SDSC HPC CI 2022
Introduction to Neural Networks, Convolution Neural Networks, and Deep Learning

Paul Rodriguez
PhD

Overview

- Overview of Neural Networks (aka Multilayer Perceptron)
- What is Deep Learning?
- Convolutions and feature discovery
- Convolution Neural Networks
- MNIST demonstration with Keras on Expanse
- What next?

to get neural network:

Consider the Logistic Function

(aka sigmoid)

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

to get neural network:

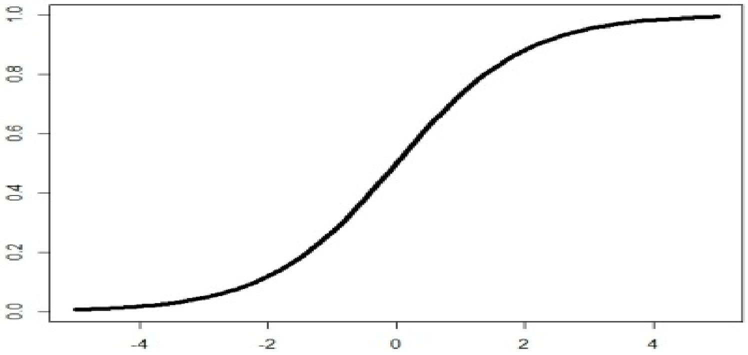
Consider the Logistic Function

(aka sigmoid)

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

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

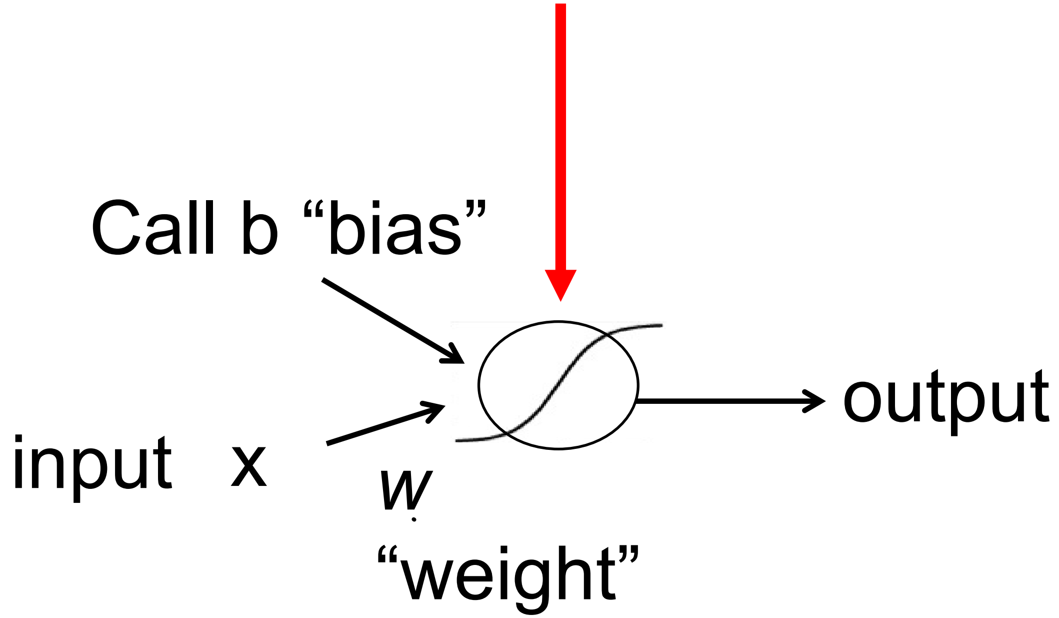
$y = f(x)$



x

Make a graphical description of Logistic Function

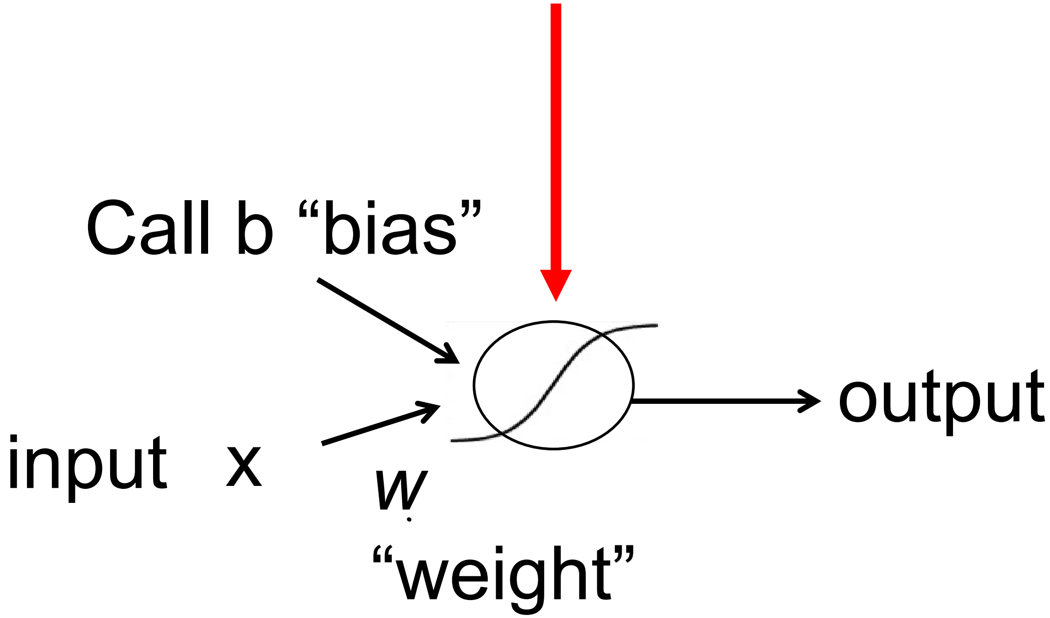
$$f(x) = \frac{1}{1 + exp^{(-(b+wx))}}$$



this node (or unit) will transform input to output with logistic activation function

Make a graphical description of Logistic Function

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



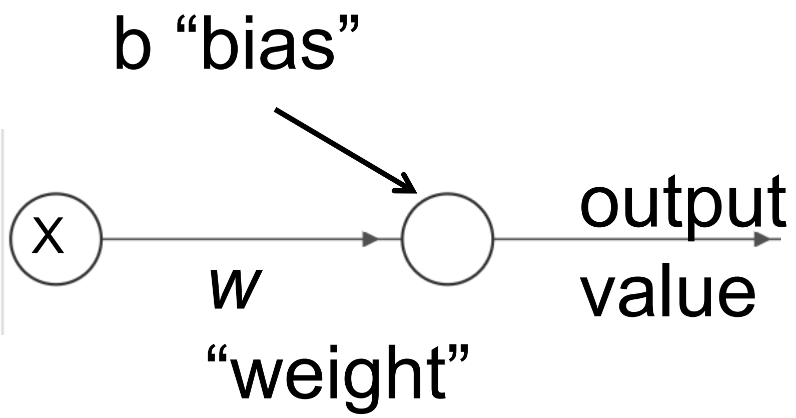
this node (or unit) will transform input to output with logistic activation function

Solution: find *w* & *b* that maximizes likelihood that output =1

(by using derivatives)

How does changing parameters affect function?

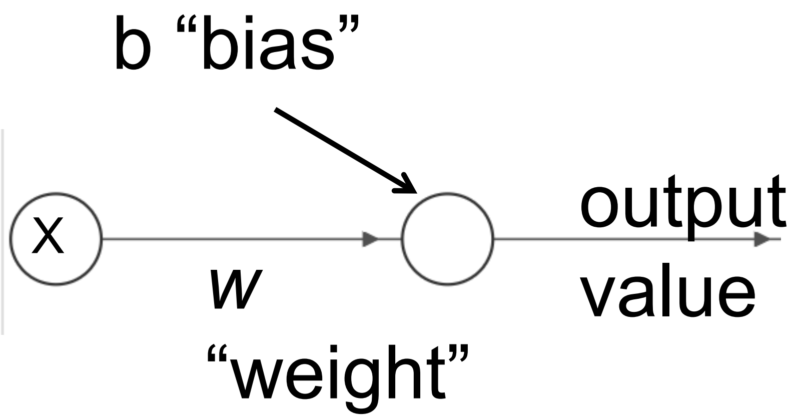
$$f(x) = \frac{1}{1 + exp^{(-(b+wx))}} \Leftrightarrow$$

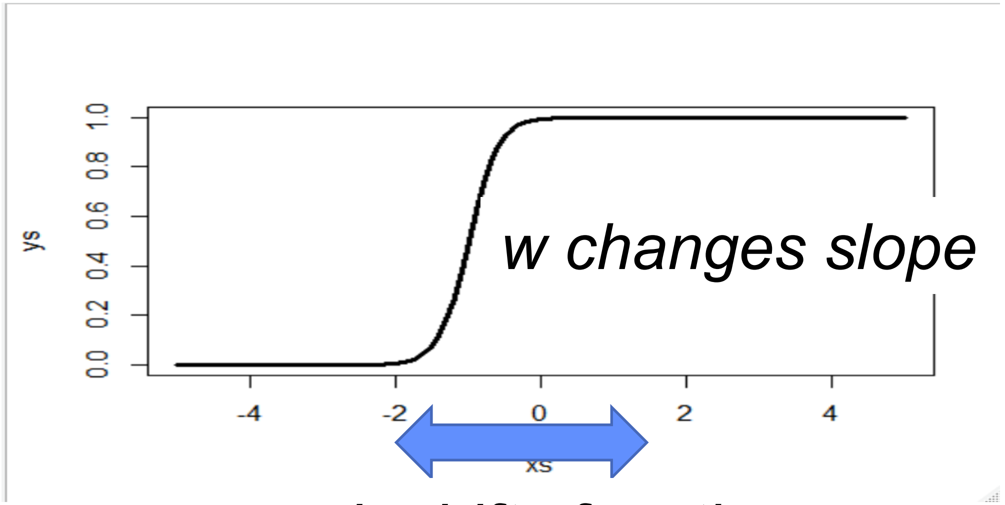


How does changing parameters affect function?

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

$\Leftrightarrow$

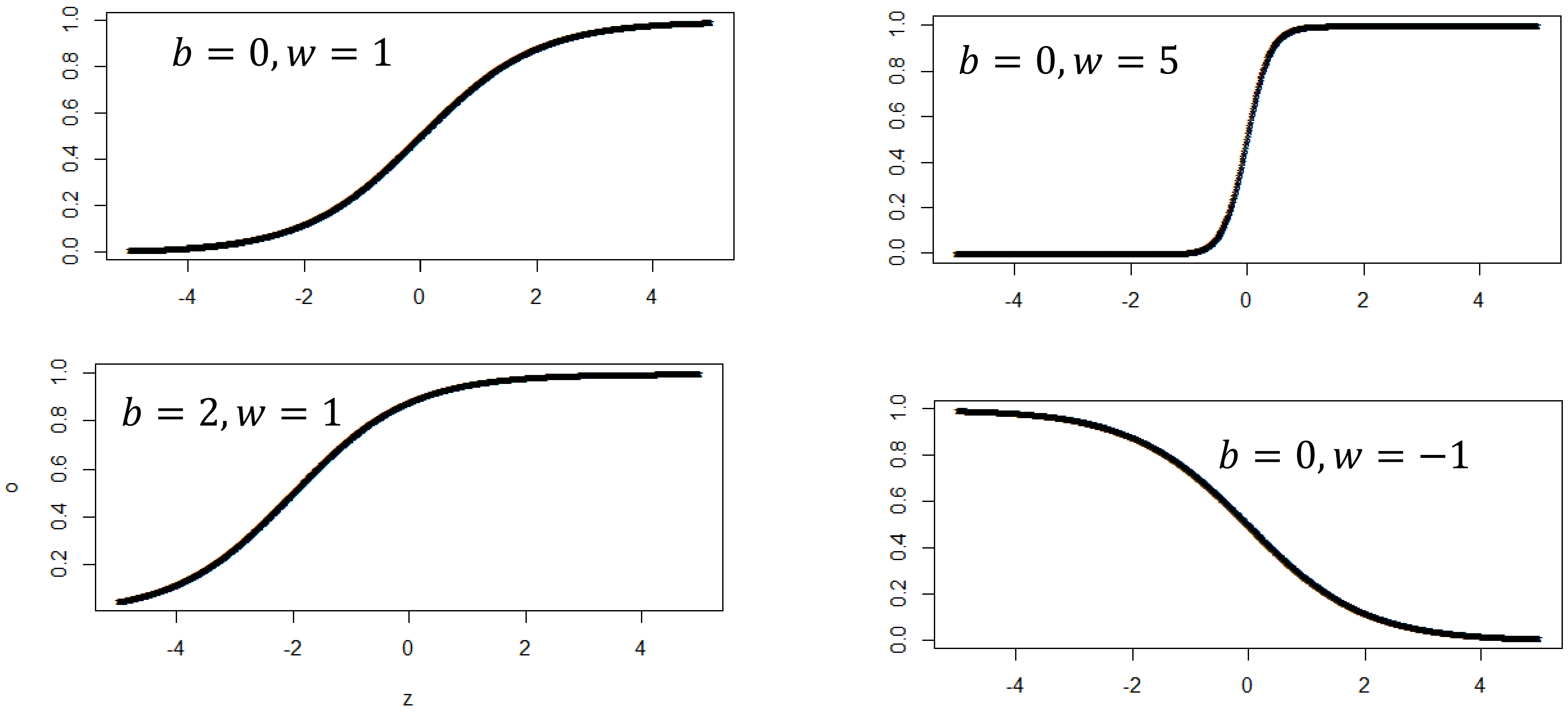




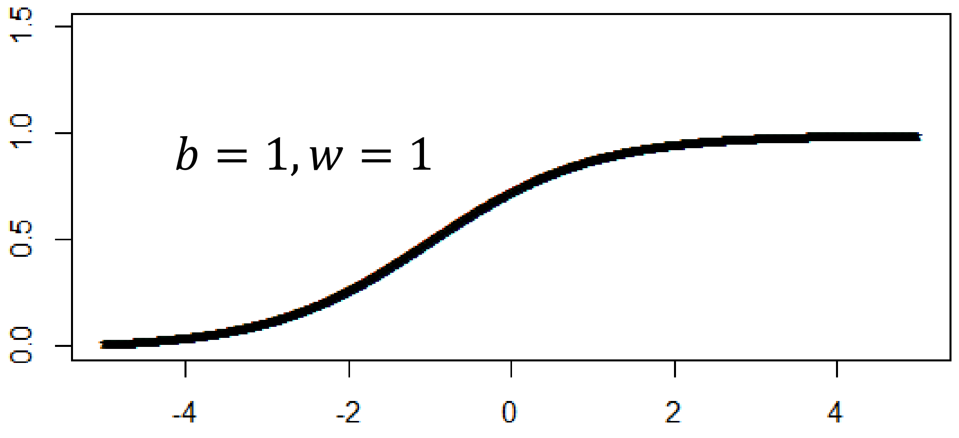
b shifts function

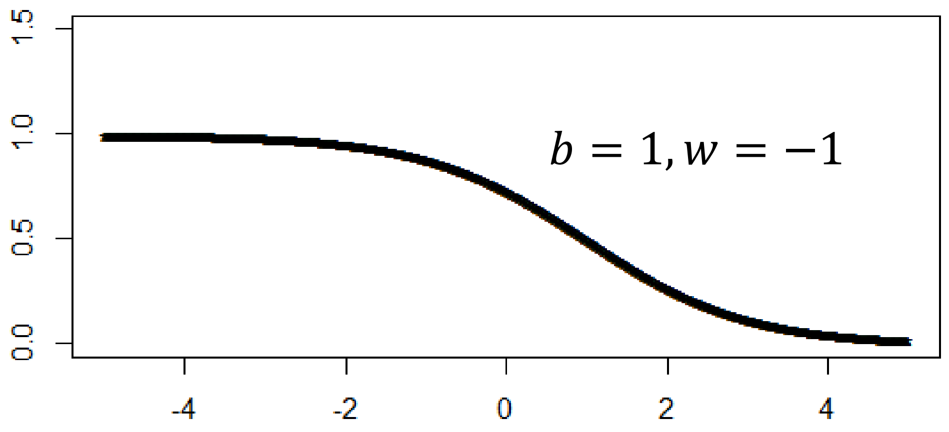
Logistic function w/various weights

$$\text{for } y = f(x) = \frac{1}{1 + exp^{(-(b+wx))}}$$



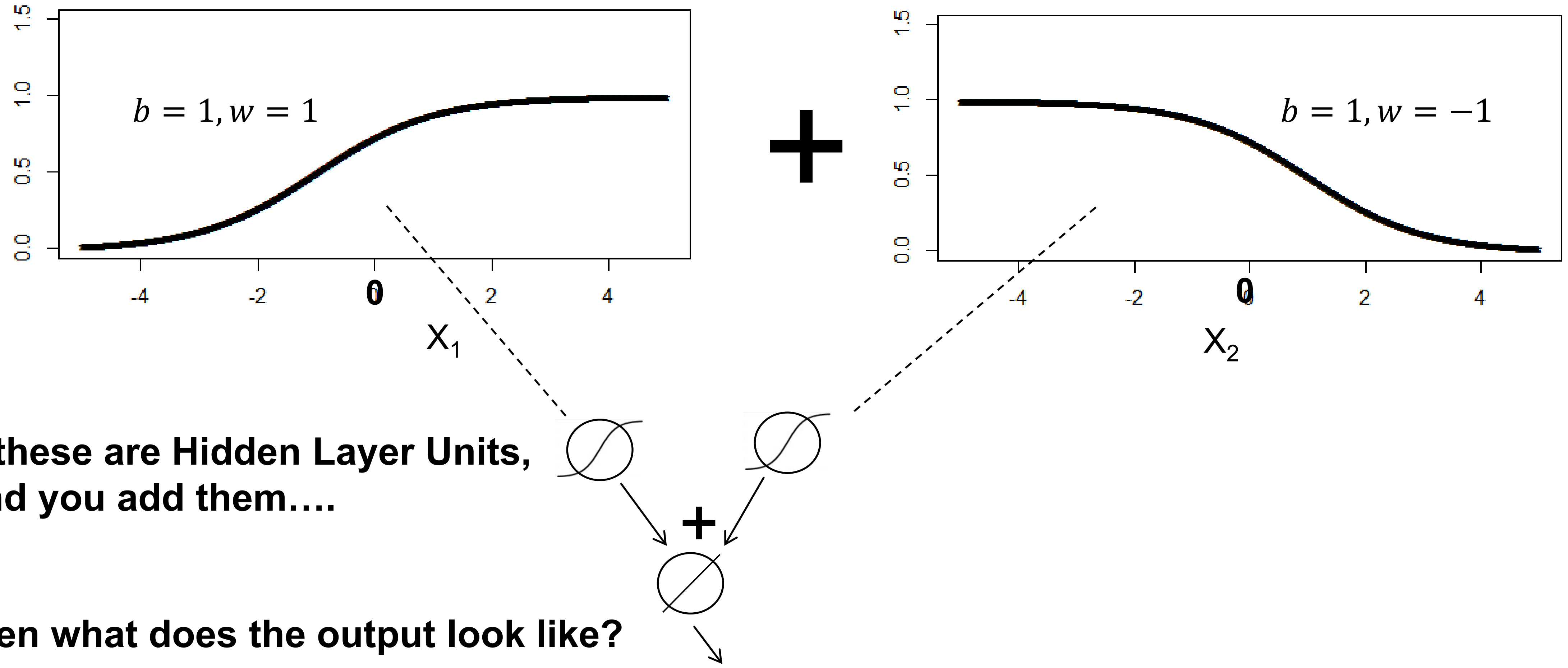
So combinations are highly flexible and nonlinear





(Note: these are both slightly shifted)

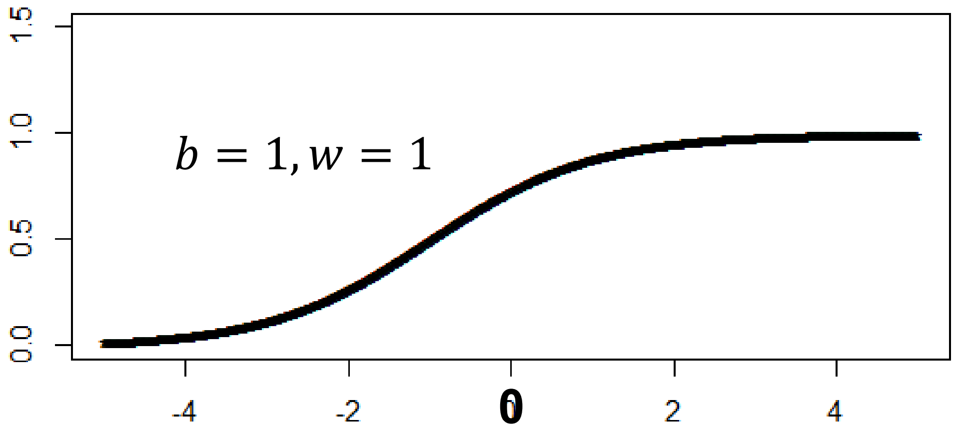
So combinations are highly flexible and nonlinear



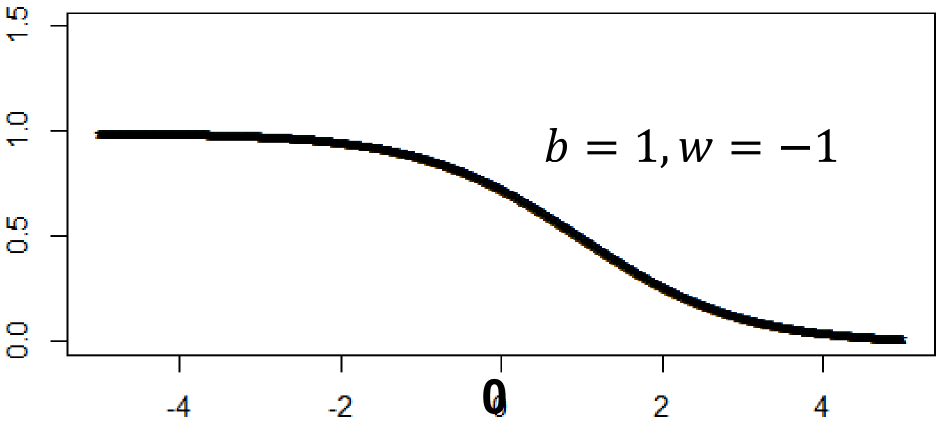
If these are Hidden Layer Units, and you add them….

then what does the output look like?

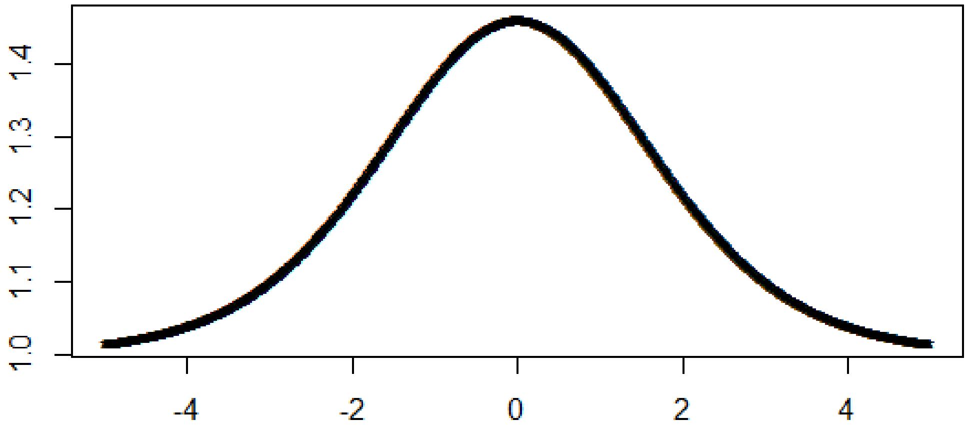
So combinations are highly flexible and nonlinear



+

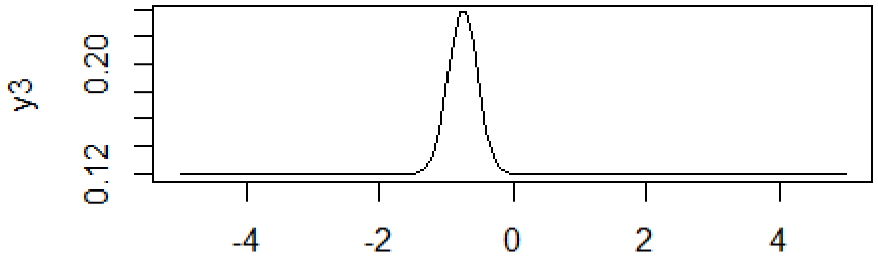


=

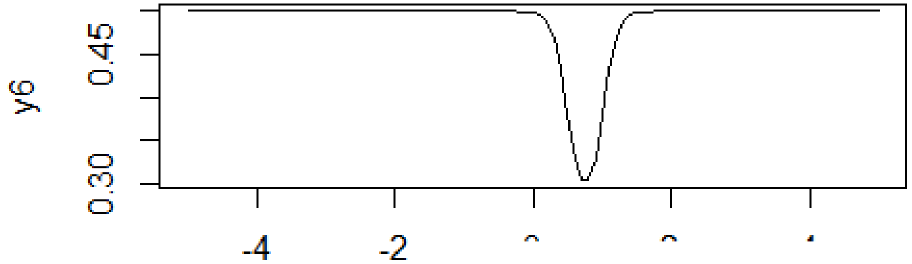


Higher level function combinations

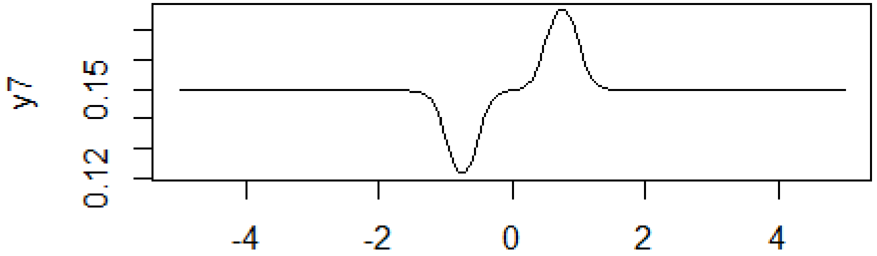
```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```




```
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")
```

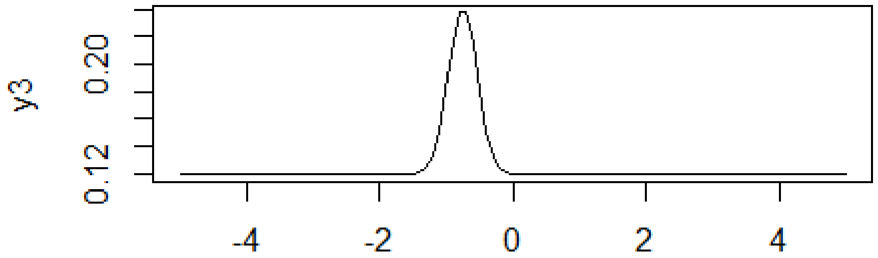



```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```



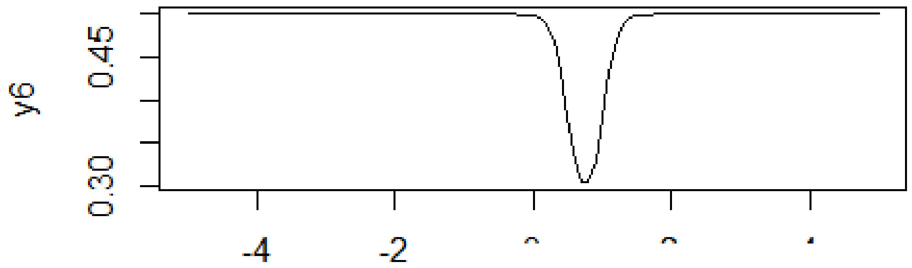
Higher level function combinations

```
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")
```

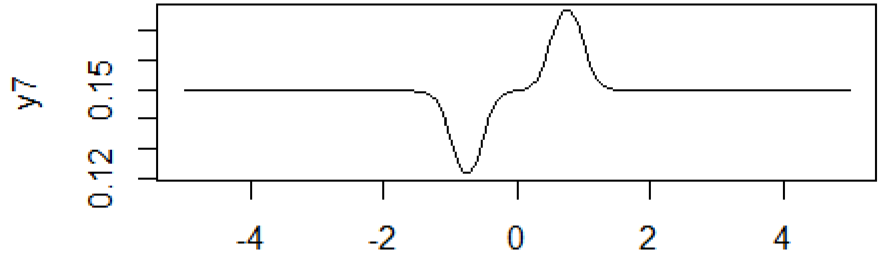


Multiple layer networks can represent any logical or real-valued functions (unbiased, but potential to overfit)

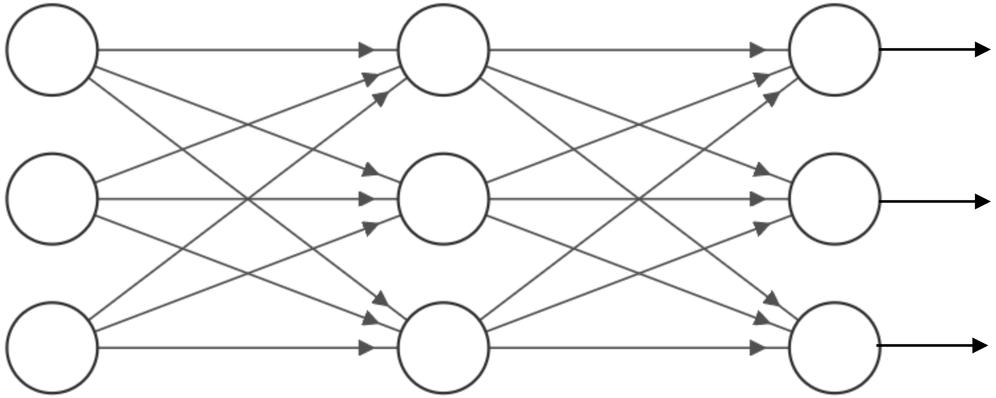
```
y4=1/(1+exp(10+ (-10)*x))
y5=1/(1+exp(-5+(10)*x))
y6=1/(1+exp(1-1*y4-1*y5))
plot(x,y6,type="l")
```




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



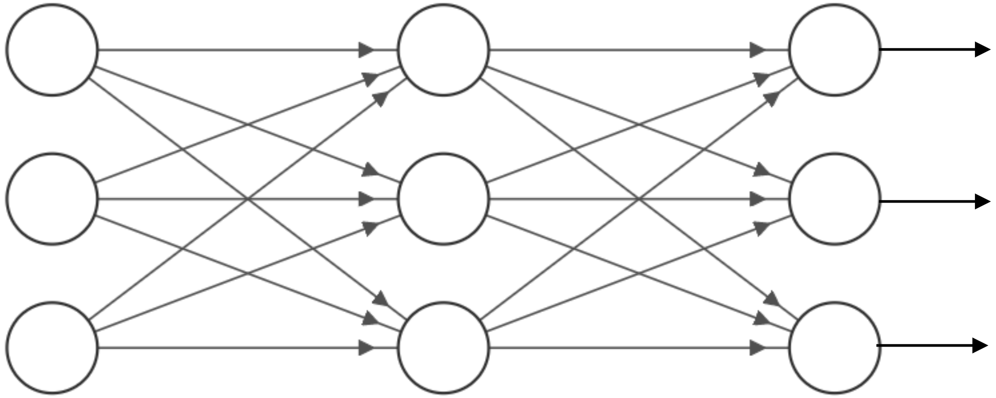
we can add layers and nodes



we can add layers and nodes

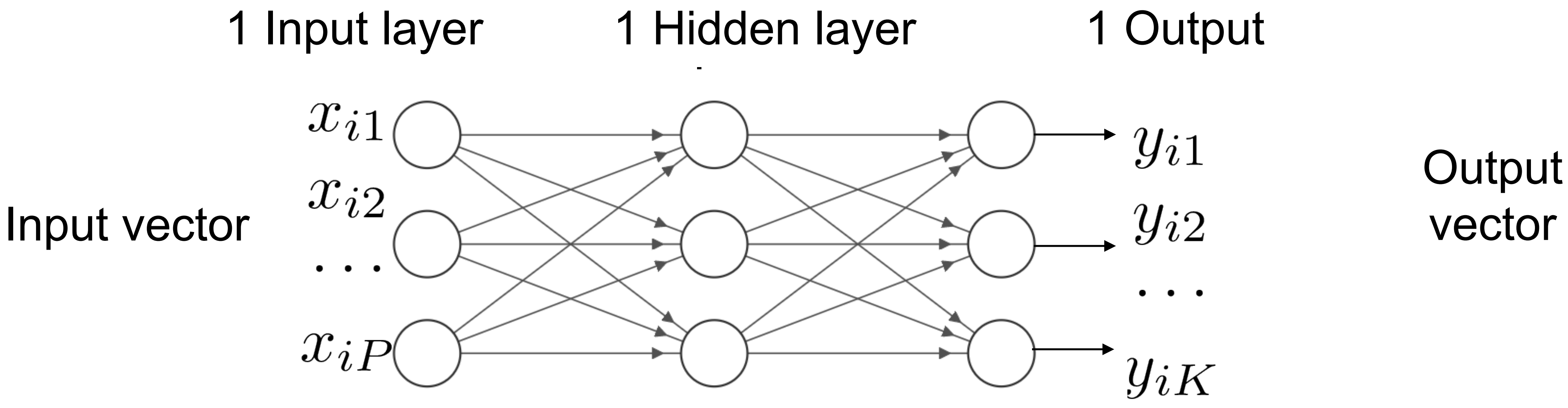
Multilayer Perceptron

1 Input layer   1 Hidden layer   1 Output



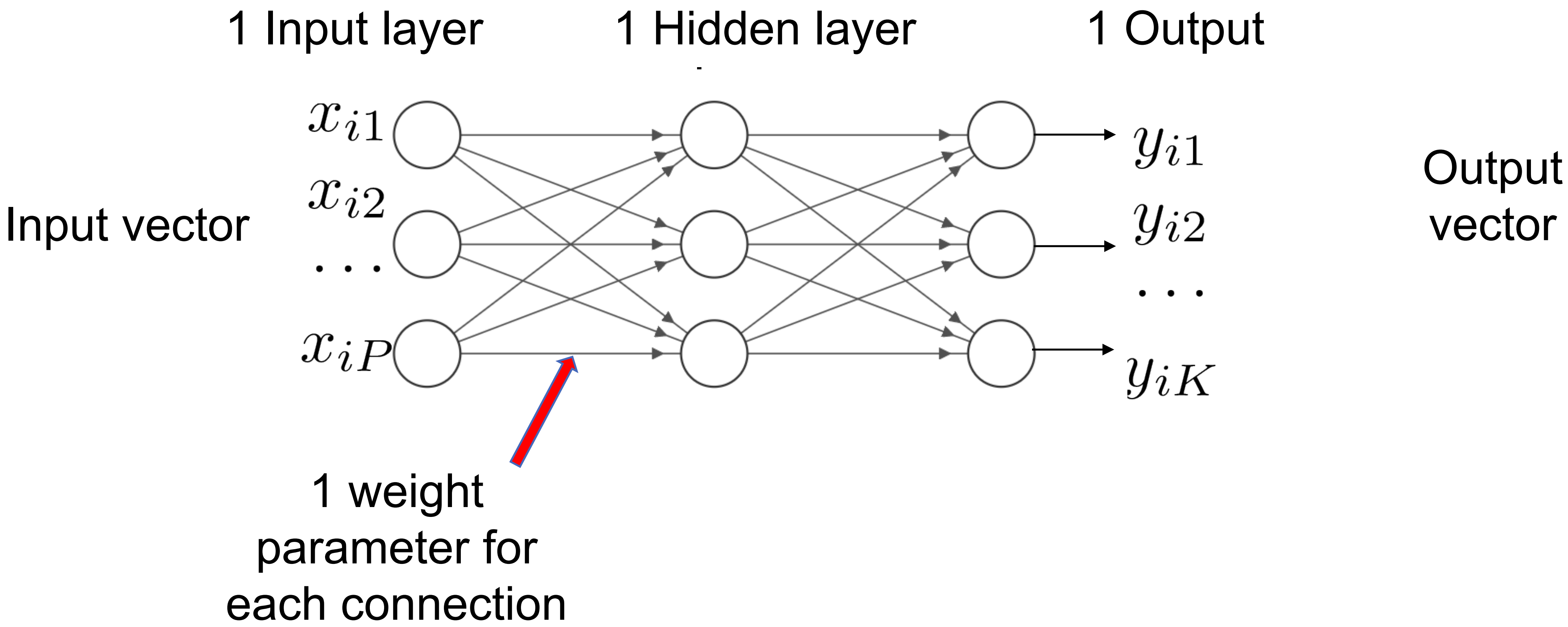
we can add layers and nodes

Multilayer Perceptron

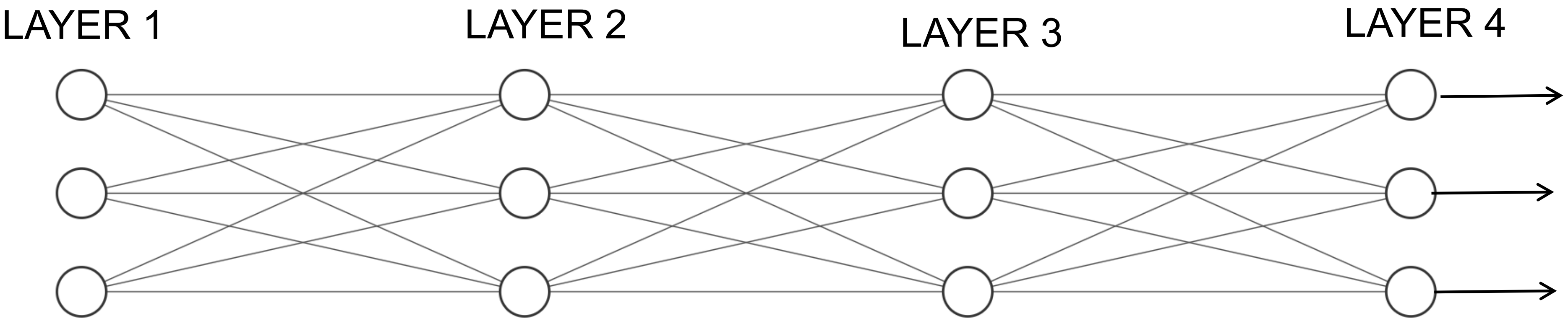


we can add layers and nodes

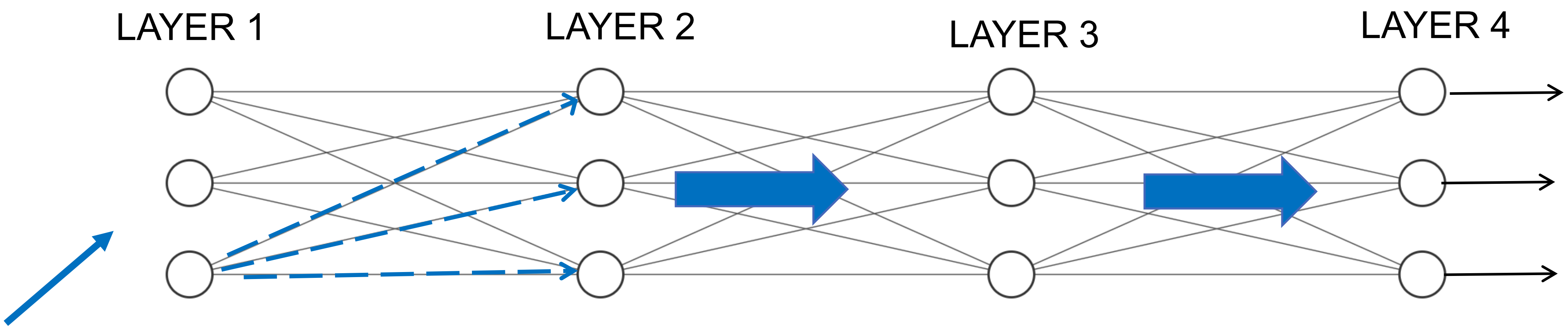
Multilayer Perceptron



First step: choose layers, connectivity, and activations



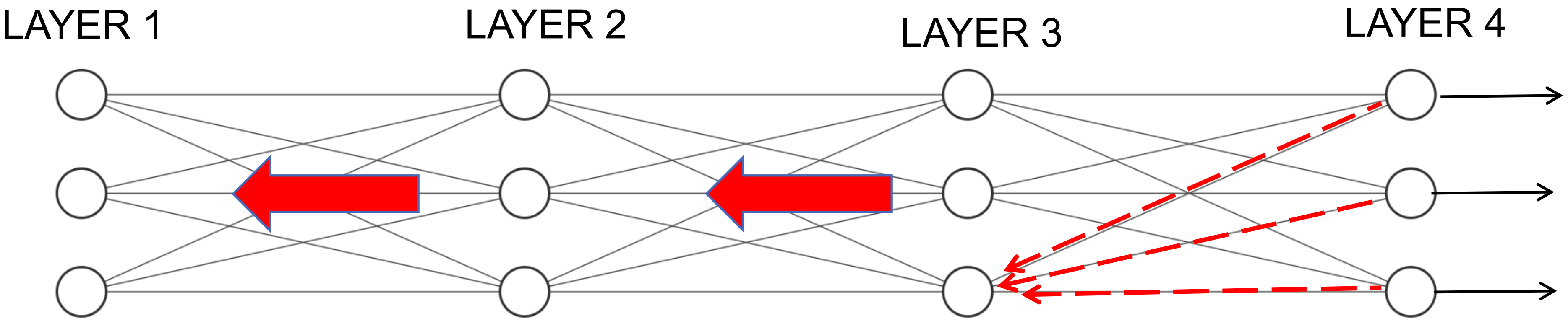
Algorithm steps:



1. **FORWARD PROPAGATE ACTIVATION**:

apply input data x_i ,
calculate all node activations

Algorithm steps:

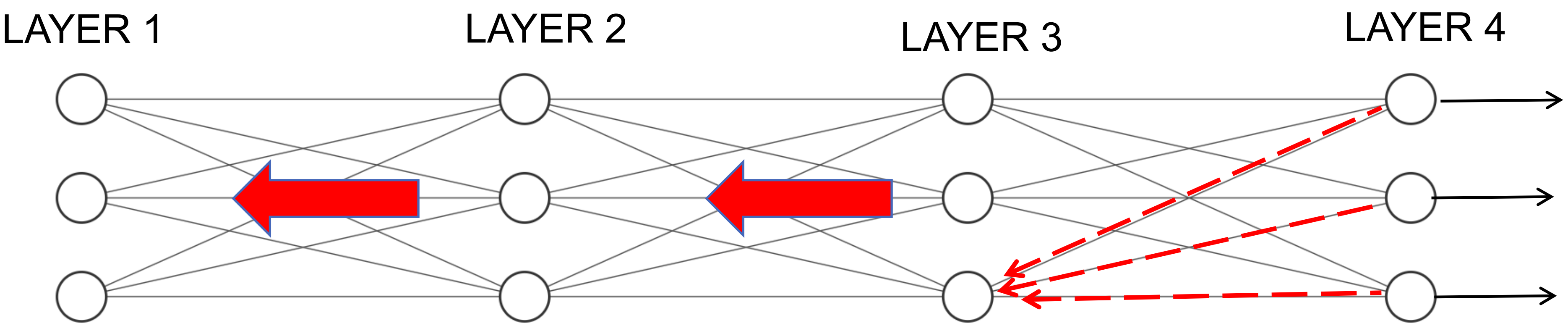


1. **FORWARD PROPAGATE ACTIVATION**:

apply input data x_i ,
calculate all node activations

2. BACKWARD PROPAGATE ERROR:
calculate Error (or Loss) derivatives, dE/dY,
pass it back to lower layer

algorithm steps:

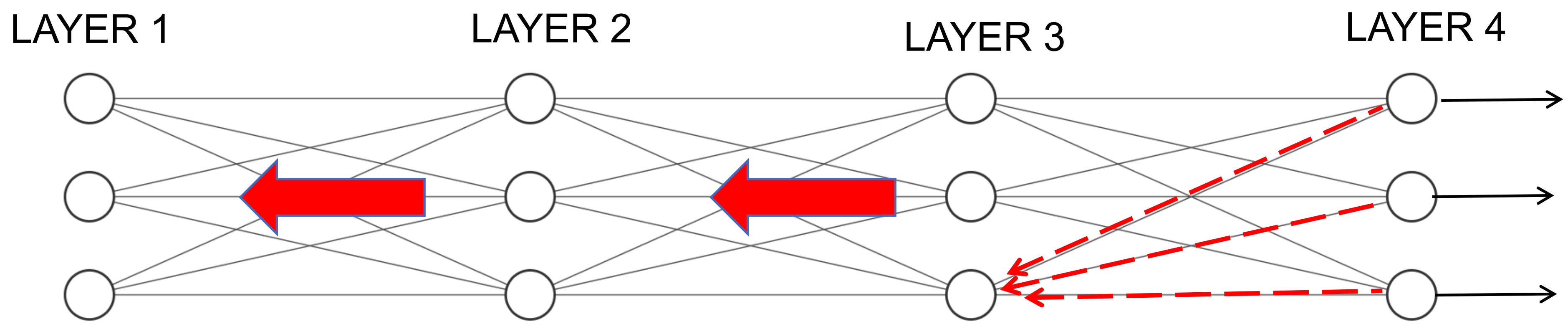


1. FORWARD PROPAGATE ACTIVATION:
apply input data x_i ,
calculate all node activations

2. BACKWARD PROPAGATE ERROR:
calculate Error (or Loss) derivatives, dE/dY,
pass it back to lower layer

For hidden layers use chain rule:
(dE/dY dY/dH_3 dH_3/dH_2 etc…)
needs a summation of previous layer

algorithm steps:



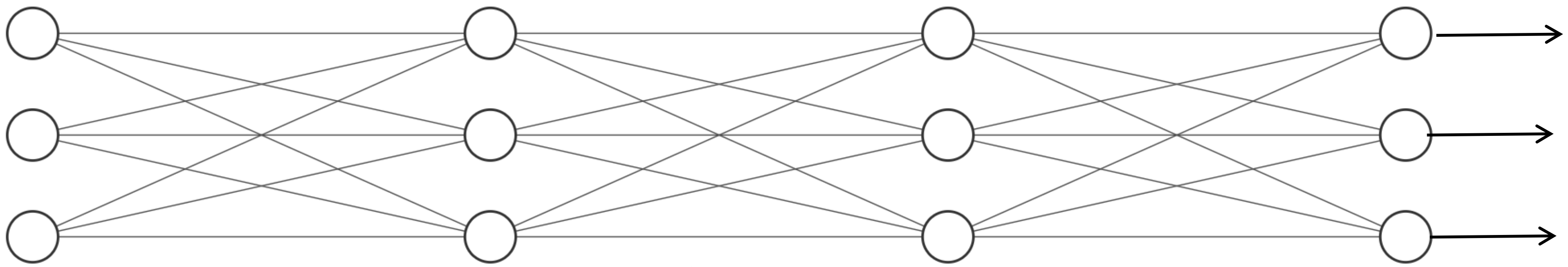
1. FORWARD PROPAGATE ACTIVATION:
apply input data x_i ,
calculate all node activations

2. BACKWARD PROPAGATE ERROR:
calculate Error (or Loss) derivatives, dE/dY,
pass it back to lower layer

For hidden layers use chain rule:
($dE/dY \quad dY/dH_3 \quad dH_3/dH_2$ etc…)
needs a summation of previous layers

Beware: error signals get diluted as you go backward - the 'vanishing gradient' problem

algorithm steps:



1. **FORWARD PROPAGATE ACTIVATION**:

apply input data x_i ,
calculate all node activations

2. BACKWARD PROPAGATE ERROR:
calculate Error (or Loss) derivatives (dE/dY)
pass it back to lower layer

3. Update weights and bias terms

$$w_{ji} = w_{ji} - \eta \frac{dE}{dw_{ji}}$$

NN Algorithm

INITIALIZE WEIGHTS (small random values)

NN Algorithm

INITIALIZE WEIGHTS

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

NN Algorithm

INITIALIZE WEIGHTS

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all derivatives to ***minimize Loss (dL)***

NN Algorithm

INITIALIZE WEIGHTS

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all derivatives to ***minimize Loss (dL)***

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

NN Algorithm

INITIALIZE WEIGHTS

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all derivatives to ***minimize Loss (dL)***

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

STOP: when validation loss reaches minimum or converges

NN Algorithm
[heuristics, options to learn faster and/or better]

INITIALIZE WEIGHTS **[use truncated distributions]**

LOOP until stopping criterion: **[work in batches of input]**

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all derivatives to ***minimize Loss (dL)***

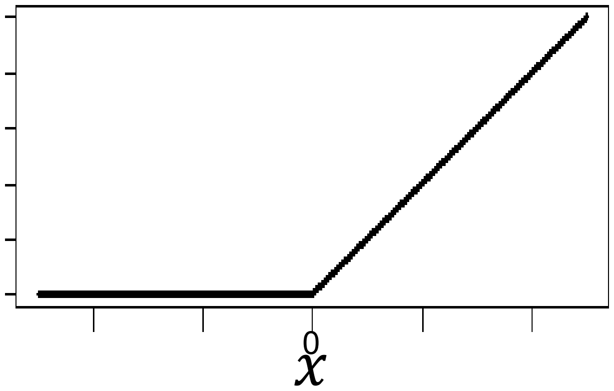
UPDATE WEIGHTS: $w \leftarrow w - learning\_rate * \frac{dL}{dw}$ **[adapt learning rate, use momentum]**

STOP: when validation loss reaches minimum or converges **[several metrics of loss are possible]**

A heuristic for deep networks

RELU activation function

RELU (rectified linear

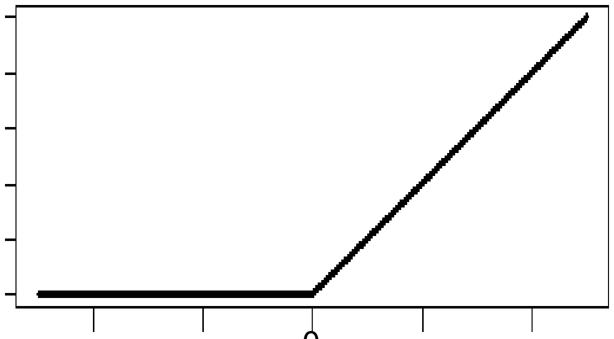


$$f(a) = \begin{cases} a & a > 0 \\ 0 & a <= 0 \end{cases}$$

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

A heuristic for deep networks

RELU (rectified linear



RELU activation function

It is unscaled (bad!)

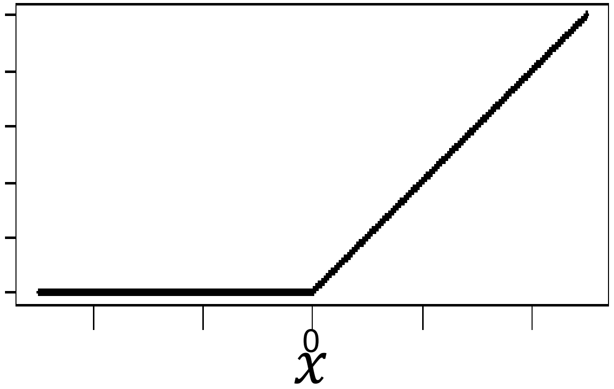
But *df/da* is constant (good!)

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

where $a = XW$

A heuristic for deep networks

RELU (rectified linear



RELU activation function

It is unscaled (bad!)

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

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

where $a = XW$

RELU helps mitigates vanishing gradients

Summary:

Pro:

- Neural Networks in general, are flexible, powerful learners
- Hidden layers learn a nonlinear transformation of input
- Many heuristics about what works

Summary:

Pro:

- Neural Networks in general, are flexible, powerful learners
- Hidden layers learn a nonlinear transformation of input
- Many heuristics about what works

Con:

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

What is deep learning?

What is deep learning?

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

What is deep learning?

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

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

What is deep learning?

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 pause

onto Convolution Networks

Image features

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

  (National Inst. of Standards and Technology)

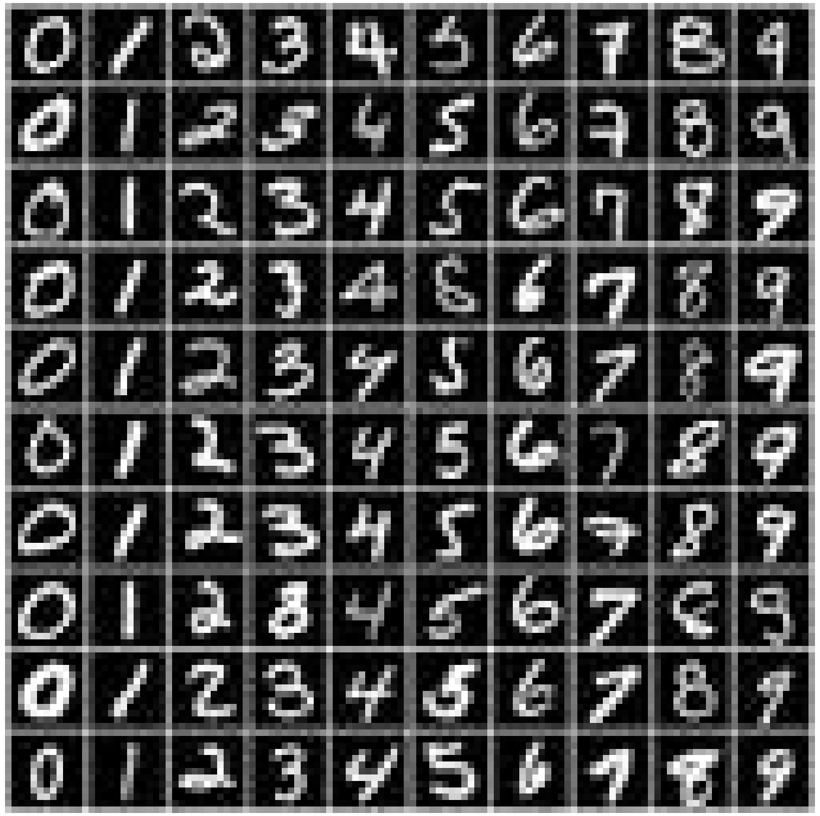
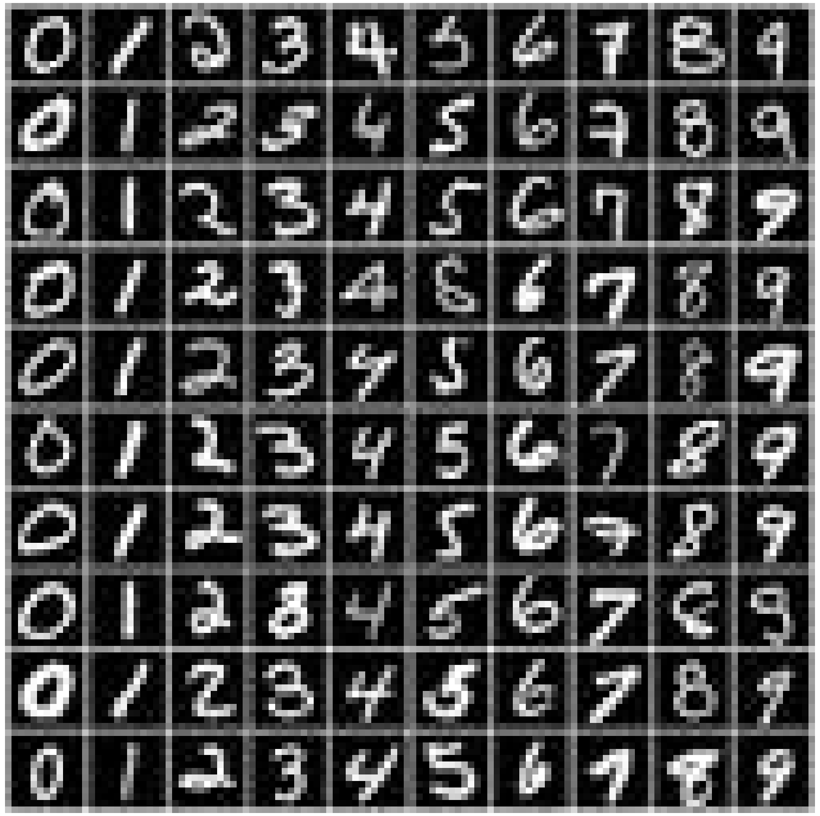


Image features

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

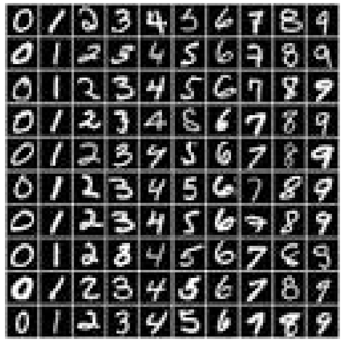
  (National Inst. of Standards and Technology)



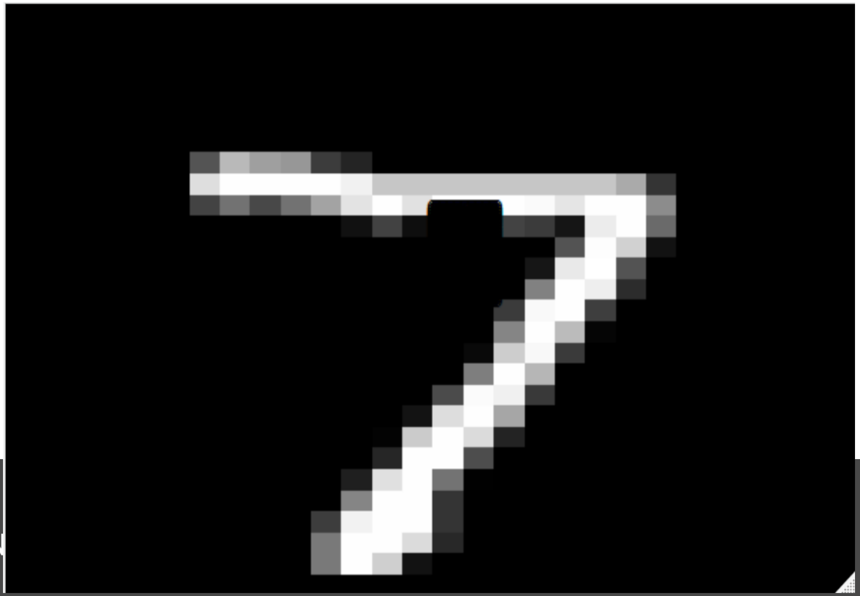
How to classify digits?

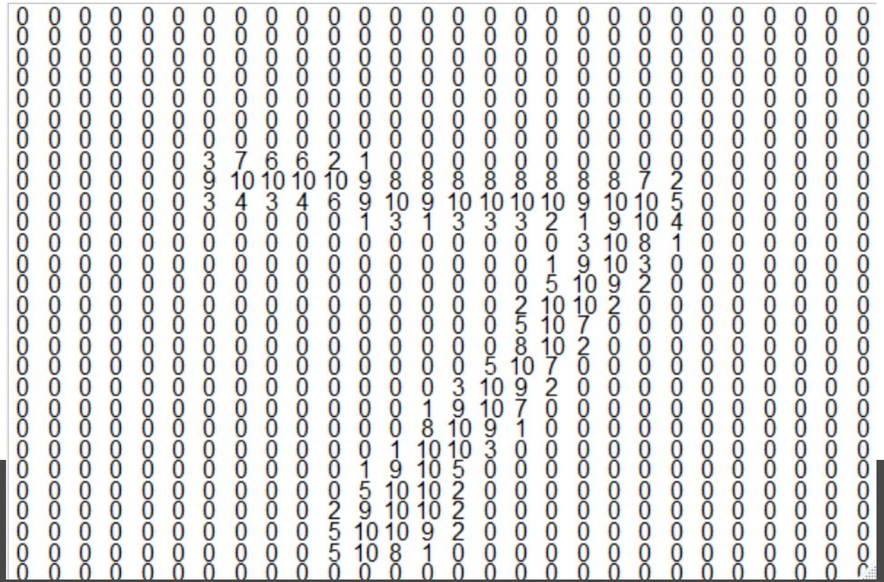
Image features

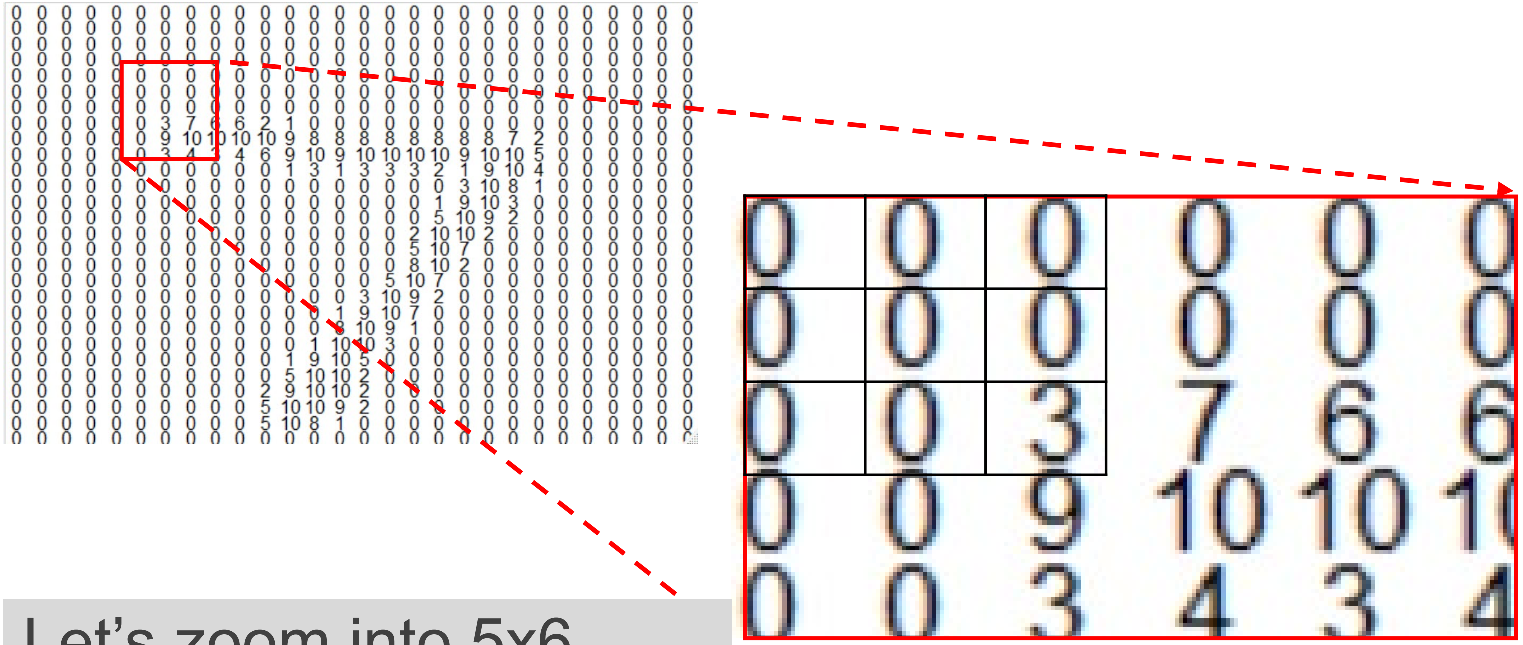
- **MNIST - A database of handwritten printed digits**
  (National Inst. of Standards and Technology)



How to classify digits?

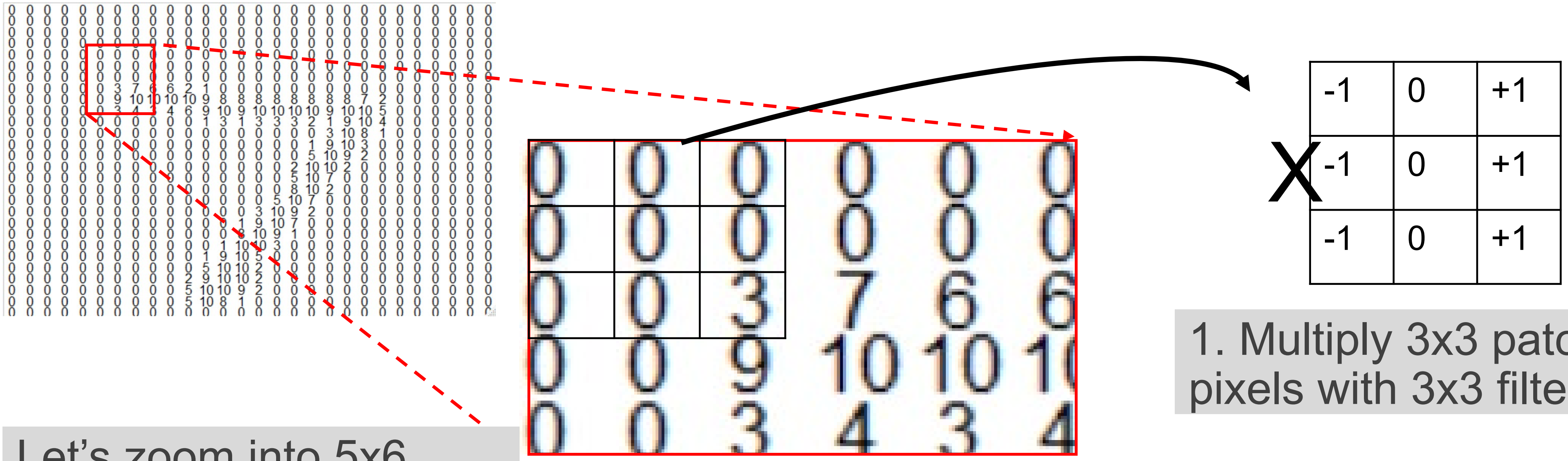






Let’s zoom into 5x6 window of pixels near the tip of ‘7’

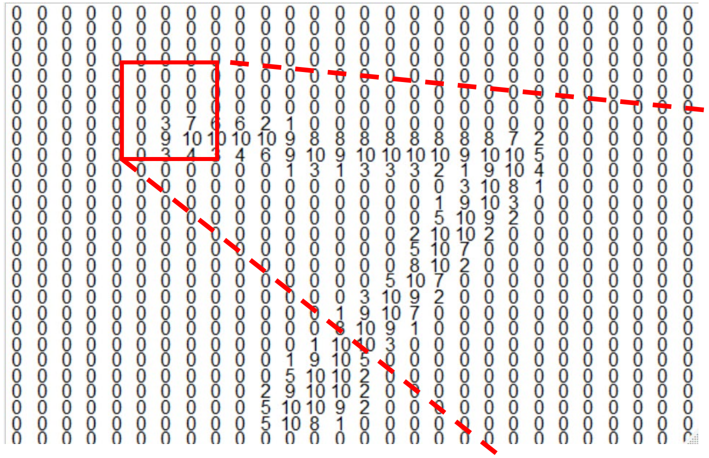
Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge

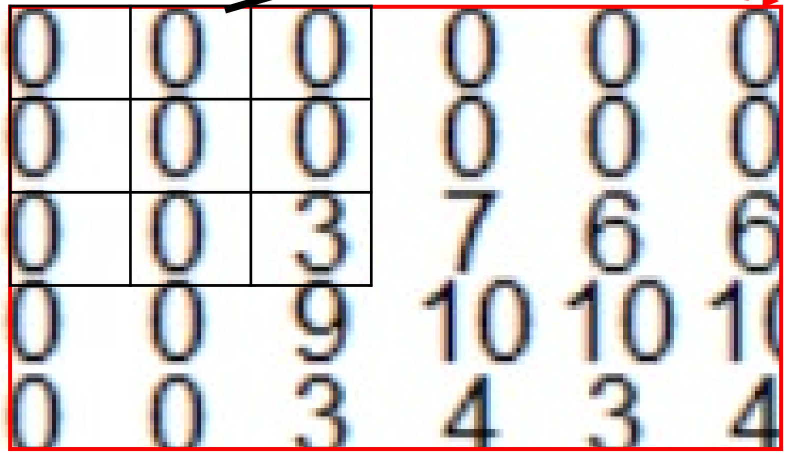


1. Multiply 3x3 patch of pixels with 3x3 filter

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





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

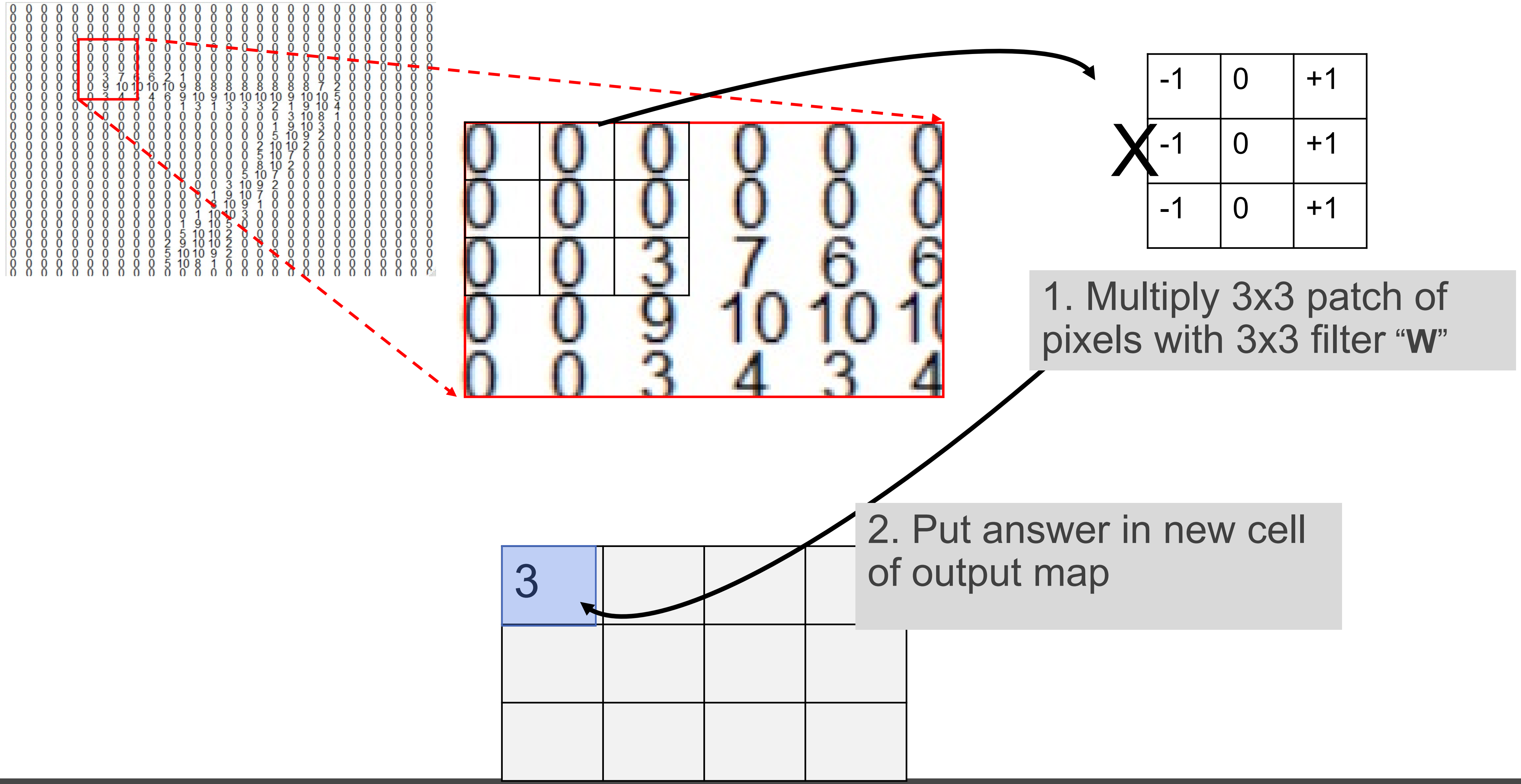
Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge

(our weight parameters)

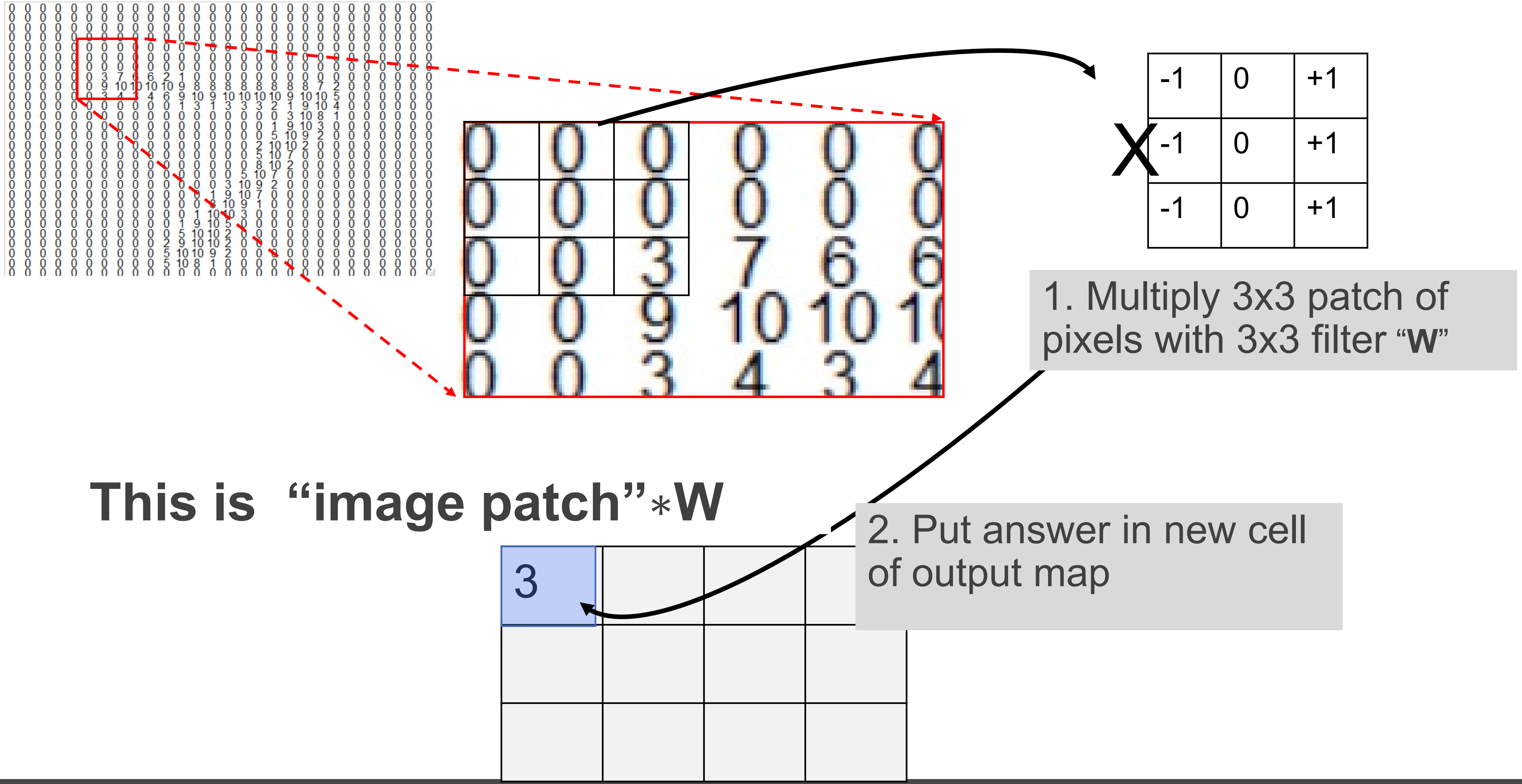
X

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

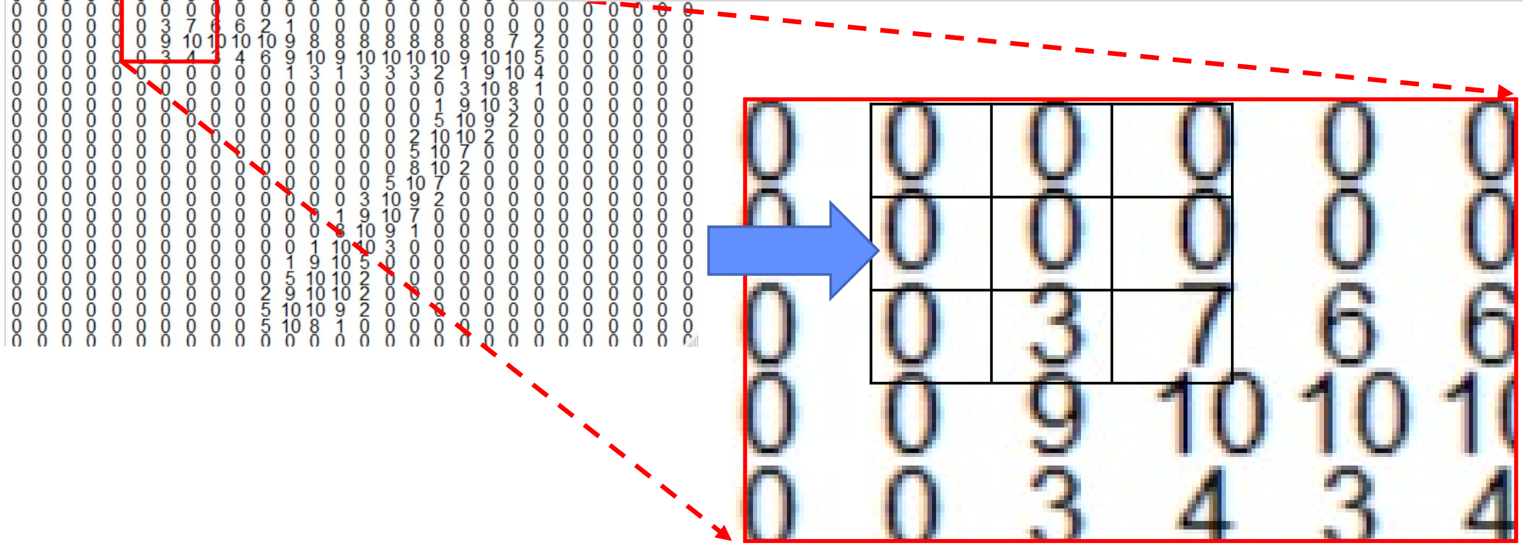
1. Multiply 3x3 patch of pixels with 3x3 filter “**W**”

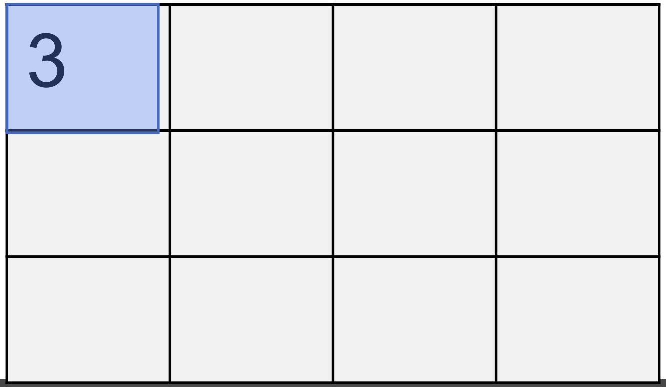

0 0 0 0 0 0
0 0 0 0 0 0
0 0 3 7 6 6
0 0 9 10 10 1(
0 0 3 4 3 4
X
-1 0 +1
-1 0 +1
-1 0 +1
1. Multiply 3x3 patch of pixels with 3x3 filter “**W**”
2. Put answer in new cell of output map
3

This is "image patch"$*$W

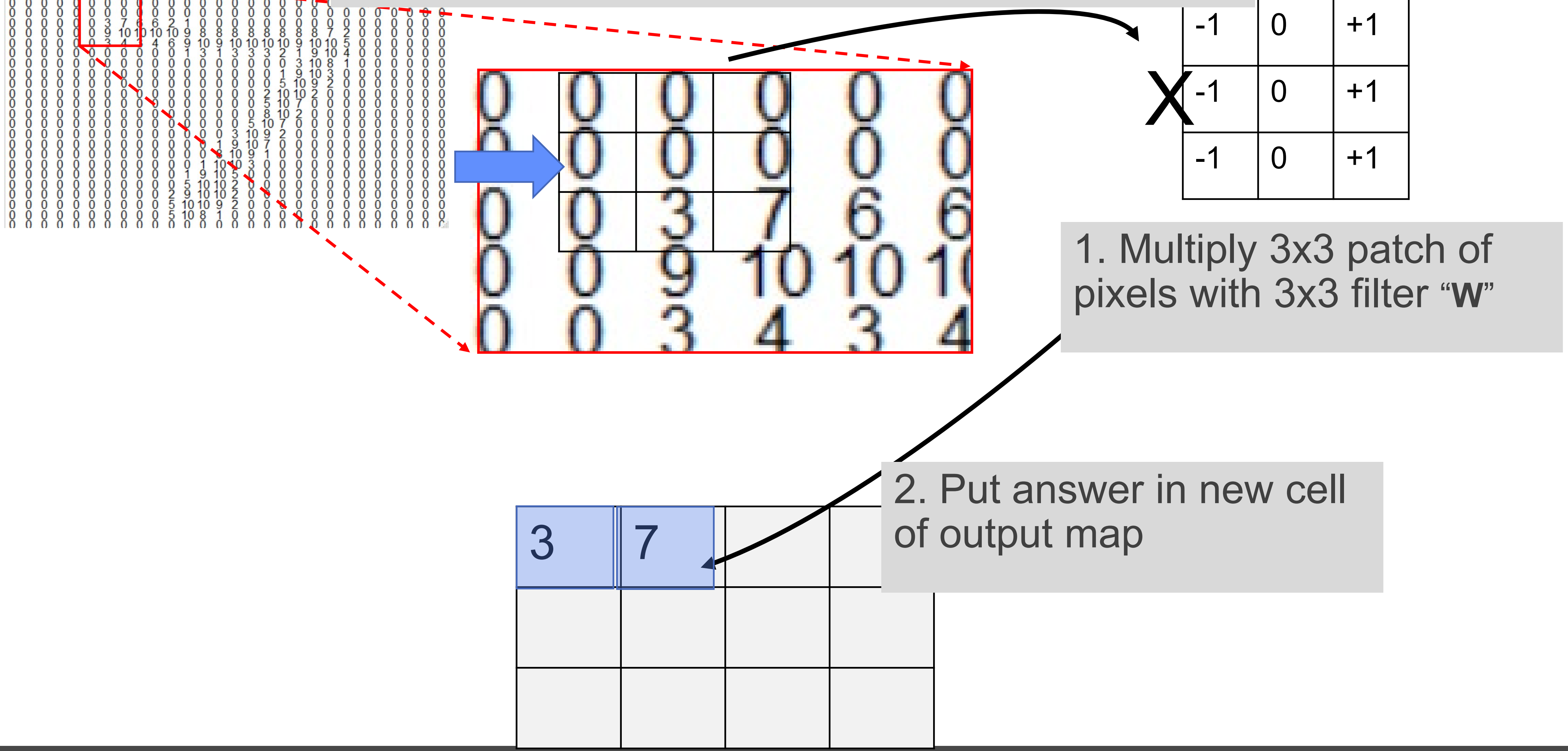


3. Slide filter horizontally to get next output value

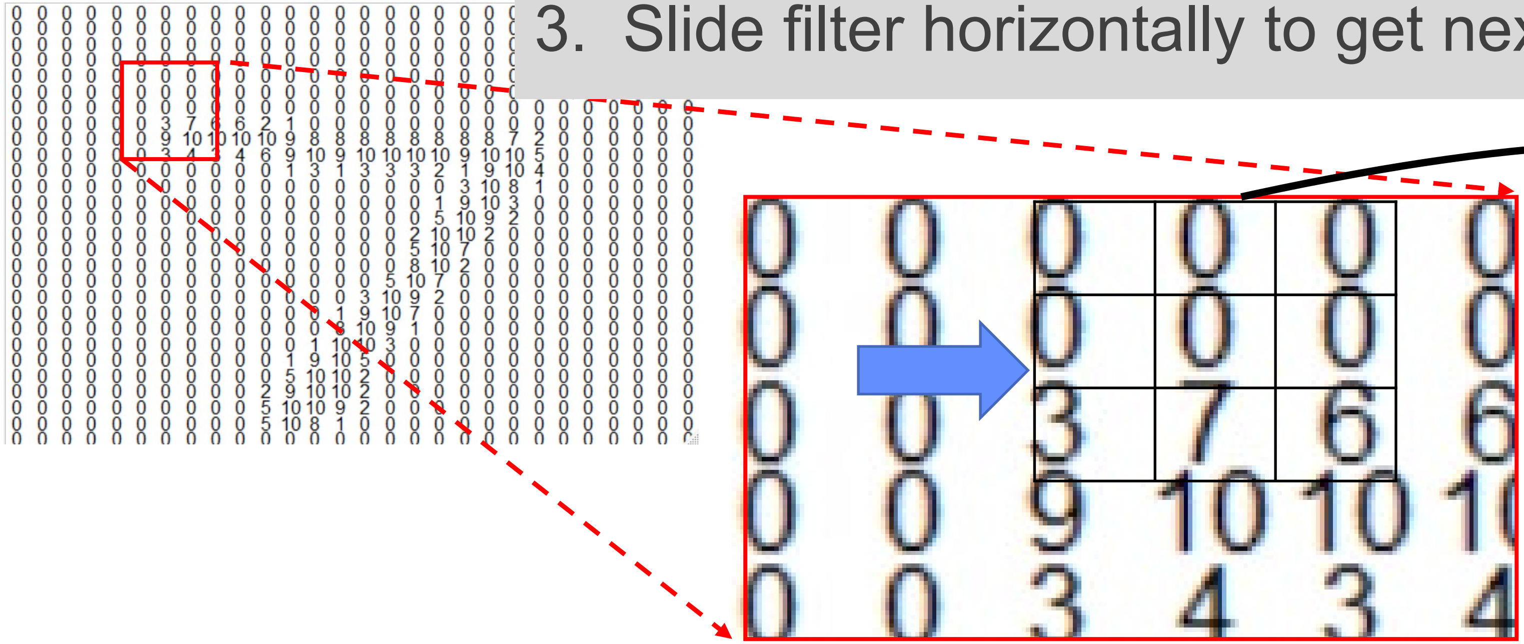




3. Slide filter horizontally to get next output value



3. Slide filter horizontally to get next output value

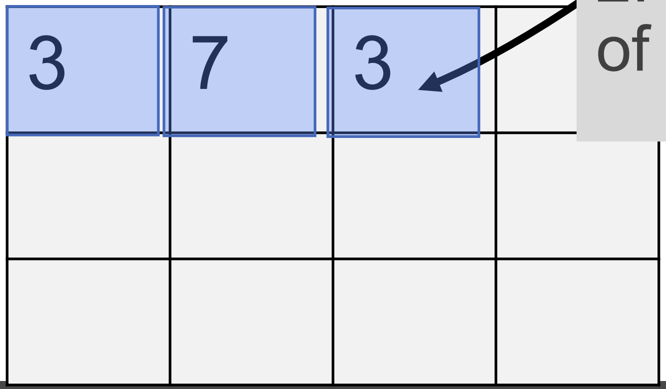


X

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

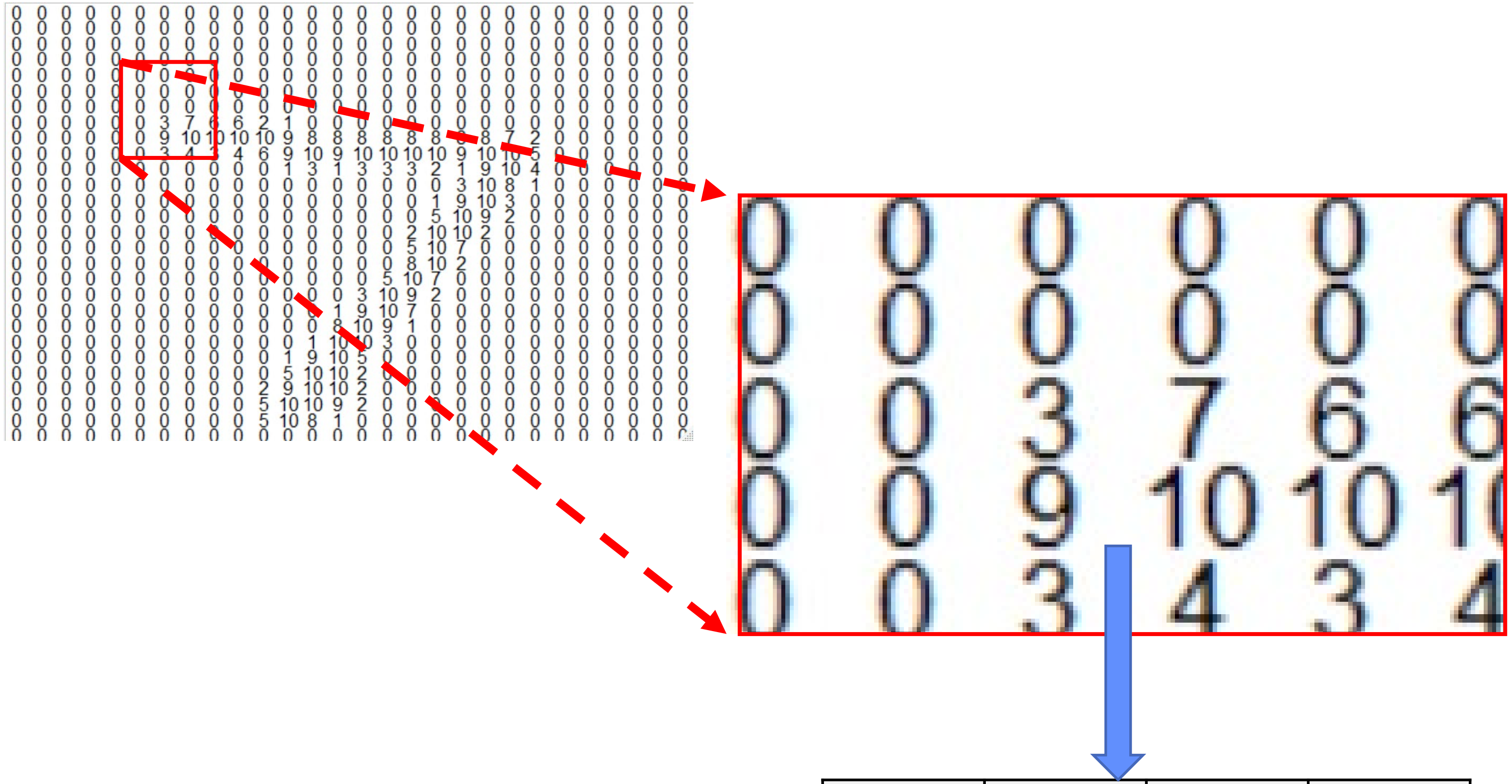
1. Multiply 3x3 patch of pixels with 3x3 filter “**W**”

2. Put answer in new cell of output map

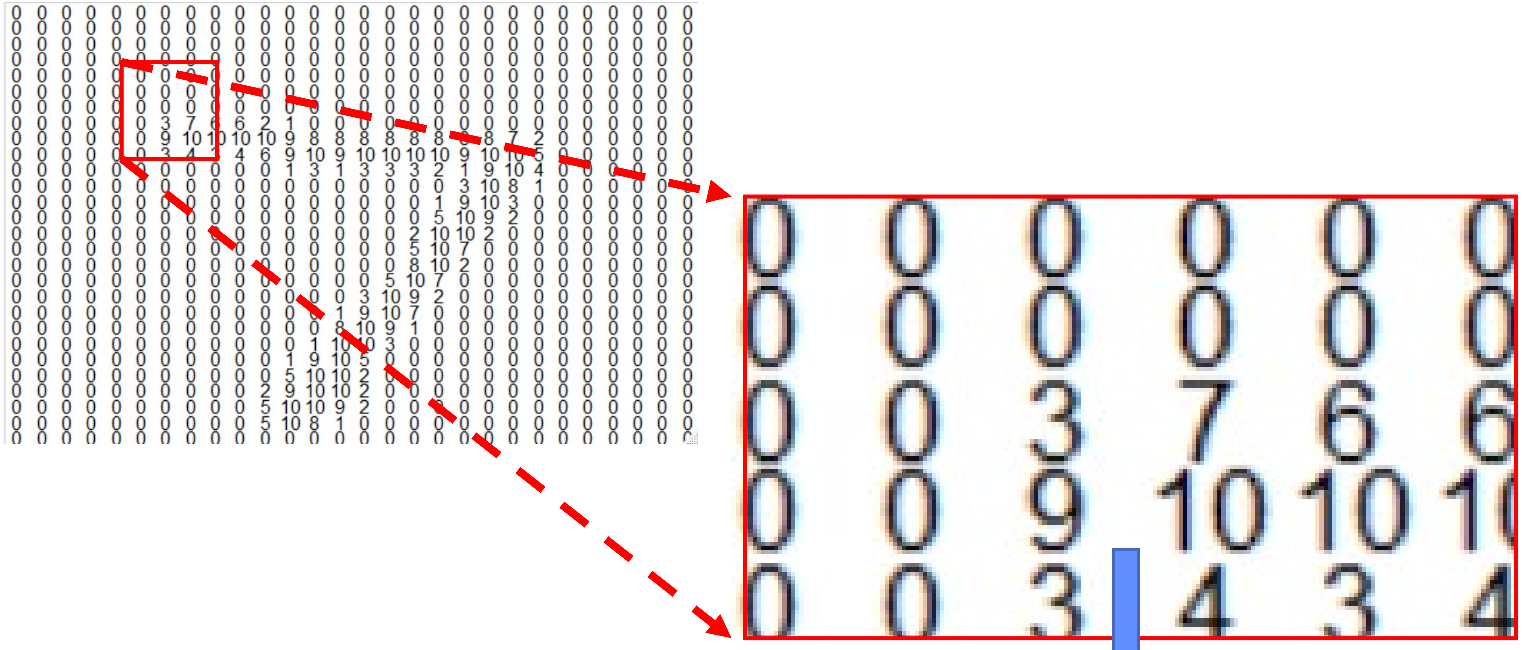


NOTE: sliding a filter is known as a “convolution” operation

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



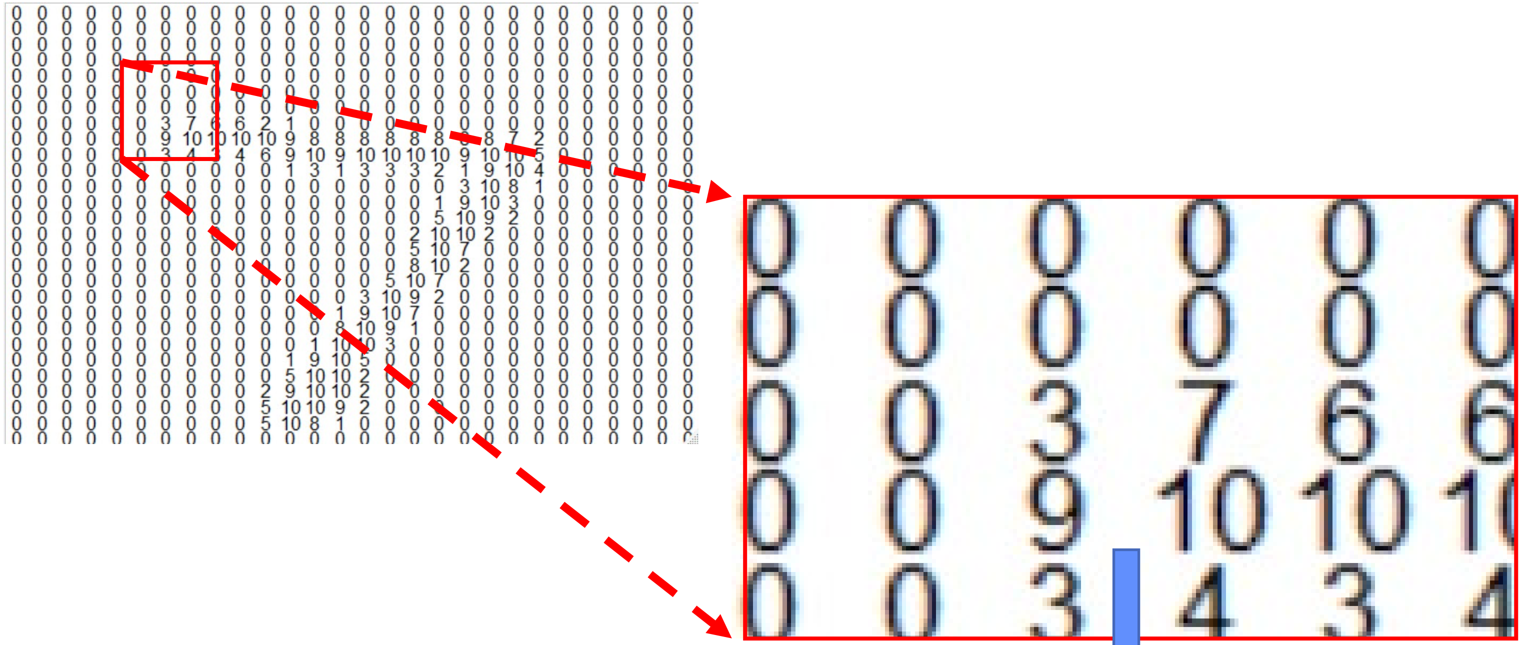
3	7	3	-1
12	17	4	-1
15	21	4	-1



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.***

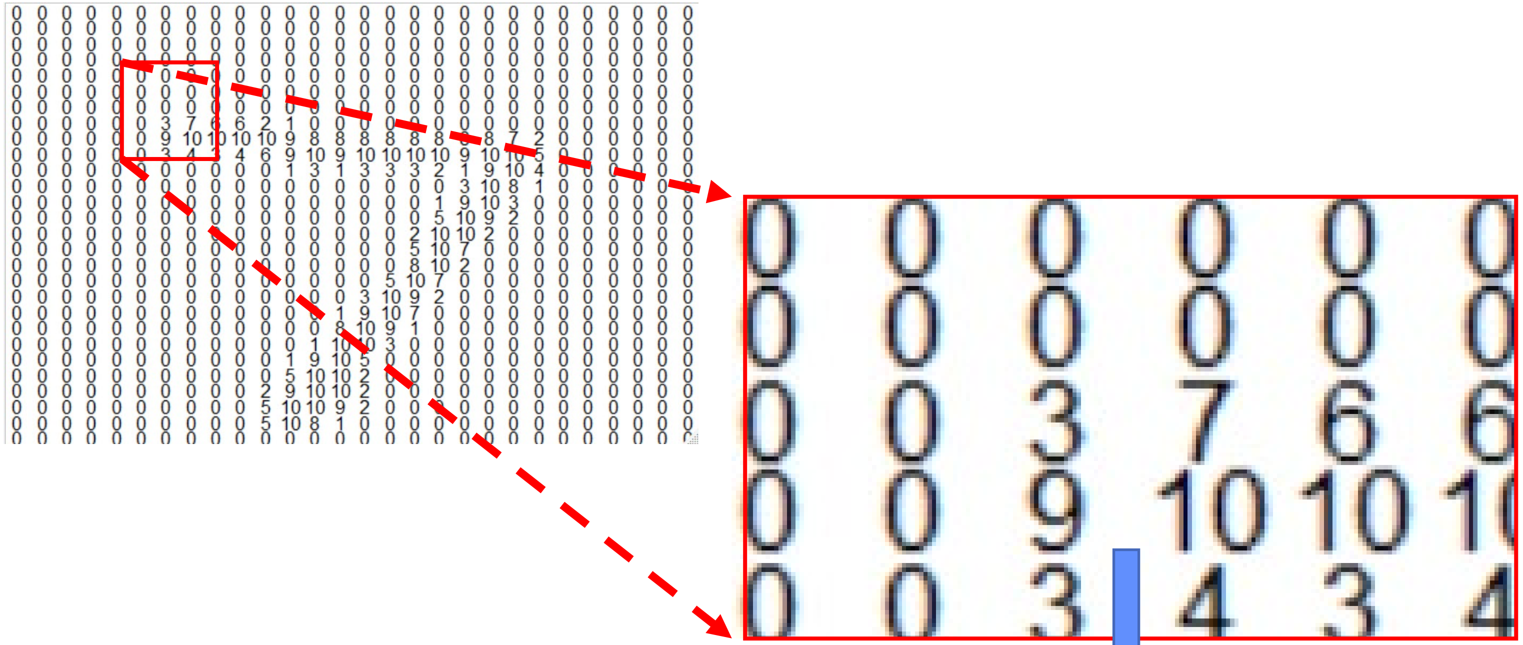
What do the highest values in the feature map represent?



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”



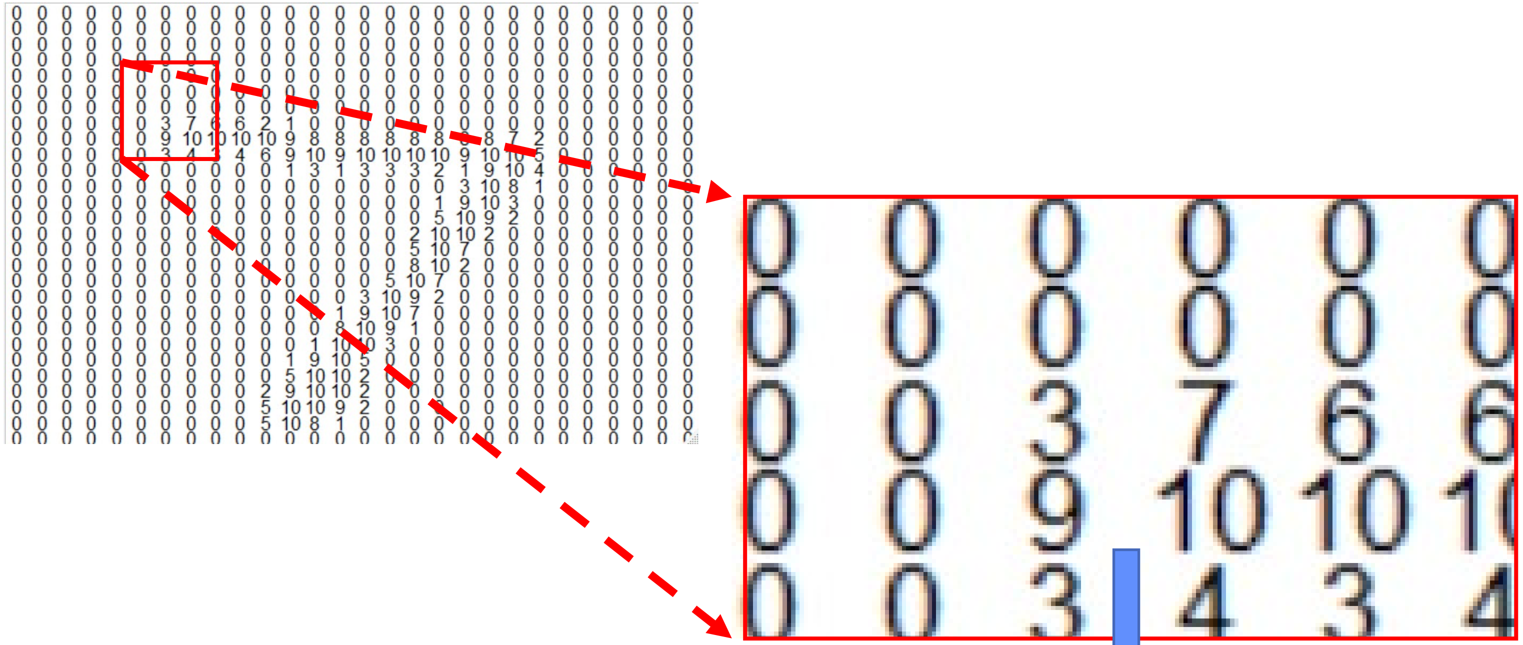
3	7	3	-1
12	17	4	-1
15	21	4	-1

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

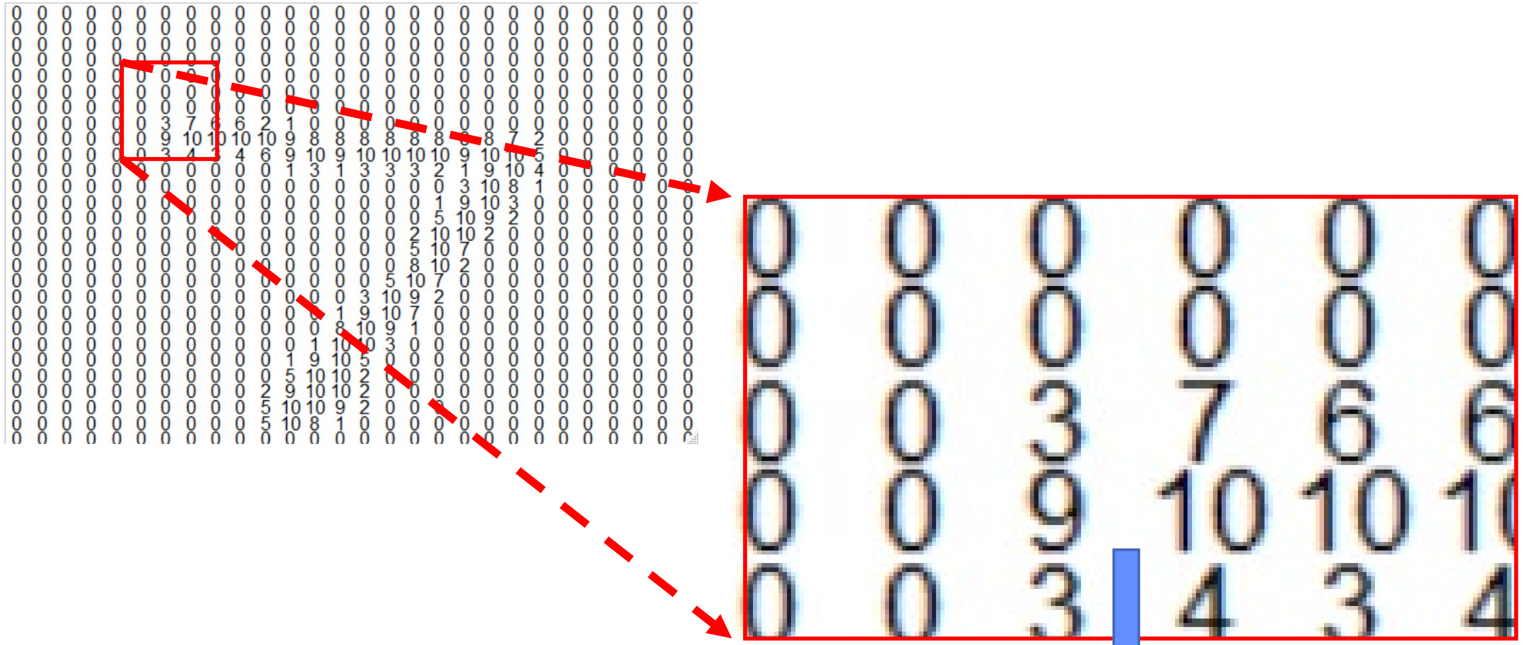
Optional next step:

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

Slide filter …

17	17	4
21	21	4

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



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

Slide filter …

17	17	4
21	21	4

Feature engineering

In Computer Vision there are many kinds of edge detectors and many ways to scale them

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

But building features is hard, so if you have enough data …

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 is more general)

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

Convolution Neural Network (CNN)

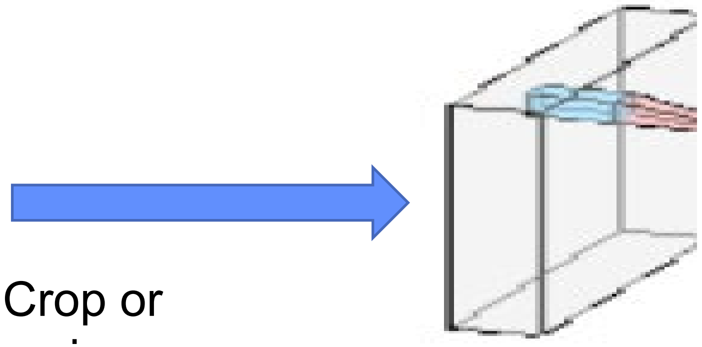
More hyperparameters:

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

Convolution with image

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

Crop or resize image as needed

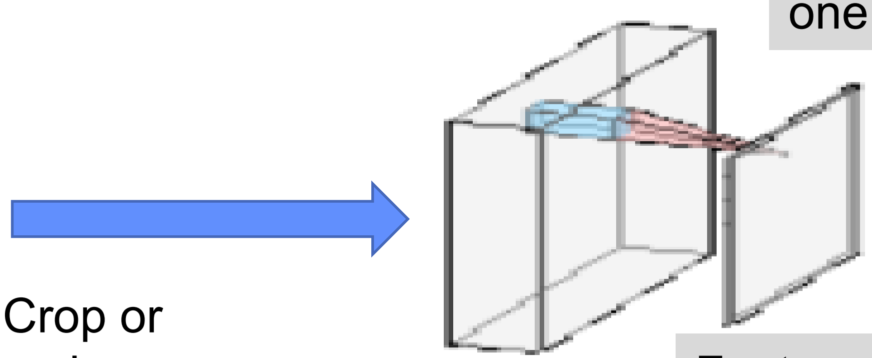


224x224x3

Convolution with image

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

Crop or resize image as needed



224x224x3

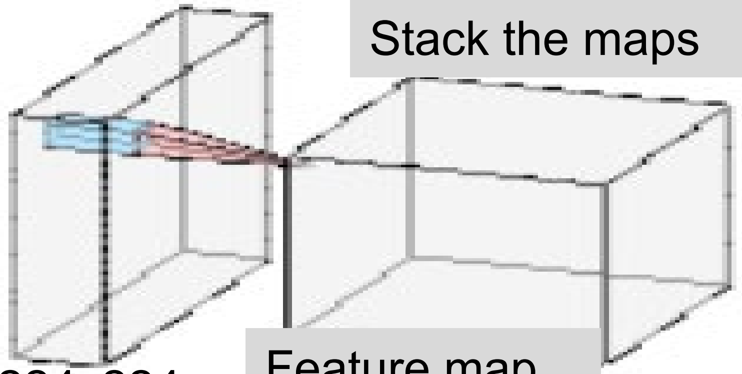
one 5x5x3 filter

Feature map Size depends on filter size and slide parameter

Convolution with image

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

Crop or resize image as needed



224x224x3

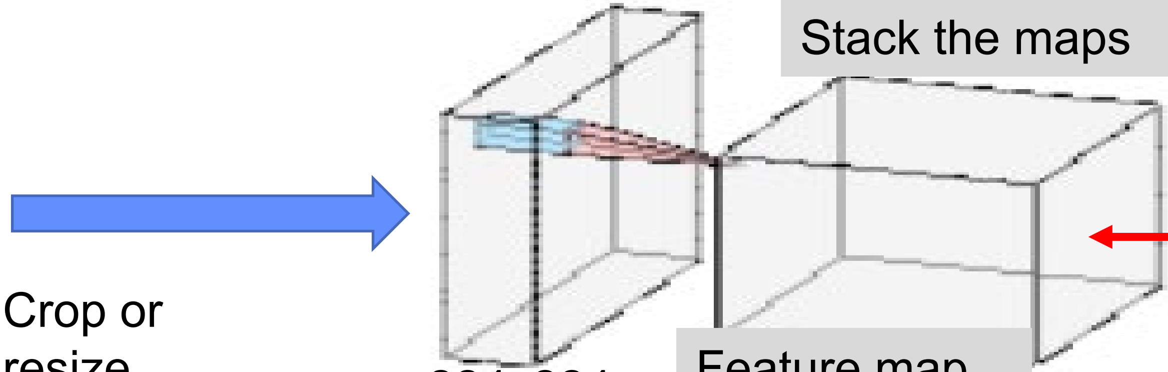
Repeat for many filters, all 5x5x3.
Each filter leads to 1 feature map
Stack the maps

Feature map
Size depends on filter size and slide parameter

Convolution with image

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

Crop or resize image as needed



Repeat for many filters, all 5x5x3.
Each filter leads to 1 feature map
Stack the maps

224x224x3

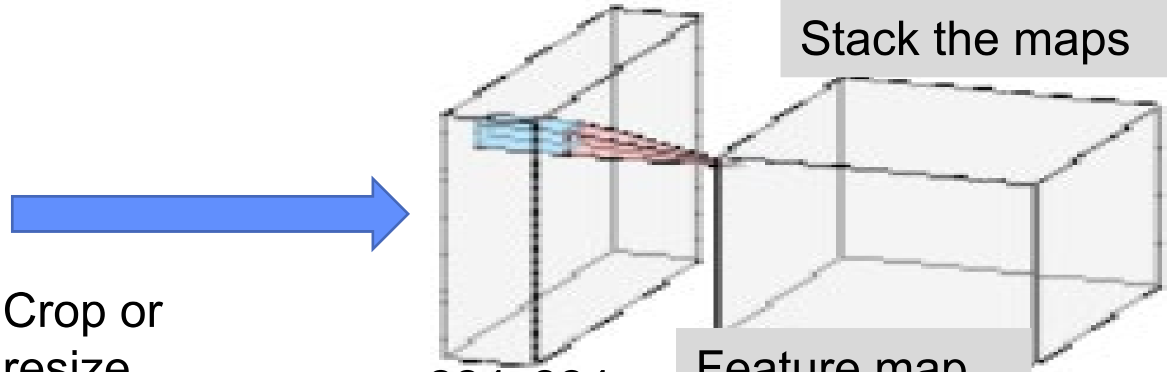
Feature map
Size depends on filter size and slide parameter

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

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 5x5x3.
Each filter leads to 1 feature map
Stack the maps

Feature map
Size depends on filter size and slide parameter

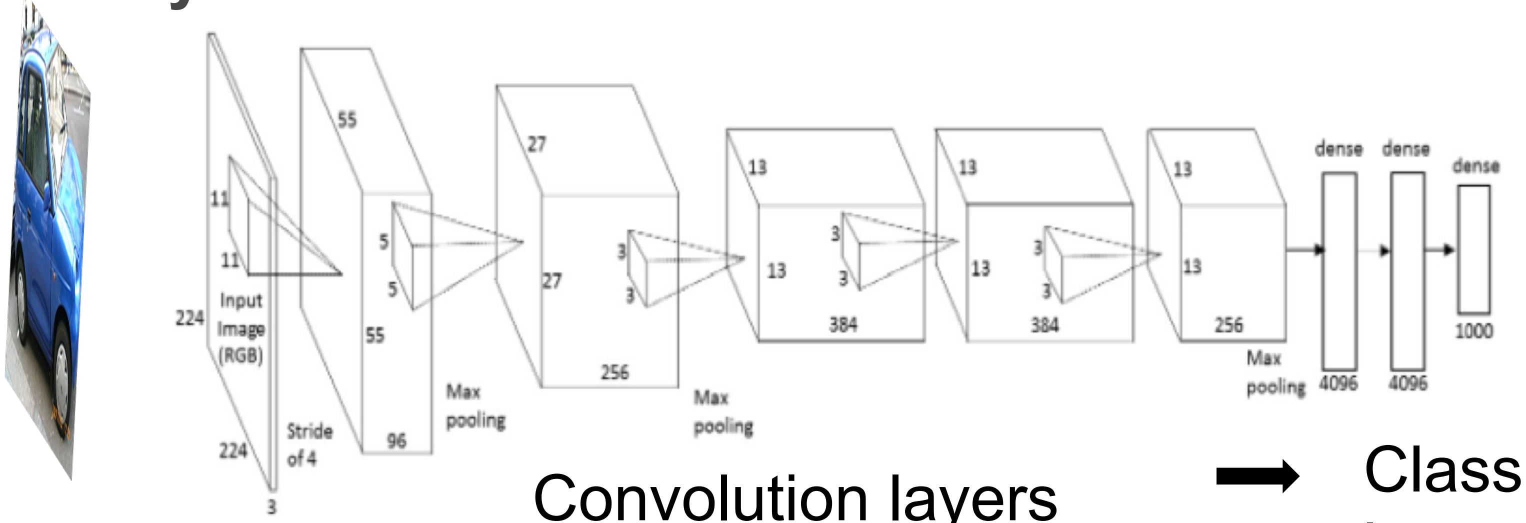
Add a layer of max-pool to down sample each feature map

OR

Add a next convolution layer

Large Scale Versions

- Large (deep) Convolution Networks are turning out to be feasible with GPUs (some are 100+ layers)
- Need large amounts of data and many heuristics to avoid overfitting and increase efficiency

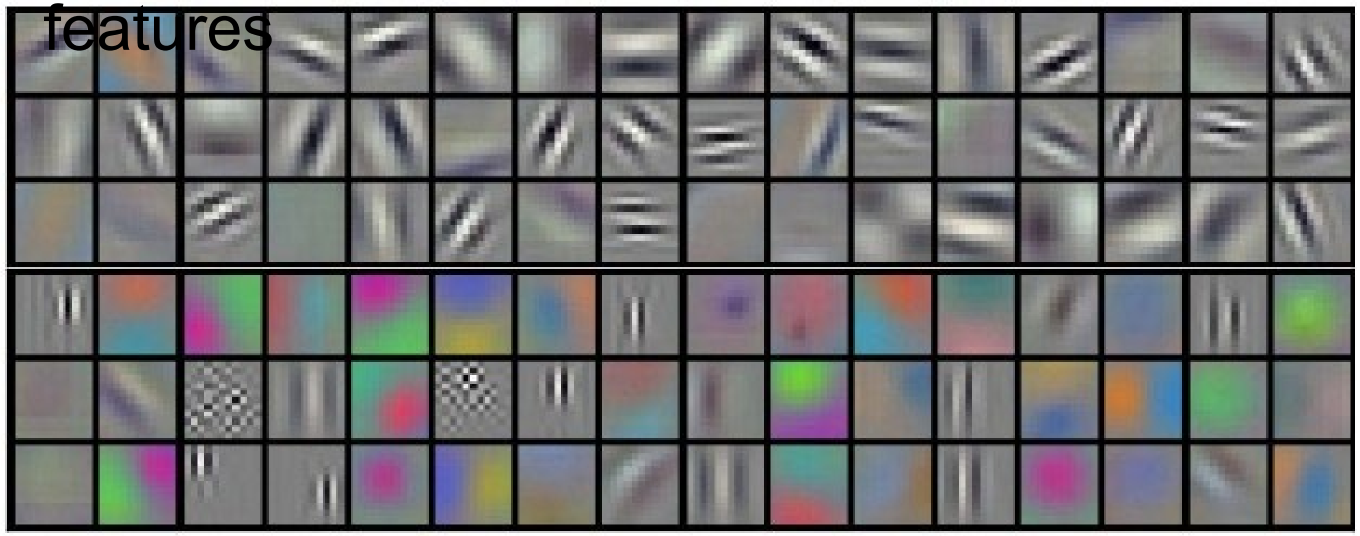


Convolution layers

➡ Classification layers and output

What Learned Convolutions Look Like

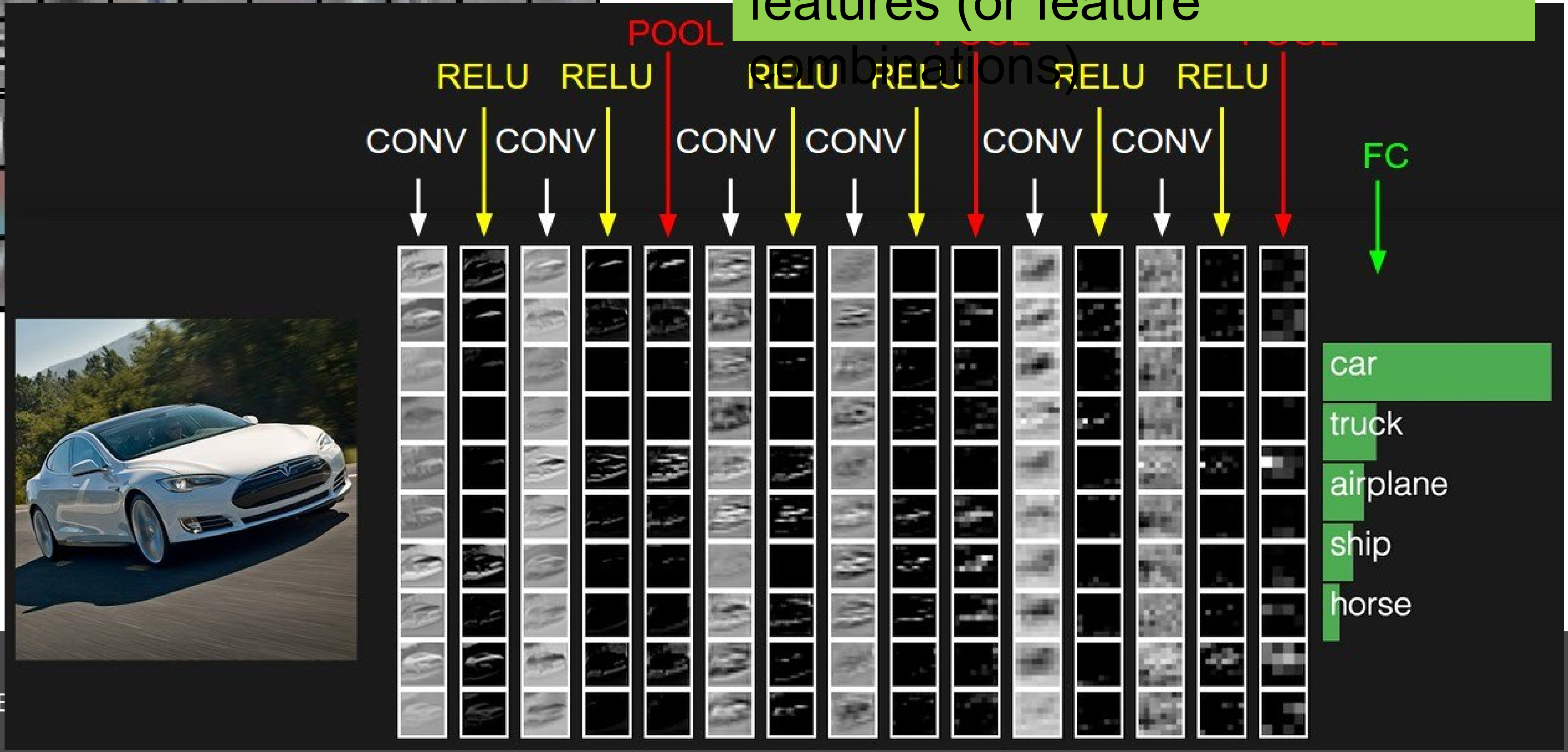
First convolution layer filters are simple features



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

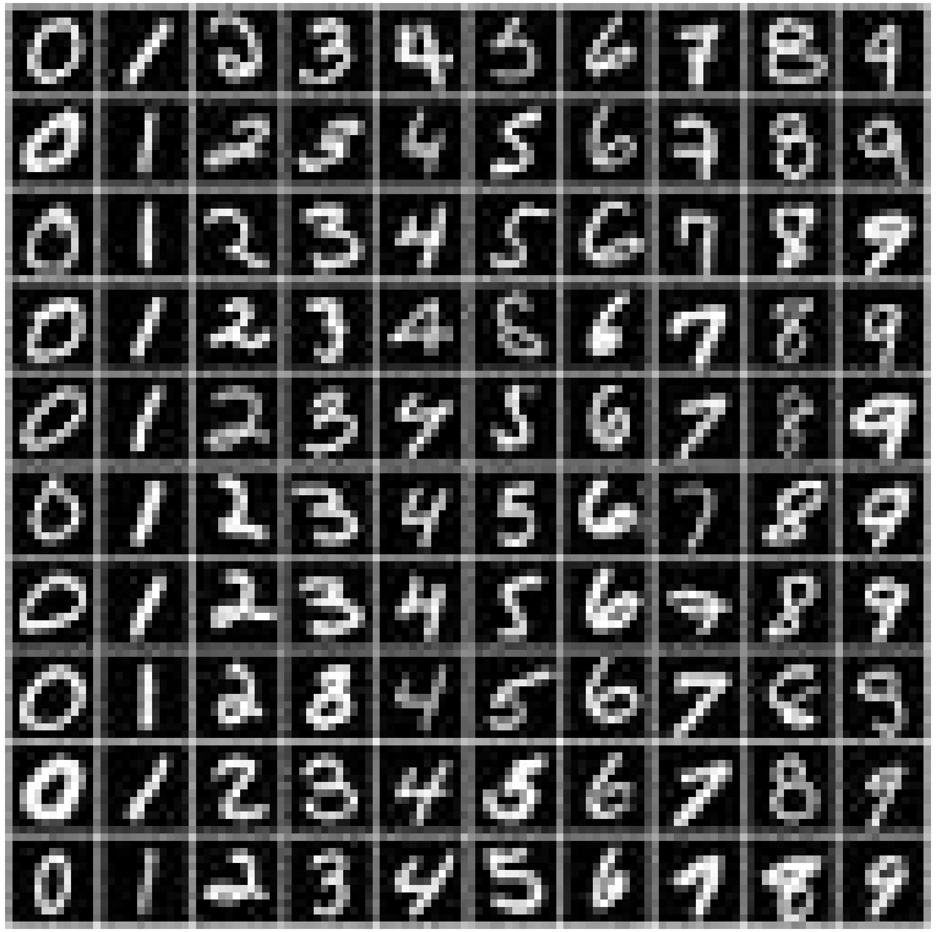
Low layers have less filters that represent simple local features for all classes

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

- **pause**

Demo CNN for Digit Classification

- The 'hello world' of CNNs
- Uses MNIST dataset and Keras



intro_mnist_mlp2 Last Checkpoint: a few seconds ago (autosaved)

View Insert Cell Kernel Widgets Help

Run Code

```
import warnings
warnings.filterwarnings("ignore")
import tensorflow as tf
tf.get_logger().setLevel('ERROR')

#load and prepare data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test                      = x_train / 255.0, x_test / 255.0

#specify the neural network model and optimization
my_model   =  tf.keras.models.Sequential([
                       tf.keras.layers.Flatten(input_shape=(28, 28)),
                       tf.keras.layers.Dense(10)   ])
loss       =  tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer  =  tf.keras.optimizers.SGD(learning_rate=0.01)
my_model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

#fit the model
fit_history= my_model.fit(x_train, y_train, epochs=5, batch_size=128)

#evaluate the fit
my_model.evaluate(x_test, y_test)
```

A basic workflow in 4 steps

load/prepare data

define a model

fit a model

test the model

Zooming in on keras.models statements

```
#specify the neural ne                              ation
my_model   =  tf.keras.models.Sequential([
                    tf.keras.layers.Flatten(input_shape=(28, 28)),
                    tf.keras.layers.Dense(10)  ])
```

Use a sequence of layers

Flatten image into vector

Create hidden layer that is fully connected to input

In a nutshell:
Many hyperparameter choices for defining the model depends on task
Many algorithm parameters for optimization depends on heuristics

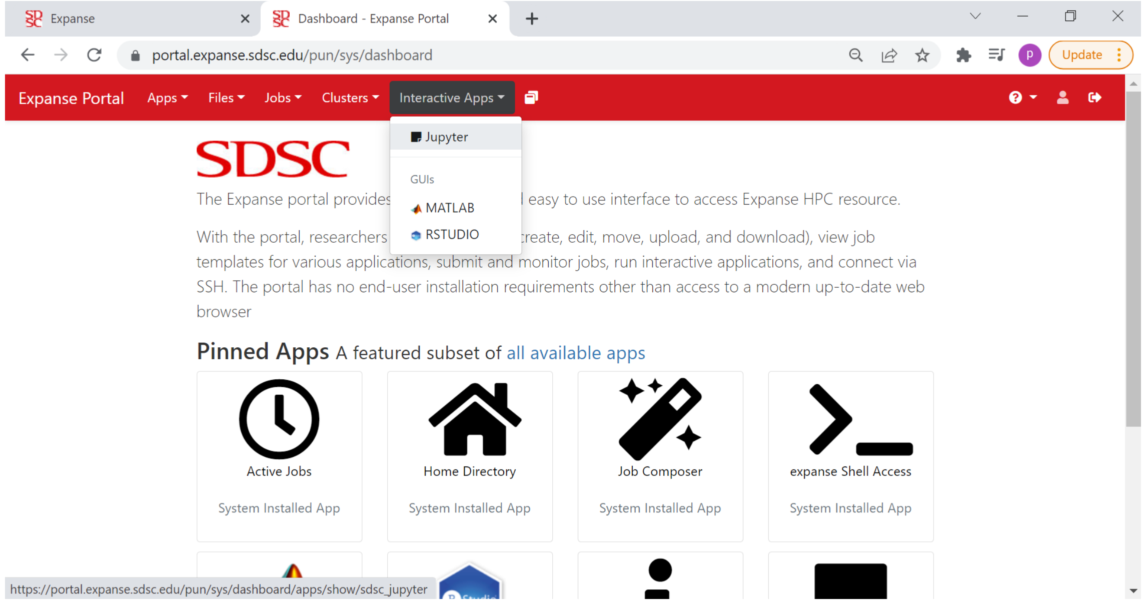
Zooming in on keras convolution layers statements

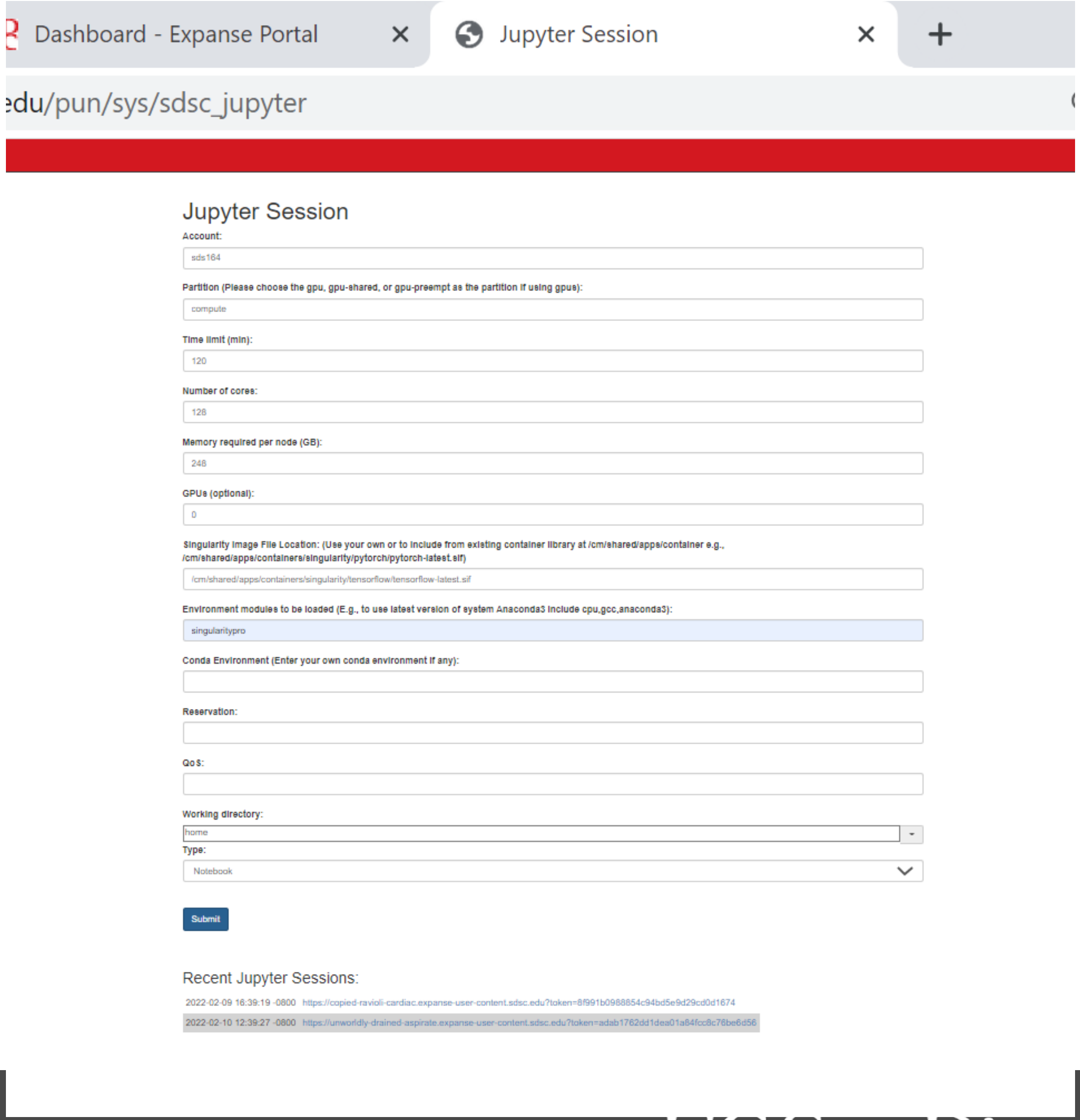
Use 16 filters, each of size 3x3

```
#   argument to Sequential
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

HPC portal can launch a jupyter notebook session





Demo

Things to think about

- On HPC, CPU work fine for many cases, you can will want to use GPUs for ‘large’ models and/or large datasets. Test with small datasets, and few epochs to evaluate
- Hyperparameter search is a bit of exploration, then focused trial and error – figure out work flow to save results and parameters together.
- Model saves and/or checkpoints are available in tensorflow; tensorboard available but needs to be secure (ask for details)

Things to think about

- **On HPC you may want to run batch jobs on a script not a notebook.**

You can use "*jupyter nbconvert --to script your-python.ipynb***" line command as part of your job and keep using the notebook**

And you would use these matplot imports and plt.savefig()

```
import matplotlib
matplotlib.use('Agg’)
import matplotlib.pyplot as plt
```

And you would use arguments or a configuration file to pass in parameters

Where to go from here

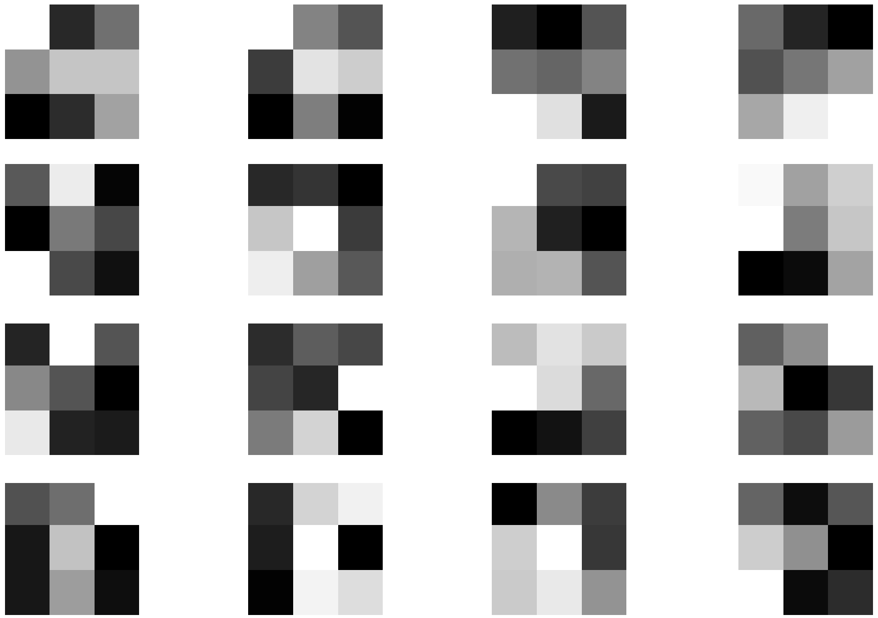
- Find relevant examples to your domain or task
- Tensorflow has many examples with tutorials in their documentation
- Tensorflow hub and model examples have code and pretrained models

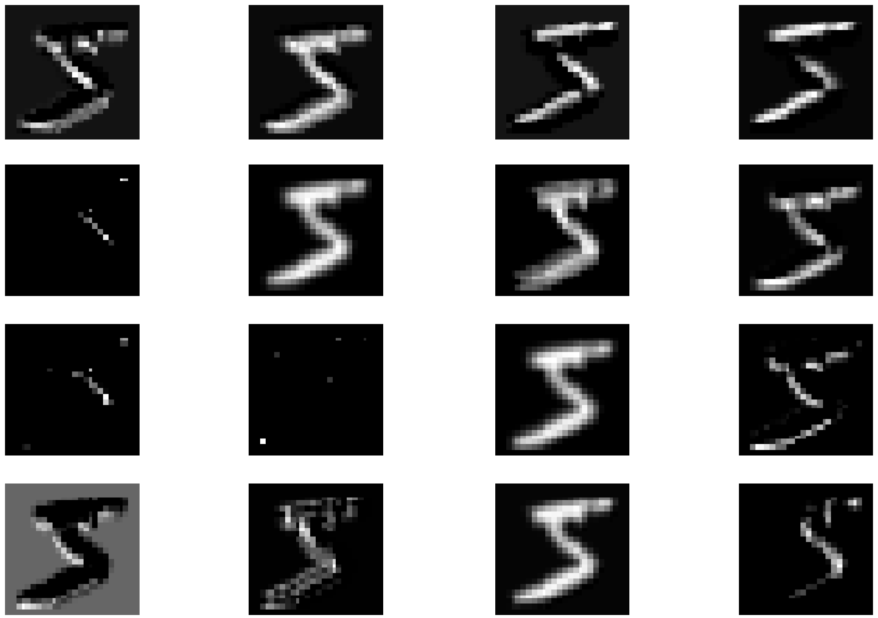
https://tfhub.dev/google/imagenet/inception_v1/classification/4

https://keras.io/examples/

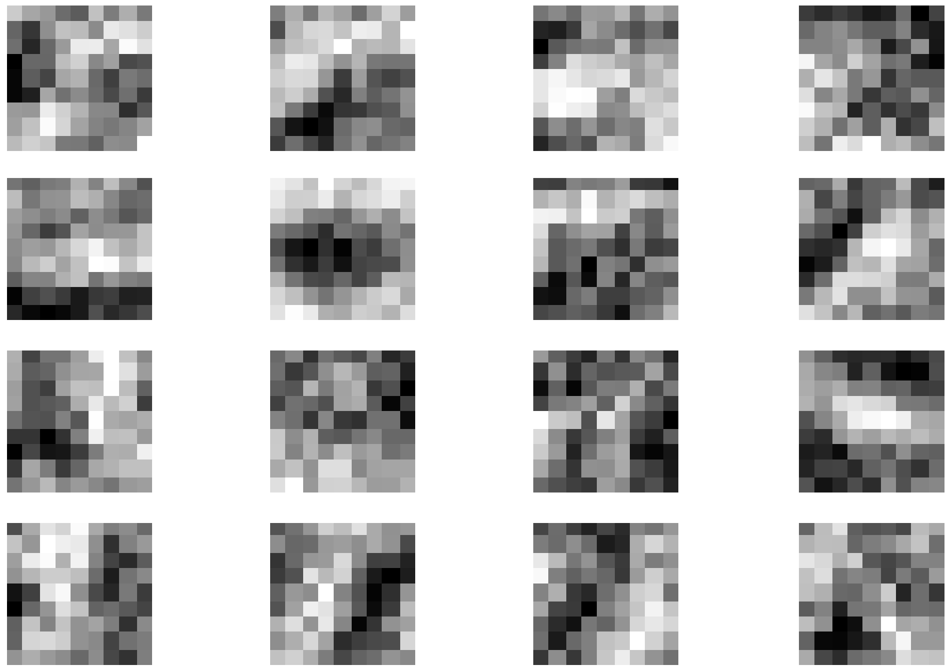
- **End**

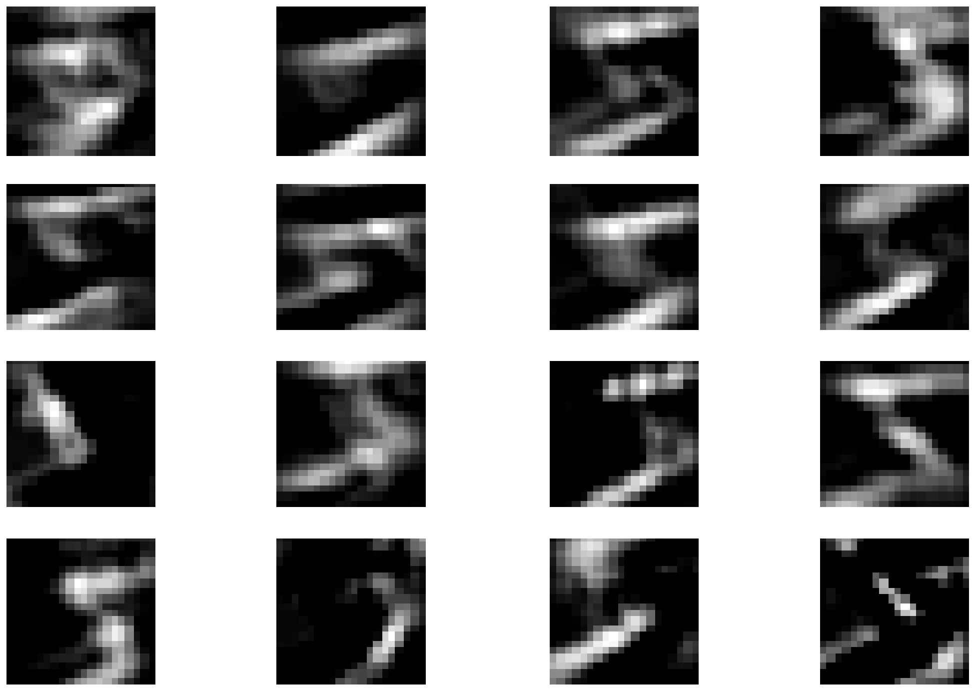
3x3 first convolution layer filter and activation





9x9 first convolution layer filter and activation





View Insert Cell Kernel Widgets Help

Run Code

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 26, 26, 16)        160
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 24, 24, 16)        2320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 16)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_6 (Dense)              (None, 32)                73760
_________________________________________________________________
dense_7 (Dense)              (None, 10)                330
=================================================================
Total params: 76,570
Trainable params: 76,570
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
```

Filter_wts X num_filters + filter_bias:
(3x3x1) *16 + 16*1 = 9*16+16=160