# RECURRENT NEURAL NETWORKS ( RNNs )

As we delve deeper into language processing, two major issues arise.

Issue 1:

**Variable length sequences of words**

▪ With images, we forced them into a specific input dimension

▪ For example, classify tweets, which can have variable number of words, as positive, negative, or neutral

Issue 2:

**Ordering of words is important**

▪ An ordered implementation works better than “bag of words” implementations

▪ Ideally, each word is processed or understood in the appropriate context

▪ Need to have some notion of “context”

▪ Words should be handled differently depending on “context”

▪ Also, each word should update the context

To handle the issues mentioned above, we use the notion of “**recurrence**”

▪ Input words one by one

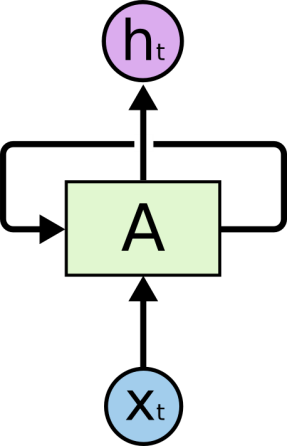
▪ Network outputs two things:

– Prediction: That is, it would try to predict what would be the possible options if the sequence ended with a particular word

– State: Summary of everything that happened in the past

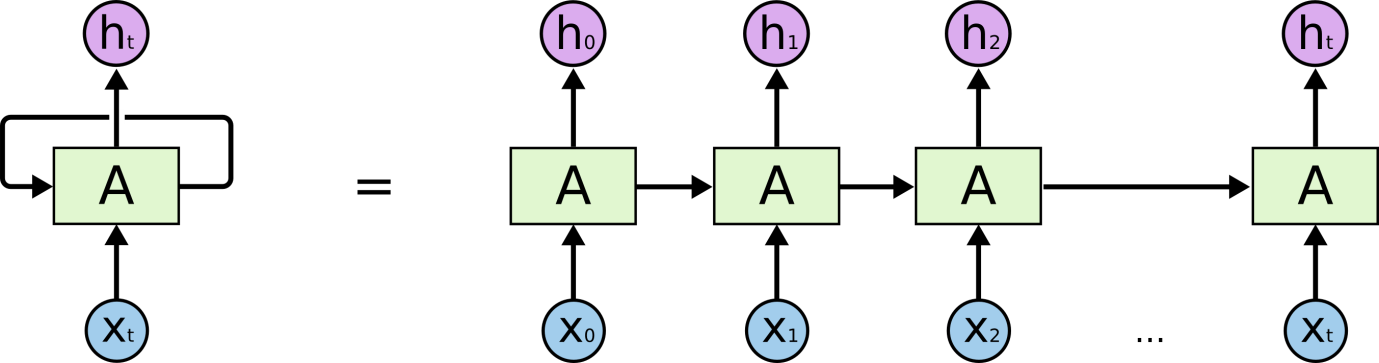
▪ This way, we can handle variable lengths of text

▪ The response to a word depends on the words that preceded it



In the above diagram, a chunk of neural network, A, looks at some input xt and outputs a value ht. A loop allows information to be passed from one step of the network to the next. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:

“Unrolling” the RNN



As you can see in the unrolled version. First, it takes the x0 from the sequence of input and then it outputs h0 which together with x1 is the input for the next step. So, the h0 and x1 is the input for the next step. Similarly, h1 from the next is the input with x2 for the next step and so on. This way, it keeps remembering the context while training.

**The following are the few applications of the RNN:**

• Next word prediction.

• Music composition.

• Image captioning

• Speech recognition

• Time series anomaly detection

• Stock market prediction



A RNN and the unfolding in time of the computation involved in its forward computation. Source: Nature

**Mathematical Details**

▪ xi is the word at position i

▪ si is the state at position i

▪ oi is the output at position i

▪ si = f(Uwi + Wsi−1 ) (Core RNN)

▪ oi = softmax(Vsi ) (subsequent dense layer)

In other words:

▪ current state = function1(old state, current input)

▪ current output = function2(current state)

▪ We learn function1 and function2 by training our network!

More Mathematical Details

▪ r = dimension of input vector

▪ s = dimension of hidden state

▪ t = dimension of output vector (after dense layer)

▪ U is a s x r matrix

▪ W is a s x s matrix

▪ V is a t x s matrix

Note: The weight matrices U,V,W are the same across all positions.

**Practical Details**

▪ Often, we train on just the ”final” output and ignore the intermediate outputs

▪ Slight variation called Back Propagation Through Time (BPTT) is used to train RNNs

▪ Sensitive to length of sequence (due to “vanishing/exploding gradient” problem)

▪ In practice, we still set a maximum length to our sequences – If input is shorter than maximum, we “pad” it – If input is longer than maximum, we truncate

**Other Uses of RNNs**

We have focused on text/words as application But, RNNs can be used for other sequential data

– Time-Series Data

– Speech Recognition

– Sensor Data

– Genome Sequences

**Weaknesses of RNNs**

▪ Nature of state transition means it is hard to keep information from distant past in current memory without reinforcement

This is overcome by LSTMs.

References:

1. colah.github.io/posts/2015-08-Understanding-LSTMs/

2. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/