Using Long Short Term Memory Neural Network to Predict Bitcoin Stock Prices

Authors: Jaron Ritter, Cole Kassing

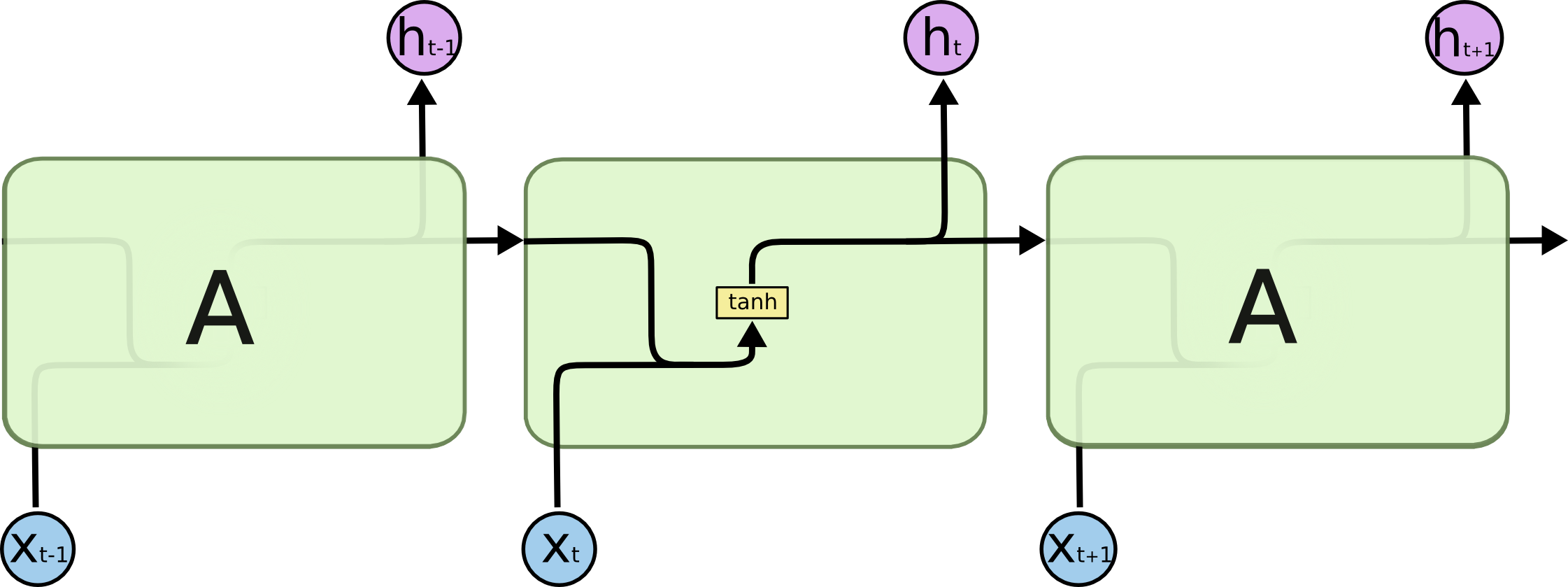
Abstract

The prerogative of our project was to try and accurately predict Ethereum stock prices with neural networks under supervised learning. Our primary intention was to use Recurrent Neural Networks (RNN). However, we eventually decided to move away from standard RNN and towards Long-Short Term Memory (LSTM) neural networks. In addition to using LSTM, we investigated the features and data input required to give us the most accurate predictions. We used six features: high minus low, close minus open, seven-day moving average (MA), fourteen-day MA, twenty-one-day MA, and seven-day standard deviation. In our experiments, we used three different forms of LSTM construction to evaluate what would perform best. All our models used the loss function mean squared error and the optimizer Adam, which contains a learning rate of 0.001.

1. LSTM

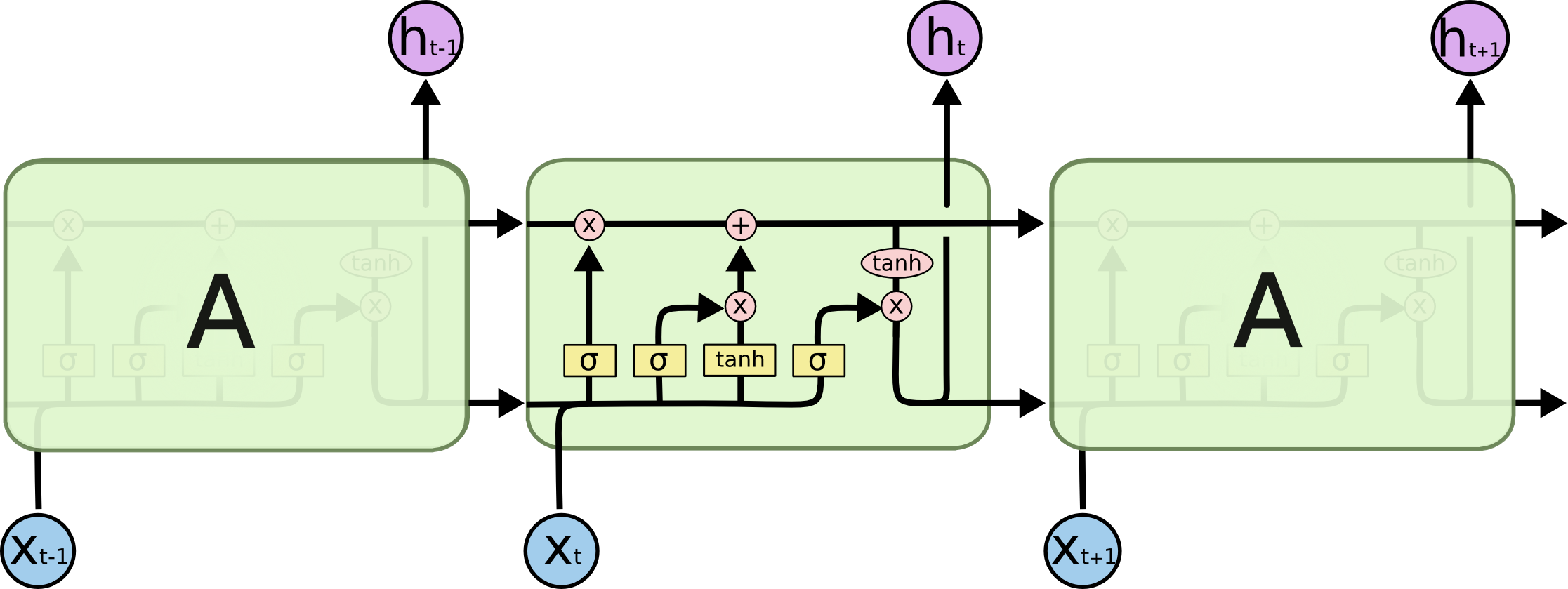
RNN is a great tool to predict sequential data like image recognition and language recognition. However, even though effective, the typical RNN is not as powerful as LSTM. (1) The main obstacle in RNN is when one adds hidden layers and time increases. Known as the “Vanishing Gradient Problem,” when the network becomes deeper and more hidden layers have been added, the value of the gradient becomes lower and lower, affecting the training process. Consequently, the model will not complete the training. (2) On top of a vanishing gradient, an exploding gradient can also occur, resulting in weights overflowing. Furthermore, as input is fed into an RNN, relevant information becomes harder to connect. As a result, long-term dependencies become extremely difficult to deal with even though they are theoretically possible (3).

1.1. Typical Repeating module in standard RNN

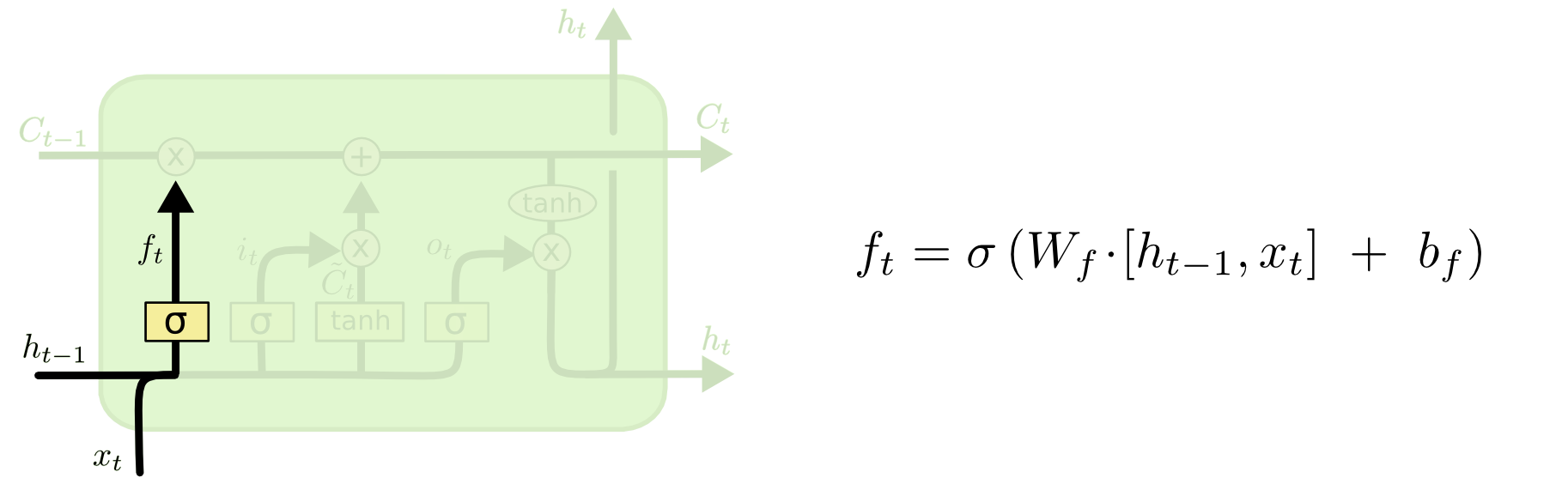
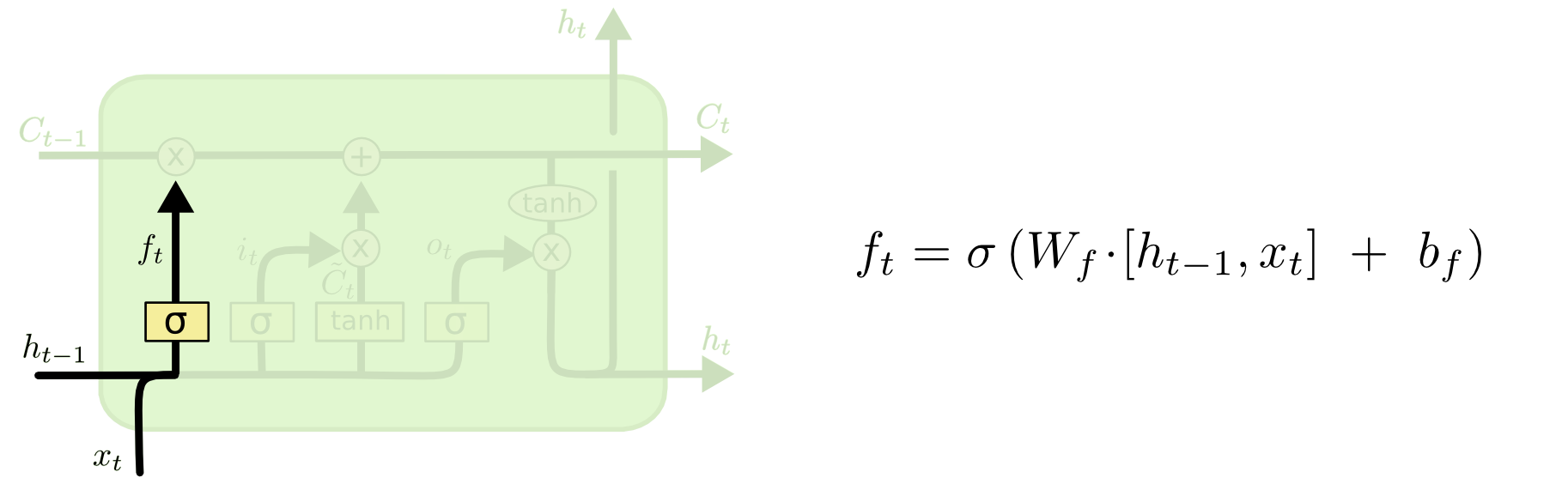


Each block above represents a timestep in a particular sequence, part of an input vector. Tanh represents the activation function for a timestep value, represented above in the cell. As timesteps of the input vector are inserted, the activation function also moves down, allowing previous outputs to be used as inputs while having hidden states. The previous activation function, in conjunction with the new timestep value, creates a new activation function. One thing to note about RNNs is that they share parameters within a module. The weight or bias would not change within this module, but the weight would still be adjusted in gradient descent/backpropagation as multiple modules of data are introduced. Effectively making combinations of neural networks in a loop. (3-5)

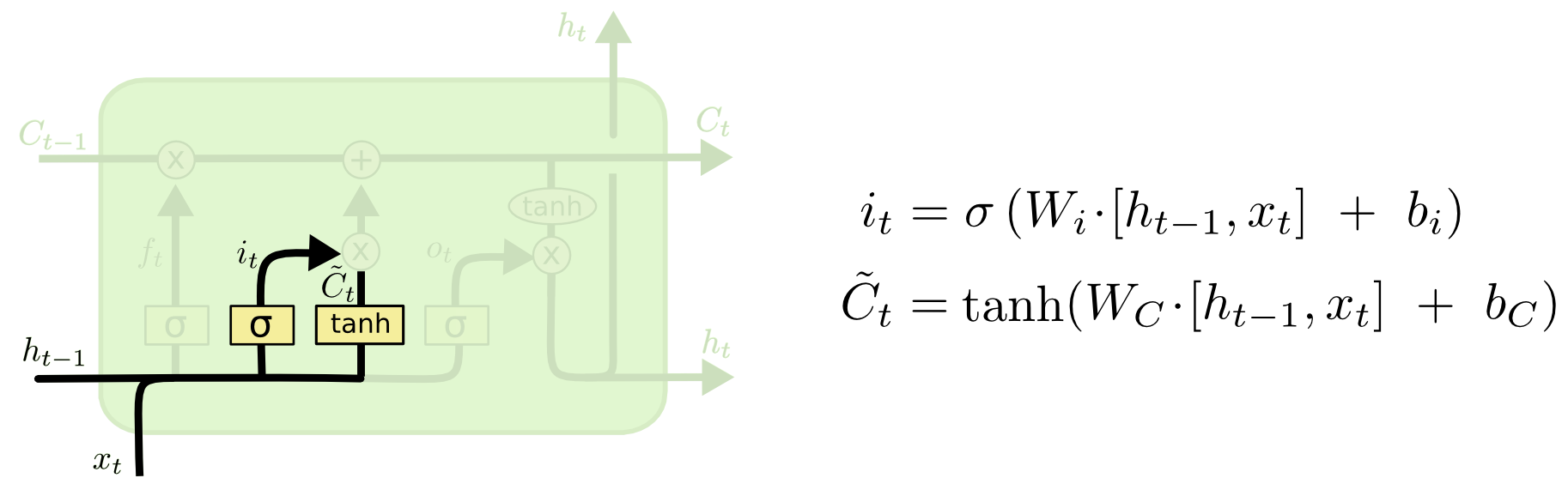
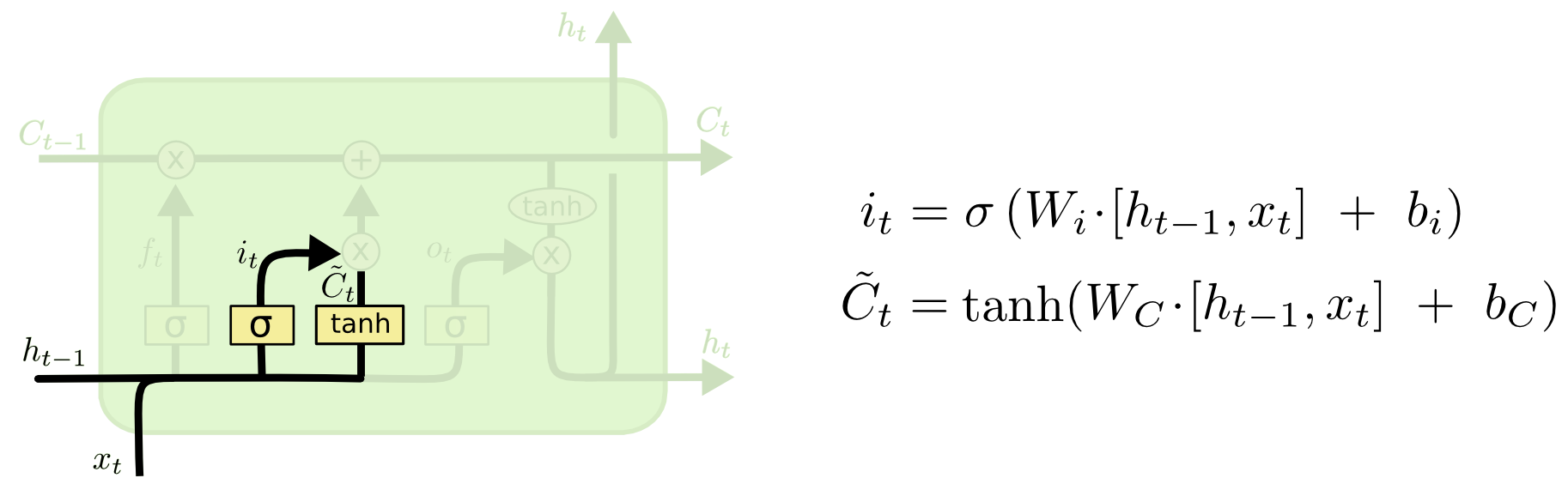
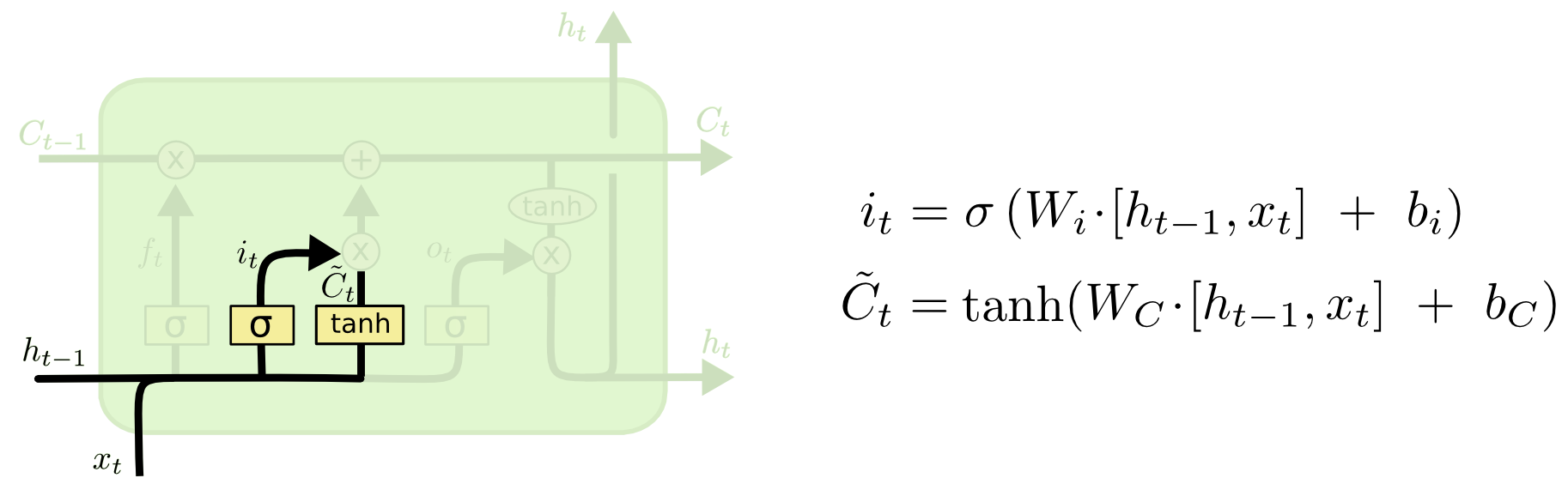
1.2 Repeating Module in Standard LSTM

While retaining similar rules such as sequenced inputs, and static weight/bias within a module, there are a lot of changes. A typical LSTM module consists of three gates. First, a forget gate uses a sigmoid layer to determine what information to forget from the previous sequence. Second, an input gate that uses a sigmoid layer in conjunction with an activation function on a new timestep value produces new information to the cell. Finally, an output gate that contains another sigmoid layer in conjunction with another activation function of the cell state. The cell state is the information retained after input and forgets operations have occurred in the cell. (3,6)

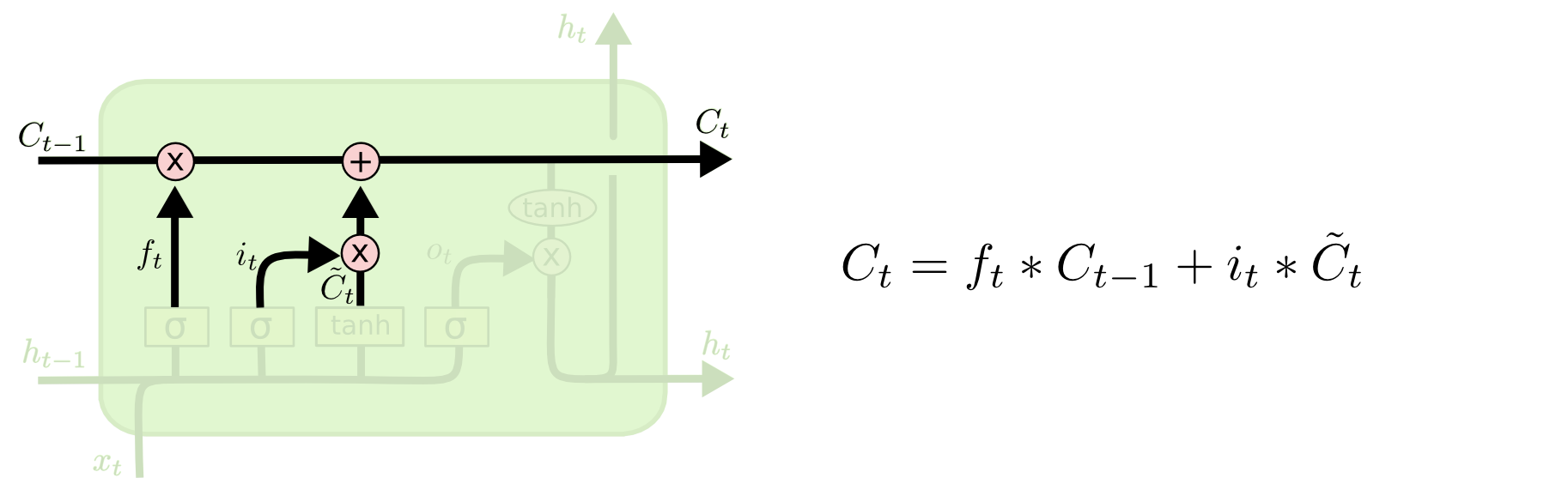
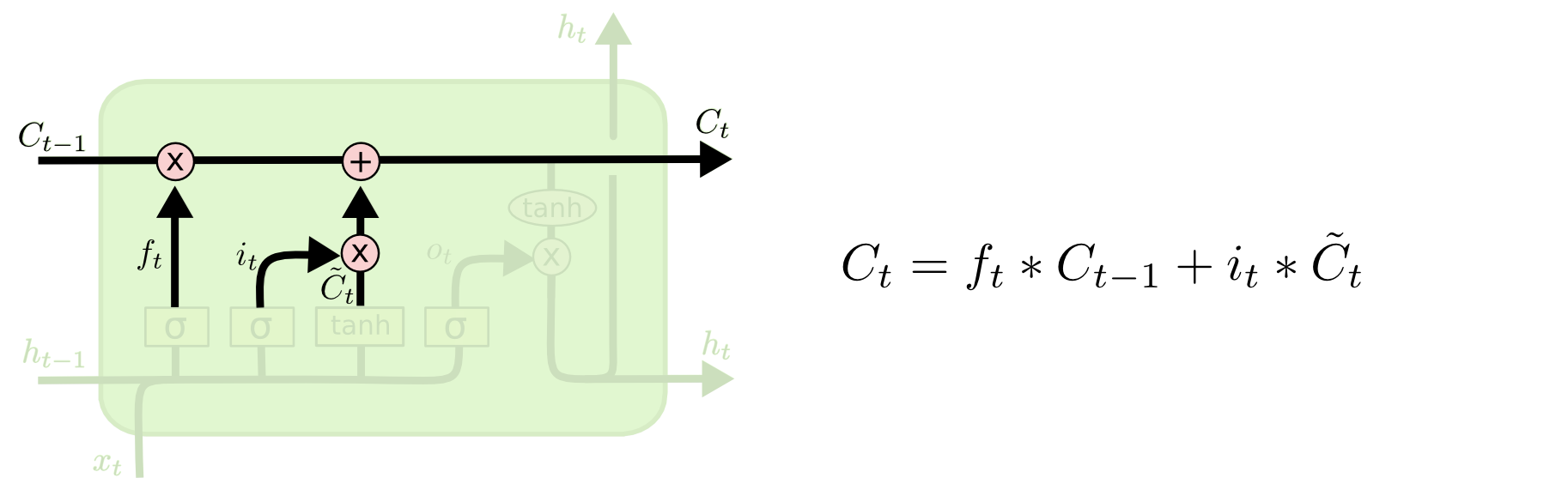
Forget Gate

The output of the sigmoid layer, in combination with the new input(x), previous value(h), weight (W), and bias (b), produces a value between 0 and 1. This value (ft) is used later in conjunction with the input layer on the cell's "Memory."   
(3)

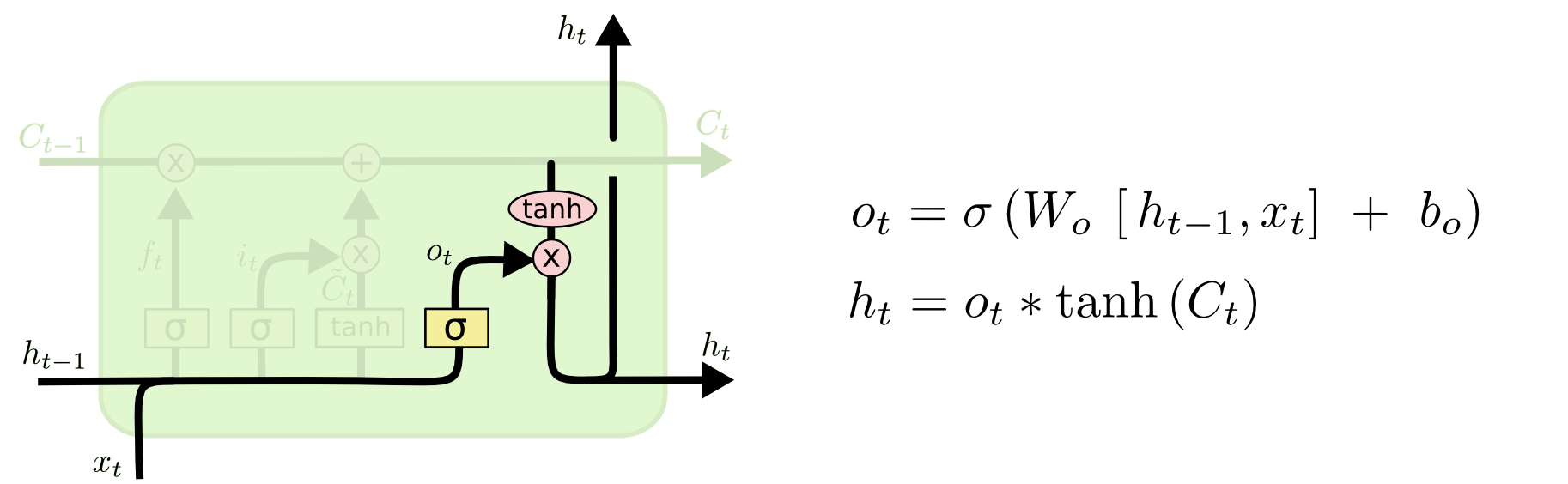
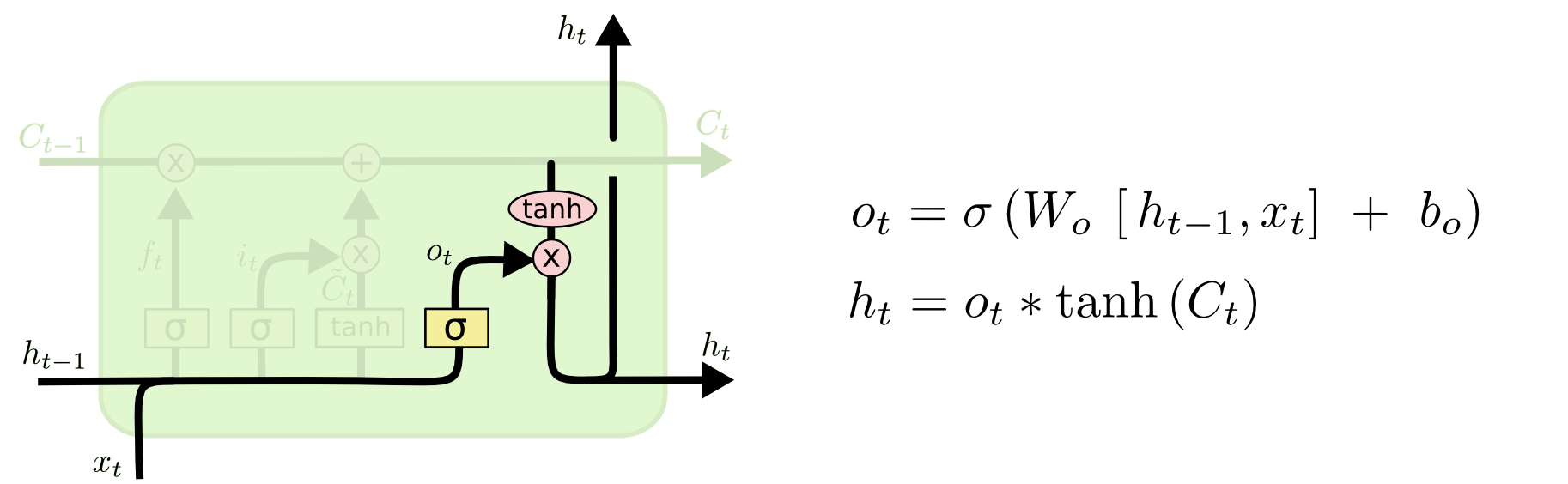
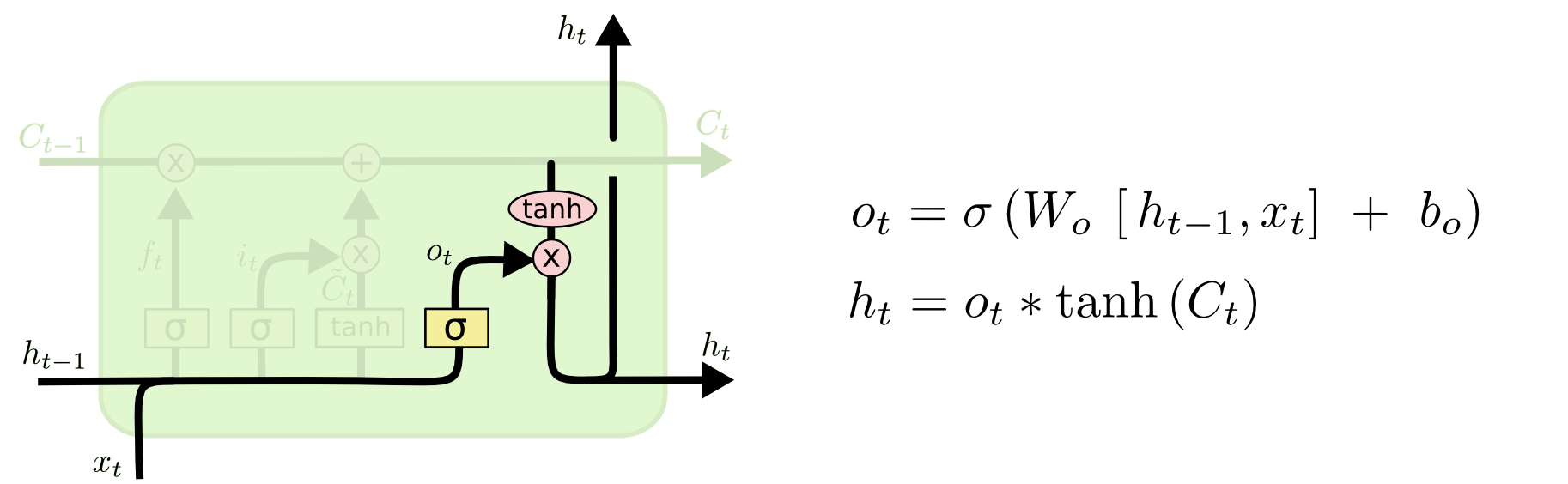
Input Gate and New values

The input gate Determines the new information to be kept in the cell state. The sigmoid layer in this figure is the input gate and will produce a value between 0 and 1 (It).  A new set of state values needs to be produced for the input gate to work, and an activation function (tanh) will create our new candidate values (~C) for the current state.   
 (3)

Modifying the Cell state

The Cell State serves as a conveyor belt for the transition of timesteps in the module. The (X) on the cell state represents the forget value multiplied by past information, ergo keeping, or removing it. The (+) represents an additional function where the possible state values(~Ct) are added with past values kept from forget operation (ft \* Ct-1) to find a new cell state (Ct)(3)

Output Gate

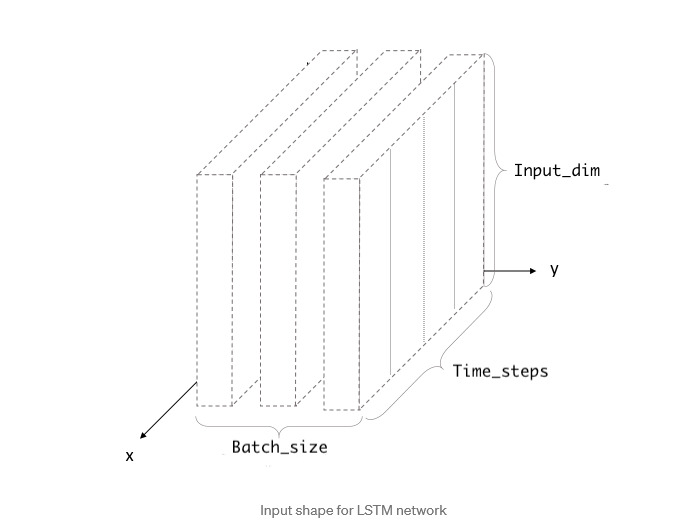
The output gate is based on a sigmoid layer producing a value between 0 and 1 (Ot). While an activation function runs on the new cell state (Ct) and then multiplies it against the output gate (Ot). This new value is passed to the next timestep.(3)

2. Data

For our dataset, we used a comma-separated file with one-hour interval data of Ethereum from 07-08-2015 to 16-04-2020 (7). To ensure that our data did not need to be cleaned, we checked the noise of the data. After some investigation, we concluded that volume was the only data with too much noise.

2.1 Feature Choice

Upon investigation, one finds that moving average (MA) is a good feature. Since our data is passed as a series of discrete points, having an average that changes when moving forward in time makes changes not as dramatic. (8) We decided to use three different MAs: **seven-day, fourteen-day, and twenty-one-day**. We also found that three other features had positive results: **close-minus-open, high-minus-low, and seven-day standard deviation** (9). While also using our target for training being the next value hour closing price (9). After calculating all feature values, we standardized them to be between values of 0 and 1.

2.2 Training and Testing Data

When using sequential time-based data involving any RNN, a single input is an array containing many different values over a specific interval. We use more than one feature for a batch in our model, so we need to pass a 3-d array as our training data.  
  
**Time\_steps** = number of respective intervals (hours in this case) per feature in a batch.   
  
**Input\_dim** = number of features in an input   
  
\*The two above dimensions create a 2-D array as an input module  
  
**Batch\_size** = number of inputs modules in the dataset, making a 3-D array

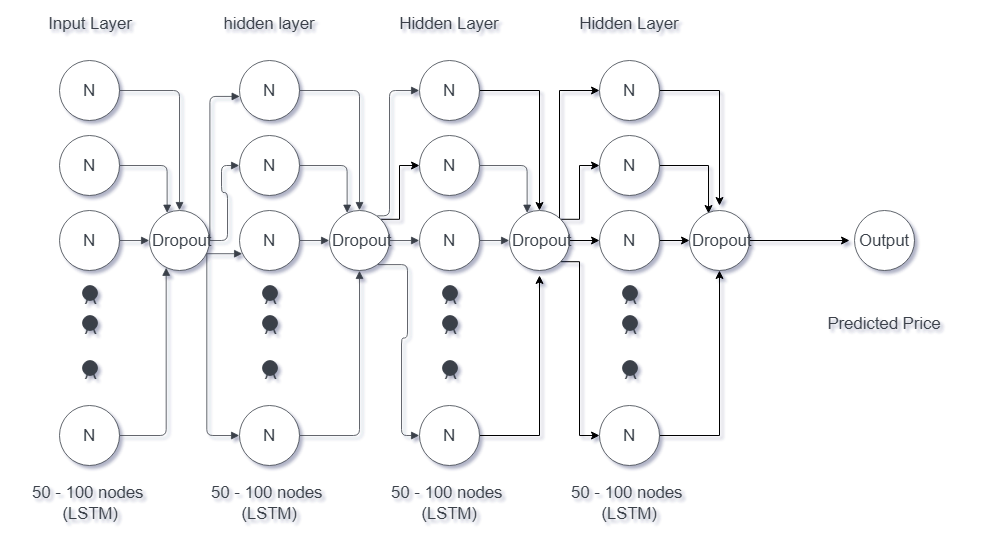
Sets

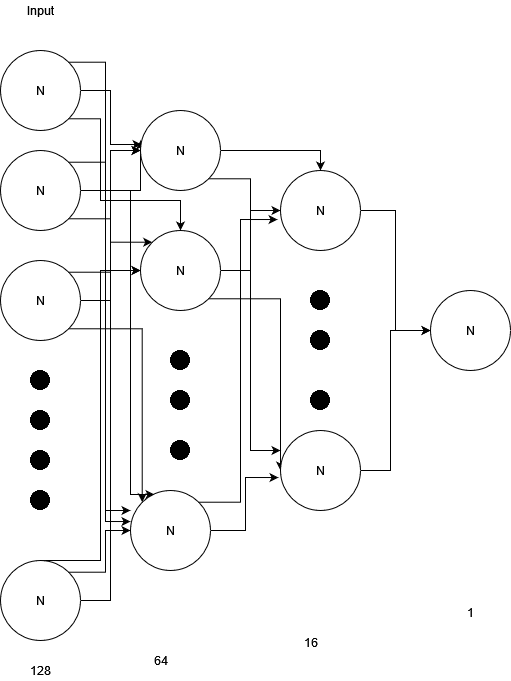
For training and testing sets, the values represented in brackets are [batch\_size, time\_steps, input\_dim].

* **Training se**t = First 32064 values of data [1002, 32, 6].
* **Test set** = Last 2432 values of data [76,32,6].
* **Test/Test labels** = Close values of respective timestep. 1-D array

3. Modeling and Error

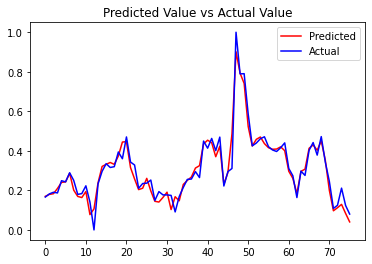
With an overall approach defined, we decided to experiment with three different models. Our first **(1.0)** model was designed using one LSTM input layer, three LSTM hidden layers, and an output layer. Modifying the model's layout slightly **(1.1)**, we also incorporated dropouts. Dropout prevents overfitting and allows the combination of multiple neural networks together. It works by temporarily removing a unit and its connections outside the network. This choice is made randomly and is determined by probability (p). In our model, we chose a dropout rate of 0.2 or 20%. (10-11).

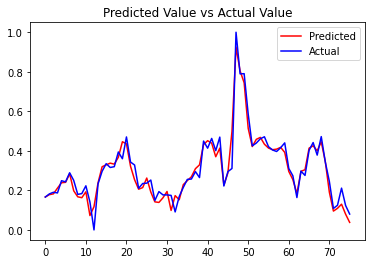


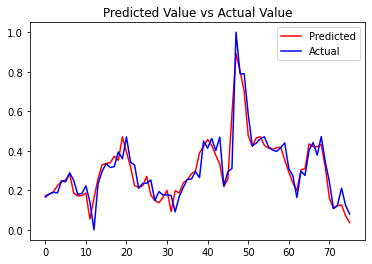
Our final model **(2.0)** was not formatted as similarly between the first two. Consisting sequential input layer followed by two LSTM layers two dense layers. The first dense layer consists of a ReLU activation, and the second dense layer consists of a linear activation function. (1). All our experiments were conducted with the same parameters for accuracy. Using optimizer Adam and accuracy metric Mean Square Error for compiling. (8)

4.Experiment results and Conclusion

The test accuracy is the mean value of the difference between actual and predicted values.

Model 1.0  
  
  
1 LSTM input layer (50 units)  
3 LSTM hidden layer (50 units each)  
1 Dense activation layer  
  
  
test loss: 0.00170604488812387  
test accuracy: 0.02631578966975212

Model 1.1  
  
  
1 LSTM input layer (50 units) \*  
3 LSTM hidden layer (50 units each) \*  
1 Dense activation layer  
\*Dropout p = 0.2  
  
test loss: 0.0018714434700086713  
test accuracy: 0.02631578966975212

Model 2.0  
  
  
1 sequential input layer  
2 LSTM layers (128 units and 64 units)  
1 dense layer (16, activation of ReLU)  
1 dense layer (1, activation linear)  
  
test loss: 0.003886935766786337  
test accuracy: 0.02631578966975212

While the test accuracy for all three was the same, our Model 1.0 had the best loss function. This result comes as a shock since it was expected that the dropout rate would have helped. Regardless, the loss function did not seem to affect the outcome of the neural network much, as all three models produced the same accuracy.

References

1. Murtaza Roondiwala, Harshal Patel, Shraddha Varma. Predicting Stock Prices Using LSTM. [https://www.*researchgate*.net/publication/327967988\_Predicting\_Stock\_Prices\_Using\_LSTM](https://www.researchgate.net/publication/327967988_Predicting_Stock_Prices_Using_LSTM)
2. Ken Jon M. Tamate, Dr. Madahvi Dervaj, Joel C. DE Goma. Overcoming The Vanishing Gradient Problem Of Recurrent Neural networks in the Iso 9001 Quality Management Audit Reports Classification. <http://www.ijstr.org/final-print/mar2020/Overcoming-The-Vanishing-Gradient-Problem-Of-Recurrent-Neural-Networks-In-The-Iso-9001-Quality-Management-Audit-Reports-Classification.pdf>
3. Olah, Christopher. Understanding LSTM Networks; 2017.  
    <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
4. IBM Cloud Education. Recurrent Neural Networks; 2020 <https://www.ibm.com/cloud/learn/recurrent-neural-networks>
5. Afshine Amidi, Shervine Amidi. Recurrent Neural Networks cheatsheet; <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>
6. Jitendra Kumar, Rimsha Groomer, Ashutosh Kumar Singh. Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model For Cloud Datacenters; 2017.   
   <https://www.sciencedirect.com/science/article/pii/S1877050917328557>
7. Kaggle Kerneler, Ethereum Historical Dataset; 2020. <https://www.kaggle.com/datasets/prasoonkottarathil/ethereum-historical-dataset>
8. Katherine (Yi) Li. Predicting Sock Prices using Machine Learning <https://neptune.ai/blog/predicting-stock-prices-using-machine-learning>
9. Mehar Vijh, Deeksha Chandola, Vinay Anand Tikkiwal, Arun Kumar. Stock Closing Price Prediction using Machine Learning Techniques <https://www.sciencedirect.com/science/article/pii/S1877050920307924>
10. IDK how to reference this on   
    <https://colab.research.google.com/drive/18WiSw1K0BW3jOKO56vxn11Fo9IyOuRjh#scrollTo=_qTK4Ajv3qRz>
11. Nitish Srivastav, Geoffrey Hinton, Alex Krizhevsky, Ilya Salakhutdinov, Ruslan Salakhutdinov. Dropout: A simple way to prevent Neural Networks from Overfitting; 2014  
    <https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>