Title: Machine Learning (2021/2022)

Speakers: Lauren Hayward

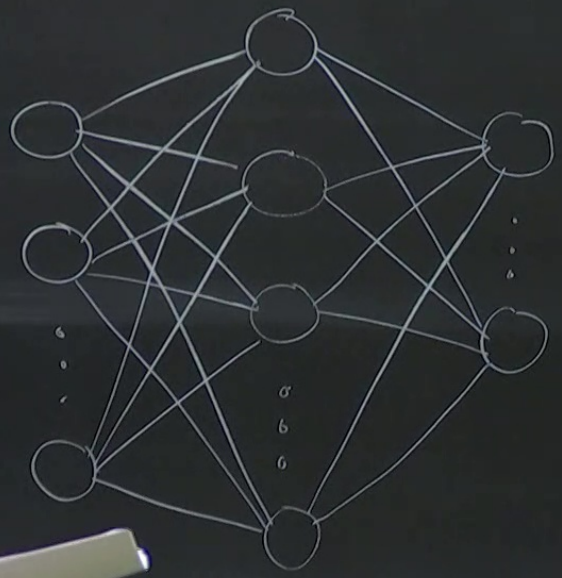
Collection: Machine Learning (2021/2022)

Date: April 19, 2022 - 11:30 AM

URL: https://pirsa.org/22040071

Abstract: This course is designed to introduce modern machine learning techniques for studying classical and quantum many-body problems encountered in condensed matter, quantum information, and related fields of physics. Lectures will focus on introducing machine learning algorithms and discussing how they can be applied to solve problem in statistical physics. Tutorials and homework assignments will concentrate on developing programming skills to study the problems presented in lecture.

Lectures 4-6: fully-connected feedforward neural networks (FFNNs)



Here every neuron in layer l is connected to every neuron in layer $l+1$

Issue: Theses FFNNs do not exploit properties such as locality and translational invariance

$\Rightarrow$ a lot of spatial information is lost or must be learned

not
ls
2
r

Consider a 4x4 image (picture of a dog, spin config. in 2D, etc.)
Let us label the pixels (sites) as:

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

L=4

As input to a FFNN:

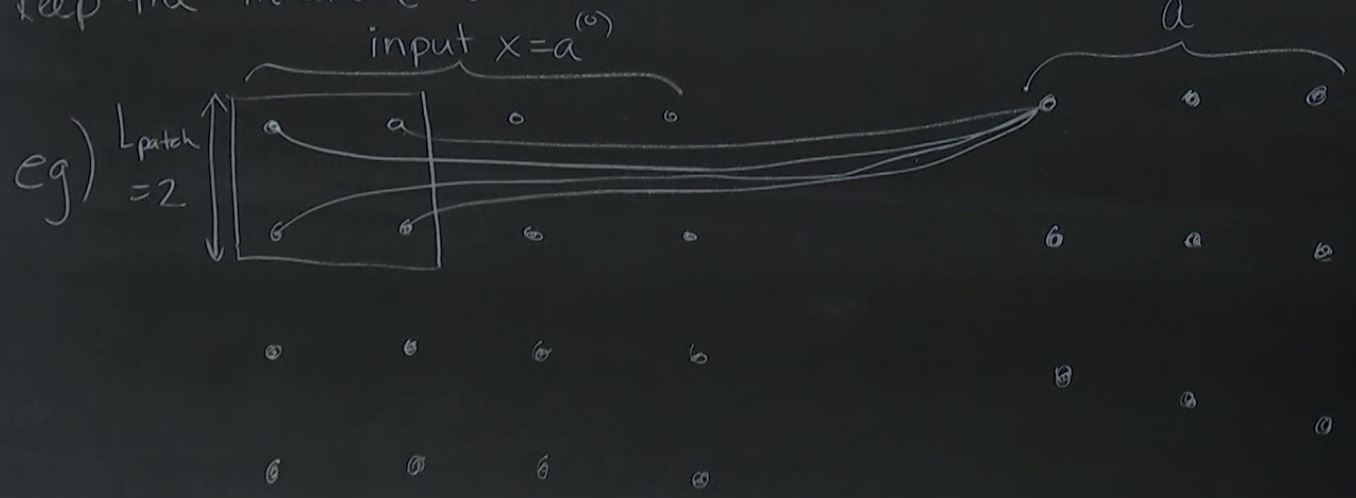
1
2
3
4
5
.
.
.
16

} spatially close
} not spatially close

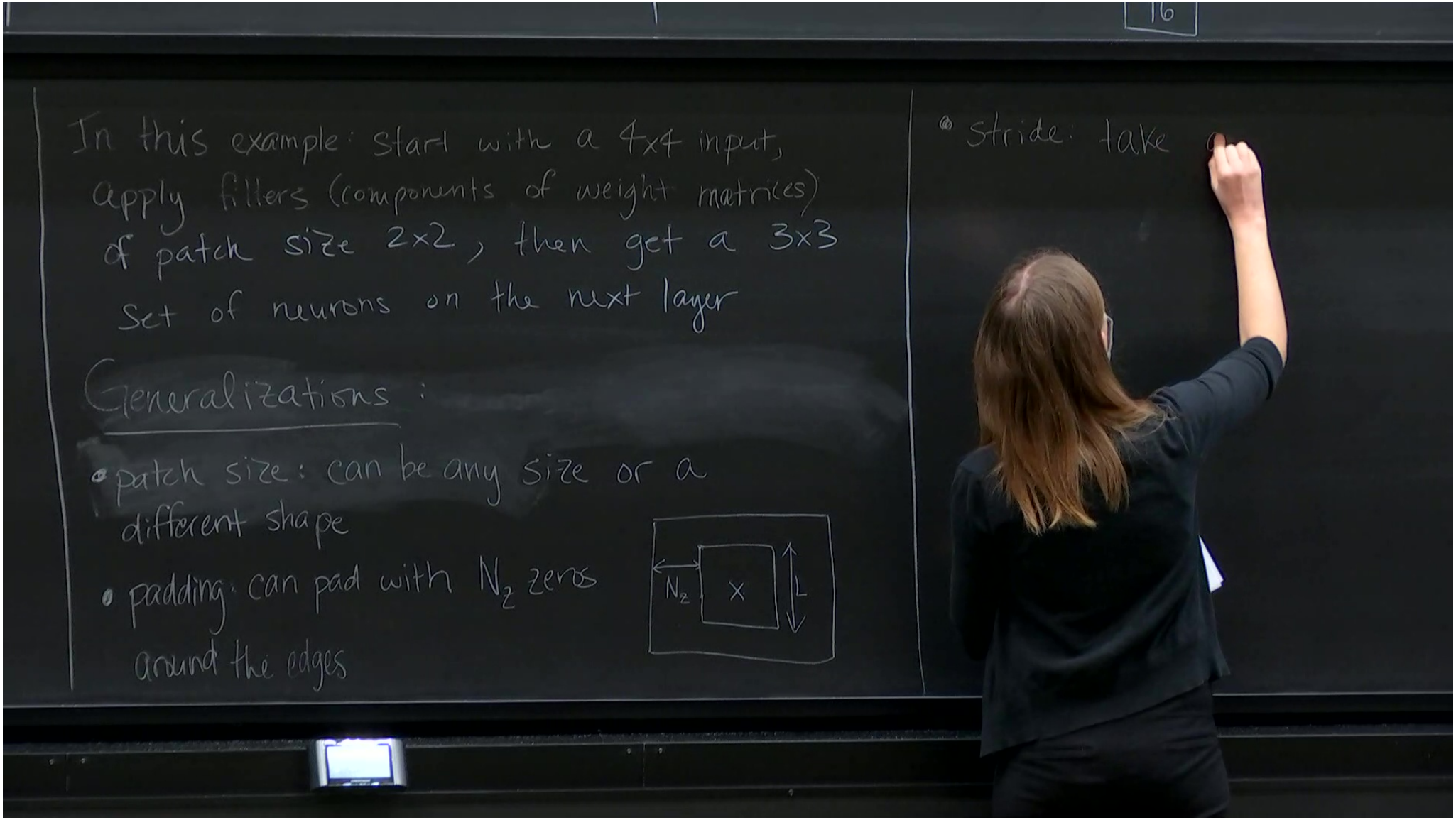
locality is easily lost!

Solution: Local Receptive Fields (LRFs)

Idea: process the data in local "patches" that keep the inherent dimensional structure



In this examp


16
In this example: start with a 4x4 input, apply filters (components of weight matrices) of patch size 2x2, then get a 3x3 set of neurons on the next layer
Generalizations:
• patch size: can be any size or a different shape
• padding: can pad with N_2 zeros around the edges
N_2
X
L
• stride: take

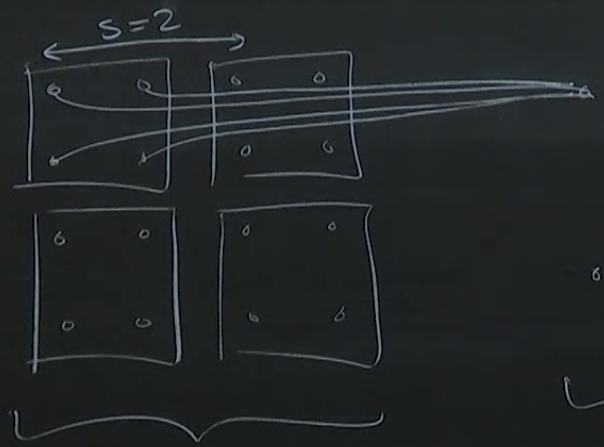
f input,
matrices)
t a 3×3
layer

a

N_2 X L

• stride: take a step of size S before applying the next filter

eg)



$a^{(0)}$ is 4×4

$a^{(1)}$ is 2×2

If $a^{(0)}$ has $L \times L$ neurons, then $a^{(1)}$ has $L_1 \times L_1$, with:

$$L_1 = \frac{L - L_{patch} + N_z}{S} + 1$$

(assuming square patches)

Note: if we apply periodic boundary conditions rather than padding:

$$L_1 = \frac{L}{S}$$

around the edges

For a 2d image, in addition to the spatial x and y coordinates, we can have different "channels"

input: $a^{(0)}$

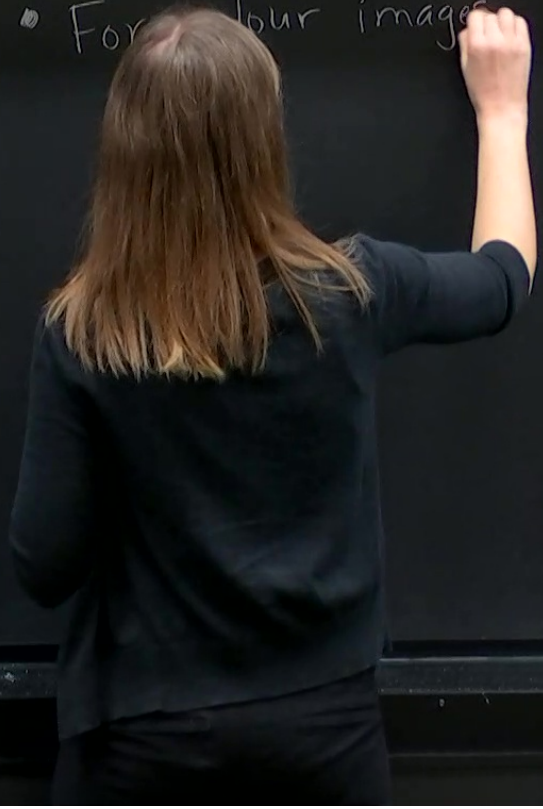
x, y, c

spatial indices

channel index

Channels:

- For colour images



addition

coordinates,

"channels"

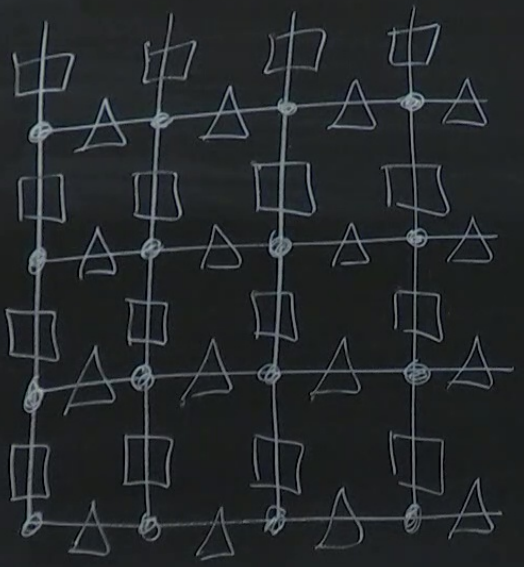
Channels:

- For colour images, c can label the R, G, B channels
- For spin configs on lattices, c can label the sublattices

Each filter is then size $L_{patch} \times L_{patch} \times N_c$

(N_c: # of channels)

For example, recall the $\mathbb{Z}_2$ Ising gauge theory in 2d:

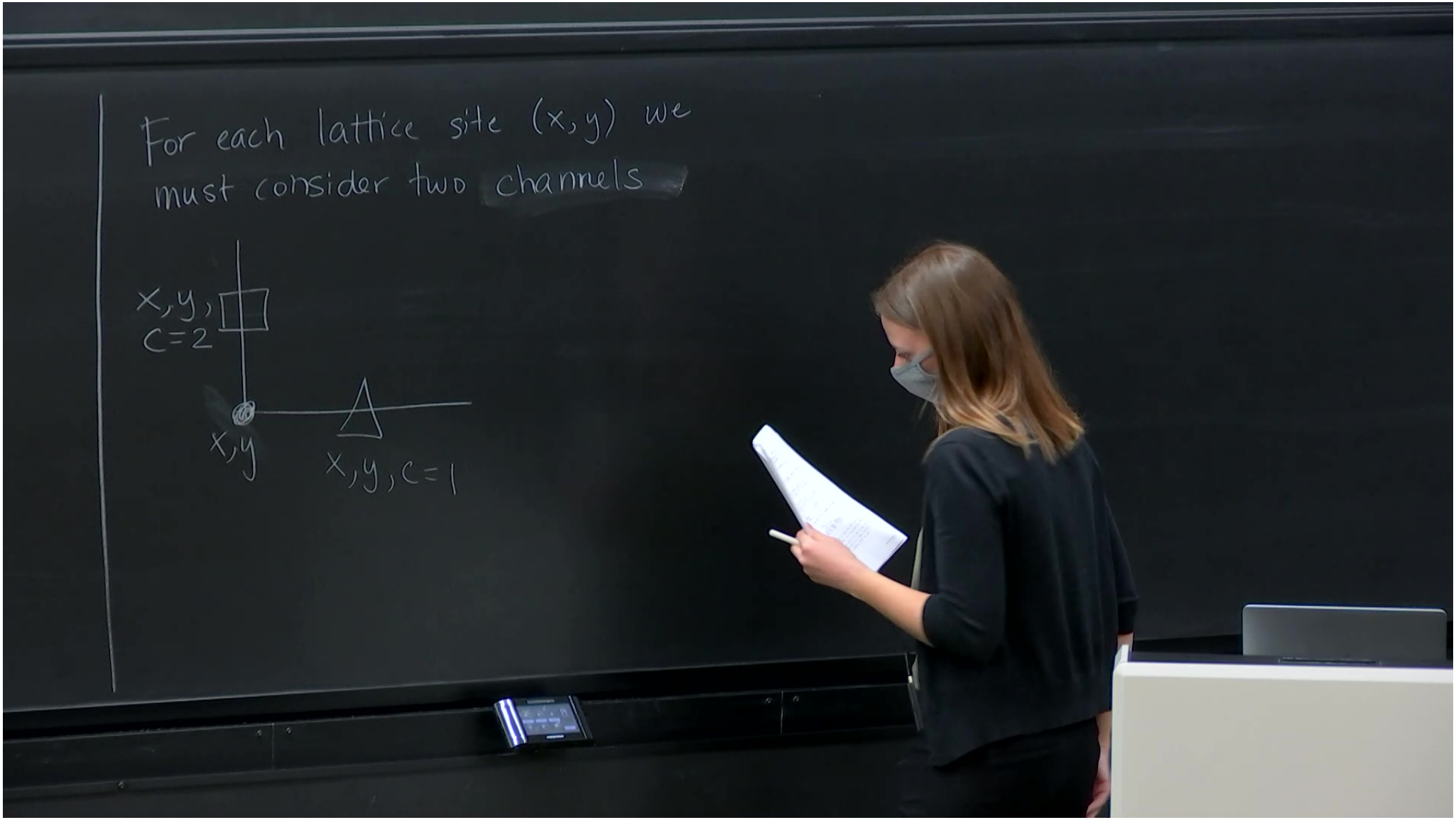


● : lattice sites

△ : sublattice 1

□ : sublattice 2

For each lattice site (x, y


For each lattice site (x,y) we must consider two channels
x,y, c=2
x,y
x,y,c=1

Shared weights and biases

Idea: the LRF moves across the input data (x and y directions) and the weights and biases associated with the filters do not change with x and y

$\Rightarrow$ the transformation applied by the convolutional layer is translationally invariant

(Compare with the coupling J being constant for an Ising model $H = -J \sum_{\langle i,j \rangle} s_i s_j$)

At layer l, we can apply N_f different filters of size $L_{patch} \times L_{patch} \times N_c$ to the output $a^{(l-1)}$ from the previous layer so that:

$$a^{(l)}_{x,y,k} = g_l\left(\sum_{m=1}^{L_{patch}} \sum_{n=1}^{L_{patch}} \sum_{c=1}^{N_c} a^{(l-1)}_{x+m-1,\, y+n-1,\, c}\, W_{mnck} + b^{(l)}_k\right)$$

g_l: activation function

$\sum \sum \sum \; a \; W$: convolution

$b^{(l)}_k$: bias

where we take square patches and stride $s=1$ for simplicity here

(Compare with the coupling J being constant for an Ising model $H = -J\sum_{\langle ij \rangle} s_i s_j$) | where we take squa

The weight indices are:

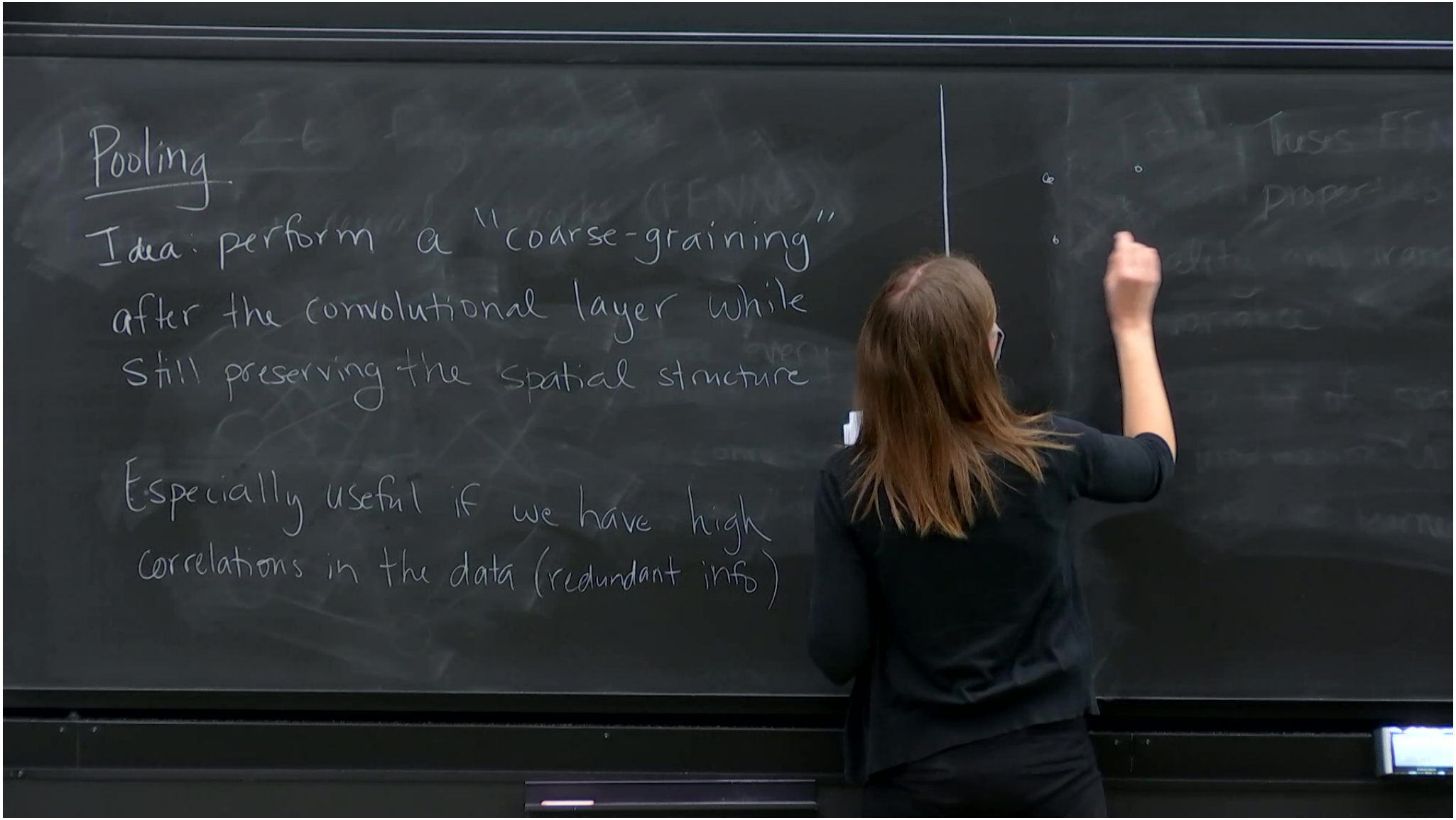
$W_{m\ n\ c\ k}$

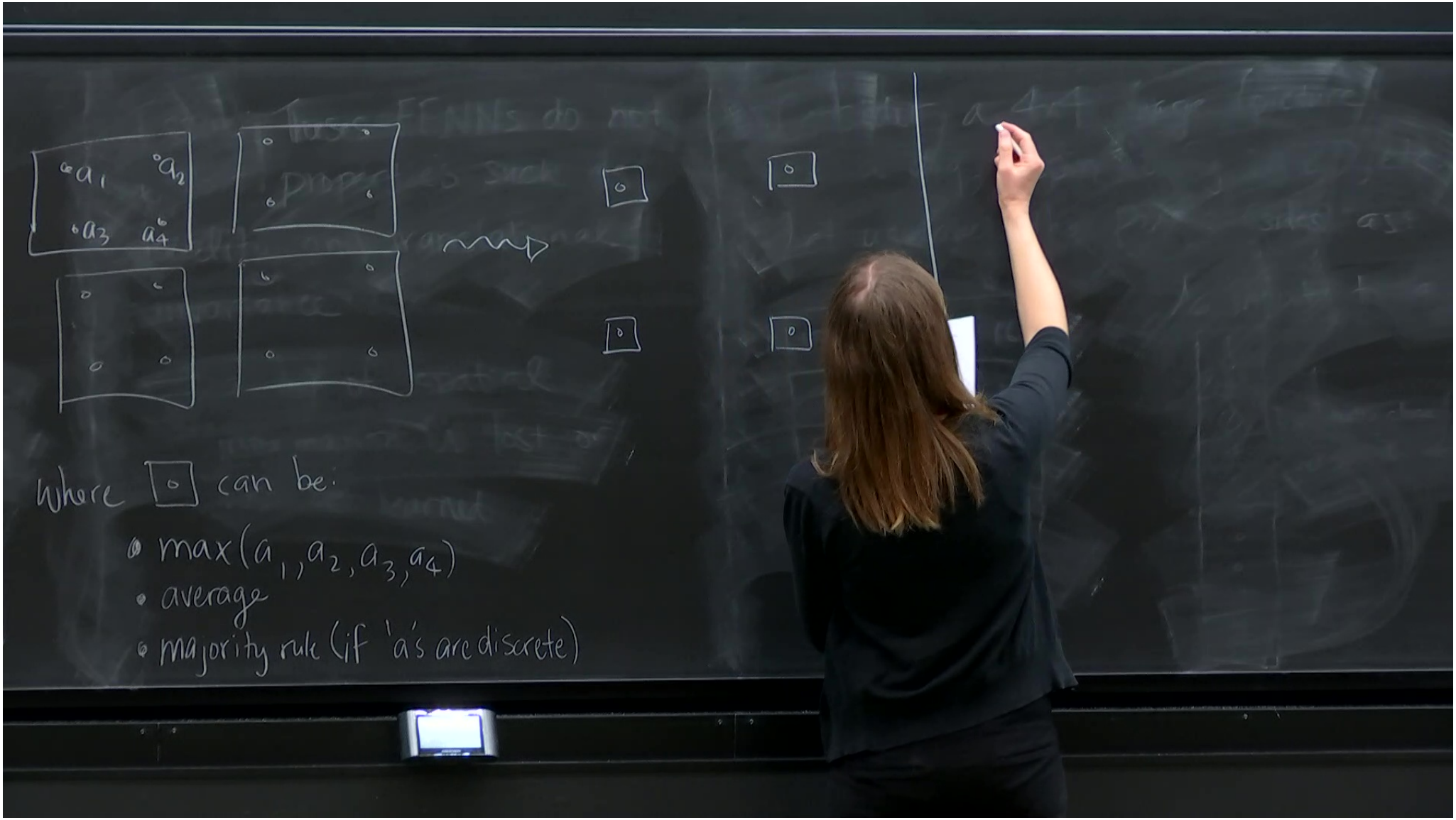
- m: slides along x
- n: slides along y
- c: Channel/sublattice index
- k: labels the different filters $(1 \leq k \leq N_f)$

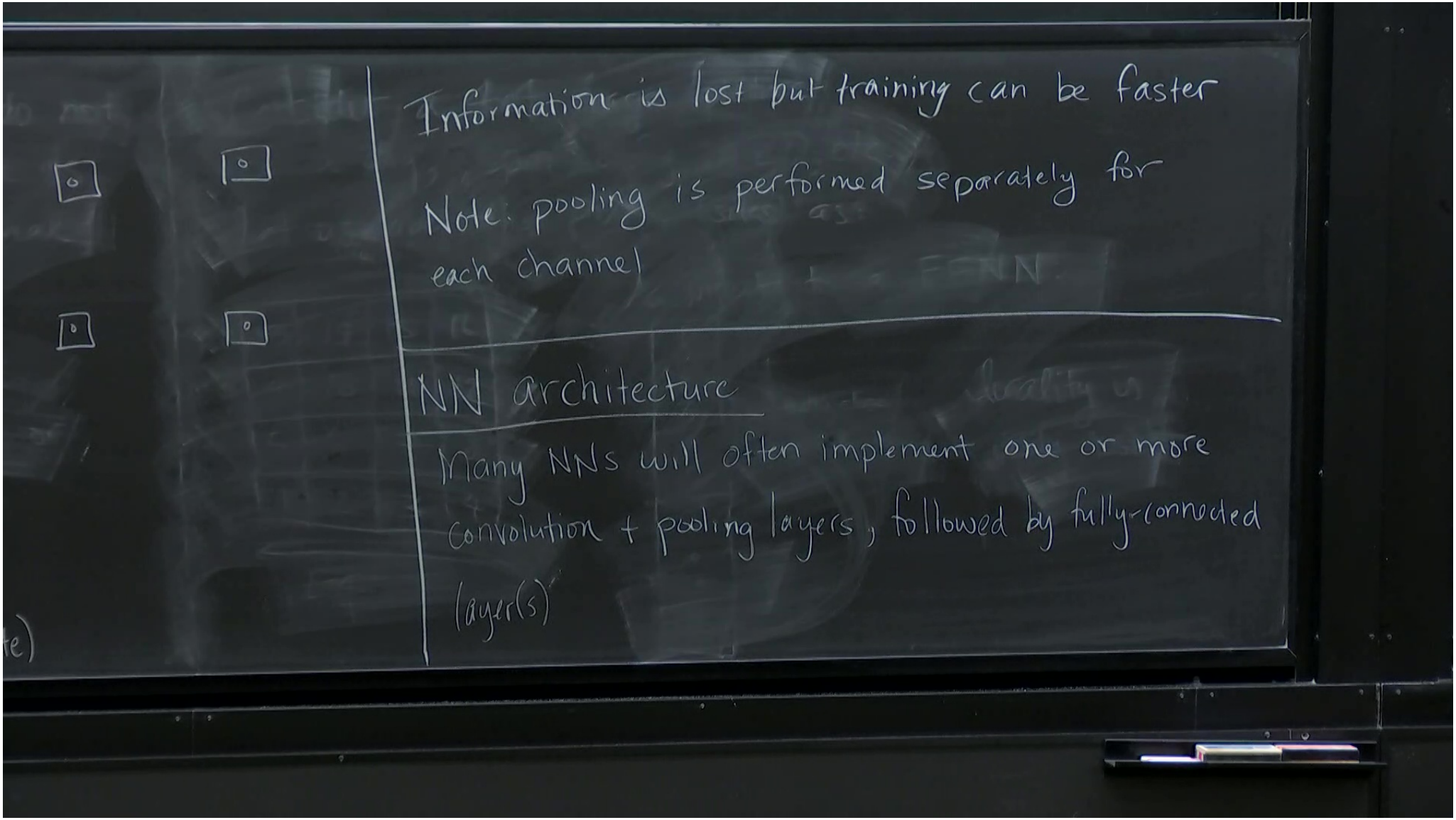
$H = -J \sum_{\langle i,j \rangle} s_i s_j$ | where we take square patches and stride $s=1$ for simplicity here

With shared weights & biases, CNNs tend to have far fewer free parameters to learn during training compared to FFNNs

$\Rightarrow$ it is often possible to train CNNs for larger input and deeper layers than FFNNs (assuming the data is translationally invariant and local)


Pooling
Idea: perform a "coarse-graining" after the convolutional layer while still preserving the spatial structure
Especially useful if we have high correlations in the data (redundant info)


a_1
a_2
a_3
a_4
Where □ can be:
• max(a_1, a_2, a_3, a_4)
• average
• majority rule (if 'a's are discrete)


Information is lost but training can be faster
Note: pooling is performed separately for each channel
NN architecture
Many NNs will often implement one or more convolution + pooling layers, followed by fully-connected layer(s)