L12 - CNNs pt III

BR: last lecture on CNNs (ensure you complete CNN material in DL)

(*) note the conventional CNN hierarchical story.

(*) General architec. of NN

- recall downsampling is optional
- general recap of convolutional layer
  - convolution
  - activation

(W): (A1): A little rusty on multiple maps element of CNNs + convolutive maps

- 11.3 essential → review.

(*) CNN - vector not.

(A2) - understand distinction in pseudocode between using stride = 1 and stride > 1.
- for CNN vect not.

- canshrink maps via — i) convolution with stride > 1
  \ ii) downsampling

(*) pooling / downsampling

- max pooling → typically stride > 1
- after convolutional layer (i.e. conv / affine and activation already applied) max pooling is applied

i) searching for index of largest value (see pseudocode!)
  - maxidx(...)

2) store value at that index location. max.

- performed independently for every map (max pooling)
- convolutions combine maps

(*) there are differences between this S.2019 and S.2020 slides
- Make sure you can reconcile

(*) Mean pooling → use different op.

- Image is RGB → stack ~~not~~ 3 layers → cube

(*) Each layer of CNN consists of several stacked filters → cuboid

(*) These filters scan input and produce a collection of planes (conv.)

(*) Ops which you perform pooling to reduce no. of planes.

- convolve - pool - c - p -

(*) Then flatten and pass {C-P-C-P} output through softmax / MLP.

final

(*) Learning network

- Parameters to be estimated :-
  - weights of neurons in final MLP
  - weights and biases (aka filters) for every conv. layer

- Recap finished

(*) Param est just like conventional MLP :-

but i) shared parameters
ii) scanning input

} giant MLP with shared params.

(*) Training examples

(*) Variant of gradient descent ; use backprop

(*) Defining loss

- Typically cross-entropy or KL divergence between network output, desired output

---

(*) Problem setup

Overall loss is average of divergences over training set

$$\text{Loss} = \frac{1}{T}\sum_{i=1}^{T} \text{div}(\underline{Y}_i, \underline{d}_i)$$

(*) Training CNNs through g.d.

(*) Need to update weights/biases for every layer; and for every filter

$$w(l,m,j,x,y) = w(l,m,j,x,y) - \eta \frac{d\,loss}{dw(l,m,j,x,y)}$$

(?) (A2) - l,m,j,x,y → clarify what these index (can't recall)

Total train loss: $Loss = \frac{1}{T}\sum_i Div(Y_i, d_i)$

Total deriv: $\frac{dLoss}{dw(l,m,j,x,y)} = \frac{1}{T}\sum_i \frac{dDiv(Y_i, d_i)}{dw(l,m,j,x,y)}$

(*) Backprop kicks in here

(*) Backprop - final flat layers

(*) Backprop does not differ from MLP up to flattened layer; only receding past flattened layer do things change.

(*) Recall :- Derivatives wrt — i) weights, biases (as a consequence)
ii) output of previous layer (recursively)

(*) Final layer - $Y(\underline{X})$
Derivative of div wrt final layer = $\nabla_{Y(\underline{X})} Div(Y(\underline{X}), d)$

(*) Backprop - convol. and pooling

(*) Special treatment of:-
i) shared computation in conv. layers
ii) pooling layers

(*) B.P. - conv layer

(*) consider what is happening during forward propagation through conv. layer
(*) and; derivative gives info about change of a variable; and effect of that change on final function (divergence)

- focus on blue box in $Y(l-1)$

(*) for every l^{th} layer filter (convolution); each position in map in $(l-1)^{th}$ layer affects several positions in the map of l^{th} layer.

(*) in slides; the blue box features in every 'scan' → see slides

(*) that was convolution

(*) now pass through activation part of C.L.

$Y(l-1)$ → $z(l)$ → (activation fn.) → $Y(l)$

$Y(l-1)$: output of $(l-1)^{th}$ layer

$z(l)$: affine combo at l^{th} layer

$Y(l)$: output of l^{th} layer

- Backprop (one recursive step):-

$\nabla_{Y(l-1)} Div \leftarrow \nabla_{z(l)} Div \leftarrow \nabla_{Y(l)} Div$

- recall that $\nabla_{Y(l)} Div$ is given from previous layer
- we need to compute $\nabla_{z(l)} Div$ and $\nabla_{Y(l-1)} Div$.
- not treating bias independently from weights notationally (absorb)

(*) BP. conv layer

- computing $\nabla_{z(l)} Div$:- ✓

- component-wise computation

$$\frac{dDiv}{dz(l,m,x,y)} = \frac{dDiv}{dY(l,m,x,y)} f'(z(l,m,x,y)) \quad (I)$$

l - layer
m - map
x, y - position

(*) BP conv layer

(*) computing $\nabla_{y(l-1)}^{biv}$; $\nabla_{w(l)}^{biv}$

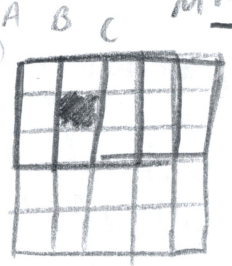
(*) each $Y(l-1,m,x,y)$ affects several $z(l,*,x,y)$ terms.

- All contribute to derivative wrt $y(l-1,m,x,y)$

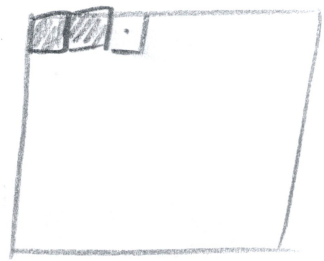
(*) BP-conv. layer

(*) each $y(l-1,m,x,y)$ affects several $z(l,n,x,y)$ terms

- see slides with arrows from $Y(l-1)$ to $z(l)$



mth map of Y

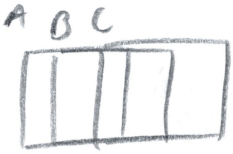


nth map of z

- Isolating how location of square on left affects activation output

- As square is outside of filter C, there is no influence of square on activation.

- Hence empty square in z.



- consider 3 filter locations at A, B, C

- A way of visualis. influence

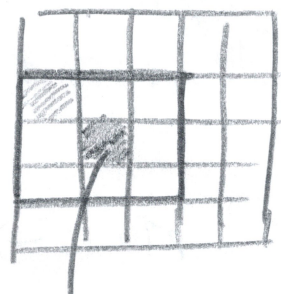
- carry on populating sq.

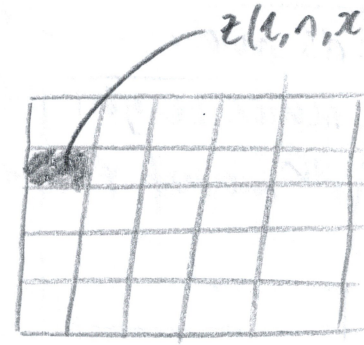
(*) The last z that y will influence → any principle?

- YES. occurs when dark square on Y is in top-left (front-left) of filter

(*) For a particular mth map of Y and nth map of z, we want to assess influence of $y(l,m,x',y')$ on $z(l,n,x'',y'')$ where (x',y') and (x'',y'')

(*) To do so, must place filter with 'this z' at top left corner





- we want weight that multiplies $y(l,m,x',y')$ to $z(l,n,x'',y'')$ transforms -

- each $y(l-1,m,x',y')$ affects several $z(l,n,x'',y'')$ terms (all) through weight $w(m,n,x'-x'',y'-y'')$ (or)

(*) Affects terms in all l^{th} layer z-maps i.e. $\begin{cases} z(l,0,x'',y'') \\ z(l,1,x'',y'') \\ \vdots \\ z(l,N,x'',y'') \end{cases}$

- for all l^{th} layer z-maps $(n=0,1,\ldots,N)$

(*) All of them contrib. to divergence wrt $y(l-1,m,x,y)$ (derivativ.)

- can you see multiplicity of influence paths now?

$$\sum \frac{dDiv}{dz}\cdot\frac{dz}{dy}$$

"z influenced by y"

- note:- (drop layer indices)

$y(l-1,m,x,y) = y(m,x,y)$

$z(l,n,x',y') = z(n,x',y')$

$$-\frac{dDiv}{dy(m,x',y')} = \sum_{n=1}^{N}\sum_{x'',y''}\frac{dDiv}{dz(n,x'',y'')}\cdot\frac{dz(n,x'',y'')}{dy(m,x',y')} \quad (I)$$

$$= \sum_{n=1}^{N}\sum_{x'',y''}\frac{dDiv}{dz(n,x'',y'')}\cdot w(m,n,x'-x'',y'-y'')$$

(weights)

- (W): BR uses (x,y) and (x',y') to refer to pos. of y-map and z-map

(E): I used (x',y') and (x'',y'') (to differentiate between argument and instantiation)

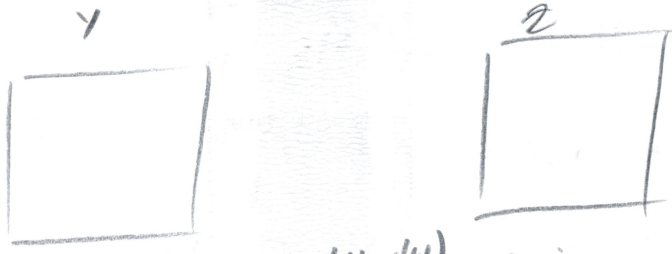
(*) Towards finding derivative of weights (BP conv. layer)

- A particular weight $w(m,n,x,y)$ governs the rel between the m^{th} map of Y to the n^{th} map of Z

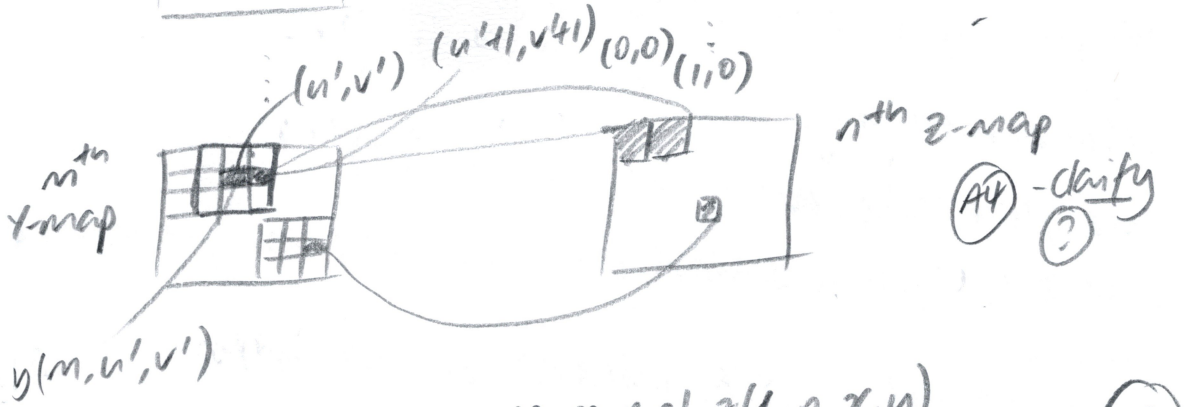
-

PTO. →

$w(m,n,u',v')$



(**) - slides missing
- see S2020 slides



A4 - clarity ?

- Each weight $w_l(m,n,x',y')$ also affects several $z(l,n,x,y)$ for every n (A3)
  - Affects terms in only one z-map
  - All entries in map contrib. to the derivative of the divergence wrt $w_l(m,n,x',y')$ (?)

$$\frac{dDiv}{dw_l(m,n,x,y)} = \sum_{x'',y''} \frac{dDiv}{dz(l,n,x'',y'')} \frac{dz(l,n,x'',y'')}{dw_l(m,n,x,y)}$$

$$= \sum_{x'',y''} \frac{dDiv}{dz(l,n,x'',y'')} y(l-1,m,x''+x,y''+y)$$

(*) Same impl. simple than maths

(*) CNN forward

- key principle:- $z = wY$, $f(z)$

$$\frac{df}{dw} \mathrel{+}= \frac{df}{dz} Y$$

$$\frac{df}{dY} \mathrel{+}= \frac{df}{dz} w$$

- use this for CNN backward layer l

- compute z values within each loop

(*) Go through layers, filters; each filter scans input.
- position filter at a partic. location, compute pairwise product of filter pos and y and add them up.

(*) Backward layer l

(*) Pseudocode parts (I), (II), (III) together for CNN backprop

(*) only 3 lines changed in backprop compared to forward prop

(*) multiple way of casting as tensors

(*) Backward with strides

- messy algebraically; but trivial in code. (?)
- (A5): make sure you understood distinction between pseudocode.

(*) How a single $y(l-1,m,x,y)$ influences $z(l,n,x',y')$

- Examine how each (x,y) position in m^{th} Y-map (one map) influences various locations in an arbitrary n^{th} Z-map
- Previously; for CNN backprop, we examined how m^{th} Y-map influenced multiple N Z-maps

(*) (A6): Review interactive, color coded slides

(*) Have to reverse direction of influence to compute the deriv. of that x,y component of Y.

(*) Each z is sum of component-wise product of filter elements and elements of the region of Y it is placed on.

(A7): influence pattern is mirror image of weights

(*) (A7): Review formulae here; index correspondences

(A8): $$-\frac{\partial Div}{\partial y(l-1,m,x,y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l,n,x-x',y-y')} w_e(m,n,x',y')$$

mirror image

(*) (A9): write some examples if this is unclear during review.

(*) computing derivative

(*) Each deriv wrt y at (x,y) is obtained by flipping the filter left-right, top-bottom and computing the inner product

with respect to the square patch of $\frac{\partial Div}{\partial z}$ ending at (x,y).

- This would be for any (x,y).

(48). Draw diagrams during review (see slides)

- (57): $\frac{\partial Div}{\partial y(l-1,m,x,y)}$ is obtained by taking filter $W_l(m,n,*,*)$
  - Then we flip filter left-right, top-to-bottom; and convolve with ✓ $\frac{\partial Div}{\partial z(l,n,x',y')}$
- (57): Except we are not just interested in one map - interested in many

---

(*) Moving to derivatives over multiple maps

(60): in reality, derivative at each (x,y) location is obtained from all z-maps

- filter will have one plane for each map
  - each of these will be mirror-imaged (is this kernel flipping?)

(49): see slides, colored interactive to go up another level of abstraction from this.

BR: maths is tedious, consuming; but rely on visual intuition here to trivialise the difficulty.

(*) observe: to compute derivative at top-left corner, filter begins outside the $\frac{\partial Div}{\partial z(l,n,x',y')}$ plane

- more precisely; convolution of flipped filter and $\frac{\partial Div}{\partial z(l,n,x',y')}$ only results multiplication of bottom right corner of flipped filter and top-left corner of $\frac{\partial Div}{\partial z(l,n,x',y')}$ at beginning of 'scanning' from top-left

(*) note flipped filter will be 'hanging off edge' (see slides)

(*) Backprop → have to allow filter to fall off edge

→ require zero-padding

(*) computing deriv.

- convolution of $\frac{\partial Div}{\partial z(l,n,x,y)}$ by the inverted filter

- after 0 padding with (L-1) zeros on every side ~ (?)

- can rely on forward code - amend.

(*) derivatives for a single layer l

- vector not.

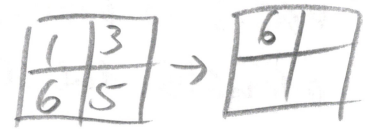
(A10) - review pseudocode

(*) For derivatives wrt weights (as opp. to y)

- interpretation needs treating...

(*) pooling and downsampling (stride > 1 typically)

(*) max pooling



- At each location:-

i) find position of input where value is maximum

ii) copy value at that position

- recall defn of deriv; change of output wrt change of param

BR: If 6 is 'copied' value; will modif. of 1,3,5 change output?

- NO. (as they do not 'influence' output)

- "derivative with respect to 1,3,5" - 0

(~) (?) (A11) - tighten logic loose

(A11) - review max pooling equations

(*) derivative of max pooling

(A11): $\frac{dDiv}{dY(l,m,k,t)} = \begin{cases} \frac{dDiv}{dU(l,m,i,j)} & \text{if } (k,t) = P(l,m,i,j) \\ 0 & \text{otherwise} \end{cases}$

max pooling op: $P(l,m,i,j) = \underset{\substack{k \in \{(i-1)d+1, (i-1)d+K_{lpool}\} \\ n \in \{(j-1)d+1, (j-1)d+K_{lpool}\}}}{\text{argmax}} Y(l,m,k,n)$ (A12) - review

$U(l,m,i,j) = Y(l,m,P(l,m,i,j))$

layer

(*) max pooling at layer l

- pseudocode (A12)

(*) mean pooling:-

- mean pooling op: $U(l,m,i,j) = \frac{1}{K_{lpool}^2} \sum_{\substack{k \in \{(i-1)d+1, (i-1)d+K_{lpool}\} \\ n \in \{(j-1)d+1, (j-1)d+K_{lpool}\}}} y(l,m,k,n)$

(*) derivative of mean pooling:

- derivative of mean pooling is distributed over the pool:-

$k \in \{\dots\}$ (from above); $\frac{dy(l,m,k,n)}{du(l,m,k,n)} = \frac{1}{K_{lpool}^2}$

$n \in \{\dots\}$

(*) mean pooling layer at layer l + derivative

- pseudocode (A13)

- typo (!) - $dy(l,j,p,n+i,m+j) \mathrel{+}= (1/K_{lpool}^2)\, du(l,j,x,y)$

(*) Learning network

- Backprop for CNN complete (after review!!)
- Now embed in gradient descent

(*) Implicit assumption

- Can subsequent maps increase in size?

(*) 1D figure

- Scanning with stride of 18
- Arrow ⇒ contain 6 weights (2 input neurons → 3 output neurons)
- distributed parameter network
- individual neurons have smaller strides (covered in H/W)
- No. of circles within bar → no. of channels
- Scanning ⇒ no. of bars stays the same or goes down (tree structure)
- stride > 1 ⇒ no. of bars reduces

(A12) - logic - make sure watertight

59:00 →

with hope of increased resolution

- 'neighbors' → middle gets only one contributions from lower bars
  left and right gets multiple —11— —11—

(*) Transposed convolution

(A13) - seriously confused

- see this for 2-D

(*) 2D-expanding convolution

- output size typically an integer multiple of input
  - +1 if filter width is odd.

$z(l,i,j) = \sum_m \sum_k \sum_t w(l,m,i-kb, j-cb) I(m,k,t)$ (?)

b is stride → scaling factor between size of z-map and Y-map.

(*) Recasting CNN as shared param MLP; expanding layers have weight matrices that are taller than wide.

(?): ↑ no. of weights matrices ⇒ ↑ no. output channels

↑ stride ⇒ ↑ size of individual maps

(*) use scale factors to better understand relation between scale factor between z maps and stride

---

(*) Invariance:-

- shift invariance accounted for
- How about rotation/scale/reflection invariance ?

(*) Shift invariance - different perspective

(AIL) notation unclear

(*) transform invariance

(*) 1 filter produces a set of transformed maps
(*) network becomes invariant to all transforms considered

- affects backprop

(*) transform invariance

- (AIS) - review pseudocode (enumerate over transforms; compute cor scan with transformed filte)

(*) BP with transform invariance

- derivatives flow back through transforms to update individual filters.

- As long as you have correspondence between original and transformed filter, all fine.
- # computations not covered in same amount of depth.

(*) Exact location of object

- implicitly ⟶ by recording where max is triggered in final layer
- explicitly → incorporate additional output layer
  - predicts corners of bounding box of object (8 coordinates)
  - A16: why 8?
- can be used for pose estimation

training: compute 2 divergences over 2 output layers (conventional, bound box)
- final divergence is weighted sum of 2 divergences
  (e.g. X-entropy of class. layer + L_2 loss of bounding box)

(*) Model variations

- depth-wise convolutions ⟶ MobilNet (HW2)

(*) depth-wise convolutions

- Alternate view of convolution (standard) A17 - review - not clear

depth-wise convol. as a switch in order
↳ only perform a 'local convolution' at each map?
- layerwise convolution over all maps and then add
(think this just a slight variation on how to view the operation from a matrix/tensor multiplication perspective).
  - still same op

- ? A: collapsing? → review convolution op.

(*) convolutional vs depth-wise convol.

- Review this slide
- shows efficiency (in terms of no. params required) for depth wise convol.
- note use of 30 vs 20 filters in terms of no. parameters.

(*) what do filtes actually learn/est?

- receptive fields
- subsequent neuron weights afte 1st (more difficult)
- ImageNet

(*) Training Issues

- convergence issues } review slides, for history of ImageNet
- data aug.