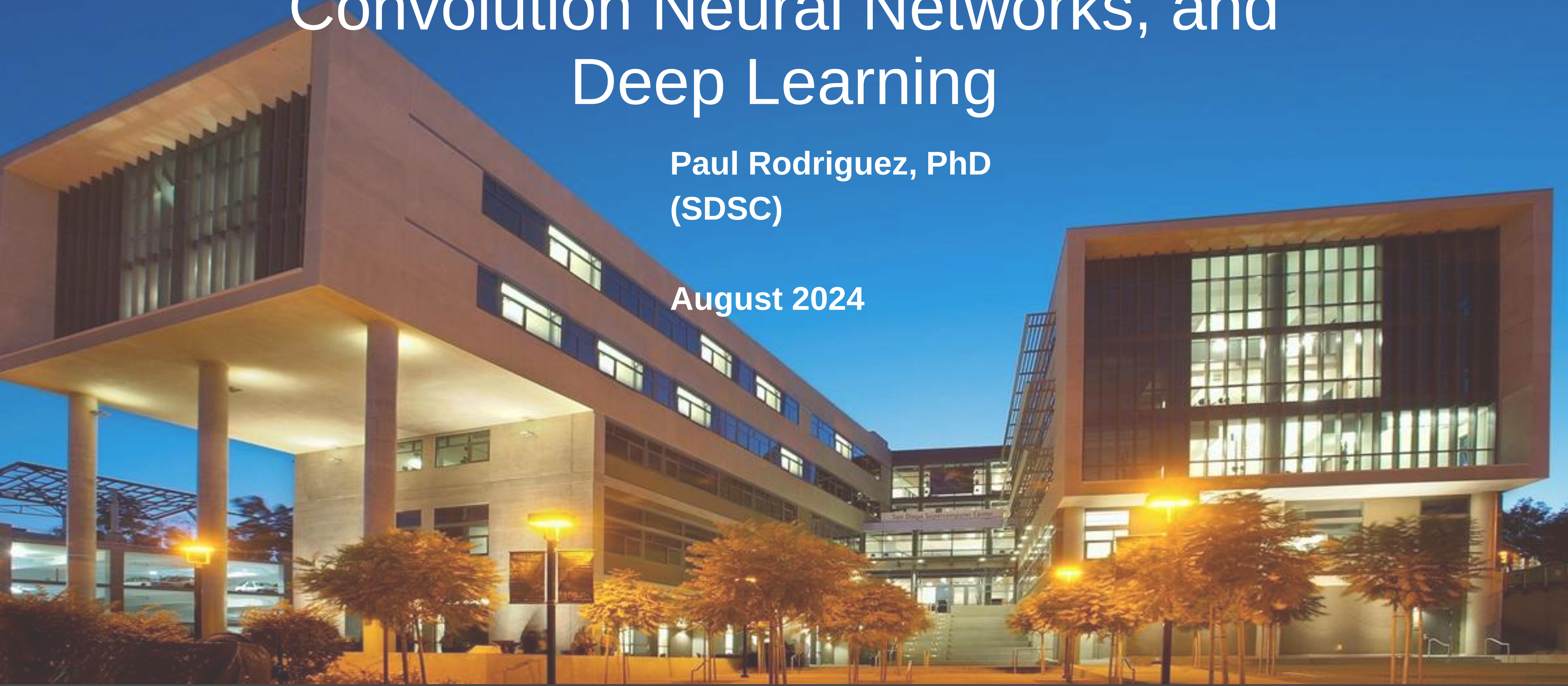
Introduction to Neural Network, Convolution Neural Networks, and Deep Learning

Paul Rodriguez, PhD
(SDSC)

August 2024



Outline

- **Part I**

Overview of Neural Networks (aka Multilayer Perceptron)

Convolution Neural Networks and Scaling

Exercise, MNIST classification

- **Part II**

Practical Guidelines: Hyperparameters, Workflows, Batchjobs, GPUs

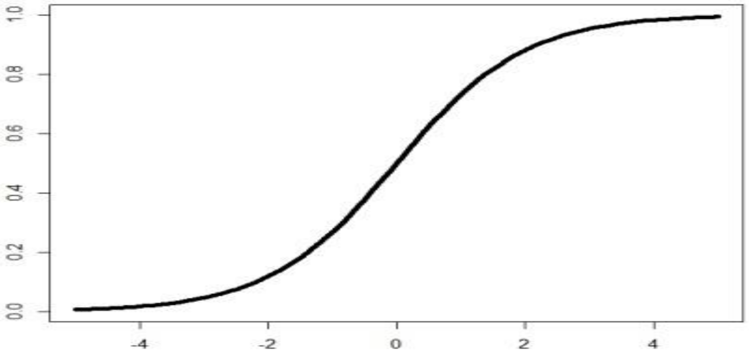
Exercise, Multinode MNIST

Logistic Regression to Neural Network

$$f(x,b,w) = \frac{\exp^{(b+w*x)}}{1+\exp^{(b+wx)}} = \frac{1}{1+exp^{(-(b+wx))}}$$

for parameters: $b = 0$, $w_1 = 1$

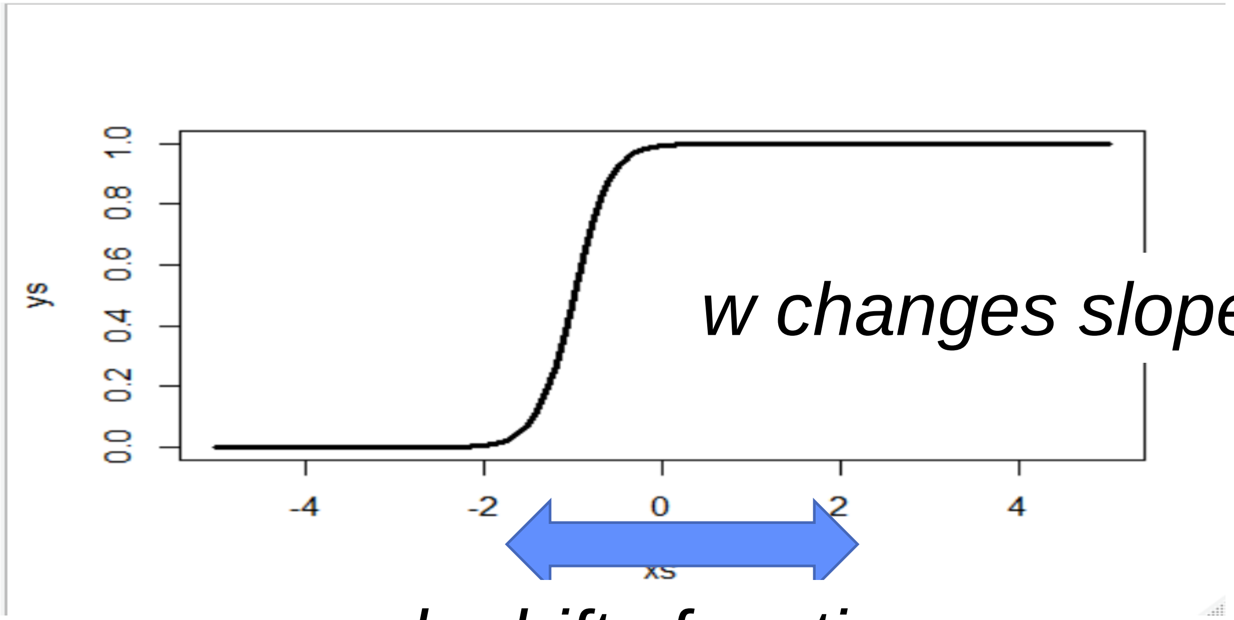
$y = f(x)$



x

Logistic Regression to Neural Network

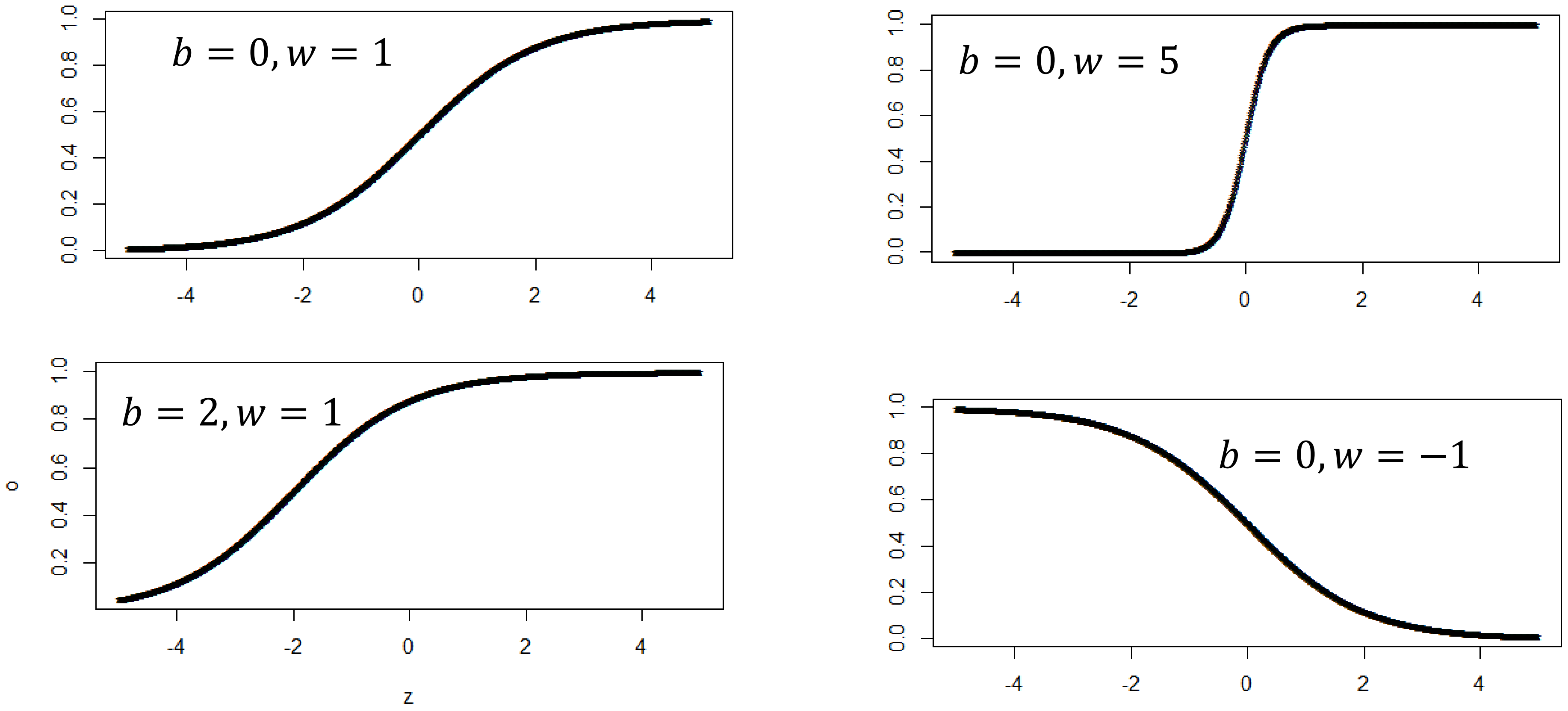
$$f(x, b, w) \quad = \quad \frac{\exp^{(b+w*x)}}{1 + \exp^{(b+wx)}}$$



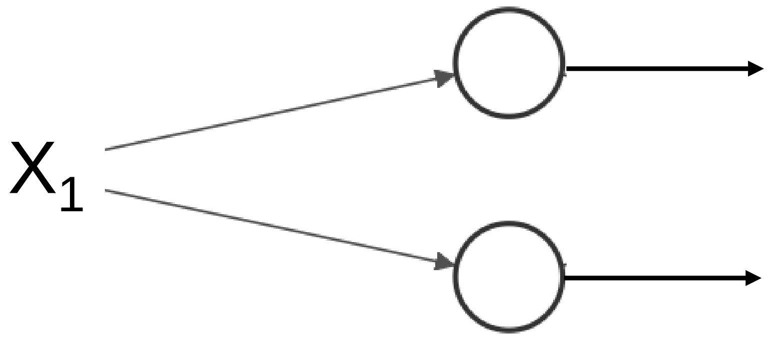
w changes slope

b shifts function

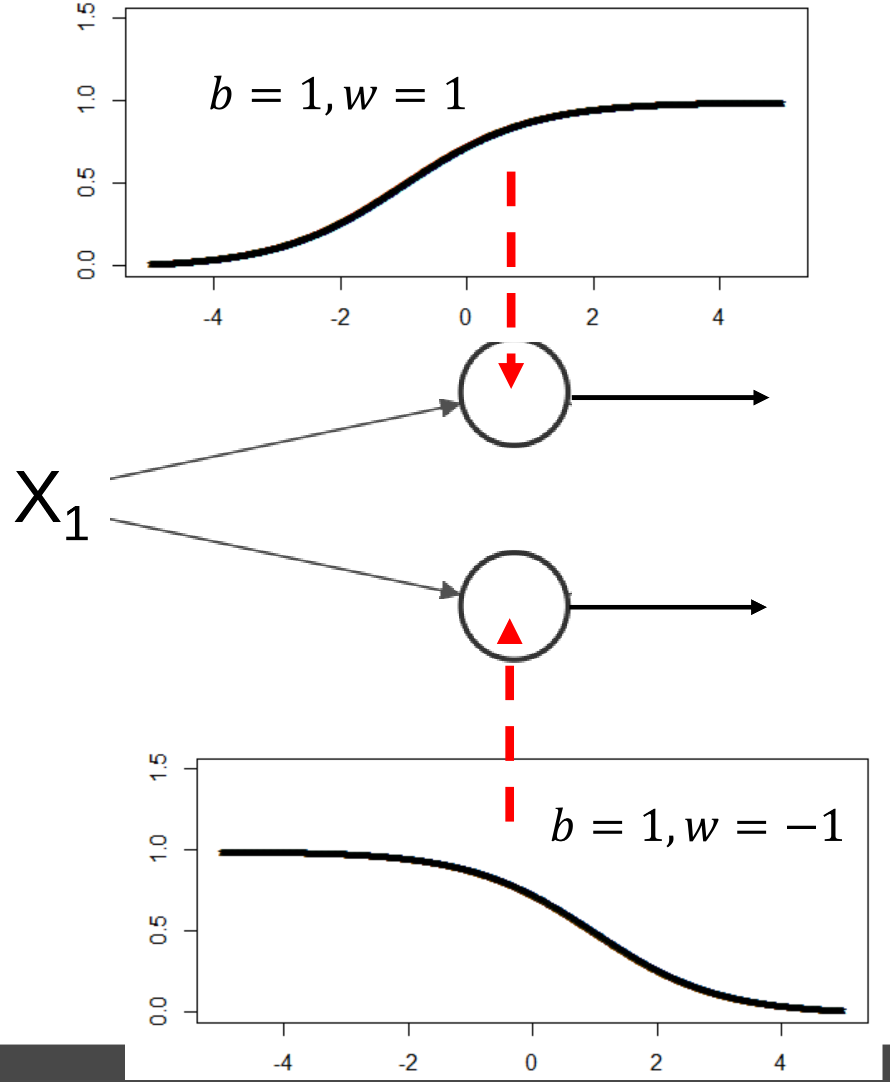
Logistic function w/various weights



Example: 1 input into 2 logistic units

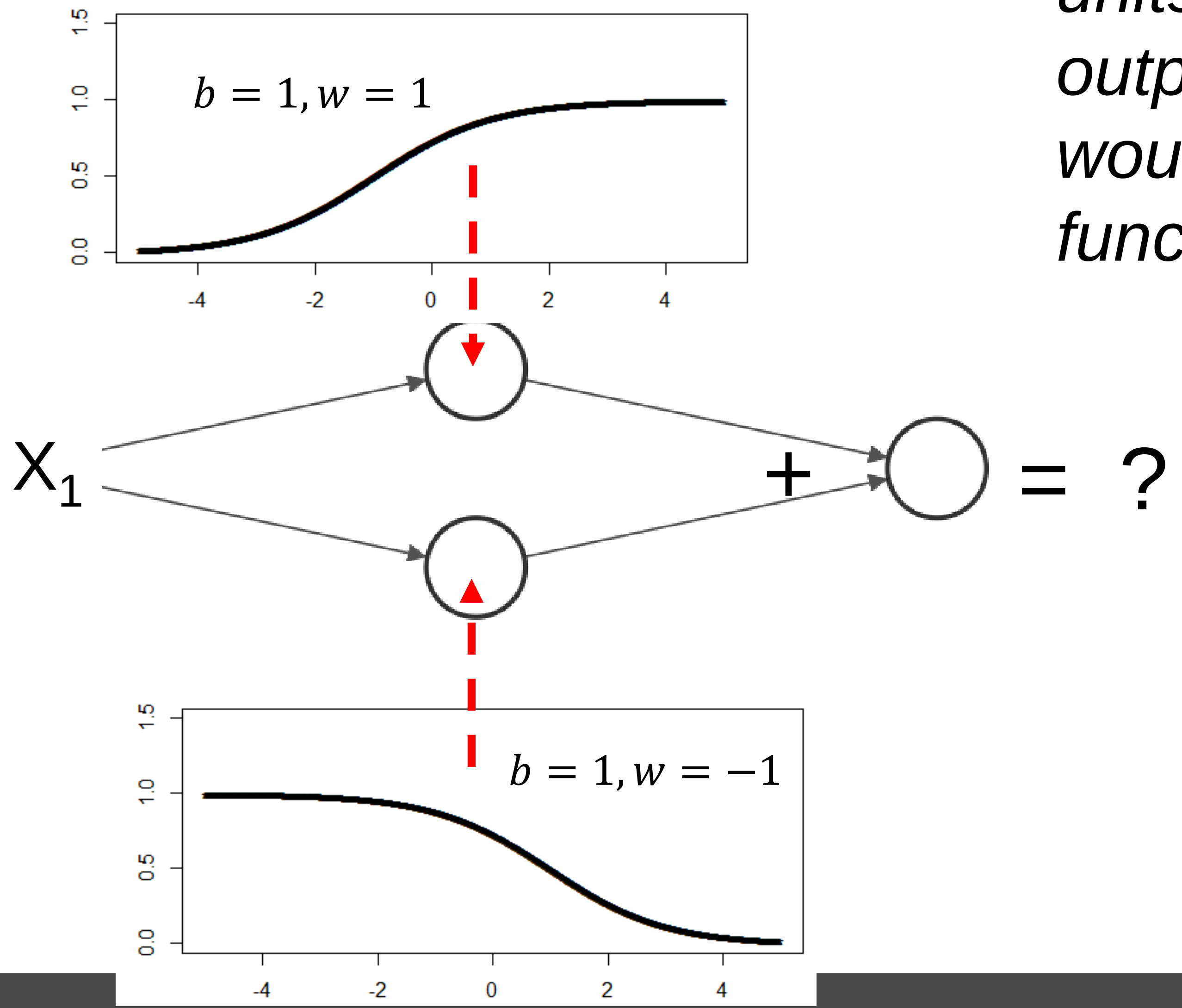


Example: 1 input into 2 logistic units with these activations



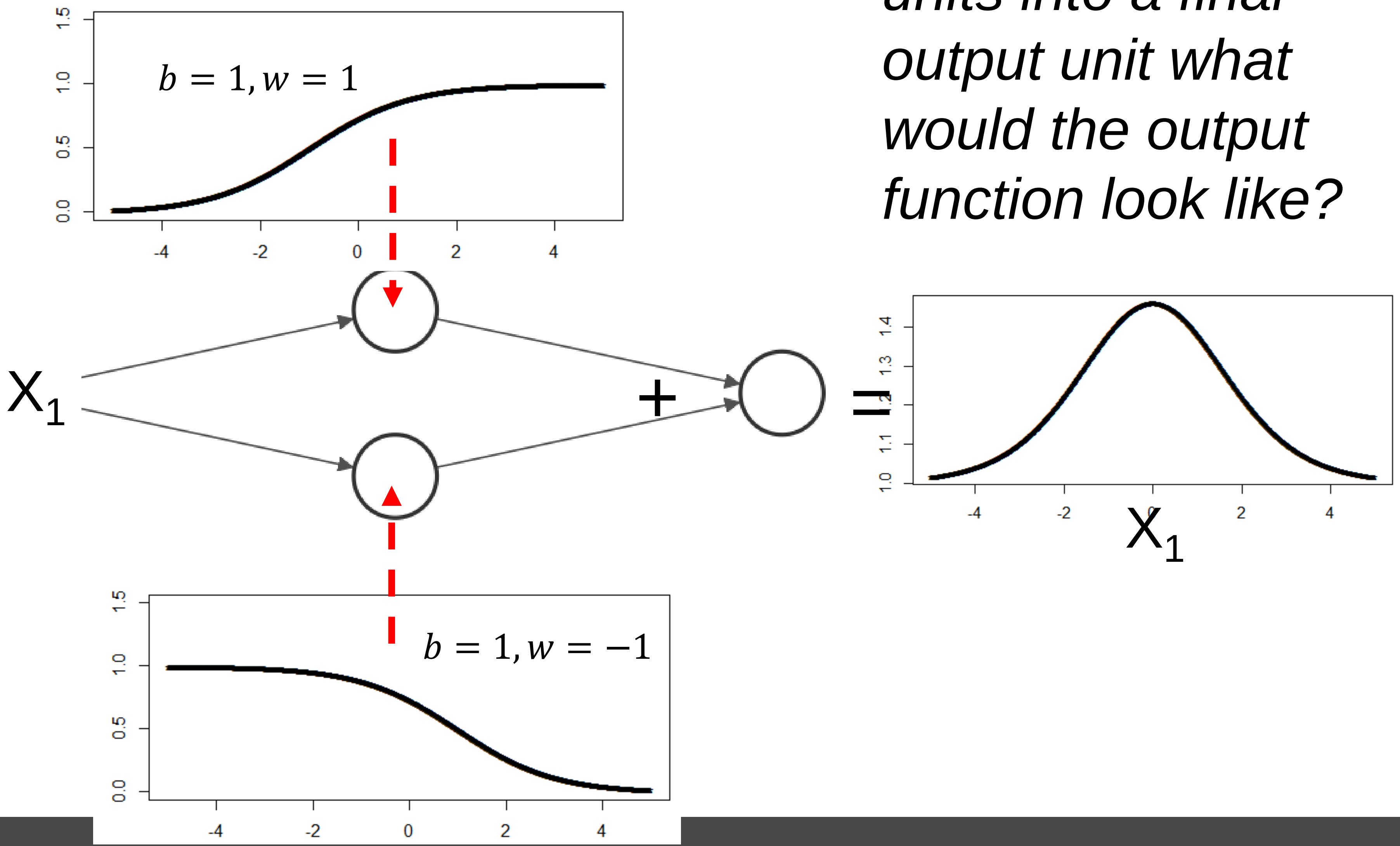
Example: 1 input into 2 logistic units with these activations

If you add these 2 units into a final output unit what would the output function look like?



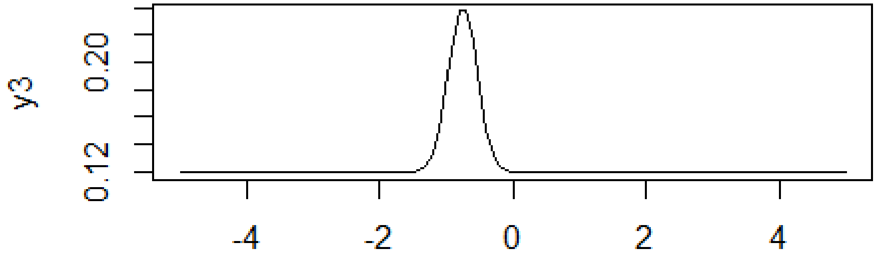
Example: 1 input into 2 logistic units with these activations

If you add these 2 units into a final output unit what would the output function look like?

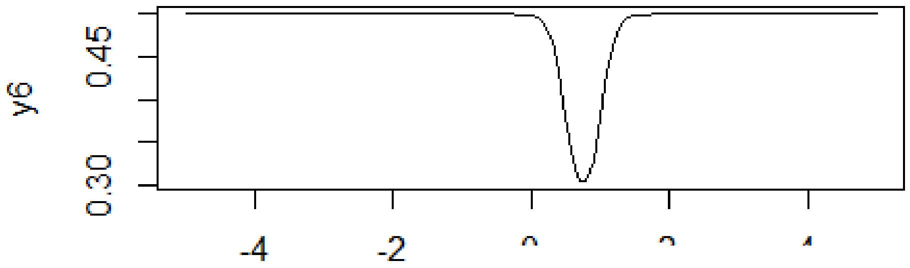


Higher level function combinations

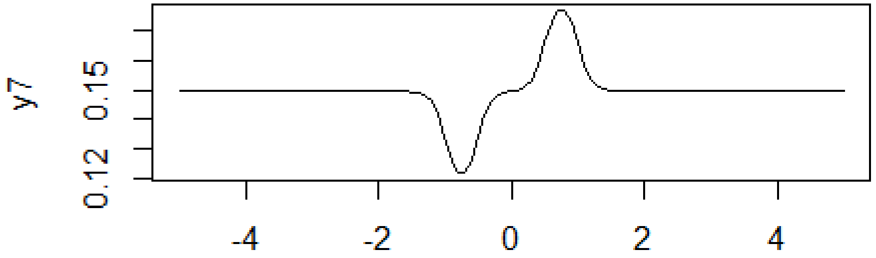
```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```




```
y4=1/(1+exp(10+ (-10)*x))
y5=1/(1+exp(-5+(10)*x))
y6=1/(1+exp(1-1*y4-1*y5))
plot(x,y6,type="l")
```

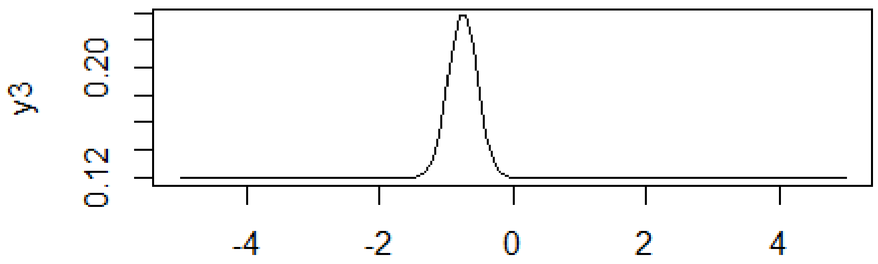



```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```



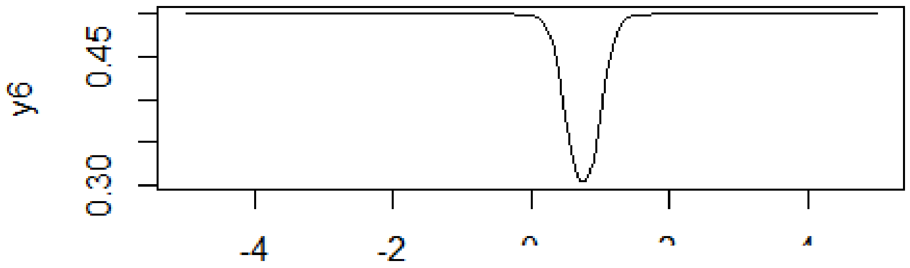
Higher level function combinations

```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```

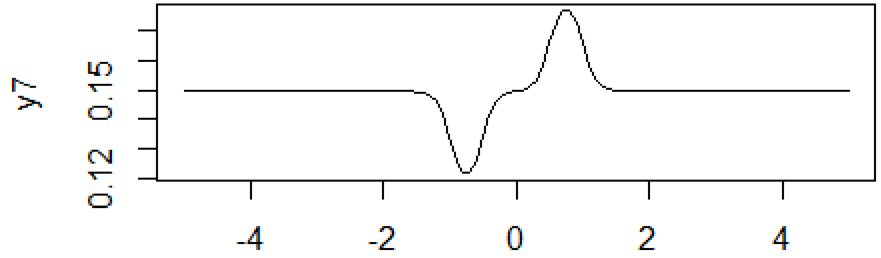


Multiple layer networks can represent any logical or real-valued functions (unbiased, but potential to overfit)

```
y4=1/(1+exp(10+ (-10)*x))
y5=1/(1+exp(-5+(10)*x))
y6=1/(1+exp(1-1*y4-1*y5))
plot(x,y6,type="l")
```




```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```



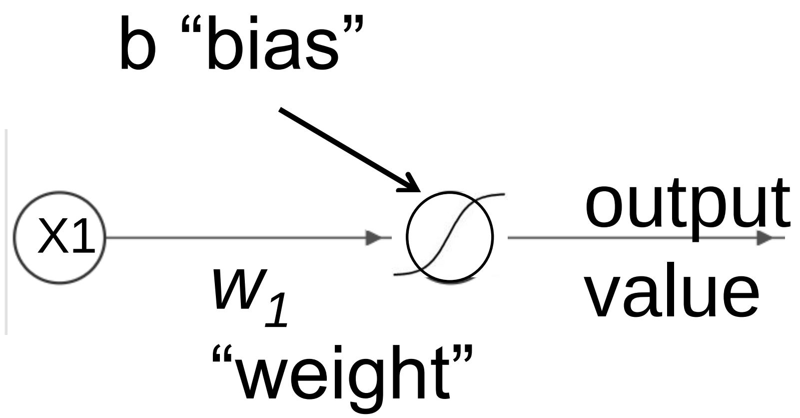
Logistic to Neural Network model

$$f(x, b, w) \quad = \quad \frac{\exp^{(b+w*x)}}{1+\exp^{(b+wx)}}$$

Draw out function as a little graph, 1 input

Logistic to Neural Network model

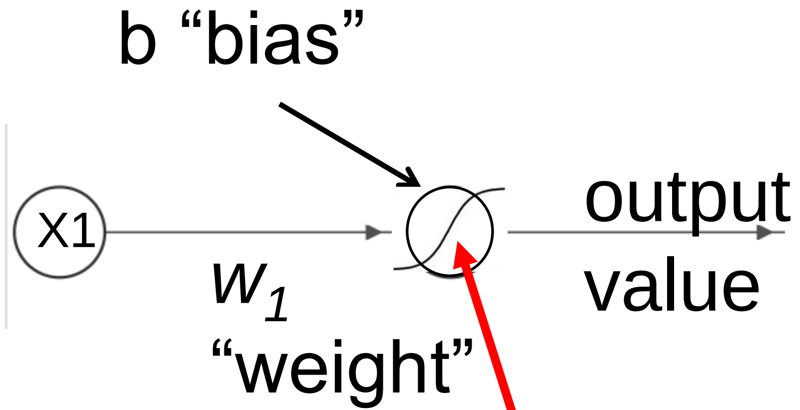
$$f(x,b,w) = \frac{\exp^{(b+w*x)}}{1+\exp^{(b+wx)}}$$



Draw out function as a little graph, 1 input

Logistic to Neural Network model

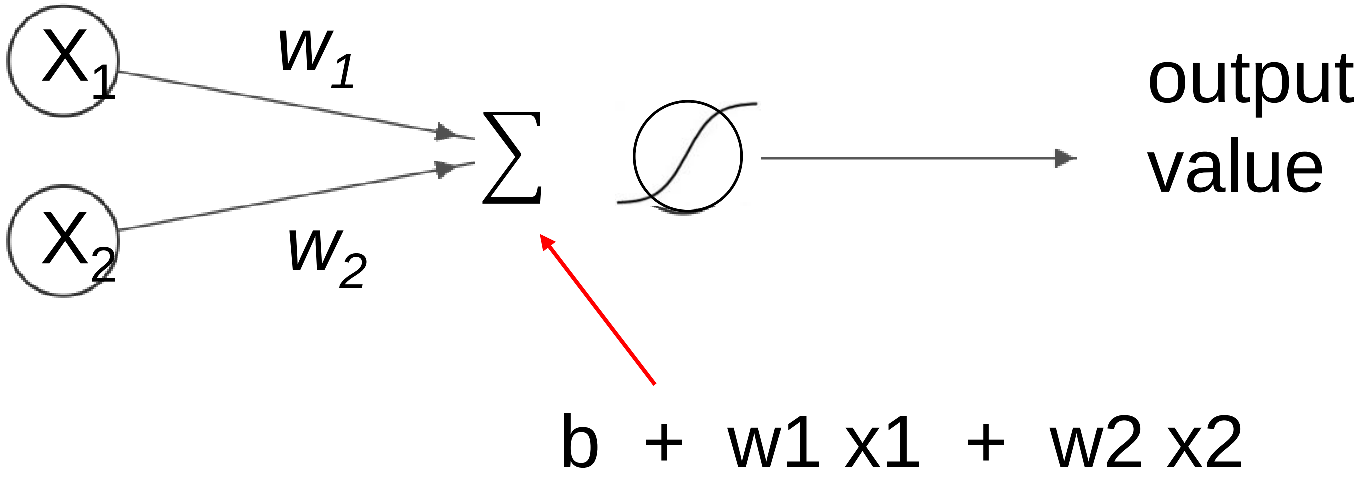
$$f(x,b,w) = \frac{\exp^{(b+w*x)}}{1+\exp^{(b+wx)}}$$



Draw out function as a little graph, 1 input

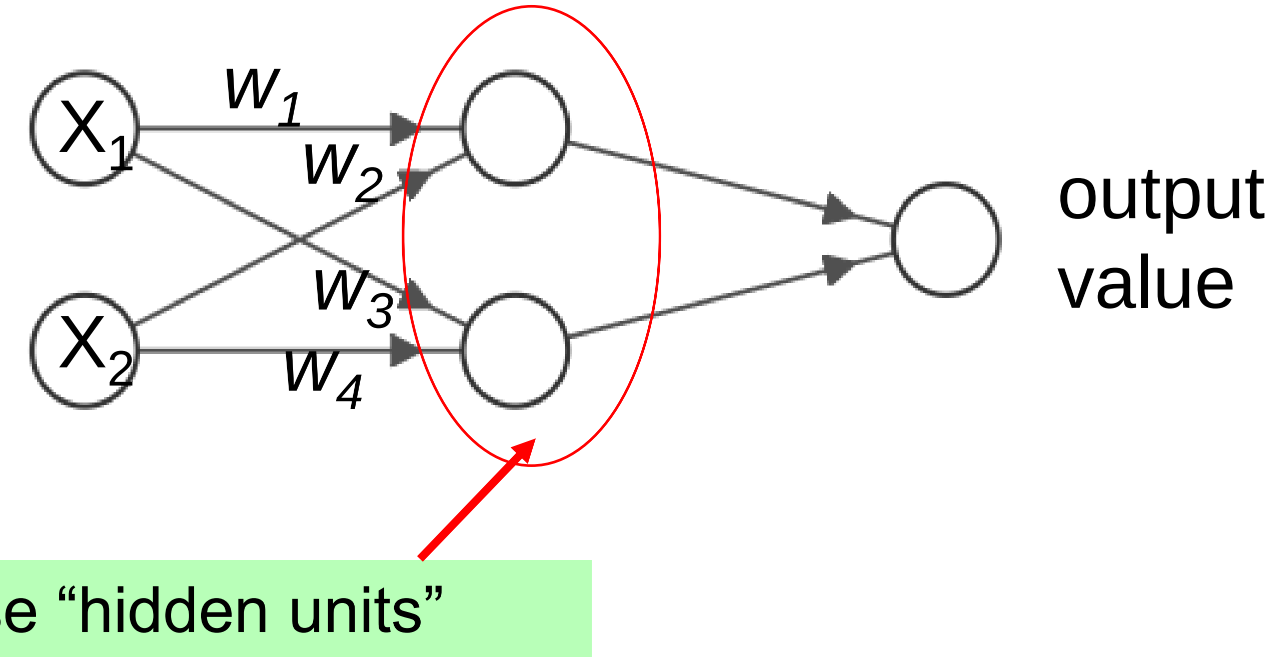
logistic function will transform input to output – call it the 'activation' function

Using 2 input units, the graph model would be:

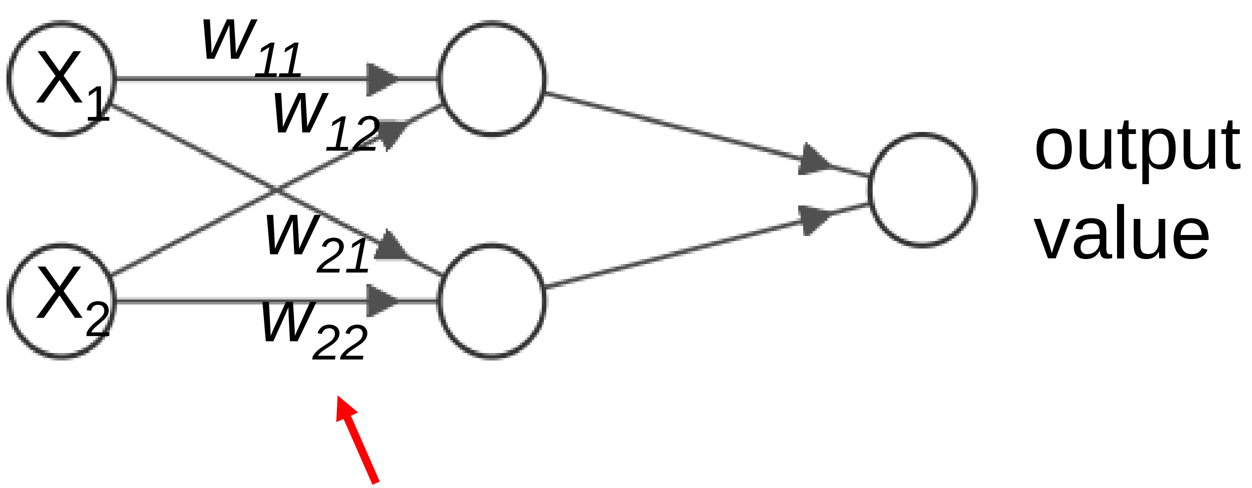


We usually don't draw the bias.
We assume inputs*weights are summed (e.g. a dot product)

Using 2 input units, 2 intermediate units, and 1 output:

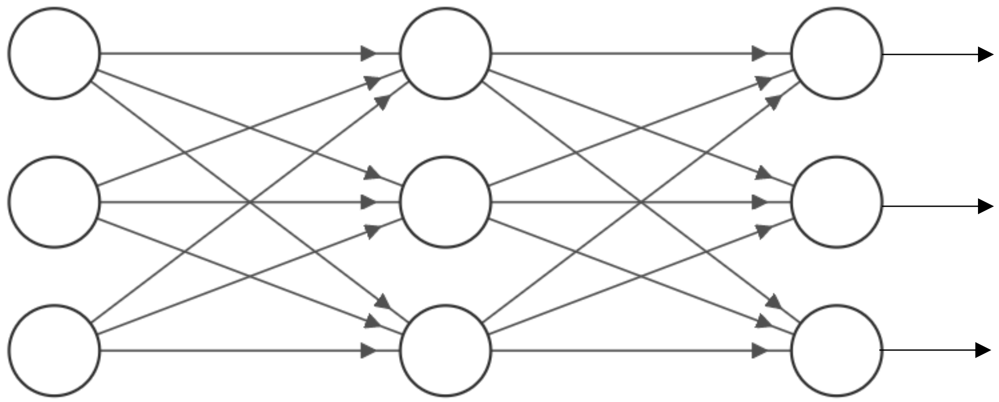


Using 2 input units, 2 intermediate units, and 1 output:



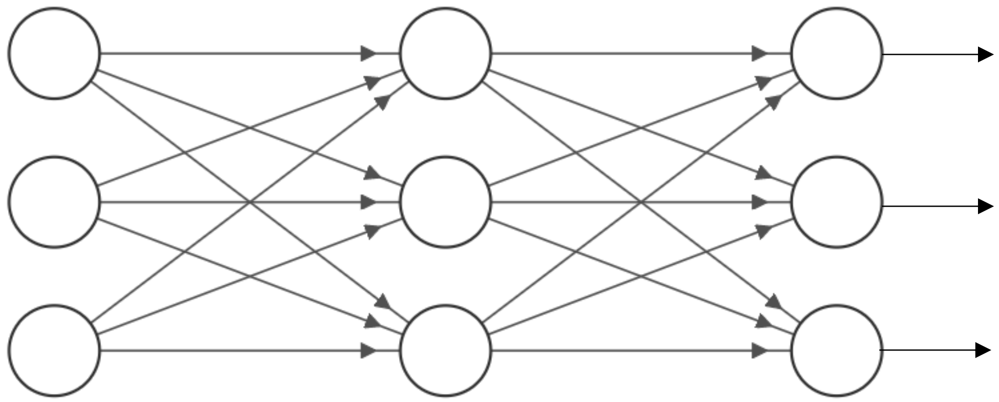
For X a Px1 vector, we set up a weight matrix W so that:
W*X are the activations going **forward** to hidden units

More generally, we can add a hidden layer, and have many inputs and outputs



A “Multilayer Perceptron”

1 Input layer 1 Hidden layer 1 Output

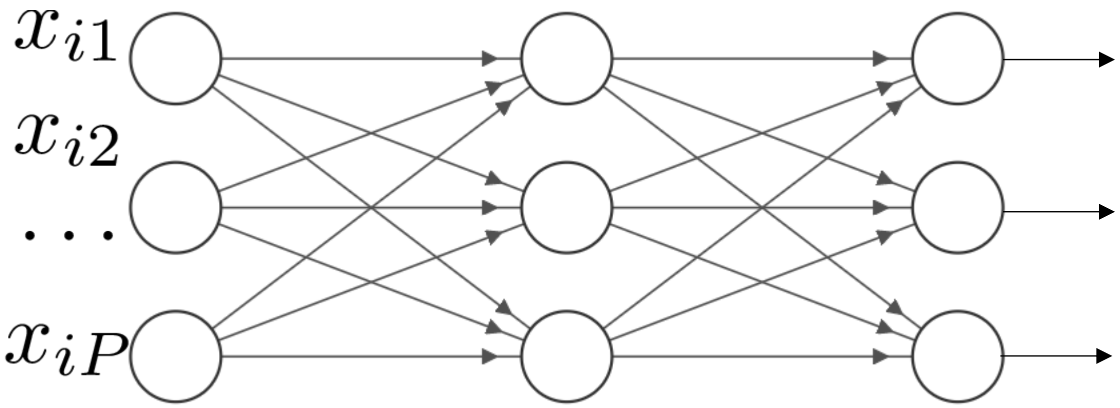


A “Multilayer Perceptron”

1 Input layer 1 Hidden layer 1 Output

The *i-th* input is a vector

(with P elements)



A “Multilayer Perceptron”

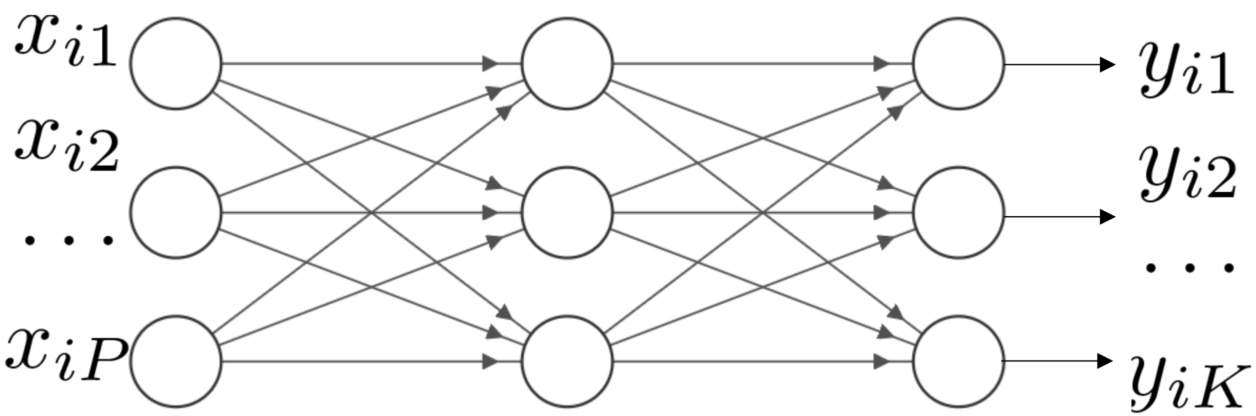
1 Input layer

1 Hidden layer

1 Output

The *i-th* input is a vector

(with P elements)



The *i-th* output can also be a vector

hidden units

A “Multilayer Perceptron”

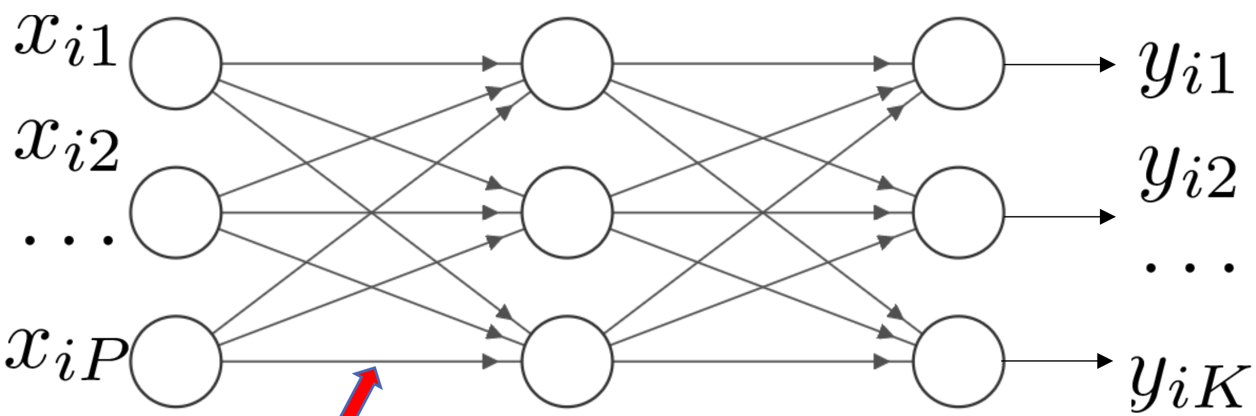
1 Input layer

1 Hidden layer

1 Output

The *i-th* input is a vector

(with P elements)



The *i-th* output can also be a vector

hidden units

1 weight parameter for each connection

A “Multilayer Perceptron”

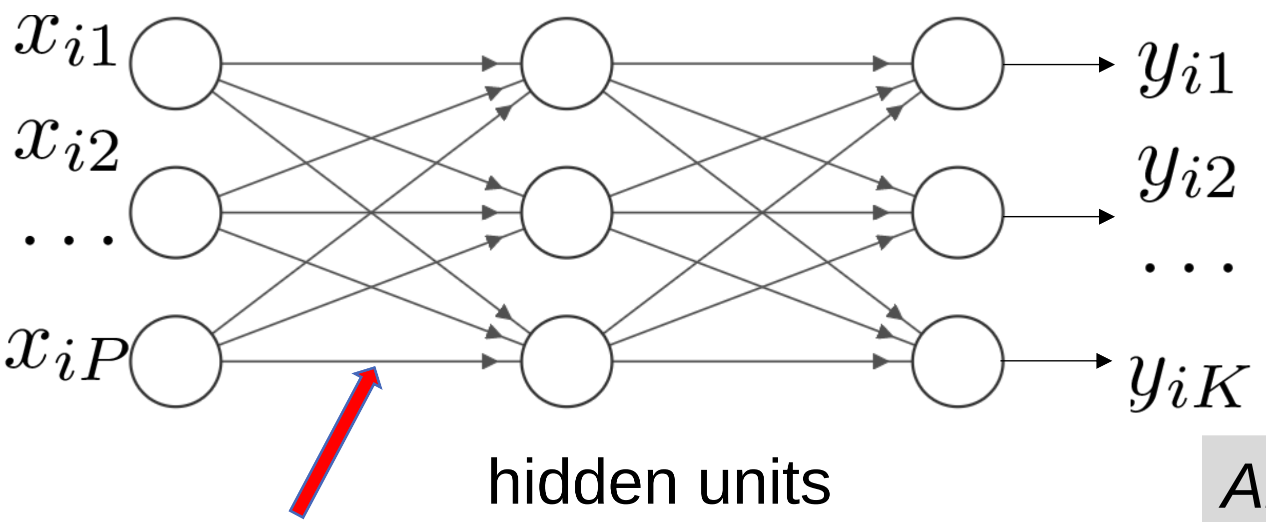
1 Input layer

1 Hidden layer

1 Output

The *i-th* input is a vector

(with P elements)



The *i-th* output can also be a vector

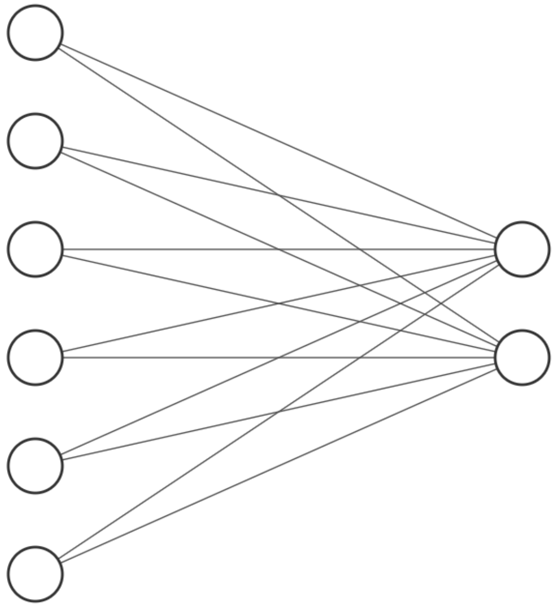
hidden units

1 weight parameter for each connection

And we usually have a ‘target’ such that (Target – Y) is the error

Quick side note: fewer units at the hidden layer creates an ‘***embedding***’ of the X input into a lower dimension

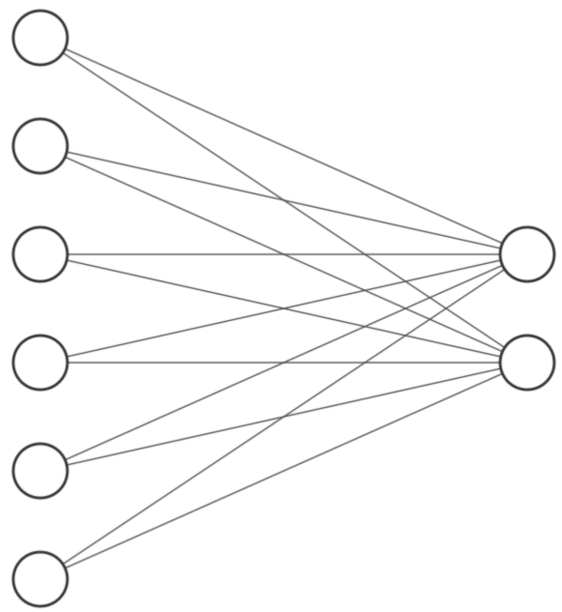
Here, the input vector has 6 values, so it’s 6 dimensions



This vector has 2 dimensions

Quick side note: fewer units at the hidden layer creates an ‘***embedding***’ of the X input into a lower dimension

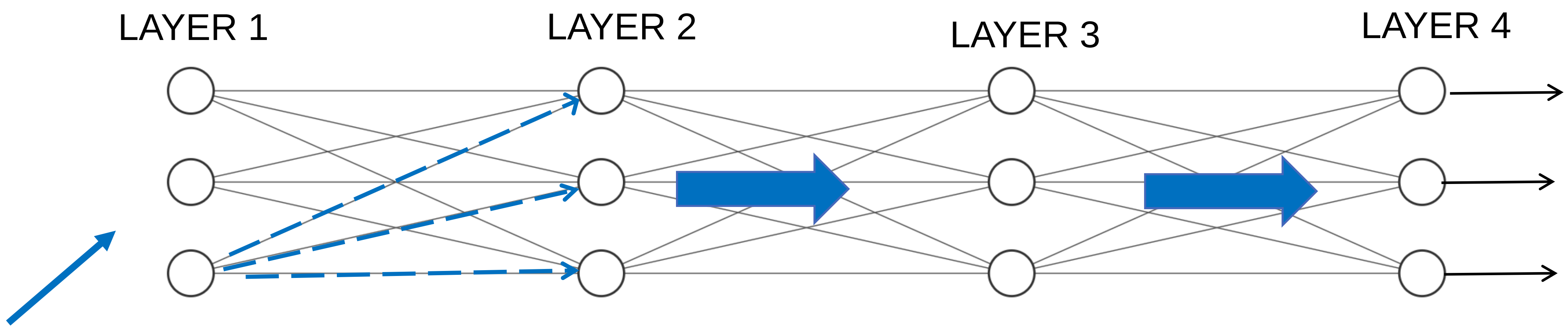
Here, the input vector has 6 values, so it’s 6 dimensions



This vector has 2 dimensions.

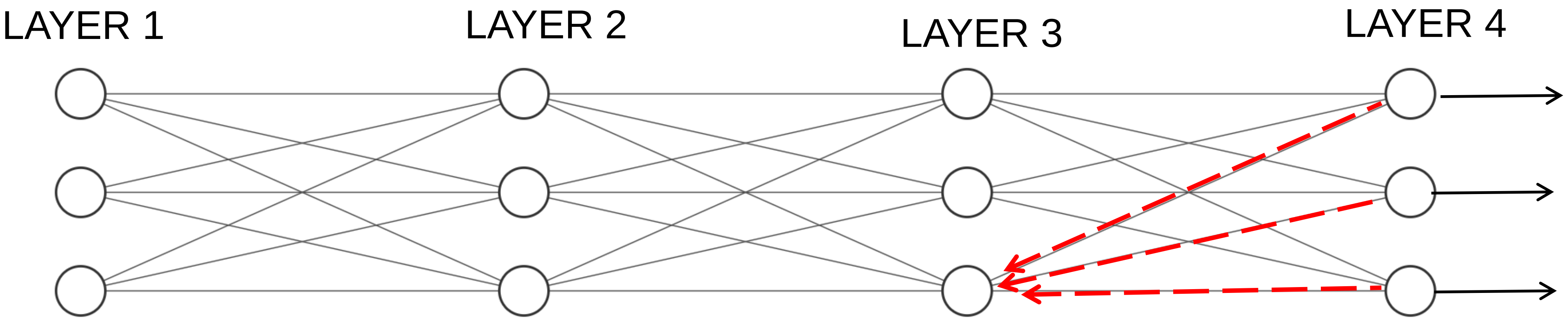
Learning a good embedding means learning the most relevant information (i.e. signal) for the task

Algorithm steps



1. FORWARD PROPAGATE INPUTS to get OUTPUTS

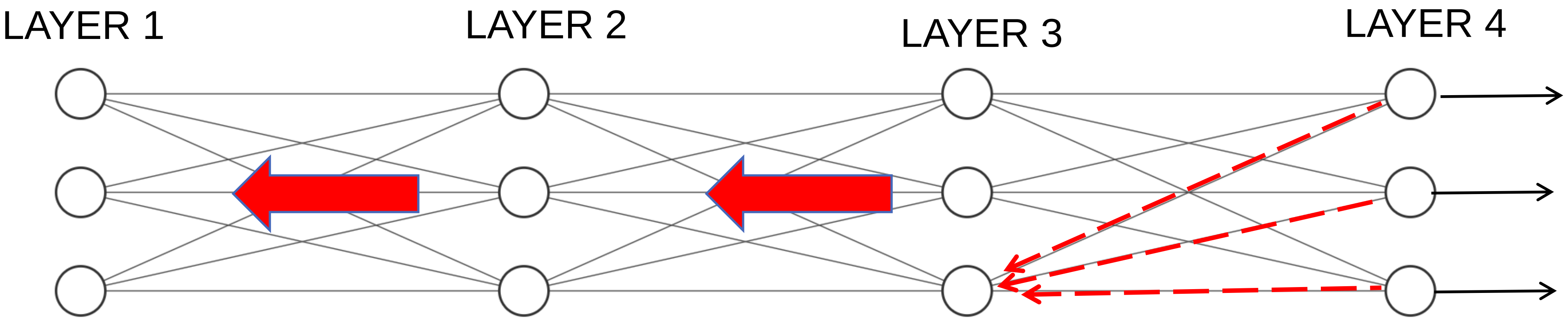
Algorithm steps



2. BACKWARD PROPAGATE ERROR USING DERIVATIVE CHAIN RULE:

$$\frac{dE}{dw_{mp}} = \sum_{k}^{K} \frac{dE_k}{d\hat{y}_k} \frac{d\hat{y}_k}{da_k} \frac{da_k}{dh_m} \frac{dh_m}{da_m} \frac{da_m}{dw_{mp}}$$

Algorithm steps and Vanishing Gradients

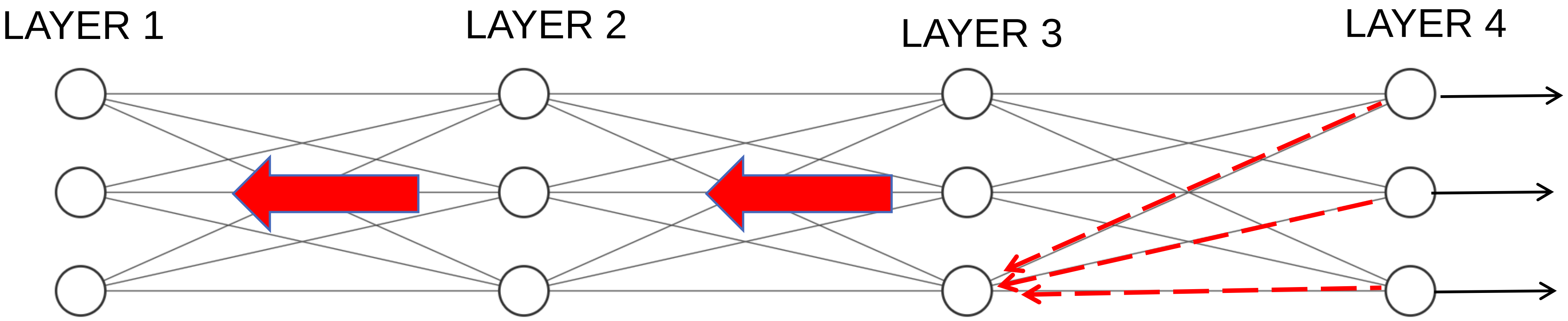


2. BACKWARD PROPAGATE ERROR USING DERIVATIVE CHAIN RULE:

$$\frac{dE}{dw_{mp}} = \sum_{k}^{K} \frac{dE_k}{d\hat{y_k}} \frac{d\hat{y_k}}{da_k} \frac{da_k}{dh_m} \frac{dh_m}{da_m} \frac{da_m}{dw_{mp}}$$

Note: As you go farther back, the error information gets diluted and the error gradient starts 'vanishing'

Algorithm steps and Vanishing Gradients



2. BACKWARD PROPAGATE ERROR USING DERIVATIVE CHAIN RULE:

$$\frac{dE}{dw_{mp}} = \sum_{k}^{K} \frac{dE_k}{d\hat{y_k}} \frac{d\hat{y_k}}{da_k} \frac{da_k}{dh_m} \frac{dh_m}{da_m} \frac{da_m}{dw_{mp}}$$

Note: As you go farther back, the error information gets diluted and the error gradient starts 'vanishing'

A different activation function helps …

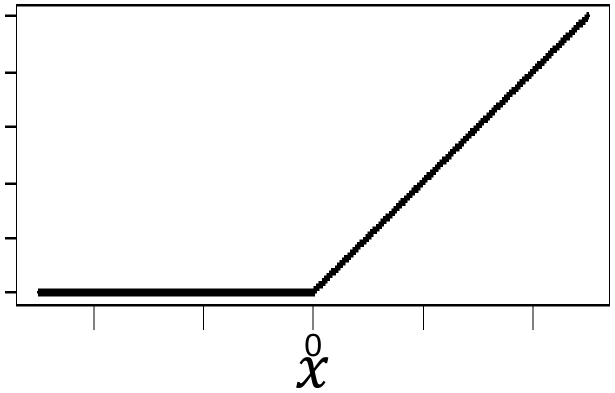
The rectified linear unit (RELU)

RELU activation function

It is unscaled (bad!)

But *df/da* is constant (good!)

RELU (rectified linear unit)



$$f(a) = \begin{cases} a & a > 0 \\ 0 & a <= 0 \end{cases}$$

$$\text{where } a = XW$$

Overall, RELU mitigates vanishing gradients

The Neural Network Algorithm

INITIALIZE weights to small value (for example: +/- <0.3)

LOOP until stopping criterion:

The Neural Network Algorithm

INITIALIZE weights to small value (for example: +/- <0.3)

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all error derivatives to ***minimize Loss***

The Neural Network Algorithm

INITIALIZE weights to small value (for example: +/- <0.3)

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all error derivatives to ***minimize Loss***

UPDATE WEIGHTS: ***w*** **←** ***w*** − ***learning_rate*** $* \ \frac{dL}{dw}$

The Neural Network Algorithm

INITIALIZE weights to small value (for example: +/- <0.3)

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all error derivatives to ***minimize Loss***

UPDATE WEIGHTS: ***w*** ← ***w*** − ***learning_rate*** $* \frac{dL}{dw}$

STOP: when validation error reaches minimum or after a max number of epochs

The Neural Network Algorithm [in practice]

INITIALIZE weights to small value (for example: +/- <0.3)

LOOP until stopping criterion:

[work in batches of input]

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all error derivatives to ***minimize Loss***

UPDATE WEIGHTS: $w \leftarrow w - \; learning\_rate * \frac{dL}{dw}$

[adapt learning rate, use momentum]

STOP: when validation error reaches minimum or after a max number of epochs

[several metrics of loss are possible]

Neural Network main options to choose:

- 1 Architecture: number of hidden units & layers
- 2 Optimizer and learning rate for weight updates
- 3 Loss function depends on task
- Note: more hidden layers, more hidden units => more potential for overfitting

terminology and cheat sheet on output activations (for reference):

Type of Problem	Y outputs	Output Activation Function (this gives a SCORE= $\hat{Y}$:)	Output PREDICTION (what you decide to predict)	Output Loss Function	Evaluative Measure
Regression: map into to K real valued predictions	$\text{if } Y \in (-\infty, +\infty)^K$	$\hat{Y} = XW$	$\hat{Y}$:	Sum Squared Error (SSE)	Mean Squared Error (MSE)
Multivariate output of 0's and 1's	$\text{if } Y \in [0, 1]^K$	$\hat{Y} = \frac{1}{1 + exp^{-(XW)}}$	1 or 0	SSE	MSE
Binary Classification	$\text{if } Y \in \{0, 1\}$	$\hat{Y} = \frac{1}{1 + exp^{-(XW)}}$	A probability given by $\hat{Y}$: $P(y = 1 \mid x)$	Cross Entropy $L = -ylog(\hat{y}) - (1 - y)(log(\hat{y}))$	Accuracy, ROC
Multiclassification	$\text{if } Y \in \{0, 1\}^K$	$\hat{Y}_k = \frac{exp^{-(XW_k)}}{\sum_k exp^{-(XW_k)}}$	Max class	Cross Entropy $L = -\sum_k y_k log(\hat{y_k})$	Accuracy

Summary:

Pro:

- Multilayer Perceptron, and Neural Networks in general, are flexible powerful learners
- Hidden layers learn a nonlinear transformation of input

Summary:

Pro:

- Multilayer Perceptron, and Neural Networks in general, are flexible powerful learners
- Hidden layers learn a nonlinear transformation of input

Con:

- Lots of parameters
- Hard to interpret
- Needs more data

A neural network can discover visual features using ‘convolutions’

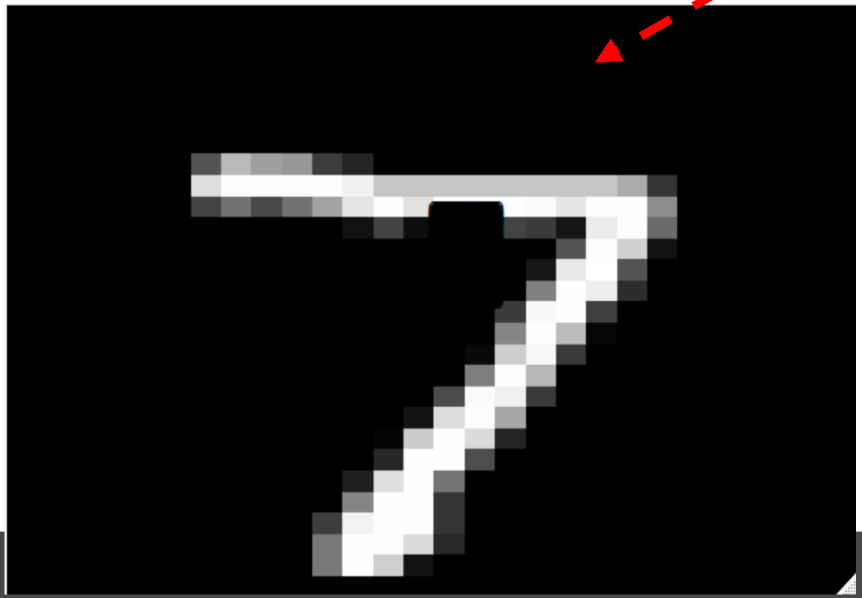
Next: Image classification of digits

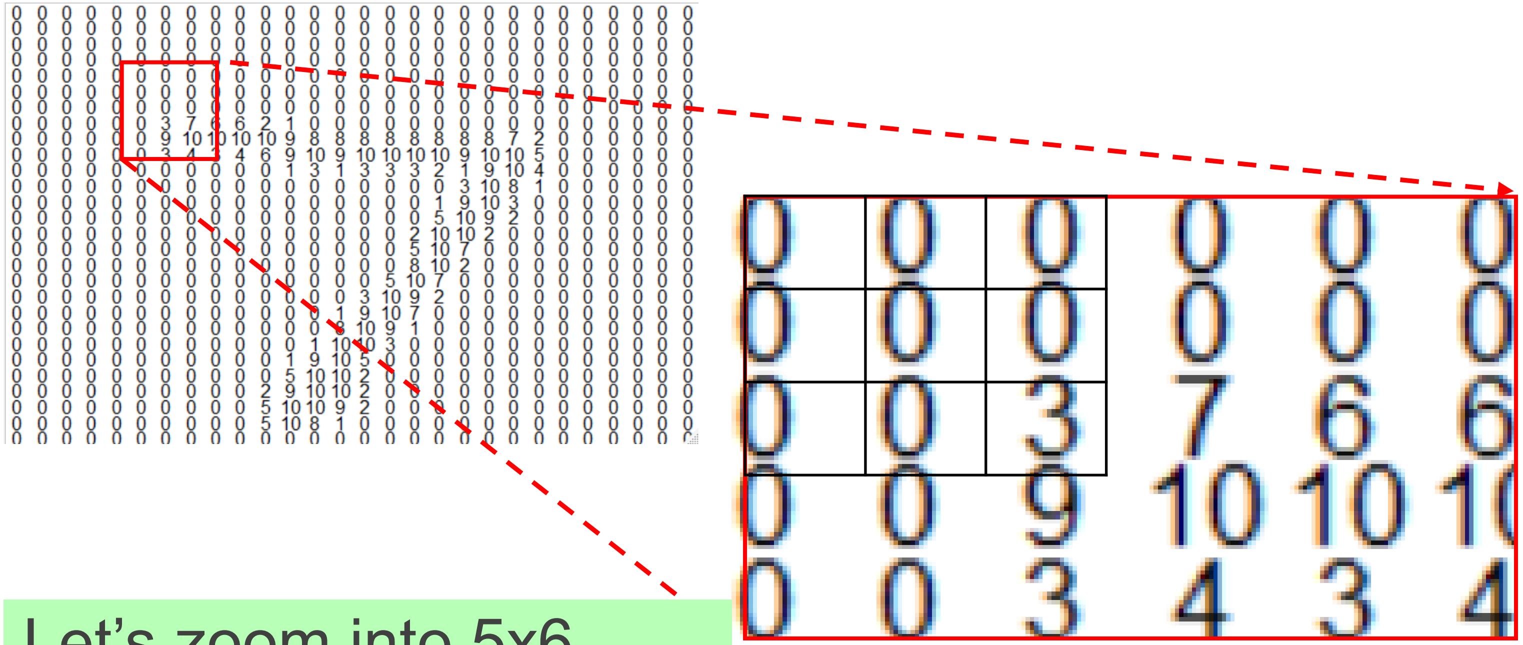
Image features

- **MNIST - A database of handwritten printed digits**

  (National Inst. of Standards and Technology)

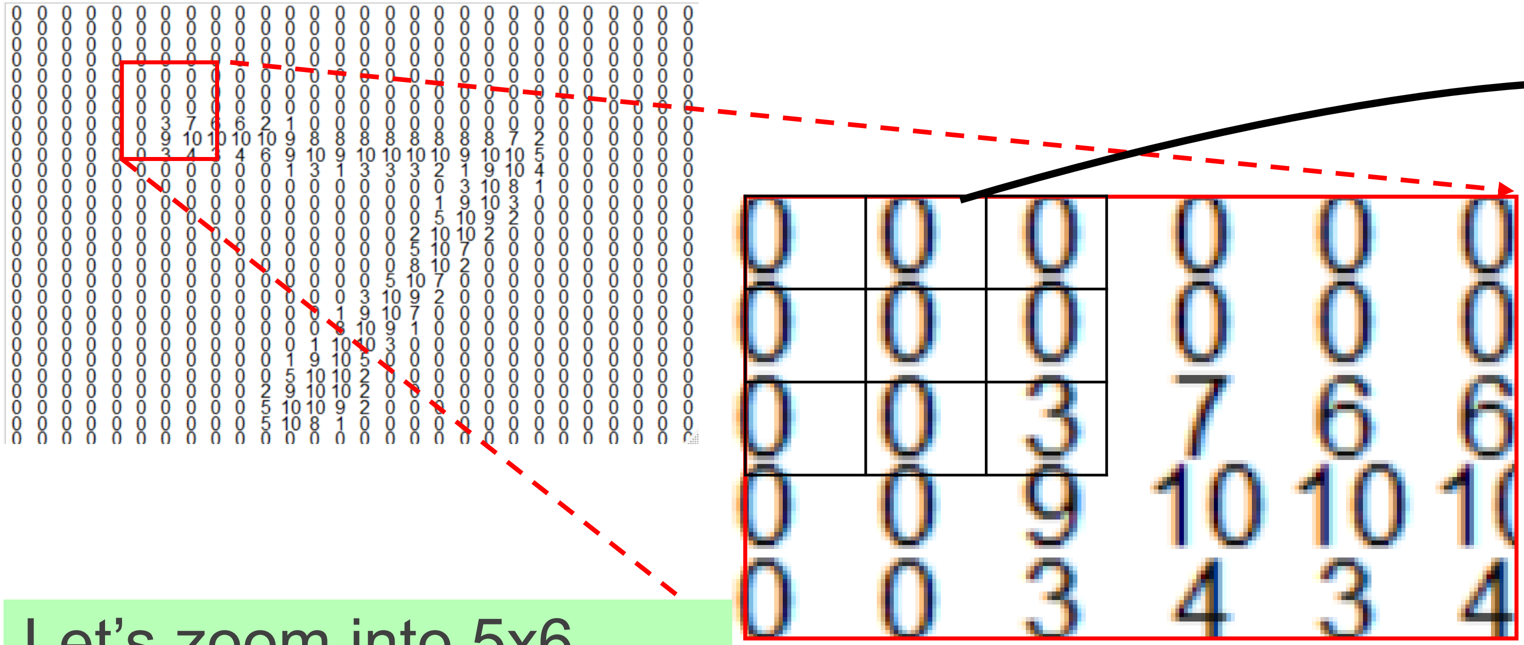
How to classify digits?





Let’s zoom into 5x6 window of pixels near the tip of ‘7’

Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge



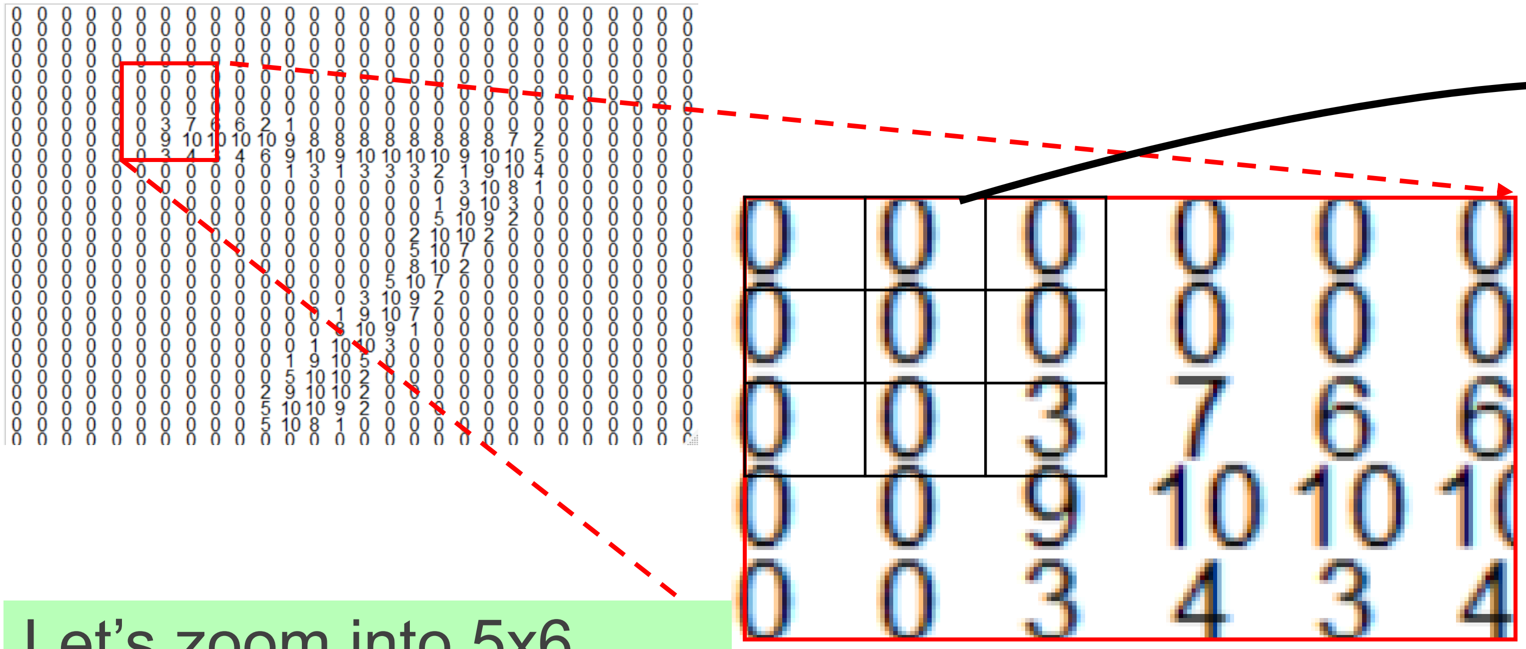
Let’s zoom into 5x6 window of pixels near the tip of ‘7’

Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge

X

-1	0	+1
-1	0	+1
-1	0	+1

1. Multiply 3x3 patch of pixels with 3x3 filter



Let’s zoom into 5x6 window of pixels near the tip of ‘7’

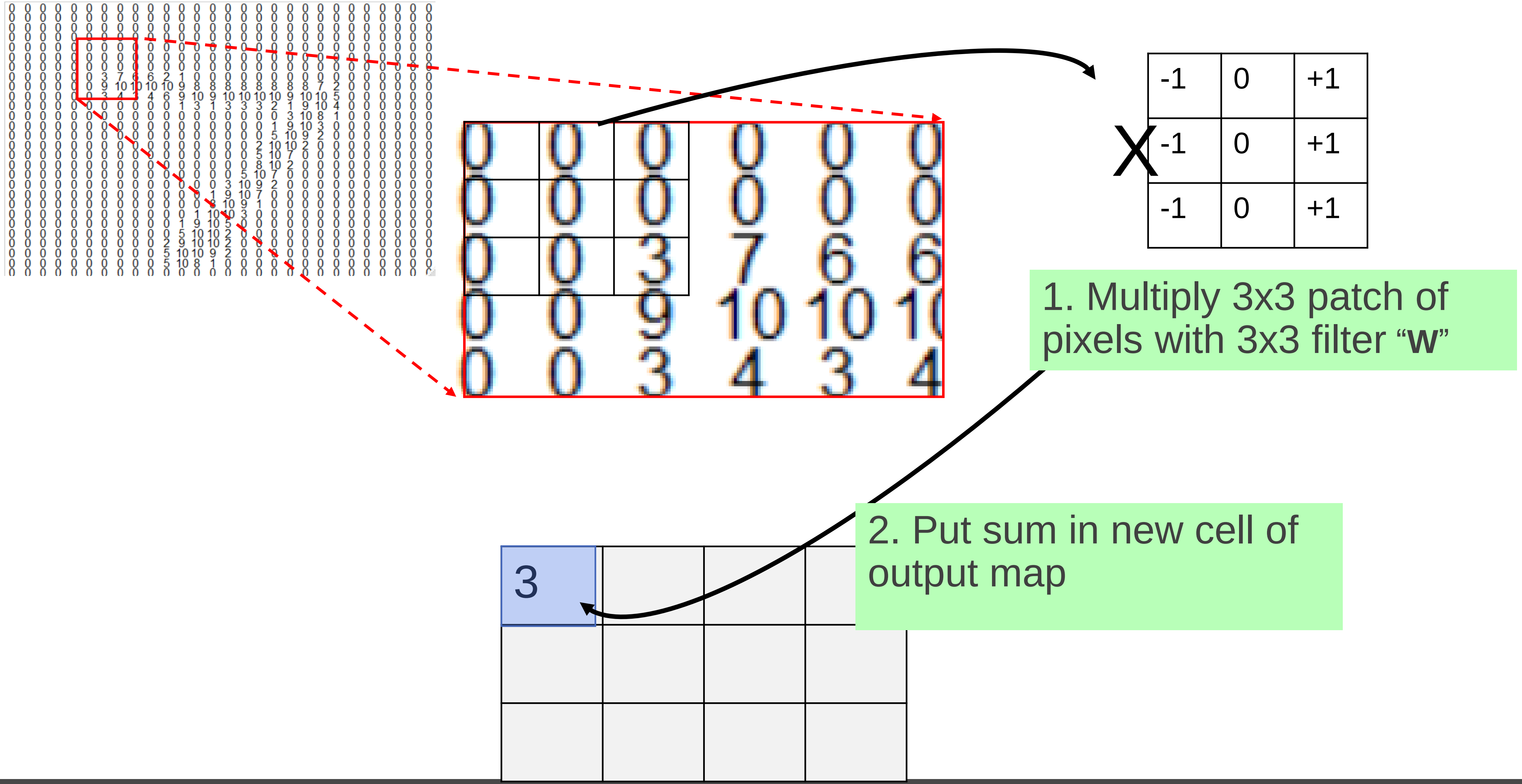
Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge

(our weight parameters)

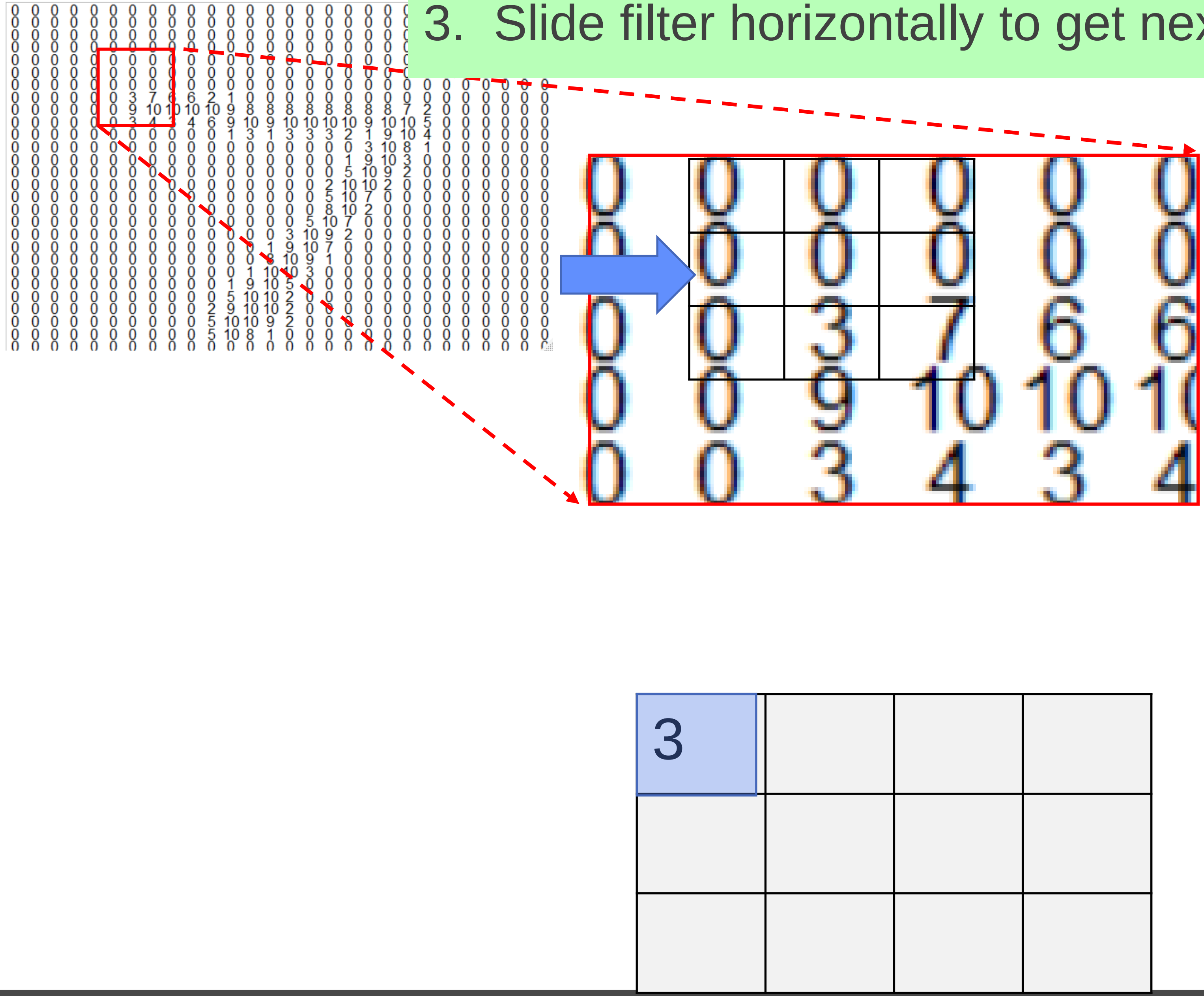
X

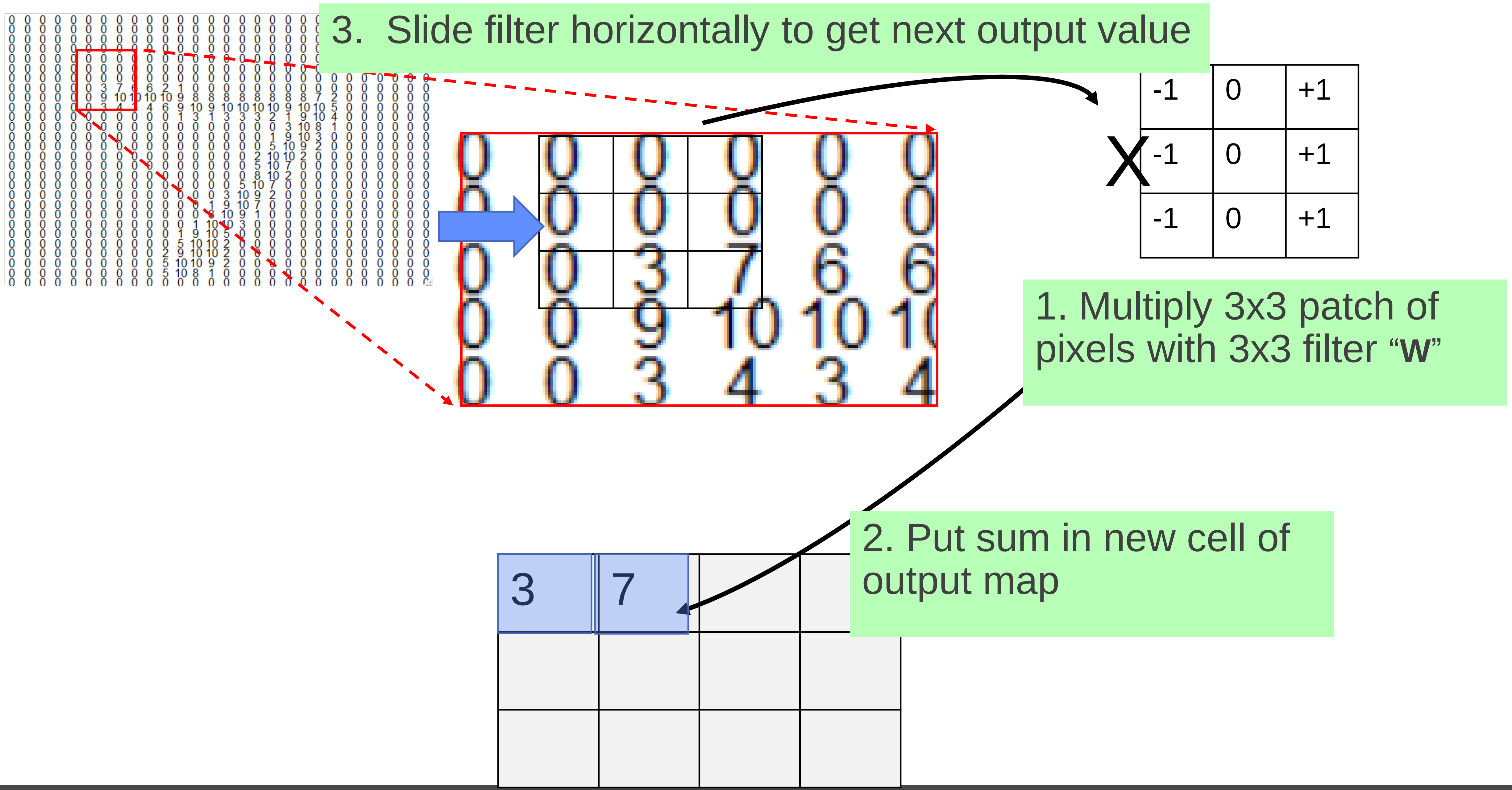
-1	0	+1
-1	0	+1
-1	0	+1

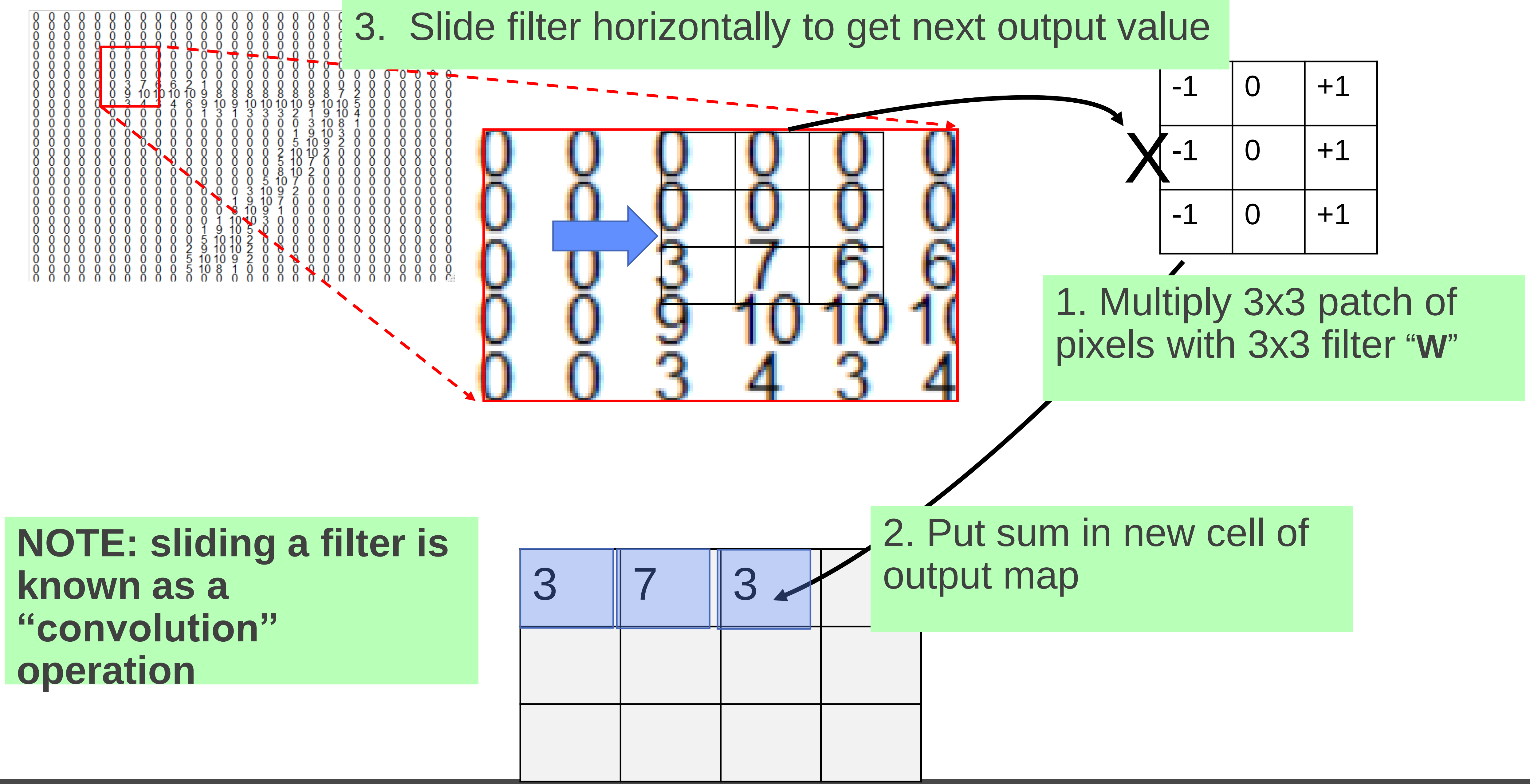
1. Multiply 3x3 patch of pixels with 3x3 filter “**W**”


-1 0 +1
X
-1 0 +1
-1 0 +1
1. Multiply 3x3 patch of pixels with 3x3 filter “**w**”
2. Put sum in new cell of output map
3

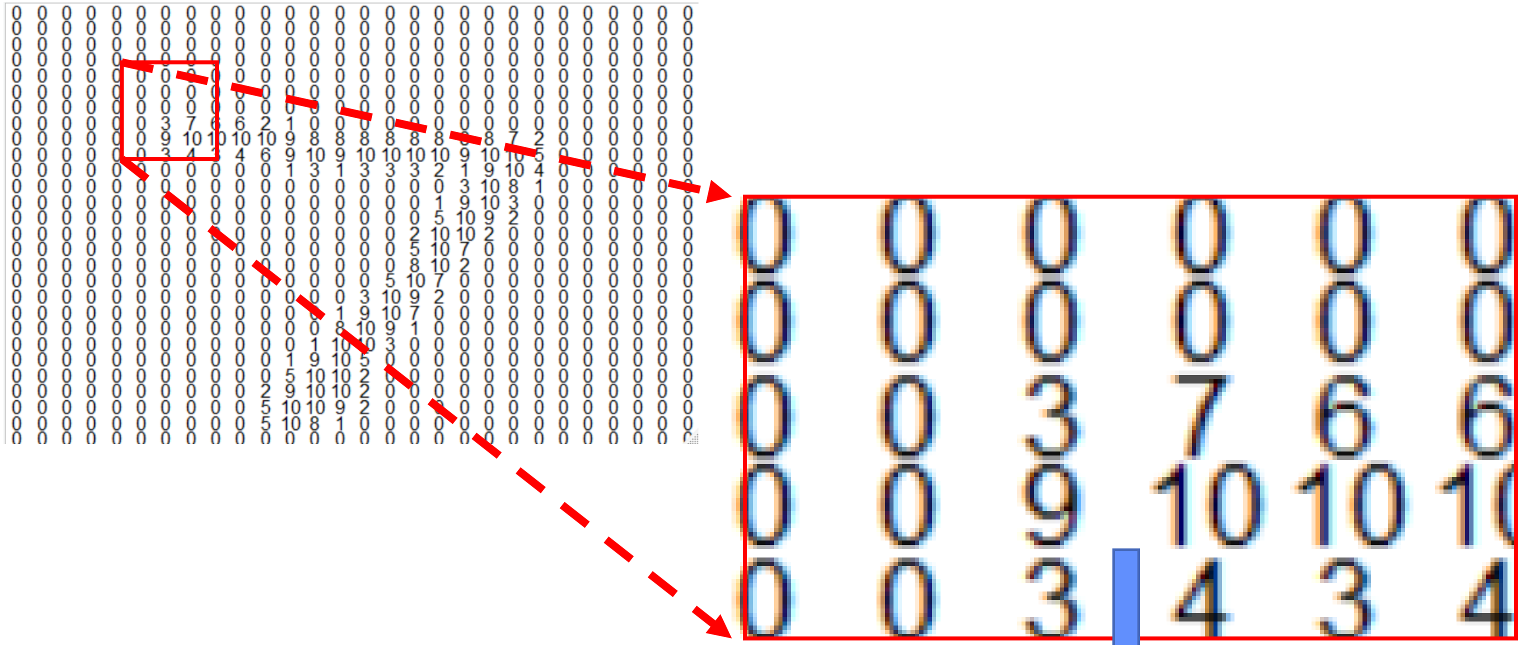
3. Slide filter horizontally to get next output value



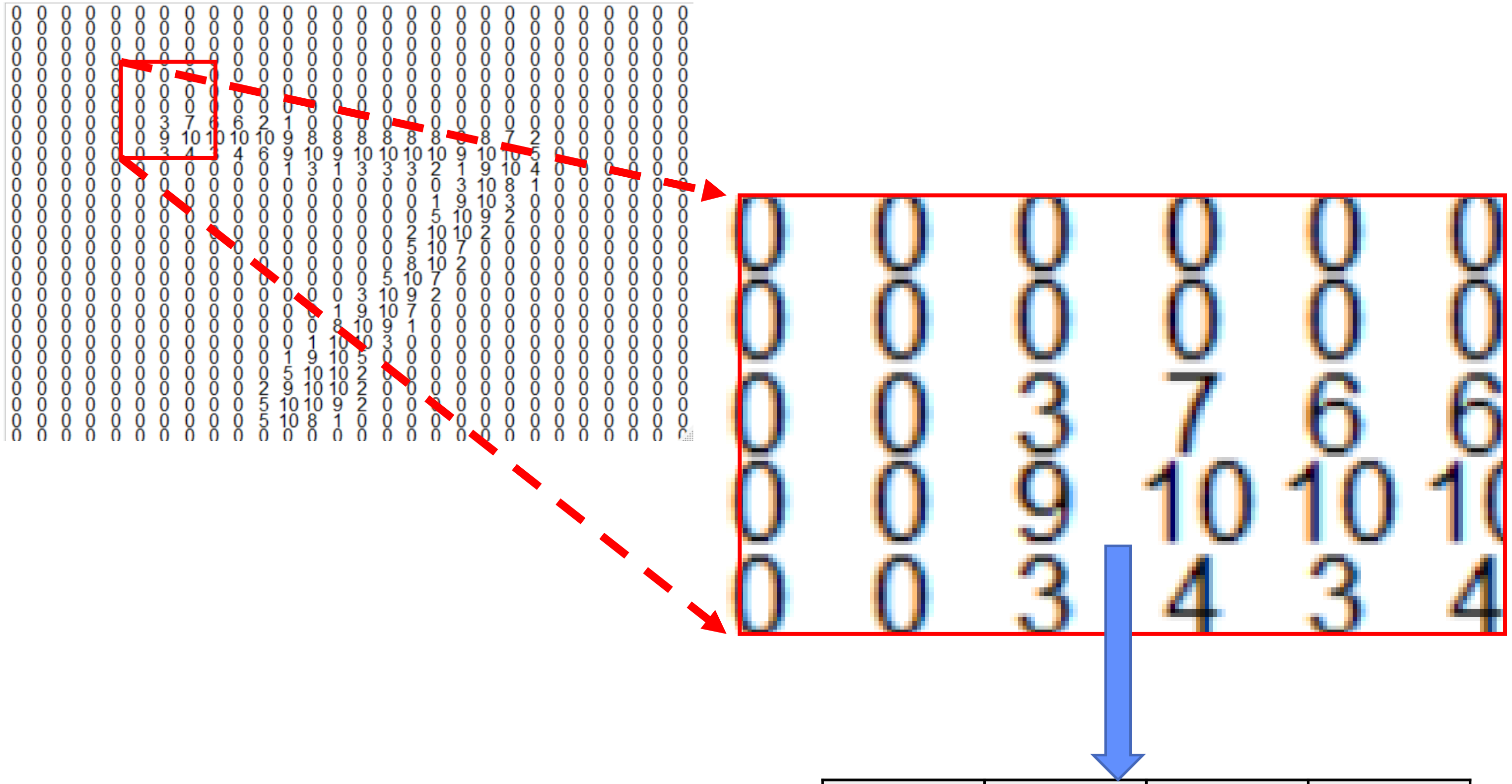

3. Slide filter horizontally to get next output value
-1 0 +1
-1 0 +1
-1 0 +1
X
1. Multiply 3x3 patch of pixels with 3x3 filter "w"
2. Put sum in new cell of output map
3
7


3. Slide filter horizontally to get next output value
1. Multiply 3x3 patch of pixels with 3x3 filter “w”
2. Put sum in new cell of output map
NOTE: sliding a filter is known as a “convolution” operation
-1 0 +1
-1 0 +1
-1 0 +1
X
3 7 3

*After vertical and horizontal sliding the 5x6 patch is now a 3x5 **feature map.***



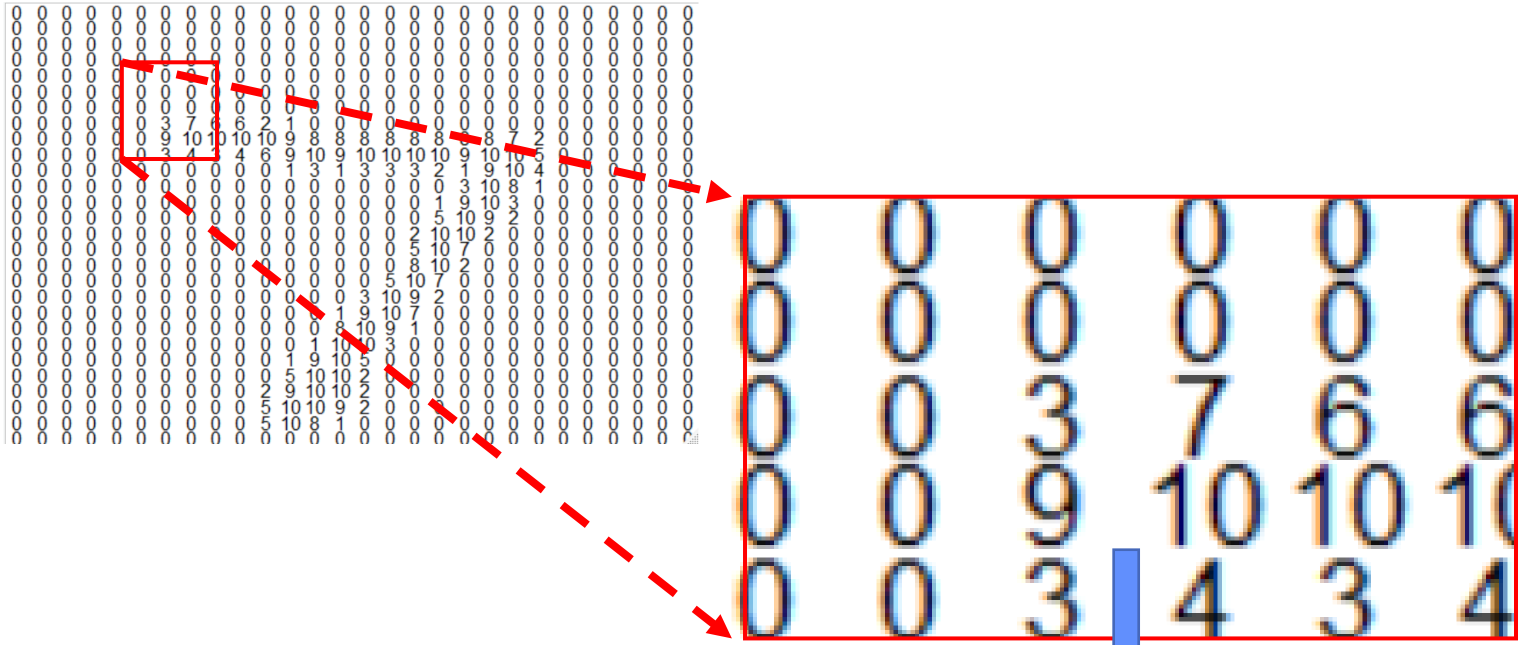
3	7	3	-1
12	17	4	-1
15	21	4	-1



3	7	3	-1
12	17	4	-1
15	21	4	-1

After vertical and horizontal sliding the 5x6 patch is now a 3x5 ***feature map.***

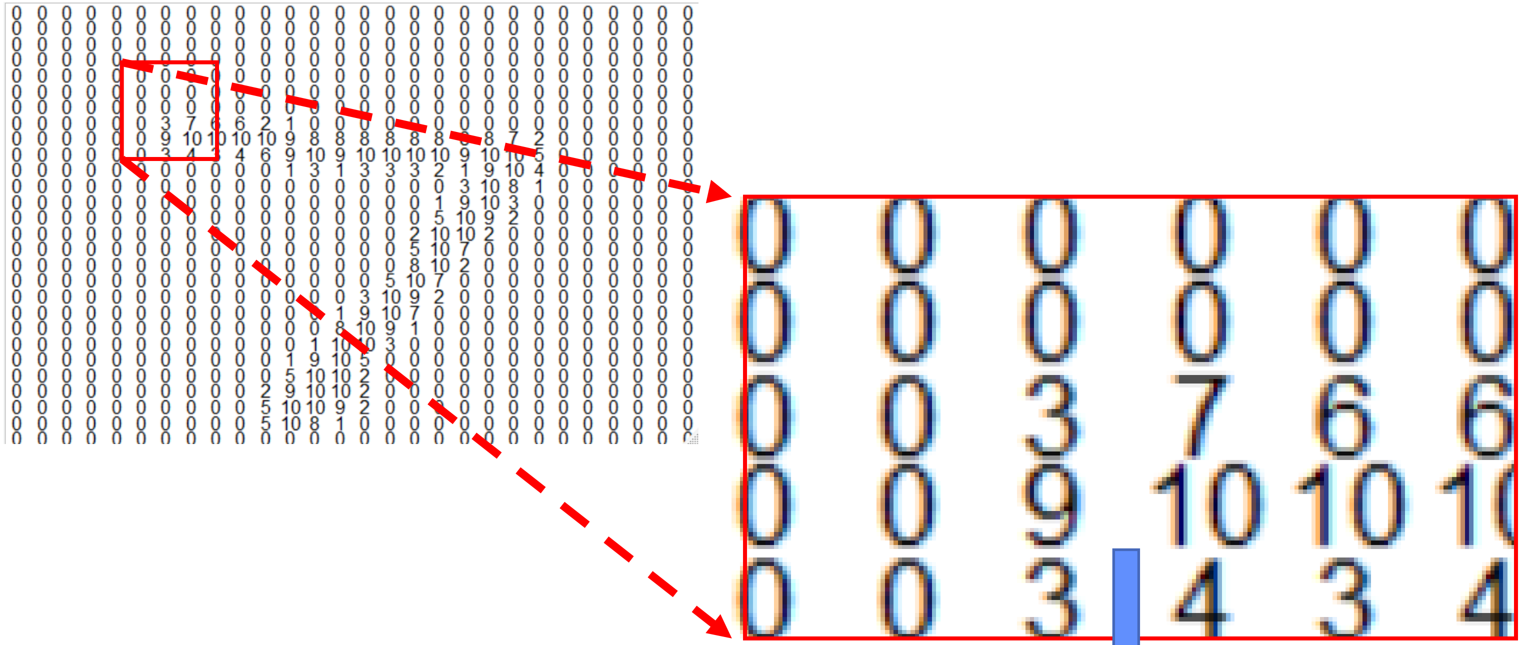
What do the highest values in the feature map represent?



3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

Use another filter, and take maximum over elements - “max pooling”



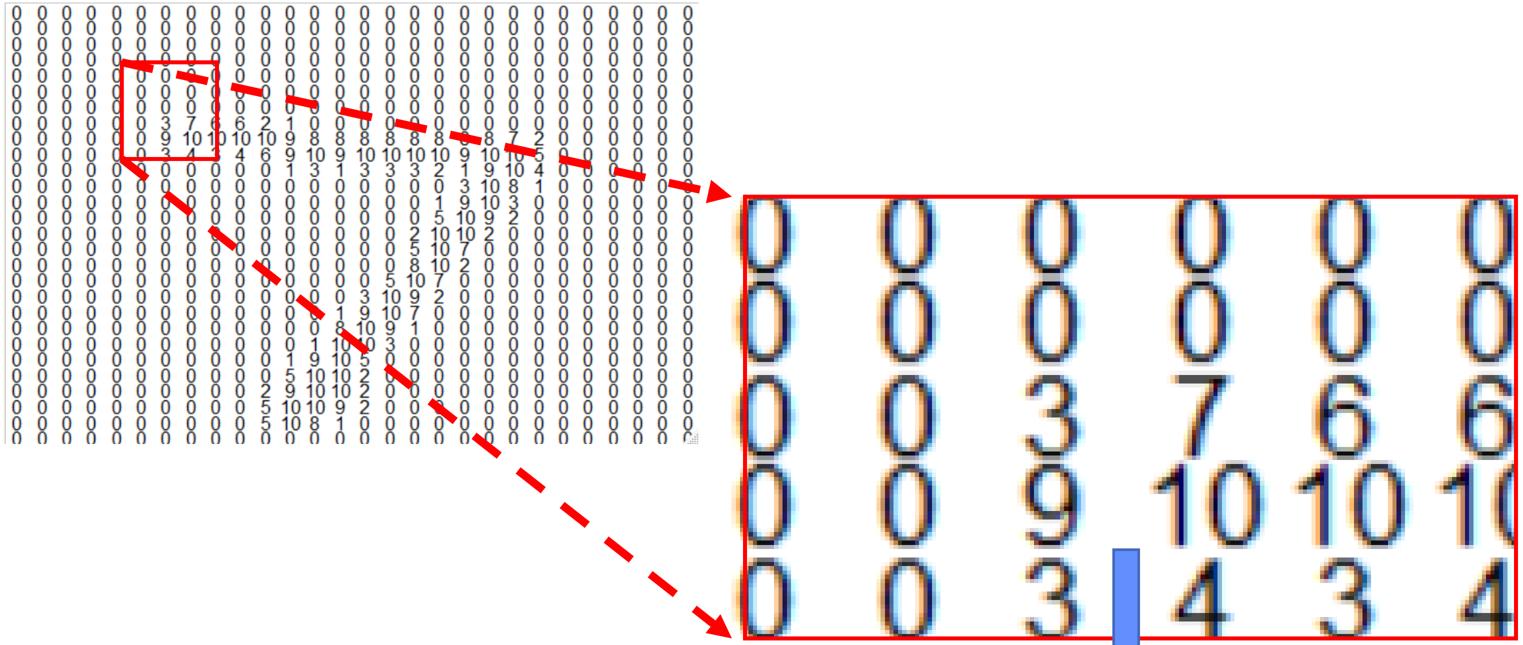
3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

Use another filter, and take maximum over elements - “max pooling”

2x2 filter has max=17

17		



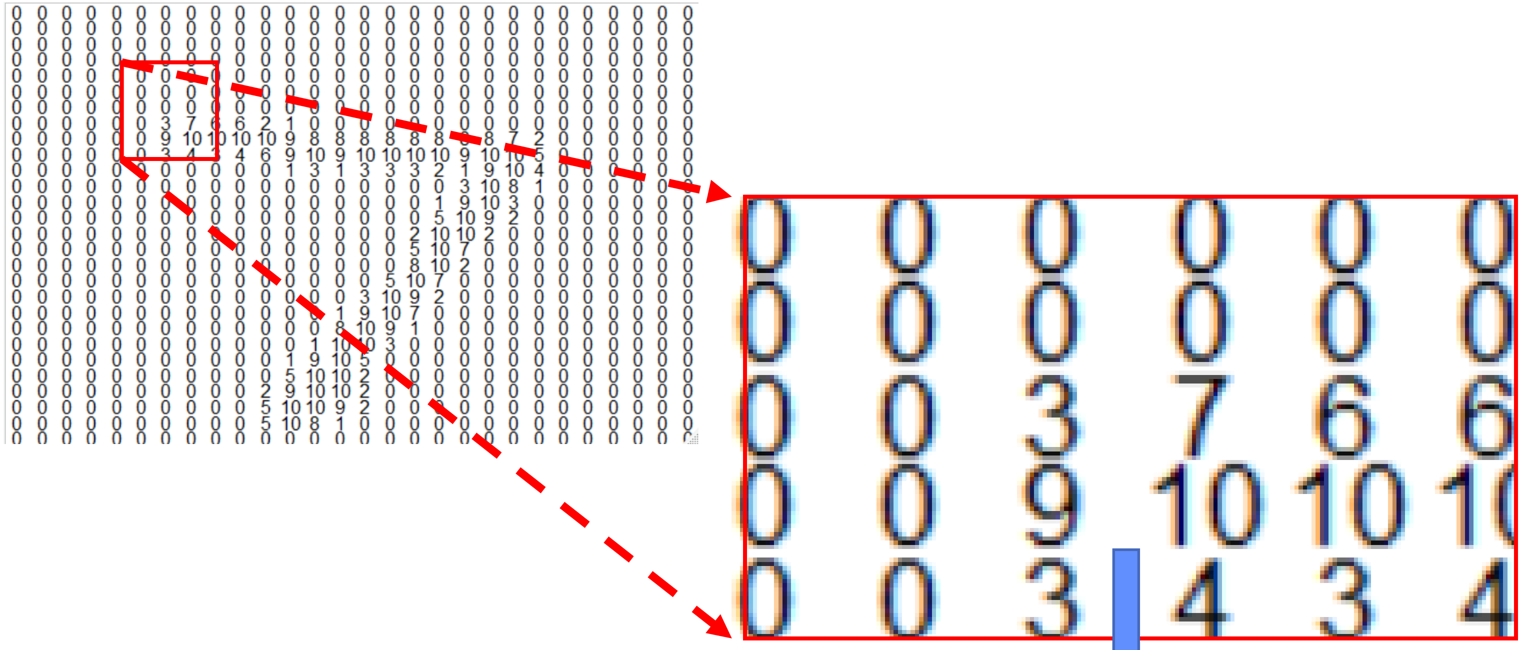
3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

Use another filter, and take maximum over elements - “max pooling”

Slide filter …

17	17	4
21	21	4

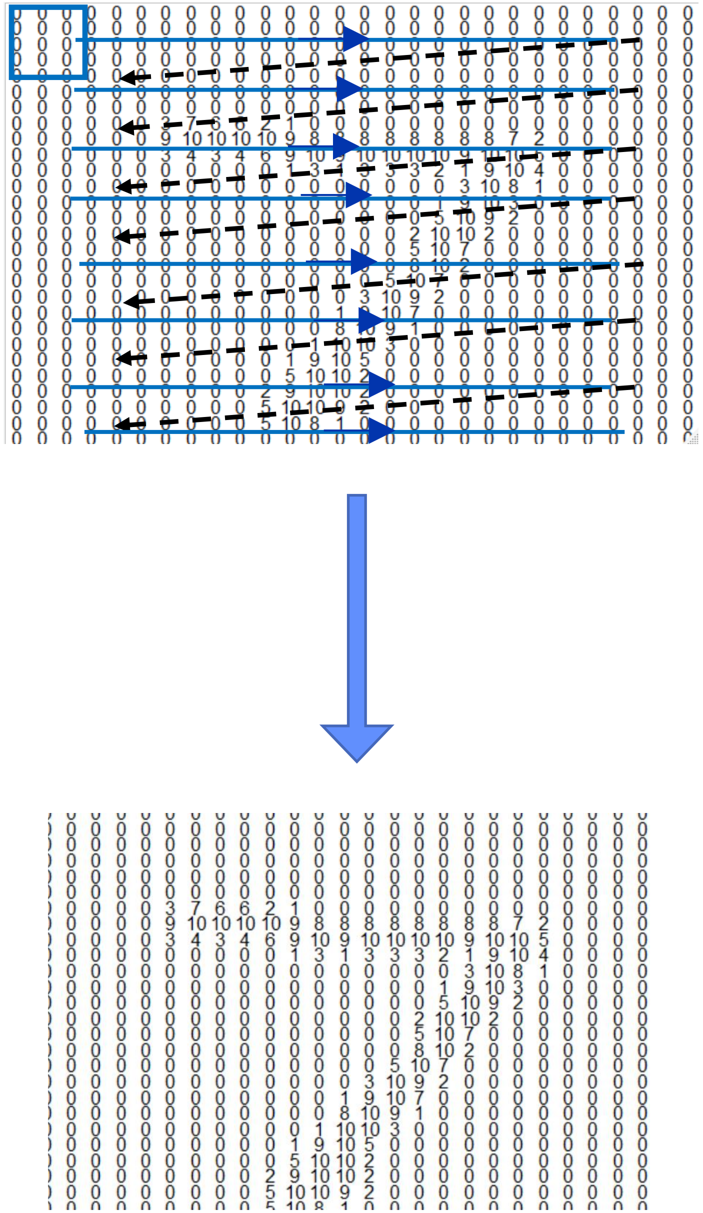


3	7	3	-1
12	17	4	-1
15	21	4	-1

After convolution and pooling,the 5x6 patch is ***transformed*** *into a 2x3 feature map of 'edge gradients'*

Slide filter …

17	17	4
21	21	4



A convolution of one filter is applied to the entire image across and down.

*The entire 28x28 input is **transformed** into a smaller feature map of ‘edge gradients’*

Pooling is optionally applied

Convolution Neural Network (CNN)

In CNNs the filter values are weight parameters that are learned (**feature discovery**)

W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}

Convolution Neural Network (CNN)

In CNNs the filter values are weight parameters that are learned (**feature discovery**)

W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}

A convolution layer is a set of feature maps, where each map is derived from convolution of 1 filter with input

Convolution Neural Network (CNN)

More hyperparameters:

Size of filter (smaller is more general)

Convolution Neural Network (CNN)

More hyperparameters:

Size of filter (smaller, like 3x3, is more general)

Number of pixels to slide over (1 or 2 is usually fine)

Convolution Neural Network (CNN)

More hyperparameters:

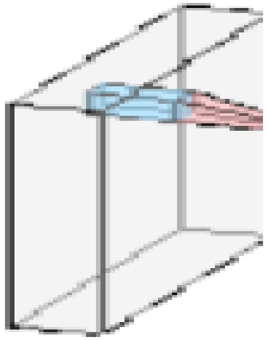
- Size of filter (smaller, like 3x3, is more general)
- Number of pixels to slide over (1 or 2 is usually fine)
- Max pooling or not (usually some pooling layers)
- Number of filters (depends on the problem!)

- **A large CNN example**

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for Red,Green,Blue channels)**

Crop or
resize
image as
needed



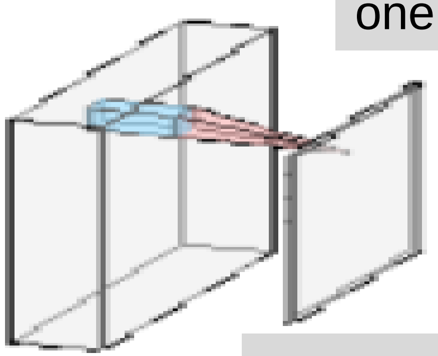
224x224x
3

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**

Crop or resize image as needed

one 3x3x3 filter

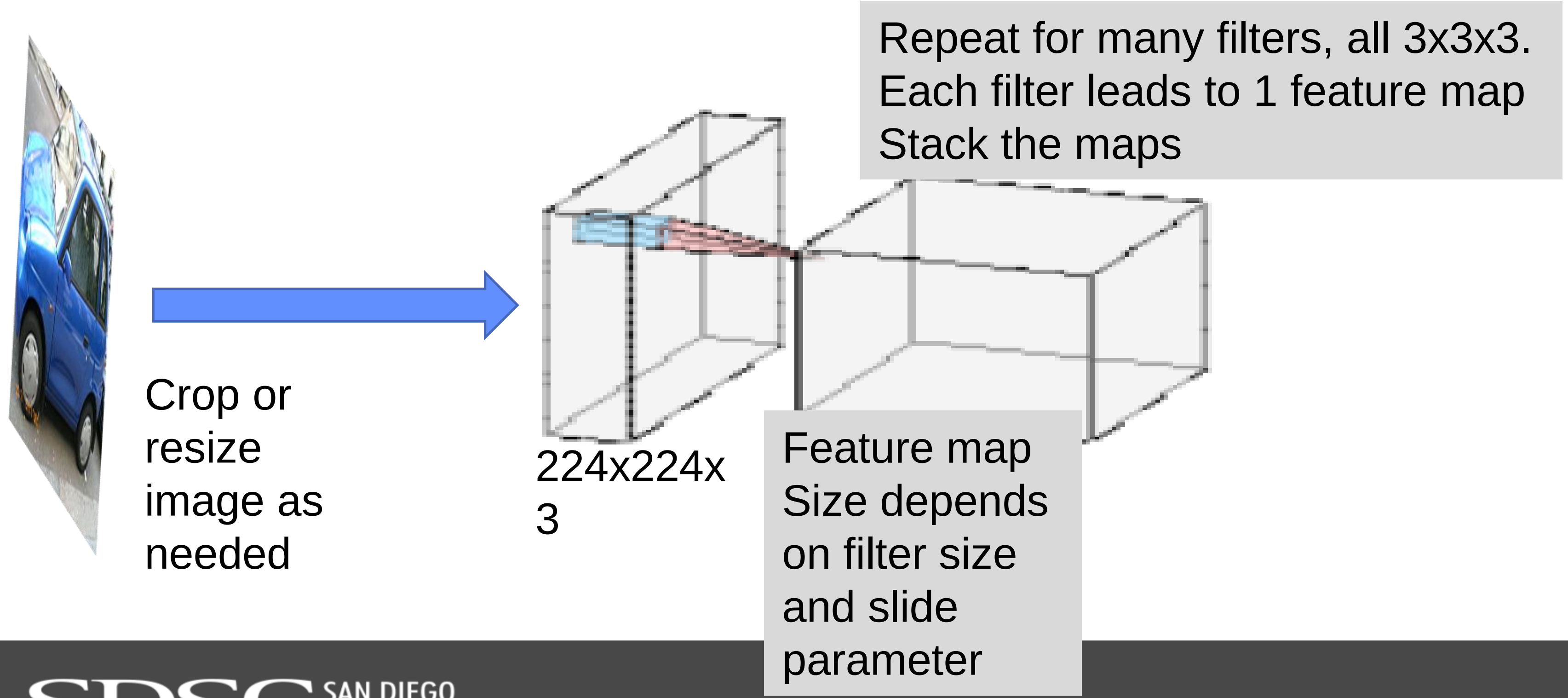


224x224x3

Feature map Size depends on filter size and slide parameter

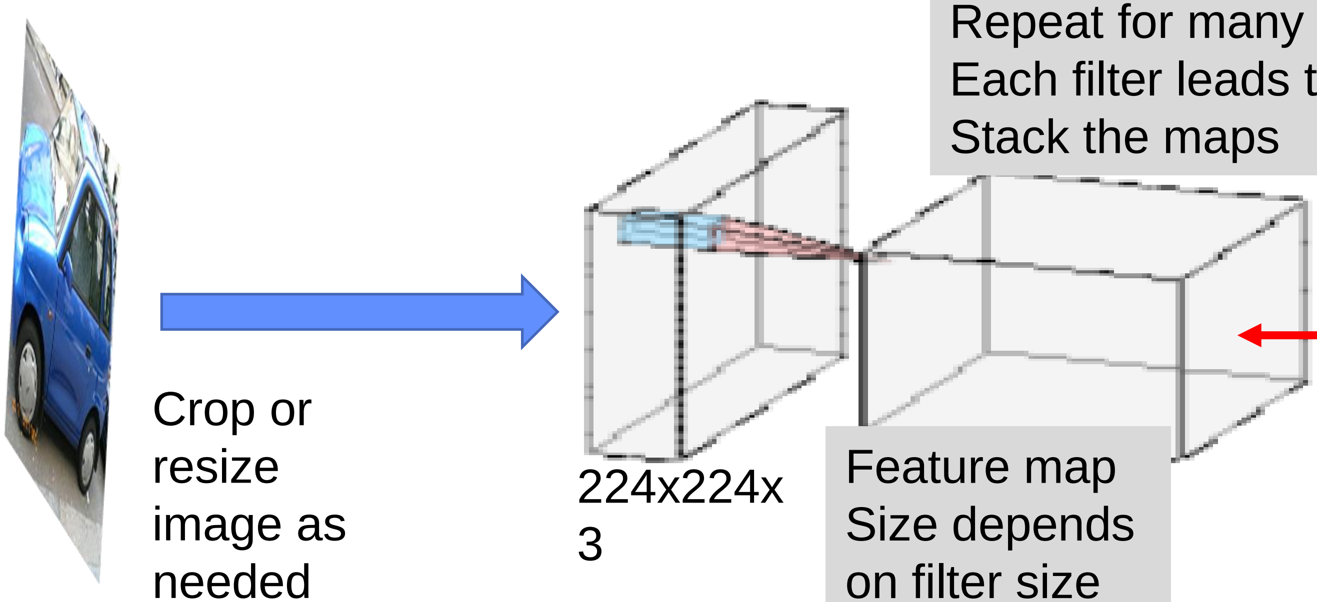
Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**



Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**



Crop or resize image as needed

224x224x3

Repeat for many filters, all 3x3x3.
Each filter leads to 1 feature map
Stack the maps

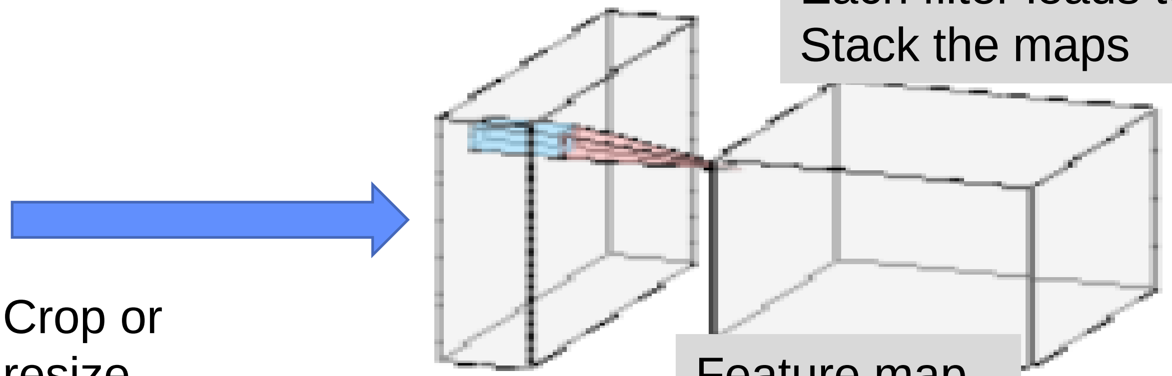
Feature map
Size depends on filter size and slide parameter

This is a "convolution layer"
The thickness is the number of different feature maps, sometimes called 'channels' or 'number of filters' or 'depth'

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**

Crop or resize image as needed



224x224x3

Repeat for many filters, all 3x3x3.
Each filter leads to 1 feature map
Stack the maps

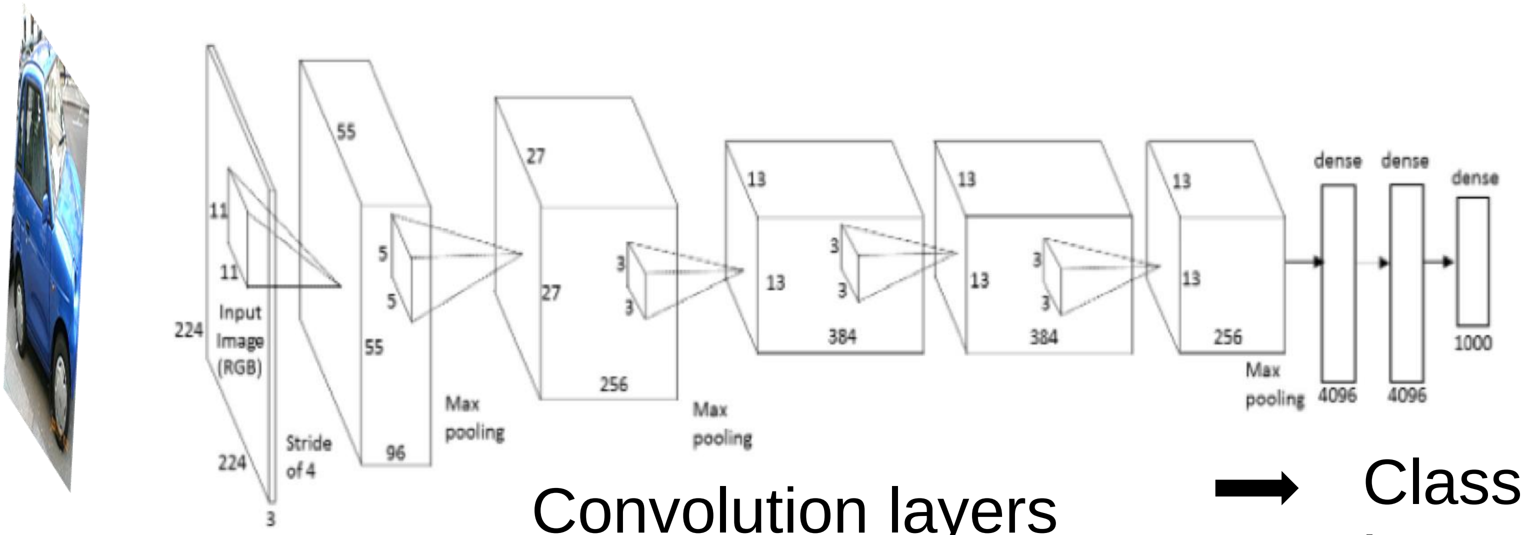
Feature map
Size depends on filter size and slide parameter

Add a next convolution layer
The filter size is HxWx num-of-filters-previous-layer

Also, add a layer of max-pool to down sample a feature map

Large Scale Versions

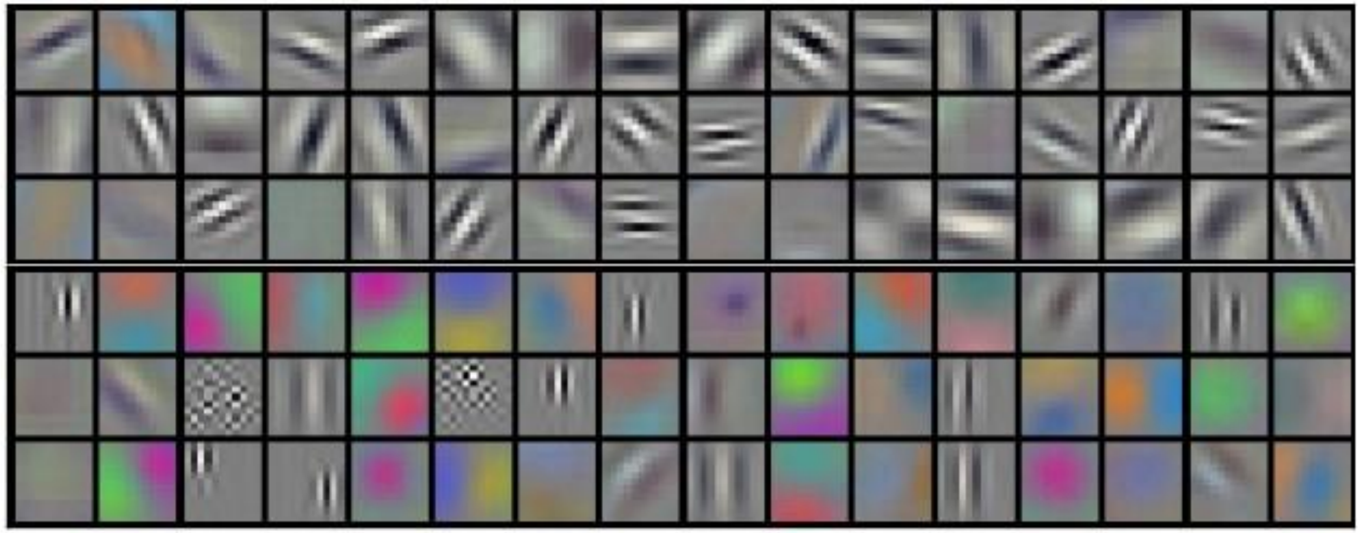
- **Large Convolution Networks – Alexnet, VGG19, ResNet, GoogLeNet, …**



Convolution layers ➡ Classification layers and output

What Learned Convolutions Look Like

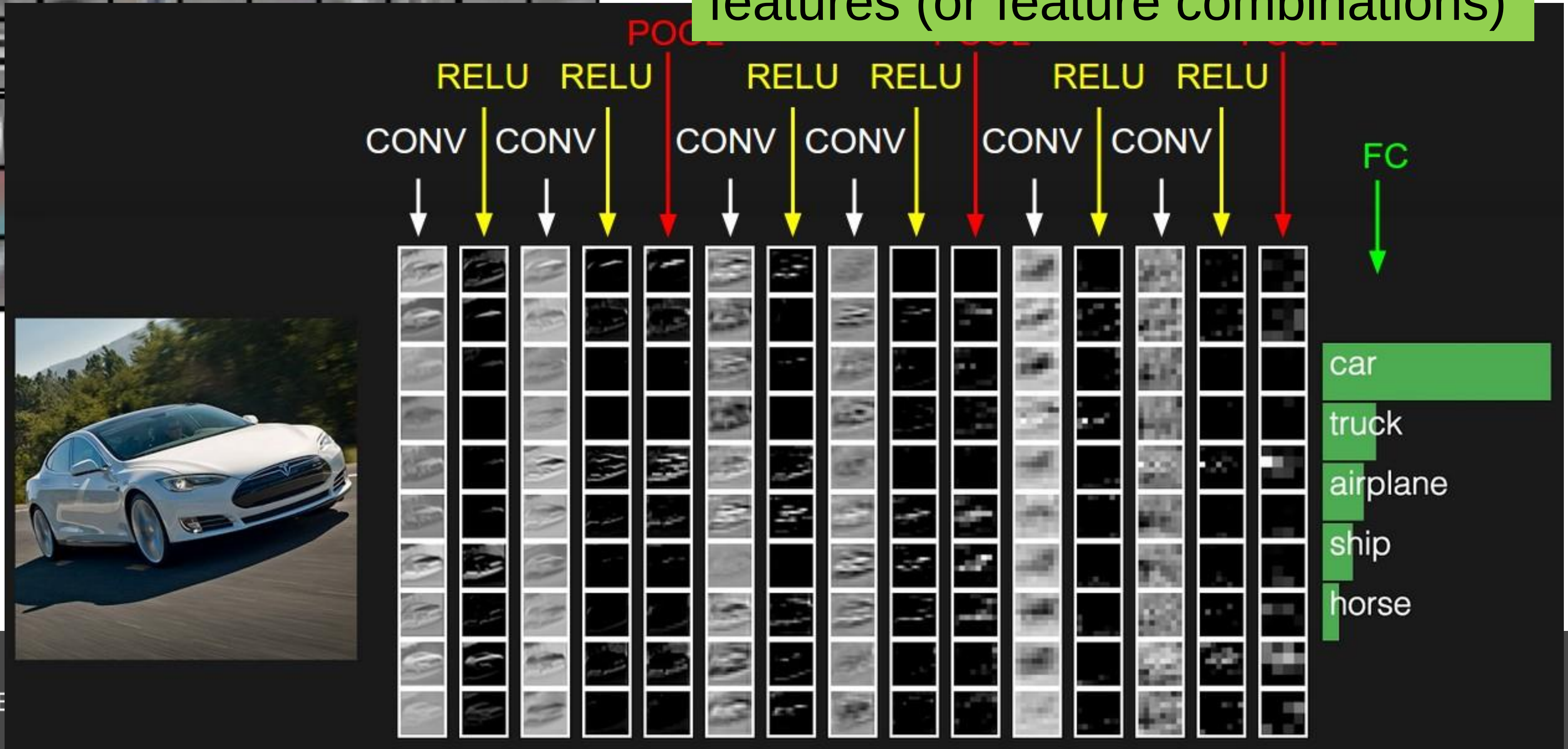
First convolution layer filters are simple features



What Learned Convolutions

First convolution layer filters are simple features

Higher layers are more abstract features (or feature combinations)



Convolution Neural Network Summary

CNNs works because convolution layers have a special architecture and function – it is biased to do certain kind of transformations

Low layers have less filters that represent simple local features for all classes

Higher layers have more filters that cover large regions that represent object class features

Deep Learning in general:

Deep learning refers to learning complex and varied transformations of the input

Deep learning refers to **discovering** useful features of the input

Deep learning is a neural network with many layers

- **Next, notebook demo**

Exercise CNN for Digit Classification

- The ‘hello world’ of CNNs
- It uses MNIST dataset and Keras/Tensorflow
- Goal: Get familiar with Keras and CNN layers coding, and CNN solutions
- We will login and start a notebook

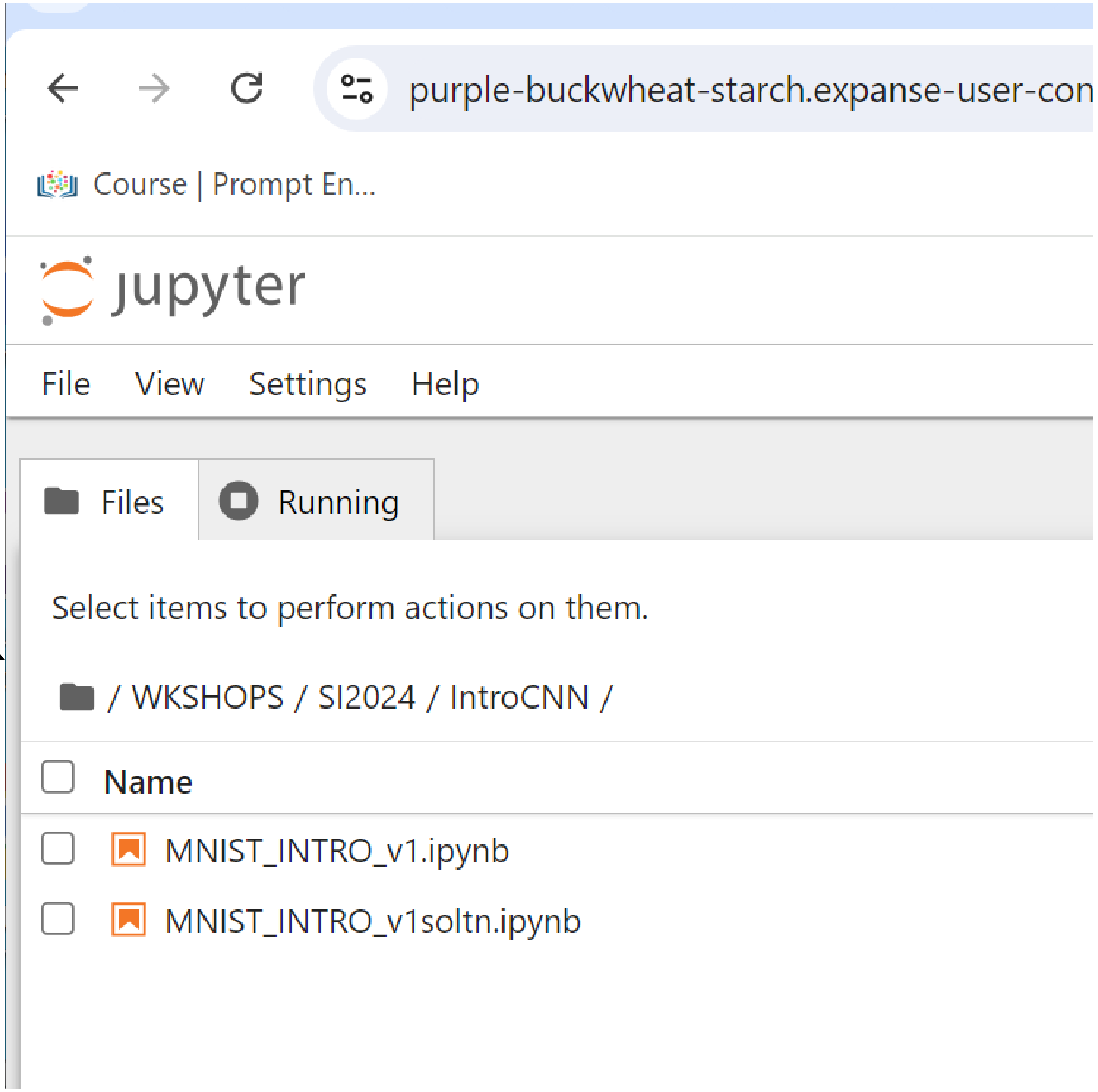
(see next pages for quick overview)

login to Expanse

$ jupyter-compute-tensorflow

In jupyter notebook session open the MNIST_Intro notebook

Follow instructions in the notebook



Keras code for a convolution neural nework

```
# --------------Set up Model ---------------------
def build_model(numfilters):
    mymodel = keras.models.Sequential()
    mymodel.add(keras.layers.Convolution2D(numfilters,      #<<<< ------ 1
                                   (3, 3),
                                   strides=1,
                                   data_format="channels_last",
                                   activation='relu',
                                   input_shape=(28,28,1)))
    #add another conv layer?   mymodel.add(keras.layers.Convolution2D( ...

    mymodel.add(keras.layers.MaxPooling2D(pool_size=(2,2),strides=2,data_format="channels_la
    mymodel.add(keras.layers.Flatten())            #reorganize 2DxFilters output into 1D

    #----------------Now add final classification layers
    mymodel.add(keras.layers.Dense(32, activation='relu'))
    mymodel.add(keras.layers.Dense(10, activation='softmax'))

    # --------- Now configure model algorithm -----
    mymodel.compile(loss='categorical_crossentropy',
             optimizer=keras.optimizers.Adam(learning_rate=0.001),
```

A sequential model

Add convolution layer

Add max pooling, then flatten into a vector for classification layers

- Remember every layer has some input, ouput
- Keras calculates the matrix shapes
- Not every layer in Keras has trainable parameters – like which one of these?

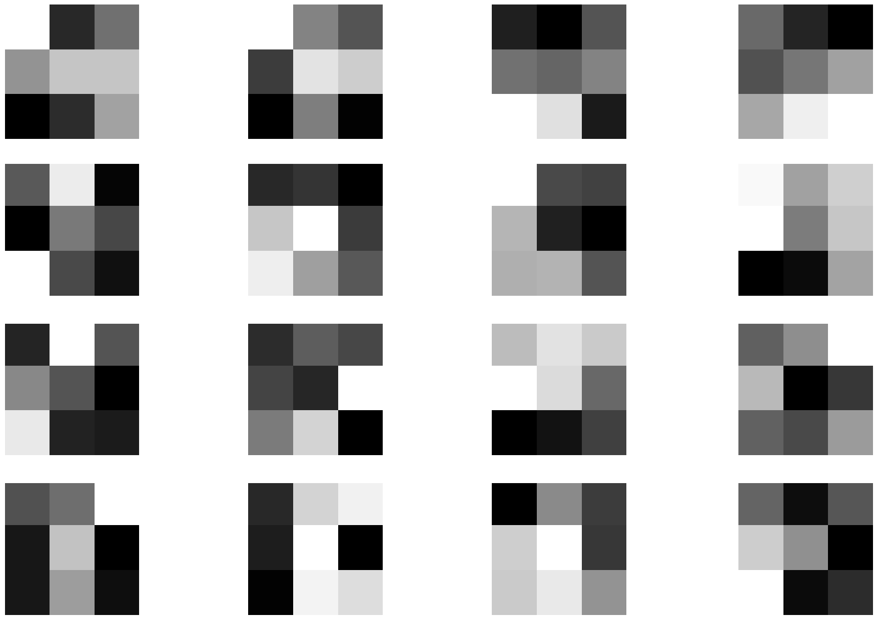
Zooming in on keras convolution layers statements

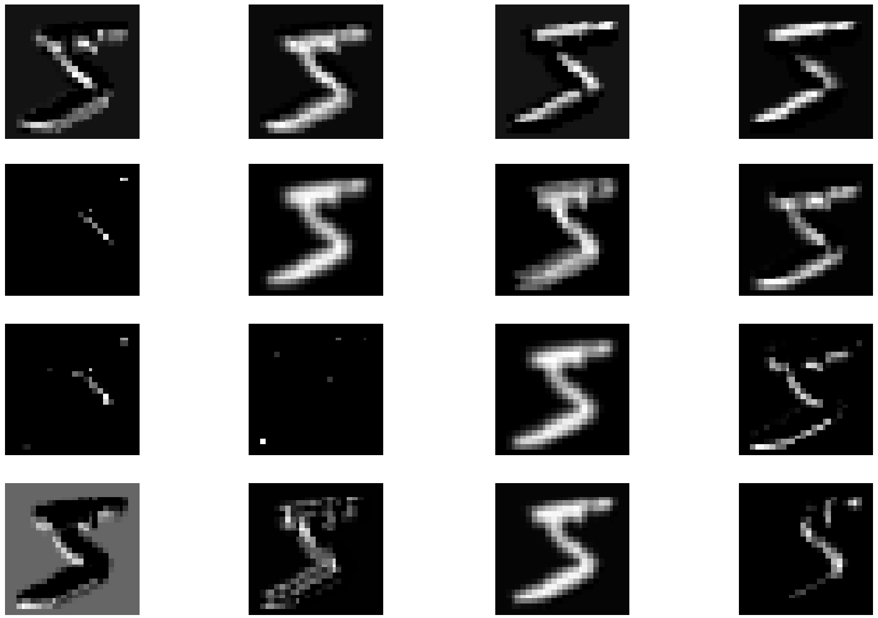
Use 16 filters, each of size 3x3

```
my_model.add(tf.keras.layers.Convolution2D(filters=16,
                                           kernel_size=(3, 3),
                                           strides=1,
                                           data_format="channels_last",
                                           activation='relu',
                                           input_shape=(28,28,1)))
```

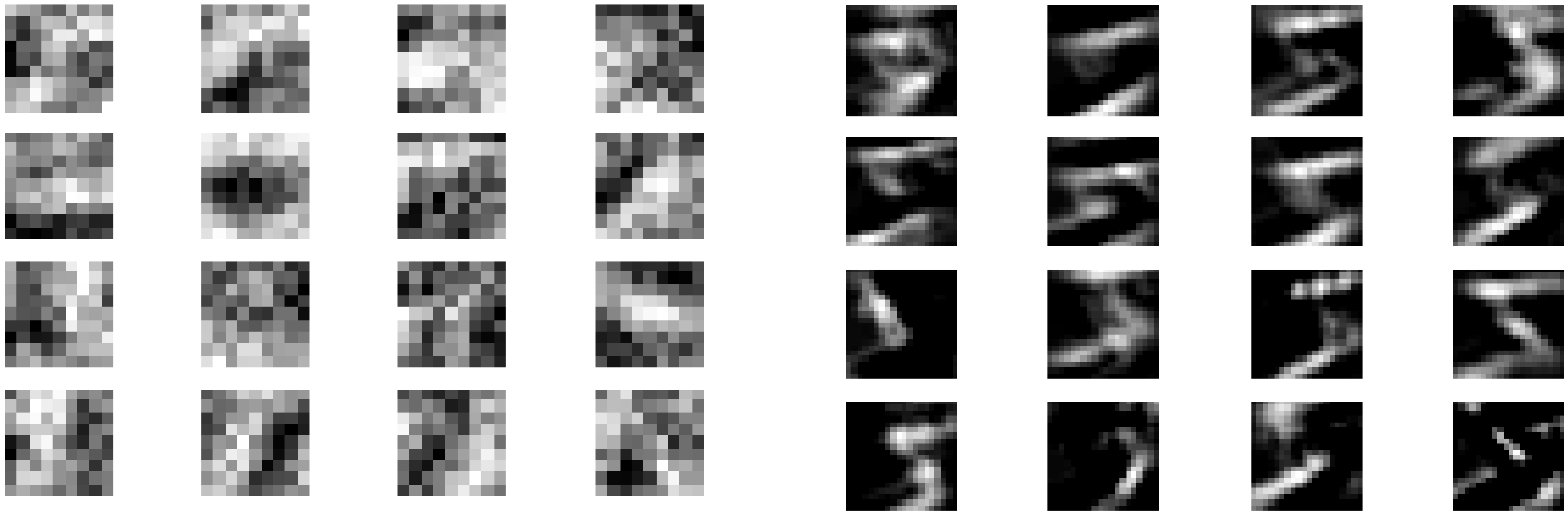
Input shape does not include number of images

Exercise notes: 3x3 first convolution layer filter and activation





9x9 first convolution layer filter and activation



End