**Generalizing GRU to graph input**

*Introduction*

The purpose of this document is to generalize the structure of a GRU recurrent unit – which generally combines an input vector at time t with a memory vector at time t-1 to produce a memory vector at time t (see figure 1) – to a model which combines a *graph* at time t with a memory vector at time t-1 to produce a memory vector at time t. The generalization is based on the observation that all recurrent memory cells (even a simple RNN) combine the incoming memory and the incoming input *linearly* to produce the outgoing memory. Thus, every component of the new memory is a linear combination of all of the old memory components and all of the input components. With a graph it is only necessary to enumerate all of its components and define weights that map each to the new memory components.

The objective is to design a model architecture which preserves the graph structure created with the similarity and spatial edges and to analyze their time-dependence by inputting each graph at each time t into a GRU unit in the same way that sequences of vectors are time-analyzed.

The problem is complicated somewhat in that the elements of a graph are edges (which can be thought of as matrices of real numbers) and node (or vertex) labels. The labels are themselves vectors representing the node features (e.g. “boy,” “trampoline,” etc.). Thus it is useful (for writing clarity) to calculate node and edge influences separately. Similarly, since there are two types of edges (meaning that the edges carry a label) we must define separate weights for each type (although we could possibly share weights to see if that works).

*Basic theory*

Let be a graph at time , such that with a set of vertices and a set of edges labelled by (both vertices and edges implicitly time dependent).

Let each vertex be given by a one-hot class label vector: , where m is the mth word in vocabulary V. Therefore the set of vertices is equivalent to the set of indices of the class labels, .

Let the edges be given by:

where the overlaps and are defined in Chang Liu’s presentations.

The standard form of a gated recurrent unit is illustrated in figure 1.

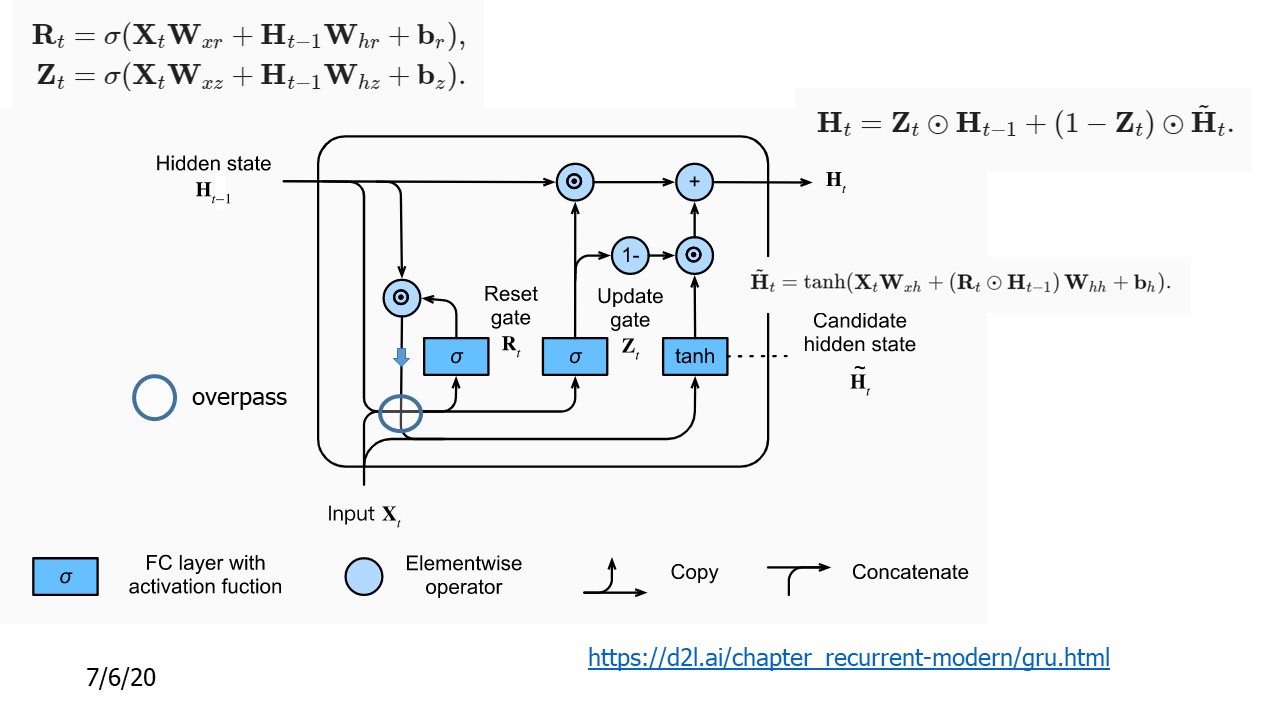


Figure 1 Standard GRU unit showing reset and update gates calculated from memory at t-1 and input at t.

The outputs of the reset and update gates, and are linear combinations of the elements of the incoming memory vector (hidden state) and the input vector . We will call the dimension of the memory vector m. Since and are multiplied elementwise with they must also have dimension m. Therefore, using n (lower case) to denote the size of the input vectors , we have the dimensions of and as n x m and the dimensions of and as m x m.

We propose to modify the GRU architecture by using the time dependent graph as the input in place of as follows. The edges of the graph are represented by two ordinary matrices with zero diagonals and above. The elements of these matrices are simply real numbers. The vertices of the graph, however, are features and they are represented by vectors. Beginning with these features (which are essentially vertex labels) we proceed as follows.

For the reset gate, define the class label weight matrix and the class-reset components as:

and similarly for the update gate we have a set of weights giving update elements:

Note that we are simply adding together (sum on i) all of the feature vectors. However since these are one-hot vectors this is not an instance of weight sharing. Each feature can be understood as a component of the feature vector.

The contribution of the vertices to the reset and update gates should be included independently. Furthermore, in order for the model to learn the graphs and their evolution it is ideal (and perhaps necessary) that each element of the memory vector be linearly connected to each vertex and edge of the graph. Thus, weight “matrices” , , and are in fact third rank tensors:

Here q is the dimension of , sometimes called the “number of neurons.”

The full operation for is indicated in figure 2. The equation for follows analogously.

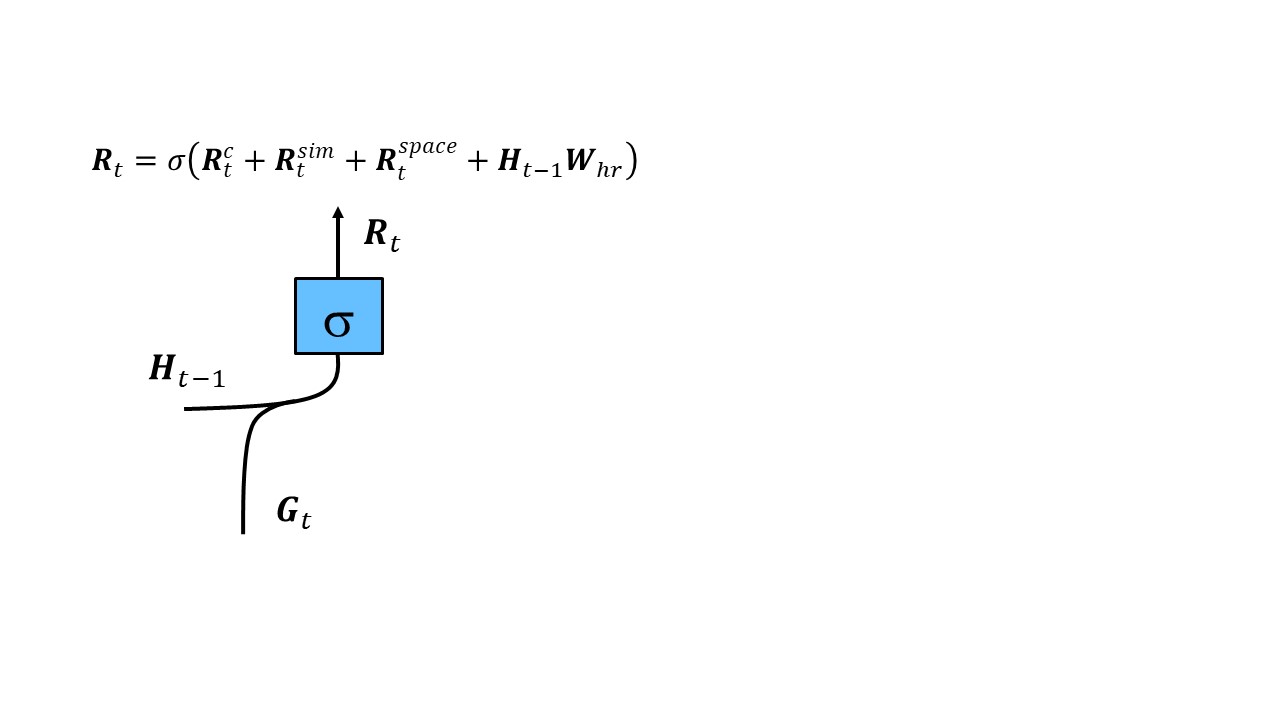


Figure 2 Combining the graph at time t with the memory vector at time t-1 in the reset gate.

Finally, in the GRU the output of the reset gate is multiplied elementwise with the memory state and combined with the weighted input as shown in figure 1. Analogous to the treatment of and we define the calculation of the candidate hidden state as (we have suppressed the biases throughout):

where

A few comments.

First, while it would be much simpler to take the feature vectors directly and use them in a GRU layer in the standard fashion, the construction of graphs at each time step and the proximity and similarity measures given by the graph edges are critical information that would be lost in that approach.

Second, the number of additional weight parameters could become very large if the matrices themselves – and specifically the number of vertices – becomes large. However, if typical node numbers remain small (N~10) then the number of weight parameters will scale as (m is the memory size) and might not be prohibitive. If the size of the matrices is prohibitively large it is possible that some kind of pruning method could be used to scale down the problem.

Note that I have only generalized the simple GRU unit. More recent work takes the output of a set of recurrent memory cells, be they GRU or LSTM, from all times and then combines these in some form of attention unit. Because the way I generalized the GRU results in exactly the same output as a normal GRU (i.e. the output is a vector with a certain number m of “neurons”) the process of computing attention should go through in the same manner.

Finally, there is no built-in ML algorithm for this kind of GRU unit and one would have to be coded and inserted into Caffe or Tensorflow. This includes the code for back propagation to learn the weights. However, a crucial feature of this method is that all weights and graph inputs are incorporated linearly. Thus only minor modifications should be needed to the standard GRU backpropagation techniques.