Swim Onnx Model

The ACTUAL onnx model which is related to swim classification is composed upon 3 LSTM layers

An LSTM (Long Short-Term Memory) node in an ONNX graph represents a layer or cell of a recurrent neural network (RNN) model, specifically an LSTM architecture. ONNX, which stands for Open Neural Network Exchange, is a standard format for representing machine learning models, including neural networks. Here's a summary of what an LSTM node looks like within an ONNX graph:

# LSTM Node in ONNX:

* Name: A user-defined name for the LSTM node.
* Op Type: The operation type, which is typically "LSTM."
* Inputs:
  + Input X: The main input to the LSTM cell, often representing a sequence of data. This is where your input data, such as sequences of vectors or embeddings, is fed.
  + Input W: The weight matrix associated with the input data.
  + Input R: The weight matrix associated with the recurrent connections.
  + Input B: Biases for the LSTM gates.
* Initial Hidden State (optional): The initial hidden state of the LSTM cell. This can be omitted if not needed.
* Initial Cell State (optional): The initial cell state of the LSTM cell. Similar to the hidden state, this can also be omitted if not required.
* Outputs:
  + Output Y: The output of the LSTM cell, which often represents the processed sequence data.
  + Output Y\_h: The final hidden state of the LSTM cell after processing the entire sequence.
  + Output Y\_c: The final cell state of the LSTM cell.
* Attributes: Various attributes specific to the LSTM configuration, such as the number of hidden units, activation functions, and other hyperparameters that govern the LSTM's behavior.

# Current ONNX Model

## First Layer

Une image contenant texte, capture d’écran, diagramme, Parallèle

Description générée automatiquement

Figure 1First LSTM Layer with inputs

Has we see in the figure one, the LSTM layer is feed by two branches. One given input data (at the left) and the other given initial hidden and cell state (at the right).

### The Right Branch

The right branch is chaining several node to initialize both hidden and cell state. The transformation is performed using

1. Shape : Takes a tensor as input and outputs an 1D int64 tensor containing the shape of the input tensor
2. Cast : The operator casts the elements of a given input tensor to a data type specified by the 'to' argument and returns an output tensor of the same size in the converted type. <https://github.com/onnx/onnx/blob/main/docs/Operators.md#Cast>
3. Slice: Produces a slice of the input tensor along multiple axes. Similar to numpy. https://github.com/onnx/onnx/blob/main/docs/Operators.md#Slice
4. Concat : Concatenate a list of tensors into a single tensor. All input tensors must have the same shape, except for the dimension size of the axis to concatenate on. <https://github.com/onnx/onnx/blob/main/docs/Operators.md#Concat>
5. Cast : See above
6. Expand : Broadcast the input tensor following the given shape and the broadcast rule. The broadcast rule is similar to numpy.array(input) \* numpy.ones(shape). <https://github.com/onnx/onnx/blob/main/docs/Operators.md#Expand>
7. Unsqueeze : Insert single-dimensional entries to the shape of an input tensor (data). Takes one required input axes - which contains a list of dimension indices and this operator will insert a dimension of value 1 into the corresponding index of the output tensor (expanded).

For example, given an input tensor (data) of shape [3, 4, 5], then Unsqueeze(data, axes=[0, 4]) outputs a tensor (expanded) containing same data as data but with shape [1, 3, 4, 5, 1]. <https://github.com/onnx/onnx/blob/main/docs/Operators.md#Unsqueeze>

### The Left Branch

The left branch is only responsible to Transpose the input tensor.

1. Transpose : Transpose the input tensor similar to numpy.transpose. For example, when perm=(1, 0, 2), given an input tensor of shape (1, 2, 3), the output shape will be (2, 1, 3). <https://github.com/onnx/onnx/blob/main/docs/Operators.md#Transpose>

## Subsequent layers

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

Figure 2:subsequent layer

When dealing with subsequent layer, we still have 2 branches, which are inverted (Right Become Left and Left become Right ) compared to the previous layer. Note this disposition is purely graphical and has NO consequence on the model itself.

Prior to branching we performs 2 operation which are

1. Squeeze : Remove single-dimensional entries from the shape of a tensor. Takes an input axes with a list of axes to squeeze. If axes is not provided, all the single dimensions will be removed from the shape. If an axis is selected with shape entry not equal to one, an error is raised. <https://github.com/onnx/onnx/blob/main/docs/Operators.md#Squeeze>
2. Transpose : See Above

However, on the figure 2 Right branch we see a 2 consecutive Transpose operations with the same permutation parameter [1,0,2] which means a void operation.

The Left branch to initialize forget and cells remains unchanged.

# Recommendation of changes

Since the current input is a tensor of dimension 3, with an unknown first dimension. And the current inference implementation is only supporting tensor of dimension 2, we MUST change the graph to accept only [13,16] tensor (or [16,13]).

Suppress the branches to initialize and replace it, with pre-defined inputs initializer.

Avoid Transpose as possible, and certainly consecutive Transpose.

# Need of unit test

In order to unit testing the graph, we need to provide a set of csv file with the main input and output, along with the intermediary outputs of each node.