

Tensor Network Builder Guide

Kiana Gallagher
(Dated: September 2, 2024)

I. INTRODUCTION

The frontend for the Tensor Network Builder was built using HTML, CSS, JavaScript, and React. It is recommended that the developer has experience in these languages. The majority of the interface is coded in React, and the styling is done in CSS. The React package used is **ReactFlow**. This specific package was chosen because it is used to create graphs commonly found in computer science, and the package was well documented. Computer science graphs and tensor networks have quite a bit of overlap in their structure. The nodes and edges can easily be generalized to tensors and indices. The website has two main components: the network canvas and the drag and drop menu.

Any functions called using ReactFlow are of the form **ReactFlow.FunctionName()**. This was due to an issue importing ReactFlow into the Genie environment. It should be noted that all the ReactFlow elements should be inside the **<ReactFlow.ReactFlowProvider>** element as this is the ReactFlow environment.

II. NETWORK CANVAS

The network canvas is where the tensor network is displayed. The current canvas is provided by the ReactFlow package. The network canvas is a **div** element inside the **<ReactFlow.ReactFlowProvider>** element. The canvas is where the tensors are displayed, moved, and edited.

The **<ReactFlow.ReactFlow>** element is used to specify the functions called when specific events occur. These events include deleting tensors, deleting indices, moving tensors, and validating connections. If no function is passed through the element then the default functions are used which can be found on the ReactFlow website documentation. The **<ReactFlow.Controls>** element is for the controls at the bottom left corner of the canvas. The **<ReactFlow.MiniMap>** element is for the mini version of the canvas at the bottom right. The **<ReactFlow.Background>** element is for the background of the canvas

III. DRAG AND DROP MENU

The drag and drop menu allows the user to create random tensors, upload tensors, and apply operations on created tensors. It is contained inside the **<ReactFlow.ReactFlowProvider>** element.

The drag and drop menu has a tag of **<DNDMenu>**. It also contains all the elements for the tensor operations.

The variables that are used for all the tensor operations are passed through the drag and drop menu element. The drag and drop menu also keeps track of which operation is currently active and only allows one active operation at a time. This was a design choice, but it is not necessary.

IV. TENSORS

Nodes in ReactFlow are created by making an object in JavaScript that has specific properties and passing it to the **setNodes** function. This function will create a node on the canvas with the specified values. The object must contain certain properties such as **id**, **position**, and **type**. There are properties that are optional such as **style** and **data**. There is no strict limit to the properties, but any custom properties should be passed through the value linked to the data property.

Two custom nodes were created to represent tensor networks. The two types are **tensor** and **invisible**. The code for these tensors can be found in **TensorNode.js** and **InvisibleNode.js**. To make the simplest node, an empty div element can be returned, but we want to be able to connect tensors. The **<ReactFlow.Handle>** element is used to create a handle that allows edges to be connected to it. So a custom node is created by creating a div and specifying the handles and any other desired HTML or React elements. The tensor type is used to represent a tensor in a tensor network. The invisible type is required for free indices because the package required edges to have two connection points.

When creating a tensor type node, the **data** value is an object that must include certain properties such as the **rank** as an Int, **mutable** as a boolean, **label** as a string, and **color** in the form of an Int from 1-100. Furthermore, the style property is an object that should include a **backgroundColor** property in the form of a string; it is currently "hsl(" + tensorColor + ", 100%, 80%)". If the tensor is being created by uploading then the data property must also include a **input** property as a string. The string is currently "", but this can be changed.

In the case of an invisible node, it is much simpler. The **data** value must contain a **mutable** property which should be a boolean value.

The rank of the tensor is equal to the number of handles for the tensor, so a tensor of rank 3 can have only have 3 connection (a connection to any type of node is considered a connection). Additionally, the IDs must be unique, so a **nodeId** variable is incremented every time a tensor node is created. Similarly, if an invisible node is created then its ID is 'f\${freeNodesId}' where **freeNodesId** is incremented every time an invisible node is created. If a node is deleted then all its edges will be deleted or split. If the tensor is connected to another tensor then that edge will be split otherwise the edge will be deleted. This is done by the **onNodesDelete** event.

V. INDICES

Connecting two tensors is done by dragging from one handle of a tensor to a handle of another tensor. Free indices are created by dragging from a handle of a tensor onto the network canvas. Similar to nodes, edges are created by making an object in JavaScript with specific properties and passing it to the **setEdges** function. This function will create the edges on the canvas. The object must contain certain properties such as **id**, **sourceX**, **sourceY**, **targetX**, **targetY**, **sourcePosition**, **targetPosition** and **type**. There are properties that are optional such as **style** and **data**.

Similar to the nodes, custom edges can be made. There is only one type of custom edge required which is of type **index**. The code for the edge type can be found in **DimensionEdge.js**. For an index type edge, the data value must contain the property **dim** which is a positive Int, **chosenDim** which is a boolean, and **lineStyle** which is "Bezier" or "Free". The dimension of an index is immutable, so once the user chooses the index then **chosenDim** becomes true. The **lineStyle** is "Free" if the edge is connected to an invisible type node otherwise it is "Bezier".

Since IDs must be unique, if a edge is created then its ID is 'e\${eId}' where **eId** is incremented every time an invisible edge is created. If an edge is deleted and it connected to an invisible node then the invisible node will be deleted as well. This is done by the **onEdgesDelete** function.

VI. CONTRACT

The contract operation accepts two different tensor IDs and contracts the tensors with these IDs. The contraction will cause the two tensors to be replaced with a new tensor that represents the contraction of the

original tensors. The two tensors are contracted along indices that are named the same. Therefore an index that directly connects two tensors will be contracted. If tensor one and tensor two both have an index labeled the same thing and they are the same dimension then that index will also be contracted along. This is done by deleting the given two tensors and all indices that are being contracted along. Any indices that are not contracted will still be connected to the new resulting tensor.

An error will occur if the contraction is invalid or too many indices have the same name, or either tensor does not exist.

VII. SVD DECOMPOSITION

The SVD composition accepts the ID of a single tensors and two groups of the tensor's indices. The decomposition will result in 3 tensors. The tensors will be two complex unitary matrices and a diagonal matrix. This is done by replacing the original tensor by these 3 tensors. The tensor that results from the first group of indices will be connected to the tensors that the original edges were connected to (excluding the original tensor) and the diagonal matrix. The same applied to the tensor produced by the second group of indices. The diagonal matrix is connected to the unitary matrices.

Errors will occur if the matrix does not exist or not all the indices are grouped properly.

VIII. QR/LQ DECOMPOSITIONS

The QR/LQ decomposition accept the same input as the SVD decomposition. The resulting tensors will be the Q(L) and R(Q) matrices if the QR(LQ) decomposition is done. The tensor that results from the first group of indices will be connected to the tensors that the original edges were connected to (excluding the original tensor). The same applied to the tensor produced by the second group of indices.

An error will appear if the original tensor does not exist or the indices are not grouped properly.

IX. EIGENVALUE DECOMPOSITION

The eigenvalue decomposition accepts the ID of a single tensor and returns three square matrices. These matrices are the diagonal matrix of eigenvalues, the matrix of eigenvectors, and the matrix of inverse eigenvectors. This is done by replacing the original matrix

with these square matrices and connecting them (in the order of eigenvectors-eigenvalues-inverse eigenvectors).

An error will occur if the selected tensor is not isolated or it is not a square matrix.

X. SPLIT EDGE

Splitting an edge does not change the tensors in the backend, but it makes the tensor network easier to understand visually. Splitting an edge takes the label of the edge as input. The tensors that the original edge is connected to will then be connected to an invisible tensor instead.

This is done by creating two new edges and two new invisible nodes. The new edges will have the same label as the original edge.

XI. CONNECT EDGE

Connecting an edge does not change the tensors in the backend, but it makes the tensor network easier to understand visually. Connecting edges accepts two edge labels that are connected to invisible nodes as input. It will then connect the tensor nodes directly. This is done by deleting the two original edges and replacing it with a single edge. The new edge will have the label of the first input edge.

An error will occur if the dimension of the indices do not match or one of the indices does not exist.

XII. COPY AND PASTE

Tensors are copied by selecting a group of isolated tensors and either clicking the copy button or using **command+c**. It does not matter if an edge or a invisible node is selected. An error will occur if the tensors are not isolated. First, the copy function will gather all the tensors that have been selected. It will then copy all the invisible nodes connected to the tensor nodes. The edges connected to all nodes are then copied. The copying step requires the IDs of all edges, and nodes to be updated as they must be unique. Furthermore, the target/targetHandle and source/sourceHandle must be updated for all the edges or they will connect to the original target and source.

Pasting what has been copied will add the saved nodes and edges to the network canvas. In the case that

the original tensor(s) need to be duplicated multiple times then the pasting function will preform a copy on the current copied contents. This guarantees that it is possible to duplicate the same thing multiple times. Otherwise the IDs would not be unique and thus would cause bugs.

XIII. UNDO AND REDO

A history is saved for each operation. The status of each tensor and edge is saved as a history, so it possible to revert the status. If a new operation is done then the previous history will be rewritten.

XIV. ADDING NEW OPERATIONS

Adding new operations requires the frontend to produce an accurate representation of the operation being done. It is easiest to use tensor IDs and edge labels as the input. Once those have been identified, it is important to determine which edges or nodes must be removed, updated, or added. It is possible to get the lists of nodes and edges using `const getNodes and getEdges`. It is then possible to set the nodes and edges using `setNodes and setEdges`. These functions can be called using `ReactFlow.useReactFlow()`.

XV. CONNECTING TO BACKEND

When an operation is done, it passes the name of the operation done and the JavaScript objects of the tensors and edges involved.

XVI. CHOSEN CONVENTIONS

- **invisible** nodes are **ALWAYS** the target node, never the source. Changing this will cause some errors in the code.

XVII. ADDITIONAL INFO

- Multiple elements can be selected by **holding command+clicking**
- React elements must start with a capital letter or errors may occur (this is a convention but it is easy to forget when learning)