

With neural networks, information goes from one layer to the next in the following

order: input to hidden to output. The connections help to pass the information.

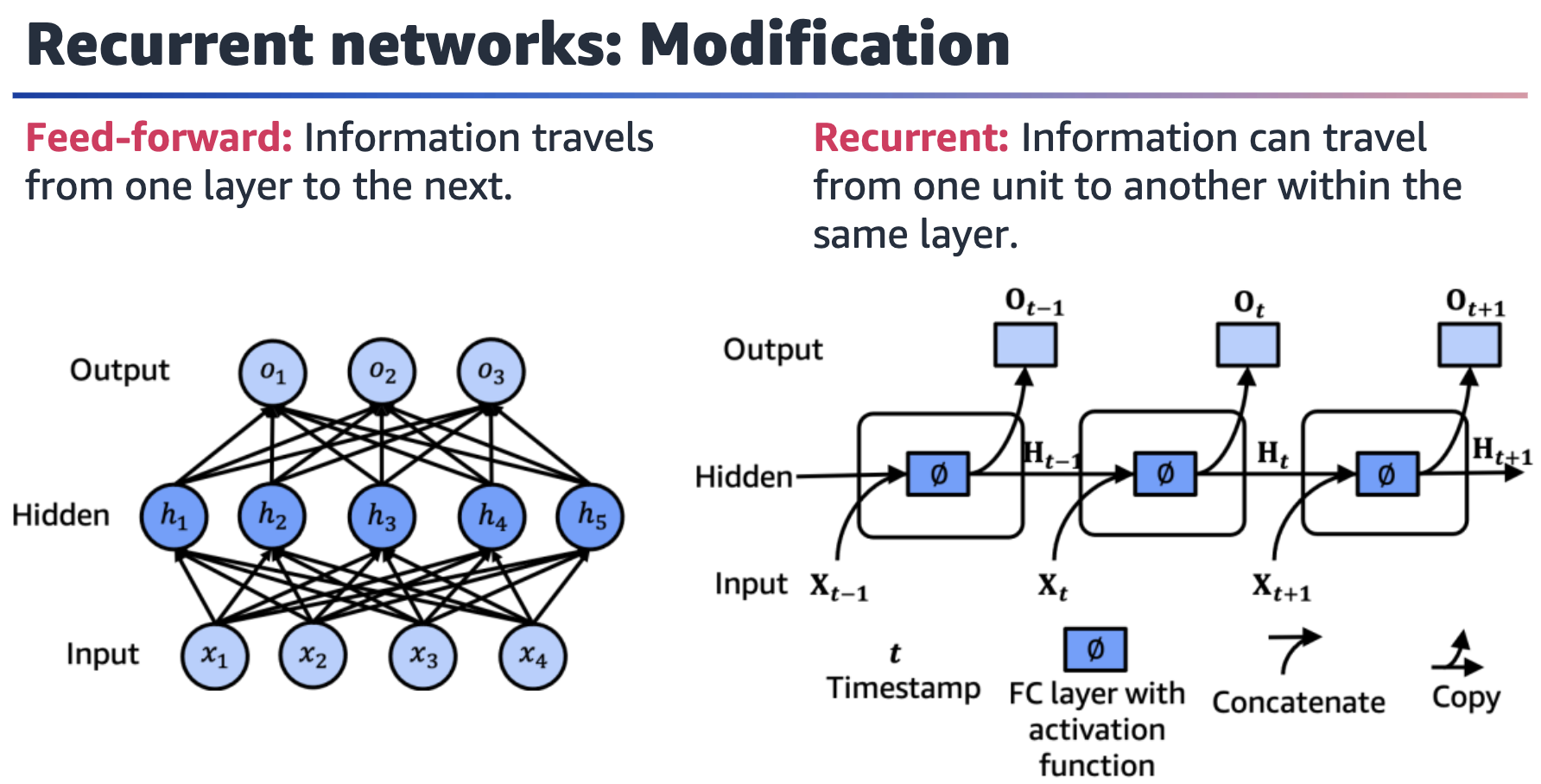
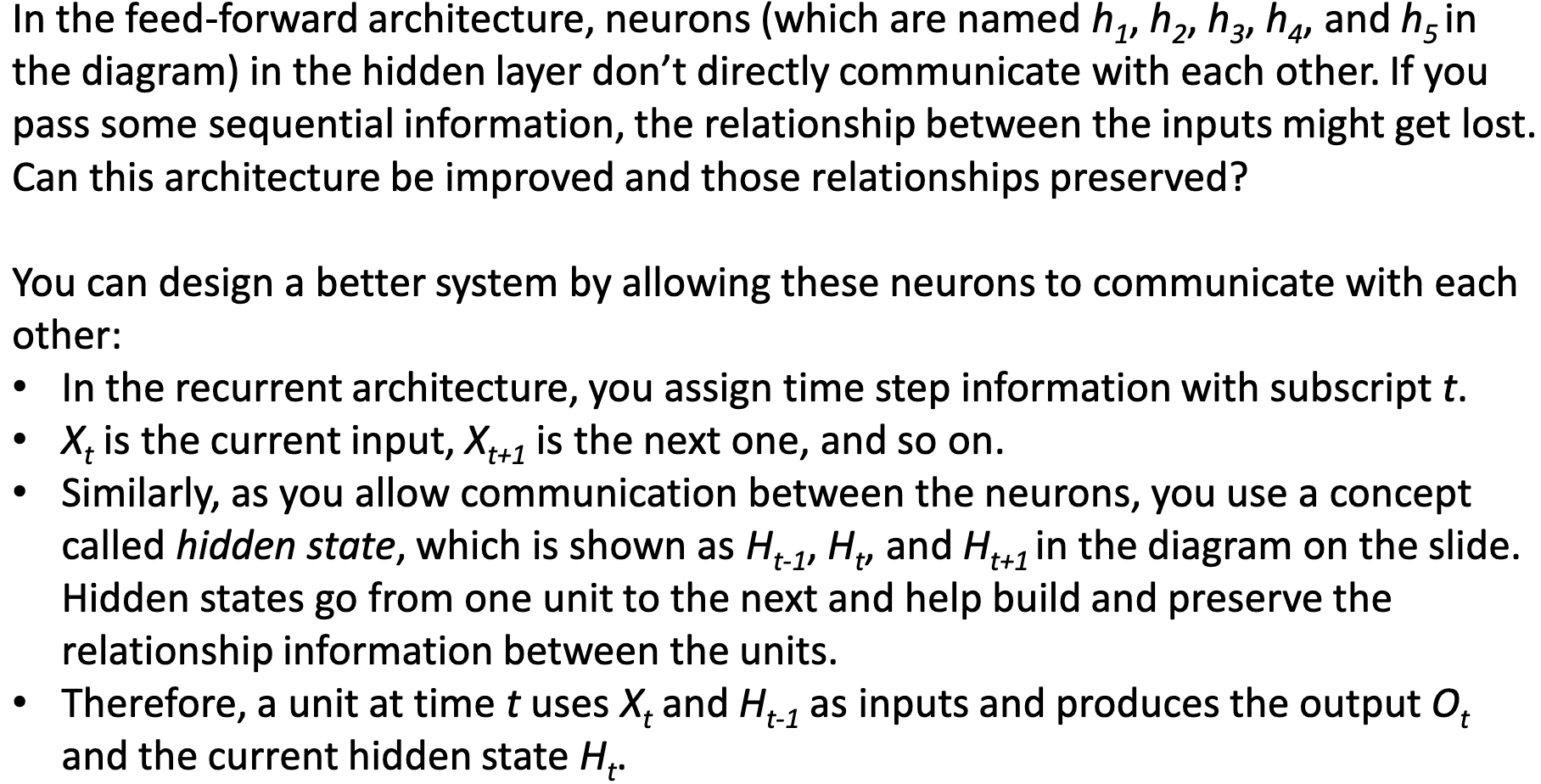
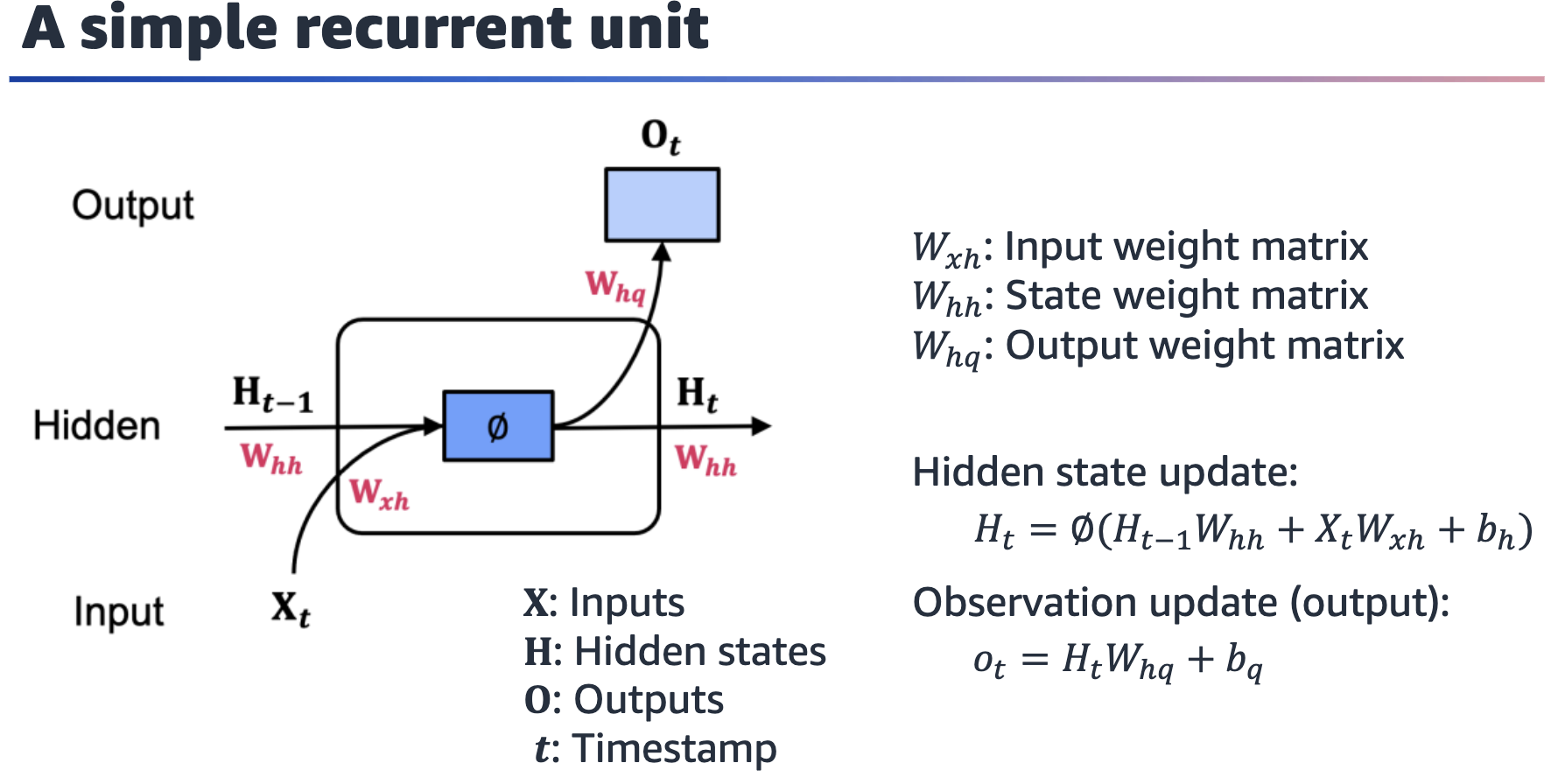


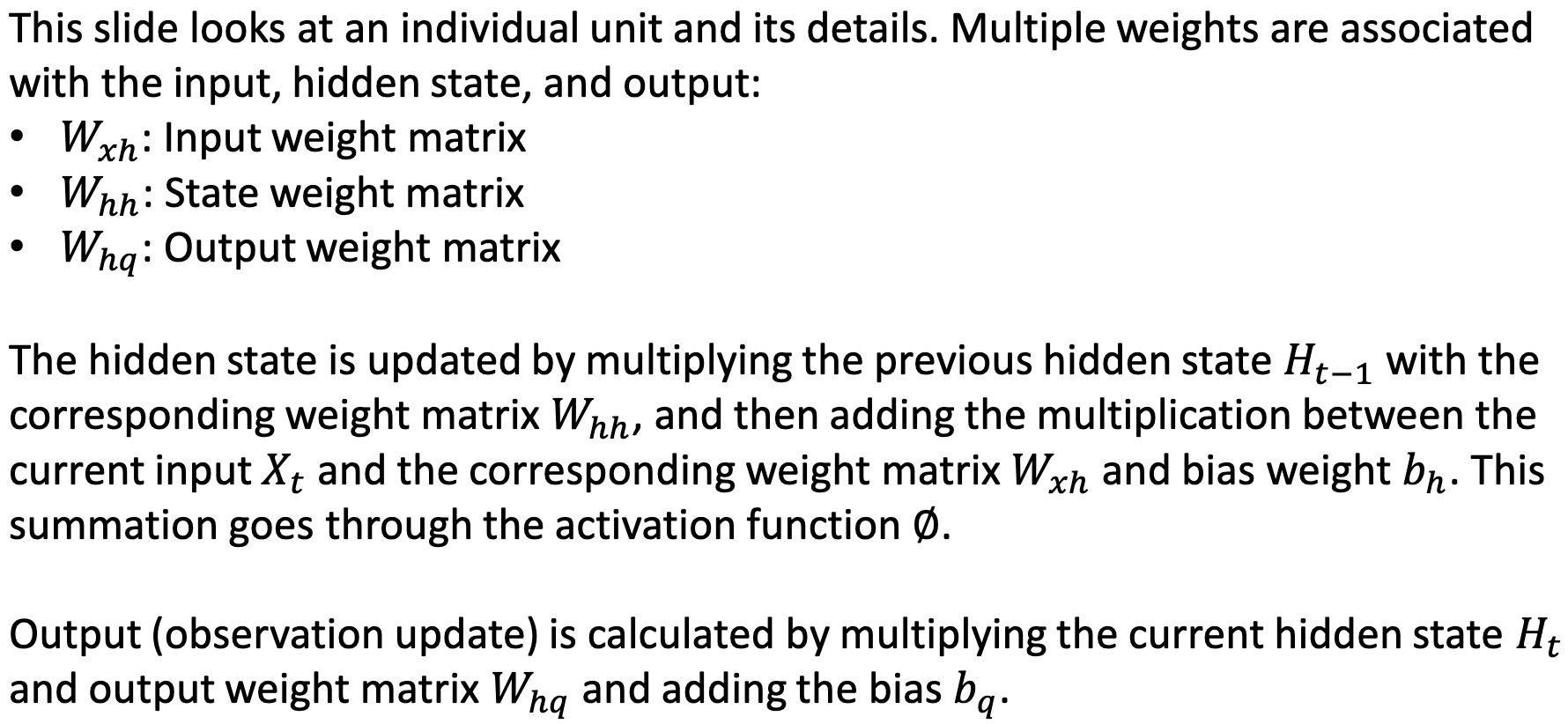
Figure of a multilayer neural network with three layers: input, hidden, and output. The input layer has four inputs, the hidden layer has five units, and the output layer has three units.

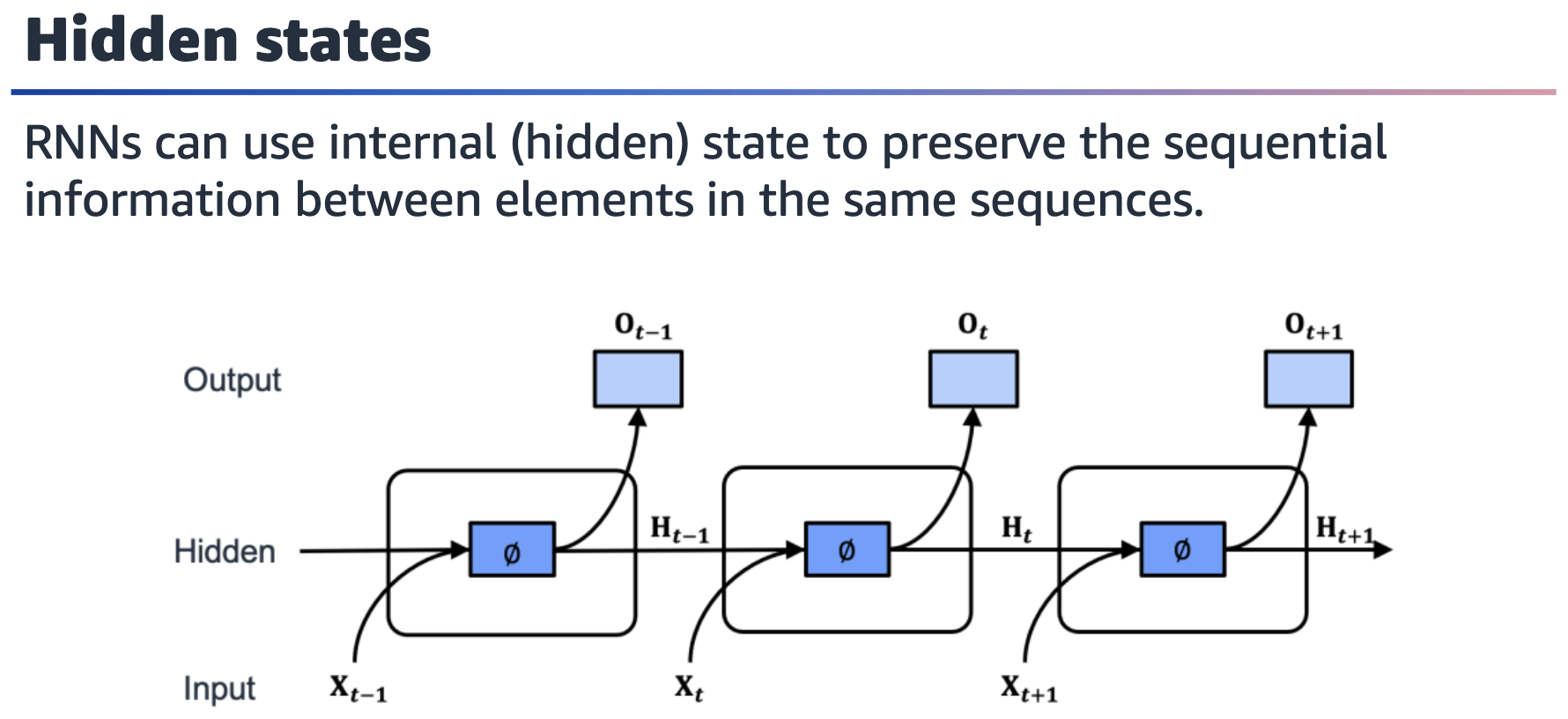


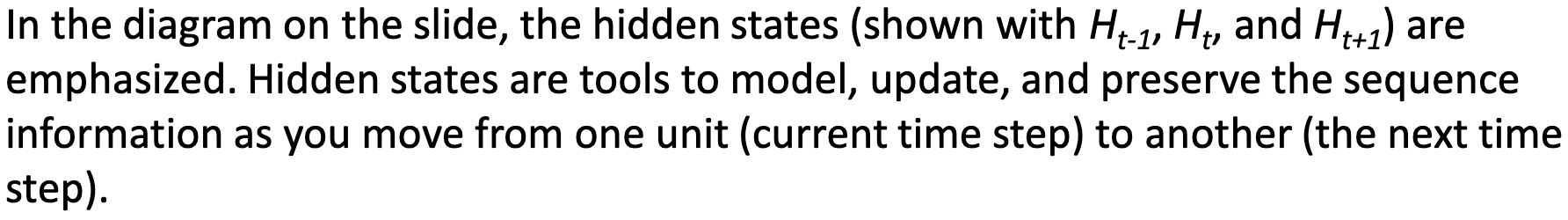
* Time step *t* extends until the end of the time horizon that you use (in other words, the number of inputs).

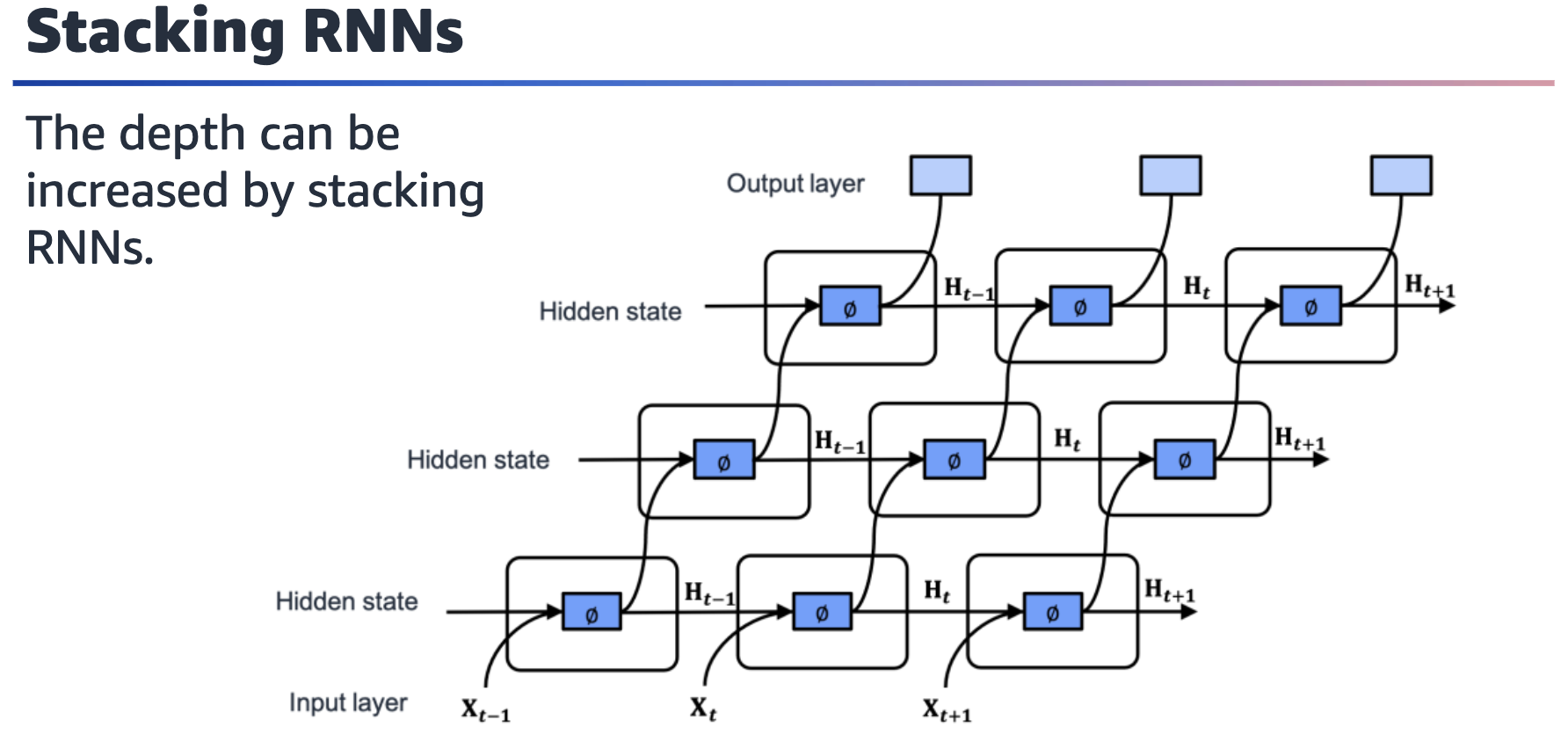


Single RNN unit with input, hidden, and output layers. Multiple weights are associated with the input (***Wxh***), hidden state (***Whh***), and output (***Whq***).









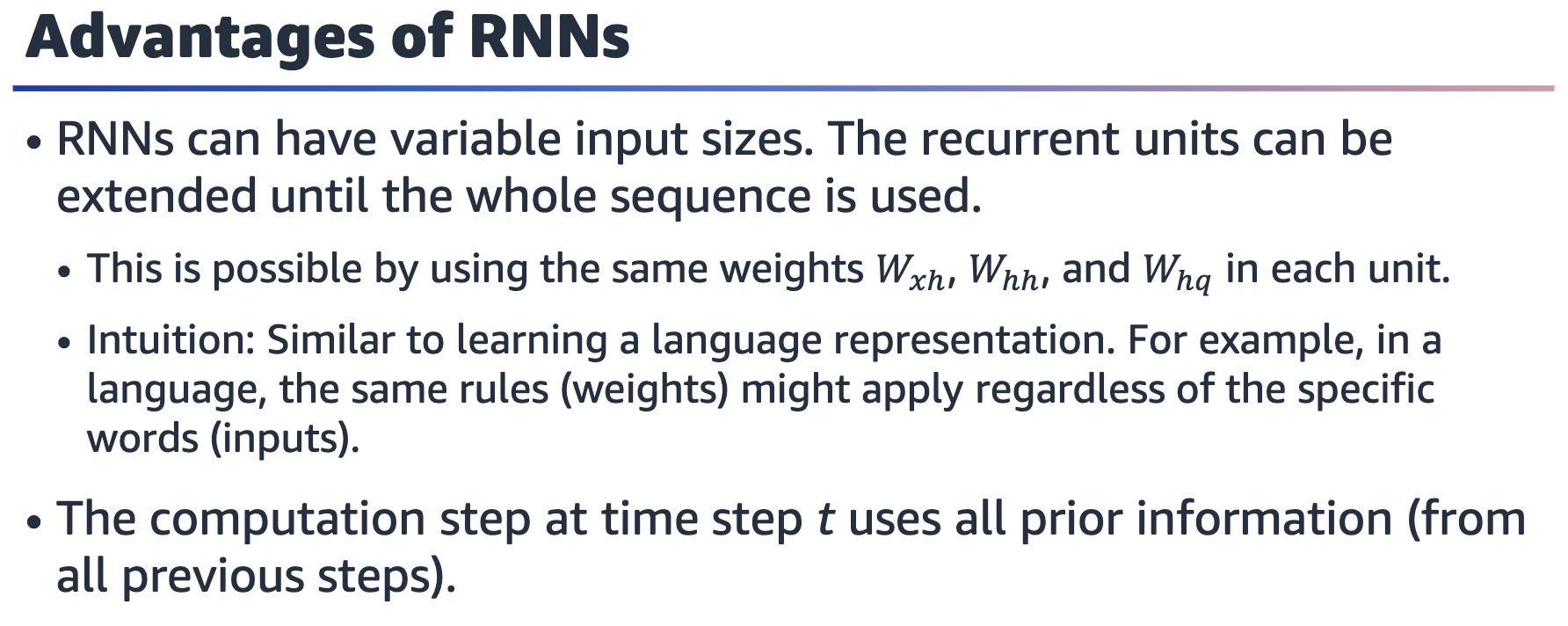
Stack of three RNNs. Outputs from one layer are the inputs of the next layer. At the end, the output of the whole system comes from the output of the last layer.

You can build multilayer neural networks by stacking multiple layers. You can also do

this with RNNs. You simply connect the outputs from one layer to the inputs of the

next layer. At the end, the output of the system comes from the output of the last

layer.





RNNs are inherently slow. All time steps in the data need to be processed. During the

training and backpropagation, similarly, you would need to consider all time steps. During the training, the backpropagation process is applied as *backpropagation through time (BPTT)*, which uses all time steps of the input. This is an additional burden in terms of the time commitment for training.

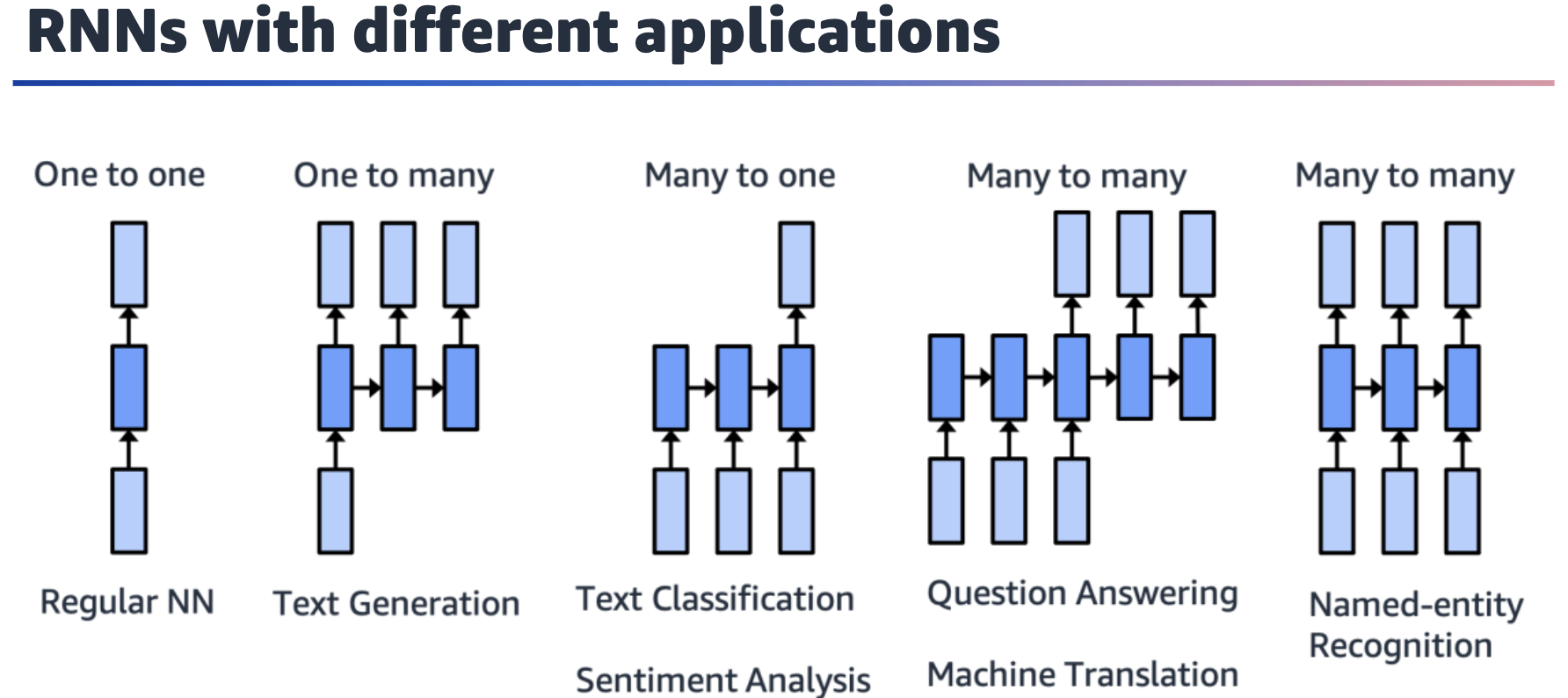
Vanishing (or exploding) gradients: During the backpropagation process, the math formula that gives the gradients involves many multiplicative terms because it has multiple steps. Due to the nature of the multiplication, if the multiplication operation has many small values, the whole value gets very small, which leads to vanishing gradients. Additionally, large values get even larger when multiplied, which leads to exploding gradients.

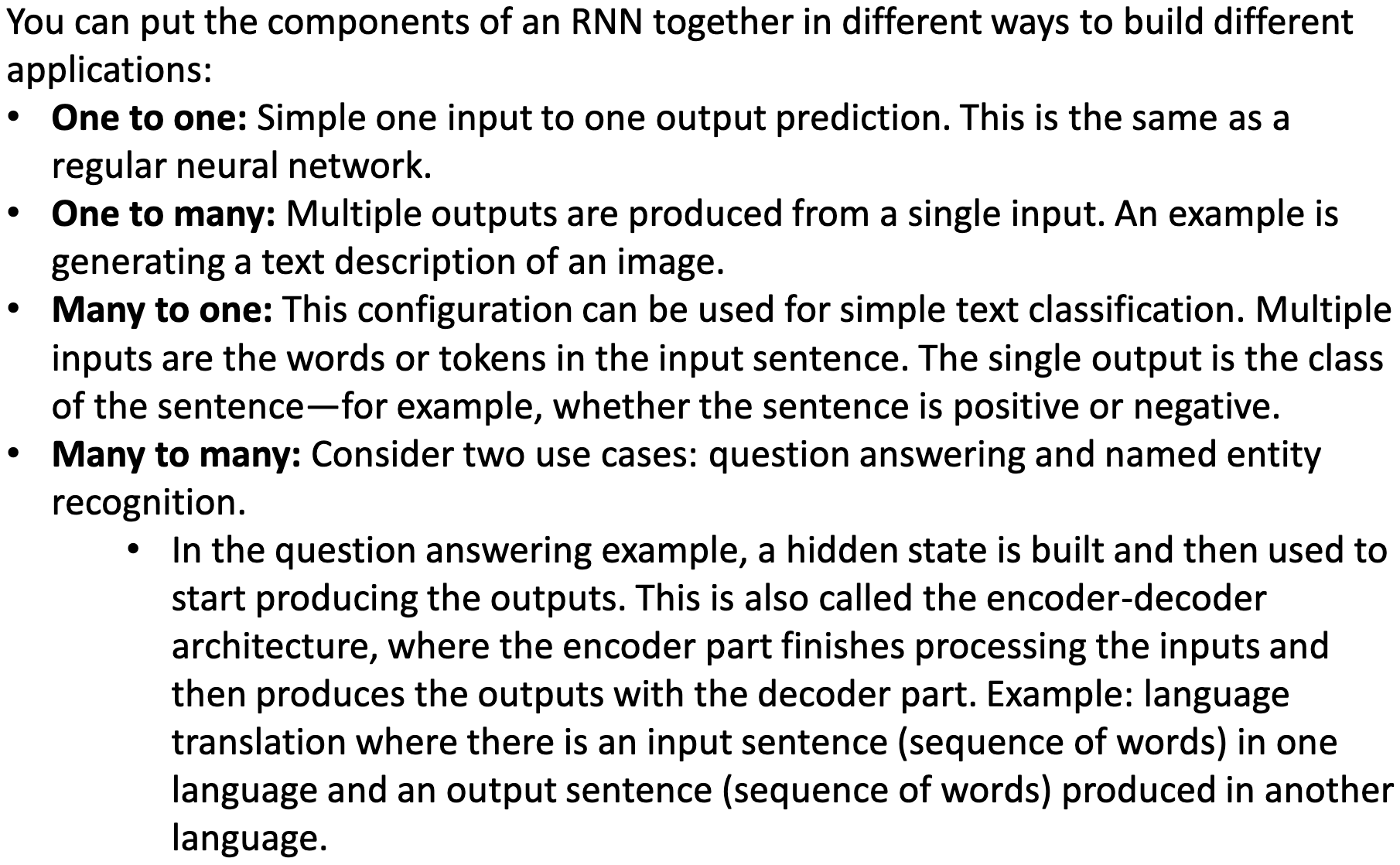
Long-term dependency problem: As the sequence length increases, it becomes more

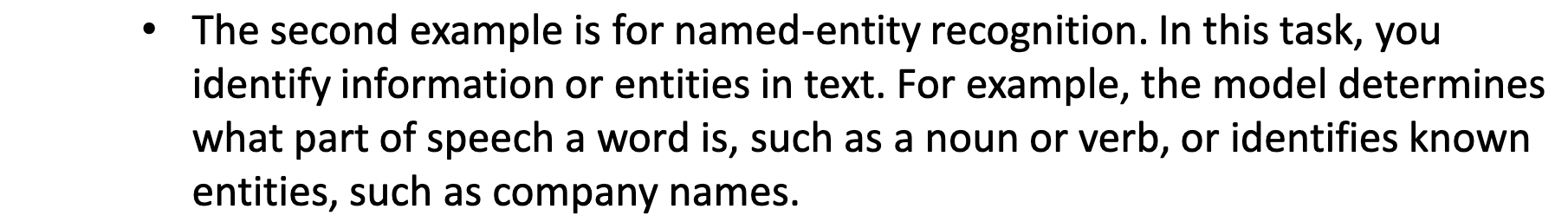
difficult to keep track of the overall relationship between the time steps and the early

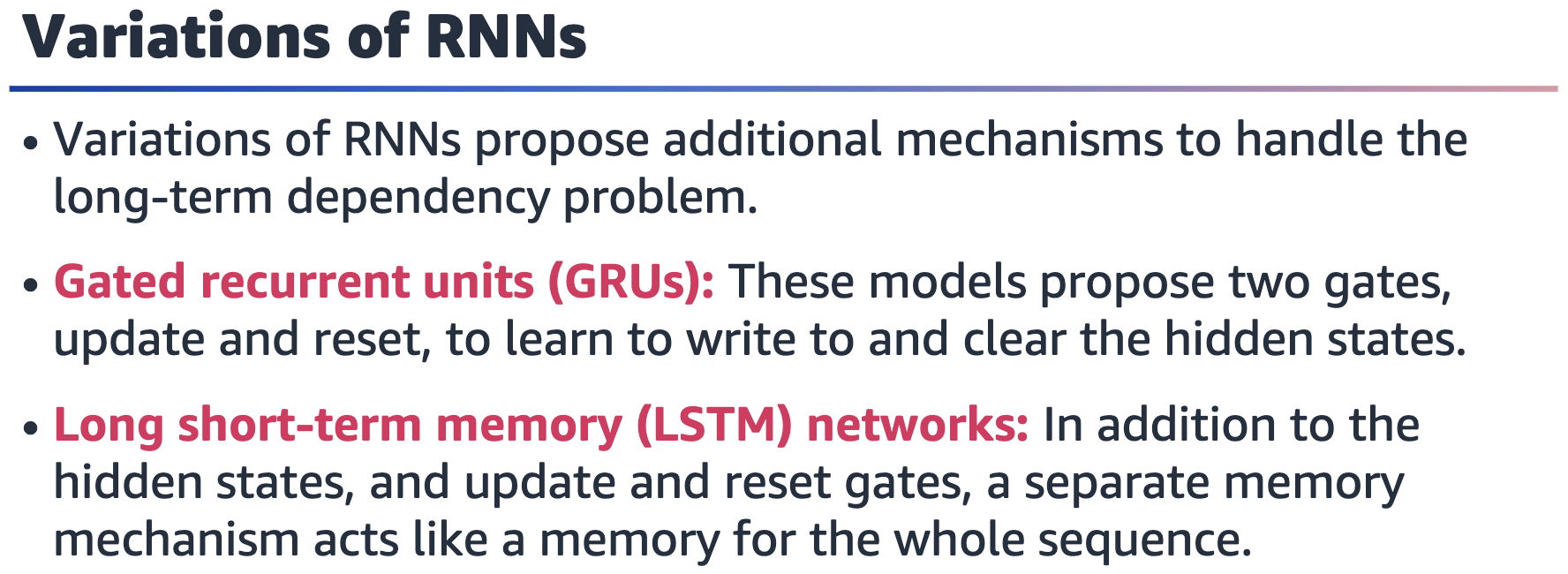
information. The information that passed using the hidden state can get corrupted.

This is called the long-term dependency problem. RNNs work better with short sequences.









Variations of RNNs include gated recurrent units (GRUs) and long short-term memory (LSTM) networks:

* GRU: The model uses two gates, update and reset, to manage the hidden states. For more information, see Gated Recurrent Units (GRU) in *Dive into Deep Learning* at <https://d2l.ai/chapter_recurrent-modern/gru.html>
* LSTM network: This is a more advanced GRU model. Additionally, a memory concept acts like a memory of the sequence. For more information, see Long Short-Term Memory (LSTM) in *Dive into Deep Learning* at <https://d2l.ai/chapter_recurrent-modern/lstm.html>

.

