. computing gradients of Tensors.

Core Functionality:-

1. Automatic Differentiation:- The primary utility of "autograd" is its ability to provide automatic differentiation for all operations on Tensons.

Each Tensor has an attribute called "grad-fn" that reference a function that has created the tensor (except the tensor created directly by the User, which has 'None' as their "grad_fn").

2. Gradient Computation:- When we perform operations on tensor ,pytorch dynamically creates a graph representing these operations in terms of the tensors and their dependencies.

When ".backward()" method on a tensor is called, Pytorch computes the gradients of that tensor w.r.t. to the Scalar value usually a loss.

How it Works:

1. Create Tensors: Define tensors with requires_grad=True if you need gradients with respect to these tensors during optimization. By default, tensors have .. requires_grad=False.

2. Define Operations: Perform operations on these tensors. PyTorch tracks these operations to construct the computation graph. The graph edges store the gradients functions of each operation.

3. Gradient Calculation: Once the forward pass is completed (your computations for output), invoke the .backward() method on the output tensor (e.g., loss in neural networks). This will trigger the backward pass, and PyTorch computes the gradients from the end of the graph (where the loss is) to the inputs.

4. Update Weights: Finally, use the gradients stored in .grad attributes of tensors (weights of the model) to update the weights.

Benefits:-

Efficiency: The use of dynamic computation graphs (define-by-run) allows for efficient and intuitive gradient calculations.

Flexibility: autograd supports complex dynamic architectures that are changeable as you run the model, which fits well with Python's programming style.