You are already familiar with how the weights get multiplied in ANN model. Let's understand how the calculation happens within an RNN model in this reading.

1. **Input Text Vectorization**: Suppose each word in your text is represented by a d-dimensional vector (for example, using word embeddings like Word2Vec or GloVe). If d = 100, each word is represented as a 100-dimensional vector. So, the shape of the input vector x_t for each word at time step t is 100 X 1.
2. **Hidden State**: The hidden state in the RNN has 64 nodes. Therefore, the shape of the hidden state vector h_t at any time step t is 64 X 1.
3. **Weight Matrices**:
   - Input to Hidden Weight Matrix (W_x): This matrix maps the input vector to the hidden layer. Since the input vector has a dimension of 100 and the hidden layer has 64 nodes, the shape of W_x will be 64 X 100.
   - Hidden to Hidden Weight Matrix (W_h): This matrix is used to map the previous hidden state to the current hidden state. Since both the previous and current hidden states have 64 units, the shape of W_h will be 64 X 64.
   - Hidden to Output Weight Matrix (W_y) (if predicting the next word): If the RNN is used to predict the next word, and the vocabulary size is **p**, then W_y maps the hidden state to the output space. The shape of W_y would be pX64.
4. **Bias Vectors**:
   - Hidden Layer Bias (b_h): This is added to the hidden layer before applying the activation function. Its shape is 64 X1.
   - Output Layer Bias (b_y) (if predicting the next word): This is added to the output layer, with a shape of p X 1, where p is the vocabulary size.

So, when processing each word, the RNN takes the input vector (shaped (100 X 1)), transforms it with W_x (shaped (64 X 100)), updates its hidden state (shaped (64 X 1)) using W_h(shaped (64 X 64)), and optionally produces an output (for example, the next word prediction) using W_y (shaped (p X 64)).