How human mind processes the images?

* Light from retina goes through optical nerves (cells) to the thallimius and from there to the V1. Electrochemical single is the method through which light moves thorough the nerves.
* Main takeaway is different cells process different aspects

CNN: Architecture is as such where you have Image arrays as input and then you apply filters on it and do the convolutional process. (with padding, strides or without them). And then add pooling to address few issues raised such as

* Reduced array shape
* More impact to the centered pixels
* Overfitting
* Computational efficiency.
* Memory issue
* And translational invariance.

How does the convolutional operation happen:

* 2D: A black and white image is applied with the filter and gets you reduced 2D array with cross multiplying each corresponding element and then adding all the products (dot product).
* For 3D you do the same but with 3D filter but the output you get is 2D array only after every entire operation. Cause the 3D convolutional operation happens with the filter having the same depth as your input and you don’t stride/move your filter backwards but you only move them in 2D and i.e. you get 2D array output as a **FEATURE MAP.** Size of the feature map is always a **2D array X no.of filters.**

Trainable parameters: What you calculate weights and biases rights. (Keep the same principle here as well):kk

What could be the trainable parameter?

Feature maps does look like a trainable parameter but it is not is it?, Those arrays are something similar to the output the ANN. Not something but kind of exactly like the output of the ANN

And then what is similar to the weights?

* Filters are what we want to create and developed for a particular image cause those are the arrays or rather values which are getting applied on our input and producing output right. So, we have to decide those values and those exact values are nothing but the comparatives of the weights.
* And the biases of each filter

So, trainable parameters are size of the ( filter array \* No. of. Filters ) + biases.

Main thing about the CNN that trainable parameters do not depends upon input parameters at all which was the case in ANN that was creating the high computational complexities and memorization error.

Padding and Strides :

Pooling :

* Pooling solves two problems
* 1. Memory Issue:
* 2. Translation Invariance ( Only for minor translation chanes)

RNN: Recurrent Neural Network

How is the input given to the RNN, now as we have seen that we cannot give zero padded input like even if we try ANN.

So input is given to the RNN in timestamp, and for each record (row) the no. of timestamp is as equal to the no. of words in the record.

First of all let’s understand how do we encode the data?

Record : My name is BMF.

The ultra ultra strong MF.

No. of unique words = 8

my = [1,0,0,0,0,0,0,0]

name = [0,1,0,0,0,0,0,0]

is = [0,0,1,0,0,0,0,0]

and likewise…….

Now the record would be represented as 2D array of [4 X 8] each row is word of 1 X 8.

And the final input to the RNN would be a 3D tensor of [5 X 8 X 2].’

5 for max no. of word in all the records, that means whenever we have a word less than the max of words in a row there will be a empty row for that record.

And third dimension is no. of records.

This is how the tensor is given as the input to the RNN.

Now let’s see how each input is feed as a timestamp to the RNN.

For row one, there would be 4 timestamp and first would word “My” with it’s encoding and let’s say we’ve RNN architecture as such as

input timestamp[ for word “My” 8 X 1] followed by 3 RNN nodes and followed by 1 fully connected node.

So when for the first timestamp the “My’ enters with 8 inputs of [1,0,0,0,0,0,0,0] and connects to the RNN layer nodes.

Extra thing here is the output of the first time timestamp [01] will be fed back to the nodes.

And then the processed value is send to the next node.

Before the formulae of forward propagation let’s calculate the no. of trainable prameters, here

W1 = 8 X 3

W1R = 3 X 3

W2 = 3

Biase1 = 3

Biase2 = 1

Total = 40

Formula is

(W1 . X11) + (W1R . O1) + b1

W1 = [3 X 8] . X11=[8 X 1] + W1R=[3 X 3] . [3 X 1] + [3 X 1] = [3 X 1]

One thing for the first RNN layer as we don’t have any of the outputs, we select either zero-zero or random outputs. And this is how we calculate forward propagation.

How do

Architecture Diagram?

In RNN we have hidden recurrent layer which gives back the output of the first input as a second input along with weight for second timestamp.

Backpropagation is very similar to the ANN, nothing sort of complicated.

Encoding techniques :

OHE we represent a word by a vector and sends the 3D tensor as a input to the RNN.

Where as in

**Types of RNN:**

**Issues in RNNs :**

**There are majorly two issues in RNN’s:**

1. **Vanishing gradient descent. Or inability to capture longer term dependencies.**
2. **Exploding gradient problem.**
3. **Vanishing gradient: It is the main problem lying under the inability to capture the long term dependencies.**

* **Usually for longer timestamps the when you update the weight by gradient descent those values are interdependent on all the previous timestamp derivative values. In the above scenario if you have if you the derivative values to in between 0-1 and tanh is always in between 0-1 then the gradient comes to be very small and that implies the impact of that timestamp would be very less.**
* **Solution:**
* **Activation (relu, leaky relu)**
* **Bette weight unitization**
* **LSTM**

1. **Exploding gradient: It is the main problem where the gradient values are way more consistently and therefore those values comes out be very high for later stamps.**

* **What is the solution:**
* **1. LSTM**
* **Tanh**
* **Clipping**

**LSTM’s Core Idea:**

**LTM,STM and interaction between both.**

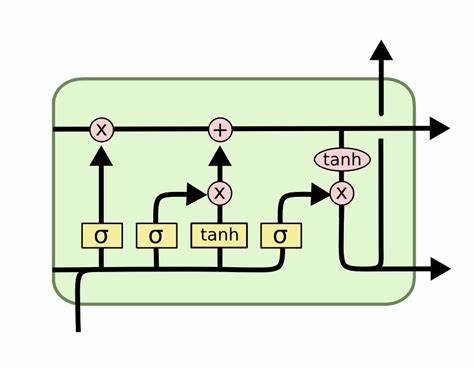
As the main reason that RNN’s can not keep track of the old context, we have to make sure somehow that RNN’s keep track of the longer-term dependencies. That is exactly what is done in LSTM’s. So, what does LSTM’s do is, along with short term dependencies like in normal RNN’s they also keep track of the long-term dependencies. So, in LSTM we have one **LTM** (Longer term memory) which track the long term meaning/memory/context and **STM** (short term memory) tracks the shorter term memory.

So now in LSTM instead of one STM recurrent state we have STM as well as LTM inputs feedback to the nodes of the layers.

In LSTM we have three gates and two operations:

Operation 1: Forget things from cell state and add things in cell state.

Operation 2: Calculates the hidden state



1. Forget gate.
2. Input gate.
3. Output gate.

Cell state is LTM, and hidden state is STM:

So, every state we either forgot something from the LTM or add something to the Cell state (LTM),

Whereas updating the STM.

**How does the everything looks like mathematically in LSTM .**

Cell state, hidden state looks same where they are vectors of both same size. **(Ct, Ht).**

**Like the above two there are four more vectors like Ft, It, Ct, Ot and interestingly they are of same shape as Ct, Ht. i.e. means these six vectors are of same shape**

**Xt –** input vector ( can be of any length not restricted to any).

**Pointwise operations :**

Arrays pointwise operations like (+ , X)

**Nodes:** Each gate array is neural layers with activation function. It is a hyperparameter where the no. of nodes are decided and they are same for all the layers. The no. of nodes are same as the no. of elements in those arrays like LTM, STM, Ft etc.

LTM and STM are of same shape at both the states but the input Xt can be different.

**Forget GATE:**

Let’s suppose we have [1 X 4] input in Xt with 3 nodes in forget gate layer, in that case

The total input to the forget fate layer would be (Xt and ht) right which is called concatenation input, in this case that would be (4 + 3) which would be [1 X 7]

And weight matrix would be (3 X 7) as there are 3 nodes in the layer.

So the output of the forget gate would be:

= (3 X 7)

= (3 X1)

= (4 X 1)

Then the output of this operation would have point wise operation with Ct. So, basically what is happening is we have previous hidden state on processing that hidden state with input of current state we get Ft which is nothing but what to remove from the cell state and that is why we do point wise operation between the Ft which now knows what to remove from the cell state.

Let’s suppose our cell state is [ 5,10,20] and our Ft comes out be [0.5,0.25,0.1] then current cell state after removal would be [2.5,2.5,2] and this is what happens in the forget gate.

**Output GATE :**

**On magnifying the gate we we’ll two layers in output gate first one is input layer with It array and other one is candidate cell state.**

**Ct is nothing but the potential information to be added in the cell state where as It decides what and which of that potential info to actually add. It has sigmoid loss function where as Ct has tanh.**

**The output equations looks the same as forget gate with one change in error function.**

**For candidate cell state it looks like:**

Though the product of weight and input state is same but the values are different.

And similarly for **It:**

On calculating the It and Ct we have multiplicative point wise operation which decides the which information out of the potential info is needed to be added in the cell state.

This quantity ( Ct X It ) which is the actual filtered info to be added into the cell state is added in the cell state with point wise addition operation.

So, ultimately LSTM has the ability to add and maintain the whichever pervious info with the help of this gates. So, the main problem in the RNN which was vanishing gradient or loosing info over the longer distance can be easily fixed in the above case with Ft being one that is nothing is removed from the cell state and the product of Ct and it is zero then the actual cell state is maintained and carried throughout without loosing the info.

**Output Gate :**

**Output gate calculates the hidden state from the CELL STATE.**

**The output hidden layer has equal number of neurons as all the hidden layers with activation function as sigmoid.**

**Output array is calculated form the hidden state input ht and Xt concatenation with its weight matrix and the multiplicative point wise operation is done with the cell state array after applyting tanh activation function. And that gives you the new cell state which is basically calculated from the output cell.**

**The equations are exactly the same there is not much change as such. Just the direction is different.**