11-785 CNNs review

$t = 1 : T-K+1$

$x(:, t:t+K-1) = x_{seg}$ i.e. a matrix? - what is dimensionality (?)

$y(t) = MLP(x_{segment})$.

(?)

- Recall that previously we have $(\underline{x}_i, y_i)$ $\quad y_i \in \mathbb{R}^L$ or $\{0,1\}$

$\boxed{\underline{x}_i \in \mathbb{R}^D}$

- Now, because sequential, each
  - we have
- Bit stumped by this.
- This is the 1D case ?
- Refresher on 1D:-

we have n input-output pairs:-

$(\underline{X}_i, \underline{d}_i)$ i.e.

- Are we now saying we have N number of (D x T) sequences ?, in the 1D case?

$\underline{X}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix}$

$t, t+1, \ldots t+K$ 0,9

(?) $K+1$

- Then we scan each?

so if we do $\underline{x}^{(i)} \in \mathbb{R}^{D \times T}$ $\quad i = 1, \ldots, N$, we have

so ~~X_{n1}~~ ~~x_{n1}~~ ~~x~~

$$\underline{x}^{(i)} = \begin{bmatrix} x_{11}^{(i)} & \cdots & x_{1T}^{(i)} \\ x_{21} & & \vdots \\ \vdots^{(i)} & & \\ x_{D1} & \cdots & x_{DT}^{(i)} \end{bmatrix} = \begin{bmatrix} | & | & | \\ \underline{x}_{:,1} & \underline{x}_{:,2} & \underline{x}_{:,T} \\ | & | & | \end{bmatrix}$$ ✓

- Then

$x_{segment} = x(:, t:t+K-1)$

$y(t) = MLP(x_{segment})$

so $x_{segment} \in \mathbb{R}^{dx}$

i.e. $\underline{x}^{(i)} = \begin{bmatrix} | & | & | \\ \underline{x}_{:,1} & \underline{x}_{:,2} & \cdots & \underline{x}_{:,T} \\ | & | & | \end{bmatrix}$

$t : t+K-1$

i.e. K vectors

- So what we do is

$$\underline{X}^{(i)} = \begin{bmatrix} | & & | \\ \underline{x}_{:,1} & \cdots & \underline{x}_{:,T} \\ | & & | \end{bmatrix} \quad \underline{X}^{(i)} \in \mathbb{R}^{d \times T}$$

- If $\underline{X}[:, t:t+K-1]$? ✓ if $K=2$ and $t=0$ then $0:1 \rightarrow$ 1 unit if

$K=3$

$t, t+1, t+2, t+3$

t

- So $y(t) =$

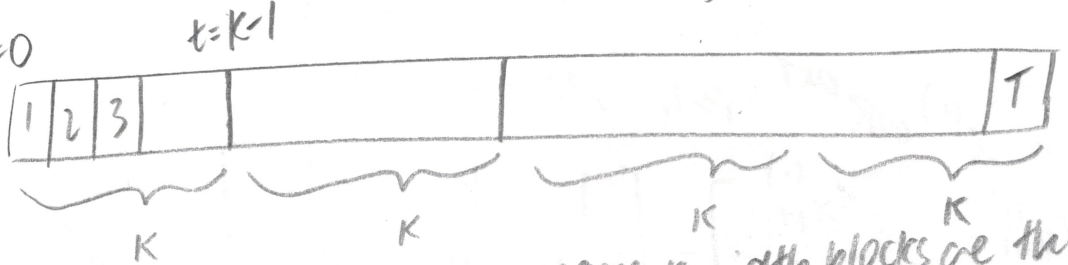
$T-K+1 : (T-K+1) + K - 1$

$T-K+1 : T$

- dimensionality of softmax output? - Each $y(t) \in \mathbb{R}^L$ where L is no. of ~~logits~~ units in final layer (or no. of classes)

- how about a giant MLP?

$Y = \text{giant MLP}(\underline{X})$ - trying to clarify



- For a length T sequence, how many K-width blocks are there?

- ~~There are~~

- we can rewrite multiplicatively as $T = K \cdot m$ where $m = \frac{T}{K} \in \mathbb{Z}$

- we ~~parse~~ pass each of the K-width blocks into y.

- so we will have

- This is now sorted $N = \left\lfloor \frac{T-K}{S} \right\rfloor + 1$.

- still not clear what the dimensionality of $\underline{Y}$ is.

Hence $Y \in \mathbb{R}^{m \times (T-K+1)}$?

- If we have a ~~K-cl~~ M-class classification problem, then each $y(1) \in \mathbb{R}^M$,
- what do we want as output?
- consider 2 class classification first.
- Prior to YES/NO, we have softmax output our 2 classes
- Recall we are going through each sliding window, of which there are $T-K+1$ sliding windows and declaring whether that window contains a 'welcome'.
- Hence $Y \in \mathbb{R}^{M \times (T-K+1)}$ ✓.

- 10/06/21
- How do we have softmax from this?
- Or a giant MLP?

$Y \in \mathbb{R}^{M \times (T-K+1)}$

- $\hat{Y} = \text{giantMLP}(\underline{X})$ where $\underline{X} = [X[:,1:K], \ldots, X[:,T-K+1:T]]$ and $\underline{X} \in \mathbb{R}^{D \times T}$, $X[:,1:K] \in \mathbb{R}^{D \times K}$
- Generalisation to 2D case, after 1D case is simple.
- Need to figure out exactly what is going on with the block matrix parameter sharing.
- What is kernel-width?
- I think I'm going to watch all the lectures again on CNN.
- Shared parameter network

11-785

10/06/2021

L8 - CNNS : Supplementary review

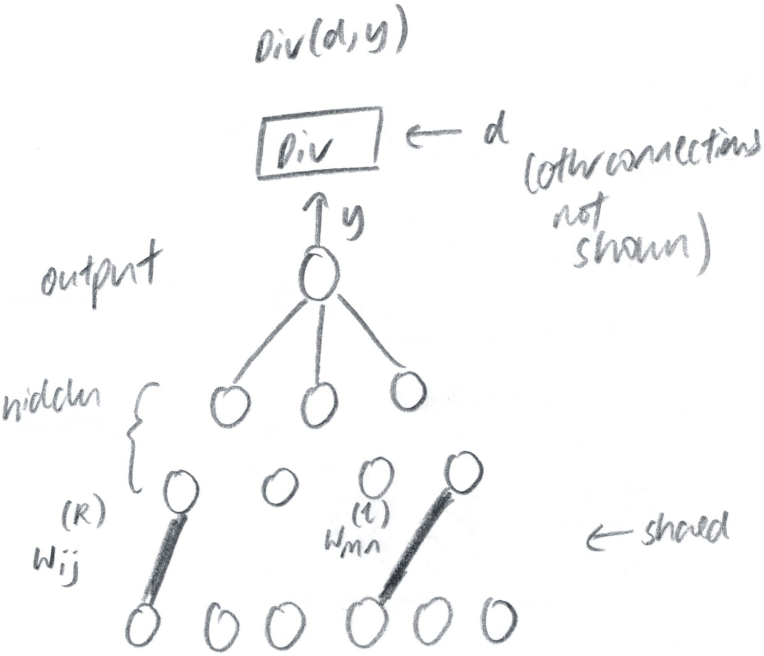
- For the assignment HW2pt1, need to further clarify :-
  1) Link between scanning MLP and CNN
  2) Shared parameter network
  3) Counting parameters
  4) CNN terminology — stride, channel, kernel width, filters

- Notational difference between S.2019 and S.2020
  - BR uses loss/error interchangeably

- Shared parameter networks
- See S.2019 slide 58 lectures.
- The notion of a shared parameter network is that all subnets in slide 58 of S.2019 are identical
- That is the architecture (weights and biases dimensions, activations, final out) are the same.
- And also the values of the parameters in each subnet should also be identical
- And also in the fact that one update of one copy of the subnet (i.e. its parameters using gradient descent after backprop) must equally update all copies
- Every blue-circled subnet in slide 58 is identical
- The following shows shared parameters for a simple network (not sure if scanning)

Learning in shared param networks (S59)

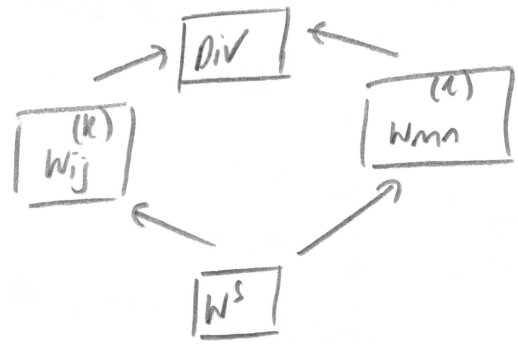
- Shared weights
- For example, if $w_{ij}^{(k)} = w_{mn}^{(l)} = w^s$
- That they are shared means that they are identical, and can be referred to using common notation w^s



- It is then stated that:-
- for any training instance $\underline{x}$, a small perturbation of w^s perturbs $w_{ij}^{(k)}$ and $w_{mn}^{(\ell)}$ identically.
- Each of these perturbations will influence divergence $Div(d,y)$.
- WL: I think 'perturbation' here refers to the act of updating the shared weights w^s using gradient descent and backprop.

computing the divergence of shared params (S59)

- influence diagram



- consider computing derivative of loss/divergence with respect to shared param w^s.
- chain rule:

$$\frac{dDiv}{dw^s} = \frac{\partial Div}{\partial w_{ij}^{(k)}} \cdot \frac{\partial w_{ij}^{(k)}}{\partial w^s} + \frac{\partial Div}{\partial w_{mn}^{(\ell)}} \cdot \frac{\partial w_{mn}^{(\ell)}}{\partial w^s}$$

$$= \frac{\partial Div}{\partial w_{ij}^{(k)}} + \frac{\partial Div}{\partial w_{mn}^{(\ell)}}$$

- Because $\frac{\partial w_{ij}^{(k)}}{\partial w^s} = \frac{\partial w_{ij}^{(k)}}{\partial w_{ij}^{(k)}} = 1$.

computing the divergence of shared parameters (S63)

- key detail here:-
- S - the set of edges - this is some kind of index set

$S = \{e_1, e_2, \ldots, e_N\}$

- w^s is the common weight i.e. shared parameter of the set
- In S63, note that

$$\boxed{\frac{dDiv}{\partial w^s} = \sum_{e \in S} \frac{dDiv}{\partial w^e}}$$

- extends previous formula.

- The key idea:

- Training networks with shared params (S64)

- Note that BR's notation for layer k weight matrix is W_k.

- All we do here is update every shared parameter weight.

- Preliminary pseudocode

- Gradient descent:

  - Initialise all weights $W_1, W_2, \ldots, W_K$

  - DO:

    - For every set S:

      - compute $\nabla_{W^S} loss = \frac{dloss}{dW^S}$ → use previous formula to compute this see (I) below

      - compute $W^S \leftarrow W^S - \eta \nabla_{W^S} loss^T$

      - For every $(k,i,j) \in S$, update:-

        $W^{(k)}_{i,j} = W^S$

  - until loss has converged.

(OR): The idea is that we iterate through all sets of shared indices e.g. $S_1, S_2, \ldots$ (however many shared parameter sets there are).

- update the shared parameters for that particular set S i.e. W^S

(OR): Can view this as updating weights in a single network, but updating each weight in shared parameter network amounts to updating multiple copies of each weight identically

(I) $\nabla_{W^S} loss = \frac{dloss}{dW^S}$ → computed by

- For every training instance X
- For every set S:-
- For every $(k,i,j) \in S$:- → computed via backprop.

$$\nabla_{W^S} Div \pm \frac{\partial Div}{\partial w^{(k)}_{i,j}}$$

- $\nabla_{W^S} loss \pm \nabla_{W^S} Div$.

e.g. S_1 corresponds to all weights coloured red in subnets.

- That is, use previous formula $\frac{dDiv}{dw^s} = \sum_{e \in s} \frac{\partial Div}{\partial w^e}$ as a subroutine in main gradient descent pseudocode.

- looking at slide 63:
- we compute using backprop: $\frac{\partial Div}{\partial w^e} = \frac{\partial Div}{\partial w_{i,j}^{(k)}}$ for every $e \in S$ (i.e. compute derivative of divergence w.r.t. red edges)

- we then sum these to get the derivative of divergence wrt. shared weights :- $\nabla_{w^s} Div \triangleq \frac{\partial Div}{\partial w^e}$ or

$$\nabla_{w^s} Div \triangleq \frac{\partial Div}{\partial w_{i,j}^{(k)}}$$

- However, question that remains is (possibly notation) is how or why $\nabla_{w^s} Loss \triangleq \nabla_{w^s} Div$ or in S.2019 slides $\nabla_{w^s} Err \triangleq \nabla_{w^s} Div$
- I think BR is referring to $\nabla_{w^s} Div$ as per-training-instance gradient, whilst $\nabla_{w^s} Loss$ is per-batch or per-mini-batch gradient (i.e. additive sum). ✓
- Yes your understanding is completely correct
- re: the precise nature of computations here is still not entirely clear

(Q): whilst it seems that we are saying updating w^s means we update all w^e where $e \in S$ i.e. update all shared weights w^s ~~i.e. all~~ or multiple copies of the same weights in same set S

(Q): However does that in general mean that $\frac{\partial Div}{\partial w^e}$ in $\frac{\partial Div}{\partial w^s} = \sum_{e \in s} \frac{\partial Div}{\partial w^e}$ will be the same?

- my guess is no, because we are feeding/using a different window of 1D sequence at each sliding window. X

- Key idea:

(6.1): Modify backpropagation rules to combine gradients from parameters that share the same values.

1. Link between scanning MLP + CNNs - recorded computations and shared params

---

- Scanning order → 1. Pass window by window through entire network.
2. Pass window by window through neuron by neuron

(see orig. notes or review)

- Can view it as column-wise or row-wise
- (6.11) - MLP scanning uses the latter order of operation
i.e. for $t = 1:L$ (vectorised). - see slide 116 scanning order
for $t = 1:T-K+1$
- That is run through layers one-by-one; and each layer scans
- This is precursor to scanning in CNNs
- Idea is you can switch around for loops without affecting code; even though it alters intuition of what is going on.

- I'm a bit rusty on the hierarchical explanation of scanning
- That is, from slide 117 onwards i.e. "Scanning in 2D: a closer look"
use the pseudocode as the foundation; overlay the intuition on top of that
That is, look at 2D scanning pseudocode, and the re-ordered computations; overlay intuition on top of that.
- and also from slide 162 onwards, "Distributing the scan."
- See in particular, see slide 193 "Reordering the computation."
- It seems the convolutional part comes from the fact that maps scanned by each layer are a different size

---

convolutional neural network (SM6)
(aka scanning within an MLP)

- The key point here is the definition of convolution; or exposure to idea.
- This is covered in more detail and defined in CNN pt II and III (late lectures).
- Also note it is the re-ordered, layer first iteration computation that is basis for CNNs

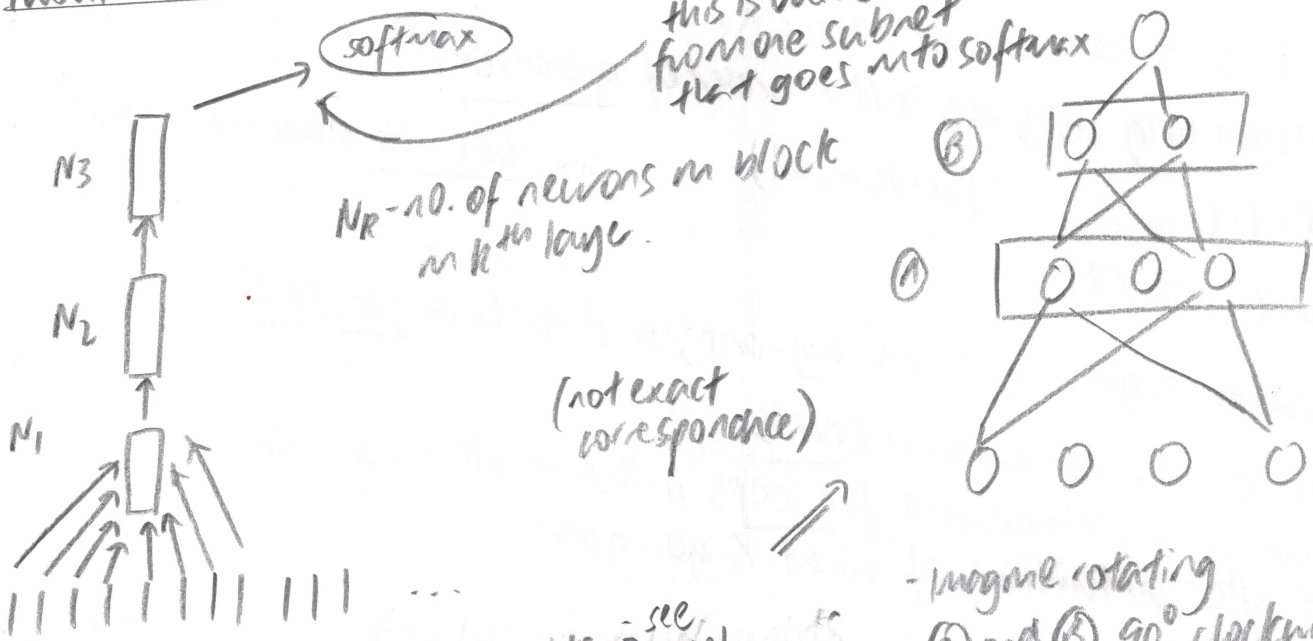
(6m): that is, for precursor to CNNs, we do iterate through layers first

- The 3D tensor / 3D scan is a bit obtuse to me (slide 197).

---

3) counting parameters

scanning without distribution (slides in S.2020 suppl. slides part II)

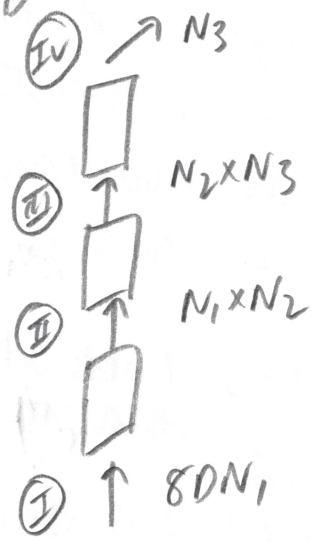


- Each arrow represents edges / weights → see suppl. slides S.2020 pt(I)
- imagine rotating (A) and (B) 90° clockwise

(6n) - best to see this in context of how multiple copies of identical subnets (i.e. with shared parameters) feed into a final softmax layer for non-distributed scanning.

- counting parameters.
- we have a window of 8 timesteps (or L timesteps in general).
- Each window 8 input vectors, each of dimensionality D
- There are N_1 neurons in the 1st block.
- There are N_2 neurons in the 2nd block.
- There are N_3 neurons in the 3rd block
- Hence total no. of params N_W :-

$$N_W = \underbrace{8DN_1}_{\text{(I)}} + \underbrace{N_1N_2}_{\text{(II)}} + \underbrace{N_2N_3}_{\text{(III)}} + \underbrace{N_3}_{\text{(IV)}}$$



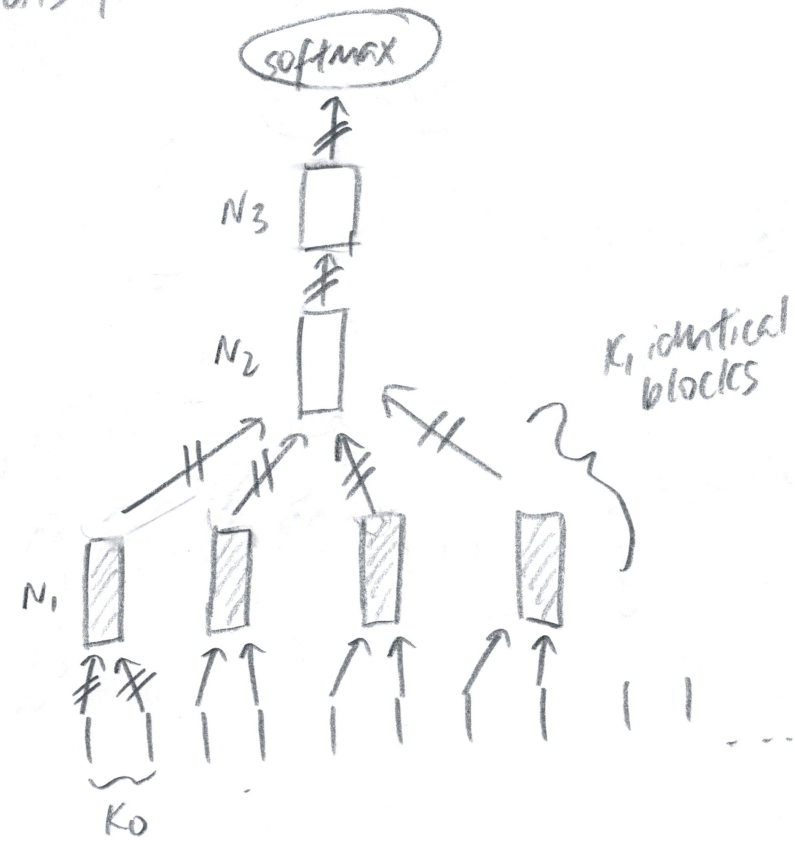
- In general, for L time instances/window width:-

$N_W = LDN_1 + N_1N_2 + N_2N_3 + N_3$ (in a subnet)

- This ignore bias terms in the computation; this for without distribution
- we only count parameters for one column (i.e. subnet I think) since other columns/subnets are identical.
- Scanning (without distribution) uses these shared/identical parameters i.e. the same subnet scanning

Distributed scanning (slides in S.2020 suppl. slides part III)

- currently, on review, it is unclear how the notation maps to the parameter quantity computation.
- List the parameter quantity calculations for distributed and non-distr case (there are ambiguities.).



- This new diagram represents a scan of 8-time step wide patterns with a stride of 2 time steps distributed over layers.
- All double marked arrows ⇉ denotes the unique weights for distributed scanning
- This is what we count.
- Lower level shaded blocks are identical i.e. clones of one another

Counting parameters:

- A window of 2 time steps (or K_0 timesteps in general).
- Each

11-785 02/06/2020

L8 - CNNs - review

- Supplements key points in notes and slides.

(*) Perceptron as a correlation filter

- Note:-

$$y = \begin{cases} 1 & \text{if } \underline{x}^T \underline{w} \geq T \\ 0 & \text{otherwise} \end{cases}$$

- we can view this through the lens of correlation
- Perceptron fires if correlation between weights and inputs exceed a threshold

1.

(*) Perceptron fires if input patterns looks like pattern of weights.

(*) Following this; can view input layer as 'feature detectors' that detect if certain patterns have occurred in input.

(*) Network can be viewed as a function over feature detectors

(*) SHIFT-INVAR ✓

(*) scanning with an MLP

- sliding windows → each produces a classification
- combination into final output :-
  - OR function :- if any window contains 'welcome' (speech recognition)
  - equivalent to MAX function
  - softmax/logistic : adjacent windows can combine evidence

(*) clarity on input of MLP :- later.

(R1) (*) See end of these notes.

(*) Basic scanning for 1D/2D input pseudocode.

- K×K - MLP patch-size
  W - width of image
  H - height of image (2D-image)

```
FOR i = 1:W-K+1
  FOR j = 1:H-K+1
    img segment = img(i:i+W-1, j:j+W-1)
    y(i,j) = MLP(imgsegment)
Y = softmax(y(1,1), ..., y(W-K+1, H-K+1))
```

'1D-data'

- K - width of patch

```
FOR t = 1:T-K+1
  Xsegment = x[:, t:t+K-1]
  y(t) = MLP(Xsegment)
Y = softmax(y(1), ..., y(T-K+1))
```

(R2)

(*) Regular networks vs scanning networks

- Regular NN:
- each neuron in a layer is connected by a unique weight to every unit in previous layer (may be an input)
- E.g. first layer weights:-

$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1M} \\ w_{21} & w_{22} & \cdots & w_{2M} \\ \vdots & & & \\ w_{N1} & w_{N2} & \cdots & w_{NM} \end{bmatrix} \quad W^{(R)} \in \mathbb{R}^{N \times M}$$

(*) weights matrix: (NxM) fully populated with NM unique parameters

- Scanning NN:
- each neuron is connected to a subset of neurons in the previous layer.
  - sparse weights matrix
  - weights matrix structured with identical blocks

$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & & & & \\ w_{21} & w_{22} & & & & \\ w_{31} & w_{32} & & & & \\ & & w_{11} & w_{12} & & \\ & & w_{21} & w_{22} & & \\ & & w_{31} & w_{32} & & \\ & & & & w_{11} & w_{12} \\ & & & & & w_{31} \; w_{32} \end{bmatrix}$$

- rest are zero entries
- (?) unclear
- especially as there is a time-dimension
- do we use a weights matrix that is a larger placeholder? i.e. a $K^2(NM)$ as large where K is patch size?

- network is a shared parameter network

(*) stride → need to clarify

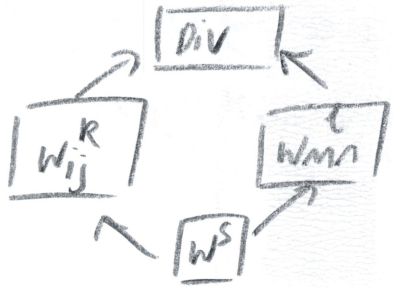
(*) Param estimation in shared param nets:-

as $W^S = w_{ij}^{(R)} = w_{mn}^{(L)}$

$$\frac{\partial Div}{\partial W^S} = \frac{\partial Div}{\partial w_{ij}^R} \frac{dw_{ij}^R}{dW^S} + \frac{\partial Div}{\partial w_{mn}^L} \frac{dw_{mn}^L}{dW^S} = \frac{\partial Div}{\partial w_{ij}^R} + \frac{\partial Div}{\partial w_{mn}^L}$$

$= 1$ $\quad = 1$

- influence diagram:



(*) computing divergence of shared parameters.

- Define $S = \{e_1, \ldots, e_N\}$
- S any set of edges that have a common value, and W^S is the common weight of the set. (see slides in colour)

(611) $$\frac{dDiv}{dW^S} = \sum_{e \in S} \frac{\partial Div}{\partial W^e}$$

(*) individual terms in sum → use backprop

(*) training networks with shared param. PSEUDOCODE

- Gradient descent algorithm
- Initialise all weights $\underline{W}_1, \underline{W}_2, \ldots, \underline{W}_K$

DO:-

- For every set S

$$\nabla_S Err = \frac{dErr}{dW^S}$$

- For every training instance X
  - For every set S:-
    - For every $(k,i,j) \in S$

$$\nabla_S Div \mathrel{+}= \frac{\partial Div}{\partial W_{i,j}^{(k)}} \quad (*)$$

$$W^S = W^S - \eta \nabla_S Err$$

- For every $(k,i,j) \in S$ update:-

$$W_{i,j}^{(k)} = W^S$$

$$\nabla_S Err \pm \nabla_S Div$$

- until Err has converged

(*) computed via backprop

(*) Scanning order (clarity)

1) Pass each window through entire network, this happens W times for each window i
   - These W outputs are then passed through softmax

2) Pass each window through neuron, neuron by neuron, layer by layer.

- 1st neuron - output for W windows — 1st layer
- 2nd neuron — " —

⋮

- 1st neuron — " — 2nd layer

⋮

(*) (6) : order of computation merely amounts to exchaging loop nesting order
↳ does not change final output

(*) scanning order pseudocode (1-D)

order 1 - window by window through entire net.

```
FOR t = 1:T-K+1      → over window
  FOR l = 1:L        → over layers
    FOR j = 1:D_l
      if (l==1) #first layer operates on input
        y[0,:,t] = x[:, t:t+K-1]
      end
      z(l,j,t) = 0
      FOR i = 1:D_{l-1}
        z(l,j,t) += w(l,i,j) y(l-1,i,t)
      y(l,j,t) = activation(z(l,j,t))

Y = softmax(y(L,:,1), ..., y(L,:,T-K+1))
```

(*) exchanged order 2:-

```
FOR l = 1:L
  FOR t = 1:T-K+1
    FOR j = 1:D_l
```

(*) scanning MLP - vector not. (1-D)

```
FOR l = 1:L
  FOR t = 1:T-K+1
    if (l == 1)
      y(0,t) = x[:, t:t+K-1]
    end
    z(l,t) = W(l) y(l-1,t)
    y(l,t) = activation(z(l,t))
Y = softmax(y(L,1), ..., y(L,T-K+1))
```

(*) can view activity at one neuron as:-

$\left(\sum_{i,j} w_{ij} p_{ij} + b\right)$ for one window of input data p

(*) (OH): view slides interactively → helps with the visual intuition of slides (excellent on BR part)

(*) scanning - a close look

- Recursing the logic

(*) second-level neurons are scanning the rectangular outputs of the 1st level neurons.

(*) They are jointly scanning multiple 'pictures'

- Each location in output of 2nd level neuron considers corres. locations from outputs of all first-level neurons

(*) output of penultimate layer is a grid/picture

(*) This entire grid is passed through a final layer/neuron that performs:-

i) logical OR

ii) max output

iii) softmax

iv) MLP

(*) 2-D scanning pseudocode

(*) non vector form

```
FOR x = 1:W-K+1
  FOR y = 1:H-K+1
    FOR l = 1:L                    # layers operate on vector at (x,y)
      FOR j = 1:D_l
        IF (l==1) #first layer ops on input
          Y(0,:,x,y) = Img(1:C, x:x+K-1, y:y+K-1)
        end
        z(l,j,x,y) = 0
        FOR i = 1:D_{l-1}
          z(l,j,x,y) += w(l,i,j) Y(l-1,i,x,y)
        Y(l,j,x,y) = activation(z(l,j,x,y))
Y = softmax(Y(L,:,1,1) ... Y(L,:W-K+1, H-K+1))
```

(*) adjusted order:

```
FOR l = 1:L
  FOR j = 1:D_l
    FOR x = 1:W-K+1
      FOR y = 1:H-K+1
```

(*) vector version → see slides.

(*) Distributing the scan/behaviour of layers.

(W): notes are a little opaque on this

(0/5 1): Finding the distinction between earlier presentation and explanation of how scan is distributed difficult to follow

(*) in this context:

- stride - no. of steps we move 'sliding window' across by
- Also have to specify size of window

(RI): Here, we have n inputs $\underline{x}^{(1)}, \underline{x}^{(2)}, \underline{x}^{(3)}, \ldots, \underline{x}^{(n)}$ (1-D case)

- Each i^{th} input is a $D \times T$ matrix.
- D - dimensions and T - temporal length.

- Notes from this point onwards clarify things which came up.

$$\underline{x}^{(i)} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{0T} \\ \vdots & \vdots & & \vdots \\ x_{01} & x_{02} & \cdots & x_{0T} \end{bmatrix}$$

- For 1D sequences; if we want a width K patch to scan over $\underline{x}^{(i)}$, then we do

$\underline{x}^{(i)}[:, t:t+K-1]$ $\longrightarrow$ slices are using Python slice notation.

- For clarity, for a window of width K :-

$$\underbrace{t, t+1, t+2, \ldots, t+K-1}, t+K$$

a window of width K (i.e. accounting for slicing notation).

- For clarity on the loop

FOR $t = 1 : T-K+1$

x_segment = $\underline{x}[:, t:t+K-1]$

$y(t)$ = MLP(x_segment)

Y = softmax($y(1), \ldots, y(T-K+1)$)

- At $t=1$

x_segment = $X[:, 1:K]$

- At $t = T-K+1$

x_segment = $X[:, T-K+1:T]$

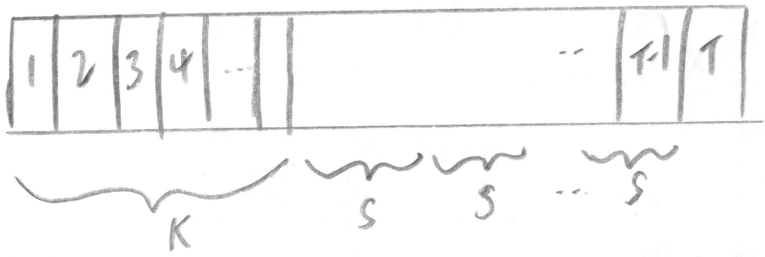
- We are proceeding along a length T sequence K steps at a time.

- It is important to note that we proceed with a width-K sliding window and the no. of steps this sliding window advances by 1 time step at each point.

- That is the above example has stride $S=1$.

- To see how the general formula works:-
- for the number of width-K windows in a sequence of length T, which increases by S time steps / has a stride of S, note:-

(*) this is simple stuff, but is crucial to understand convolutional neural networks. and tricky examples



sequence of length T

- in its initial position, the width-K window occupies K time steps.
- It will advance with a stride length of S time steps, until termination.
- we want to know the number of width-K windows, which we call N.
- Notice that if we subtract K time-steps from the total length of the sequence T, we have (T-K).
- Then the no. of strides of length S taken by the sliding window is just:-

$$\frac{T-K}{S}$$

- accounting for the fact that (T-K) may not be a multiple of the stride length S, we carry out remainder division, and forget about remaining steps at the end.
- The number of remaining time steps at the end will be less than S (otherwise we could allow sliding window to progress another S time steps).
- Hence the no. of strides of length S taken by the sliding window is :-

$$\left\lfloor \frac{T-K}{S} \right\rfloor$$ where $\lfloor \cdot \rfloor$ is the floor function

- As the number of width-K windows is just one more than the number of strides, i.e. accounting for the initial position of the sliding window, the general formula is

$$N = \left\lfloor \frac{T-K}{S} \right\rfloor + 1$$

- In the case where the stride length $S=1$ (i.e. in the previous example in the slides)

e.g. $Y = \text{softmax}(y(1), \ldots y(T-K+1))$ earlier

- Note that here $S=1$, and the formula for no. of sliding windows is

$$N = \left\lfloor \frac{T-K}{1} \right\rfloor + 1 = T-K+1 \quad \text{as required.}$$

(R2) - On the dimensionality of :-

$$Y = \text{softmax}(y(1), y(2), \ldots, y(T-K+1)).$$

- NOTE $Y \in \mathbb{R}^{M \times (T-K+1)}$ in the time-delay NN/1D example, where M is no. of classification categories (if we are doing classification).
- In the example in slides of detecting whether there is a "welcome" in an audiosignal.
- Note that we could model with $M=2$ i.e. yes or no
- The softmax output $y(1) \in \mathbb{R}^2$ in this case (consisting of probabilities).
- The reason for Y having $(T-K+1)$ columns is because we want probabilities over 2 classes for each of $(T-K+1)$ individual windows of width K.

(or. softmax stage)