1 Introduction

1.1 GRU

Modifying architectures to include transformers has lead to dramatic gains in many domains. For this project, I wanted to see if I could adapt a transformer to work with a GRU.

GRU's have a simple architecture. They take in a hidden state and an input to generate a new state. There structure is graphically depicted in figure 1.

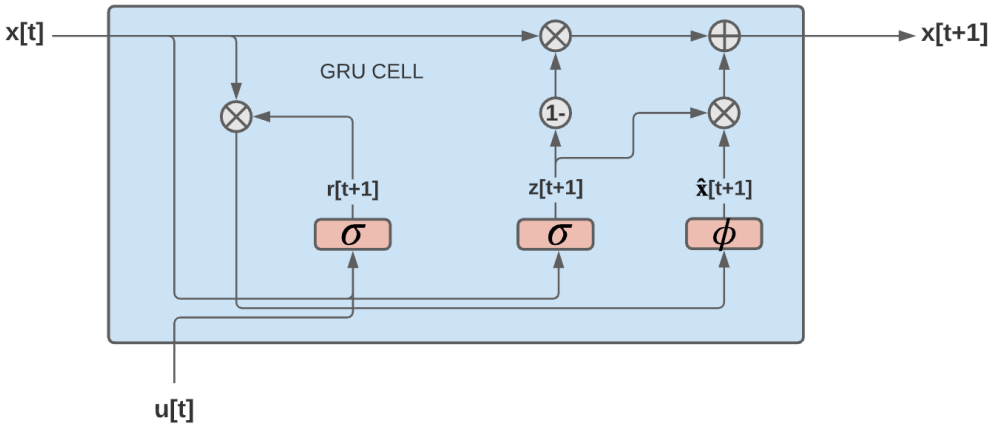

Figure 1: GRU Cell.

Mathematically, a GRU Cell is described exactly as:

$$\begin{aligned} z[t+1] &= \sigma(A_z x[t] + B_z u[t] + b_z) \\ r[t+1] &= \sigma(A_r x[t] + B_r u[t] + b_r) \\ \hat{x}[t+1] &= \phi(A_h(r[t+1] \odot x[t]) + B_h u[t] + b_h) \\ x[t+1] &= (1 - z[t+1]) \odot x[t] + z_t \odot \hat{x}[t+1] \end{aligned} \tag{1.1.1}$$

And the elements come from the following space $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, and $b \in R^n$. Note that the vectors b are optional and referred to as "biases".

The operations are defined as follows:

$$\begin{aligned} \sigma(x) &= \frac{e^x}{e^x + 1} = \text{sigmoid} \\ \phi(x) &= \frac{e^{2x} - 1}{e^{2x} + 1} = \text{hyperbolic tangent} \\ \odot &= \text{Hadamard Product (element-wise multiplication)} \end{aligned}$$

For simplicity, let's let $\text{GRU}(u[t], x[t]) = x[t+1]$ represent 1.1.1. We see that while $\text{GRU}(\cdot)$ is flexible, it has restrictive bounds on what transformations it can do to $x[t]$ when generating $x[t+1]$. While these restrictions on $x[t+1]$ act as a regularizer, it may prevent the GRU from tracking important features in temporal data.

1.2 Transformers

Transformers have produced SOTA results for sequencial data. Generaly speaking, transformers are very flexible and while that implies they can fit many output spaces, they also require a lot of data to prevent overfitting.

Transformers work largely through a mechanism called multi-headed self-attention.

While attention is introduced in the literature in a different form, attention is described mathematically as:

$$\text{attention}(x) = \text{softmax}\left(\frac{xMx'}{\alpha}\right)xW \tag{1.2.1}$$

Where $x \in \mathbb{R}^{m\times n}$, $M \in \mathbb{R}^{n\times n}$, and $W \in \mathbb{R}^{n\times q}$ and $\alpha = \sqrt{q}$ which is a that helps stabilize the gradient during backprop.

Multihead attention implements multiple attention mechanisms in a system called "attention heads". A three headed attention mechanism can be described as:

$$\begin{aligned}\text{mha}(x) &= \text{softmax}\left(\begin{bmatrix}x & x & x\end{bmatrix}\begin{bmatrix}M_1/a & 0 & 0\\ 0 & M_2/a & 0\\ 0 & 0 & M_3/a\end{bmatrix}\begin{bmatrix}x' & 0 & 0\\ 0 & x' & 0\\ 0 & 0 & x'\end{bmatrix}\right)\begin{bmatrix}x & 0 & 0\\ 0 & x & 0\\ 0 & 0 & x\end{bmatrix}\begin{bmatrix}W_1 & 0 & 0\\ 0 & W_2 & 0\\ 0 & 0 & W_3\end{bmatrix}U\\ &= \begin{bmatrix}\text{attention}_1(x) & \text{attention}_2(x) & \text{attention}_3(x)\end{bmatrix}U\end{aligned} \tag{1.2.2}$$

Where $U \in \mathbb{R}^{n\times(3q)}$ and serves to aggregate important information found by each head. The other matrices in composed of unique instantiations of 1.2.1. In the generalized case we extend the pattern seen in 1.2.2 and $U \in \mathbb{R}^{n\times(q*h)}$.

Though transformers are flexible, they have no state variables. This means that transformers cannot account for dynamic behavior that cannot be described by the window of input data. To overcome this, a mechanism called time2vec was developed which, in theory, can find a multi-dimension time-embedding to help account for time-trends in time-series data. This learned embedding has had varying degrees of success.

1.3 Feature RNN

To assist a model with learning temporal features, our proposed network is the feature recurrent neural network (FRNN). An outline of the architecture is shown in figure 2.

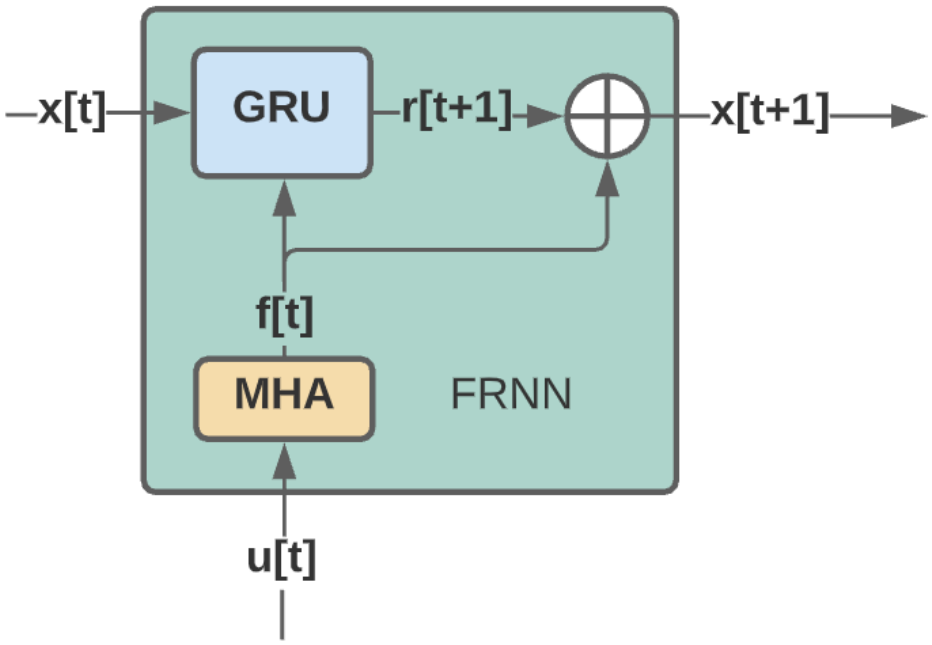

Figure 2: GRU Cell.

The FRNN builds off its predecessors so it can be understood in 3 steps:

1. FRNN accepts a new input from which it extracts relevant time-varying features.

$$\text{mha}(u[t]) = f[t] \tag{1.3.1}$$

2. FRNN combines these new features with the hidden instate to produce a new state variable

$$\text{GRU}(x[t], u[t]) = r[t+1] \tag{1.3.2}$$

3. Attention mechanisms are data hungry, so to help gradient flow the the multi-headed attention unit, the hidden state is added added and normalized (not depicted) by the current extracted features.

$$\begin{aligned} z[t] &= f[t] + r[t+1] \\ x[t+1] &= \frac{z_t}{\|z_t\|} \end{aligned} \tag{1.3.3}$$

2 Results

Training the model follows the pseudo-code outlined in 1. When I initially constructed the trianing loop, I tought that the training process would be fairly quick because I excluded the start warmup from the gradient, however each epoch was estimated to take over 3 weeks to train with $5,236$ batches of training data. Needless to say, this was prohibitively slow.

The next step would be to take a random sample of batches and enable multiple batches to be trained at a time. This is a non-trivial task because the warm-up data needs to be accessible for each batch. Perhaps this will become a summer project.

Algorithm 1 Training Loop

```
for Epoch in 1:Epochs do
  for i in 1:length(training) do
    x ← Model.init_state
    {warmup state}
    for j in 1:i do
      y, x ← Model.predict(data(j), x)
    end for
    Model.fit(data(i), x)
  end for
end for
```

Code is a available at: github.com/loganbnielsen/FeatureRNN