S21CSE471: Statistical Methods in AI **Spring 2021**

Recurrent Neural Networks (RNN)

Prepared by: Mansi Khamkar, Kunal Kandhari, Mahak Modani

In this note, we discuss Recurrent Neural Networks (RNN) which was taught in the lecture 15-April-2021

1 Motivation behind Recurrent Neural Networks (RNN)

The motivation behind RNN is the limitation of MLP to process sequential data.

First, let's understand in which tasks we really need to work with sequential data. We know several examples of such data Text, Video, and Audio, we use them every day, and all of them are sequences.

- Text is a sequence of sentences or words or even letters.
- Video is a sequence of images.
- Audio is a sequence of sounds.

But these are only the most common examples of such data. There are a lot more of them. You can find time series in a lot of different areas, in finance, in industry, in medicine, and so on. And time series, of course, are also sequences in time. For example, in medicine, all of the data from medical sensors are usually sequential. So as we can see, there are a lot of different tasks with sequential data.

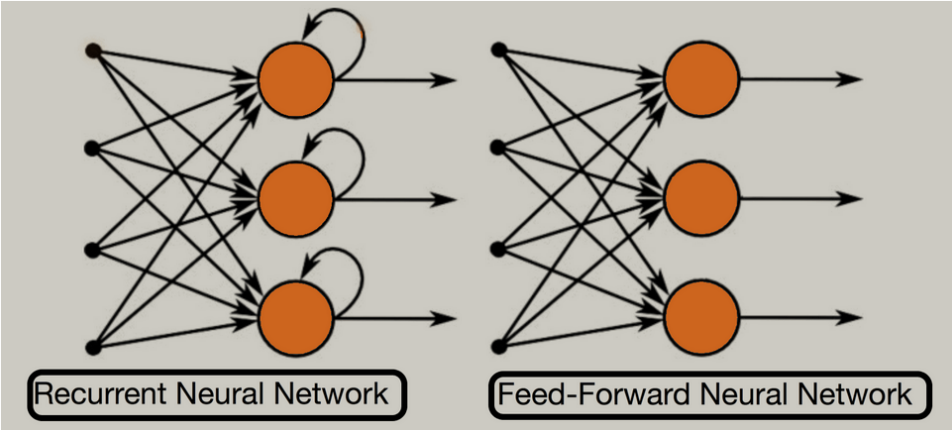
Let us consider a specific example of simple task with sequential data called **Language model**. In this task we want to train a generative model of natural language. So, we want to be able to generate text from the probability distribution P of text. We can actually use Language model in all the different tasks in practice. For example, we can use it to generate answers in chatbots or question answering systems. But of course, here, we should condition our Language model on the previous conversation, because the model should know what the question was about to answer it. Also, Language model can be useful in machine translation or speech recognition to understand which phrase is more natural to a language.. Now, let's try to use a standard multilayer perceptron to construct our Language model. Let's say that we've already generated i words, and we want to generate the next word, i + 1. Then, we can use i words we have as an input on our MLP. Of course, here, we need to use some numerical features for these words, for example one-hot encodings or word2vec embeddings. And as an output of our MLP we can have a distribution over our vocabulary, which says which word is more probable to be the next one in our sequence. Okay, this model will work. And now, we have i + 1 words, and we want to generate the next word, i + 2. And what we want to do, we actually want to give all the words we have as an input to our MLP. But we can't do it because MLP can work only with fixed number of inputs. So here we see the main problem of using standard MLP with sequential data.

The problem is that sequences can have different lengths. So, what should we do with this? We can use a simple heuristic and give not all of the previous words as an input to our MLP, but only a fixed number of them. So, we have a window of fixed size as an input.

To overcome this problem we can use a recurrent architecture. which we will study in following sections.

2 Recurrent Neural Networks (RNN)

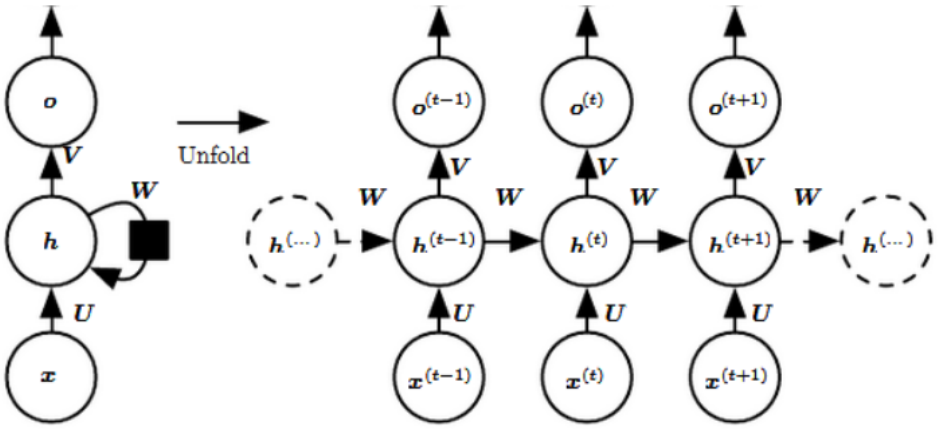
- Recurrent neural networks (RNNs), are a family of neural networks that are specialized in processing sequential data (a sequence of values x(1), . . . , x(n)).
- Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states.
- RNNs are networks with loops (hence the name 'recurrent') allowing information to persist in their memory unit (hidden state). Hence, when RNN makes a decision, it considers the current input and also what it has learned from the inputs it received previously. While the depth of the usual neural network depends on the number of hidden layers it has, the depth factor of the RNN is induced by the time component (number of timesteps it processes) because of its recurrent nature.



- An RNN consists of only one hidden/memory unit which takes input and produces an output at each timestep. This unit also has a self loop which passes the learned information about the previous input to the next timestep. Thus we can unfold the recursive computations of an RNN into a computational graph that has a repetitive structure, typically corresponding to a chain of events.

3 Architecture of a traditional RNN

Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. They are typically as follows:



The left side of the above diagram shows a notation of an RNN and on the right side an RNN being unrolled (or unfolded) into a full network. By unrolling we mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 3 words, the network would be unrolled into a 3-layer neural network, one layer for each word.

Input: $x(t)$ is taken as the input to the network at time step t. For example, $x1$, could be a one-hot vector corresponding to a word of a sentence.

Hidden state: $h(t)$ represents a hidden state at time t and acts as memory of the network. $h(t)$ is calculated based on the current input and the previous time step's hidden state: $h(t) = f(Ux(t)+Wh(t-1))$ The function f is taken to be a non-linear transformation such as $tanh$, $ReLU$.

Weights: The RNN has input to hidden connections parameterized by a weight matrix U, hidden-to-hidden recurrent connections parameterized by a weight matrix W, and hidden-to-output connections parameterized by a weight matrix V and all these weights (U,V,W) are shared across time.

Output: $o(t)$ illustrates the output of the network. In the figure I just put an arrow after $o(t)$ which is also often subjected to non-linearity, especially when the network contains further layers downstream.

Forward Pass: The figure does not specify the choice of activation function for the hidden units. Before we proceed we make few assumptions:
1) We assume the hyperbolic tangent activation function for hidden layer.
2) We assume that the output is discrete, as if the RNN is used to predict words or characters.
A natural way to represent discrete variables is to regard the output o as giving the un-normalized log probabilities of each possible value of the discrete variable. We can then apply the softmax operation as a post-processing step to obtain a vector ŷof normalized probabilities over the output.

The RNN forward pass can thus be represented by below set of equations.

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = tanh(a^{(t)})$$
$$o^{(t)} = c + Vh^{(t)}$$
$$\hat{y}^{(t)} = softmax(o^{(t)})$$

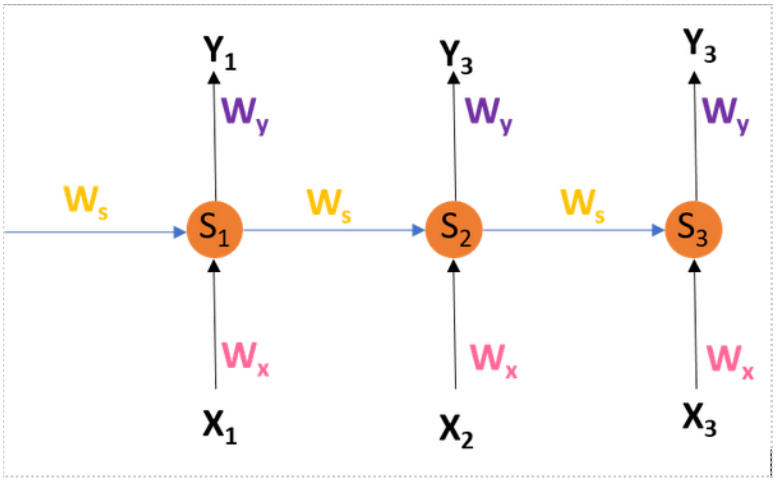
This is an example of a recurrent network that maps an input sequence to an output sequence of the same length. The total loss for a given sequence of x values paired with a sequence of y values would then be just the sum of the losses over all the time steps. We assume that the outputs $o(t)$ are used as the argument to the softmax function to obtain the vector $\hat{y}$ of probabilities over the output. We also assume that the loss L is the negative log-likelihood of the true target $y(t)$ given the input so far.

Backward Pass: The gradient computation involves performing a forward propagation pass moving left to right through the graph shown above followed by a backward propagation pass moving right to left through the graph. The runtime is $O(\tau)$ and cannot be reduced by parallelization because the forward propagation graph is inherently sequential; each time step may be computed only after the previous one. States computed in the forward pass must be stored until they are reused during the backward pass, so the memory cost is also $O(\tau)$. The back-propagation algorithm applied to the unrolled graph with $O(\tau)$ cost is called back-propagation through time (BPTT). Because 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.

Backpropagation Through Time (BPTT) for RNNs

BPTT is an algorithm used for training RNNs. It is an application of the usual backpropagation training algorithm used to train neural networks. As we know, an RNN is given one input at each timestep, for which it produces an output. So, we have a loss function at each timestep and we update the weights

with the usual update rule used in backpropagation by aggregating over all the losses. But the depth of the network in RNN is induced by the time component, hence the name - 'Backpropagation Through Time'. BPTT basically works by unrolling all timesteps, where it calculates and accumulates errors for each timestep. These timesteps are then rolled back up and the weights of the network are updated by using the standard chain rule of derivatives. Thus, we do not independently train the network at a specific timestep, but also for what the network has predicted for the previous timesteps.

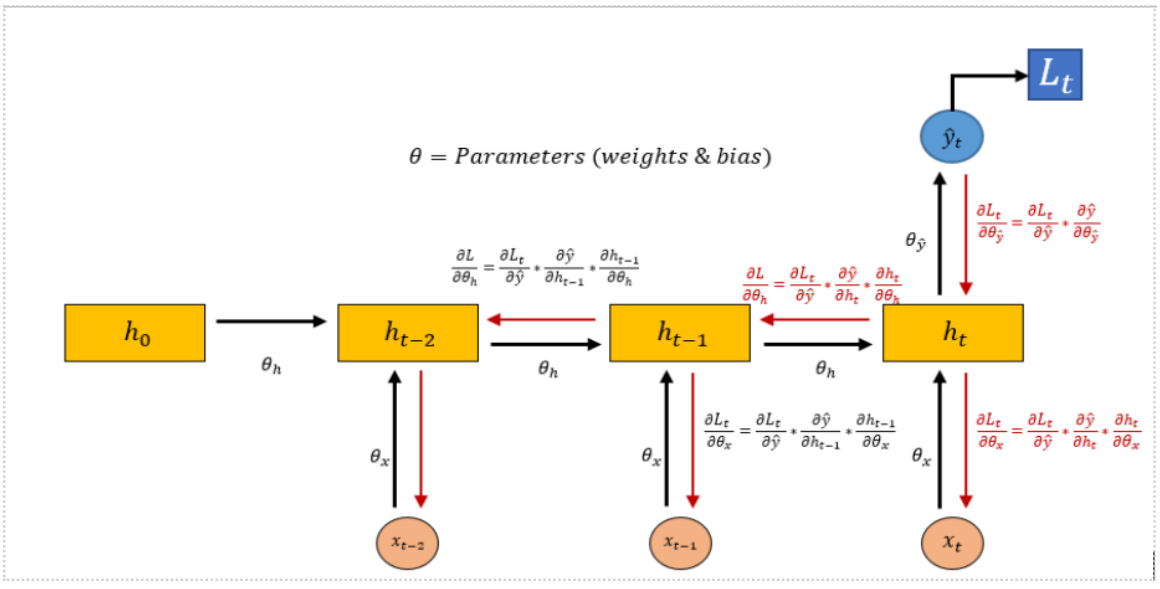


The above figure represents an unfolded RNN. X1, X2, X3 are the inputs at timestep t1, t2, t3 respectively, and Wx is the weight matrix associated with the inputs. Y1, Y2, Y3 are the outputs at timestep t1, t2, t3 respectively, and Wy is the weight matrix associated with the outputs. S1, S2, S3 are the hidden states at timestep t1, t2, t3 respectively, and Ws is the weight matrix associated with the hidden states. For any timestep, t, we have the following two equations:

$$S_t = g_1(W_x x_t + W_s S_{t-1})$$
$$Y_t = g_2(W_Y S_t)$$

where g1 and g2 are activation functions.
And let the error function be :

$$E_t = (d_t - Y_t)^2$$



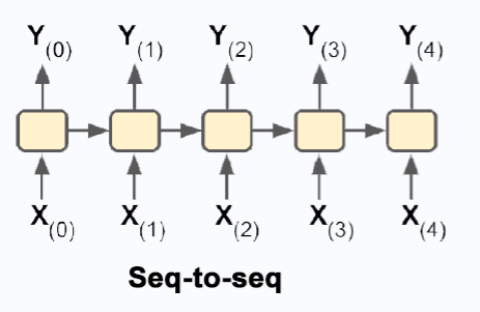
4 Training through RNN

1. A single time step of the input is provided to the network.
2. Then calculate its current state using set of current input and the previous state.
3. The current ht becomes ht-1 for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

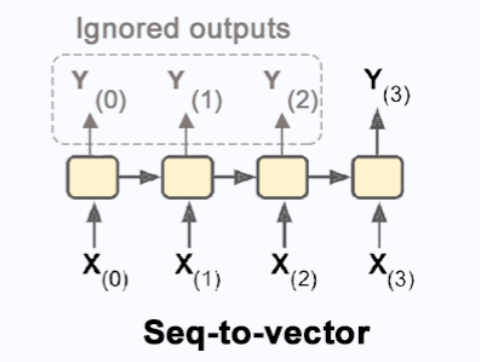
5 Types of RNN

The various types of RNN are:

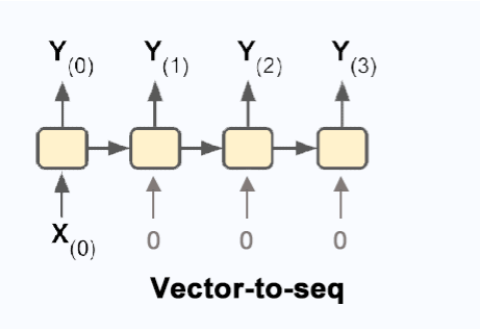
1. **Sequence-To-Sequence Network:**
   This form of sequence-to-sequence network is useful for predicting time collection which includes stock prices: you feed it the costs during the last N days, and it ought to output the fees shifted by means of sooner or later into the future.



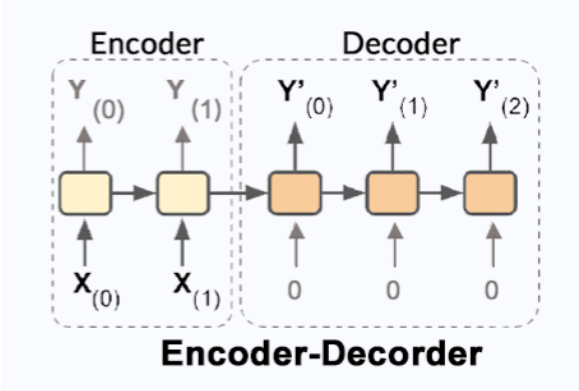
2. **Sequence-To-Vector Network:**
   You may feed the network a series of inputs and forget about all outputs besides for the final one, words, that is a sequence-to-vector network.



3. **Vector-To-Sequence Network:**
   You could feed the network the equal input vector again and again once more at whenever step and allow it to output a sequence, that is a vector-to-sequence network.



4. **Encoder-Decoder Network:**
   You can have a sequence-to-vector network, referred to as an encoder, followed by a vector-to-sequence network, called a decoder.

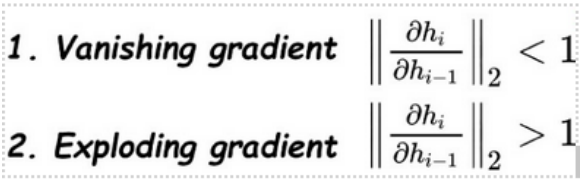


6 Limitations of RNN

With conventional BPTT applied on RNNs, we find partial derivatives by traversing the hidden states from the recent timestep to the initial timestep. When the input sequence is long, the number of timesteps increase and the hidden states go through continuous matrix multiplications in order to compute their derivatives. Hence, error signals (gradients) flowing backwards in time tend to either blow up or vanish as the temporal evolution of the backpropagated error exponentially depends on the size of the weights.

1. **Vanishing gradient problem:** The gradients carry information used for updating parameters of RNN and when the gradient becomes smaller and smaller, the parameter updates become insignificant which means no real learning is done. In this case, the gradient term goes to zero exponentially fast, which makes it difficult to learn some long period dependencies. Hence, learning to bridge long time lags in the input sequence takes a prohibitive amount of time, or does not work at all.

2. **Exploding gradient problem:** In this case, derivatives are large and the gradient will increase exponentially as we propagate backwards in time the model until they eventually go to infinity. The accumulation of large derivatives results in the model being very unstable and incapable of effective learning. Values the weights become so large that it causes overflow resulting in oscillating values or even NaN which can no longer be updated.



Due to the above mentioned problems of BPTT on RNNs, learning to store information over extended time intervals (long input sequences or gaps between relevant information) via recurrent backpropagation takes a very long time. Even if we assume that the parameters are such that the recurrent network is stable (can store memories, with gradients not exploding), the difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions involving the multiplication of many Jacobians (matrix of derivatives) compared to short-term ones. Recurrent networks involve the composition of the same function multiple times, once per time step. These compositions can result in extremely nonlinear behavior typically with most of the values associated with a tiny derivative, some values with a large derivative, and many alternations between increasing and decreasing.

Example:

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the sky," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information. But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France ..... I speak fluent French." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large. Unfortunately, as that gap grows, RNNs become unable to learn to connect previous information to the present task

7 Long Short Term Memory(LSTM):

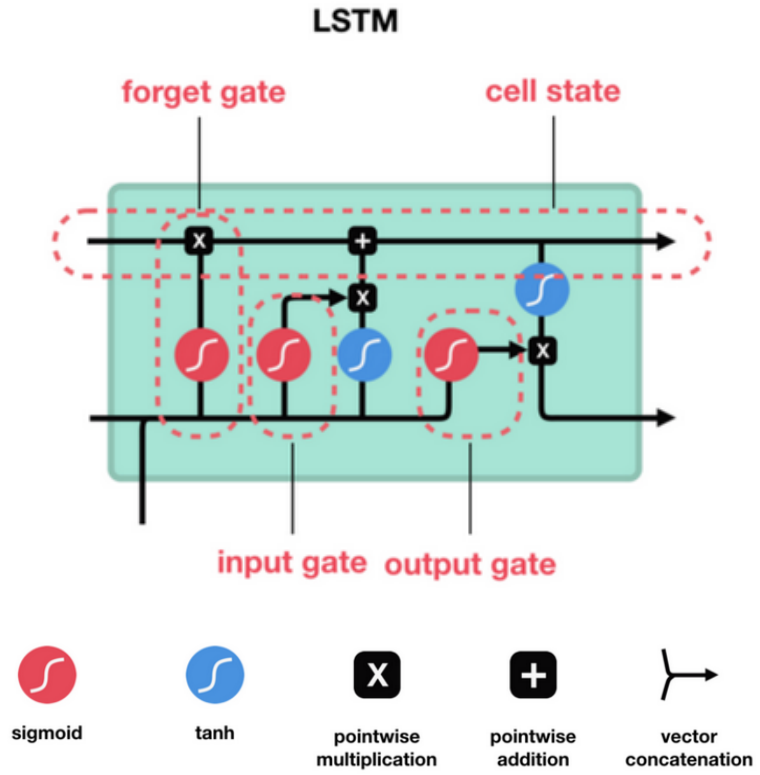
An RNN remembers each and every information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called **Long Short Term Memory or LSTM**.

The LSTM has been found extremely successful in many applications, such as unconstrained handwriting recognition, speech recognition, handwriting generation, machine translation, image captioning and parsing.

The key to LSTMs is its complex hidden unit, called memory cell, which has the ability to remove or add information to the cell state, carefully regulated by structures called gates. By making the weight of this self-loop gated (controlled by another hidden unit), the time scale of integration can be changed dynamically.

Each LSTM module may have three gates named as forget gate, input gate, output gate.

- **Forget Gate:** This gate makes a decision which facts to be disregarded from the cellular in that unique timestamp. it's far determined via the sigmoid function.
- **Input gate:** makes a decision how plenty of this unit is introduced to the current state. The sigmoid function makes a decision which values to permit through 0,1. and Tanh function gives weightage to the values which might be handed figuring out their level of importance ranging from-1 to at least one.



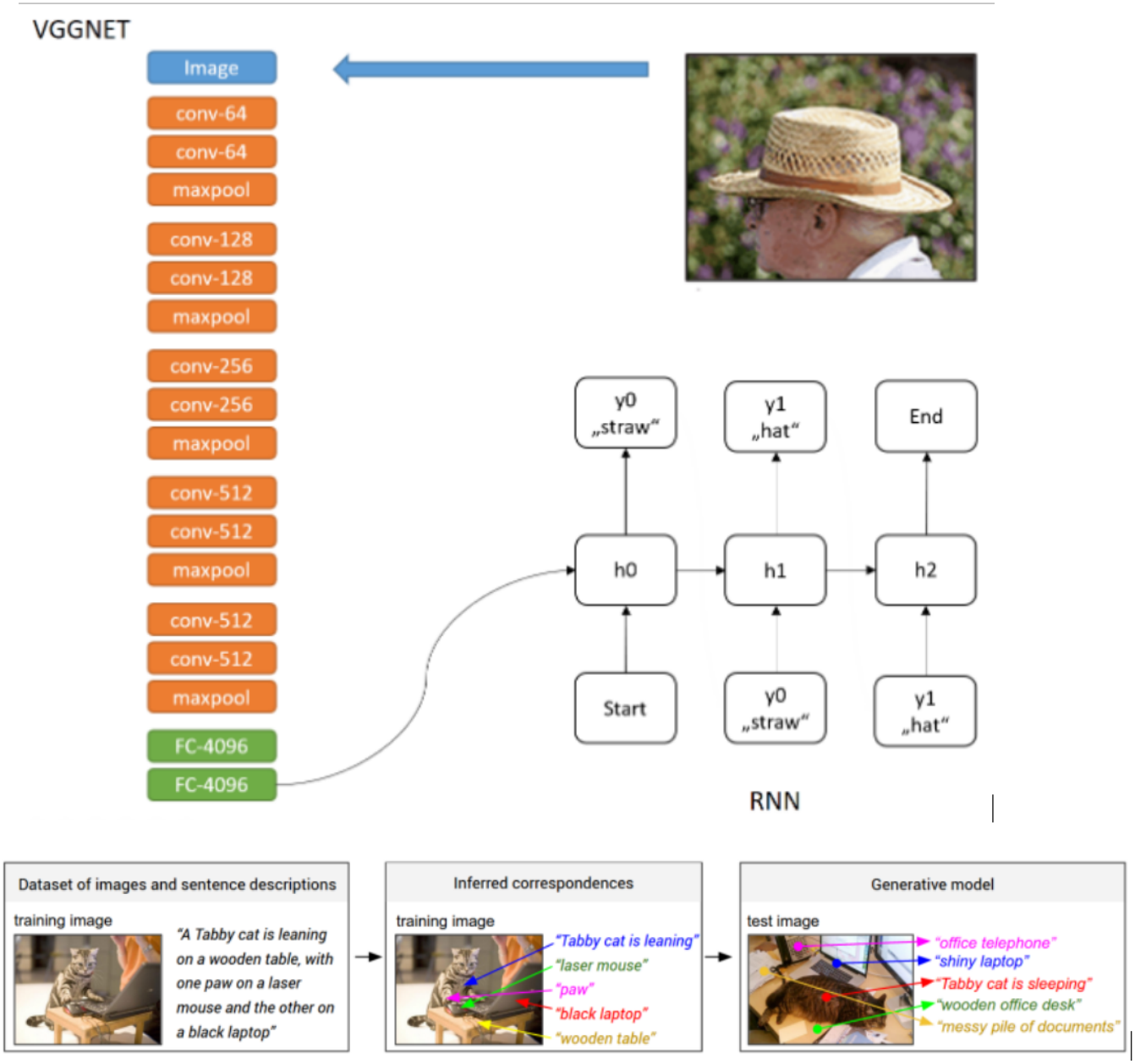
- **Output Gate:** comes to a decision which a part of the current cell makes it to the output. Sigmoid characteristic decides which values to permit thru zero,1. and Tanh characteristic gives weightage to the values which can be exceeded determining their degree of importance ranging from-1 to at least one and expanded with an output of Sigmoid.

8 RNN with CNN:

Recurrent neural network are even used with convolutional layers to extend the effective pixel neighborhood. They can be combined in two different ways described as follows:

1. **RNN after CNN:**

   The alignment model described is a CNN over image region combined with a bidirectional RNN and afterwards a Multimodal RNN architecture, which uses the input of the previous net. This multimodal RNN finally generates descriptions of image regions. As shown in the figure, the first modal is represented as the left VGGNET and the second module is shown on the bottom right as the RNN. The proposed model is trained on a dataset of images and their corresponding sentence descriptions given as input to the model. The first model (CNN) learns to align sentence snippets to the visual image regions by detecting objects in the image. This CNN is a VGGnet pretrained on ImageNet dataset with the only difference, that the last 2 fully connected layers are replaced by a Bidirectional RNN (BRNN) to compute the word representations. After aligning the data, the output of this first model is fed to the Multimodal RNN which has a typical hidden layer of 512 neurons. This network is trained to combine a word and the previous context to predict a new word as shown in the figure.

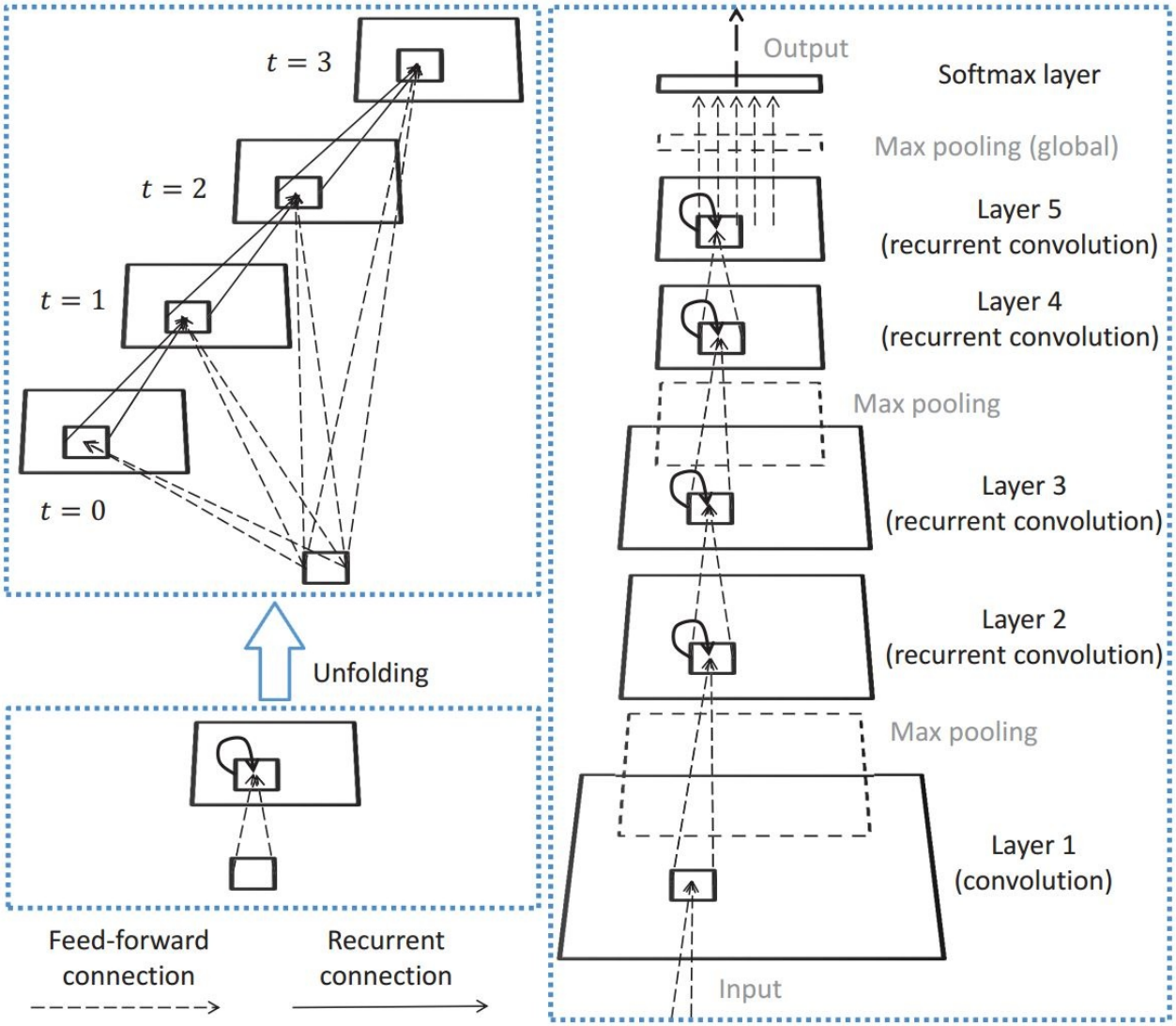


2. **Mixed CNN and RNN :** In a mixed CNN and RNN architecture the positive features of a RNN are used to improve the CNN.
The idea is inspired by the fact that CNN is typically a feed-forward architecture while in the visual

system recurrent connections are abundant, as understanding the context is important for object

recognition. As shown in the figure, without the context (face, neighboring objects in general), it is hard to recognize the black curve in the middle area as a nose. Hence, they have incorporated recurrent connections into each convolutional layer. The activities of RCNN units evolve over time so that the activity of each unit is modulated by the activities of its neighboring units. This property enhances the ability of the model to integrate the context information, which is important for object recognition. Like other recurrent neural networks, unfolding the RCNN through time can result in an arbitrarily deep network with a fixed number of parameters. Furthermore, the unfolded network has multiple paths, which can facilitate the learning process.

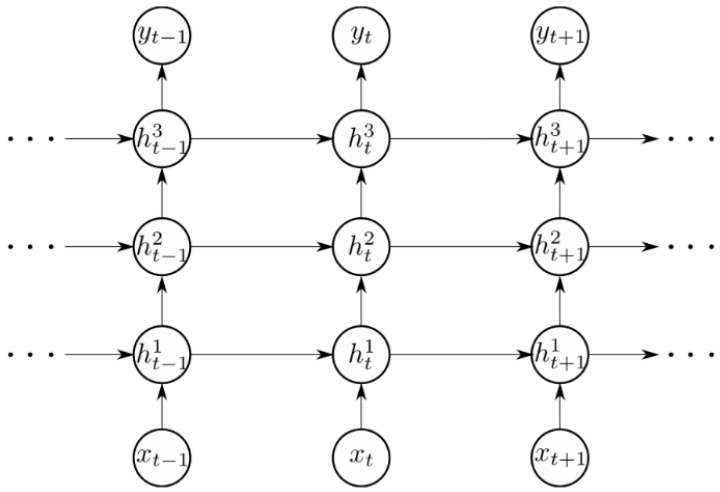
The overall architecture of RCNN is shown in the figure. On the left, the recurrent convolutional



layer (RCL) is unfolded for T=3 time steps, leading to a feed-forward subnetwork with the largest depth of 4 and the smallest depth of 1. At t = 0 only feed ... forward computation takes place. Image on right shows that the RCNN used in the paper contains one convolutional layer, four RCLs, three max pooling layers and one softmax layer. The network is trained with BPTT and therefore with the same unfolding algorithm described earlier. The recurrent connections increased the depth of the original CNN while kept the number of parameters constant by weight sharing between layers. The authors show in their paper that the recurrent connections perform better than a CNN with the same number of layers.

9 Stacked RNN :

the main reason for stacking LSTM is to allow for greater model complexity. In case of a simple feedforward net we stack layers to create a hierarchical feature representation of the input data to then use for some machine learning task. The same applies for stacked LSTM's. At every time



(a) Conventional stacked RNN

step an LSTM, besides the recurrent input. If the input is already the result from an LSTM layer (or a feedforward layer) then the current LSTM can create a more complex feature representation of the current input.

Now the difference between having a feedforward layer between the feature input and the LSTM layer and having another LSTM layer is that a feed forward layer (say a fully connected layer) does not receive feedback from its previous time step and thus can not account for certain patterns. Having an LSTM in stead (e.g. using a stacked LSTM representation) more complex input patterns can be described at every layer.

10 Applications of Recurrent Neural Network

RNNs are widely used in the following domains/ applications:

1. **Prediction problems:** RNNs are generally useful in working with sequence prediction problems. Sequence prediction problems come in many forms and are best described by the types of inputs and outputs it supports.

   Sequence prediction problems include:

   - **One-to-Many:** In this type of problem, an observation is mapped as input to a sequence with multiple steps as an output.
   - **Many-to-One:** Here a sequence of multiple steps as input are mapped to a class or quantity prediction.
   - **Many-to-Many:** A sequence of multiple steps as input are mapped to a sequence with multiple steps as output.The Many-to-Many problem is often referred to as sequence-to-sequence, or seq2seq .

2. **Language Modelling and Generating Text:** Taking a sequence of words as input, we try to predict the possibility of the next word. This can be considered to be one of the most useful

approaches for translation since the most likely sentence would be the one that is correct.In this method, the probability of the output of a particular time-step is used to sample the words in the next iteration.

3. **Machine Translation:** RNNs in one form or the other can be used for translating text from one language to other . Almost all of the Translation systems being used today use some advanced version of a RNN. The input can be the source language and the output will be in the target language which the user wants.

   Currently one of the most popular and prominent machine translation application is Google Translate.There are even numerous custom recurrent neural network applications used to refine and confine content by various platforms. ECommerce platforms like Flipkart, Amazon, and eBay make use of machine translation in many areas and it also helps with the efficiency of the search results.

4. **Speech Recognition:** RNNs can be used for predicting phonetic segments considering sound waves from a medium as an input source .The set of inputs consists of phoneme or acoustic signals from an audio which are processed in a proper manner and taken as inputs. The RNN network will compute the phonemes and then produce a phonetic segment along with the likelihood of output.The steps used in speech recognition are as follows:-

   - The input data is first processed and recognized through a neural network. The result consists of a varied collection of input sound waves.
   - The information contained in the sound wave is further classified by intent and through keuwords related to the query.
   - Then input sound waves are classified into phonetic segments and are pieced together into cohesive words using a RNN application. The output consists of a pattern of phonetic segments put together into a singular whole in a logical manner.

5. **Generating Image Descriptions:** A combination of CNNs and RNNs are used to provide a description of what exactly is happening inside an image. CNN does the segmentation part and RNN then uses the segmented data to recreate the description.

6. **Video Tagging:** RNNs can be used for video search where we can do image description of a video divided into numerous frames.

7. **Text Summarization:** This application can provide major help in summarizing content from literatures and customising them for delivery within applications which cannot support large volumes of text. For example, if a publisher wants to display the summary of one of his books on its backpage to help the readers get an idea of the content present within, Text Summarization would be helpful.

References

[1] https://en.wikipedia.org/wiki/Recurrent_neural_network

[2] https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

[3] https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

[4] https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e

[5] https://builtin.com/data-science/recurrent-neural-networks-and-lstm

[6] https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

[7] LSTM's original paper: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.4320&rep=rep1&type=pdf

[8] Fundamentals of RNN and LSTM: https://arxiv.org/pdf/1808.03314.pdf

[9] https://www.deeplearningbook.org/contents/rnn.html

[10] https://colah.github.io/posts/2015-08-Understanding-LSTMs/