Convolutional Neural Networks

PHYS591000 2022.04.13

Announcement

- It is possible that we need to go online (remote teaching) at some point this semester due to the pandemic.

- If so, we’ll use Google Meet.

- Please check your email regularly for the rest of the semester for any sudden change of schedule.

Warming up

- As usual, take 3 mins to introduce yourself to your teammate for this week!
  – “How was your spring break?”
  – “Have you decided on the final project?”

Outline

- Convolution neural network (CNN) is particularly useful for tasks related to images. Why? How does it work?

- The example for today's lecture is taken from this video of the youtube channel 'StatQuest'. You are strongly encouraged to check out this extraordinary channel of statistical science and machine learning.

DNN from Week 06

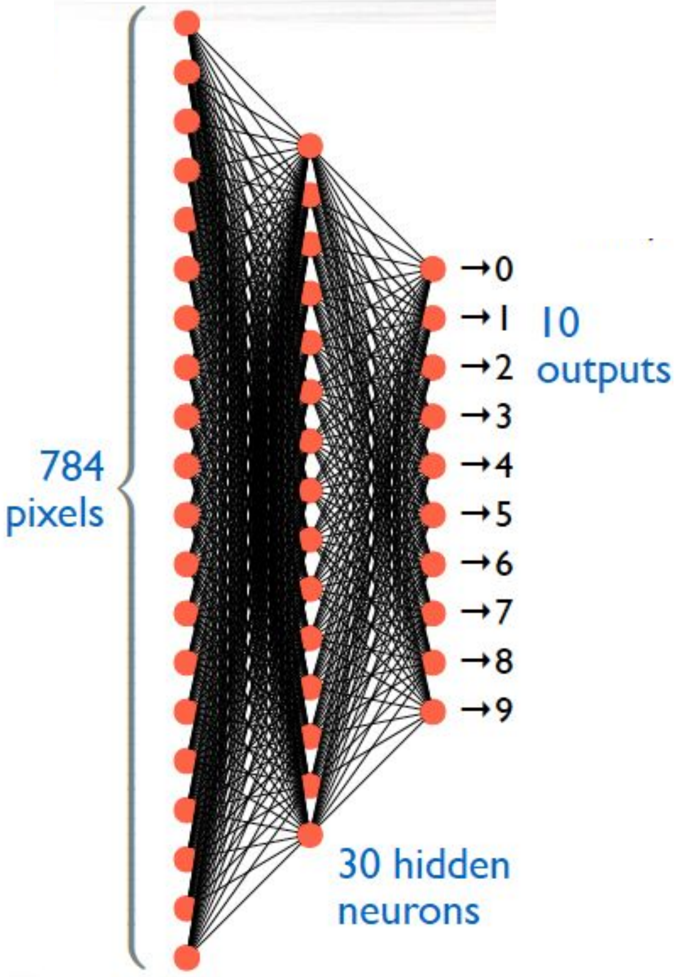
- We'll build a 784-30-10 NN:

# of biases:
0 (no biases for input)+
30 (hidden neurons)+
10 (output neurons).

# of weights:
784×30 (input to hidden)+
30×10 (hidden to output).

23860 parameters in total

Lots of parameters to be optimized in the training!



DNN v.s. CNN for image classification

- Before we take every pixel as an independent input → each pixel has a weight and a bias for each neuron in the next layer.

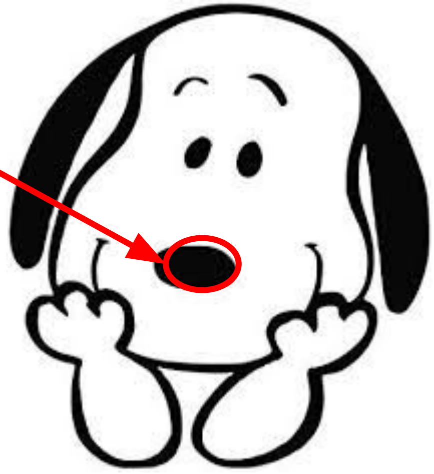
- When the NN becomes deeper and deeper (more and more hidden layers) the parameters will explode, and takes a lot of time for training.

  → Need a different NN structure.

DNN v.s. CNN for image classification

- Another useful feature that is ‘lost’ in the DNN approach is the information on the *correlations* among pixels in an image:

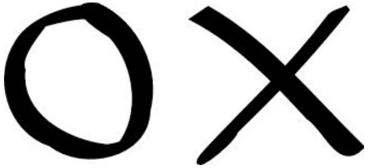
Typically black pixels are near other black pixels.

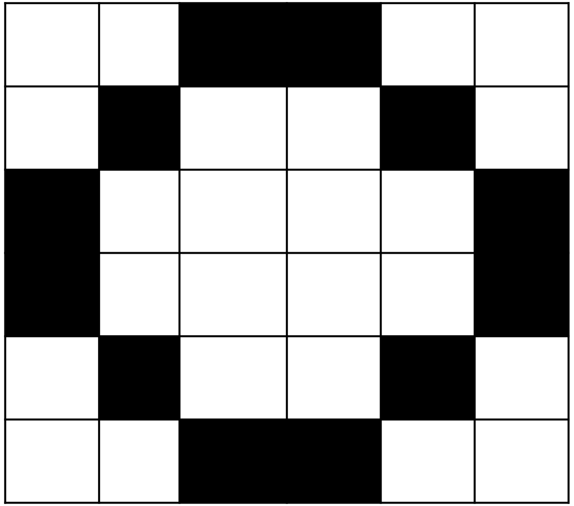


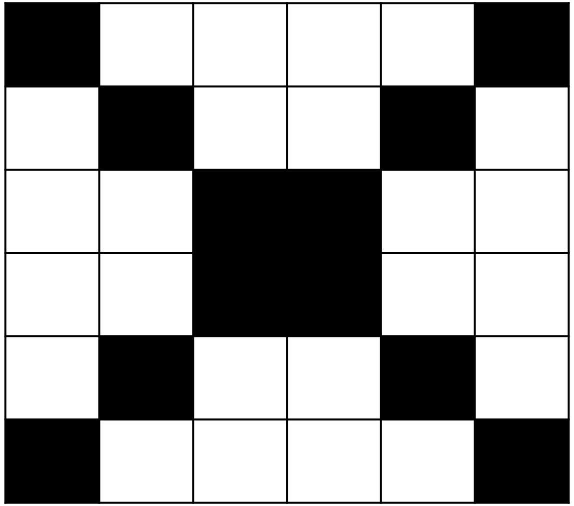
There are certain patterns of ‘edges’ between black and white pixels (e.g. for a face).

CNN for Image Classification

- Task: Classify images of O and X
    - Each image is 6x6 pixel







CNN for Image Classification

- Each image is 6x6 pixel

  Black → 1, white → 0

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1

CNN Filter (Kernel)

- CNN applies a filter (kernel) to the input image

Input

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

Filter (kernel)

0	0	1
0	1	0
1	0	0

Example: a 3x3 filter (9 weights); the weights are determined by training (i.e. **backpropagation**).

CNN Filter (Kernel)

- Overlay the filter on the image and take the dot product

Input

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

Filter (kernel)

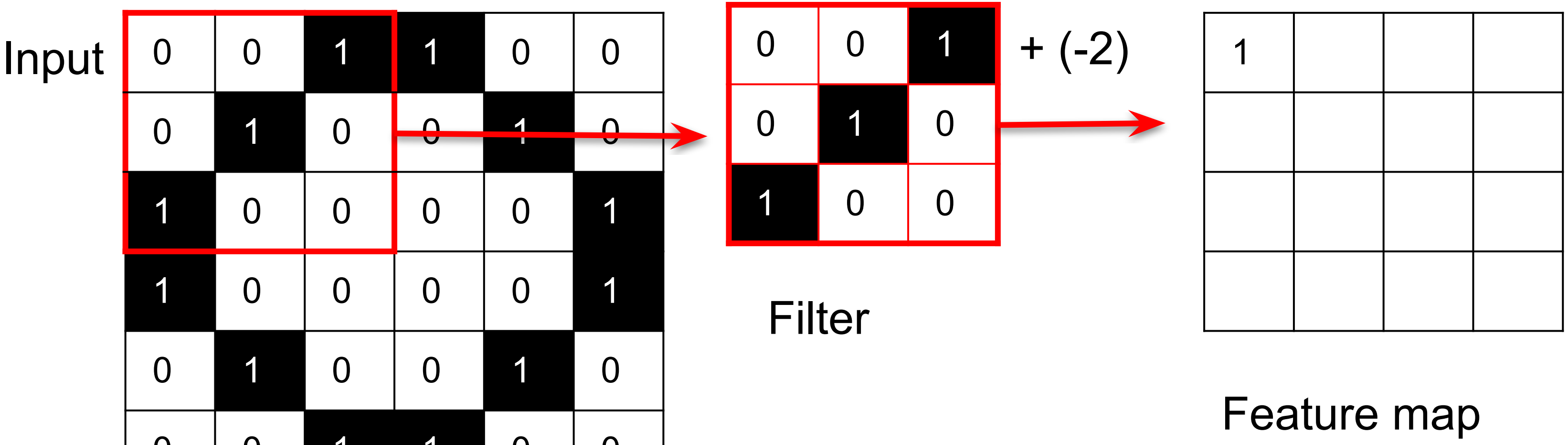
0	0	1
0	1	0
1	0	0

dot product $= 0x0 + 0x0 + 1x1 + 0x0 + 1x1 + 0x0 + 1x1 + 0x0 + 0x0 = 3$

CNN Filter (Kernel)

- Add a bias term and save this as the first element of a **feature map**

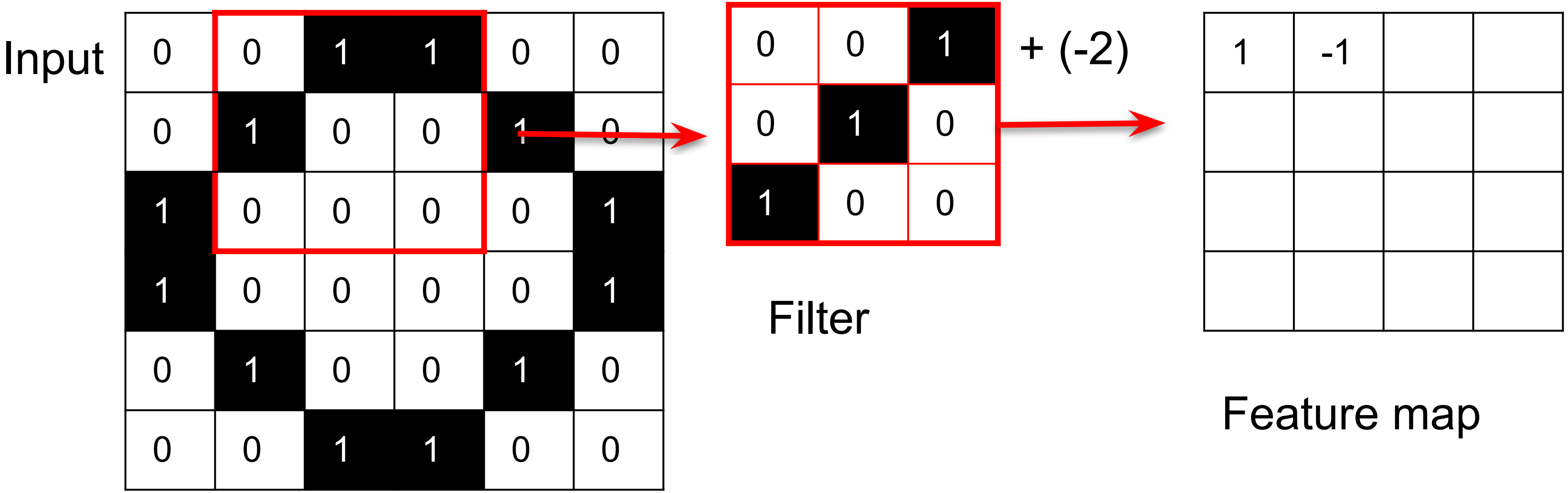
3 (dot product) + -2 (bias) = 1



CNN Filter (Kernel)

- Slide the filter by 1 pixel for the second element in the feature map.

1 (dot product) + -2 (bias) = -1



CNN Filter (Kernel)

- And repeat

0 (dot product) + -2 (bias) = -2

Input

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

Filter

0	0	1
0	1	0
1	0	0

+ (-2)

Feature map

1	-1	-2	

CNN Filter (Kernel)

- Same for the second row of the feature map

1 (dot product) + -2 (bias) = -1

Input

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

Filter

0	0	1
0	1	0
1	0	0

+ (-2)

Feature map

1	-1	-2	-1
-1			

CNN Filter (Kernel)

- We *convolve* the filter with the input to get the feature map.

dot product

Input

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

Filter (Convolution)

0	0	1
0	1	0
1	0	0

+ bias

Feature map

1	-1	-2	-1
-1			

CNN Filter (Kernel)

- The filter shares the same weights and bias for all elements in the feature map. → Reduce parameters.

Input

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

weights

0	0	1
0	1	0
1	0	0

Filter (Convolution)

+ bias →

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1
-1	-2	-1	1

Feature map

CNN Filter (Kernel)

- Each element of the feature map contains information of a group of neighboring pixels → takes correlations into account.

Input

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

Filter (Convolution)

0	0	1
0	1	0
1	0	0

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1
-1	-2	-1	1

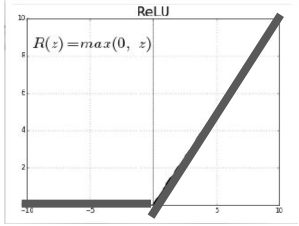
Feature map

CNN Filter (Kernel)

- Feed the feature map into an activation function, say ReLU.

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1
-1	-2	-1	1

Feature map



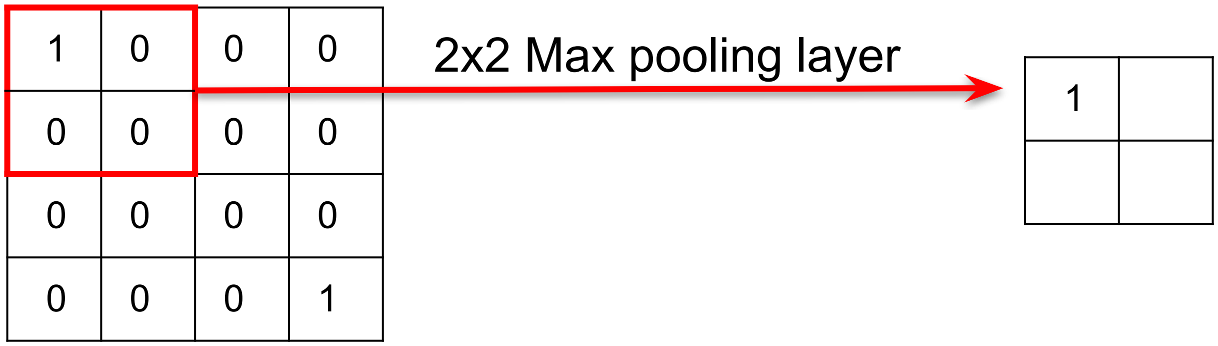
ReLU

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

Post-ReLU feature map

CNN Pooling

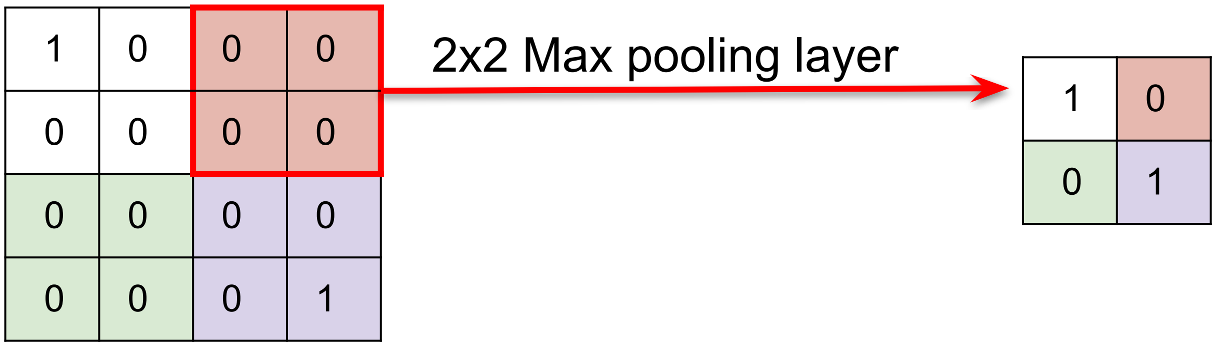
- Apply a pooling to the new feature map, e.g. **Max Pooling** which selects the maximal input



Post-ReLU feature map

CNN Pooling

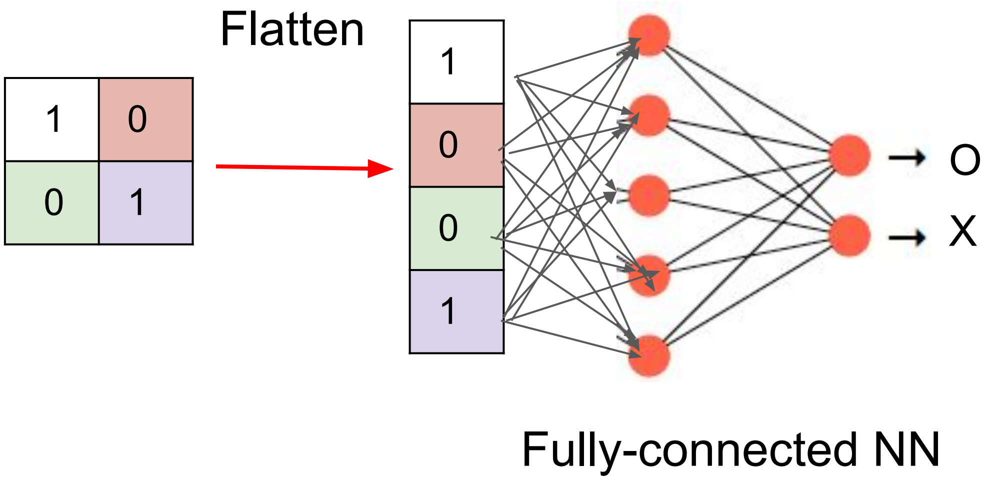
- Pooling moves in a way that's *not* overlapping



Post-ReLU feature map

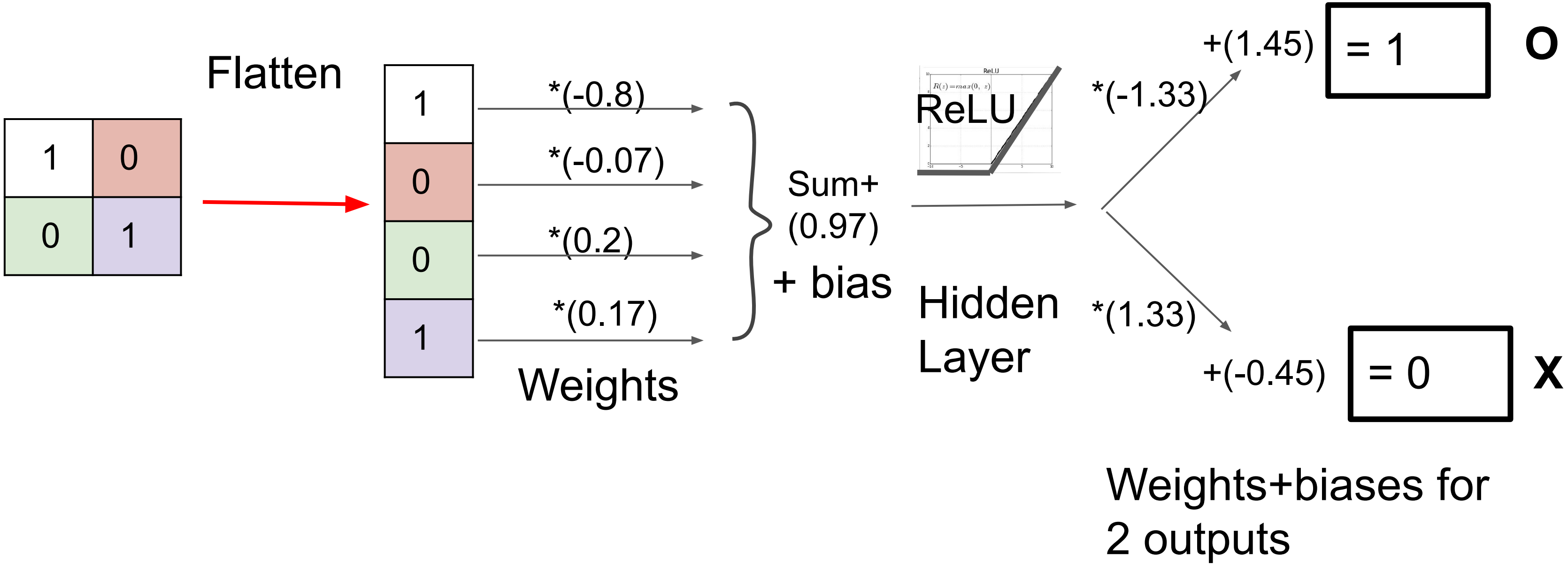
CNN Flattening and fully-connected layers

- **Flatten** the pooled layer into a 1D array and feed it into a fully-connected NN for the classification



CNN Flattening and fully-connected layers

- **Flatten** the pooled layer into a 1D array and feed it into a fully-connected NN for the classification



CNN for Image Classification

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1

0	0	1
0	1	0
1	0	0

+ (-2)

Filter

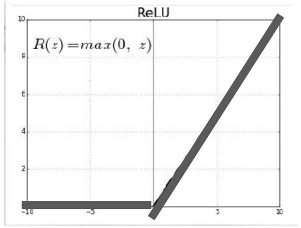
-1	-2	-1	1
-2	-1	1	-1
-1	1	-1	-2
1	-1	-2	-1

Feature map

CNN for Image Classification

Post-ReLU Feature map

0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0



ReLU

-1	-2	-1	1
-2	-1	1	-1
-1	1	-1	-2
1	-1	-2	-1

Feature map

Max Pooling

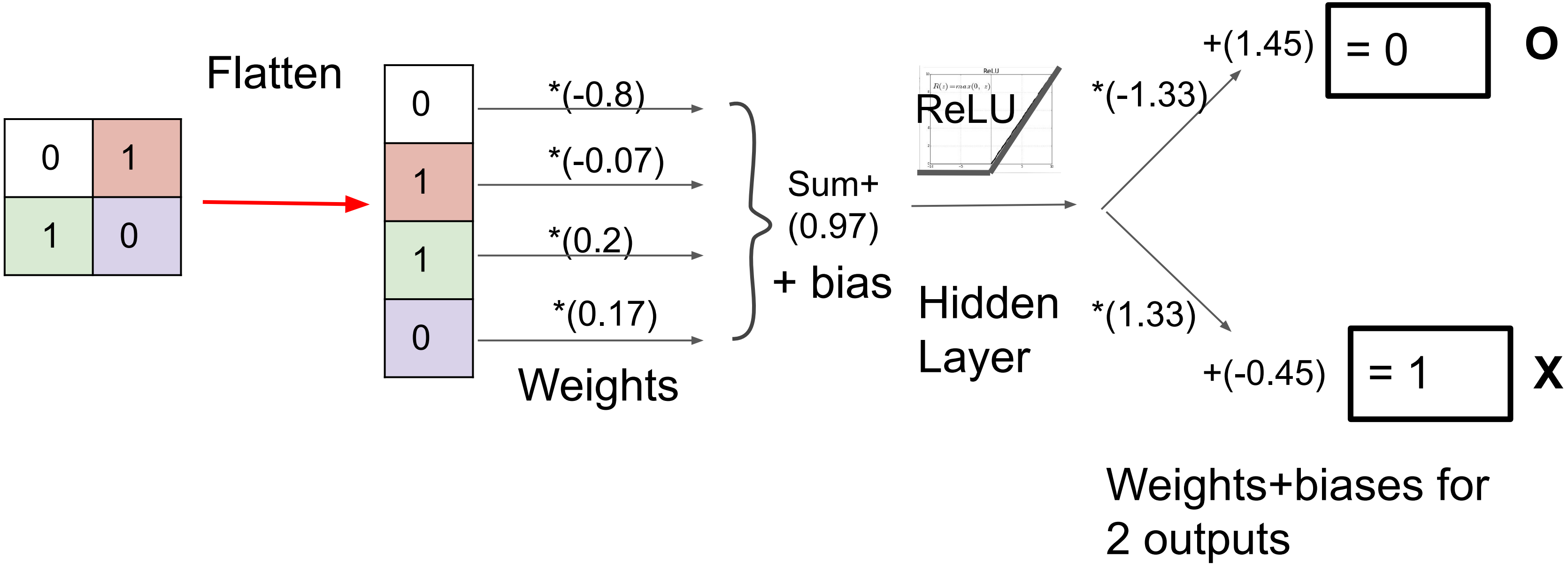
0	1
1	0

Flatten

0
1
1
0

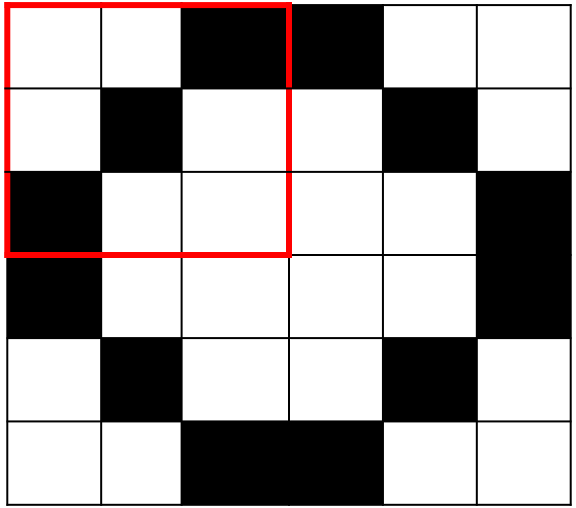
CNN Flattening and fully-connected layers

- **Flatten** the pooled layer into a 1D array and feed it into a fully-connected NN for the classification

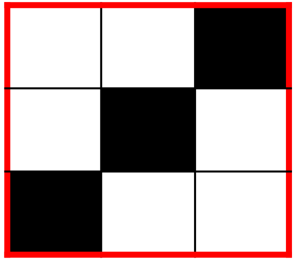


How CNN works

- To see why CNN is powerful, note that the (max) pooled layer selects the parts where the filter matches the input the best.



Input image



Filter

1	0
0	1

Max-pooled layer

CNN Pooling

- Another choice is **Average Pooling** which takes the average of each region

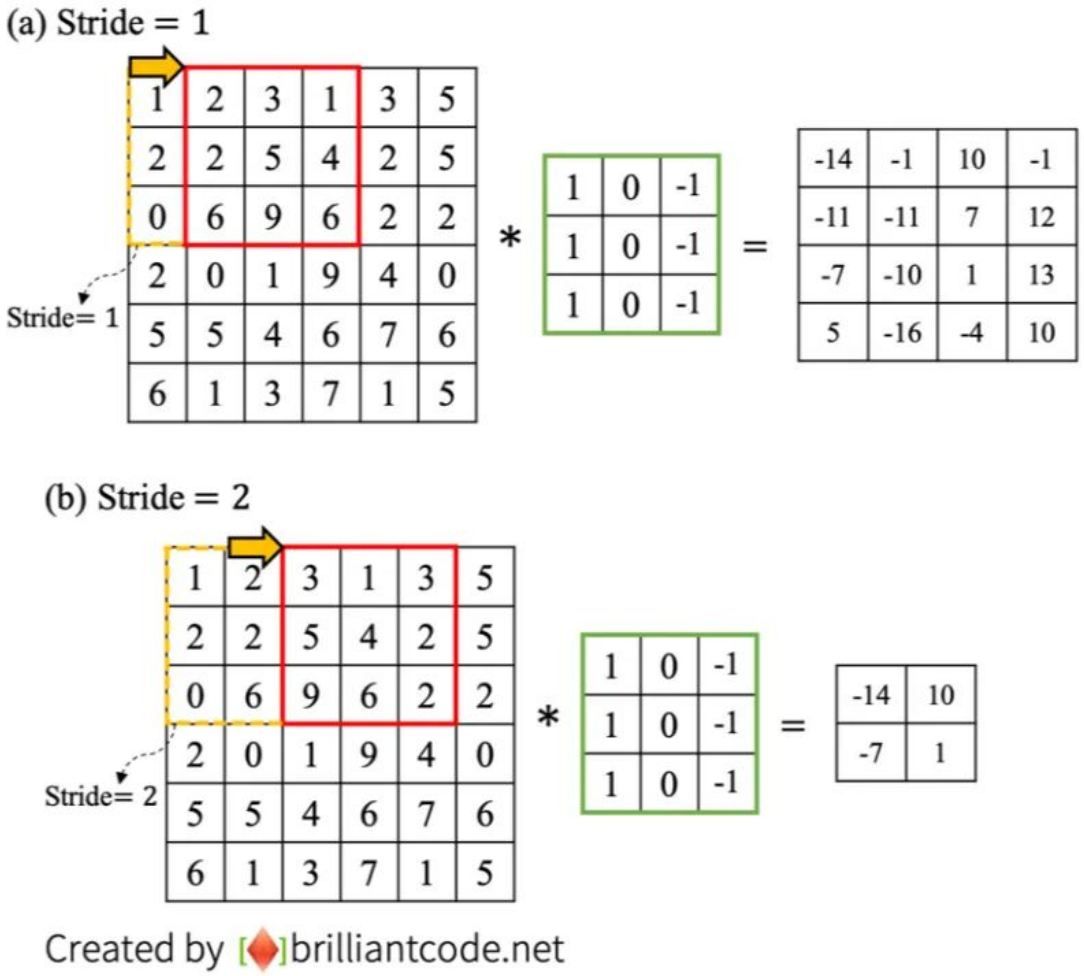
1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1

Average pooling layer

0.25	0
0	0.25

Post-ReLU feature map

CNN Strides


(a) Stride = 1
Stride= 1
(b) Stride = 2
Stride= 2
Created by [◆]brilliantcode.net

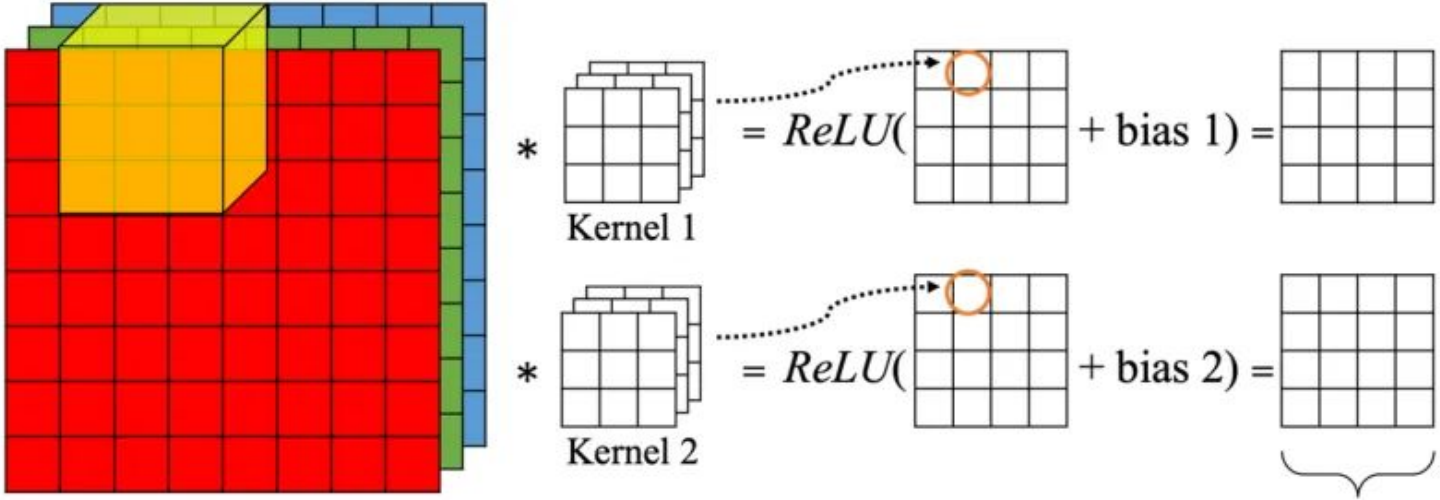
CNN Padding

- We can apply padding to allow more space for the filter to cover the image and preserve the size of feature maps.

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 3 & 1 & 3 & 5 & 0 \\
0 & 2 & 2 & 5 & 4 & 2 & 5 & 0 \\
0 & 0 & 6 & 9 & 6 & 2 & 2 & 0 \\
0 & 2 & 0 & 1 & 9 & 4 & 0 & 0 \\
0 & 5 & 5 & 4 & 6 & 7 & 6 & 0 \\
0 & 6 & 1 & 3 & 7 & 1 & 5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1
\end{bmatrix}
=
\begin{bmatrix}
-4 & -5 & -1 & 3 & -5 & 5 \\
-10 & -14 & -1 & 10 & -1 & 7 \\
-8 & -11 & -11 & 7 & 12 & 8 \\
11 & -7 & -10 & 1 & 13 & 13 \\
-6 & 5 & -16 & -4 & 10 & 12 \\
-6 & 4 & -7 & -1 & 2 & 8
\end{bmatrix}
$$

CNN image channel

- A color image has three channels (R/G/B) and thus needs three layers of kernels.



CNN hyperparameters

- Filter/Kernel size (height and width of the kernel)
- Strides and padding
- Data format/channel numbers
- Ways of pooling
- Other hyperparameter for the (fully-connected) NN, e.g. activation functions.

In-Class Exercise and Lab for this week

- For in-class exercise this week we'll use CNN to classify 0-9 with MNIST dataset.

- For the Lab this week, we continue with the same W/Z v.s. QCD jet dataset and use CNN for the classification task.

Backup

Outline

- Why need CNN; problem w/ deep NN

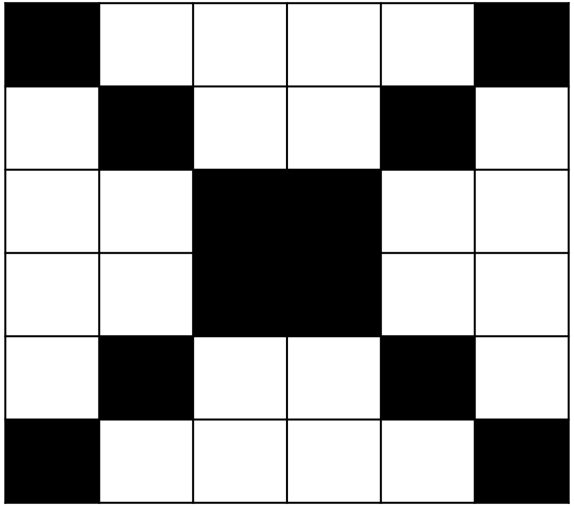
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1

Outline

- Why need CNN; problem w/ deep NN

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0



Outline

0	0	1
0	1	0
1	0	0

0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0