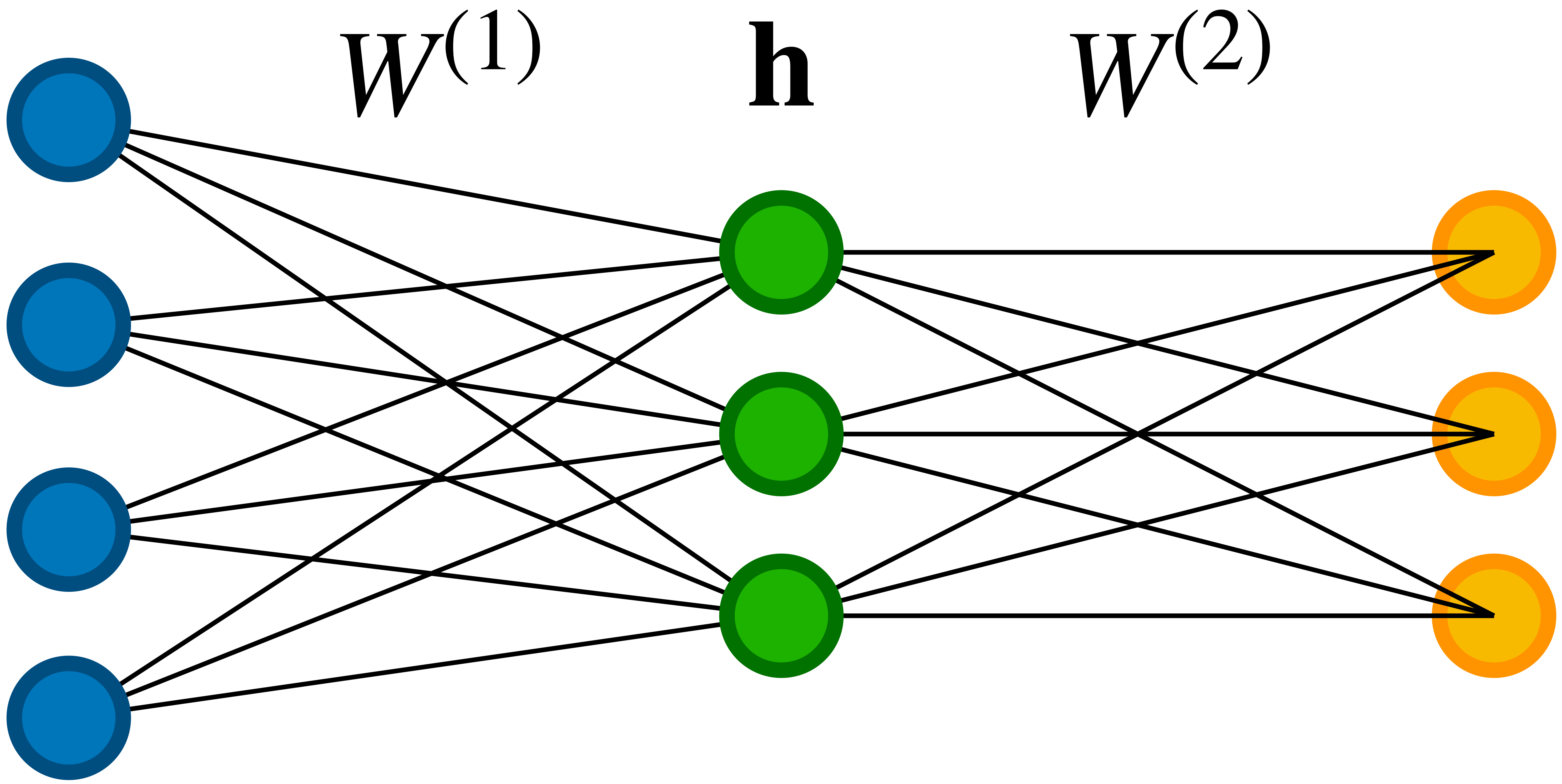
Lab 6: Neural Networks

Matt Ellis and Mike Smith

Feed Forward Neural Network

Input Vector

$$\mathbf{x}$$



Output

$$\mathbf{y}$$

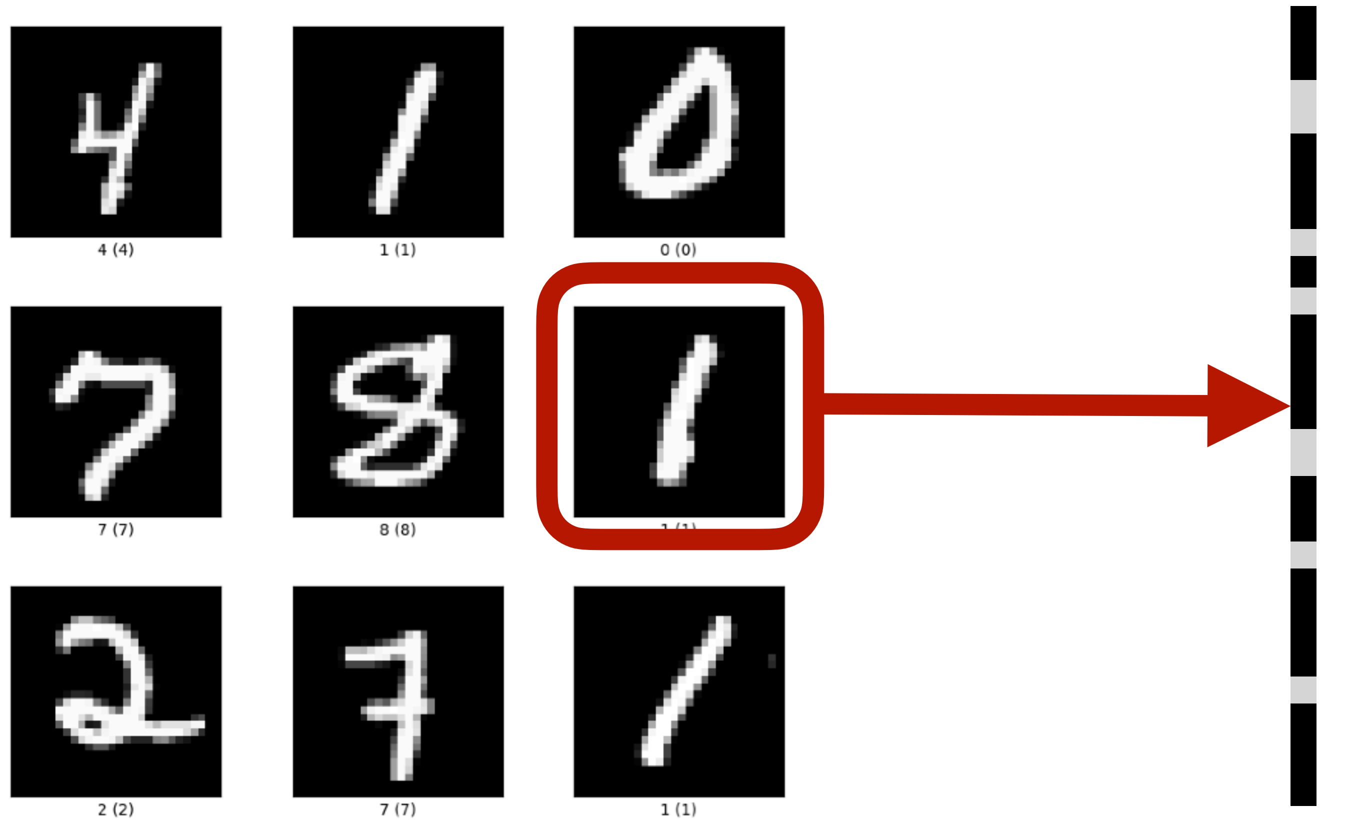
Hidden Layer

$$\mathbf{h} = f\left(W^{(1)}\mathbf{x} + b^{(1)}\right)$$

Output Layer

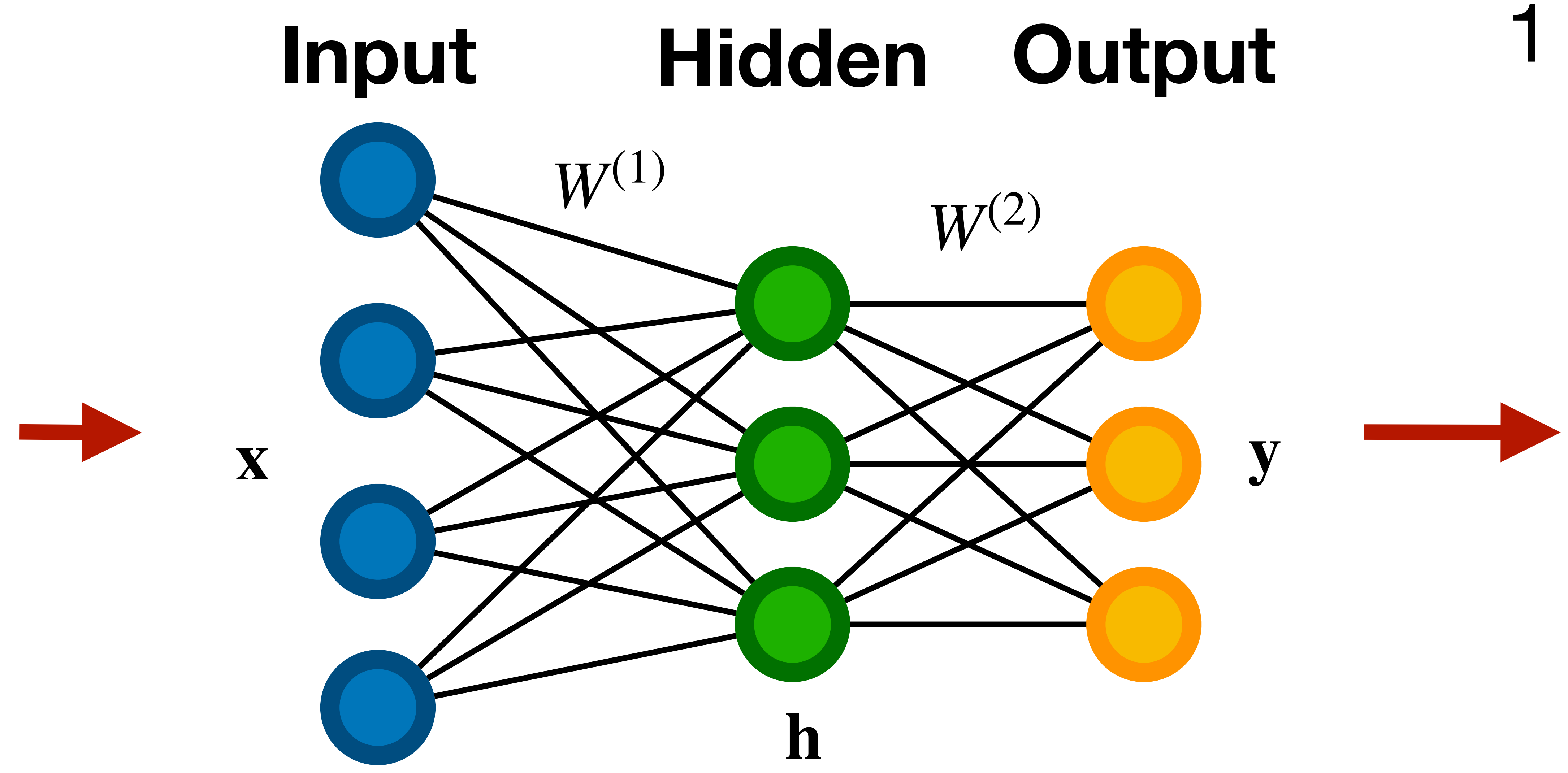
$$\mathbf{y} = f\left(W^{(2)}\mathbf{h} + b^{(2)}\right)$$

Image classification with neural networks



Flatten to a 1D array/vector.

N x N image to a $N^2 \times 1$ vector.



NN predicts class:
1 output neuron per class
(one hot encoding)

$$c = \arg\max_i (y_i)$$

Some models convert $\mathbf{y}$ into a probability, e.g softmax

Cross entropy loss is suitable for multi class predictions.

General Recipe for Gradient Learning

1. Given training data

$$\{\mathbf{x}_n, \mathbf{y}_n^{\text{true}}\}_{n=1}^{N}$$

2. Choose each of these:

- Model / decision function

$$\mathbf{y}_n = f_{\mathbf{w}}\left(\mathbf{x}_n\right)$$

- Loss function or metric

$$l(\mathbf{y}_n, \mathbf{y}_n^{\text{true}})$$

3. Define a goal:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} l(\mathbf{y}_n, \mathbf{y}_n^{\text{true}})$$

4. Optimise with gradient descent

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla l(\mathbf{y}_n, \mathbf{y}_n^{\text{true}})$$

Compute gradients using back propagation (using auto-diff)

Using torch.nn to create models

If we want to build our own models in PyTorch we can create classes inheriting from the **torch.nn.Module** class.
Internally this class can then hold various layers and operations.

```
class neural_network(nn.Module):
    def __init__(self, in_features, hidden_features, out_features, bias=True):
        super().__init__()
        self.lin1 = nn.Linear( in_features, hidden_features, bias)
        self.act_func1 = nn.ReLU()
        self.lin2 = nn.Linear( hidden_features, out_features, bias)
        self.act_func2 = nn.Sigmoid()

    def forward(self, x):
        h = self.act_func1(self.lin1(x))
        return self.act_func2(self.lin2(h))
```

Simplify using nn.Sequential

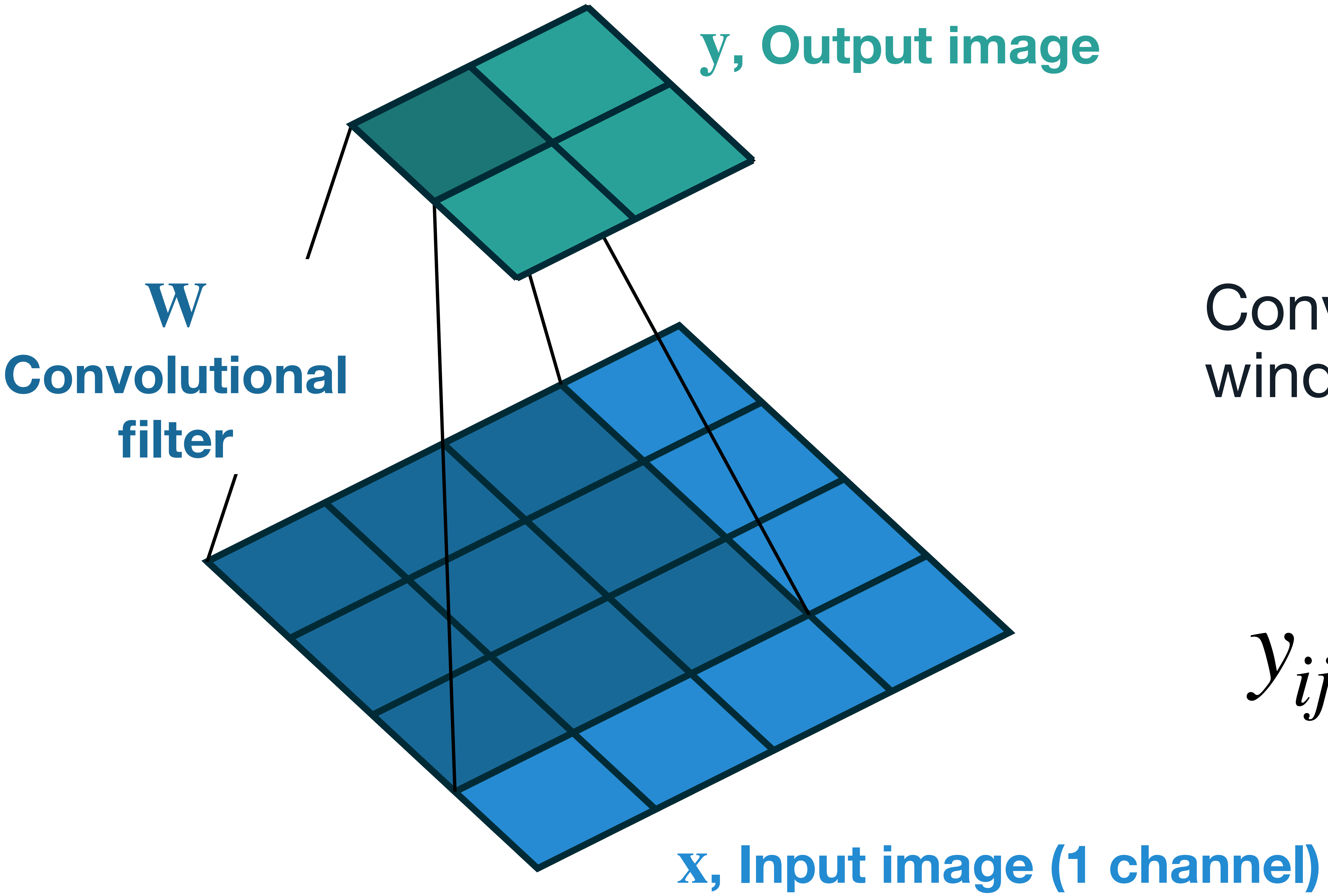
If we are chaining together layers, we can use the built in Sequential class:

```
model = nn.Sequential(
    nn.Linear(in_features, hidden_features),
    nn.ReLU(),
    nn.Linear(hidden_features, out_features),
    nn.Sigmoid()
)
```

In each case we can use the model to predict using:

```
y_approx = model(x)
```

Convolutional filters



Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolutional operation (b is bias):

$$\mathbf{y} = b + \mathbf{W} \star \mathbf{x}$$

Convolution filter is applied as a moving window over the 2D input image.

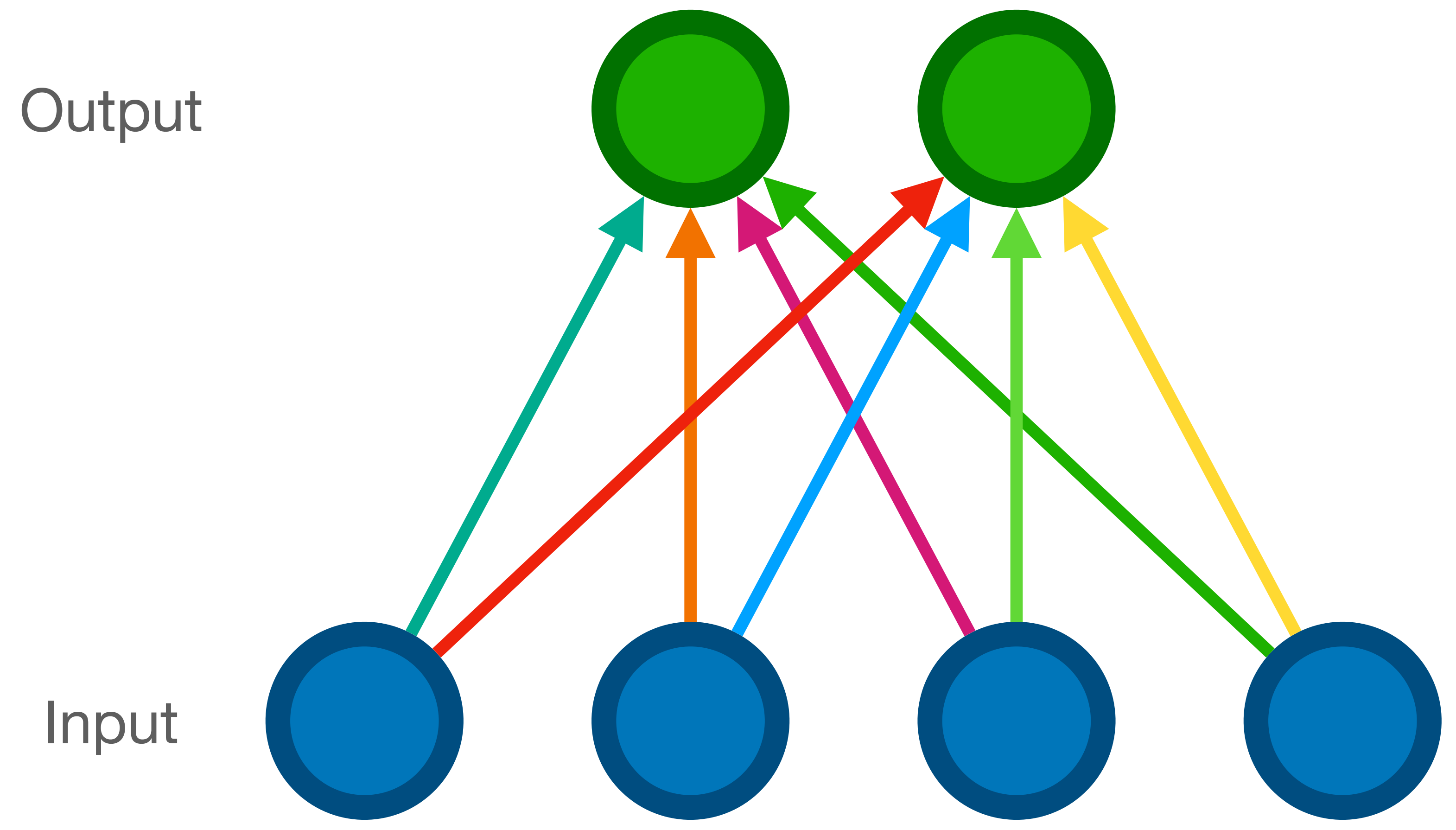
$$y_{ij} = b + \sum_{k=0}^{F-1} \sum_{l=0}^{F-1} W_{kl}\, x_{i+k,j+l}$$

Visualising convolutions in 1d

Fully Connected

Each output is connected to all inputs.

Total number of weights = number of inputs x number of outputs

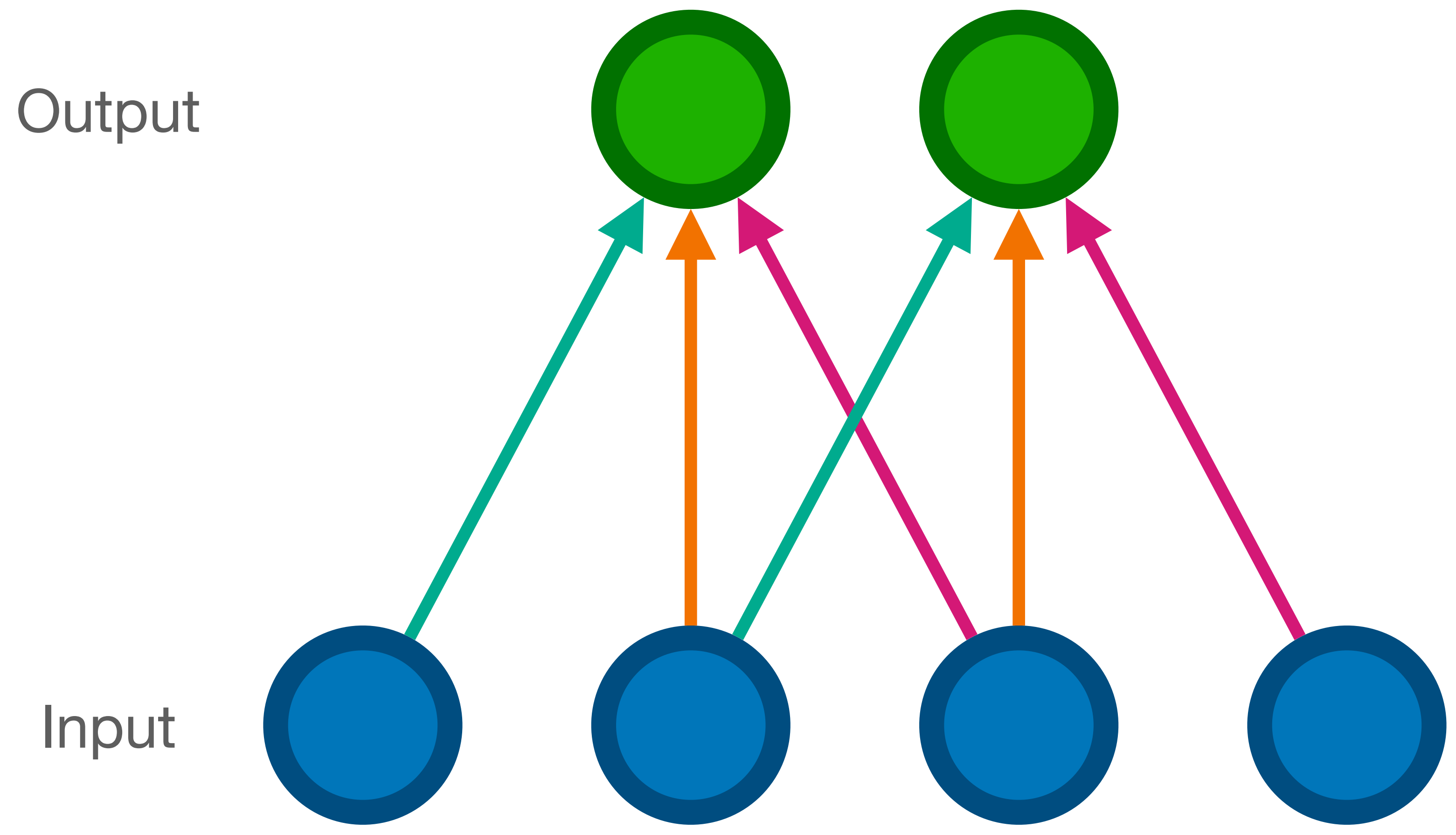


Each colour means a different weight parameter.
The same colours mean the same weight.

Convolution

Each output connects to a particular region of the input. Weights are shared.

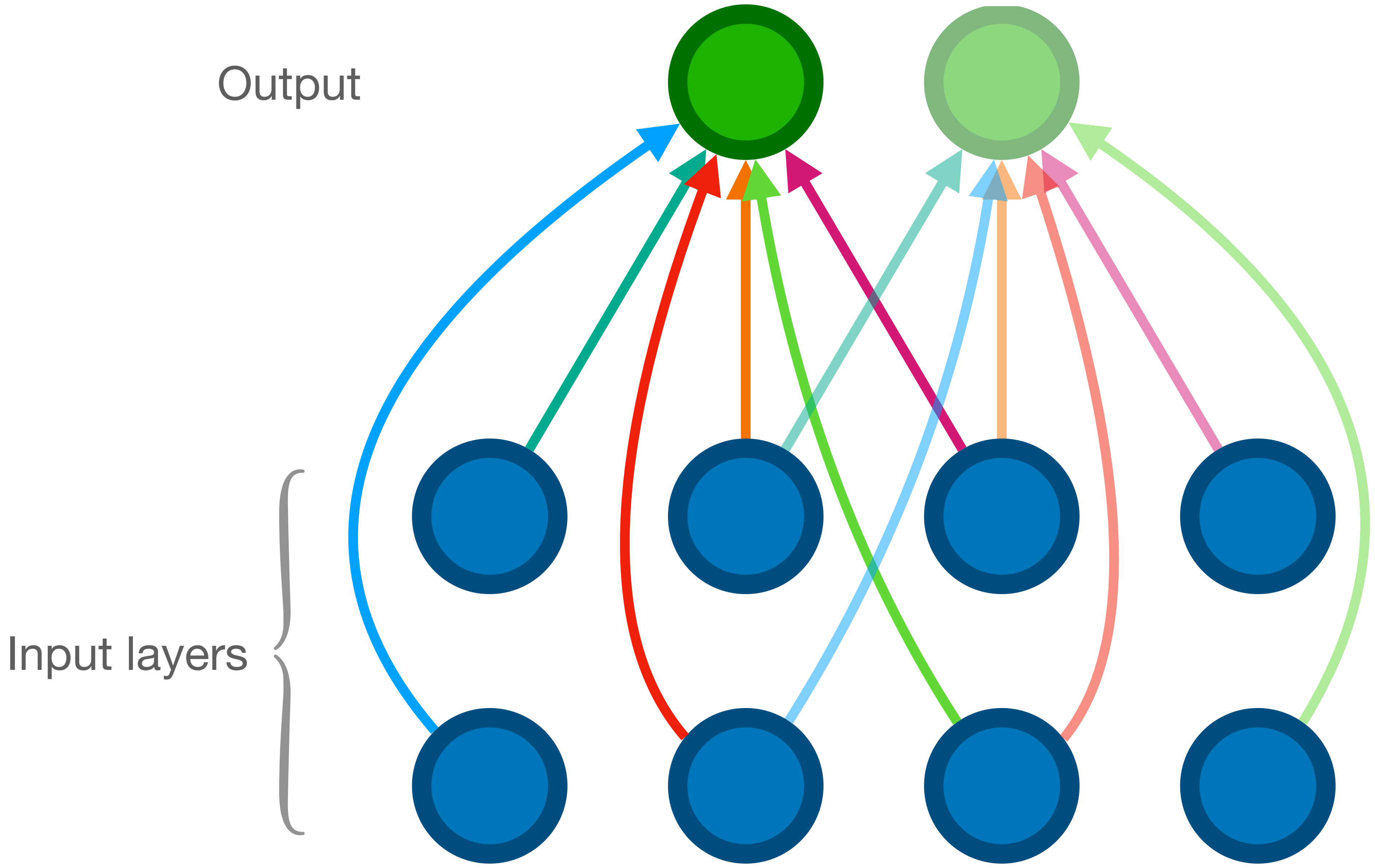
Total number of weights = Kernel size



Visualising convolutions in 1d

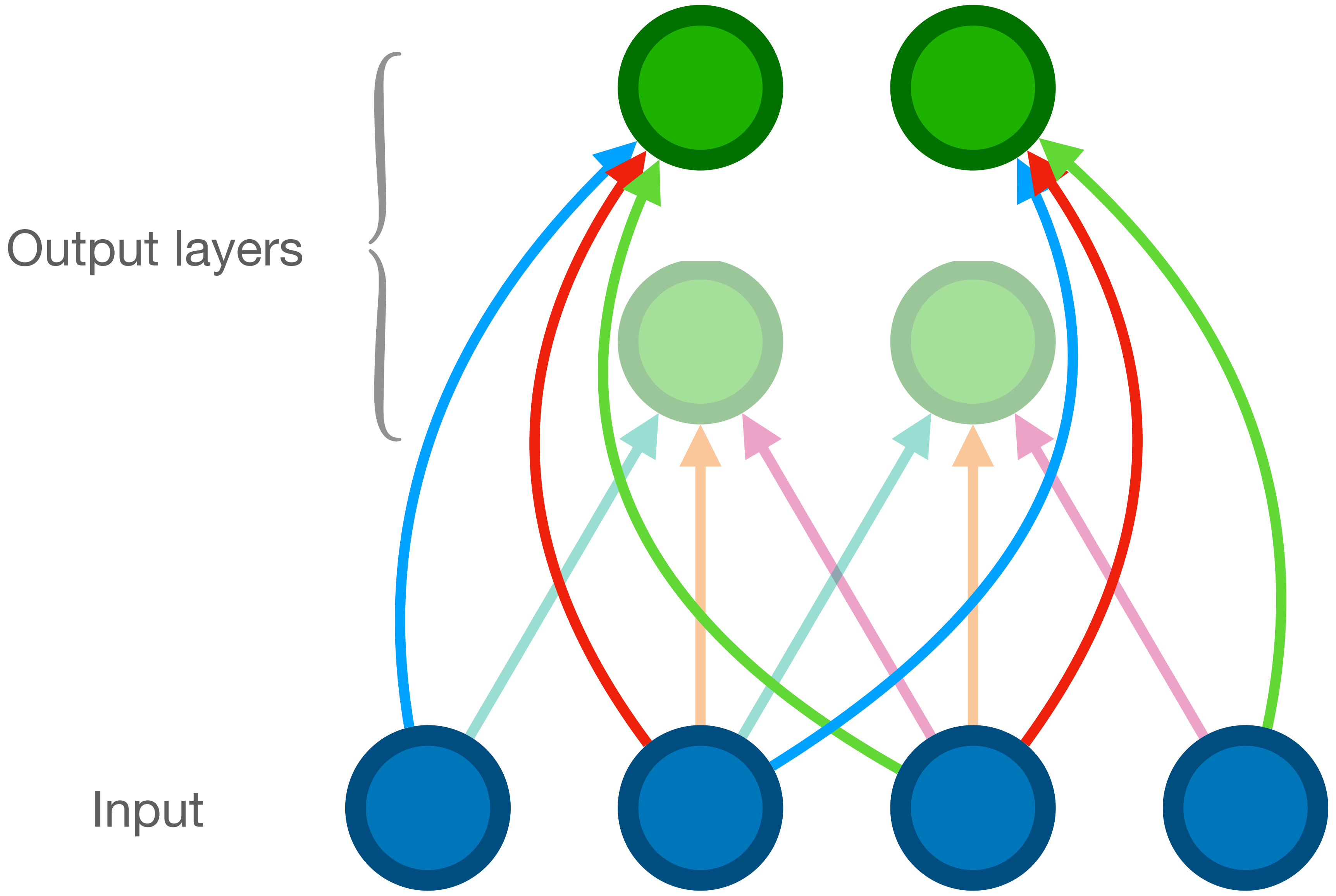
Input Channels

The convolutional kernel will have additional weights and sum over additional input channels.



Output Channels

Each kernel will learn one feature and create one output feature map. Additional kernels are used to learn more features and expand the output.



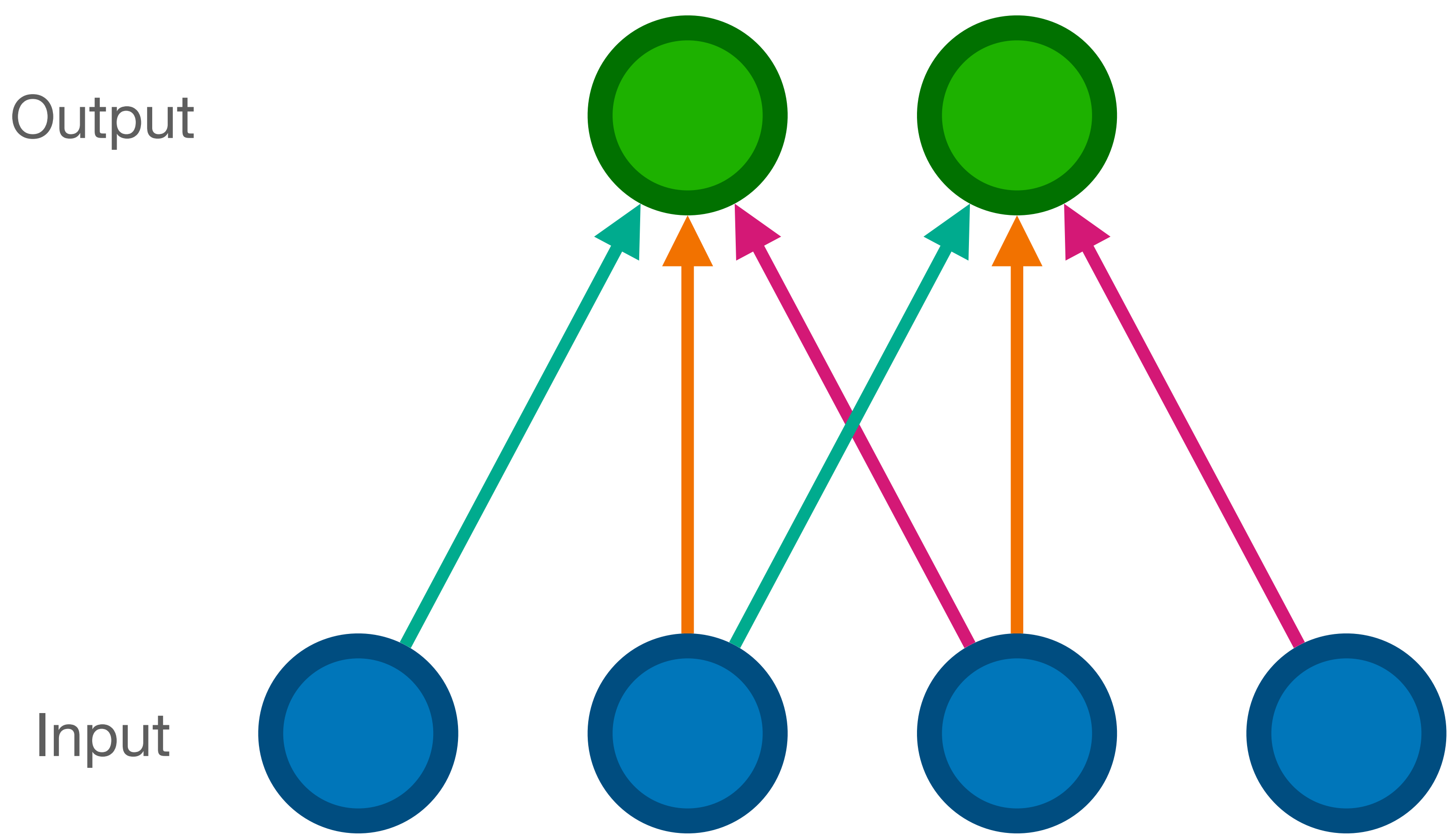
Visualising convolutions in 1d

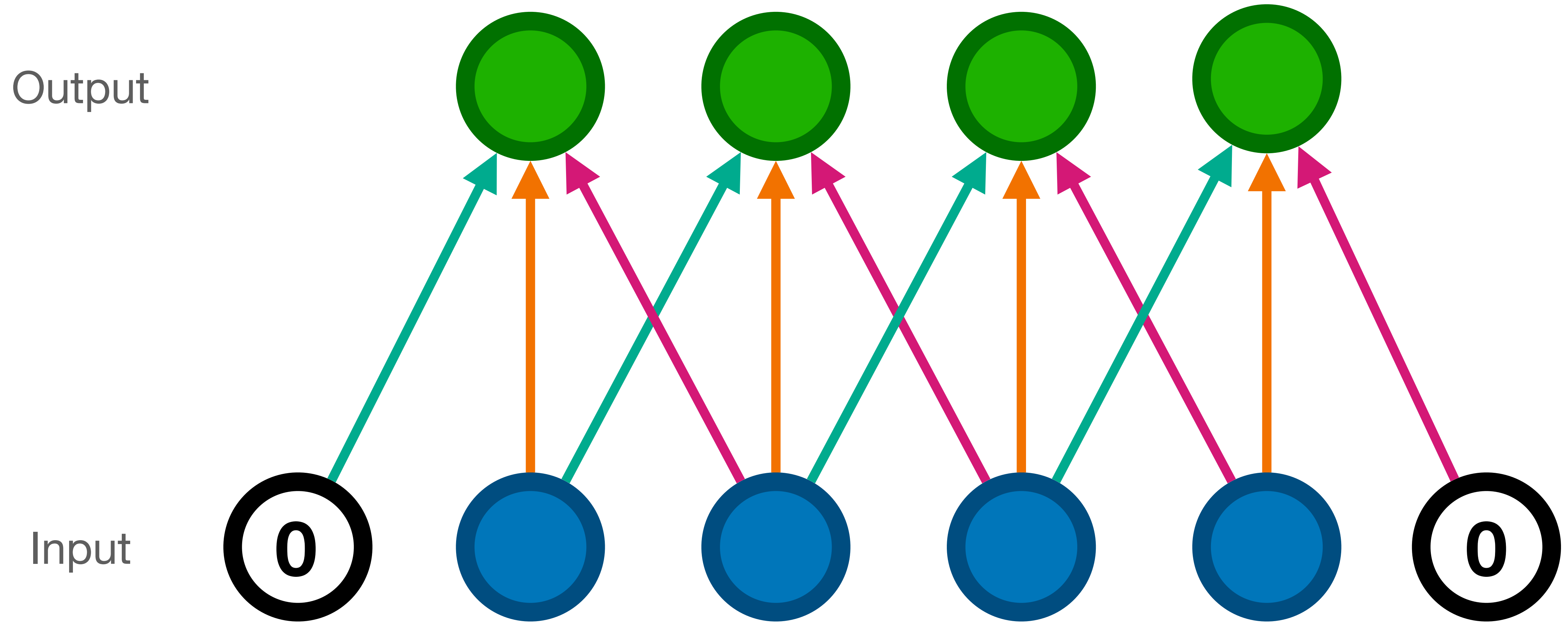
Zero padding

Convolutions reduce the input size by F - 1.

Padding adds zeros each size to manipulate the output size.

Common to use P = (F - 1)/2 to maintain input size.



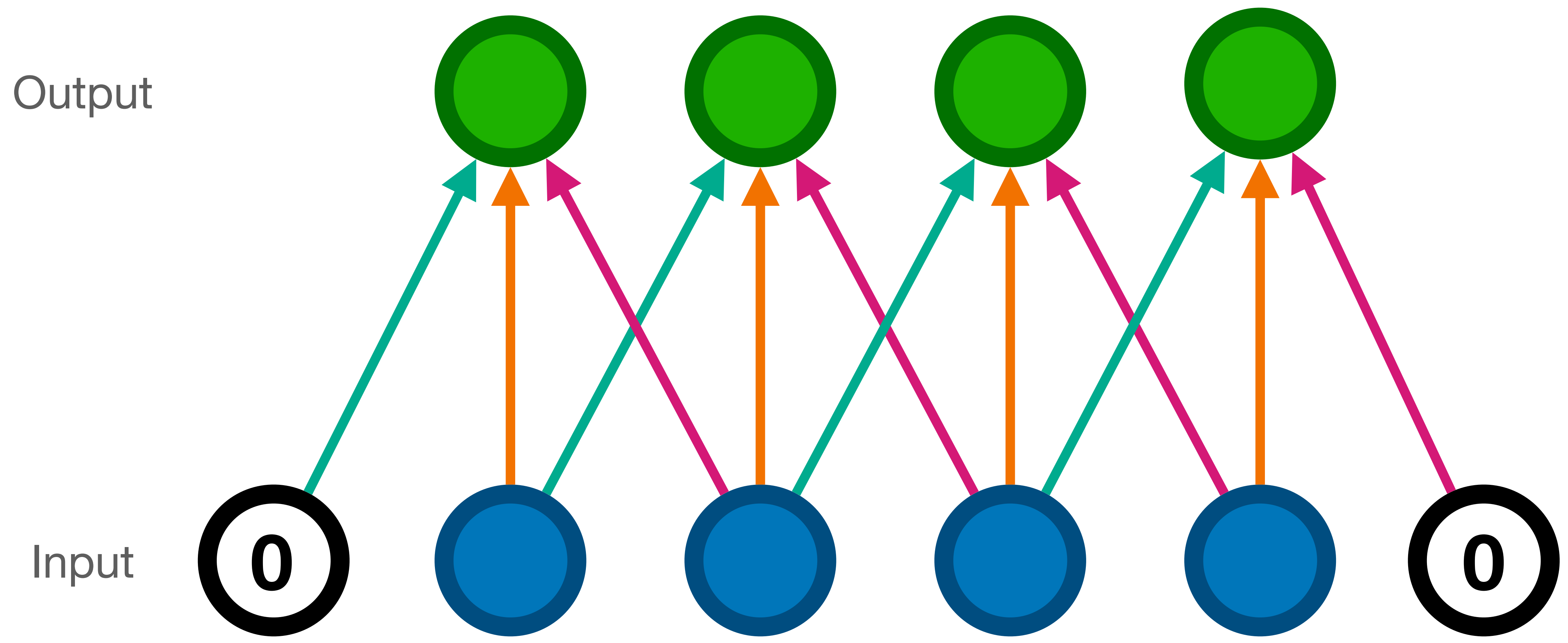


Padding of one each side.

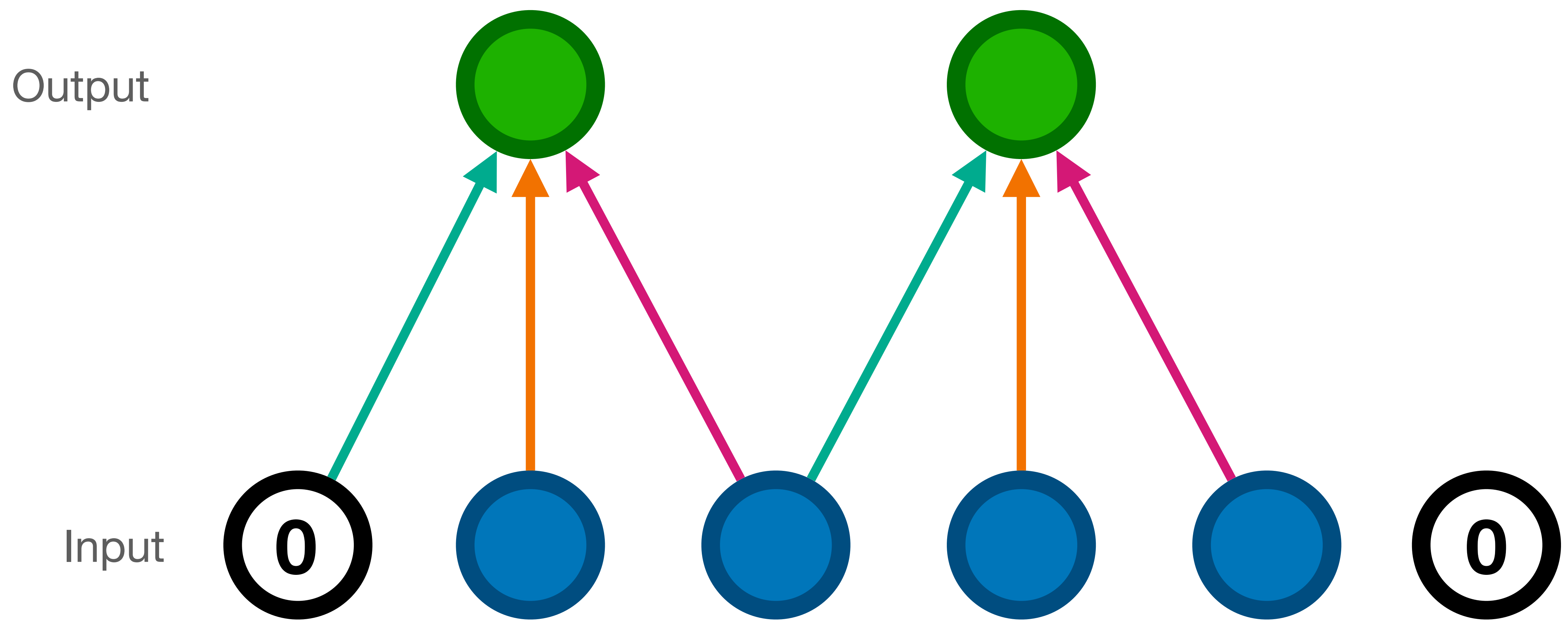
Visualising convolutions in 1d

Stride - determines where the next kernel starts relative to the last.

Effectively reduces the output size by the stride size. I.e a stride of 2 will 1/2 the output.

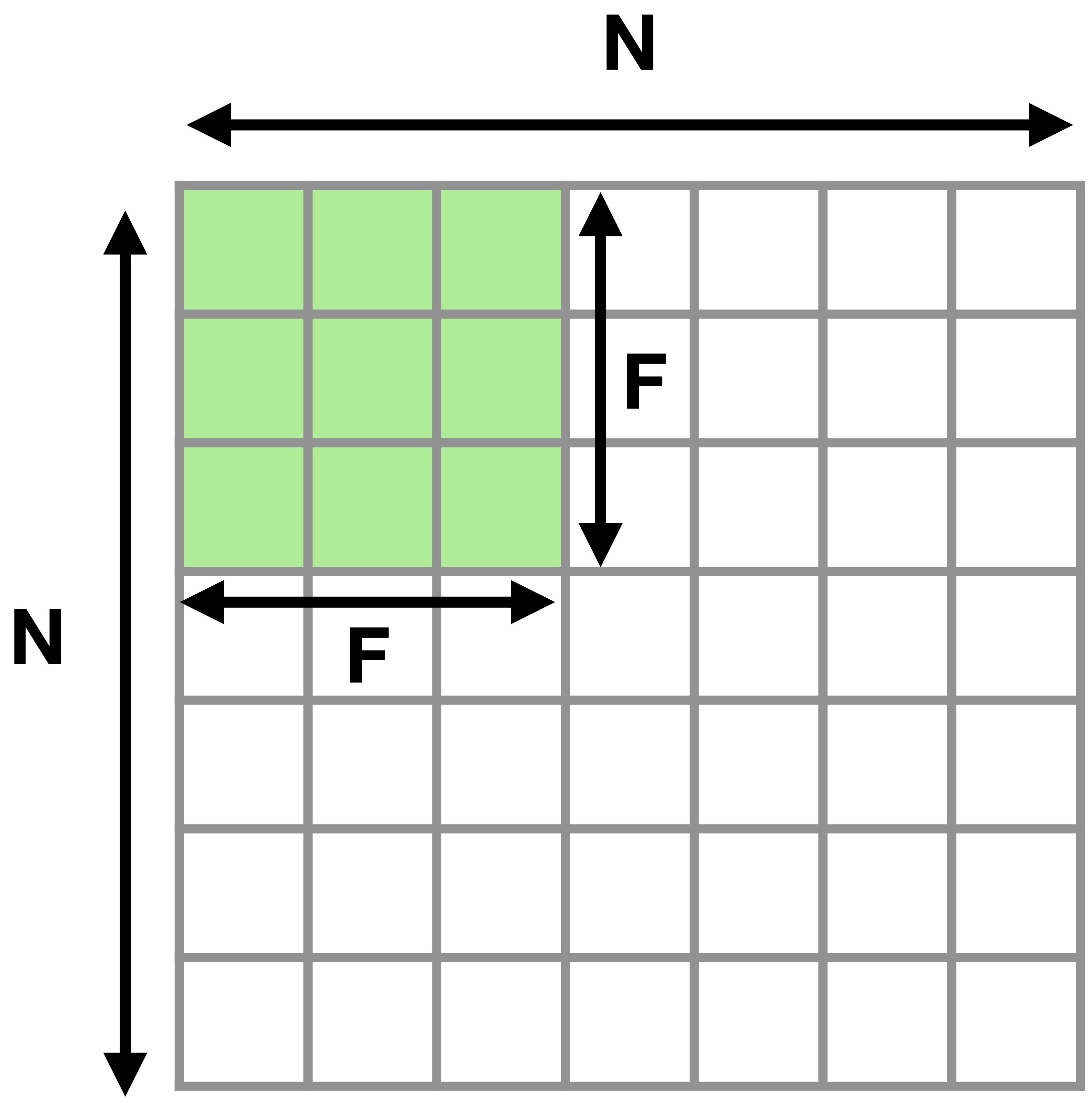


Stride = 1, Padding = 1



Stride = 2, Padding = 1

Output size after strided convolutions



$$\text{Output size} = \frac{N - F + 2P}{S} + 1$$

Example

N = 7, F = 3, P = 0

What is the output size for stride 1, 2 and 3?

Stride 1: (7 - 3)/1 + 1 = 4

Stride 2: (7 - 3)/2 + 1 = 3

Stride 3: (7 - 3)/3 + 1 = 2.333 (round down)

Reading

Neural Networks

Deep Learning by Goodfellow, Bengio and Courville

Chapter 6: sections 6.3 and 6.4 (pages 187 to 200)

Chapter 8: sections 9.1 to 9.3 (pages 326 to 339)

Available at https://www.deeplearningbook.org/