Patricia (PJ) Watson

NetID: 2887247

**Assignment:** Add at least one bullet for each layer and describe what is going on (focus on trainable parameters and output shape and use the concept of 'automated feature engineering'). You may relate your bullets to the animation (**red dots** and **green dots)** and how the data get processed in different gates (input gate, forget gate, and output gate).

**Given:**

n\_steps: 3 (can be anything, 3 for our example)

n\_features: 8 represented by **green dots**

n\_hidden: 30 represented by **red dots**

**Problem 1**

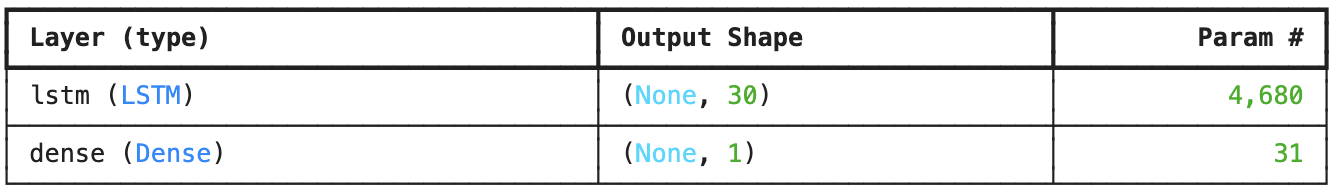
From Week 1. Simple one recurrent layer.

model = Sequential()

model.add(LSTM(n\_hidden, activation='relu', input\_shape=(n\_steps, n\_features)))

model.add(Dense(1, activation='sigmoid'))

model.summary()



Long Short-Term Memory (LTSM) model below is the example from class notebook for visual and future reference. In our Problem 1, we have **8 green dot input features** and **30 red dot hidden units**.

A diagram of a multiplying process

Description automatically generated

The LSTM unit is made up of four feedforward neural networks (FFNNs). Each of these neural networks consists of an input layer and an output layer. In each of these neural networks, input neurons are connected to all output neurons. As a result, the LSTM unit has four fully connected layers. How it works:

1. Input: The LSTM layer receives a sequence of data as input. Each element in the sequence is a feature vector representing the data at a particular timestep. The input shape is defined as (n\_steps, n\_features), where n\_steps is the length of the input sequence (3; it can be anything) and n\_features is the number of features in each timestep (8).
2. Cell Hidden States: The LSTM layer maintains two internal states: the cell state and the hidden state.
   * The cell state acts as a "memory" that carries information across timesteps.
   * The hidden state represents the output of the LSTM layer at a particular timestep. It's a function of the current input and the previous hidden state.
3. Gates: The core of the LSTM layer lies in three gates: the input gate, forget gate, and output gate. These gates control the flow of information into, out of, and within the cell state.
   * Input Gate: Determines which parts of the current input are important and should be stored in the cell state.
   * Forget Gate: Decides which parts of the existing cell state should be forgotten or discarded.
   * Output Gate: Controls which parts of the cell state should be output as the hidden state.
4. Automated Feature Engineering: The LSTM layer performs a form of automated feature engineering by learning to extract relevant features from the input sequence. This happens through the interactions of the gates and the internal states, allowing the LSTM to identify patterns and dependencies across timesteps.
5. Activation Function: The LSTM layer uses an activation function, such as Relu (Rectified Linear Unit), to introduce non-linearity and enable the network to learn complex patterns.
6. Output: The output of the LSTM layer is the hidden state, which is a sequence of vectors representing the processed information at each timestep. This output is then passed to the next layer, the Dense layer.

The LSTM layer processes the input sequence step-by-step, using its gates and internal states to selectively store and retrieve information. It effectively learns to represent the temporal patterns in the data, which can be beneficial for tasks like time series prediction or sequence classification.

**Layer 1** **lstm\_1 (LSTM) : trainable parameters**

This is Layer 1:

model.add(LSTM(n\_hidden, activation='relu', input\_shape=(n\_steps, n\_features)))

Generic RNN layer is g \* [h(h + i) + h] where :

g = 4 because an LSTM has 4 FFNNs

h = n\_hidden = 30 hidden units within LSTM, which would be **30 red dots**

i = n\_features = 8 the number of features, which would be **8 green dots**

Trainable parameters = 4 \* [30 (30 + 8 ) + 30 bias)] = 4 \* 1170 = 4680

The LSTM layer Output Shape is (none, 30) where “none” refers to the batch size, which is flexible and can vary. It means the network can process any number of input sequences at once. “30” is the number of hidden units (dimensions) in the LSTM layer's output. This value is determined by the n\_hidden parameter = 30.

**Layer 2** **dense\_1 (Dense): trainable parameters**

This is Layer 2:

model.add(Dense(1, activation='sigmoid'))

This dense layer takes the output of the LSTM layer as input and produces a single output value.

Generic dense layer is (i \* h + h \* o) + (h + o) where:

i = n\_features = 30 the number of features

h = n\_hidden = 1 which for this Dense Layer, we set to 1  
o =  0 the number of output features, which is also 0 in this case

Trainable parameters = (30 \* 1 + 0) + (1 + 0) = 30 + 1 bias = 31

The Dense layer Output Shape is (none, 1) where “none” again refers to the batch size, which can vary. It means the network can handle any number of input samples at once during both training and inference. The None is a placeholder indicating that the batch size can vary depending on the data fed into the model. “1” means the dense\_1 layer outputs a single value for each input sample, a scalar output. Using a dense layer with one unit (model.add(Dense(1, activation='sigmoid'))) means the output is compressed down to a single value. The sigmoid function is a mathematical function that maps any input value to a value between 0 and 1, used in classification problems.

Total trainable parameters = 4680 + 31 = 4711 foots to model summary

**Problem 2**

From Week 2. Stacked recurrent layers (note the use of return\_sequences).

model = Sequential()

model.add(LSTM(n\_hidden\*2, activation='relu', input\_shape=(n\_steps, n\_features),

return\_sequences=True))

model.add(SimpleRNN(n\_hidden, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

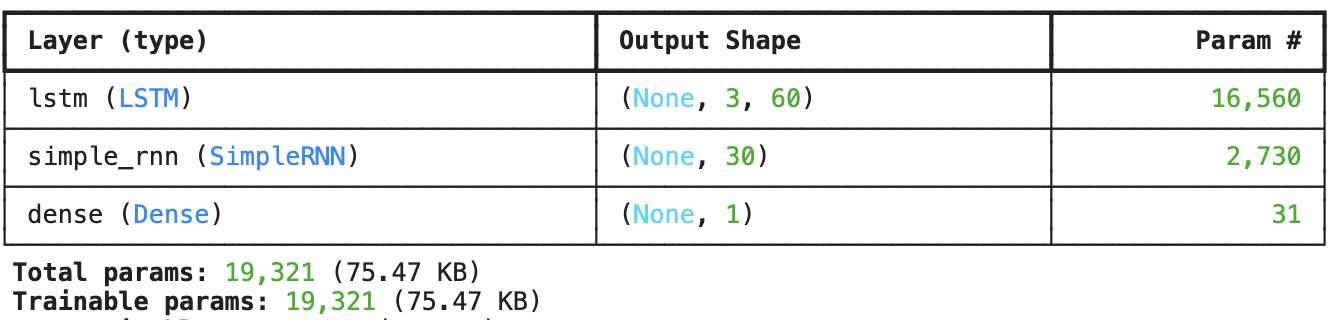
model.summary()

Refresh:

n\_steps: 3 (can be anything, 3 for our example)

n\_features: 8 represented by **green dots**

n\_hidden: 30 represented by **red dots**



Again, the LSTM unit is made up of four feedforward neural networks (FFNNs). See discussion above for how LSTM works.

**Layer 1** **lstm (LSTM) : trainable parameters**

This is Layer 1:

model.add(LSTM(n\_hidden\*2, activation='relu', input\_shape=(n\_steps, n\_features), return\_sequences=True))

Generic RNN layer formula is g \* [h(h + i) + h] where :

g = 4 because an LSTM has 4 FFNNs

h = n\_hidden \* 2 = 30 \* 2 = 60 hidden units within LSTM, which would be **60 red dots**

i = n\_features = 8 the number of features, which would be **8 green dots**

Trainable parameters = 4 \* [60(60 + 8 ) + 60 bias)] = 4 \* 4140 = 16,560

The LSTM layer Output Shape is now (none, 3, 30) . When return\_sequences=False (default, as in Problem 1), a recurrent layer only returns the final hidden state of the sequence. This hidden state represents the learned representation of the entire input sequence. The output shape in this case would be (None, n\_hidden), where n\_hidden is the number of hidden units in the recurrent layer. But when return\_sequences=True (as in Problem 2), the recurrent layer returns the entire sequence of hidden states, one for each timestep in the input sequence. The output shape becomes (None, n\_steps, n\_hidden), where n\_steps is the length of the input sequence (3) and n\_hidden is the number of hidden units (60).

**Why do this?** Setting return\_sequences=True is crucial when stacking multiple recurrent layers. If we want the next recurrent layer to receive the entire sequence of hidden states from the previous layer, we need to et return\_sequences=True in the preceding layer so the subsequent layer learns temporal dependencies across the entire sequence.

**Layer 2 simple\_rnn (SimpleRNN): trainable parameters**

This is Layer2:

model.add(SimpleRNN(n\_hidden, activation='relu'))

Note that per DrD, we would expect LTSM or GRU to be a better model and use these in practice - we wouldn't use a SimpleRNN here.

Generic RNN layer formula is g \* [h (h + i) + h]

g = 1 because there is only 1 FFNN in a simpleRNN cell

h = n\_hidden = 30 hidden units within LSTM, which would be **30 red dots**

i = n\_features =60 because the SimpleRNN layer gets input from the LSTM layer which has an output shape of (None, 3, 60),  **60 green dots**

Trainable parameters = 1 \* [30(30+ 60 ) + 30 bias)] = 1 \* 2730 = 2,730

The SimpleRNN Output Shape is (None, 30). Again, “None” represents the batch size, a dimension we can vary. It means the network can handle an arbitrary number of input samples at once. The None acts as a placeholder, indicating that the batch size can vary depending on the data fed into the model. “30”is the number of hidden units (AKA memory cells?) in the simple\_rnn layer. It's determined by the n\_hidden parameter (30.) The SimpleRNN layer outputs a single hidden state vector for each input sample in the batch. The dimensionality of this hidden state vector is equal to the number of hidden units (30).

**Layer 3 dense (Dense) : trainable parameters**

This is Layer 3:

model.add(Dense(1, activation='sigmoid'))

This dense layer takes the output of the SimpleRNN layer as input and produces a single output value.

Generic dense layer is (i \* h + h \* o) + (h + o) where:

i = n\_features = 30 the number of features

h = n\_hidden = 1 which for this Dense Layer, we set to 1  
o =  0 the number of output features, which is also 0 in this case

Trainable parameters = (30 \* 1 + 0) + (1 + 0) = 30 + 1 bias = 31

The Dense layer Output Shape is (none, 1) where “none” again refers to the batch size, which can vary. It means the network can handle any number of input samples at once during both training and inference. The None is a placeholder indicating that the batch size can vary depending on the data fed into the model. “1” means the dense\_1 layer outputs a single value for each input sample, a scalar output. Using a Dense layer with one unit (model.add(Dense(1, activation='sigmoid'))) means the output is compressed down to a single value. The sigmoid function is a mathematical function that maps any input value to a value between 0 and 1, used in classification problems.

Total trainable parameters = 16560 + 2730 + 31 = 19,321 foots to model summary