Recurrent Neural Networks

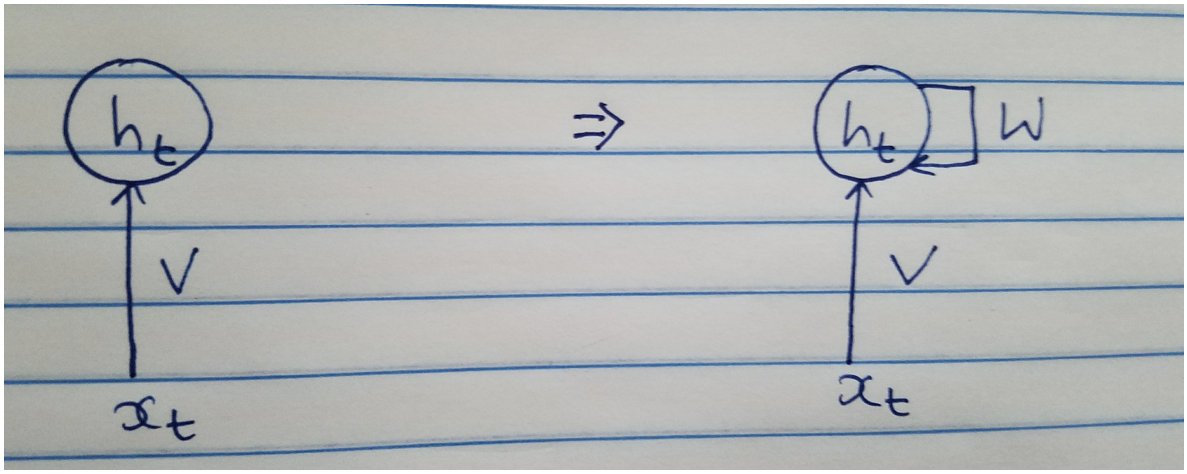
"RNN is similar to the human brain, which is a large feedback network of connected neurons that somehow can learn to translate a lifelong sensory input stream into a sequence of useful motor outputs. The brain is a remarkable role model as it can solve many problems current machines can not yet solve."

-Juergen Schmidhuber (leading researcher)

Do you know how many years it take for you(a human being) to learn how to interpret this _ _ _ _ _ _ _ _?

What do you think should be the last word in the above question? Did you predict it to be "question", "sentence", "language"? Your biological neural network (brain) is at work there. In this video, we are going to understand the neural networks called “recurrent neural networks” that are used for processing sequential data. Just like the sentence that your brain processed and filled in the blank with some word.

Let's look at what a recurrent neuron looks like:

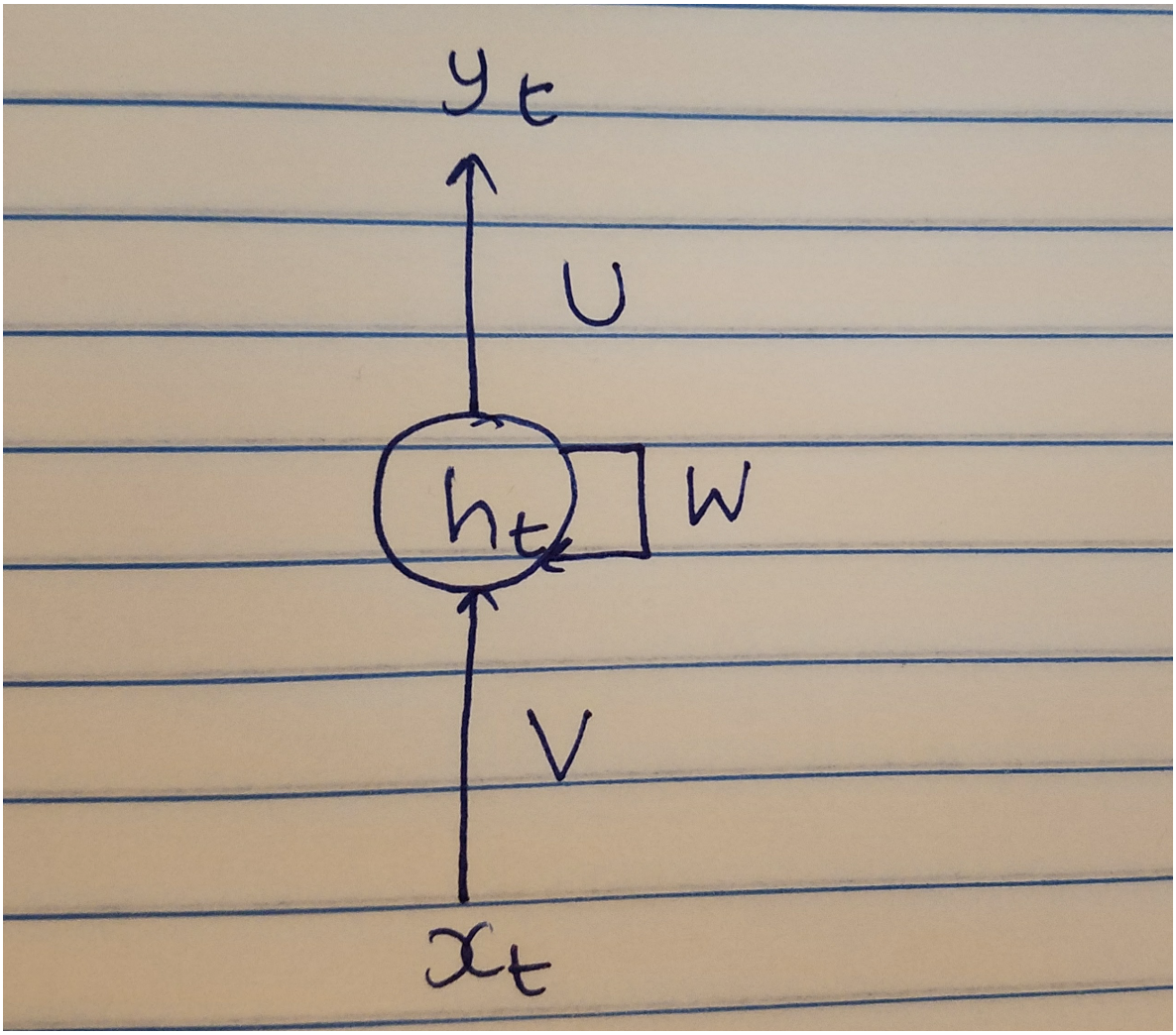


Caption: First build up the image on the left, then add the recurrent relation to it. Left: A simple linear neuron which takes input x_t and multiplies it with some weight matrix V which is passed to hidden unit h_t. Right: A recurrent neuron with weight matrix W and hidden unit value from previous time step $h_{(t-1)}$ being passed along with the input x_t and weight matrix V. Both these units do not have any output.

For now do not think about where the values of h_t, W or V comes from. The only thing I want you to understand for now is the feedback loop that is being created.

Now let's understand the architecture of a recurrent neural network:

Here in the figure below we have a recurrent neural network with input x_t, output y_t, hidden unit h_t, weight matrix V, W, U. The hidden unit h_t is a simple neuron which fires when some satisfying criteria is met. The weight matrix gets its values from training the network on training data. Training a neural network means feeding relevant data to the network to make it replicate or create something on its own. Suppose if we are training an RNN to write the next book of game of thrones then we will train it on all the game of thrones book that is released till now and it will generate next book on its own when trained. Before the training the network is initialized with some values. It can be either 0’s or random values.



Caption: A recurrent neuron with inputs x_t, hidden unit h_t,weight matrix V and weight W which is passed on from the previous iteration and the output y_t generated using h_t and weight U.

Let's see what equations this RNN would use:

$h_t = f_h(Vx_t + Wh_{(t-1)} + b_h)$

where

h_t is the value of hidden unit for current input

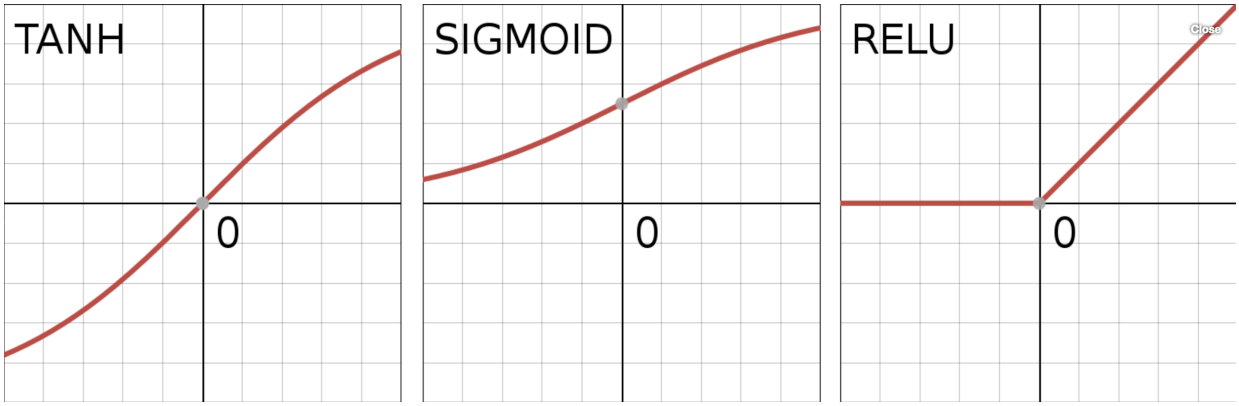
V is the matrix trained based on the input data that we passed while training the network

x_t is the current input

W is the recurrent weight matrix that passes the information about the previous step

$h_{(t-1)}$ is the value of the hidden unit at the previous timestamp

f_h can be any activation function (nonlinearity) (e.g. tanh, sigmoid, relu)



Caption: This would just be a refresher as they are mentioned in detail in the beginning of the Deep Learning series.

$y_t = f_y(Uh_t + b_y)$

where

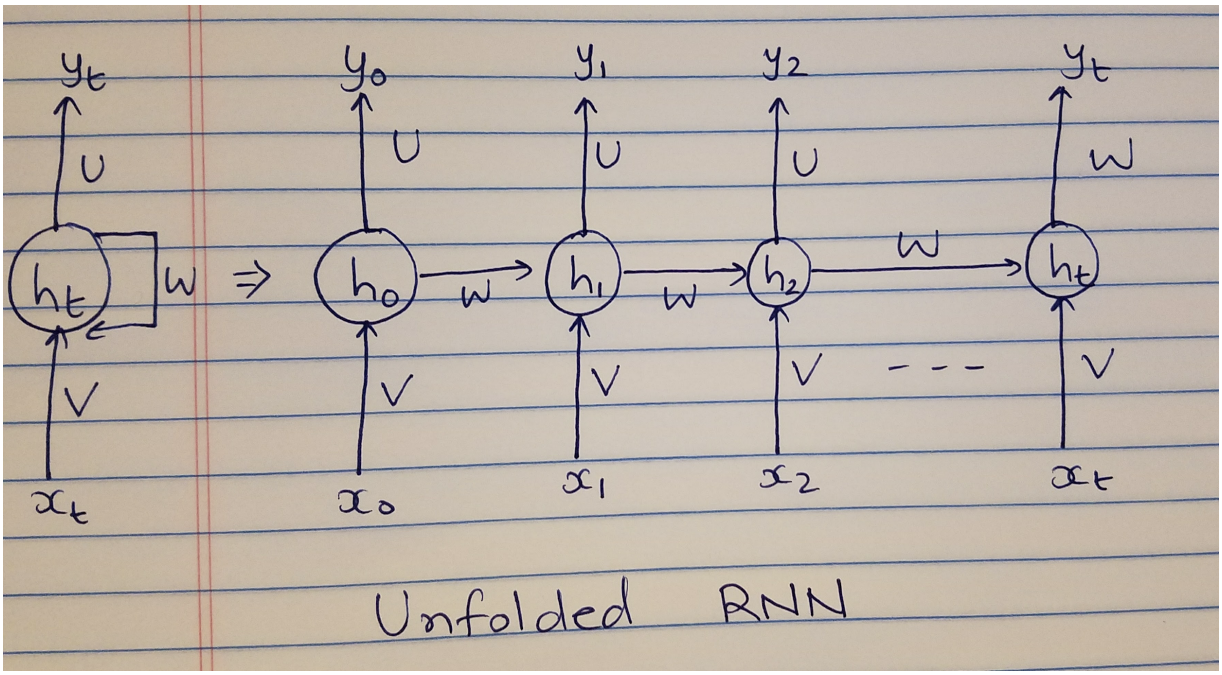
y_t is the output at current time step

U is the weight matrix trained based on the input data while training the network

f_y can be any function (nonlinearity) (e.g. tanh, sigmoid, relu)

Let's look at how the above recurrent neural network would look like for some time step t:

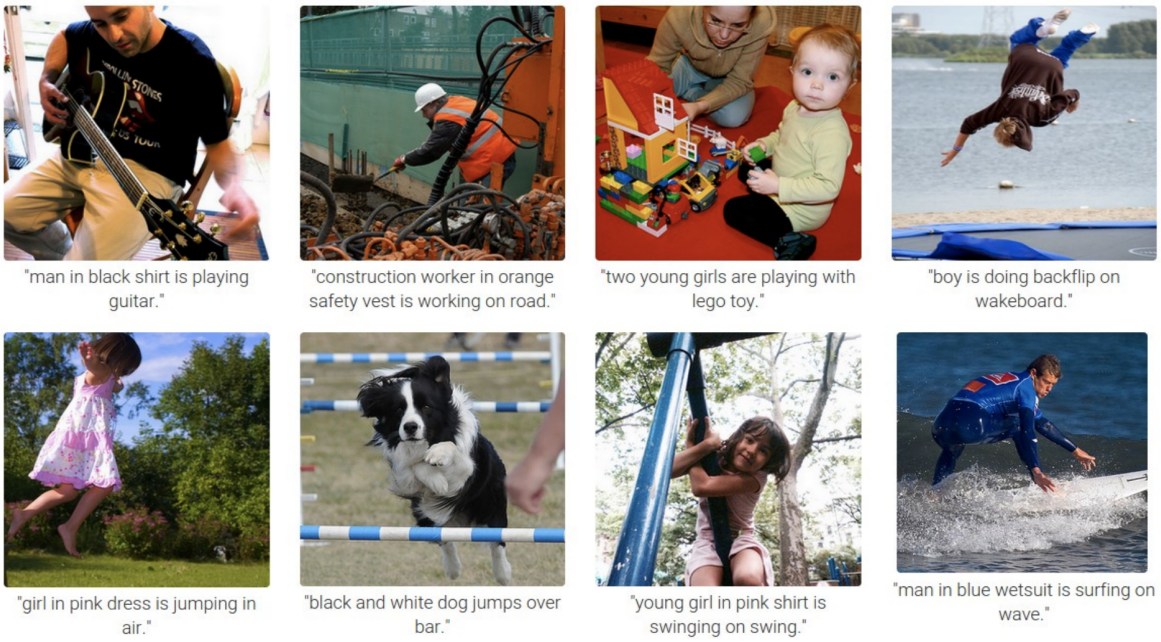
Here in the figure below, you can see that the one on the left is the same RNN as the one on the right. The one on the right is more easier to understand. At each time step whenever RNN gets an input it updates the values of parameters h_t, y_t. The values of U, V, W matrices keep on updating while we are training our RNN. From now onwards we would use this unfolded representation on the right.



Caption: Unfolded RNN

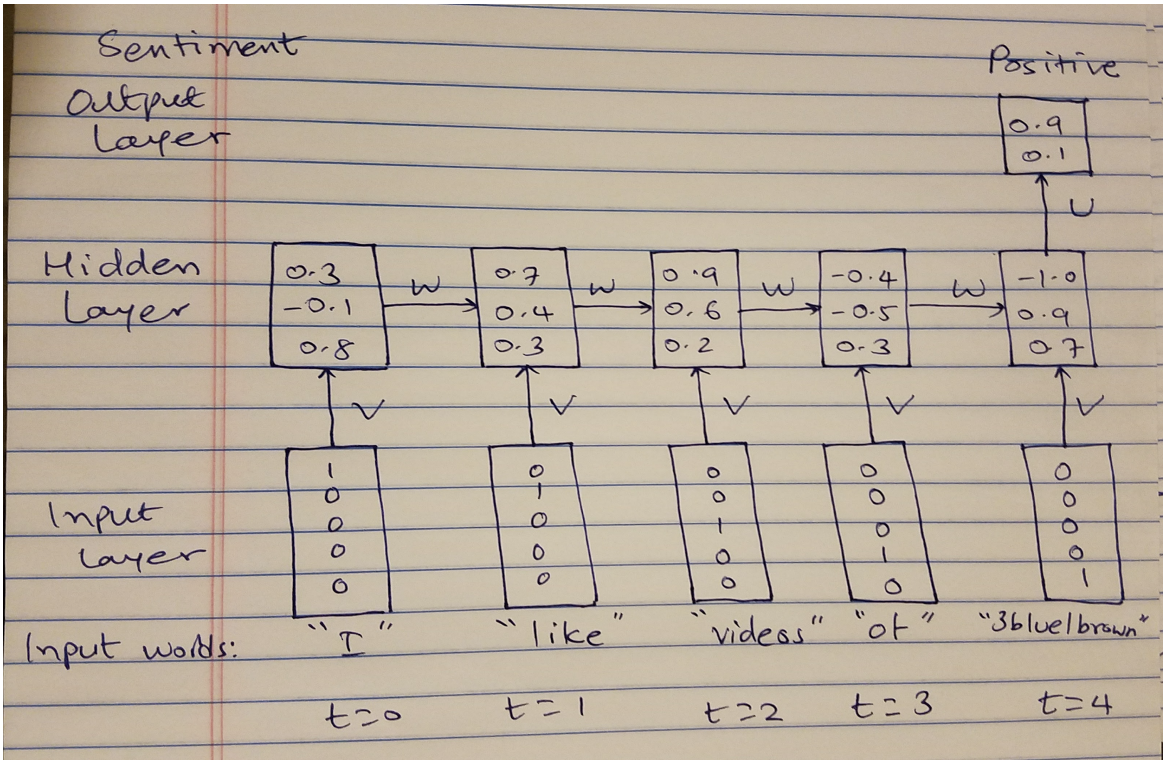
The core idea behind RNN is sharing parameters across different parts of the model. This makes it possible to extend and apply the model to parameters of different lengths and generalize across them.
In case of other feed-forward neural networks (CNN) the input parameters are of fixed size, passed only once to the model in the beginning and all the examples are treated as independent.
Now we will discuss different types of RNN architectures:

1. **One to many:** RNN that takes an image as input and generates sequence of words (word vectors) as output. Below is an example of captions generated by a recurrent neural network from these images.



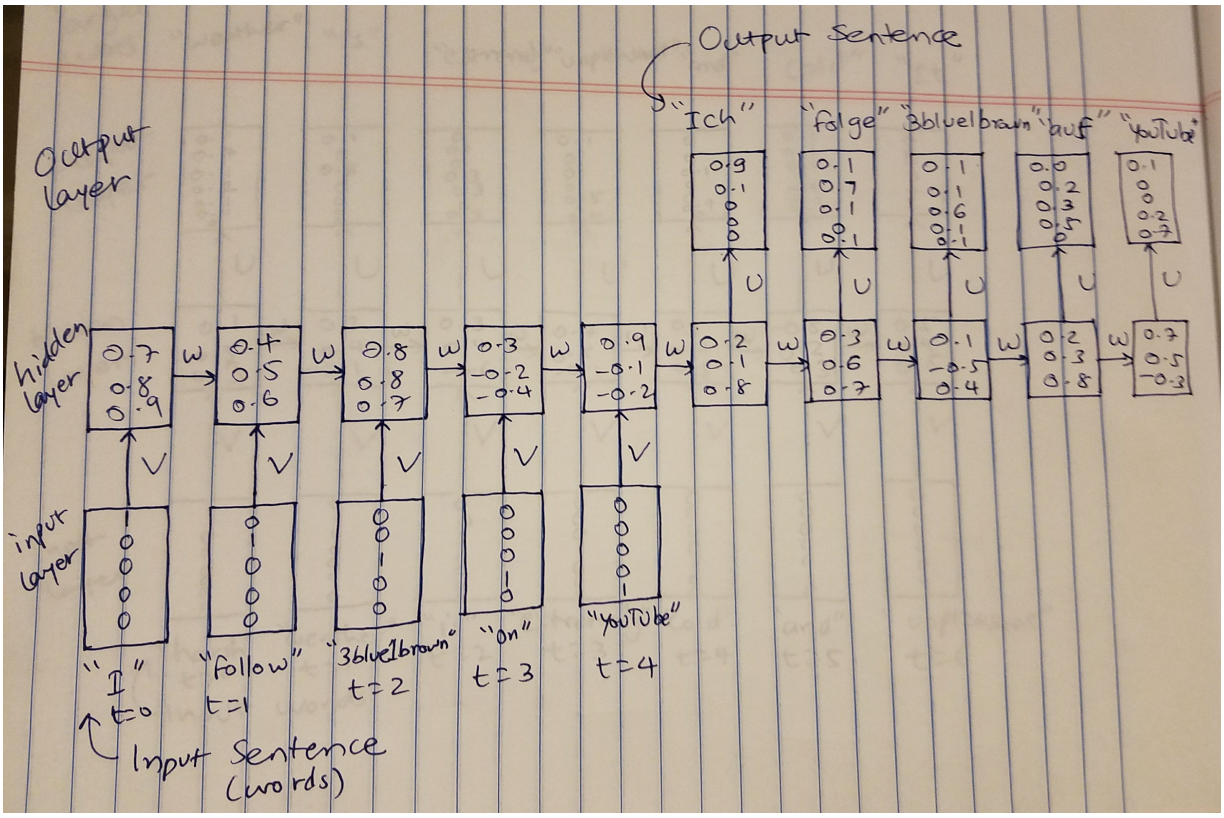
2. **Many to one:**

RNN that takes sequence as input and generates a single category as output. Mentioned below is the sentiment analysis model, the layer at the bottom is the input layer which gets input words from a tweet. Which it parses and when the sentence ends recognize the sentiment of the tweet positive or negative. Here I would like you to do the same along with the RNN.
The output layer assigns 90% probability that the tweet has a positive sentiment. While training this RNN we can feed dataset of all the tweets, based on which the model learned to categorize this tweet. Just as you categorized it.
The input is passed as one hot encoding which means that there is 1 at the location of the word that is encoded (or linked) to the vector representation of the words. In this example the training data contains these 5 words only. At each time step, it parses the words one by one and waits for the sentence to end. As it ends it gives us the output. It finds the output based on some words that describe the sentiment more appropriately. Here word “like” would be given the highest weight.



Caption: One by one each iteration would appear from left (t=0) to right (t=4).

3. **Many to many:** RNN takes in sequence as input and generates sequence as output.

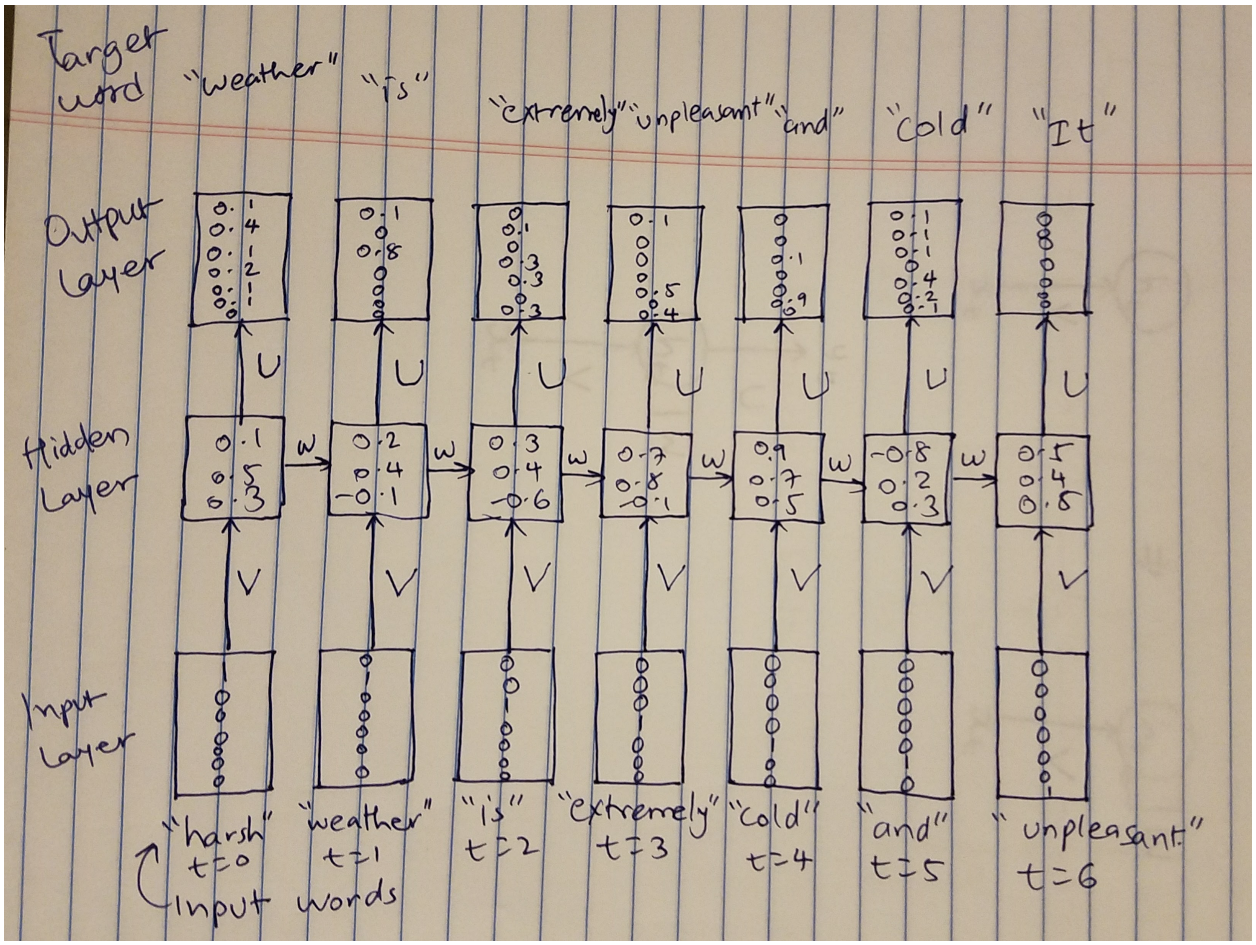


Caption: One by one each iteration would appear from left to right. The task of the viewer is to translate the sentence from English to German.

In the case of the above language translation model, the layer at the bottom is the input layer which gets input words from a sentence. Which it parses and when the sentence ends the model translates it to German. This is similar to how a human would translate a sentence. Here our input vector has size 5, our hidden vector has size 3 and our output vector has size 5. So the total number of parameters our recurrent neural network has are: W = HxH = 3x3 = 9, V = HxI = 3x5 = 15, U = HxO = 3x5 = 15
I.e. 9+15+15=39.

Language Model:

Here in the model we want to predict the probability of the next word given the current word. This is the collection of words that we have: “weather”, “harsh”, “is”, “and”, “cold”, “unpleasant”, “extremely”.



Caption: Language Model (here the task of the viewer is to predict the next word in the sentence)

As you can see at time t=2 the model shows an equal probability of 30% (0.3) for words “extremely”, “cold” and “unpleasant” as all 3 of them are equally likely and it is confusing. It chooses one of those words and it ends up being correct. Also, notice that at timestamp t=3 the network predicts that the next word would be unpleasant but it is cold. As both “unpleasant” and “cold” are equally likely to appear after extremely.

Backpropagation Through Time:

Training a RNN is similar to training a traditional neural network. We use the backpropagation algorithm. The parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps. For example, in order to calculate the gradient at t = 4 we would need to backpropagate 3 steps and sum up the gradients. This is called Backpropagation Through Time (BPTT). A vanilla RNN trained with BPTT have difficulties learning long-term dependencies (e.g. dependencies between steps that are far apart) due to what is called the vanishing/exploding gradient problem. There exists some methods to deal with these problems, and certain types of RNNs (like LSTMs) were specifically designed to get around them.

Few applications of RNN and where we get sequential data in the real world:

- Time-series prediction (stock price prediction, weather prediction)
- Video analysis (sequence of images)
- Music information retrieval (audio data)
- Natural language translation (sequence of words/sentences)

Note: If possible we can include the melody this or this one (search for results) generated by an LSTM in the video as a background audio. And inform them about it in the end, the music for this video was generated by an RNN :D

References:

1. Intro to Deep Learning Coursera
2. RNN Nervana Systems
3. Game of thrones book generator
4. Image captioning RNN (slide 4)
5. Performance RNN: Generating music using RNN
6. Andrej Karpathy's explanation for RNN
7. Chris Olah's blog for LSTM
8. Deep Learning book
9. Deep Learning book 2 (chapter 4)
10. Hands-On Machine Learning (chapter 14)
11. RNN wildml blog