COMS 4030A

Adaptive Computation and Machine Learning

7. Recurrent Neural Networks

Recurrent neural networks are used for datasets containing sequences of some sort.
Examples include video data, sound data, sensor data and text sentences and paragraphs.
The sequences may be of variable length, so standard neural network architecture cannot be used directly for such datasets since such networks require all inputs to be of the same size.

In recurrent neural networks, the terms in a given input sequence are input one at a time into the network and, using recurrent connections, a memory of past inputs is stored. 'Recurrent connections' means edges that connect from a layer of nodes back to itself, or to a previous layer. The contribution from these edges is only realised in the subsequent feedforward step when the next term in the sequence is processed.

A **recurrent neural network**, or **RNN** for short, is a neural network that contains one or more recurrent connections.

Regarding the output of a RNN, there are three main types, as follows.

(i) Given an input sequence, predict the class or value associated with the sequence.
In this case, output is only required after the whole sequence has been processed.
For example, given a video which is a sequence of images, predict some classification thereof.
Or, given a string of words, predict if they form a correct sentence.

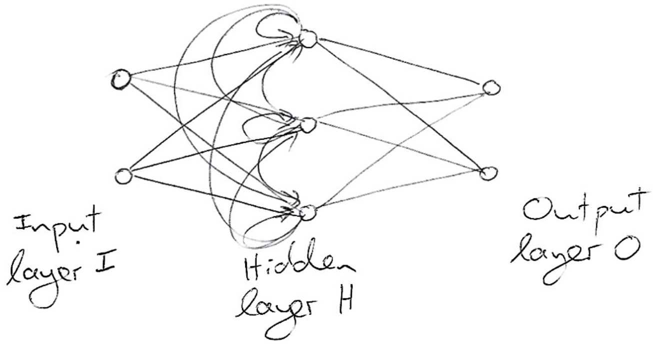
(ii) Given an input sequence, make a prediction for every term in the sequence.
In this case, output is required after every term of the sequence is processed.
For example, suppose that a sequence of sensor data is received by some system that requires actions (i.e., a prediction of what action to take) after every new set of information received.

(iii) Given an input sequence, predict the next term in the sequence.
In this case, output is required after every term in the sequence is processed. For example,

game AI's, may receive a sequence of actions or states and need to predict the next action or state at each time-step.

Note that, although the input sequence into a RNN can be of any length, the terms within the sequence must all be of the same length so that the neural network can take each term as input. For example, in the case of a video, all images making up the video must be of the same size. In the case of a sequence of actions, the representation of each action must be of some fixed length.

The figure below shows an example of a simple RNN; this kind of network is sometimes called an **Elman network**:



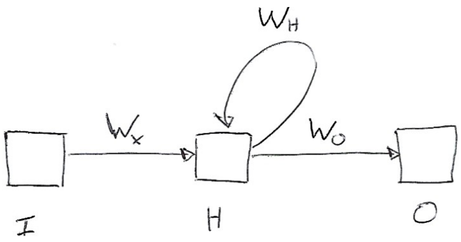
The RNN has three layers, an input layer I, a hidden layer H and an output layer O.
There are edges from layer I to layer H with associated weights; we use W_X to denote the matrix of weights from layer I to layer H. In this example, W_X has dimension 2×3.

There are also edges from layer H to layer O with associated weights; we use W_O to denote the matrix of weights from layer H to layer O. In this example, W_O has dimension 3×2.

What makes this network 'recurrent' are the edges from layer H back to itself. These edges all have weights and we denote the corresponding weight matrix by W_H; in this example W_H has dimension 3×3.

The layer H also has a vector of bias values $\boldsymbol{b}_H$ and an activation function. The output layer O also has a vector of bias values $\boldsymbol{b}_O$ and an activation function.

The following diagram is another way to represent the above RNN which shows the weight matrices between the layers:



Example: To see how an input to the above RNN would be processed, suppose that the weight matrices are as follows:

$$W_X = \begin{bmatrix} -1 & -3 & 0 \\ -2 & -1 & 2 \end{bmatrix}, \quad W_H = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 1 \\ -1 & 2 & 0 \end{bmatrix}, \quad W_O = \begin{bmatrix} 1 & -1 \\ 1 & 2 \\ -2 & 1 \end{bmatrix}$$

In addition, suppose that the vectors of bias values are as follows:

$$\boldsymbol{b}_H = (1, 0, -1), \quad \boldsymbol{b}_O = (-1, 1).$$

Lastly, suppose that the *relu* activation function is used at both the hidden layer H and the output layer O. The *tanh* activation function is often used in RNNs since this produces activation values that can be positive or negative. We use *relu* here for ease of computation.

Now, suppose that a sequence $\boldsymbol{x} = (\boldsymbol{x}(1), \boldsymbol{x}(2), \boldsymbol{x}(3))$ consisting of three terms is input into the RNN described above, where

$$\begin{aligned} \boldsymbol{x}(1) &= (-1, 1) \\ \boldsymbol{x}(2) &= (1, -2) \\ \boldsymbol{x}(3) &= (3, -4). \end{aligned}$$

Note that all of the above terms make up the input $\boldsymbol{x}$ and all terms are of the same size.

The output of the network is computed as follows.
First, $\boldsymbol{x}(1)$ is input into the network and the activation values at layer H are obtained:

$$\boldsymbol{x}(1)W_X = (-1, 1) \begin{bmatrix} -1 & -3 & 0 \\ -2 & -1 & 2 \end{bmatrix} = (-1, 2, 2).$$

Adding the bias vector gives:

$$(-1, 2, 2) + \boldsymbol{b}_H = (-1, 2, 2) + (1, 0, -1) = (0, 2, 1).$$

Lastly, applying the *relu* activation function gives:

$$relu(0,2,1) = (0,2,1).$$

The above values are the activation values at layer H at time-step 1, corresponding to input $\boldsymbol{x}(1)$, so it is denoted by $\boldsymbol{h}(1)$. Thus,

$$\boldsymbol{h}(1) = (0,2,1).$$

To get the output at time-step 1, which is denoted by $\boldsymbol{y}(1)$, the next layer is computed:

$$\boldsymbol{h}(1)W_O = (0,2,1)\begin{bmatrix} 1 & -1 \\ 1 & 2 \\ -2 & 1 \end{bmatrix} = (0,5).$$

Adding bias values gives

$$(0,5) + \boldsymbol{b}_O = (0,5) + (-1,1) = (-1,6)$$

and applying *relu* gives

$$relu(-1,6) = (0,6).$$

Thus,

$$\boldsymbol{y}(1) = (0,6).$$

The next step is to input $\boldsymbol{x}(2)$. In this step, the activation values at layer H are obtained from both the input values $\boldsymbol{x}(2)$ and the current activation values at layer H, i.e., $\boldsymbol{h}(1)$, by first computing

$$\boldsymbol{x}(2)W_X + \boldsymbol{h}(1)W_H,$$

which is

$$(1,-2)\begin{bmatrix} -1 & -3 & 0 \\ -2 & -1 & 2 \end{bmatrix} + (0,2,1)\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 1 \\ -1 & 2 & 0 \end{bmatrix} = (3,-1,-4) + (-1,4,2) = (2,3,-2).$$

Then the bias vector is added

$$(2,3,-2) + \boldsymbol{b}_H = (2,3,-2) + (1,0,-1) = (3,3,-3)$$

and *relu* is applied

$$relu(3,3,-3) = (3,3,0).$$

Thus, $\boldsymbol{h}(2) = (3,3,0)$. Next, the output $\boldsymbol{y}(2)$ is obtained by first computing

$$\boldsymbol{h}(2)W_O = (3,3,0)\begin{bmatrix} 1 & -1 \\ 1 & 2 \\ -2 & 1 \end{bmatrix} = (6,3).$$

Then the bias vector is added

$$(6,3) + \boldsymbol{b}_O = (6,3) + (-1,1) = (5,4)$$

and *relu* is applied

$$relu(5,4) = (5,4)$$

which gives the output $\boldsymbol{y}(2) = (5,4)$.

Lastly, using the third input value $\boldsymbol{x}(3)$, the computation of $\boldsymbol{h}(3)$ is as follows:

$$\begin{aligned}
\boldsymbol{h}(3) &= relu\left(\boldsymbol{x}(3)W_X + \boldsymbol{h}(2)W_H + \boldsymbol{b}_H\right) \\
&= relu\left((3,-4)\begin{bmatrix} -1 & -3 & 0 \\ -2 & -1 & 2 \end{bmatrix} + (3,3,0)\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 1 \\ -1 & 2 & 0 \end{bmatrix} + (1,0,-1)\right) \\
&= relu\left((5,-5,-8) + (3,0,3) + (1,0,-1)\right) \\
&= relu\left((9,-5,-6)\right) \\
&= (9,0,0).
\end{aligned}$$

The output $\boldsymbol{y}(3)$ is obtained as follows:

$$\begin{aligned}
\boldsymbol{y}(3) &= relu\left(\boldsymbol{h}(3)W_O + \boldsymbol{b}_O\right) \\
&= relu\left((9,0,0)\begin{bmatrix} 1 & -1 \\ 1 & 2 \\ -2 & 1 \end{bmatrix} + (-1,1)\right) \\
&= relu\left((9,-9) + (-1,1)\right) \\
&= relu(8,-8) = (8,0).
\end{aligned}$$

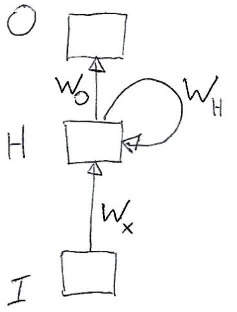
Thus, the sequence of outputs obtained for input sequence $\boldsymbol{x}$ is

$$\boldsymbol{y}(1) = (0,6), \quad \boldsymbol{y}(2) = (5,4), \quad \boldsymbol{y}(3) = (8,0).$$

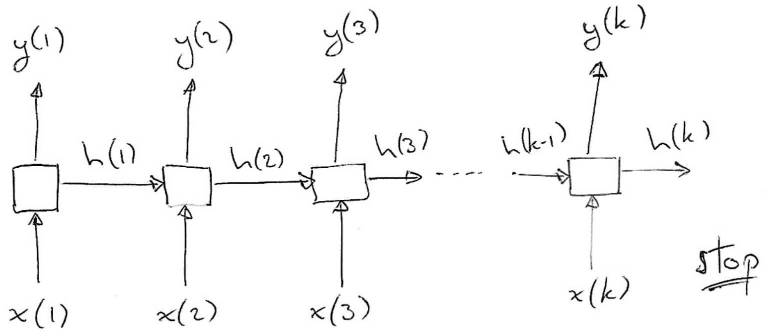
If only the last output is required, then the other outputs need not be computed, however, each of the $\boldsymbol{h}(i)$ values are needed.

7.1. Unravelling RNNs.

A way to visualise the generation of outputs and hidden values in a RNN is by **unravelling** the computation of the RNN. First, draw the RNN with vertical layers, as follows:



The next diagram shows the computation of the $\boldsymbol{h}(t)$'s and $\boldsymbol{y}(t)$'s over time-steps t from left to right:



The **formulas** for computing the outputs of a RNN with above structure, given an input sequence $\boldsymbol{x} = (\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots, \boldsymbol{x}(k))$, are as follows.

$$\boldsymbol{h}(t) = g_H\left(\boldsymbol{x}(t)W_X + \boldsymbol{h}(t-1)W_H + \boldsymbol{b}_H\right) \tag{1}$$

$$\boldsymbol{y}(t) = g_O\left(\boldsymbol{h}(t)W_O + \boldsymbol{b}_O\right) \tag{2}$$

where g_H is the activation function at layer H, $\boldsymbol{b}_H$ is the vector of bias values at layer H and W_X and W_H are the weight matrices between the layers I and H, and H and H, respectively. Also, g_O is the activation function at layer O, $\boldsymbol{b}_O$ is the vector of bias values at layer O and W_O is the weight matrix between layers H and O.

Note that the calculation of $\boldsymbol{h}(t)$ in (1) makes use of the previous $\boldsymbol{h}$ value, namely, $\boldsymbol{h}(t-1)$. In order to obtain $\boldsymbol{h}(1)$, a value for $\boldsymbol{h}(0)$ must be chosen, which is usually the zero vector, i.e.,

$\boldsymbol{h}(0) = (0, \ldots, 0)$. Thus, for time-step 1, that is $t = 1$, the value of $\boldsymbol{h}(1)$ obtained using (1) is

$$\boldsymbol{h}(1) = g_H(\boldsymbol{x}(1)W_X + \boldsymbol{b}_H)$$

For subsequent time-steps, the value of $\boldsymbol{h}(t-1)$ is used to obtain $\boldsymbol{h}(t)$ using the formula (1). At every time-step an output vector $\boldsymbol{y}(t)$ is also obtained using formula (2), which requires $\boldsymbol{h}(t)$.

The above computation, therefore, produces a sequence of outputs, which is

$$\boldsymbol{y}(1), \boldsymbol{y}(2), \boldsymbol{y}(3), \ldots, \boldsymbol{y}(k),$$

as well as a sequence of activation values at the hidden layer H, which is

$$\boldsymbol{h}(1), \boldsymbol{h}(2), \boldsymbol{h}(3), \ldots, \boldsymbol{h}(k).$$

The above sequence of $\boldsymbol{h}$ values may be considered the **memory sequence** since the values computed make use of all previous input values. Thus, at any time-step ℓ, the value $\boldsymbol{h}(\ell)$ includes a contribution from all the inputs $\boldsymbol{x}(1), \ldots, \boldsymbol{x}(\ell)$ and, hence, the output $\boldsymbol{y}(\ell)$ does as well.

In the formula (1), the computation of $\boldsymbol{x}(t)W_X + \boldsymbol{h}(t-1)W_H$ is sometimes written as

$$\boldsymbol{x}(t)\boldsymbol{h}(t-1)\begin{bmatrix} W_X \\ W_H \end{bmatrix}$$

where $\boldsymbol{x}(t)\boldsymbol{h}(t-1)$ is the **concatenation** of vectors $\boldsymbol{x}(t)$ and $\boldsymbol{h}(t-1)$, and $\begin{bmatrix} W_X \\ W_H \end{bmatrix}$ is the matrix obtained by joining W_X and W_H as shown.

To see that this works the same way, consider the step in the above example
where $\boldsymbol{x}(2)W_X + \boldsymbol{h}(1)W_H$, was computed:

$$(1, -2)\begin{bmatrix} -1 & -3 & 0 \\ -2 & -1 & 2 \end{bmatrix} + (0, 2, 1)\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 1 \\ -1 & 2 & 0 \end{bmatrix} = (3, -1, -4) + (-1, 4, 2) = (2, 3, -2).$$

Using the concatenation of vectors instead, this would be $\boldsymbol{x}(2)\boldsymbol{h}(1)\begin{bmatrix} W_X \\ W_H \end{bmatrix}$, i.e.,

$$(1,-2,0,2,1)\begin{bmatrix} -1 & -3 & 0 \\ -2 & -1 & 2 \\ 1 & -1 & 0 \\ 0 & 1 & 1 \\ -1 & 2 & 0 \end{bmatrix} = (2,3,-2).$$

7.2. **Training a RNN.**

Suppose we are given a dataset consisting of a set of sequences as inputs. Within each sequence the terms must be of a fixed size (e.g., images of a fixed size), but the length of the sequences may vary. As for the targets, it may be the case that each sequence has a single corresponding target (such as a classification), or that there is a sequence of targets $\boldsymbol{t}(1), \ldots, \boldsymbol{t}(k)$, corresponding to the k outputs that will be obtained. We wish use a RNN to model the dataset.

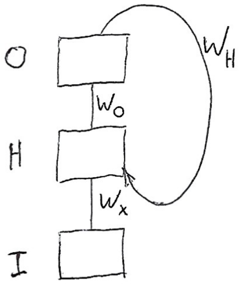
Once a RNN architecture has been selected for the dataset, the **training** of the RNN on the dataset proceeds as in standard neural networks, including dividing the data into training, validation and test sets, and selecting a suitable loss function. Next, an input sequence from the test dataset is fed into the RNN and outputs are obtained. Then the gradient descent method is applied using the backpropagation technique. The main difference in training RNNs is that the backpropagation updates must be applied at each time step used in the forward computation. This method is called **backpropagation through time**.

If only the final output $\boldsymbol{y}(k)$ has a target, then the loss between the final output and the given target is computed (using whichever loss function was chosen). This loss is propagated backwards through all the time-steps and the weights of the network are updated. Thus, weights in the network receive k updates if the input had length k.

If the output at every time-step has a target value, then for every time-step j, the loss can be computed between the ouput $\boldsymbol{y}(j)$ and the target for that time-step. During backpropagation, this loss is combined with the losses from subsequent time-steps that are being backpropagated.

7.3. Jordan Networks.

Another type of RNN is the **Jordan network**, which is similar to the Elman Networks considered above. In a Jordan network the output values $\boldsymbol{y}(t)$ are recursive fed back into the hidden layer, instead of the $\boldsymbol{h}(t)$ values.



The formulas for computation in a Jordan network are as follows.

For an input sequence $\boldsymbol{x} = (\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots, \boldsymbol{x}(k))$,

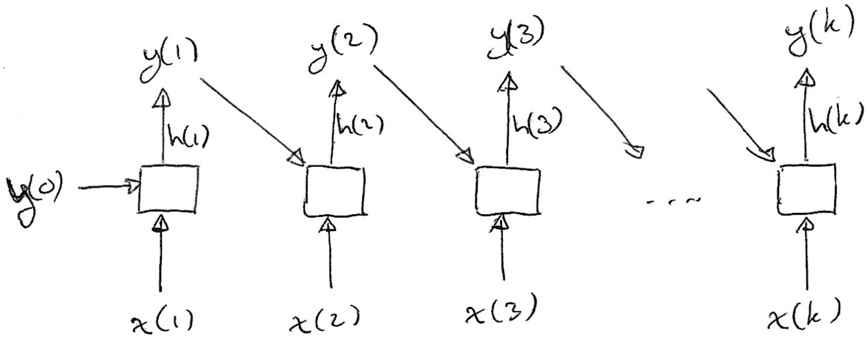
$$\boldsymbol{h}(t) = g_H\left(\boldsymbol{x}(t)W_X + \boldsymbol{y}(t-1)W_H + \boldsymbol{b}_H\right) \tag{3}$$

$$\boldsymbol{y}(t) = g_O\left(\boldsymbol{h}(t)W_O + \boldsymbol{b}_O\right) \tag{4}$$

where g_H is the activation function at layer H, $\boldsymbol{b}_H$ is the vector of bias values at layer H and W_X and W_H are the weight matrices between the layers I and H, and H and H, respectively. Also, g_O is the activation function at layer O, $\boldsymbol{b}_O$ is the vector of bias values at layer O and W_O is the weight matrix between layers H and O.

The only difference between (1) and (3) is that $\boldsymbol{h}(t-1)$ is replaced by $\boldsymbol{y}(t-1)$.
Also, for the initial step, the value of $\boldsymbol{y}(0)$ is usually set to $(0, \ldots, 0)$.

The unravelling of the Jordan network looks as follows:

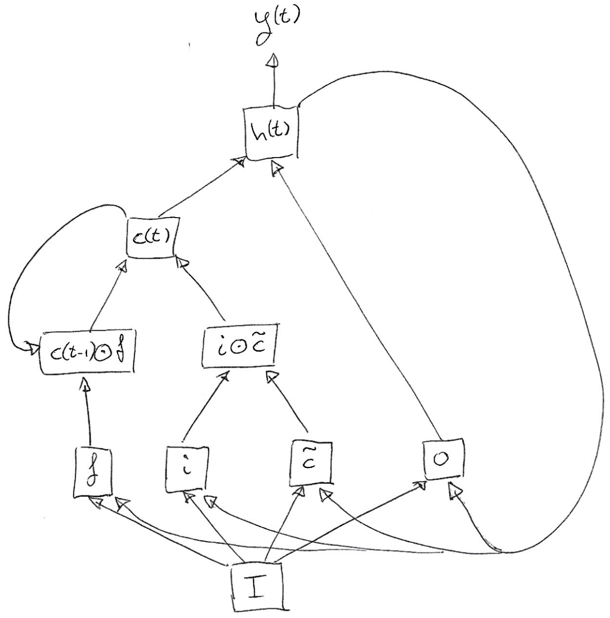


More complicated RNN architectures are used in practise, that may use more layers, or combinations of the above networks, or combinations of RNNs with CNNs, or other types of neural network. In this next section we consider one such RNN.

7.4. **LSTMs.**

A popular type of RNN is the **Long and Short-Term Memory**, or **LSTM** for short. LSTMs address the issue of 'memory loss' in an RNN, where the earliest terms of an input sequence are mostly forgotten and the contribution from the last few terms in a sequence dominates the output.

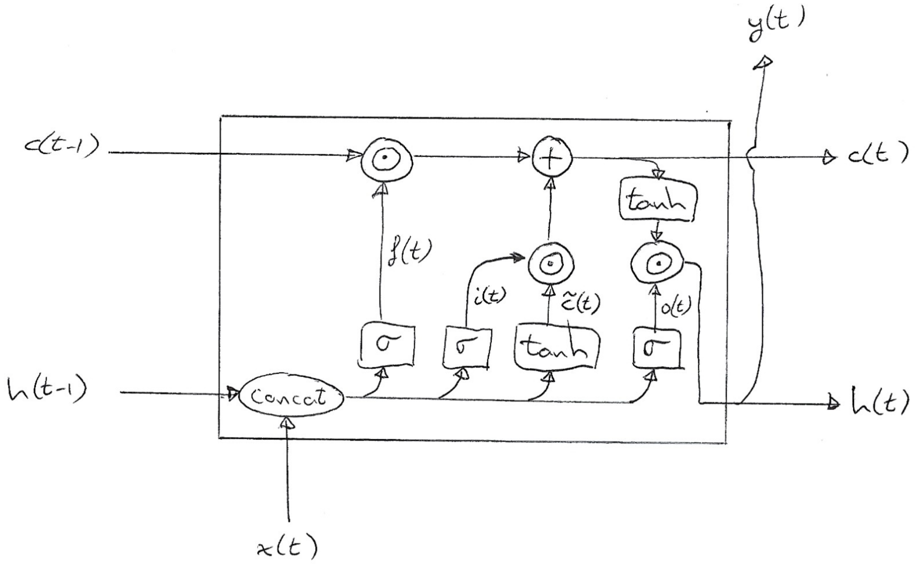
An LSTM uses a **hidden state** to maintain a **short-term memory** $h(t)$ of previous inputs, as well as a **cell state** $c(t)$ which maintains a long-term memory of previous inputs. Thus, two recurrent layers are used in an LSTM. The diagram below illustrates the basic architecture.



In the diagram,

f is called the **forget gate**,
i is called the **input gate**,
$\tilde{c}$ is called the **candidate memory**,
o is called the **output gate**.

The architecture of an LSTM is usually illustrated as in the following diagram, which also contains some information on the activation function used at each layer.



Computation in a LSTM is done by first concatenating the current input $x(t)$ with the previous short-term memory $h(t-1)$ to get $x(t)h(t-1)$ and then computing

$$\begin{aligned} f(t) &= \sigma(x(t)h(t-1)W_f + b_f) \\ i(t) &= \sigma(x(t)h(t-1)W_i + b_i) \\ \tilde{c}(t) &= \tanh(x(t)h(t-1)W_{\tilde{c}} + b_{\tilde{c}}) \\ o(t) &= \sigma(x(t)h(t-1)W_o + b_o) \end{aligned}$$

where the weight matrix W_f combines the input weights with the recurrent weights in the manner described in section 7.1. Similarly, for the weight matrices W_i, $W_{\tilde{c}}$ and W_o.

The above vectors are then combined as follows:

$$\begin{aligned} c(t) &= f(t) \odot c(t-1) + i(t) \odot g(t) \\ h(t) &= o(t) \odot \tanh(c(t)) \end{aligned}$$

where $\odot$ represents pointwise multiplication of vectors:

$$(a_1, a_2, a_3, \ldots, a_k) \odot (b_1, b_2, b_3, \ldots, b_k) = (a_1b_1, a_2b_2, a_3b_3, \ldots, a_kb_k)$$

and the $+$ represents pointwise addition of vectors. Note that the vectors $f(t)$, $i(t)$, $\tilde{c}(t)$, $o(t)$, $c(t)$ and $h(t)$ are all of the same length.

EXERCISES

(1) Using the RNN example given in the notes above, compute the outputs for the following input sequence:

$$\begin{aligned}\boldsymbol{x}(1) &= (0,1)\\ \boldsymbol{x}(2) &= (1,-2)\\ \boldsymbol{x}(3) &= (-2,3)\\ \boldsymbol{x}(4) &= (3,-1).\end{aligned}$$

(2) Consider a RNN with 3 input nodes, a hidden recurrent layer with 2 nodes and an output layer with 1 node that has the following weight matrices and bias vectors:

$$W_X = \begin{bmatrix} 2 & -1 \\ -3 & -2 \\ 2 & 0 \end{bmatrix}, \quad W_H = \begin{bmatrix} 3 & -2 \\ -1 & 1 \end{bmatrix}, \quad W_O = \begin{bmatrix} 5 \\ -4 \end{bmatrix}$$

$$\boldsymbol{b}_H = (-2,1), \quad \boldsymbol{b}_O = (-1).$$

In addition, suppose that layers H and O use the *relu* activation function.

Compute the output values for the following input sequence into the above RNN:

$$\begin{aligned}\boldsymbol{x}(1) &= (1,-1,1)\\ \boldsymbol{x}(2) &= (0,-2,2)\\ \boldsymbol{x}(3) &= (3,3,-1)\\ \boldsymbol{x}(4) &= (4,-1,0).\end{aligned}$$

(3) Suppose that in the example given in the notes, a Jordan network was used instead of the Elman network. For the network, use the same matrices for W_X, W_O, $\boldsymbol{b}_H$ and $\boldsymbol{b}_O$, and the *relu* activation function. For W_H, since the output layer is fed back into the hidden layer, a weight matrix of dimentsion 2 × 3 is required, so replace W_H by the following:

$$W_H = \begin{bmatrix} 1 & -2 & 0 \\ 0 & 1 & -1 \end{bmatrix}.$$

Using the same input values $\boldsymbol{x}$, compute the outputs for this RNN.