**How does unsqueeze help us to solve certain broadcasting problems?**

In broadcasting, the term "unsqueeze" refers to a method used to increase the dimensions of a tensor by inserting new singleton dimensions

**How can we use indexing to do the same operation as unsqueeze?**

Using indexing, we can achieve the same operation as unsqueeze by adding a new axis to the tensor.

**How do we show the actual contents of the memory used for a tensor?**

To show the actual contents of the memory used for a tensor, you can access the underlying data buffer of the tensor in Python libraries like NumPy or PyTorch. Both libraries provide methods to retrieve the raw data from a tensor, allowing you to see the values stored in the memory.

**Do broadcasting and expand\_as result in increased memory use? Why or why not?**

Broadcasting and expand\_as are different operations that can be used to reshape tensors to enable element-wise operations with tensors of different shapes. The memory usage implications for each operation are different:

Broadcasting:

Broadcasting is a memory-efficient operation as it does not create a copy of the data. It allows element-wise operations between tensors of different shapes by virtually expanding the smaller tensor to match the shape of the larger tensor without actually duplicating the data. Broadcasting is a view-based operation, meaning it does not create a new tensor but rather provides a "view" of the original tensor with appropriate dimensions expanded. As a result, broadcasting does not increase memory usage; it only affects how the tensor is interpreted during computation.

expand\_as:

The expand\_as method is used to increase the size of a tensor along specified dimensions by repeating elements along those dimensions. Unlike broadcasting, expand\_as creates a new tensor with the expanded size, effectively replicating the original data. This operation increases memory usage since the expanded tensor holds multiple copies of the same data. However, expand\_as is more memory-efficient than explicitly replicating the tensor using repeat, as it does not store the copied data multiple times in memory. Instead, it creates a view over the original tensor and reuses the existing memory for elements that are replicated.

**What are the three rules of Einstein summation notation? Why?**

Repeating Indices:

If an index appears twice in a term, once as a subscript (lower index) and once as a superscript (upper index), it implies summation over that index. The summation is performed over all possible values of the repeated index within the given range.

Free Indices:

If an index appears only as a subscript or only as a superscript (not repeated), it is considered a free index and is not summed over. Free indices represent components or elements that are kept distinct in the expression.

No Repeated Index More Than Twice:

In a single term or expression, an index should not appear more than twice. If an index appears more than twice, it can lead to ambiguity in the notation

What are the forward pass and backward pass of a neural network?

. Forward Pass:

During the forward pass, the input data is passed through the neural network to compute the predictions or outputs. The forward pass involves the following steps:

Input Data: The input data (e.g., images, text, numerical features) is fed into the neural network.

Weighted Sum and Activation: The input data is multiplied by the weights of each neuron and passed through an activation function (e.g., ReLU, Sigmoid, or Tanh). This process is repeated through each layer of the neural network until the final output is computed.

Prediction: The output of the last layer represents the predictions of the model for the given input data.

The forward pass is purely a feedforward process that does not involve any learning or weight updates. It only computes the predictions based on the current values of the model's parameters (weights and biases).

2. Backward Pass (Backpropagation):

The backward pass, also known as backpropagation, is the process of computing the gradients of the loss function with respect to the model's parameters. These gradients are used to update the parameters during the optimization process. The backward pass involves the following steps:

Loss Function: The loss function measures the difference between the model's predictions and the ground truth labels (or targets).

Gradient Computation: Starting from the last layer, the gradients of the loss function with respect to the model's outputs are computed. This is done using the chain rule of calculus, which propagates the gradients backward through each layer of the network.

Parameter Updates: The gradients of the loss function with respect to the model's parameters (weights and biases) are used to update the parameters using an optimization algorithm (e.g., stochastic gradient descent, Adam).

Iterative Process: The forward and backward passes are iteratively performed for multiple batches of data during training. The model's parameters are updated after each batch, gradually reducing the loss and improving the model's performance.