Chapter-4 Convolutional Neural Networks

4.1 Categorical Cross Entropy

Multiclass classification with NN:

For binary classification problems, we have a final layer with a single node & a sigmoid activation

This has many desirable properties:-

- Gives an output strictly between 0 & 1
- can be interopted as a probability
- Derivative is nice
- Analogous to logistic regression.

Is these a natural extension of this to a multiclass setting?

Reminder:- one hot encoding for categories.

1. Take a vector with length equal to no. of categories
2. Represent each category with one (1) at a particular position & zero (0) everywhere else

---

For multiclass classification problems, let final layers be a vector with length equal to the no. of possible classes

Extension of sigmoid to multiclass is Softmax Function

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

Yields a vector with entries that are b/w 0 & 1 & sum to 1

For loss function use "categorical cross entropy".

This is just the log-loss fn in disguise:-

$$C.E = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

Derivative has a nice property when used with softmax

$$\frac{\partial C.E}{\partial \text{softmax}} \cdot \frac{\partial \text{softmax}}{\partial z_i} = \hat{y}_i - y_i$$

4.2 Intro to CNN:

4.2.1 Motivation - Image Data

-> So far the structure of our neural network treats all inputs interchangeably.

-> If we imagine an image & the way that an image works is that each of the different pixels will have a different numerical value to give you the density within the red, green, blue spectrum.

- No relationship b/w individual inputs
- Just an ordered set of variables
- we want to incorporate domain knowledge (How images are build) into the architecture of Neural network.

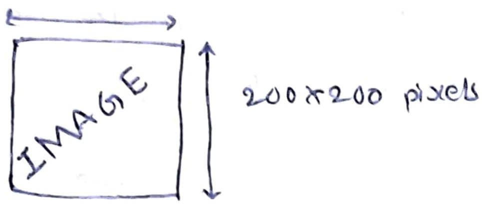
-> The convolutional networks ~~are~~ we discuss here were developed to deal with image data. Increasingly, these approaches are being applied in more common analytic problems of regressions & classification

-> Important structure in image data:

+ Topology of pixels
+ Translation invariance
+ Issues of lighting & contrast
+ Knowledge of human visual system
+ pixel tend to have similar values.
+ Edges & shapes.
+ Scale invariance.

4.3 Image Dataset:

Fully connected image networks thinking about the no. of pixels & image as that storing no. of features all being fully connected to the next layer would tend to require a vast no. of parameters.



Total pixels = 40000.

3-colors :- r, g, b

Total parameters + bias term = 120,001.

So imagine with that fully connected network we will have to start off with at least 120,001. And you can even imagine a single fully connected ~~node~~ layer would require this incredible amount of weights. So with these many weights that variance would be incredible high with very likely to overfitting scenario.

→ So we're going to introduce a bias & in this case a bias in relation to that fully connected network such that the archetecture will be adjusted to look for certain kinds of pattern. This new architecture is bcz of the different layers can learn certain intermediate featur

→ We can start off with edges which then build up into shapes which they can then be built into relations between different shapes even as well as identifying different textures within images.

4.4 Kernels:-

A kernel is a grid of weights overlaid on image centered on one pixel

- Each weight multiplied with pixel underneath it
  (number)
- Output over the centered pixels is $\sum_{p=1}^{P} W_p \cdot \text{pixel}_p$
  This is called convolutional operations

Used for traditional image processing techniques:-

- Blur, Sharpen, Edge detection, Emboss etc

Kernel 3x3 Example

Input

3	2	1
1	2	3
1	1	1

Kernel

-1	0	1
-2	0	2
-1	0	1

Output

	$\boxed{2}$	

Output will be only at one pixel as the kernel is overlaid on center pixel of Input.

$$\text{Output} = (3\times -1) + (2\times 0) + (1\times 1) + (1\times -2) + (2\times 0) + (3\times 2) + (1\times -1) + (1\times 0) + (1\times 1) = 2$$

Output = 2

Kernels as Feature Detectors:-

can think of kernels as a local feature detectors

-1	1	-1
-1	1	-1
-1	1	-1

vertical line detector

-1	-1	1
1	1	1
-1	-1	-1

Horizontal line detector

-1	-1	1
-1	1	1
-1	1	1

Corner detection

Convolutional Neural Nets:-

Primary ideas behind CNN:-

- Let the NN learn which kernals are most useful
- use same set of kernels across entire image
- Reduce number of parameter & variance (from bias-variance point of view)

4.5 Convolution for Colour Images:-

Most common way is 3 2D arrays, all stacked one on top of the other. (Red, blue, green scales)

Now to move our kernels to 3-D rather than using the convolutional operations using just this kernel that's three were going to use convolution on a filter, filter that terms once we move up to 3-D, which may be three by three by three

It's going to be three, three-by-three kernels all stacked together, so that instead of having nine multiplic- added together to get our one output, we add the sum 27 applications.

Nine → Red   Nine → Green   Nine → Blue

Total :- 27 multiplications.

When we work with these centered values, the edges of our image & the corners of our image tend to get overlooked. That's where the new concept of "padding" introduces.

4.6 Convolution Settings- Grid Size, Padding & Stride

① Grid Size :- (Height & width)

- The no. of pixel a kernel 'sees' at once
- Typically use odd numbers so that there's a center pixel
- Kernel doesn't need to be a square

② Padding :-

- Using kernels directly, there will be an edge effect
- Pixels near edges, will not be used as center pixel since they are not enough surrounding pixels
- Padding adds extra pixels around the frame, so pixels from the original image becomes center pixels as the kernel moves across this image

- Added pixels are typically of value zero. (zero padding)

Without Padding:

1	2	0	3	1
1	0	0	2	2
2	1	2	1	1
0	0	1	0	0
1	2	1	1	1

Input

-1	1	2
1	1	0
-1	-2	0

Kernel

-2		

Output

(Here output is smaller than original input)

We won't be able to center the top-left corner, or the 2 next & below it. So for that thing we use padding.

With Padding:

0	0	0	0	0	0	0
0	1	2	0	3	1	0
0	1	0	0	2	2	0
0	2	1	2	1	1	0
0	0	0	1	0	0	0
0	1	2	1	1	1	0
0	0	0	0	0	0	0

Input

-1	1	2
1	1	0
-1	-2	0

Kernel

-1				

Output

Here our output will be larger, i.e closer to the size of original image

3. Stride:

- The 'step size' as the kernel (image) moves across the image
- can be different for vertical & horizontal steps.
- when stride is greater than 1, it scales down the dimension as convolution operations are decreased.

The kernel square will be moved one by one to right until the last & comes back to the left most by-one-by-one step. & then it moves 1 down & repeats the same step.

Stride - No padding:

1	2	0	3	1
1	0	0	2	2
2	1	2	1	1
0	0	1	0	0
1	2	1	1	1

Input

-1	1	2
1	1	0
-1	-2	0.

Kernel

-2	3
0	

Output

Stride - Padding

0	0	0	0	0	0	0
0	1	2	0	3	1	0
0	1	0	0	2	2	0
0	2	1	2	1	1	0
0	0	0	1	0	0	0
0	1	2	1	1	1.	0
0	0	0	0	0	0	0

-1	1	2
-1	1	0
-1	2	0

Kernel

-2	2	
3.		

Output

4.7 Convolutional Settings - Depth & Pooling

④ Depth:-

→ In images, we often have multiples numbers associated with each pixel location

These numbers are reffered to as "channels".

- RGB image: 3 channels
- CMYK :- 4 channels

→ The no.of channels is refered to as the "depth".

So the kernel itself will have a "depth" the same size as the no.of input channels.

Example 5x5 kernel on RGB image

- There will be $5 \times 5 \times 3 = 75$ channels.

→ The output from the layer will also have a depth

- The networks typically train many different kernels
- Each kernel outputs a single ~~ta~~ number at each pixel location
- So, if you have 10 kernels in a layer, you will have the output of that layer will have depth = 10.

⑤ Pooling :-

Reduce the image size by mapping a patch of pixels to a single value

- Sinks the dimensions of image
- Does not have parameters, though there are different types of pooling operations

Max-Pool :- (common practice)

For each distinct ~~pool~~ patch, represent it by the maximum

Eg :- 2x2 maxpool :-

2	1	0	-1
-3	8	2	5
1	-1	3	4
0	1	1	-2

$\xrightarrow{\text{Maxpool}}$

8	5
1	4

Average pool :-

2	1	0	-1
-3	8	2	5
1	-1	8	4
0	1	1	-2

$\xrightarrow{\text{Avgpool}}$

2	1.5
0.25	1.5