Recurrent Neural Networks

① Let's reconsider the task of classifying DNA sequences. Suppose that a DNA sequence is in a particular family if it contains 3 or more "G" bases, e.g.

A G G T A A C G

G A T A G G G A T A

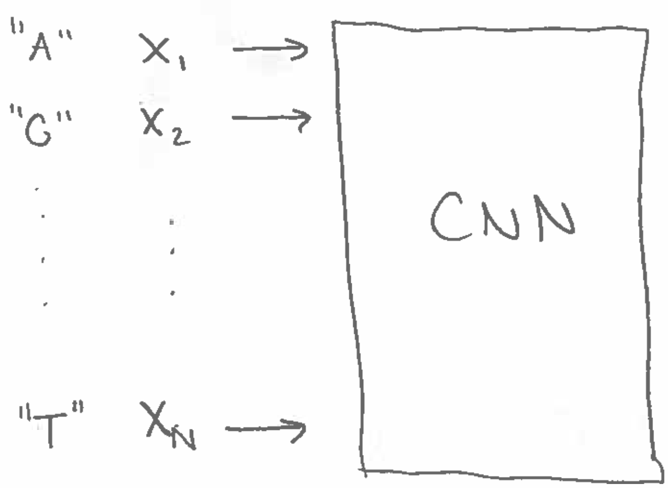
belong to the family, but:

T A T A C C A G A T T A

G A T T A C A G A T T A C A

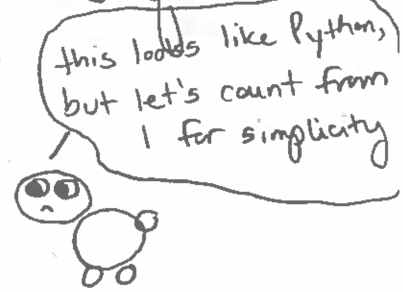
do not.

② One could try using a CNN, but maybe we don't know upfront the maximum length of a DNA sequence. That's problematic, because we need the input layer to be an upper bound on any sequence length



RECURRENT NEURAL NETWORKS

③ Alternatively, we could try to mimic the process of identifying positive sequences programmatically:




```
def f(X: string)
    Q = [False, False, False]
    for x in X:
        if x == "G" and not Q[1]:
            Q[1] = True
        elif x == "G" and not Q[2]:
            Q[2] = True
        elif x == "G" and not Q[3]:
            Q[3] = True
    return Q[3]
```

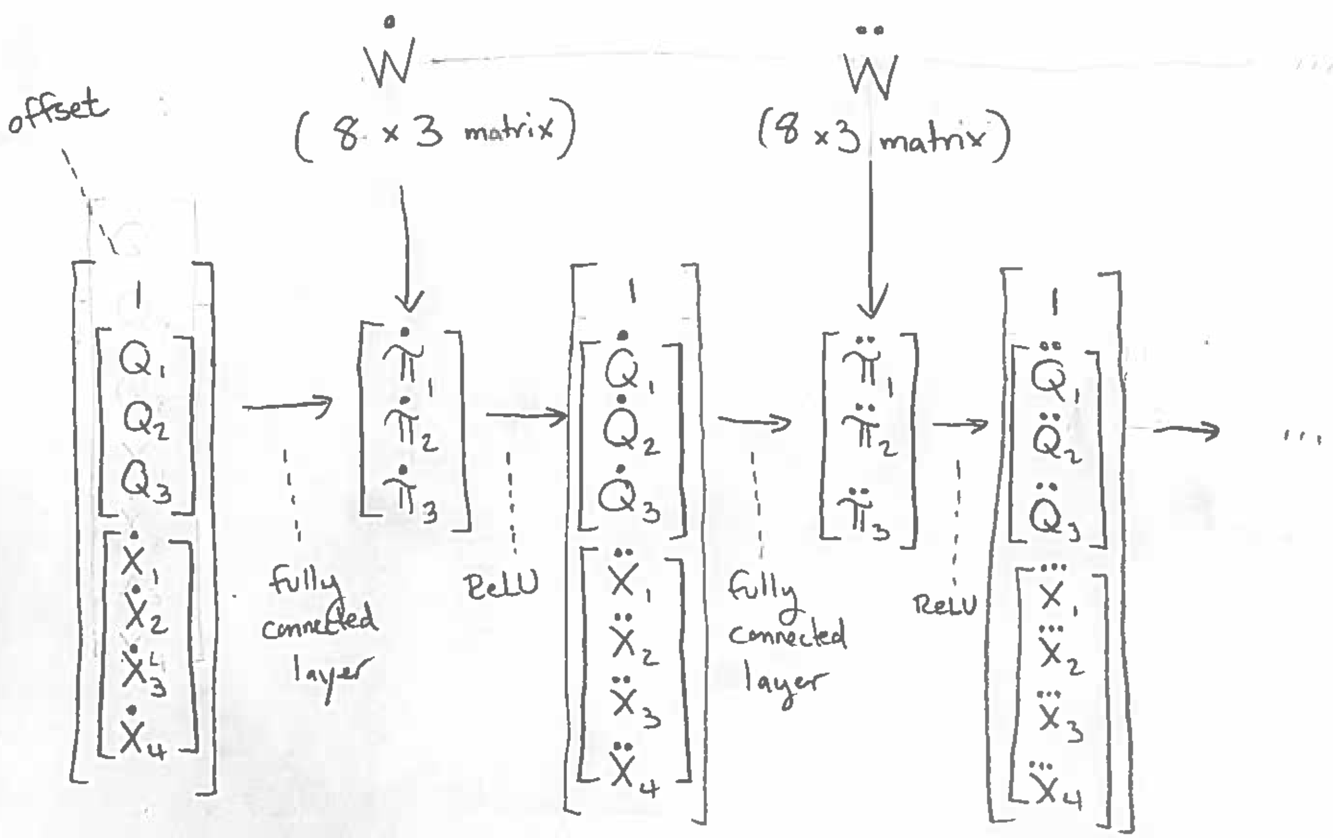
④ We could try encoding this as a neural network, in which rather than giving the entire input sequence as input to the first layer, we give the sequence one character at a time to each layer.

First, let's represent an input sequence A G G T G as one-hot vectors:

$$\begin{array}{ccccc} \dot{X} & \ddot{X} & \dddot{X} & \ddddot{X} & \overset{5}{X} \\ \text{"A"} & \text{"G"} & \text{"G"} & \text{"T"} & \text{"G"} \\ \begin{bmatrix}1\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\1\end{bmatrix} & \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \end{array}$$

Recurrent Neural Networks

⑤ The neural network representation of our Python function could look as follows:



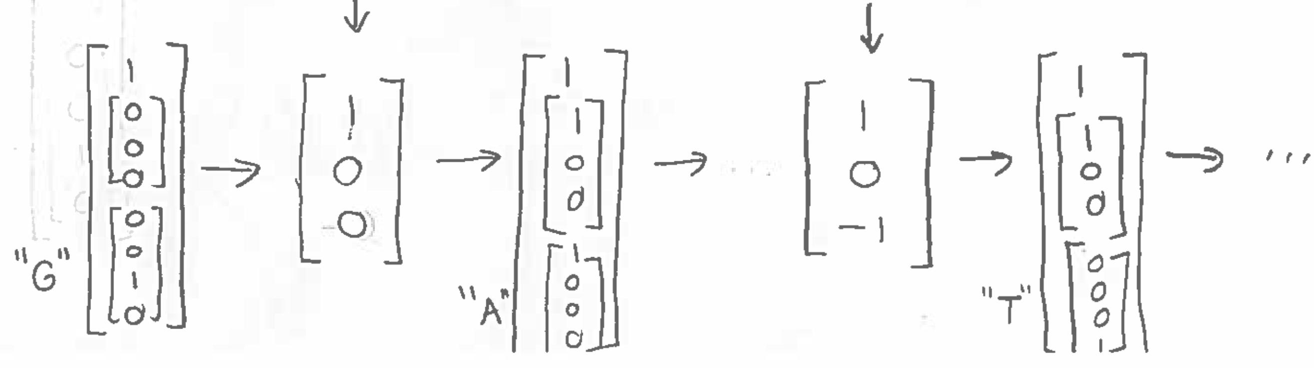
⑥ We can encode our logic (the if-statement) as the following fully connected layer

$$\begin{bmatrix} 0 & -1 & -1 \\ 1 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & -1 \\ 1 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

[Q_3 becomes positive if Q_2 was positive and $X_3 = 1$ (i.e. "G")

OR

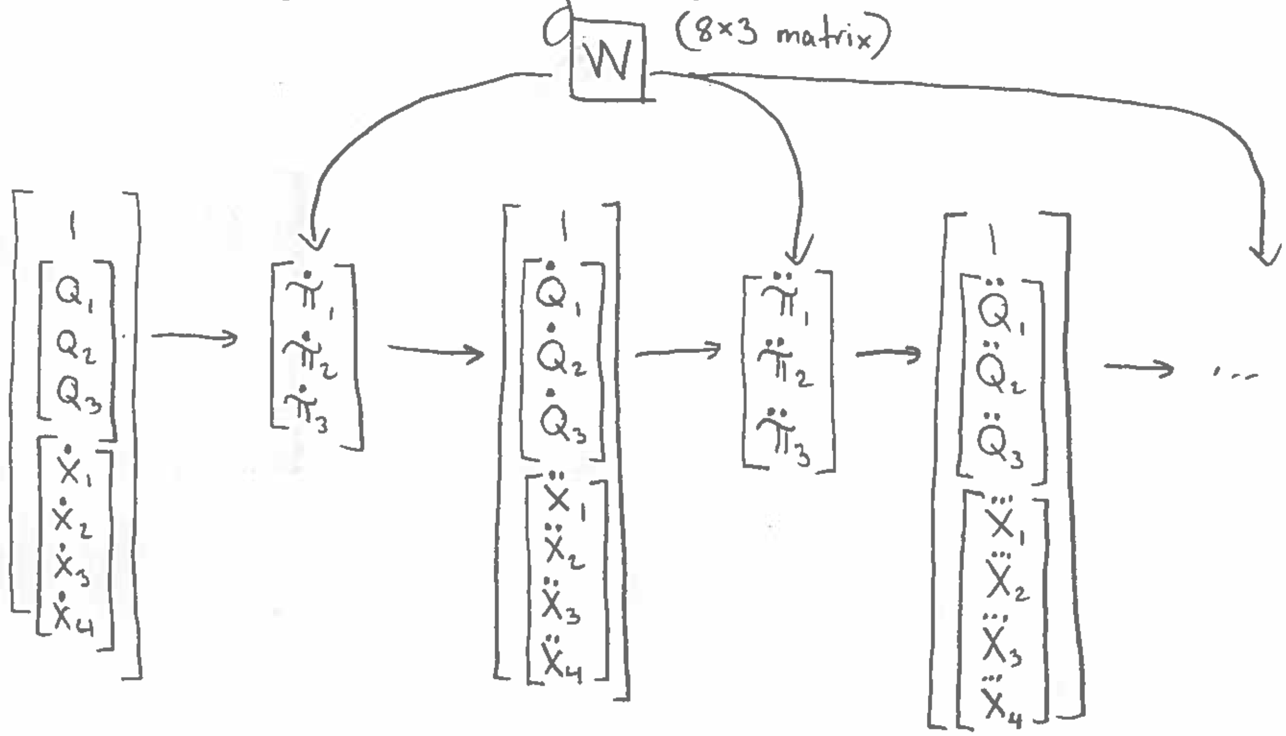
[Q_3 becomes positive if Q_3 was already positive



RECURRENT NEURAL NETWORKS

⑦ But just as, in a for-loop, the logic doesn't change from iteration to iteration, there's no need to change our weight matrix (which encodes the logic) from iteration to iteration.

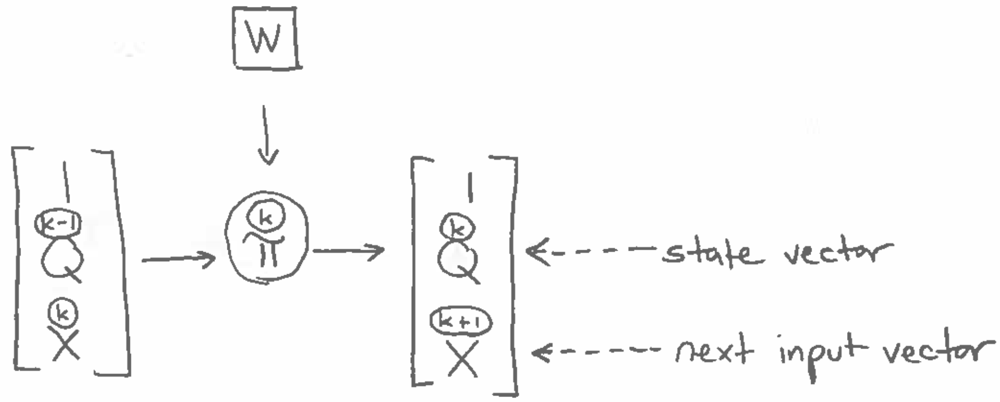
If we share these parameters between all layers, we obtain the following network:



This is called a <u>recurrent neural network</u> (RNN).

RECURRENT NEURAL NETWORKS

8) Observe that you can specify an RNN by showing how a single layer looks:

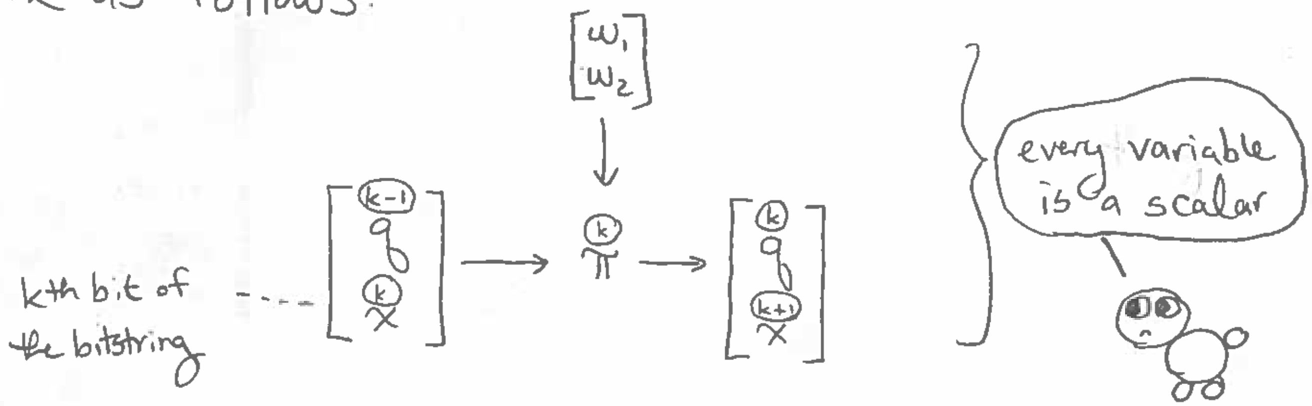


where $\overset{(k)}{\pi} = W^T \begin{bmatrix} 1 \\ \overset{(k-1)}{Q} \\ \overset{(k)}{X} \end{bmatrix}$, $\overset{(k)}{Q} = a\left(\overset{(k)}{\pi}\right)$

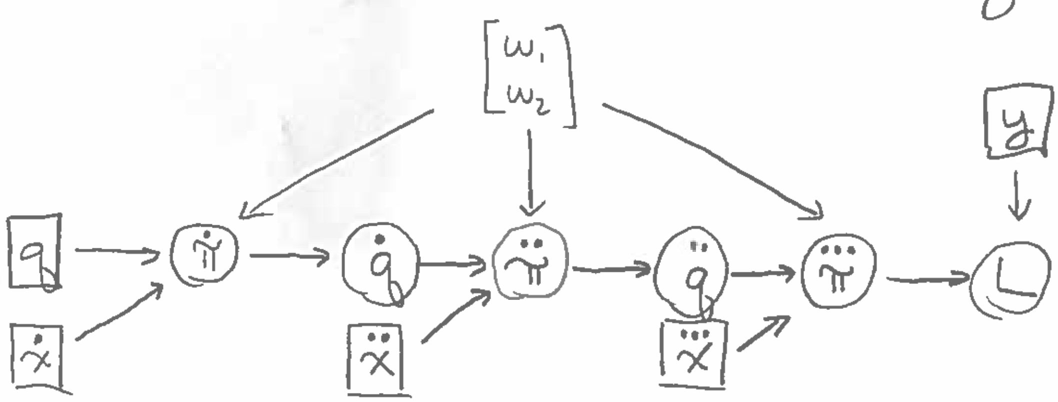
Recurrent Neural Networks

(9) Although we had weight sharing in CNNs, this is the first time we've encountered weight sharing <u>between</u> layers, so it's worth a look at how gradients are computed.

Consider an RNN for bitstrings, where the state vectors Q each have size 1. For simplicity, we won't use an offset. Thus, the RNN layers look as follows:



(10) Suppose we run this RNN on a bitstring of length 3:

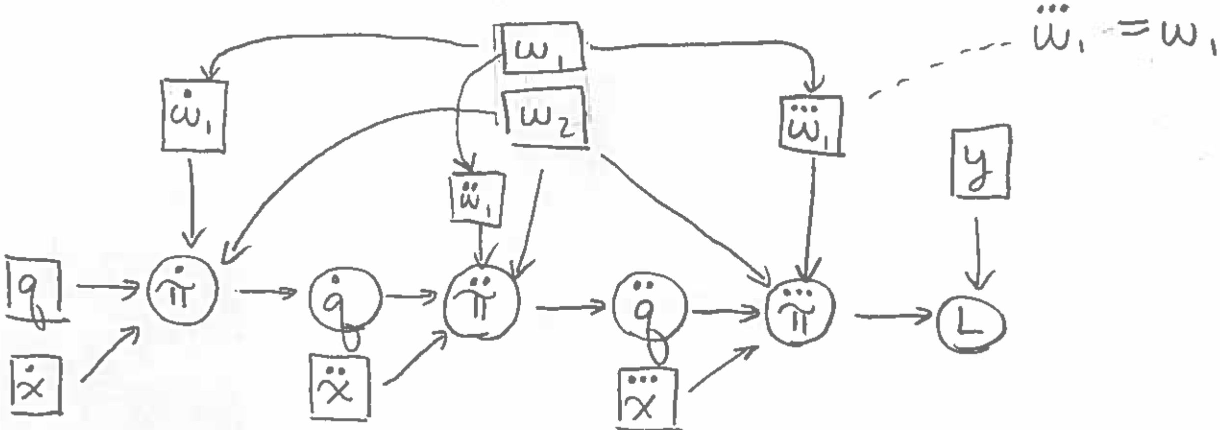


RECURRENT NEURAL NETWORKS

(11) Exercise: Compute $\dfrac{\partial L}{\partial w_1}$ for datum $(x,y) = (\text{"101"}, 15)$

at the point in the weight space $(w_1, w_2) = (4, 1)$
if $L(\dddot{\pi}, y) = (y - \dddot{\pi})^2$

It's more straightforward to apply the Chain Rule if we create copies of w_1 for each layer, i.e.



(12) Now, we can use $\{\dot{w}_1, \ddot{w}_1, \dddot{w}_1\}$ as a separating set between w_1 and $\dddot{\pi}$:

all of these equal 1

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \dddot{\pi}} \frac{\partial \dddot{\pi}}{\partial w_1}$$

$$= \frac{\partial L}{\partial \dddot{\pi}} \left(\frac{\partial \dddot{\pi}}{\partial \dot{w}_1} \cdot \frac{\partial \dot{w}_1}{\partial w_1} + \frac{\partial \dddot{\pi}}{\partial \ddot{w}_1} \frac{\partial \ddot{w}_1}{\partial w_1} + \frac{\partial \dddot{\pi}}{\partial \dddot{w}_1} \frac{\partial \dddot{w}_1}{\partial w_1} \right)$$

$$= \frac{\partial L}{\partial \dddot{\pi}} \left(\frac{\partial \dddot{\pi}}{\partial \dot{w}_1} + \frac{\partial \dddot{\pi}}{\partial \ddot{w}_1} + \frac{\partial \dddot{\pi}}{\partial \dddot{w}_1} \right)$$

$$= -2(y - \dddot{\pi}) \left(\frac{\partial \dddot{\pi}}{\partial \dot{w}_1} + \frac{\partial \dddot{\pi}}{\partial \ddot{w}_1} + \frac{\partial \dddot{\pi}}{\partial \dddot{w}_1} \right)$$

(13) The remaining derivatives can be computed efficiently through backpropagation:

$$\frac{\partial \dddot{\pi}}{\partial \dddot{w}_1} = \ddot{q}$$

$$\frac{\partial \dddot{\pi}}{\partial \ddot{w}_1} = \frac{\partial \dddot{\pi}}{\partial \ddot{\pi}} \frac{\partial \ddot{\pi}}{\partial \ddot{w}_1} = \dot{q} \cdot \frac{\partial \dddot{\pi}}{\partial \ddot{\pi}}$$

$$\frac{\partial \dddot{\pi}}{\partial \dot{w}_1} = \frac{\partial \dddot{\pi}}{\partial \dot{\pi}} \frac{\partial \dot{\pi}}{\partial \dot{w}_1} = q \cdot \frac{\partial \dddot{\pi}}{\partial \dot{\pi}}$$

where:

$$\frac{\partial \dddot{\pi}}{\partial \ddot{\pi}} = \frac{\partial \dddot{\pi}}{\partial \ddot{q}} \frac{\partial \ddot{q}}{\partial \ddot{\pi}} = \dddot{w}_1 \cdot a'(\ddot{\pi}) = w_1 \cdot a'(\ddot{\pi})$$

$$\frac{\partial \dddot{\pi}}{\partial \dot{\pi}} = \frac{\partial \dddot{\pi}}{\partial \dot{q}} \cdot \frac{\partial \dot{q}}{\partial \dot{\pi}}$$

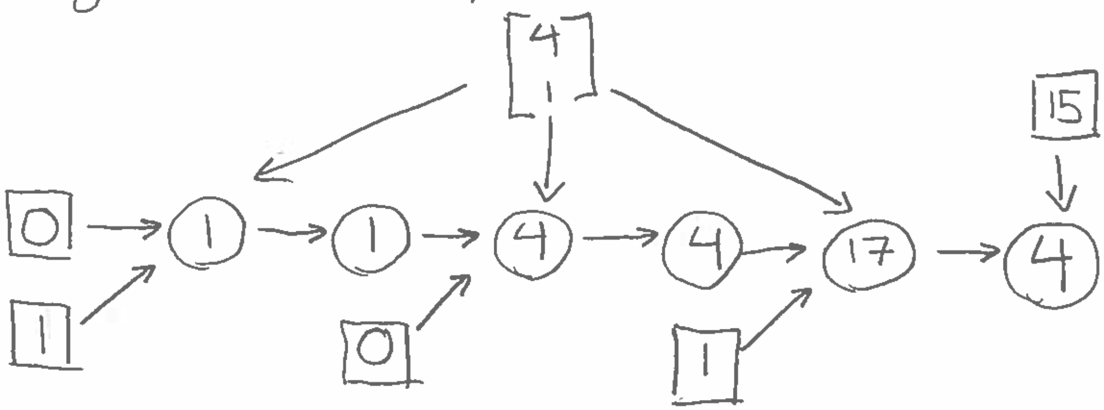
$$= \frac{\partial \dddot{\pi}}{\partial \ddot{\pi}} \cdot \frac{\partial \ddot{\pi}}{\partial \dot{q}} \cdot \frac{\partial \dot{q}}{\partial \dot{\pi}} = \left(w_1 \cdot a'(\ddot{\pi})\right) \ddot{w}_1 \cdot a'(\dot{\pi}) = w_1^2 \cdot a'(\ddot{\pi}) \cdot a'($$

(14) So then:

$$\frac{\partial L}{\partial w_1} = -2(y - \dddot{\pi})\left(\ddot{q} + \dot{q} \cdot w_1 \cdot a'(\ddot{\pi}) + q \cdot w_1^2 \cdot a'(\ddot{\pi}) \cdot a'(\dot{\pi})\right)$$

RECURRENT NEURAL NETWORKS

15 Doing the forward pass on the specific data, we get:



16 So our partial derivative evaluates to:

$$\frac{\partial L}{\partial w_1} = -2(15-17)\left(4 + 1\cdot 4\cdot 1 + 0\cdot 4^2\cdot 1\cdot 1\right)$$

$$= -2(-2)(8)$$

$$= 32$$