

Lecture 8 : Convolutional Neural Networks

* Encoding data from Images (Slide - 4- 16, 17)

- Since, here we are only dealing with black and white images (not grayscale) there can only be two values for a pixel: 0 & 1 for B&W.
- The simplest way to convert image to data is you convert all the pixels into features. (like given in the slide)
↳ 1 single image essentially becomes 1 single data point.
(or numbering)
- Note:- The ordering of the features in the image may or may not be important. It depends on what or how you're gonna process & learn the image.

* Previous Neural Nets

(& Should we use them on images?)

(Slide- 5- 29, 31)

- In the previously seen neural nets, the last layer was the deterministic one & previous layers could be called 'hidden' layers.
- The last layer must be fully connected in any case to get the results.

The 1st layer was fully connected, i.e. all inputs or you could say all features were sent to all the nodes of the layer. We had to learn the parameters of each node on the same data point. (Would that work or would that be redundant when dealing with images?)

Images are different than regular data. They have some special characteristics like spatial locality & translational invariance.

Spatial locality:- Take an example, where you wanna detect a stop sign. Now you know that the stop sign pixels aren't gonna be scattered throughout the image. They are gonna be clustered together. This is spatial locality.

Translational invariance:- The stop sign will be stop sign no matter where it is in the image. If's pixels may be clustered in the corners or of image or centre of image or anywhere else in the image. It would still mean the same thing.

(You could say that on the order of only some features may be important in the data) You wanna be able to locate a stop sign wherever it is in the image.

So, you could maybe see that fully connected features may cause a problem because they won't be able to encode or these characteristics or detect leverage these characteristics of these images.

So, you wanna be able to identify pass & analyse & train parameters on chunks of data. Or you wanna know what's happening in what chunks of data, you use convolutional layer!

* Convolutional Layer (Slide - 6 - 48, 90, 100) (1-D example)

- 1-D image may also be called a signal.
For example:- You have pointed the telescope at a point in the sky & you wanna may be detect a light that indicates / implies a satellite, star, etc. Then the 1-pixel image may be expressed as a variable w.r.t. time & you arrange these pixels in a horizontal 1-D image / signal for a certain range of time.
- You first apply a filter and the result of the filter is the convolutional layer new features layer. As you see, the filter analyses chunks of data or sequences of data & turns them into new features.
For example, if you apply a $\begin{pmatrix} 1 & -1 & -1 \end{pmatrix}$ filter on a binary sequence, you will get the degree of isolation of 1s & 0s as new features. For ex, a sequence of $0, 0, 1, 0$ will give 1 which indicates the 1 'is lonely' & sequence $1, 0, 1$ will give -2 which indicates 0 is 'lonely' in the sequence.
- But if you notice, in a filter like this, we are focusing on the middle number of a sequence, ~~of 3~~, ~~over~~ 3 pixels. So, this will tell about the degree of isolation of the second pixel in every sequence. But, when you apply this on a signal, you won't get anything on the 1st & last pixel of the signal because they ~~won't ever~~ be the 2nd pixel of a sequence.

- And, if you have a signal of length m & you apply a filter of length n on it, the resulting new features vector will be only of size $m-n+1$.
- To solve the problems given above, (i.e. reduced size of feature vector & loss of data on 1st & last pixels) we introduce padding.
- You can also add bias to the dot product by literally adding a value to ~~# in the~~ the dot product. The contents of the filter are supposed to be chosen by you depending on how you wanna analyze the raw data. The contents of the filter ~~are~~ and bias are analogous to W & b , and are called 'weights'

* Convolutional Layer (Slide - 138, 143, 154)
(2-D example)

- In 2-D & 3-D and ~~in~~ any dimensions, you do element wise multiplication of filter & sequence matrices & not matrix multiplication. You take sum of resulting elements (i.e. after multiplication)
- In this 2-D example too, for a filter of $[-1, -1, -1], [-1, 1, -1], [-1, -1, -1]$ it gives a 'lonely 1' from the original image.

* Convolutional layer: (Slide - 162, 172,
(3-D example)

- Till now, we only looked at 2-D images with only black and white colours (~~not gray~~ not to be mistaken as grayscale)
- 3-D images are just 2-D images, but when representing them as data, we represent their colour as 3rd dimension.
- For example, a single pixel may have 3 values: Red, Green & Blue values. Image may be represented as a $3 \times \text{height} \times \text{width}$ image as data.

★ Max pooling layer (Slide - 187, 189, 214, 217)

- Max pooling layer is also like a filter without the weights & dot product & bias.
- It basically tells, "do we detect something & what?" after convolutional layer / filter.
- It returns max of its arguments.
- If you're using a max pool of size 3 (i.e. 3×3) with stride 1, you basically skip 1 column of pixel while moving horizontally & 1 row of pixels when moving vertically. For stride 3, you skip 3 columns of pixels horizontally moving & 3 rows of pixels moving vertically.
- Note:- You can use different strides (strides other than 1) for filters too.

★(NNs: a taste of Backpropagation) (slide- 261)

- Consider an example where you want to do regression for a 5×1 image / signal & 3×1 padding filter & appropriate padding.
- The ~~filter~~ pre-activation vector $Z^{(1)}$ given obtained as a result of filter $W^{(1)}$ applied on input X . $Z^{(1)}$ will be of size 5×1
$$\& Z_i^{(1)} = (W^{(1)})^T \cdot X_{[i-1, i+1]}$$
- We use the activation function ReLU on $Z^{(1)}$ to get the activation $A^{(1)}$
- Now, for the output of the final regression layer,
$$A^{(2)} = (W^{(2)})^T \cdot A^{(1)}$$
- We use squared error loss, where the loss is the function of $A^{(2)}$
- So, to use SGD & calculating gradient, we back propagate in the chain rule to get the gradient / derivative.

Note:- The matrix derivatives is tricky part & learn linear algebra to understand it.