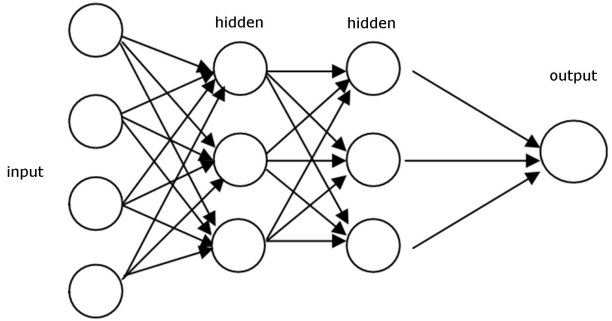
Introduction to RNNs and ML in Finance

Team Chairun - Chirag Sharma, Varun Jadia, Aaron Sun, Chethan Bhateja

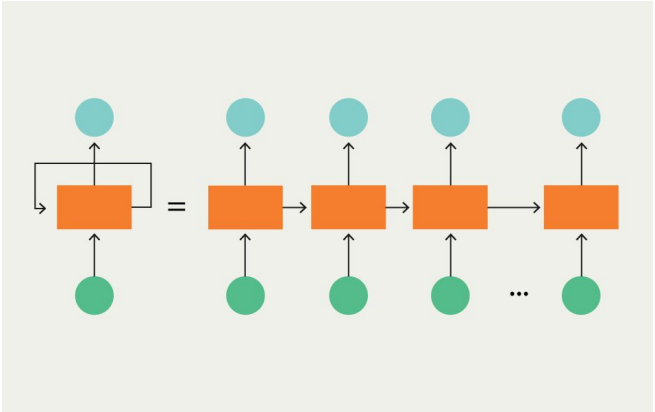
Table of Contents

Introduction to RNNs

- In Feed forward neural networks (pictured to the right), the network is acyclic, i.e. datapoints are propagated independently of each other, which makes them non-optimal for sequential data
- RNNs are a class of neural networks built to fix this flaw: they model sequence data by using sequential memory to their advantage
- They do this by introducing cycles into node connections
- RNNs are used extensively in NLP for sentiment analysis, translation, etc.



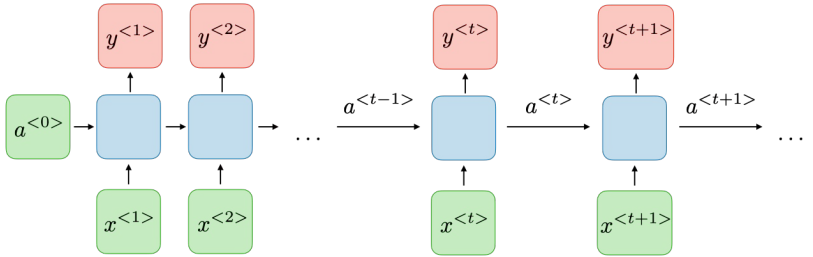
Feed forward architecture



RNN architecture

Introduction to RNNs - Continued

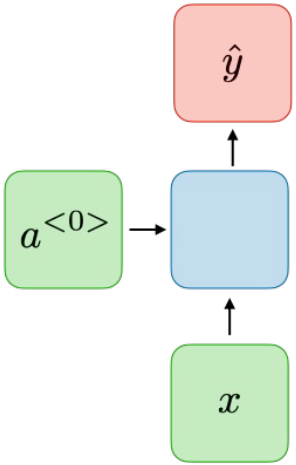
- The information captured in the loop, i.e. a snapshot of previous outputs is known as the **hidden state.**
- Mathematically, the propagation can be expressed as follows:



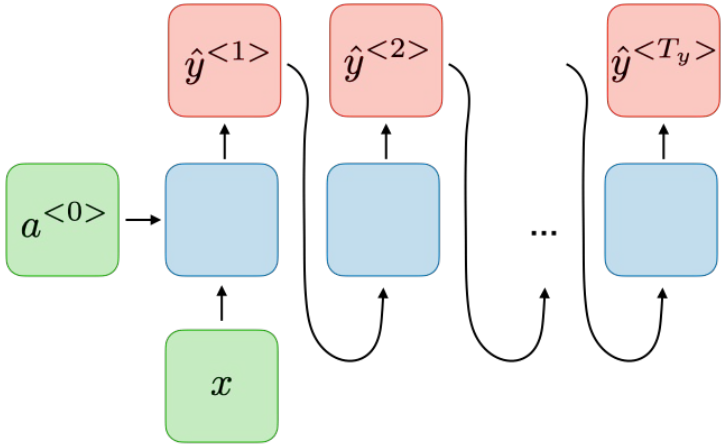
$$\mathbf{a}^{\langle t \rangle} = \mathbf{g}_1(\mathbf{W}_a \mathbf{a}^{\langle t-1 \rangle} + \mathbf{W}_x \mathbf{x}^{\langle t \rangle})$$
$$\mathbf{y}^{\langle t \rangle} = \mathbf{g}_2(\mathbf{W}_y \mathbf{a}^{\langle t \rangle})$$

- Here, g1 and g2 are activation functions, and W_a, W_x and W_y are weight matrices shared across layers

RNN types based on I/O

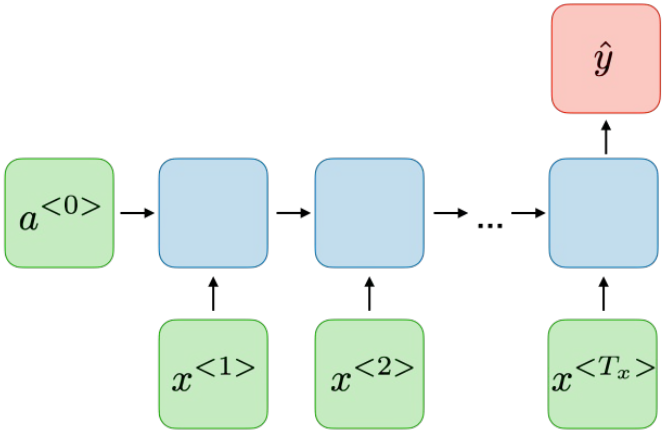


One to One, i.e. dim(x) = dim(y) = 1

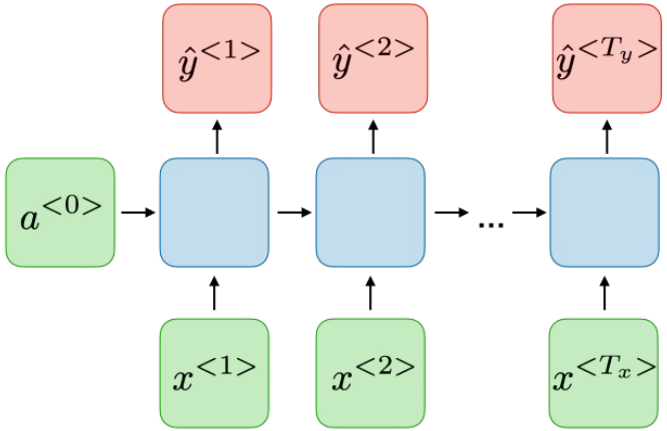


One to Many, i.e. dim(x) = 1 < dim(y)

RNN types based on I/O - Continued



Many to One, i.e. dim(x) > dim(y) = 1
(This is the RNN you build from scratch)



Many to Many, i.e. dim(x) and dim(y) > 1
(Used in Warm Up Assignment!)

Backpropagation in RNNs

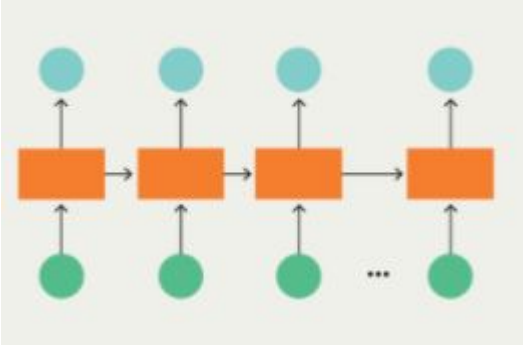
- RNNs, like Feed Forward networks, use backpropagation to calculate the gradients for the gradient descent updates, with one minor change: the backpropagation is carried out independently for each step (**backpropagation through time**)
- We also define our loss function at each time step, and hence total model error at time T is given by :

$$\mathcal{L}^{\langle T \rangle} = \sum_{t=1}^{T} \mathcal{L}(\hat{\mathbf{y}}^{\langle t \rangle}, \mathbf{y}^{\langle t \rangle})$$

- Also note that the gradient of the overall loss function w.r.t to the weight matrices is given by the sum of the gradients of the loss function at each time step

Vanishing/Exploding Gradient Problem

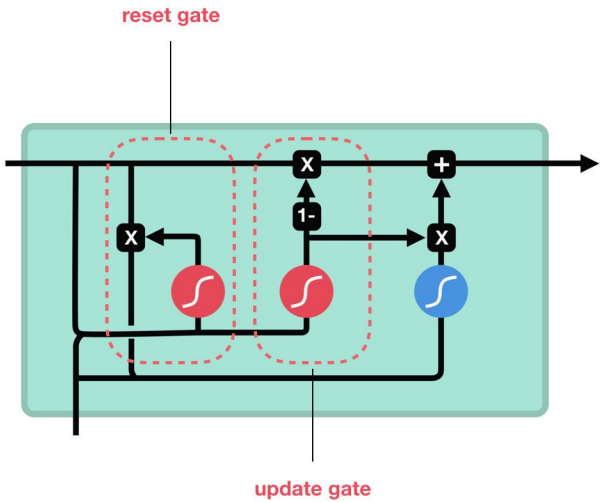
- A big problem that RNNs face is that they are unable to capture long-term dependencies in sequences
- This happens because due to the multiple matrix multiplications in backpropagation - smaller weights at the start get exponentially smaller, and vice versa
- Once the weights become too small, gradient descent can no longer update weights, which prevents further learning from taking place



An unrolled RNN, you can think of each layer adding a multiplicative factor to the backpropagation calculation

Gated Recurrent Units (GRUs)

- The GRU architecture adds an **update gate** vector and a **reset gate** vector to fix the problem discussed in the previous slide.
- The reset gate determines how much of past information is to be included in current output calculation, and the update gate determines how meaningful the current output is to future calculations



Gated Recurrent Units - Continued

- **Update Gate**:

$$\mathbf{z}^{\langle t \rangle} = \sigma(\mathbf{W}_z \mathbf{x}^{\langle t \rangle} + \mathbf{U}_z \mathbf{h}^{\langle t-1 \rangle})$$

  - The update gate introduces weights U_z and W_z and performs the above computation. H^{t-1} contains information about the previous t-1 elements
  - This weighted sum involving these new weights is passed into the sigmoid activation function

- **Reset Gate**:

$$\mathbf{r}^{\langle t \rangle} = \sigma(\mathbf{W}_r \mathbf{x}^{\langle t \rangle} + \mathbf{U}_r \mathbf{h}^{\langle t-1 \rangle})$$

  - The reset gate introduces weights U_r and W_r and performs the above computation. H^{t-1}, like previously, contains information about the previous t-1 elements
  - This weighted sum involving these new weights is passed into the sigmoid activation function

Gated Recurrent Units - Continued

- **Candidate Activation Vector**:

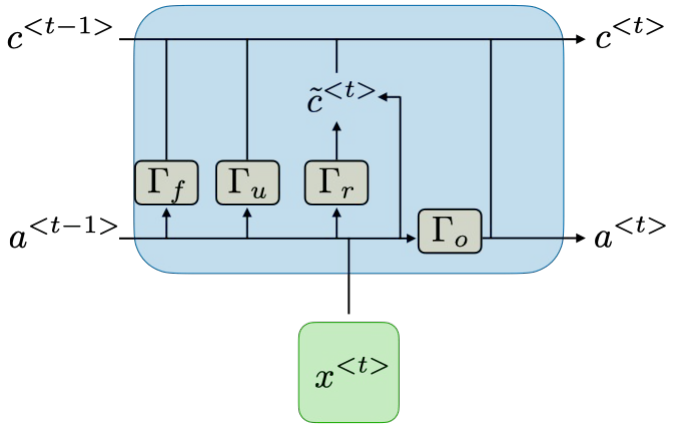
$$\hat{\mathbf{h}}^{\langle t \rangle} = \tanh(\mathbf{W}_{\hat{h}}\mathbf{x}^{\langle t \rangle} + \mathbf{r}^{\langle t \rangle} \odot \mathbf{U}_{\hat{h}}\mathbf{h}^{\langle t-1 \rangle})$$

  - The candidate activation vector introduces W_h and U_h and uses the reset gate output as well
- The overall output of the GRU is a weighted sum involving all these terms that encodes how much past information to pass on:

$$\mathbf{h}^{\langle t \rangle} = \mathbf{z}^{\langle t \rangle} \odot \mathbf{h}^{\langle t-1 \rangle} + (\mathbf{1} - \mathbf{z}^{\langle t \rangle}) \odot \hat{\mathbf{h}}^{\langle t \rangle}$$

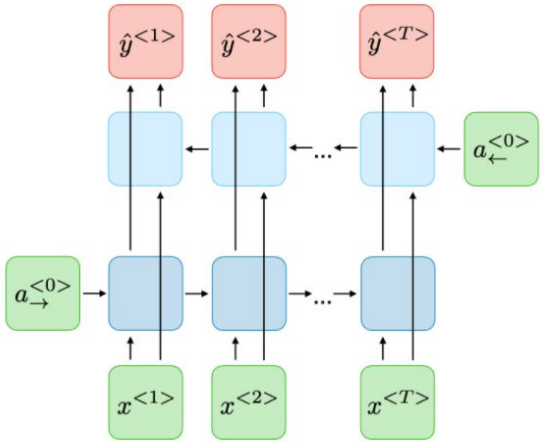
An Aside on Long Short Term Memory (LSTMs)

- LSTMs can be thought of as a generalization of GRUs
- Instead of having only a reset and an update gate, an LSTM has three gates: input gate, output gate and forget gate
- GRUs are easier (less computationally intensive) to train, but LSTMs tend to perform better for long term dependencies



Bi-Directional RNNs

- In cases where terms in a sequence depend on terms after it (instead of only before it), we make a simple addition to the conventional RNN structure to account for this
- As seen in the figure, we add an additional sequential network going in the opposite direction
- This addition can be made to GRUs and LSTMs as well, since they are both forms of RNNs



Overview of the Project

- Predicting stock prices is one of the biggest challenges in quantitative finance, and with the recent explosion of ML, several investors have begun to incorporate modern techniques, like neural networks into attempts at ‘beating’ the market
- However, these have mostly obtained mixed results (link some paper here)
- Instead of looking at it from the prediction point of view, what if we treated the problem from the classification point of view instead?

Getting Started

- The project notebook can be found in the assignments folder in the repository. It starts with a toy example to introduce simple RNNs and other architectures like LSTMs and Bi-Directional LSTMs, and how to work with them using TensorFlow
- Then, we build a stock price movement classifier:
    - First, data cleaning and processing - all required embeddings and stock price data should be available in the folder to start you off
    - Second, a description of the model and generating the required inputs
    - Third, constructing the model using keras and evaluating it
- There is also a note that elaborates on the theory behind RNNs, and a short list of quiz questions that can be used to test understanding

Sources

- https://www.aclweb.org/anthology/2020.acl-main.307/
- http://harinisuresh.com/2016/10/09/lstms/
- https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks