Machine Learning (CS60050)
Spring, 2020-2021

Instructor: Aritra Hazra
Scribed by:
Murtaza Saifee (20CS60R61)
Nikhil Kumar (20CS60R62)

Thursday, 8^{th} April 2021

1 Recap :

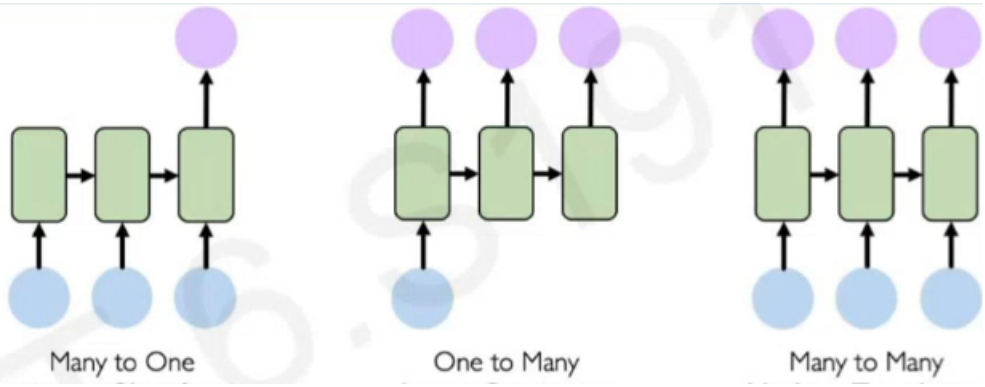
The already studied CNN model helps us targeting the spatial dependencies. Spatial dependencies refer to the nature of things being being correlated with an explainable clause. Thus a model with the hypothesis of such conversion of input X to an output y is stored and used to train the Neural Net.

But many a times the modelling has to be done in such a way that the information to be gained is present not in a particular instance but rather than in a co relation of the whole text. Simply meaning the context many times has a temporal dependency with the other context and thus deriving the meaning.

To target this we try to use a model called Recurrent Neural Network.

2 Recurrent Neural Network (RNN):

To target the temporal dependencies, the models can be by many virtue as broadly classified as :



Many to One : Dependency present in the input to help model desired output. This model can be co related to the part of categorization e.g. Sentiment Analysis

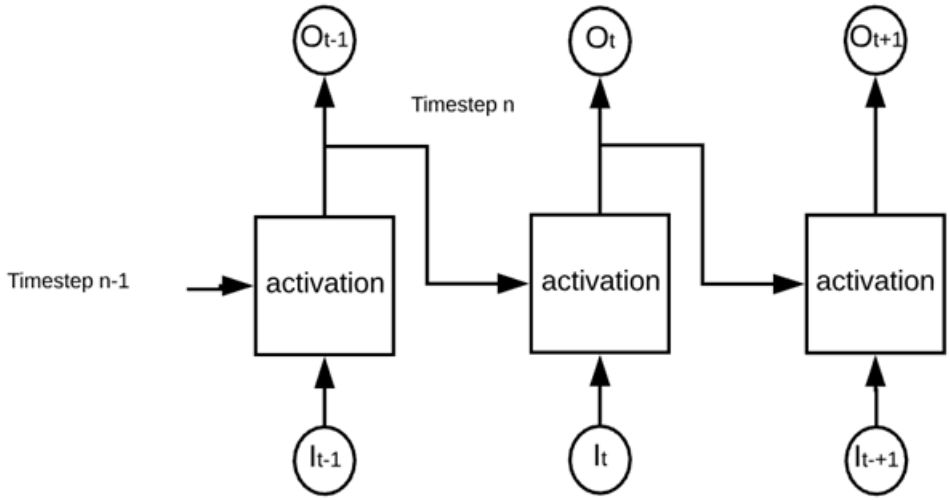
One to Many : Dependencies forming in the output generated to the co-relation with the single input. This type of neural network has a single input and multiple outputs. e.g. Image captioning

Many to Many : Way of modelling the whole input finding the dependency and converting it into suitable output. e.g. Language Translation

Moving on to the for building a RNN we try to unfold the neural network with respect to time to capture the temporal dependency.

So methematically the output at time t rather than just being dependent on the earlier case of input at time t will also have associativity with the captured information from time t-1.

Propogating this we at every timestamp the network takes in input of time stamp t and also the hypothesis output from time t-1, to generate output at time t.



2.1 Mathematical Modelling :

Formula for calculating current state:

Any current state predefined depends on the input and current state and output from last state. Thus

$$h_t = f(h_{t-1}, x_t) \tag{1}$$

Formula for applying Activation function:

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{2}$$

where:

$W_{hh} \rightarrow$ weights corresponding to recurrent neuron
$W_{xh} \rightarrow$ weights corresponding at input

With this, formula for calculating output:

$$y_t = W_{hy}h_t \tag{3}$$

2.2 Issues Arising :

With the mathematical model we can clearly see how the contextual information is passed on through the net but this has it's own merit and demerits.

2.2.1 Exploding Gradient Problem

The gradients carry information used in the RNN. When the gradient accumulated to be too large, the explosion of the gradient happens with each iteration. Though this issue can many a times be tackled with normalization of the output before passing thus keeping in check. But with that the next issue arises.

2.2.2 Vanishing Gradient Problem

When the gradient becomes too small, the parameter updates become insignificant. This makes the learning of long data sequences difficult.

In other words if the gradient captured are small with each iteration it keeps on diminishing thus a dependency over long distance this value becomes indistinguishable.

2.3 Gradient Problem Solutions

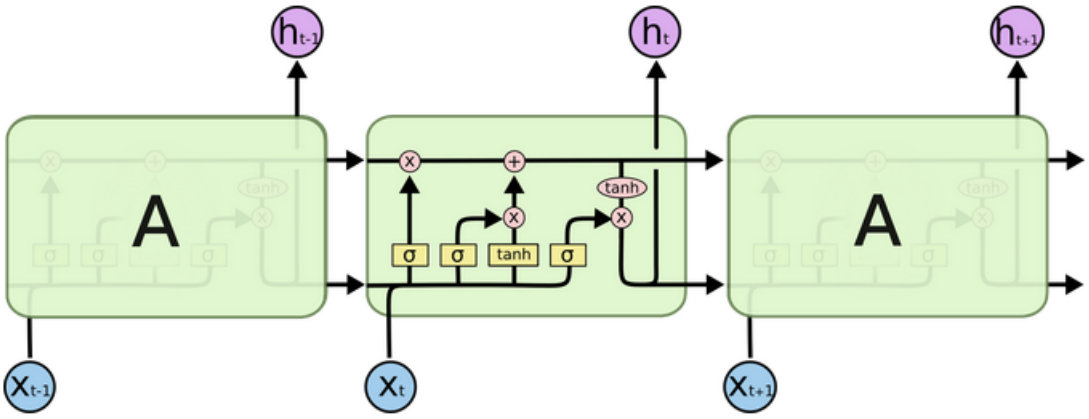
- Activation Function: The chosen activation function can be taken in accordance with the need and how the derivatives of the function behave.
- Identity Initialization: Rather than opting for a random initialization of weights it chosen to be identity to target the problem of vanishing or explosion gradient. But this too comes with its own limitations.
- Gated Cells : It simply say that from the context some is kept and propogated while the other part of context is dropped. This leads to not overloading the model.

The most suitable implementation of Gated cells in done in LSTM. Long Short Term Memory as the name suggests keeps some information retained and passed it along the network while dropping the other parts.

LSTMs also have a chain-like structure, but the repeating module is a bit different structure. Instead of having a single neural network layer, three interacting layers are communicating . These layers are :

- Forget Layer
- Store Layer
- Update Layer

The the output in terms calculated making it the fourth layer.

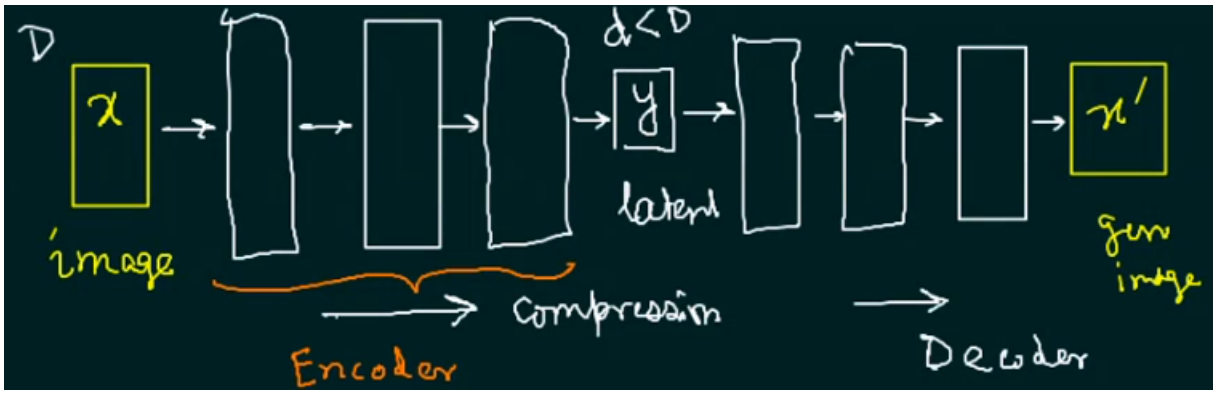


The repeating module in an LSTM contains four interacting layers.

3 Generative Models

3.1 Auto Encoder

The concept of Generative Modelling comes from the fact that, given an input x, we want to generate new input x'.
x' looks similar to distribution. It may be anomalous case or regular case as well.



The input x is passed to several neural networks(or, feature maps), then a internal latent representation(say, y) is provided, which has less dimension than x(d is less than D, where d is dimension of y and D is dimension of x).
The d dimensions are critical to express the image, like a compression.

The first set of neural networks is called encoder. Another set of neural networks will produce new images from the encoded part(y). This is called decoder.

The above discussed is popularly known as Auto Encoder.

Since we lose information while compression, how are we able to generate new image?
To generate new images, We use stochastic things like mean and variance.
We provide a mean and variance and then perturb mean and variance.
Eventually, something which is not present in the image will get generated. It puts in variation. This part is called Variational Auto Encoder.
We may use squared error to minimize loss.

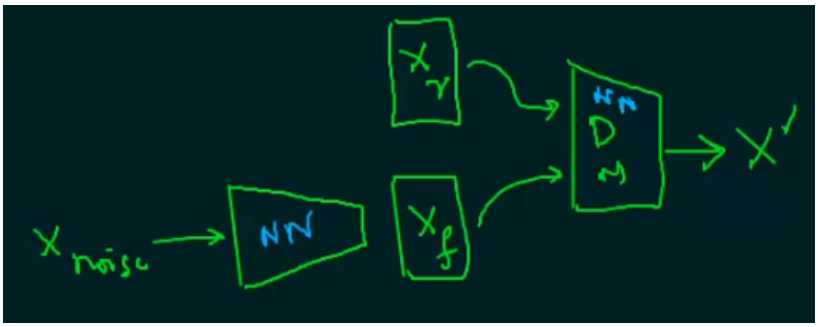
$$\alpha(x', x) = \frac{1}{2}|x - x'|^2 \tag{4}$$

For training, we need to minimize loss from the input side to the output side. We try to minimize the loss function and do other things like back-propagation.

In variational encoder, since it is probabilistic distribution of gaussian of mean and variance,We can perturb it a bit and get a new gaussian distribution and generate new images.

3.2 Generative Adversarial Network

We start with the noise(X_{noise}) and with that noise we go on to train the neural network. We generate just a fake image(X_f).



We have real image(X_r). There is discriminative model, which is like a game between real and fake image, which to update the fake image and try to get a new image which looks like the real image.

Intuition: We have real points and we generate three fake points initially at random. We place the fake points along the real points. We keep on training

over these examples. After an epoch, we can say which point has higher probability to be general.
The probability of real points being general keeps on increasing after over the course of training.

We now generate new points closer to the realm of real points. When we can't distinguish between real and fake points, then we stop.
This is called as Generative Adversarial Network.
Adversarial network takes some images and generate new images.

To generate using generative adversarial network, the sample needs to be large, otherwise the variations would be abrupt. We will gradually move the fake points near the real points. If we have very few real points, then the gradual movement towards the real boundary is not so well defined. People might catch that it is a fake image.

Can we use generative adversarial networks to generate more training data?

If we have less data to begin with, we would not be able to generate fake images in the first place. If fake is generated from data which is not dense, then we can easily catch it is fake. The generated data then may have bias.

Deepfakes are one of the examples of generative adversarial network. With Deepfake, we can create anything and make a social outrage.

Therefore, the topics, like fairness and biasness in AI, are new booming topics, in which we can learn about how we can prevent it from misusing, tell that the content generate is fake.

Contributions:
Murtaza Saifee: Recap, RNN (Modelling, Issues, and Solutions)
Nikhil Kumar: Generative Models(Auto Encoders, Generative Adversarial Networks)