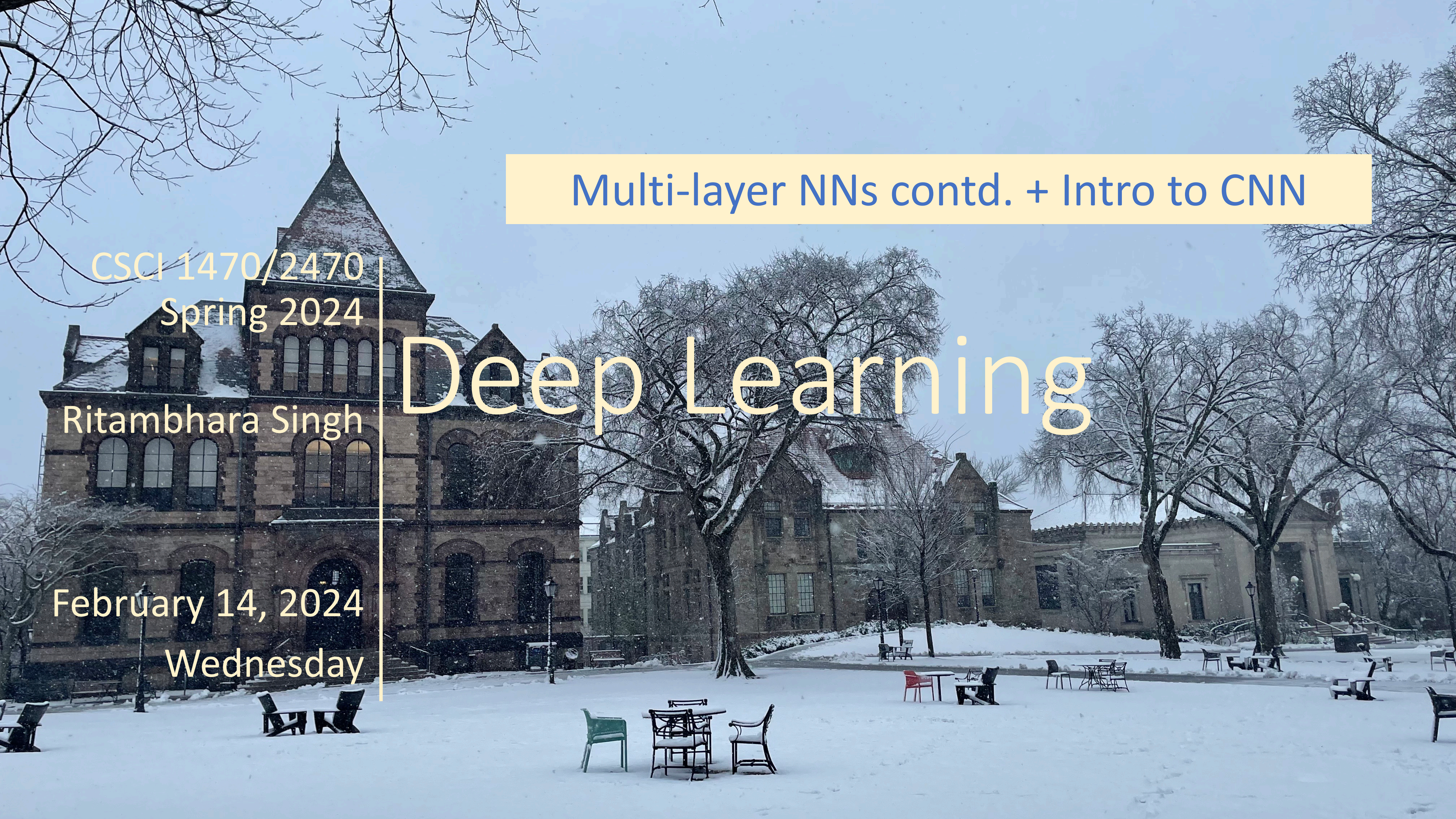


Multi-layer NNs contd. + Intro to CNN

CSCI 1470/2470
Spring 2024

Deep Learning

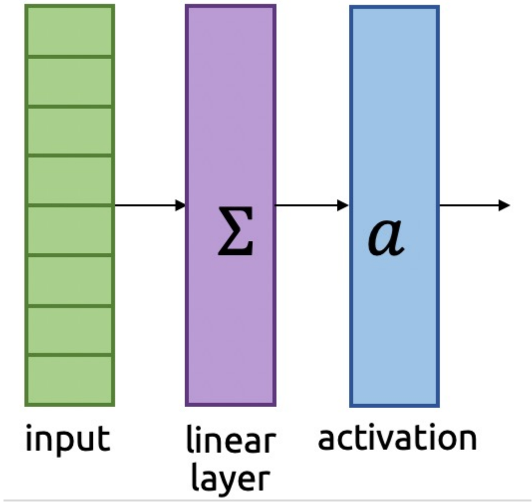
Ritambhara Singh

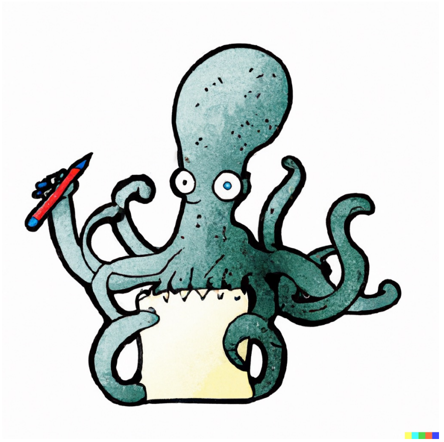
February 14, 2024
Wednesday

Recap

Stacking multiple layers

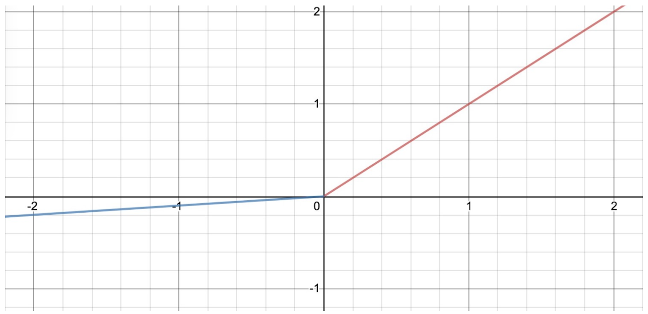
- More layers → more complicated function
- Linear layers are not sufficient!
- Need non-linearity



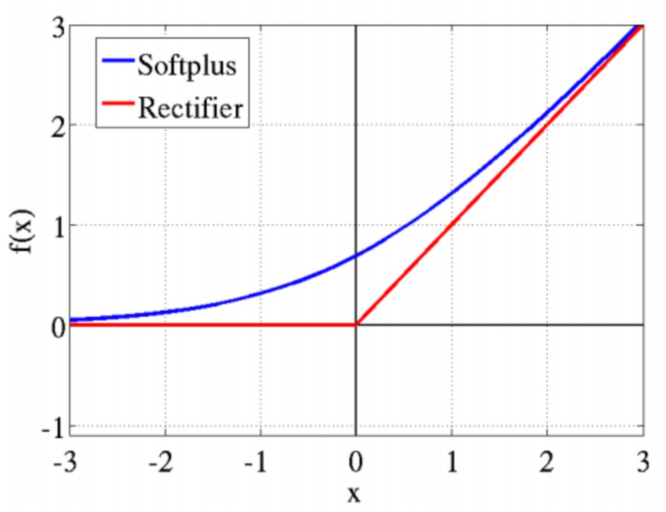


Activation functions

- Exploding gradients
- Vanishing gradients
- ReLU, Leaky ReLU



Recap: Reasons to use other activation functions



- Bounding network outputs to a particular range
  - Tanh: [-1, 1]
  - Sigmoid: [0,1]
  - Softplus: [0, ∞]

- Example: Predicting a person's age from other biological features
  - Age is a strictly positive quantity
  - We can help our network learn by restricting it to output only positive numbers
  - Use a Softplus activation on the output

Today’s goal – continue to learn about multi-layer networks and learn about convolution

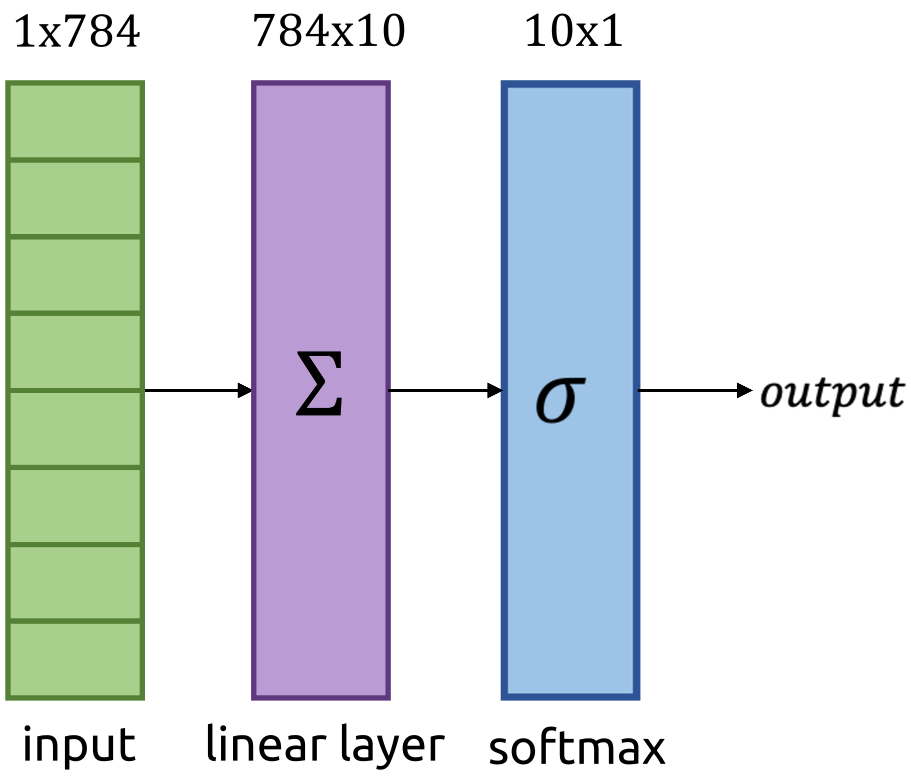
(1) What are hidden layers and hyperparameters?

(2) Universal approximate theorem – what a one-hidden layer network can learn?

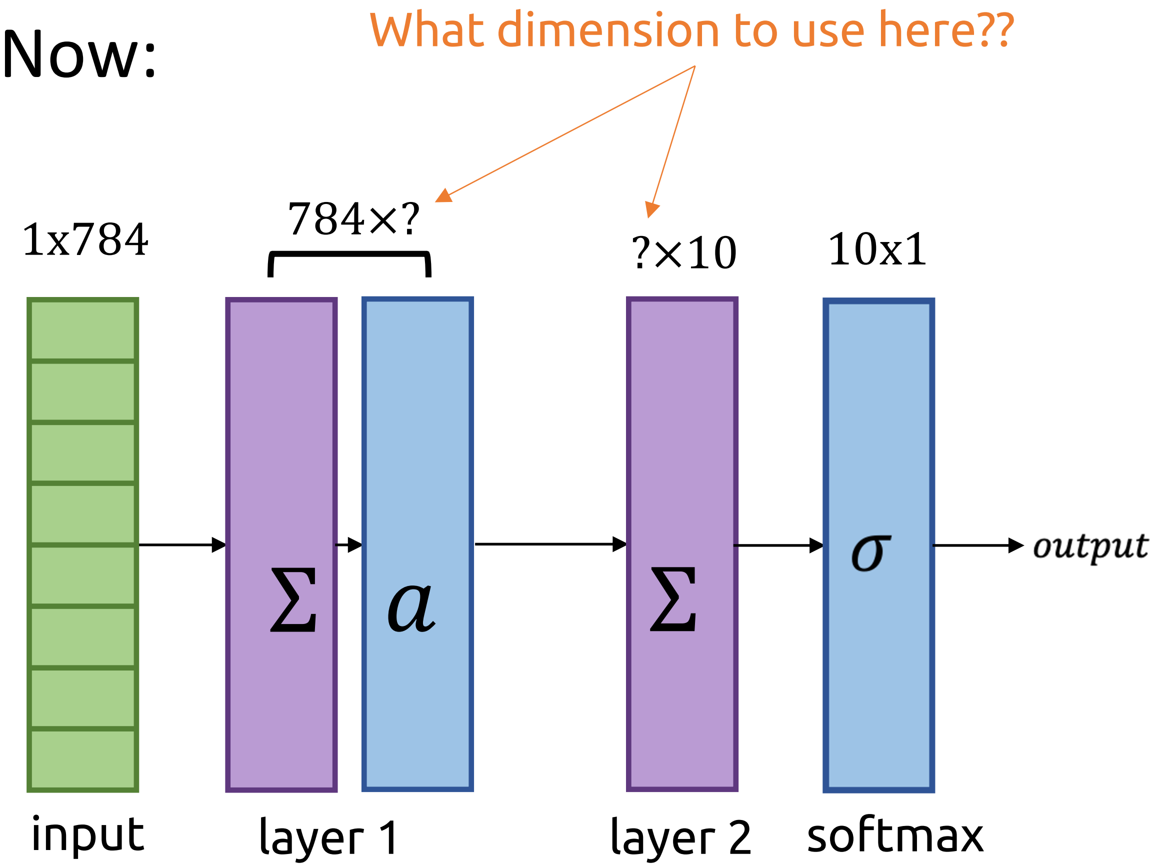
(3) Intro to CNNs – Convolution

Recap: Consequences of adding activation layers

- Previously:

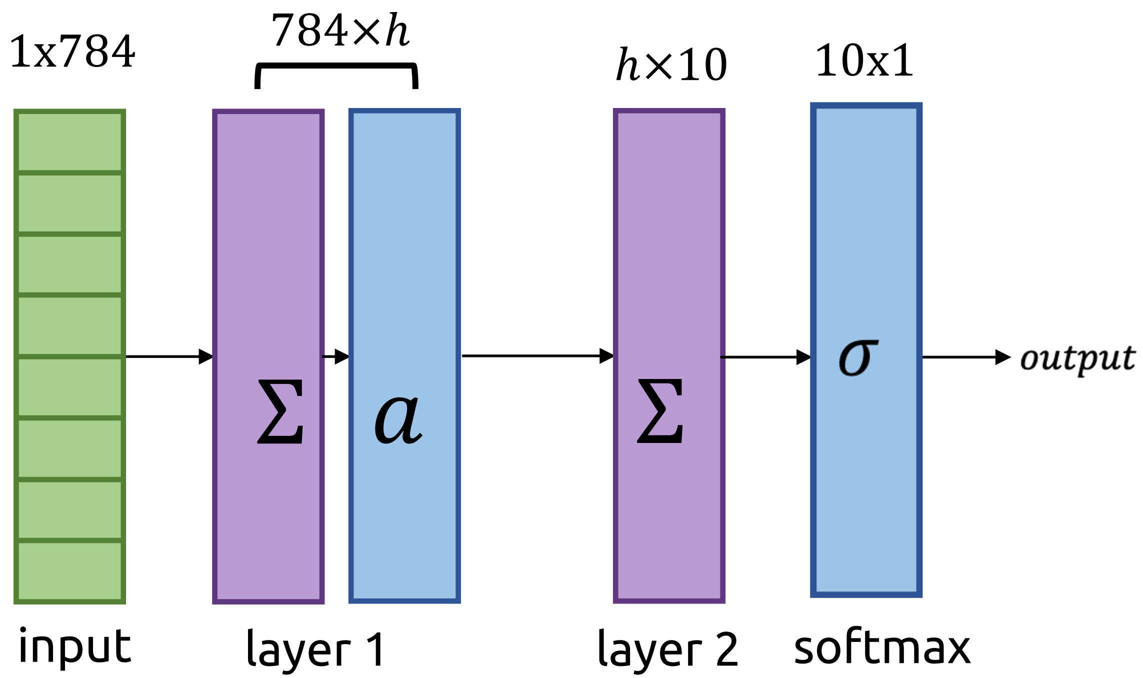


- Now:



“Hidden Layers”

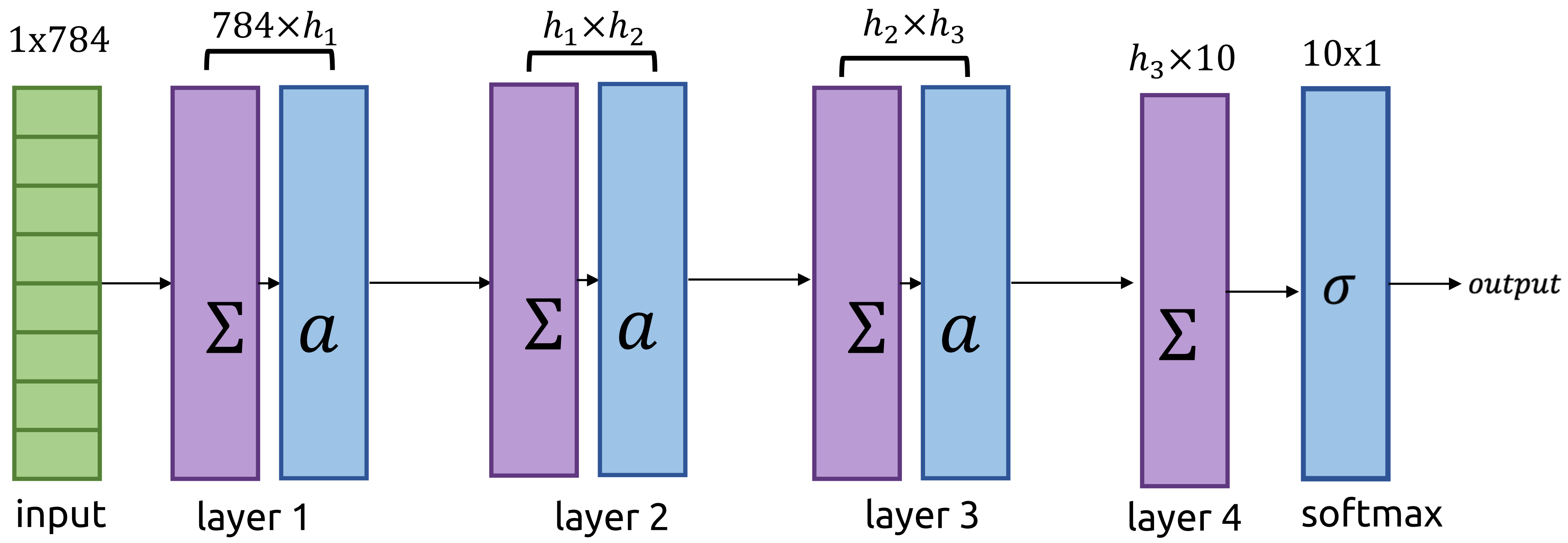
- The output of a function that doesn’t feed into the output layer (like softmax) is called a ***hidden layer***
- Have to set the size h of these hidden layers
- More linear units → more hidden layer sizes



Hyperparameters

- Hidden layer sizes are a ***hyperparameter*** — configuration external to model, value usually set before training begins
  - Number of epochs, batch size, etc.
  - Contrast this with a learnable ***parameter***, we keep talking about
- Rule of thumb
  - Start out making hidden layers the same size as the input
  - Then, tweak it to see the effect
- There are more principled (and time-consuming) ways to set them
  - Grid search, random search, Bayesian optimization...
  - See here for an overview and more references

What a multi-layer neural network could look like?



What functions can a one-hidden-layer neural net learn?

Universal Approximation Theorem [Cybenko '89]

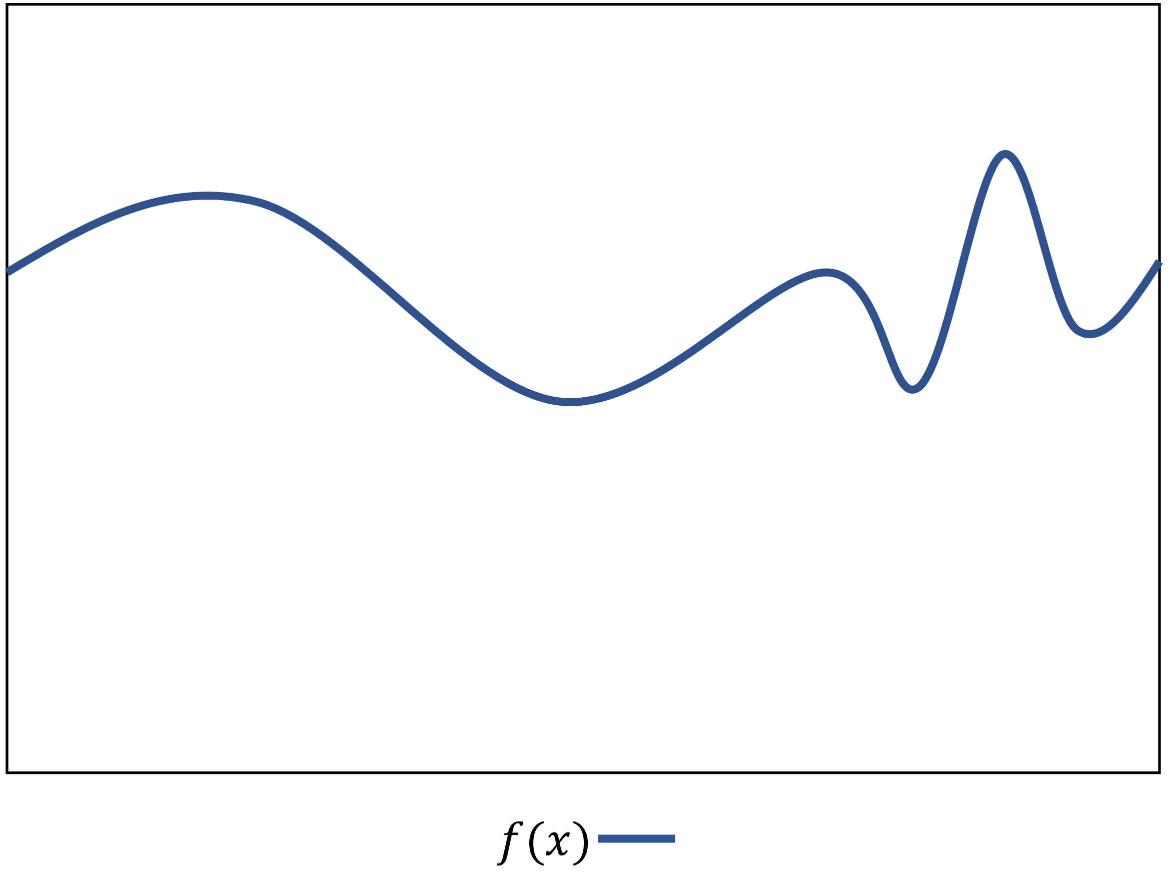
- Remarkably, a one-hidden-layer network can actually represent ***any*** function (under the following assumptions):
  - Function is continuous
  - We are modeling the function over a closed, bounded subset of $\mathbb{R}^n$
  - Activation function is sigmoidal (i.e. bounded and monotonic)

- The proof of this theorem is an existence proof
  - i.e. it tells us that a network exists which can approximate any function, not how to actually learn it

A “Proof By Picture”

Universal Approximation Theorem “Proof”

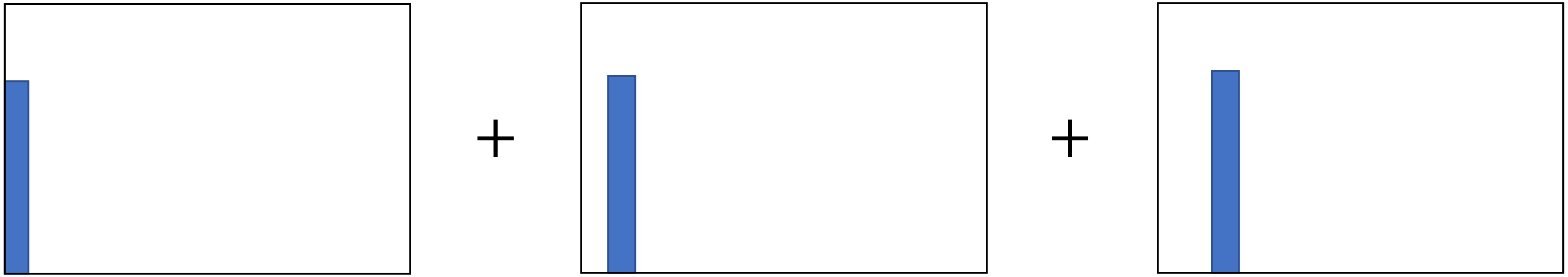
- Start with a complex one dimensional function that relates some input x to some output y
- We don’t know what the function that relates x and y is



https://hackernoon.com/illustrative-proof-of-universal-approximation-theorem-5845c02822f6

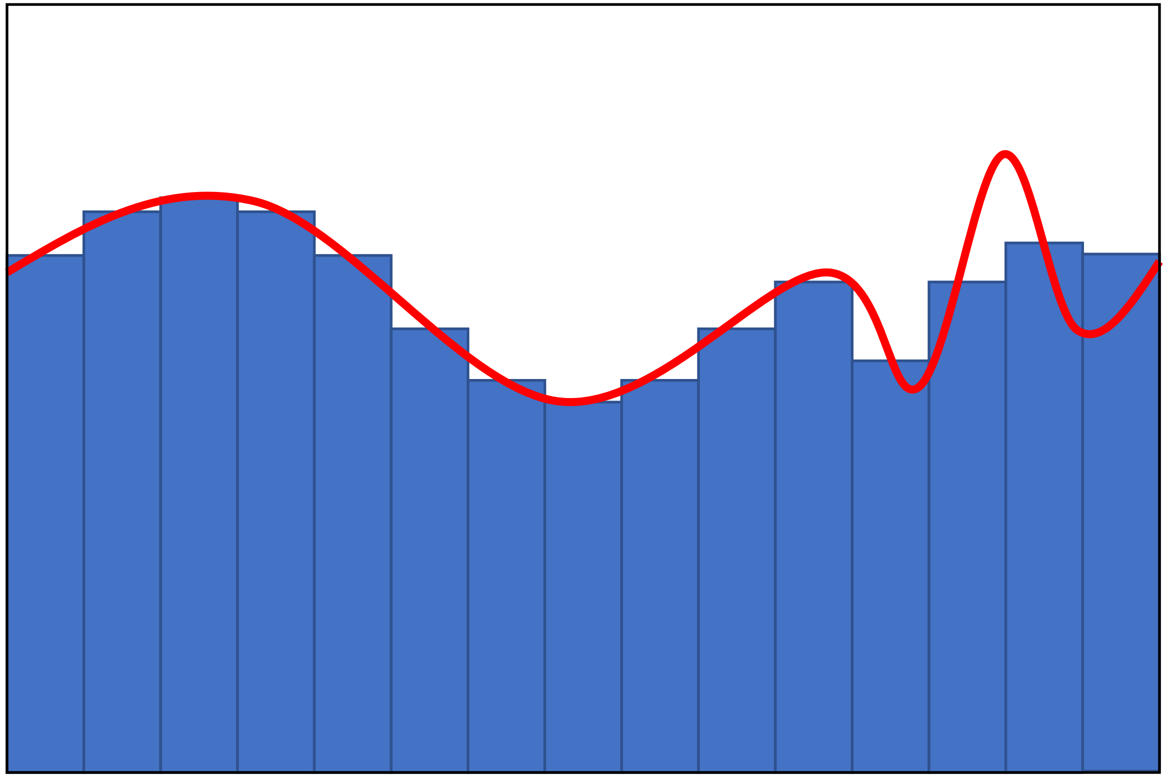
Universal Approximation Theorem "Proof"

- We can build up this function using simpler functions, i.e. box functions



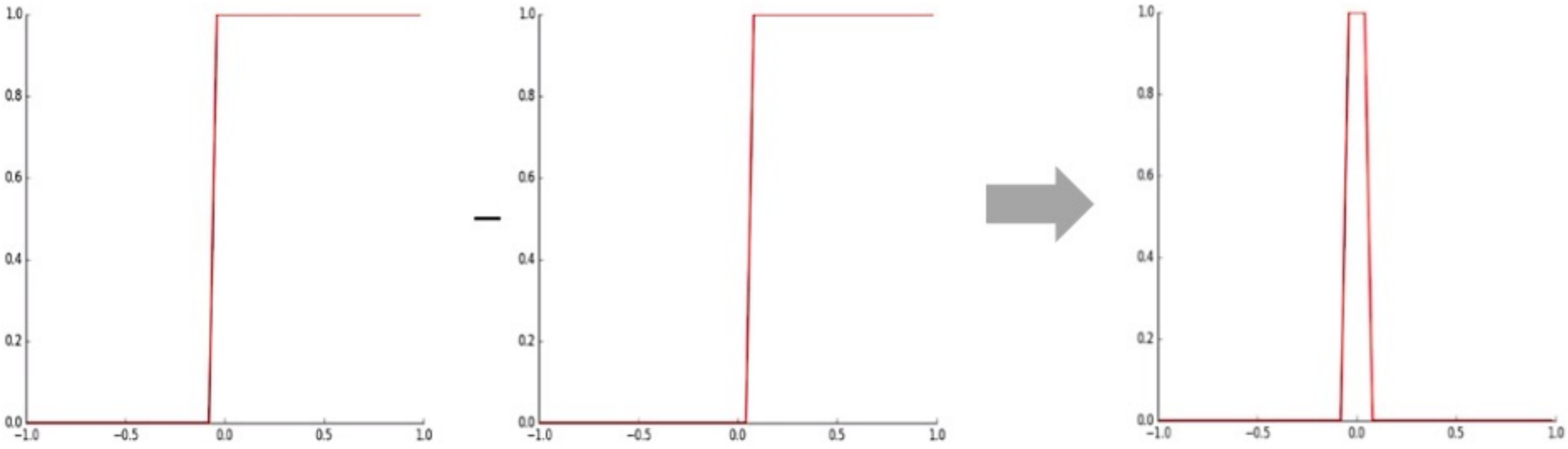
Universal Approximation Theorem "Proof"

$f(x)$



How does this relate to activation functions?

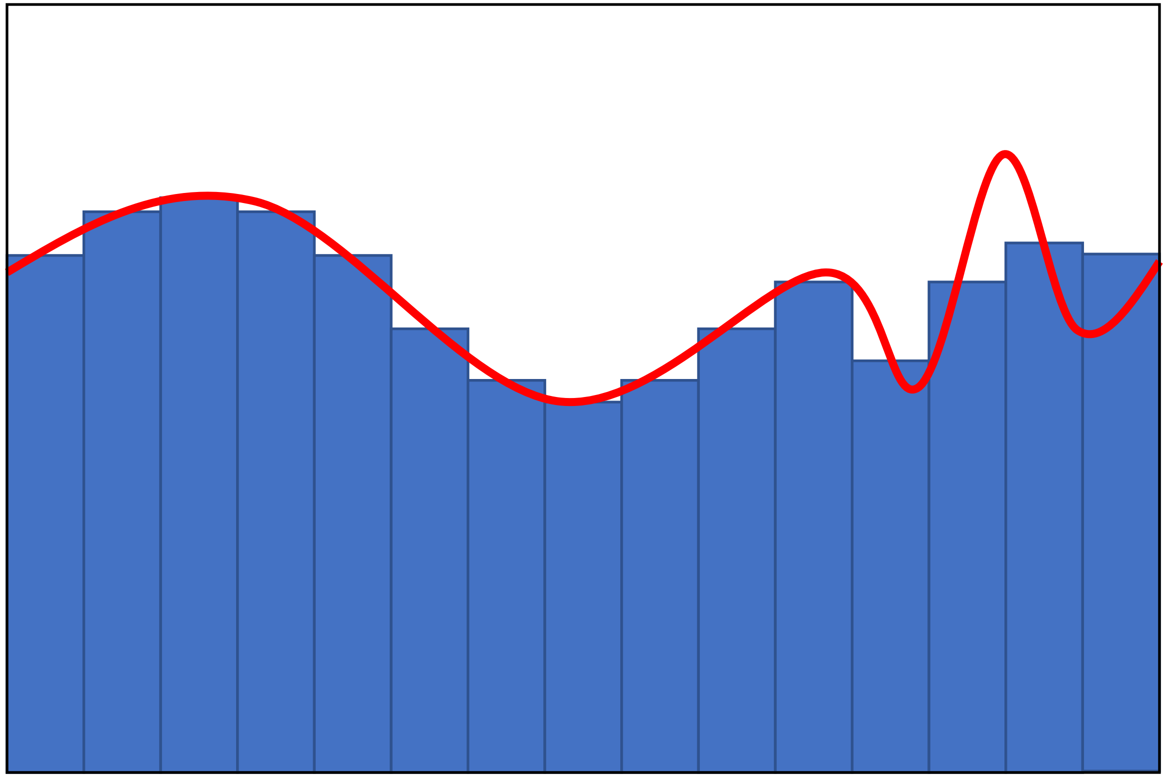
- We can subtract two sigmoids to create these box functions



Universal Approximation Theorem "Proof"

- Summing up these simpler functions can do a pretty good job of approximating the actual function

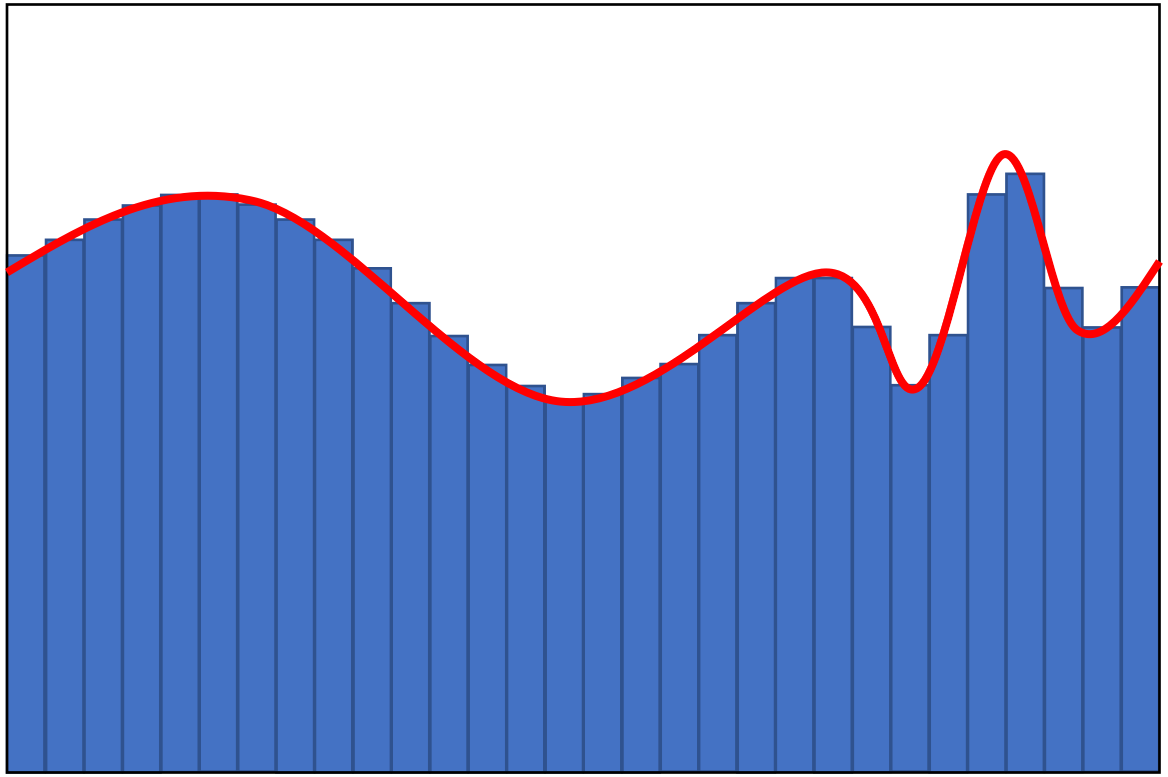
$f(x)$



Universal Approximation Theorem “Proof”

- Using more functions lets us model a complex function more accurately
  - Up to an arbitrary degree of accuracy, if we want

$f(x)$



Any questions?

Universal Approximation Theorem "Proof"

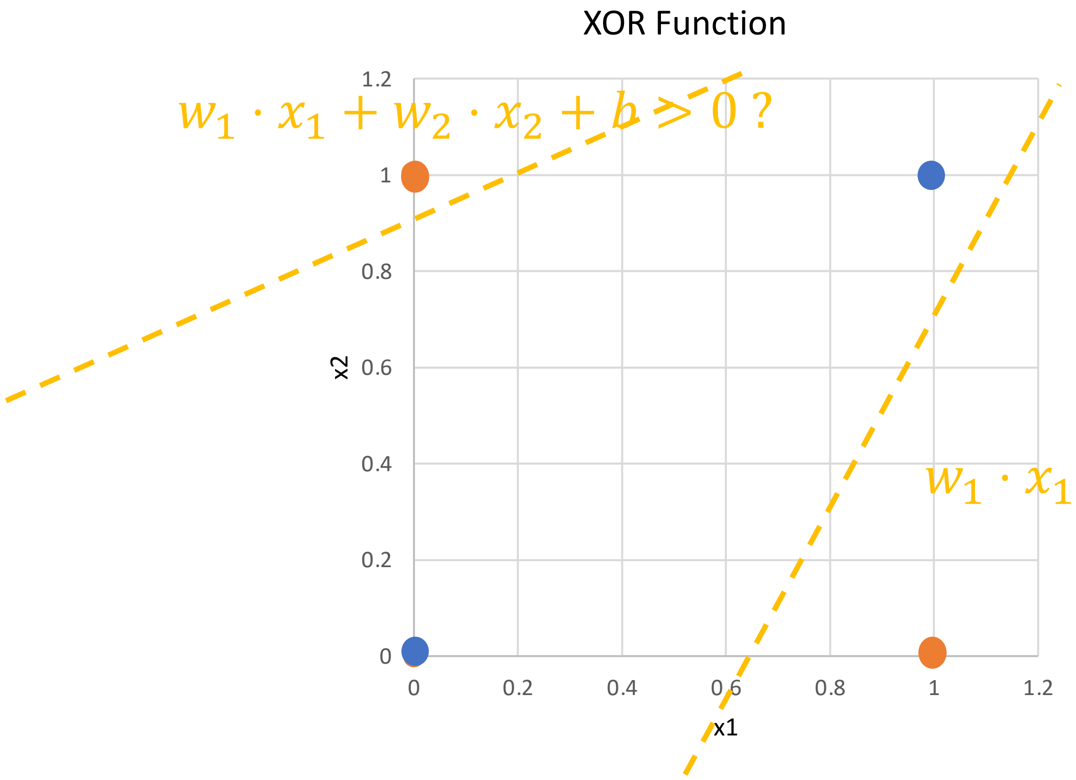
- ***Very*** inefficient way to approximate
  - Need ***lots*** of box functions → ***lots*** of sigmoids → very large hidden layer

- Real networks trained with gradient descent can't even learn these kinds of approximations
  - They **find smooth approximations**, require more hidden layers to get this same level of complexity.

- Nevertheless, the theorem is often cited to back up claims that a sufficiently complex neural net "can learn any function"

Do you remember what function a perceptron could not learn?

Can a multi-layer network learn XOR?



Output = 0

Output = 1

Let’s find out

[Google Tensorflow Playground]

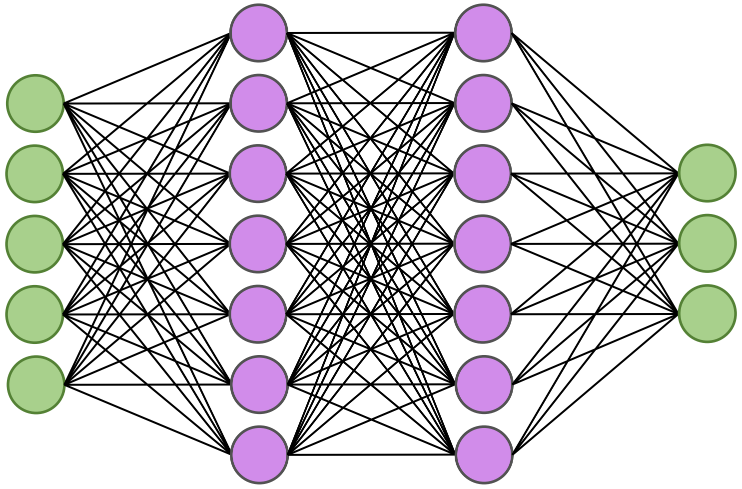
What kind of datasets CNNs are popularly applied to?

Convolution and CNNs

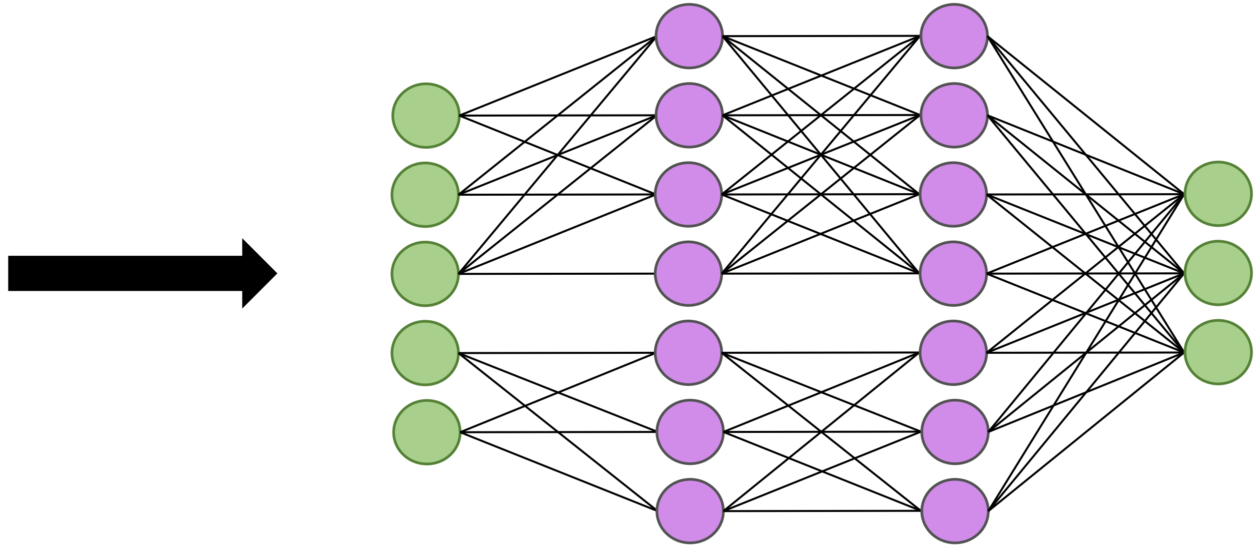
Images!

Does a network have to be fully connected?

Fully Connected



Partially Connected?



Why would you ever want to do this?

Partially Connected Networks?

- Fewer connections == Worse results? ...right?

- Advantages of Partial Connections
  - Fewer connections → fewer weights to learn
    - Faster training; more compact models; better generalization performance
  - Can design connectivity pattern that exploits knowledge of the data (like connecting patterns in features)

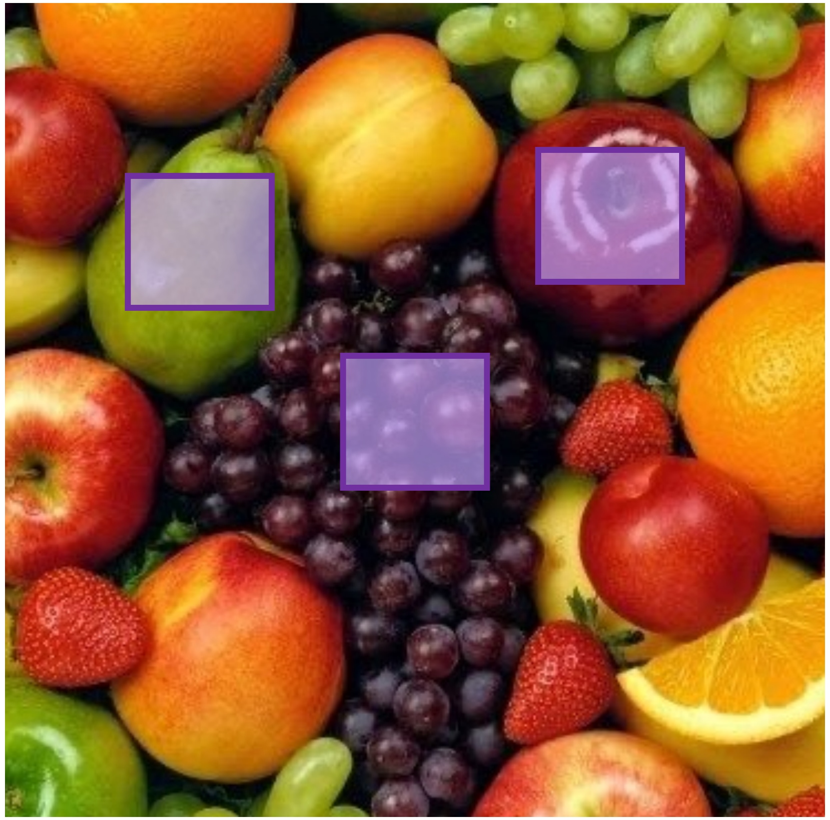
What's a data type where we can do this?

Images!

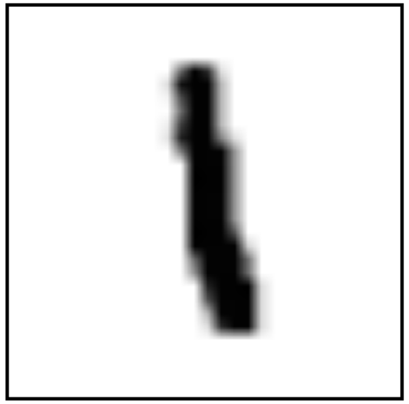
When partially connected networks are useful

- **Observation:** Nearby pixels are more likely to be related
- **Assumption:** It is okay to only connect the nearby pixels



Limitations of Full Connections for MNIST

Suppose we've got a well-trained MNIST classifier…



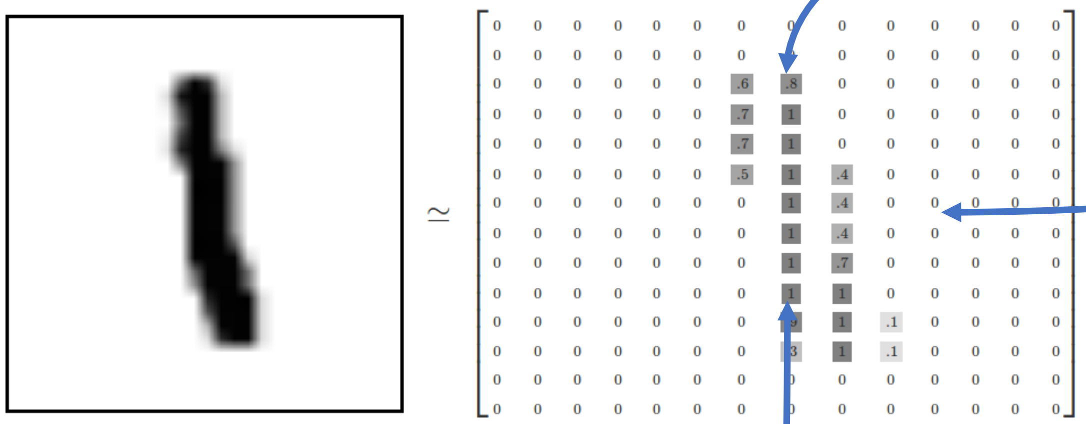
$$
\cong
\begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}
$$

#1 encoded as →

Limitations of Full Connections for MNIST

Suppose we've got a well-trained MNIST classifier...

this pixel gets weight 0.6



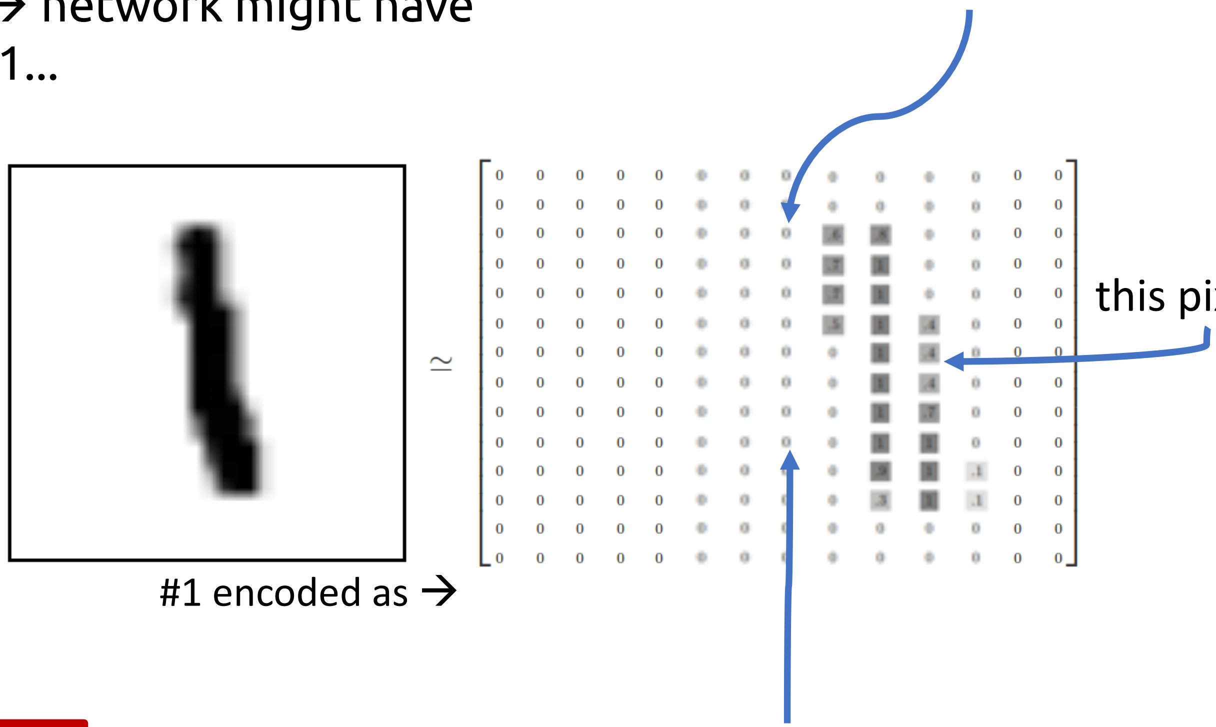
this pixel gets weight 0.1

#1 encoded as →

this pixel gets weight 0.9

Limitations of Full Connections for MNIST

If we shift the digit to the right, then a different set of weights becomes relevant → network might have trouble classifying this as a 1…

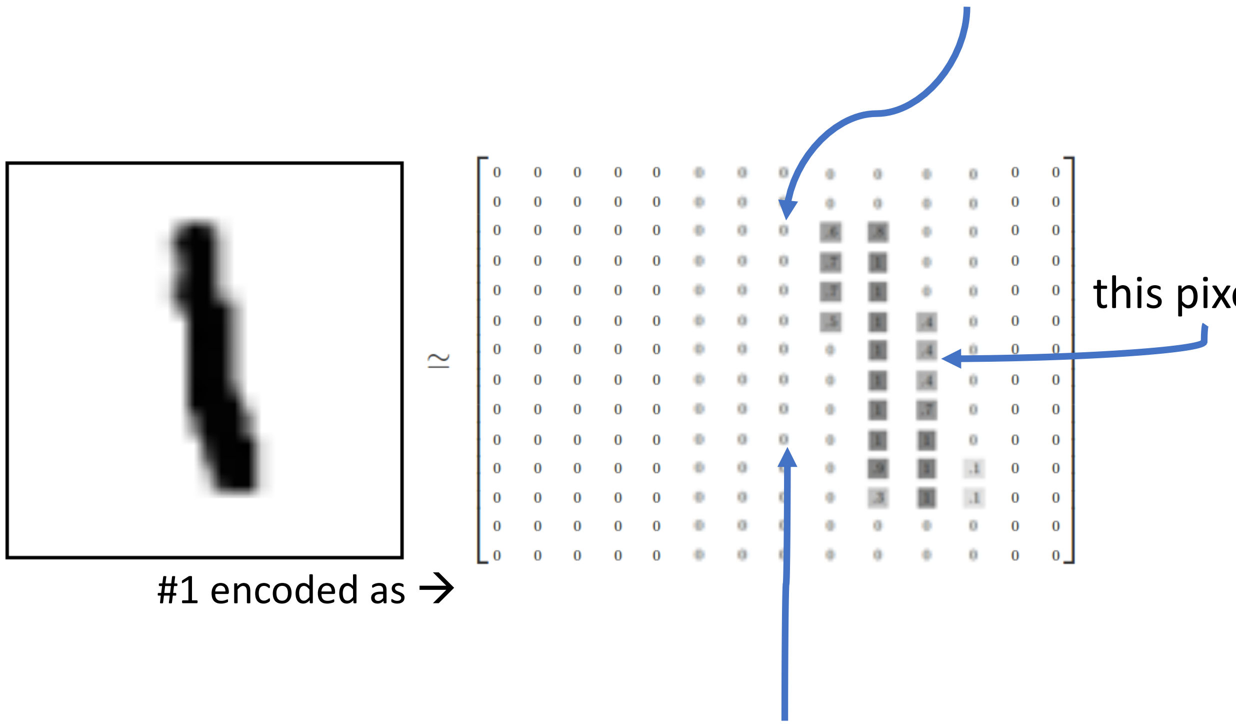


Can you tell this is a 1?

This would *not* be a problem for the human visual system

Our eyes don't look at absolute intensity values…

this pixel has a low intensity



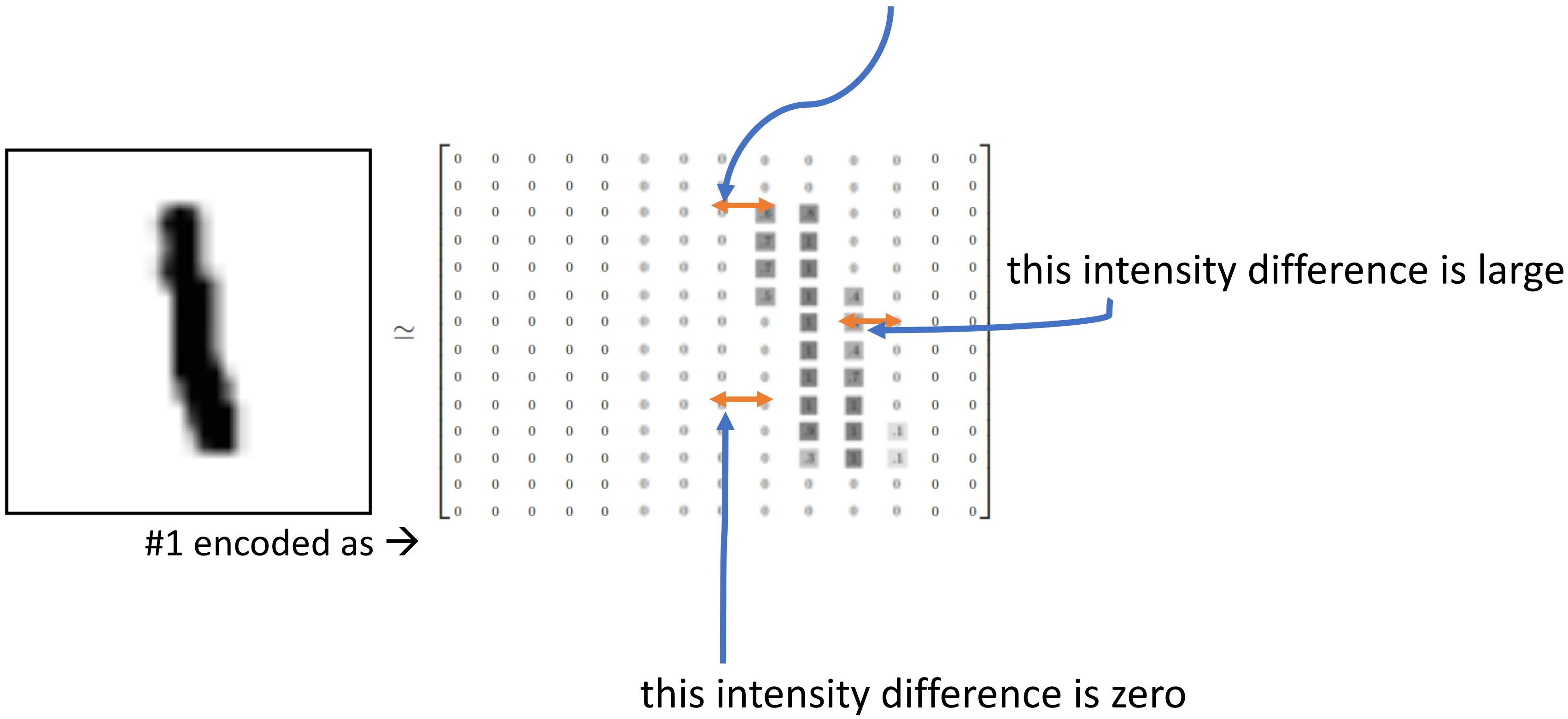
this pixel has a high intensity

#1 encoded as →

this pixel has a low intensity

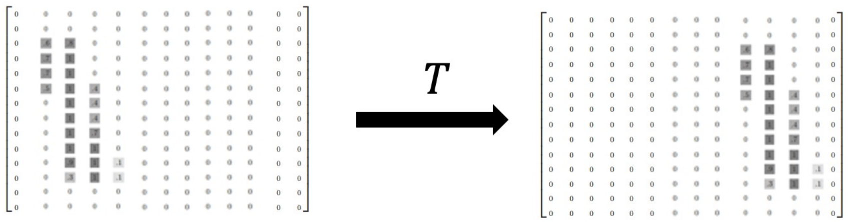
This would *not* be a problem for the human visual system

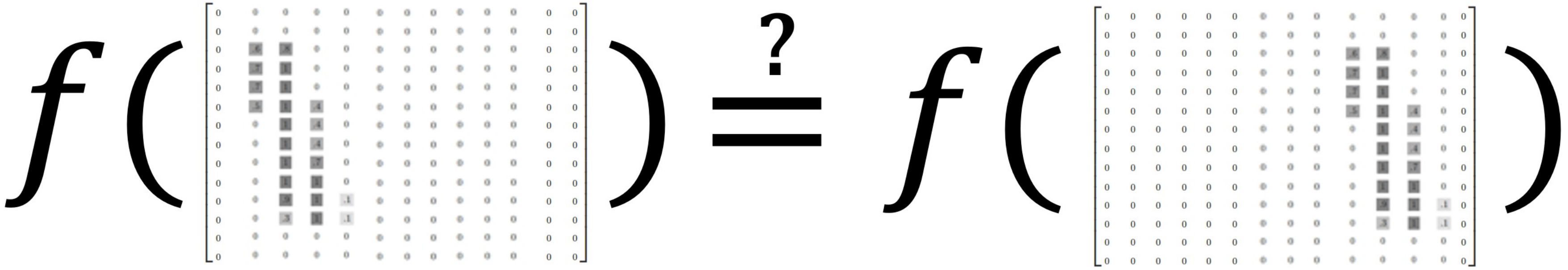
…but rather ***local differences*** in intensities



Translational Invariance

- To make a neural net f robust in this same way, it should ideally satisfy ***translational invariance***: $f\big(T(x)\big) = f(x)$, where
  - x is the input image
  - T is a translation (i.e. a horizonal and/or vertical shift)

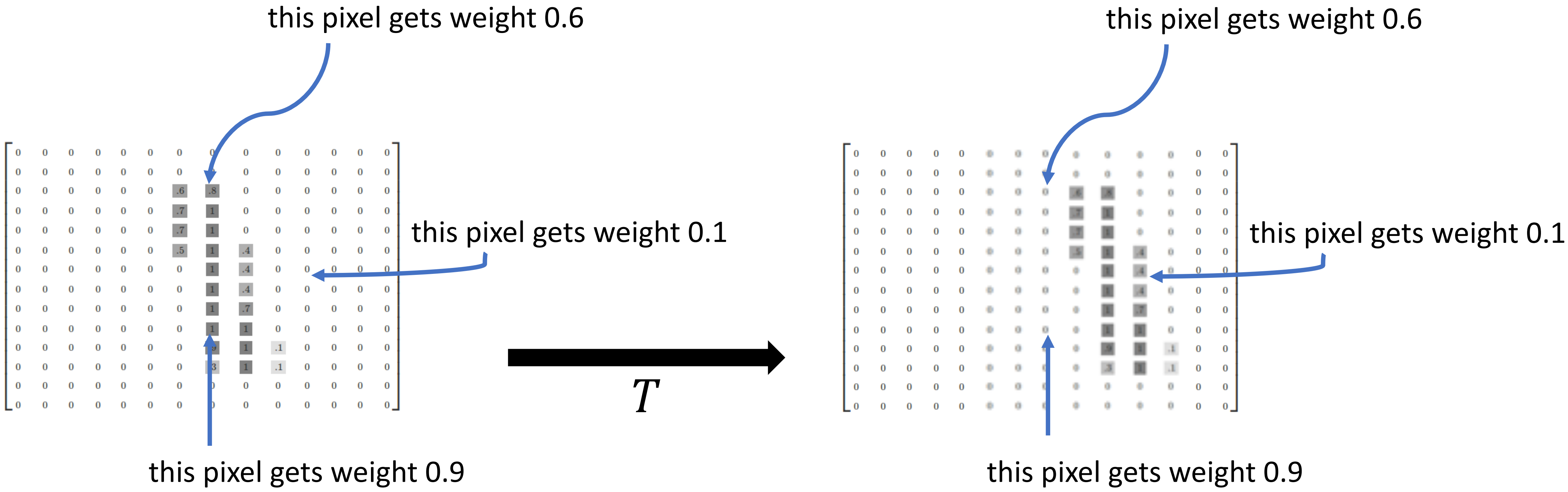




Fully Connected Nets are *not* Translationally Invariant

How to make the network translationally invariant?

Focus on local differences/patterns

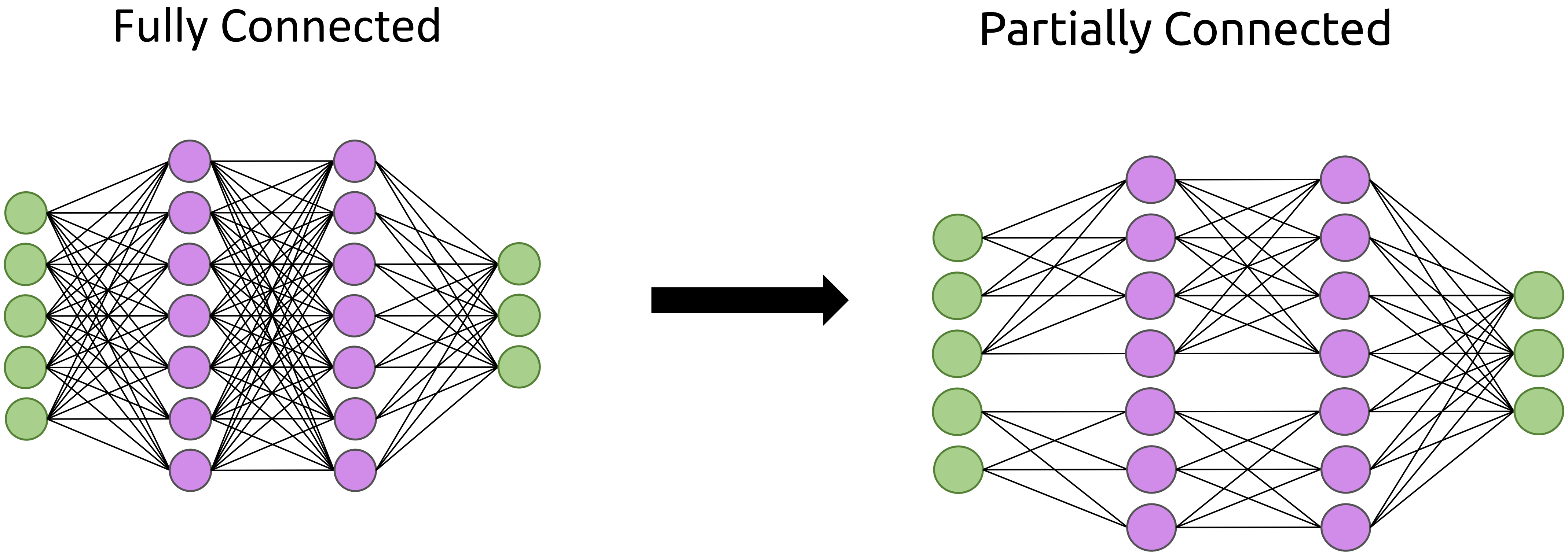


Sum of these three: $0.6 \cdot 0.8 + 0.1 \cdot 0 + 0.9 \cdot 1 = 1.38$

Sum of these three: $0.6 \cdot 0 + 0.1 \cdot 0.4 + 0.9 \cdot 0 = 0.4$

Any questions?

Focusing on local patterns = partial connections



How do we do that?

The Main Building Block: Convolution

Convolution is an operation that takes two inputs:

(1) An image (2D – B/W)

(2) A filter (also called a kernel)

1	1	1
0	0	0
-1	-1	-1

2D array of numbers; could be any values

What Convolution Does (Visually)

image

2	0	1	3
7	1	1	0
0	2	5	0
0	5	1	4

$\otimes$

filter/kernel

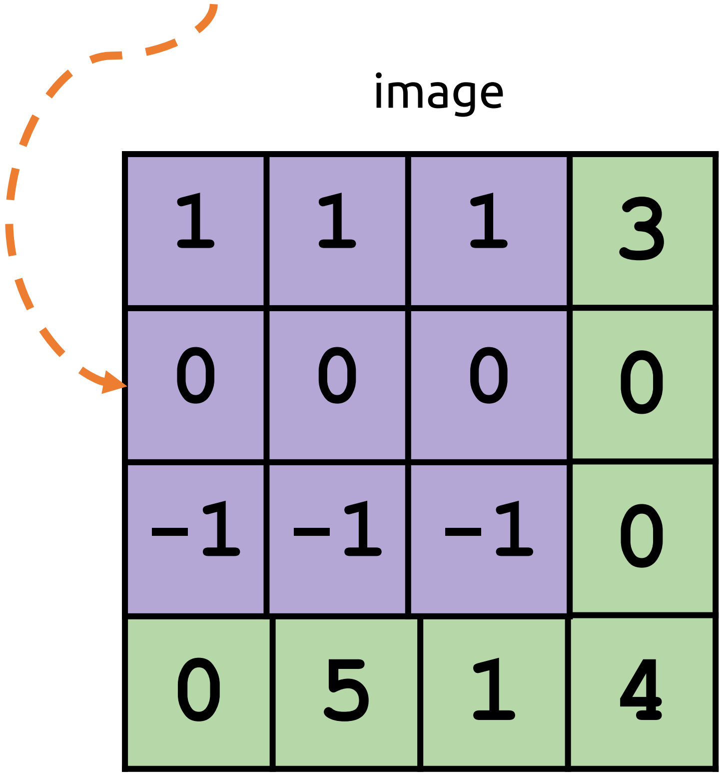
1	1	1
0	0	0
-1	-1	-1

(We use this symbol for convolution)
(The verb form is “convolve”)

What Convolution Does (Visually)

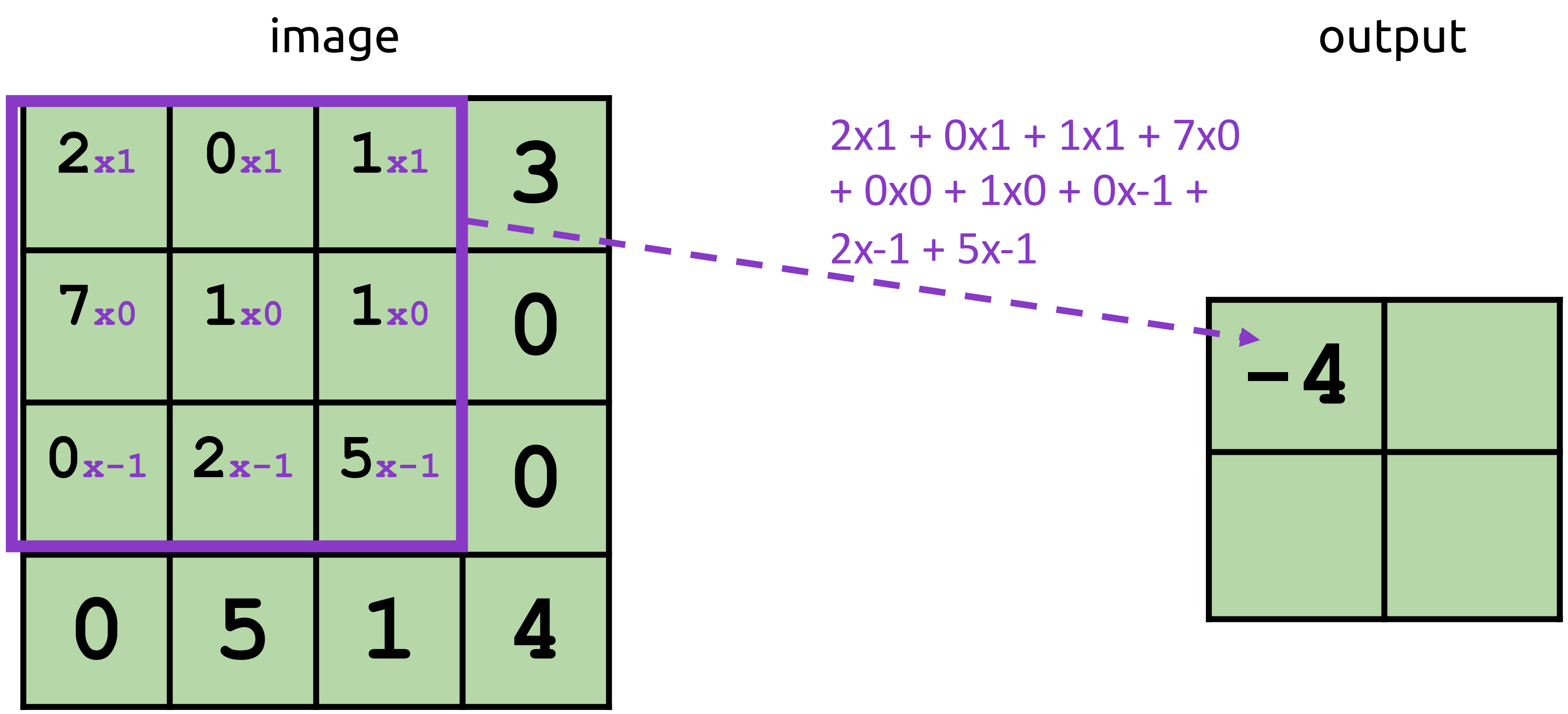
Overlay the filter on the image

image



What Convolution Does (Visually)

Sum up multiplied values to produce output value



What Convolution Does (Visually)

Move the filter over by one pixel

image

1	1	1	3
0	0	0	0
-1	-1	-1	0
0	5	1	4

output

-4	

What Convolution Does (Visually)

Move the filter over by one pixel

image

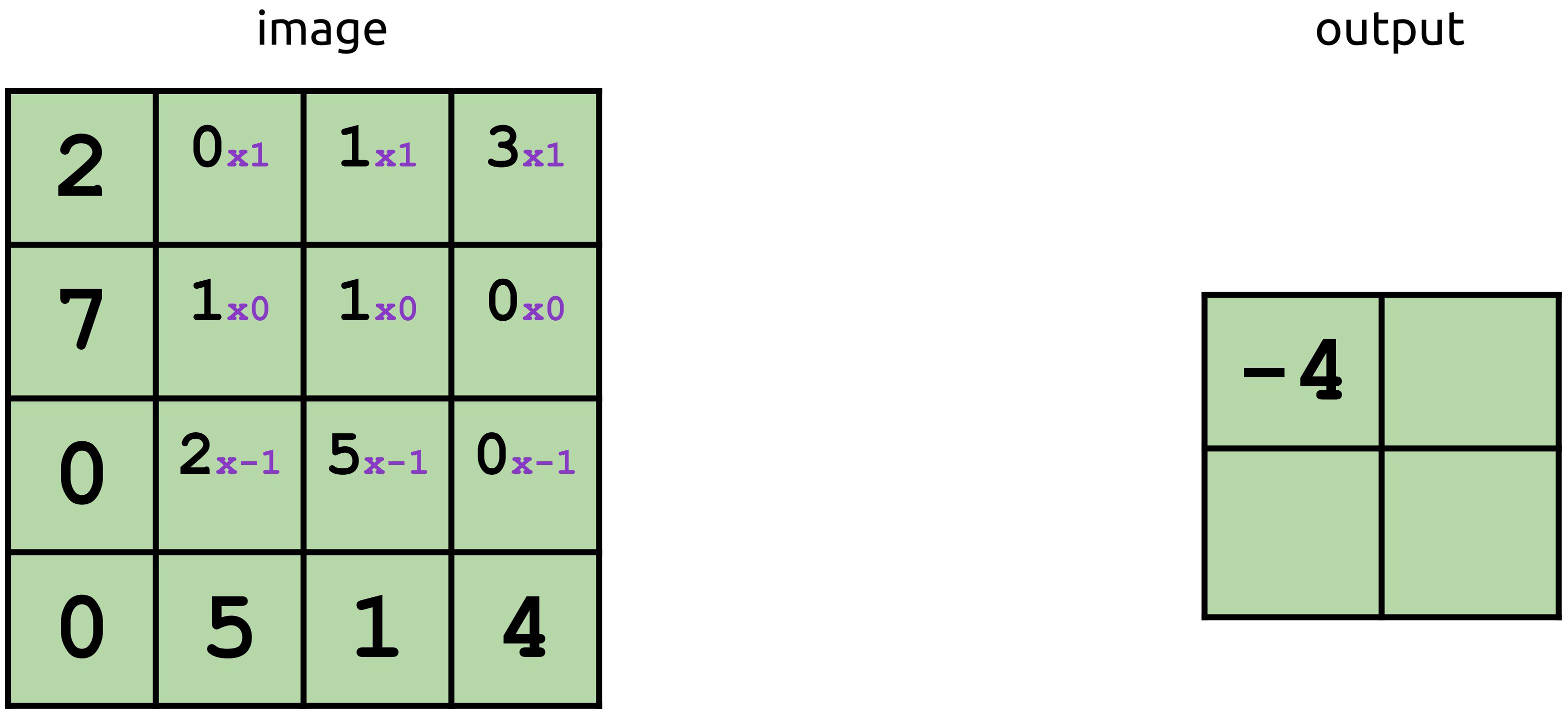
2	1	1	1
7	0	0	0
0	-1	-1	-1
0	5	1	4

output

-4	

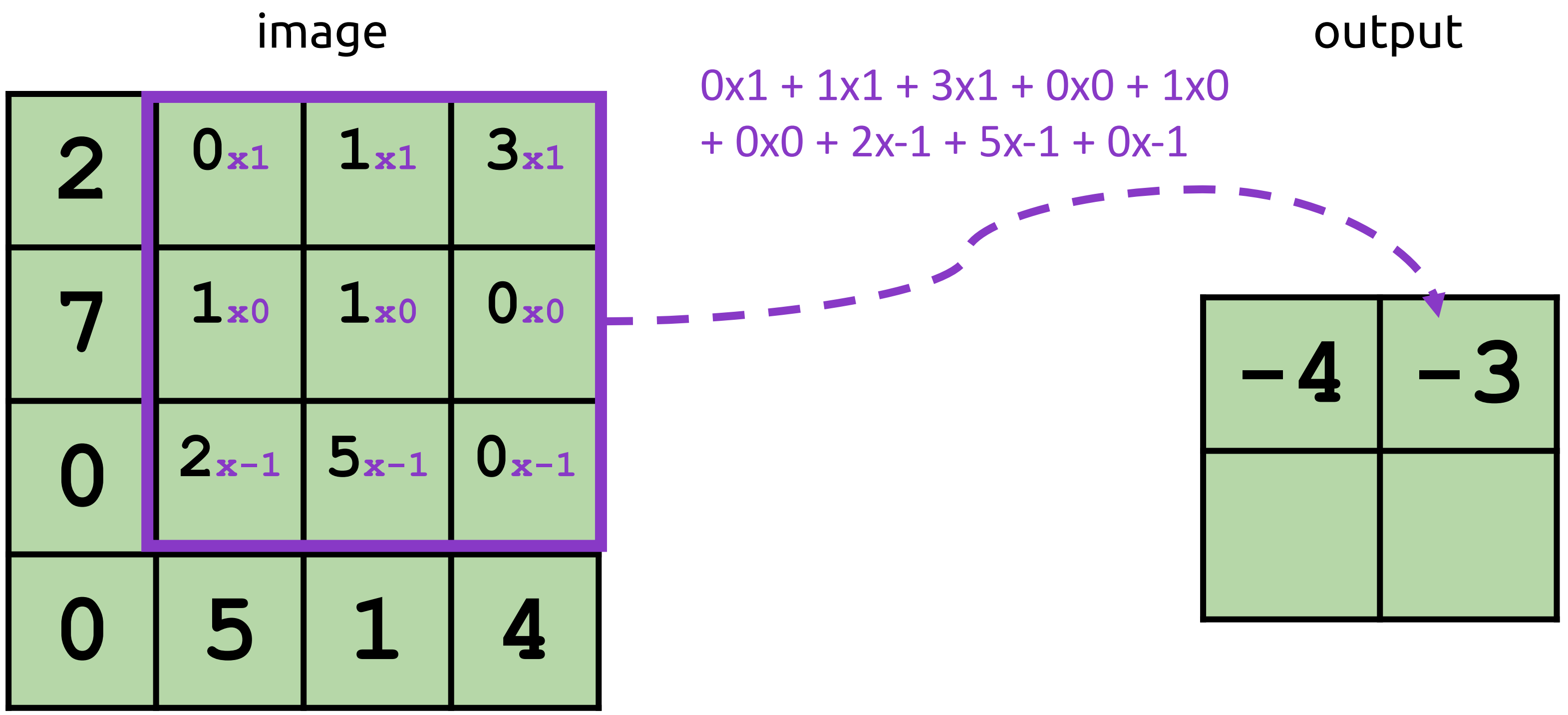
What Convolution Does (Visually)

Repeat (multiply, sum up)



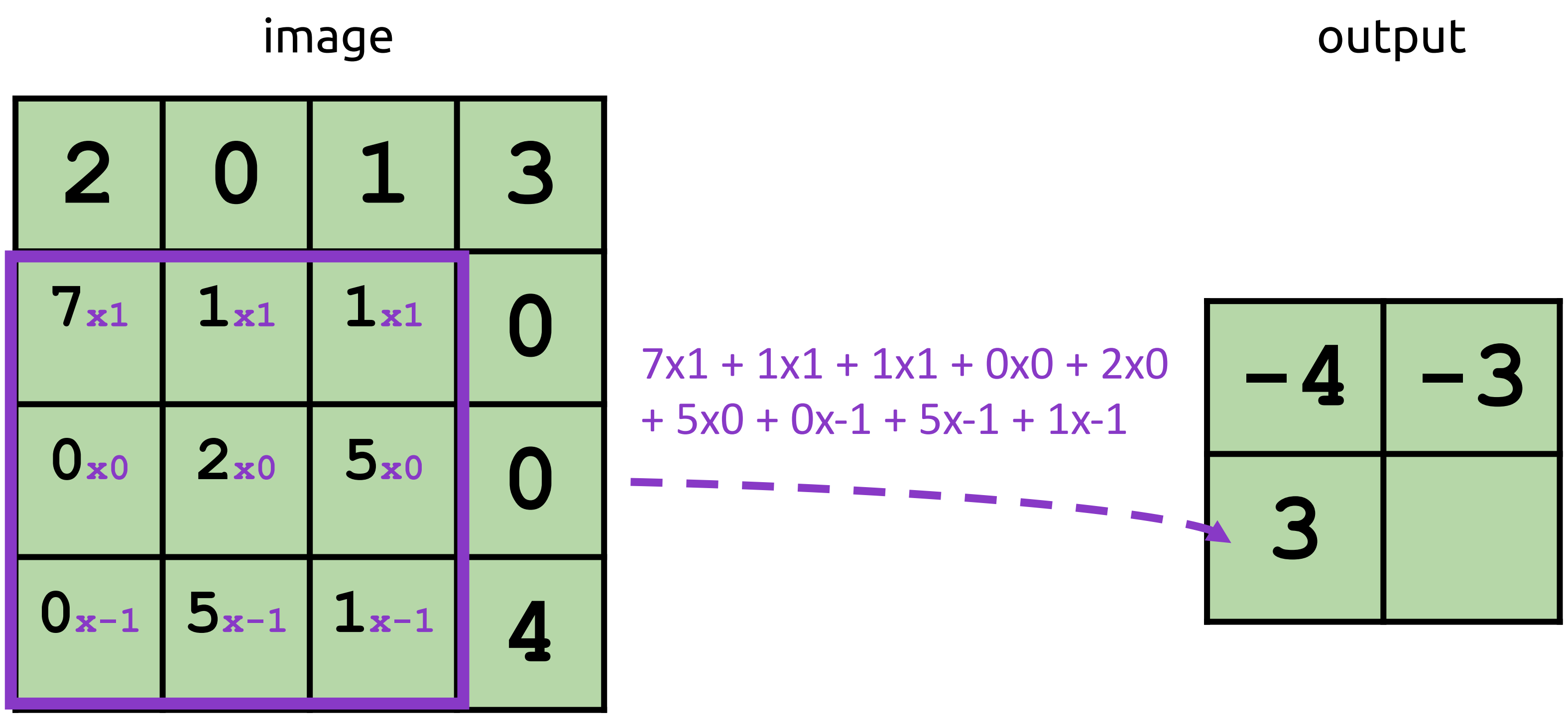
What Convolution Does (Visually)

Repeat (multiply, sum up)

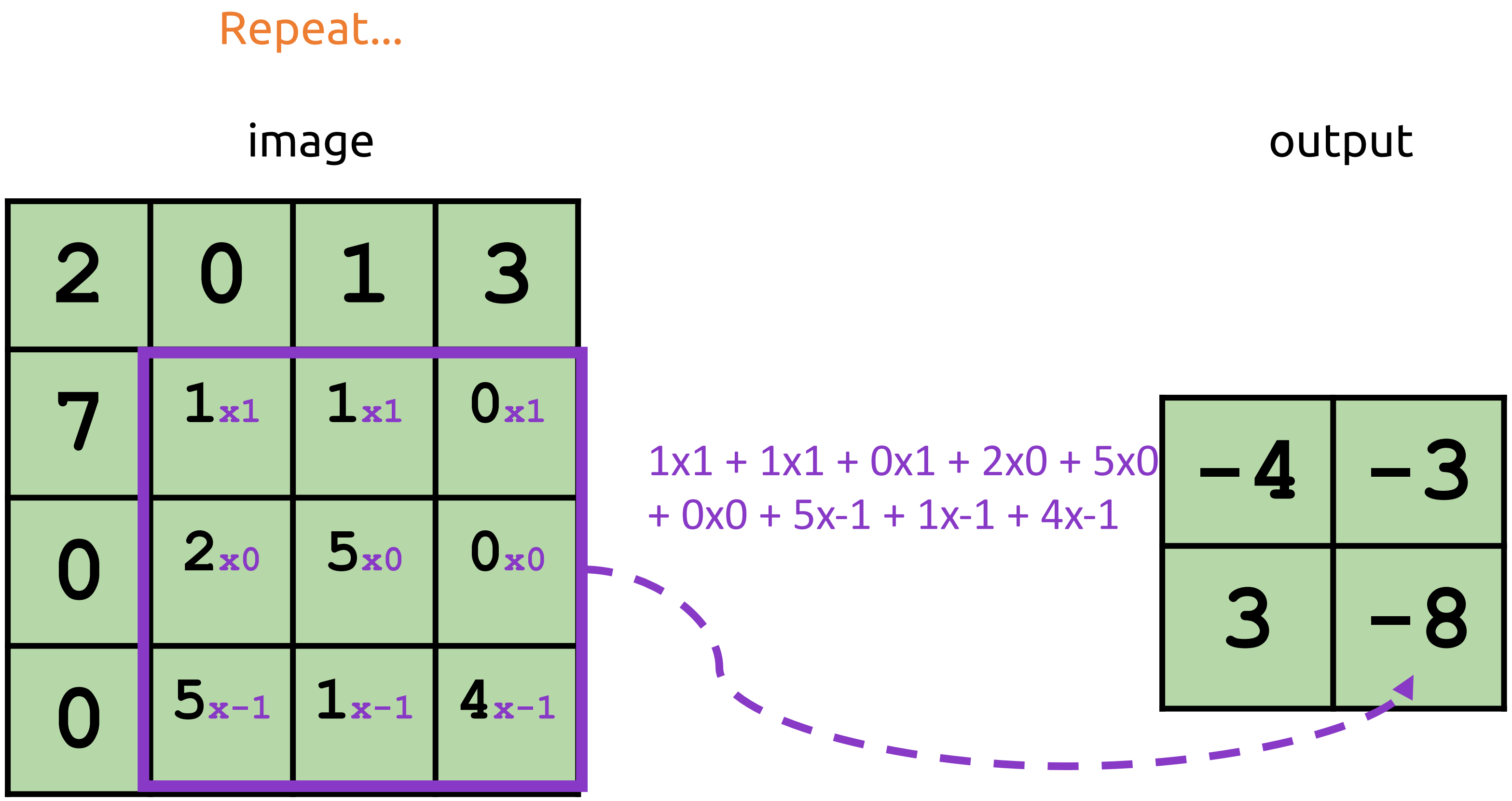


What Convolution Does (Visually)

Repeat...



What Convolution Does (Visually)



What Convolution Does (Visually)

In summary:

image

2	0	1	3
7	1	1	0
0	2	5	0
0	5	1	4

$\otimes$

filter/kernel

1	1	1
0	0	0
-1	-1	-1

$=$

output

-4	-3
3	-8

Try it out yourself!

Convolve this image

2	0	3	1
1	1	0	0
1	0	2	0
1	0	1	2

$\otimes$

With this filter

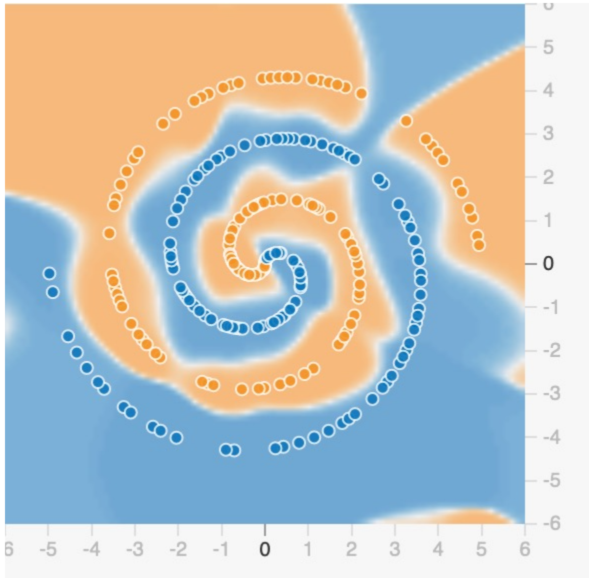
1	0	-1
2	0	-2
1	0	-1

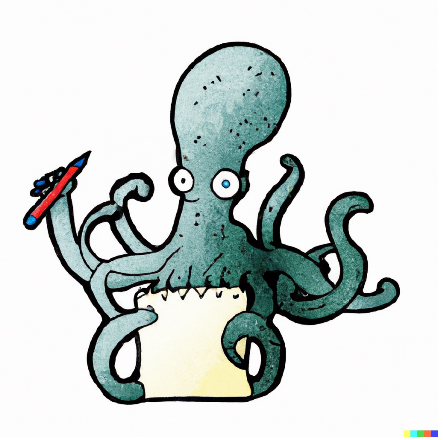
$$\begin{bmatrix}2 & 0 & 3 & 1\\ 1 & 1 & 0 & 0\\ 1 & 0 & 2 & 0\\ 1 & 0 & 1 & 2\end{bmatrix} \otimes \begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$$

Recap

Building multi-layer neural networks

- Hidden layers
- What a one-hidden layer network can learn
- What a multi-layer network can learn





Introduction to CNNs

- Partially connected networks are useful (e.g., for images!)
- Fully connected networks are not transitionally invariant
- Convolutional filter