HW2 Part 1

Convolutional Neural Networks with Numpy
Slides kindly made by Aparajith

Resampling

- For loop is not required in *python*
  - *Look up np.kron*
  - *Array slicing:* `[start:end:step]`
- Things to remember
  - Trying to compute the required shape while up sampling (some simple formula you can think of?)
  - Computing and storing the shape in forward.
    - This is because the gradient should be the same shape as the input.

Convolutions

- You can perform convolutions in 2 ways:
  - The Loopy way (Bad)
  - Tensordot (Good)
- The more for loops you use for your questions, the more time it takes to run.
- With tensordot, you don’t have to do all those broadcasting and everything given in the write-up

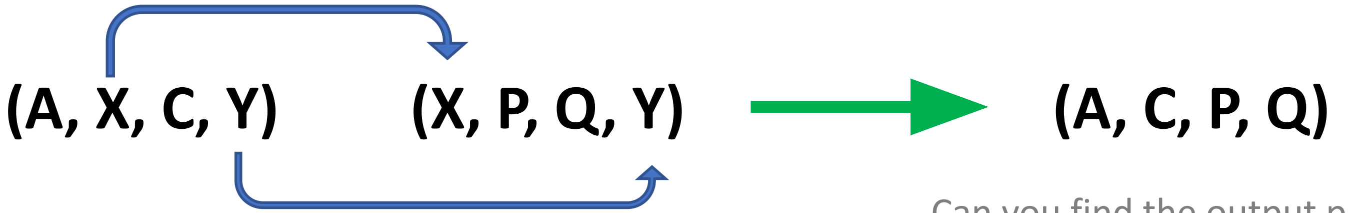
Tensordot

- Ref: https://numpy.org/doc/stable/reference/generated/numpy.tensordot.html
- Appendix of the write up has amazing documentation of it
- Don’t use for loops for convolution even though everything is given in the lecture slides
- Tensordot is faster and helps you (also TAs) to debug easily
- You only need ***1 for loop*** for conv1d and ***2 for loops*** for conv2d. If you are using more, then your implementation of *tensordot* is wrong even if you get the answer right

Tensordot

- Before starting ConvXd.py, open a notebook and try to understand tensordot with random examples
- Consider the shapes:
  - Input: X(A, B, C); Weight: W(P, Q, R)
  - You can do tensordot when you have matched shapes
  - If B = Q and C = R,
    - Tensordot(X, W, matched axes) -> Output(A, P)
    - You can think that the output shape will be the shape of the unmatched axes in that order
  - Make sure inputs (input and weight) to tensordor have some matching axes. Why do you need matching axes in convolution? (Hint: A filter only looks at a segment of input)
- Tip: Print shapes in your code to understand

Tensordot



Can you find the output pattern?

(X, Y) from input 1 matches to (X, Y) from input 2
Can you think in terms of axes?

Should match all the axes that you think needs to be matched. Not restricted to 2 axes

Conv1d to Conv2d

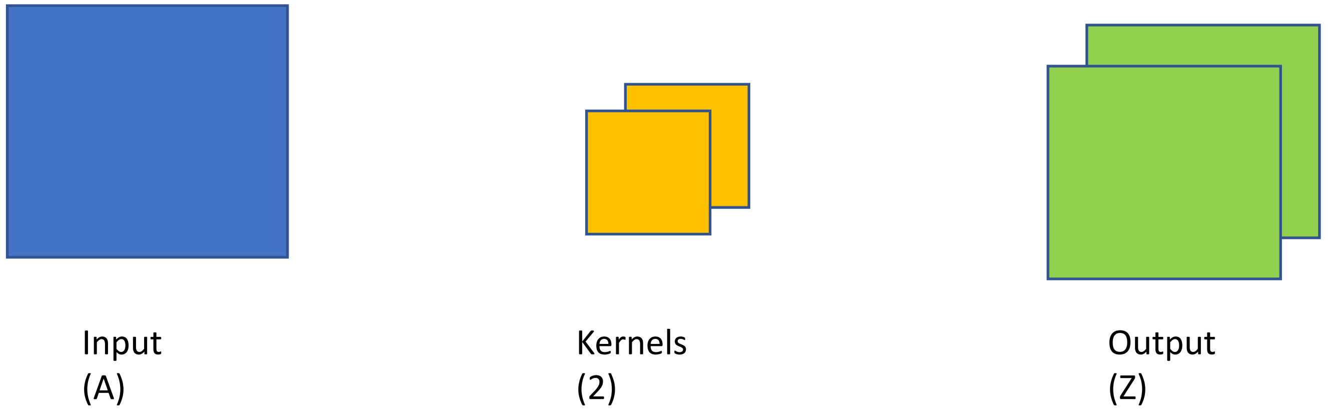
- Try to understand each step while coding conv1d
- Every step between Conv1d and Conv2d (forward and backward) are identical
- While transitioning from Conv1d to Conv2d, you just need to account for the extra dimension and do an **extra something**

Pooling

- Lectures have a basic pseudocode which can be developed
- You might need many loops for this task
    - Np.max and np.unravel_index might be useful if you want to reduce the number of loops
    - But multiple loops are acceptable for this particular task

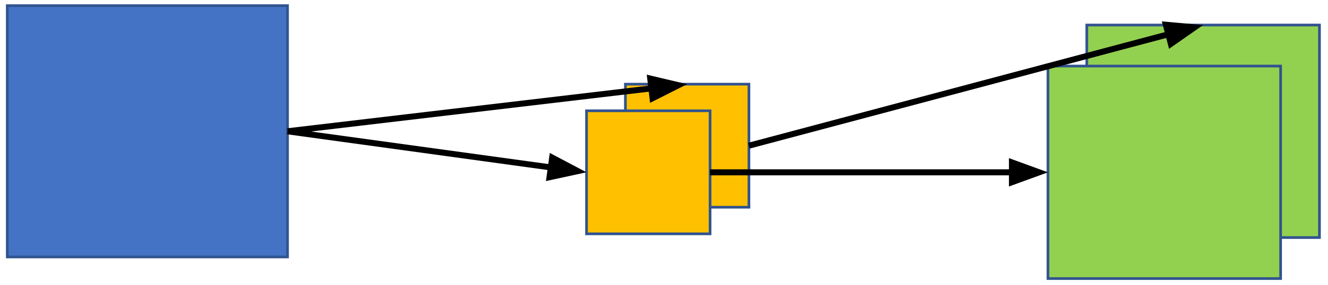
- Backprop in both might is harder than forward, but if you know the concept behind it, it will not be that hard.
- Look at the write up for images.

Easy way to understand gradient propagation



We get 2 maps in backward for dLdZ. After some process for finding dLdA, you again get 2 maps. But A has 1 map and dLdA will also have the same shape. How to understand gradient propagation?

Easy way to understand gradient propagation



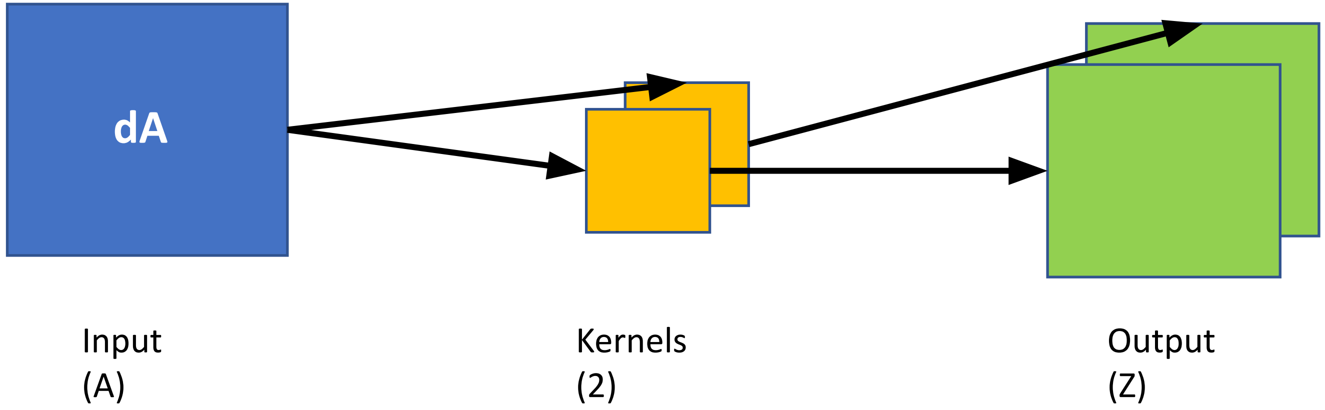
Input
(A)

Kernels
(2)

Output
(Z)

Draw the influence diagram.

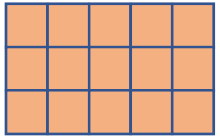
Easy way to understand gradient propagation



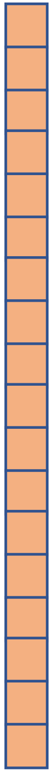
Any small change dA will cause a change in both maps of Z.

Scanning MLP

- Piazza post: https://piazza.com/class/l37uyxe87cq5xn/post/711
- Appendix of HW2P1
- Tips to understand better: Draw everything

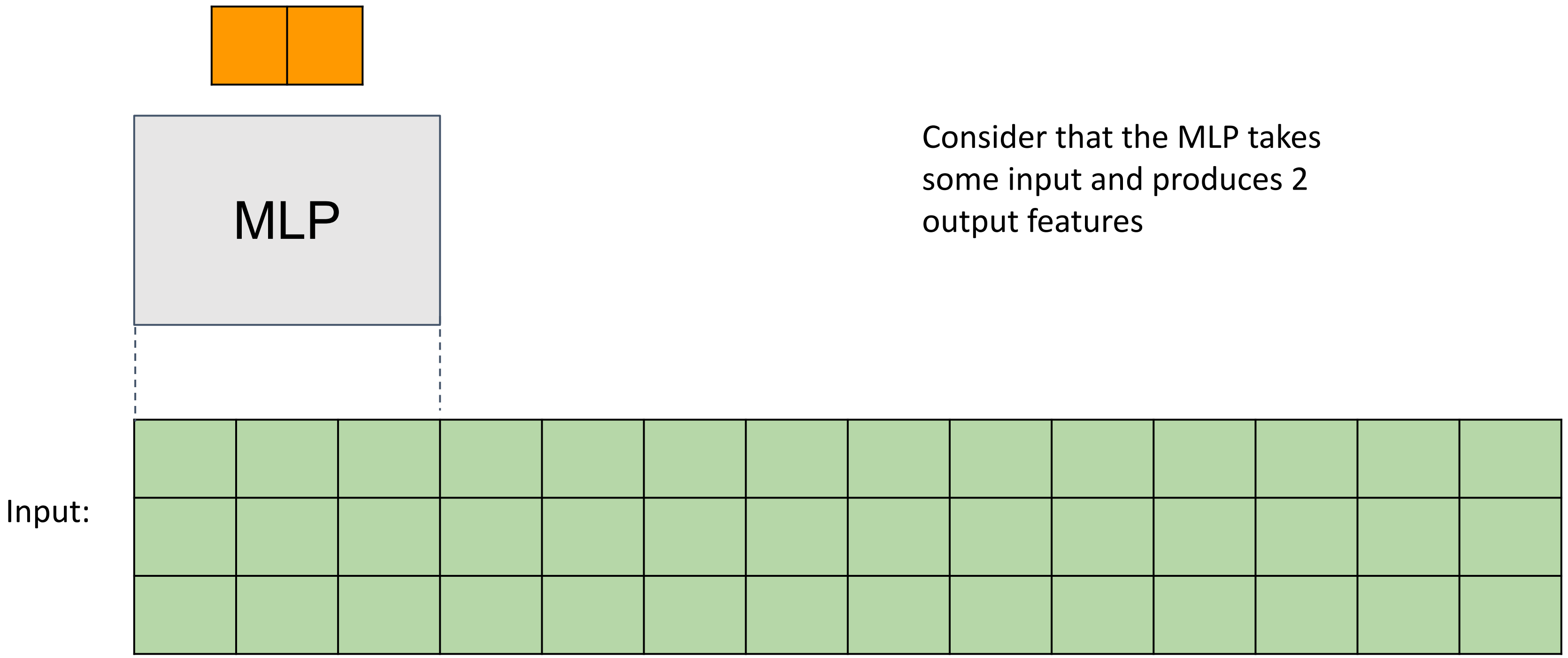


How Conv1d sees the input



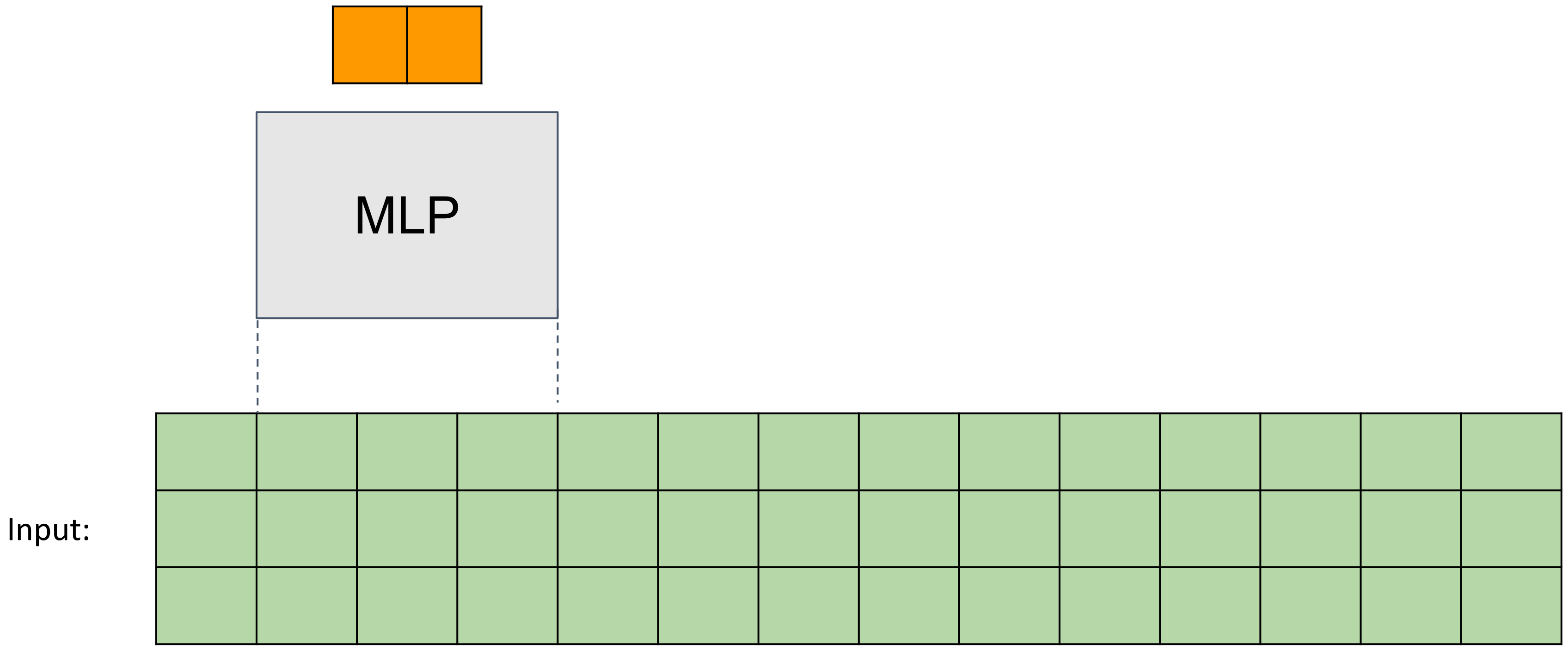
How Linear sees the input

Scanning MLP

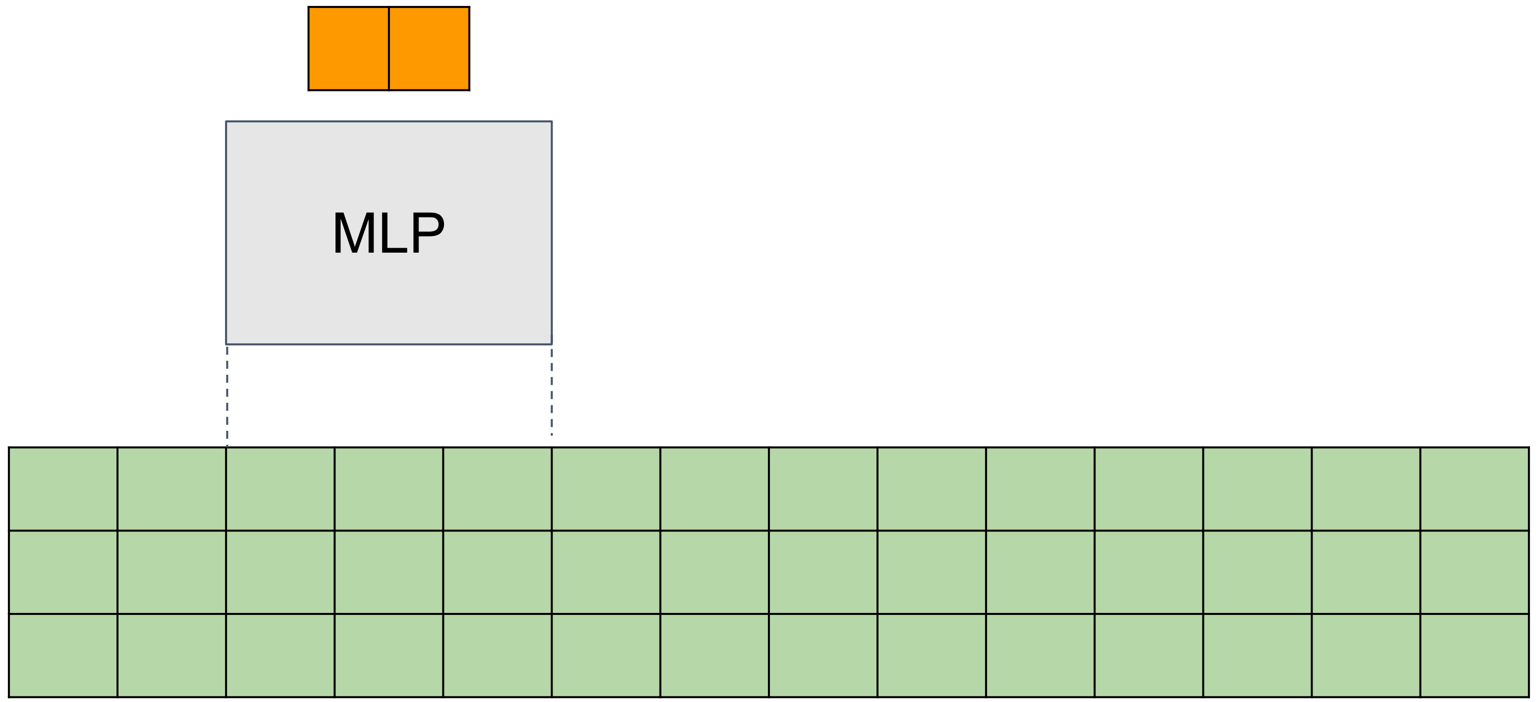


Consider that the MLP takes some input and produces 2 output features

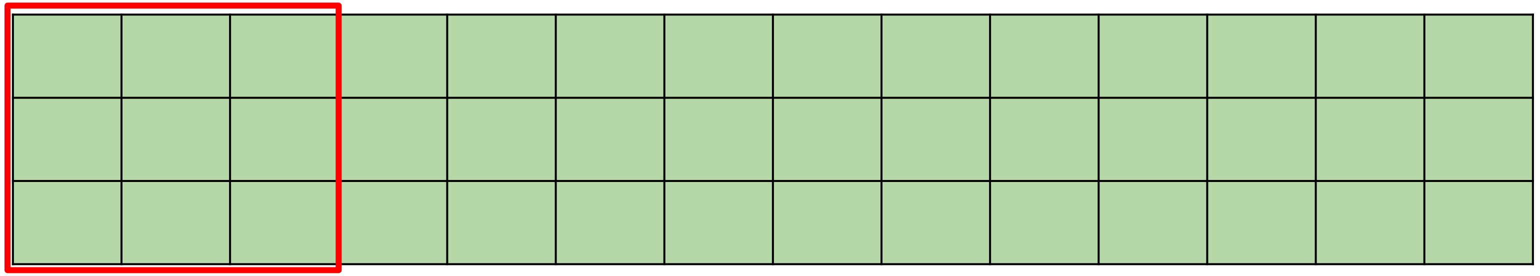
Scanning MLP



Scanning MLP

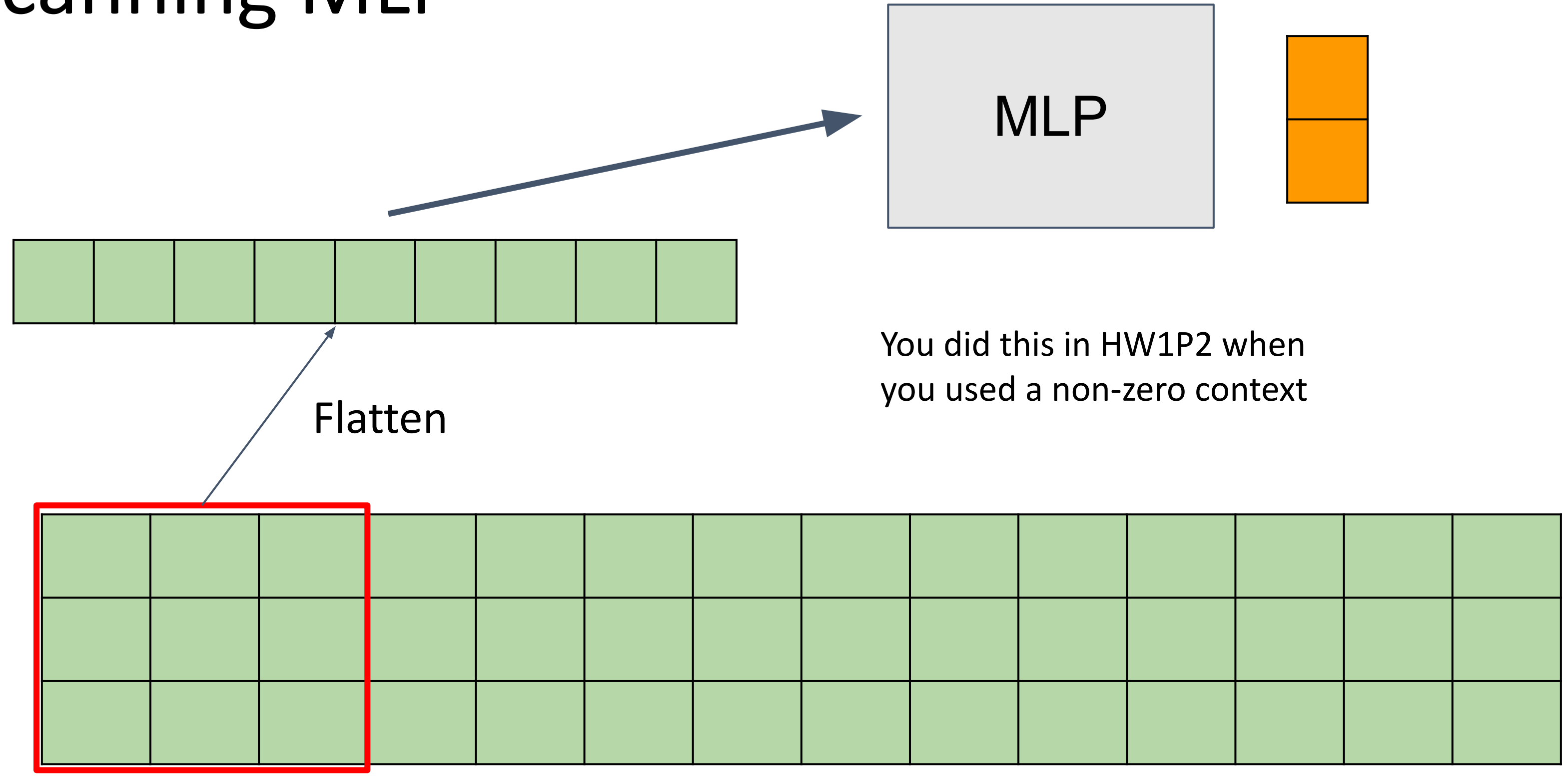


Scanning MLP



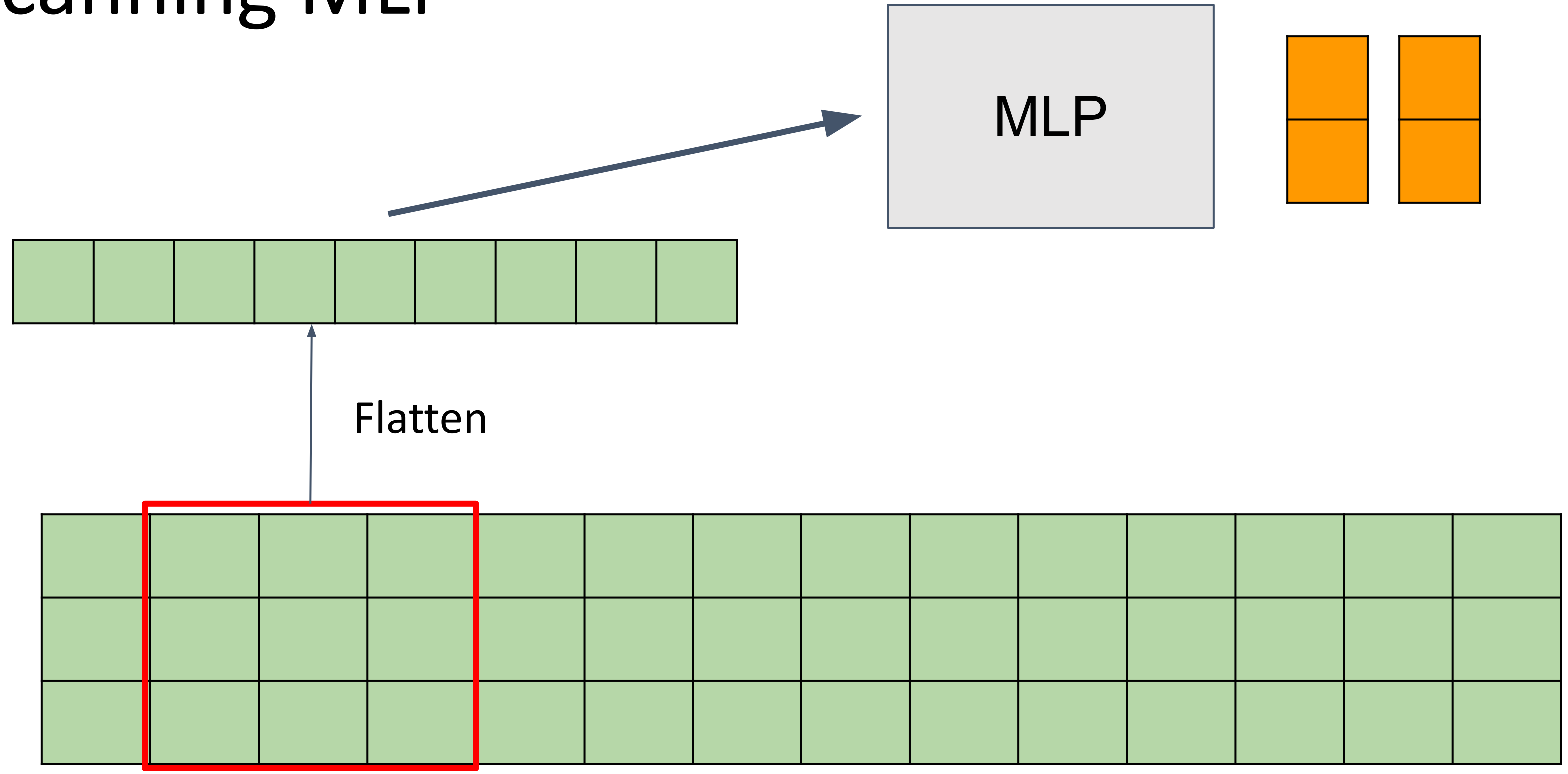
Kernel size=3

Scanning MLP

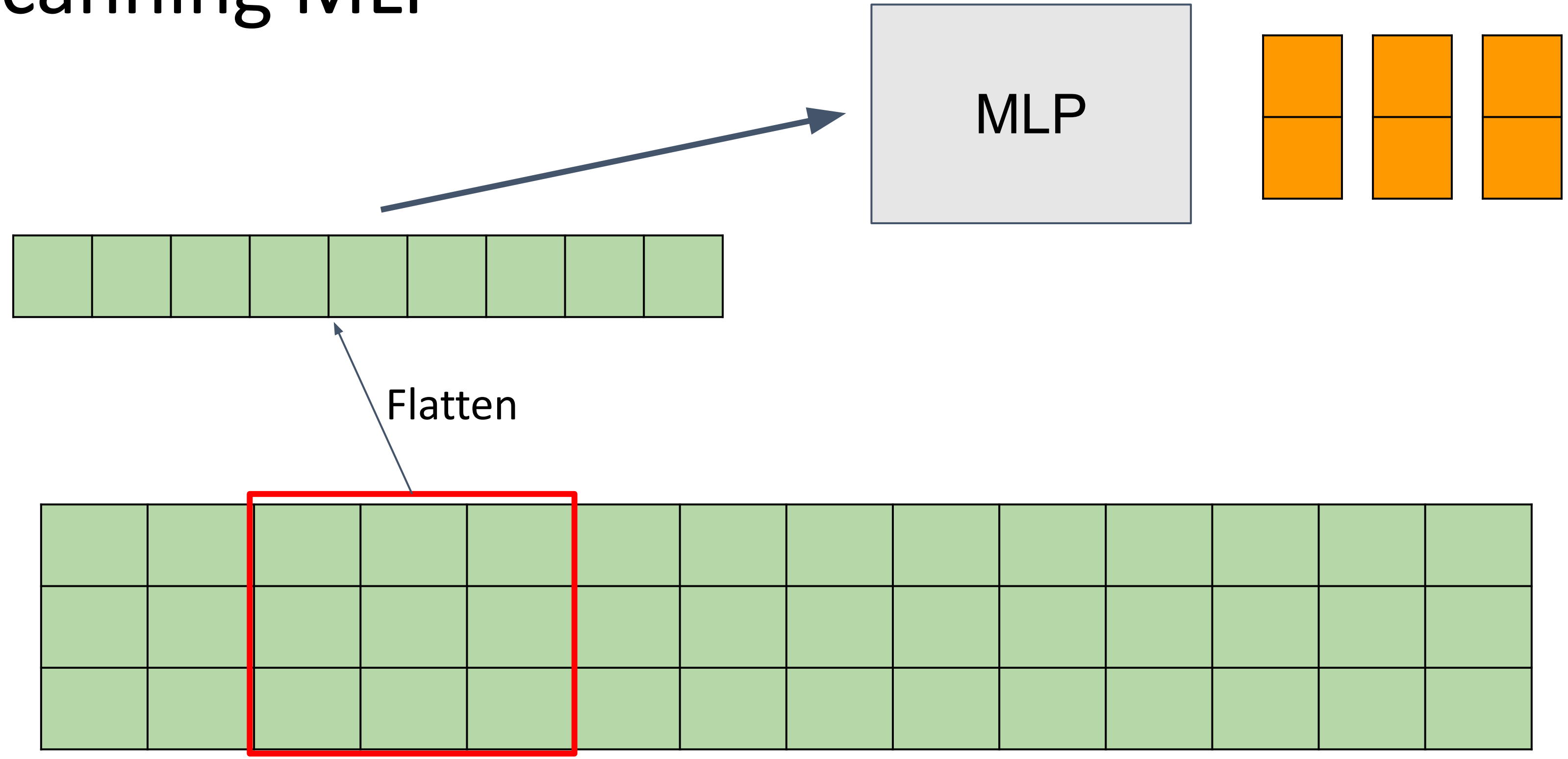


You did this in HW1P2 when you used a non-zero context

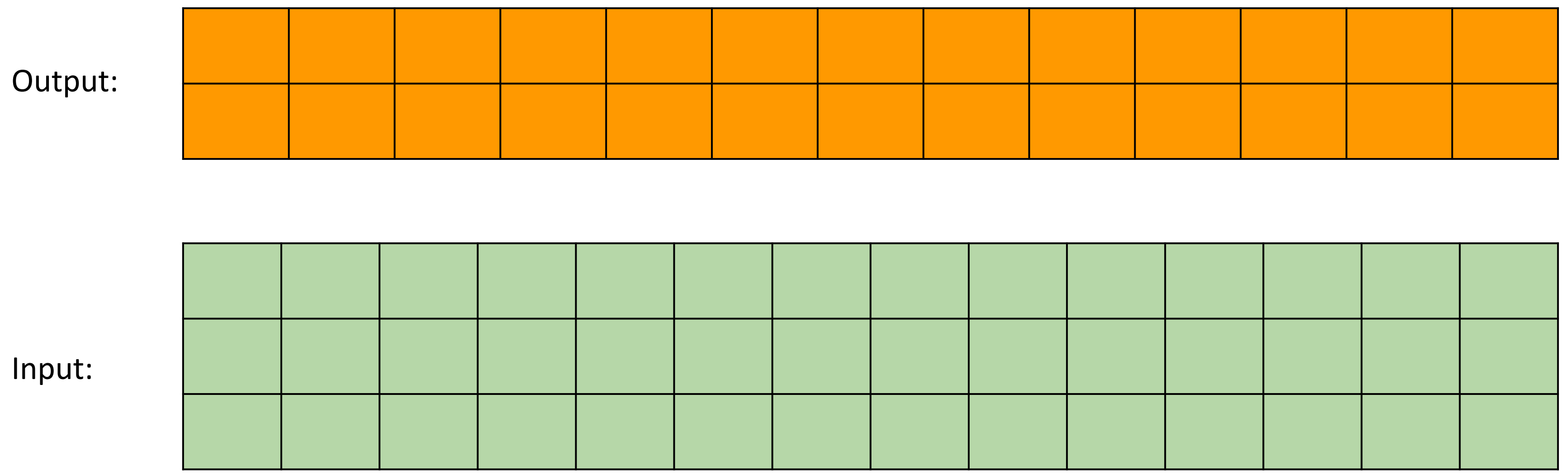
Scanning MLP



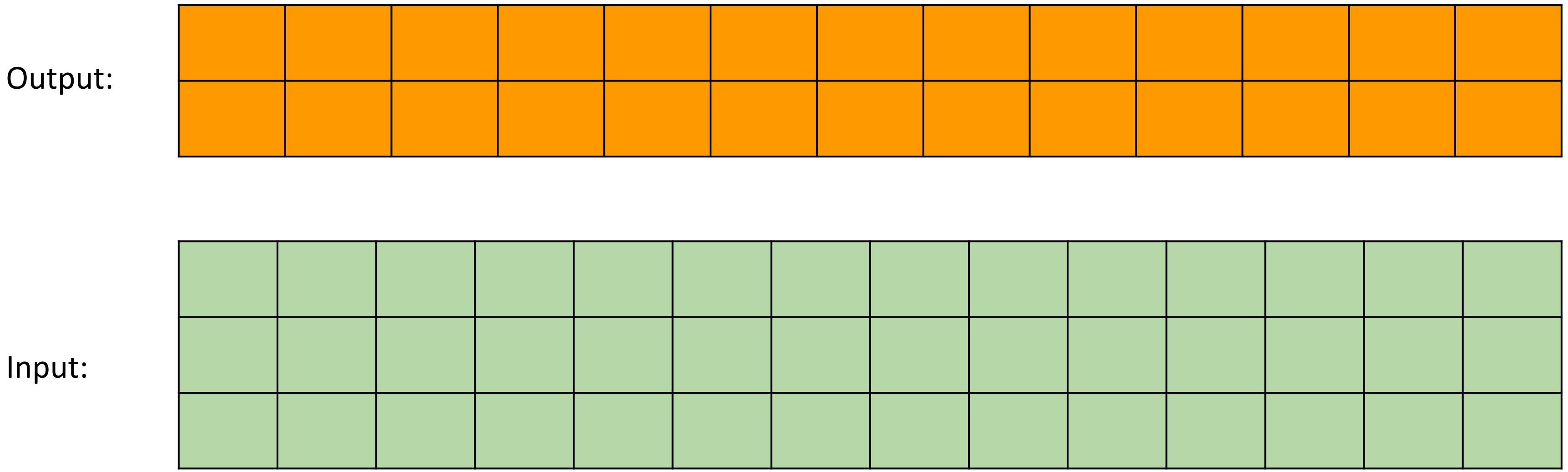
Scanning MLP



Scanning MLP



Scanning MLP



Which gives in_channels = 3, out_channels = 2, kernel_size = 3, stride = 1

We transformed ***Linear(9, 2)*** to ***Conv1d(3, 2, kernel_size= 3, stride= 1)***

CNN Model

- Just calling all the layers which you implemented previously
- Only thing to think about: Initialization size of the final Linear Layer?
- Errors which you may get:
    - If you have a closeness error (*true_divide error*), change to ***np.tanh()***