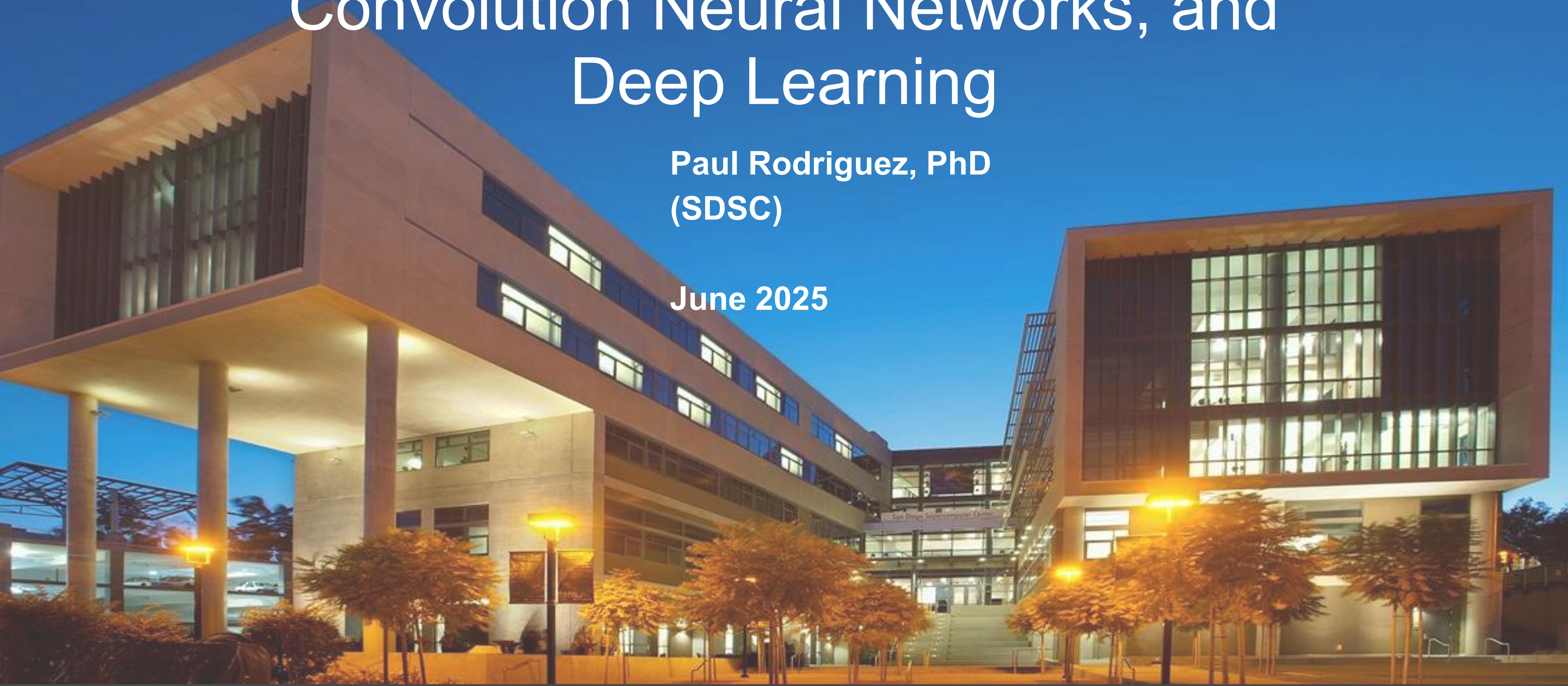
Introduction to Neural Network, Convolution Neural Networks, and Deep Learning

Paul Rodriguez, PhD
(SDSC)

June 2025



Agenda

- 8:30 - 10:00 am Intro to NNs and CNNs (including a PyTorch intro)
- 10:00 - 10:15 am Break
- 10:15 - 11:30 am Practical Guidelines for training on HPC (and multinode execution)
- 11:30 - 12:00 pm MLOps - MLflow and W&B
- 12:00 - 1:30 pm Lunch, group photo
- 1:30 - 2:15pm DL Layers & Architectures
- 2:15 - 3:45 pm DL Transfer Learning
- 3:45 - 4:00 pm Break
- 4:00 - 5:30 pm DL - Special Connections (and Transformers)

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

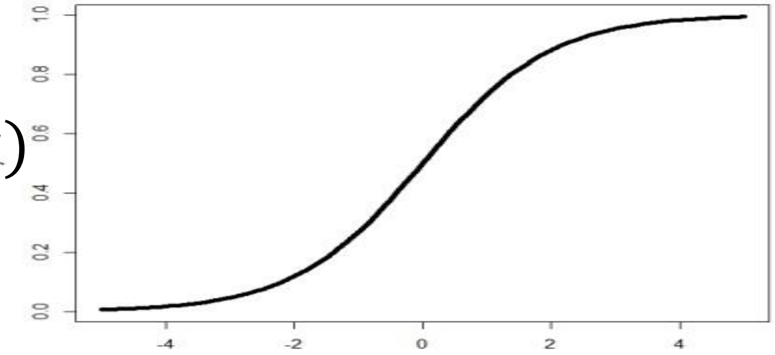
Exercise, Multinode MNIST

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: $b = 0 \;\;, \; w_1 = 1$

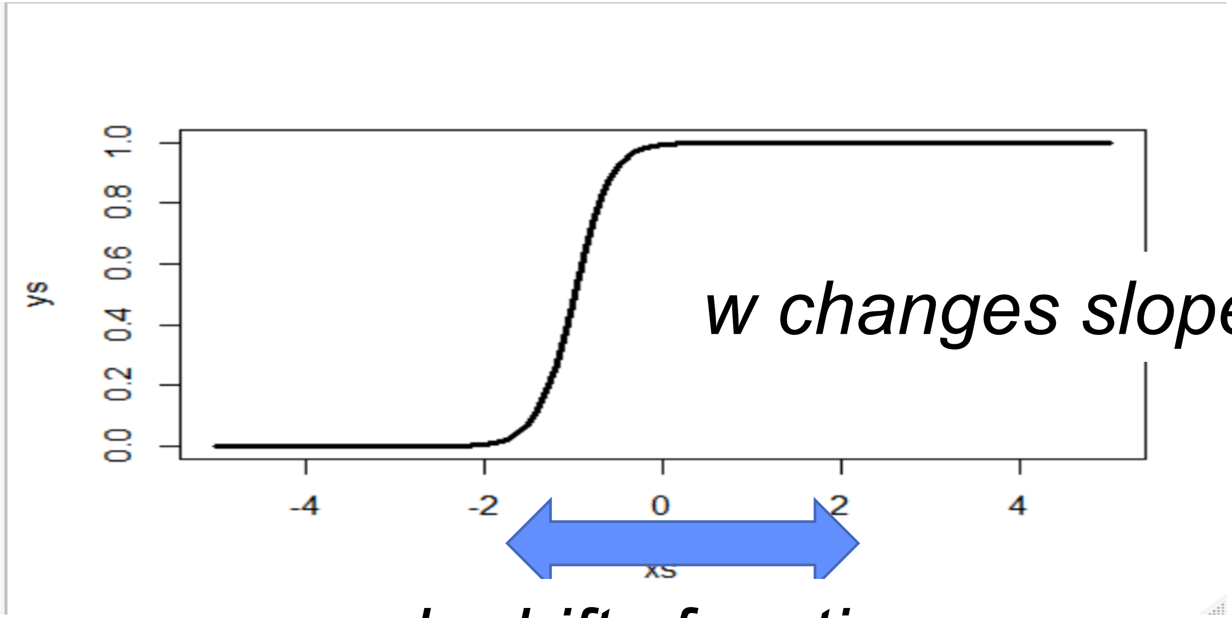
$y = f(x)$



x

Logistic Regression to Neural Network

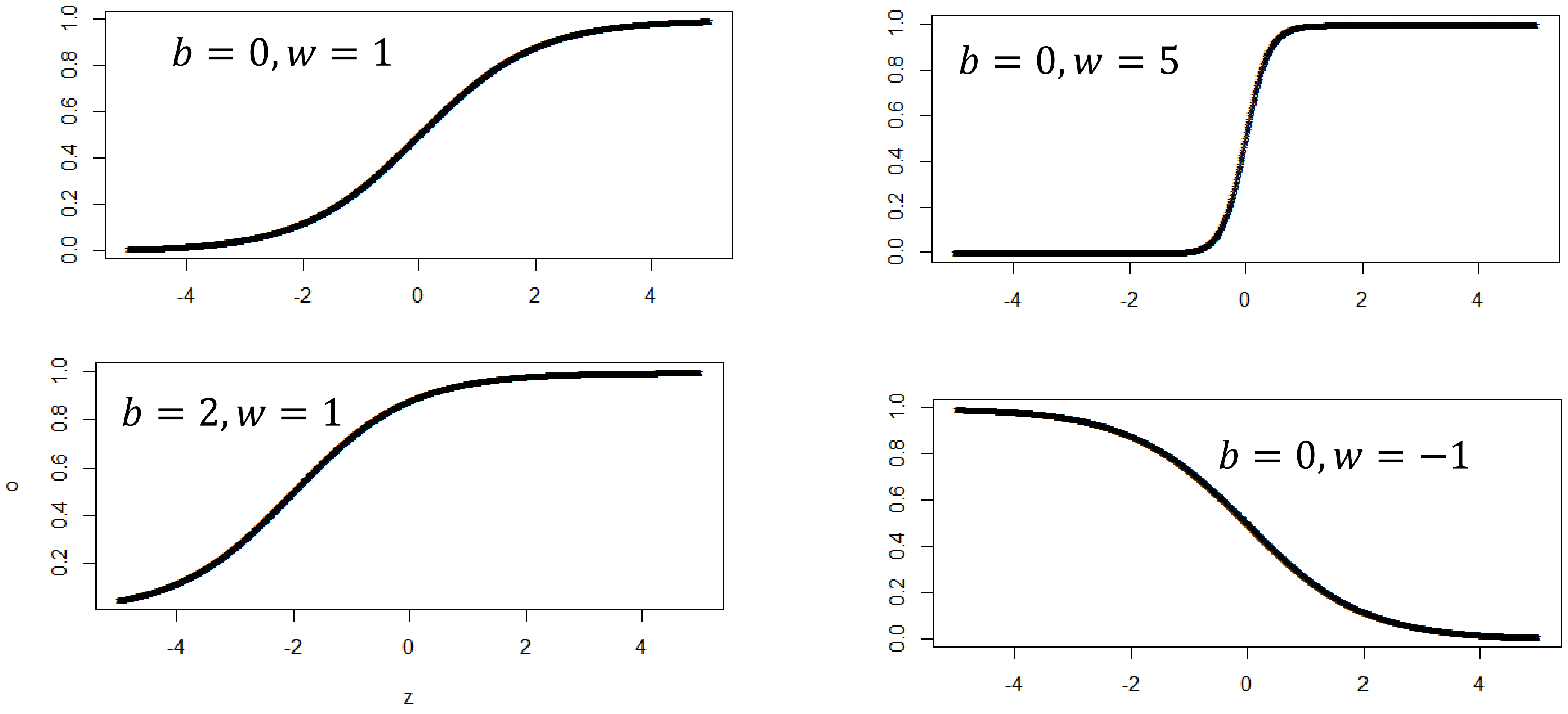
$$f(x, b, w) \quad = \quad \frac{\exp^{(b+w*x)}}{1+\exp^{(b+wx)}}$$



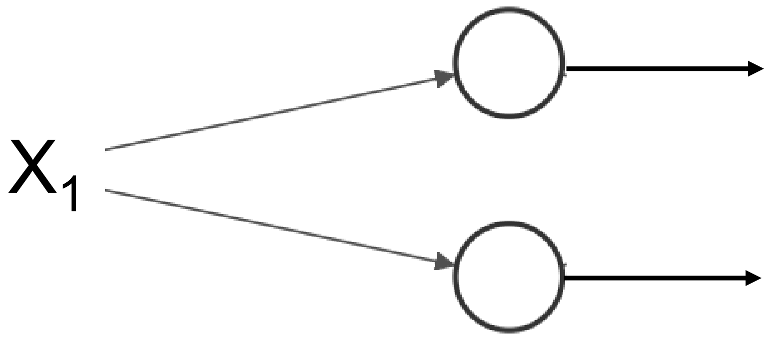
w changes slope

b shifts function

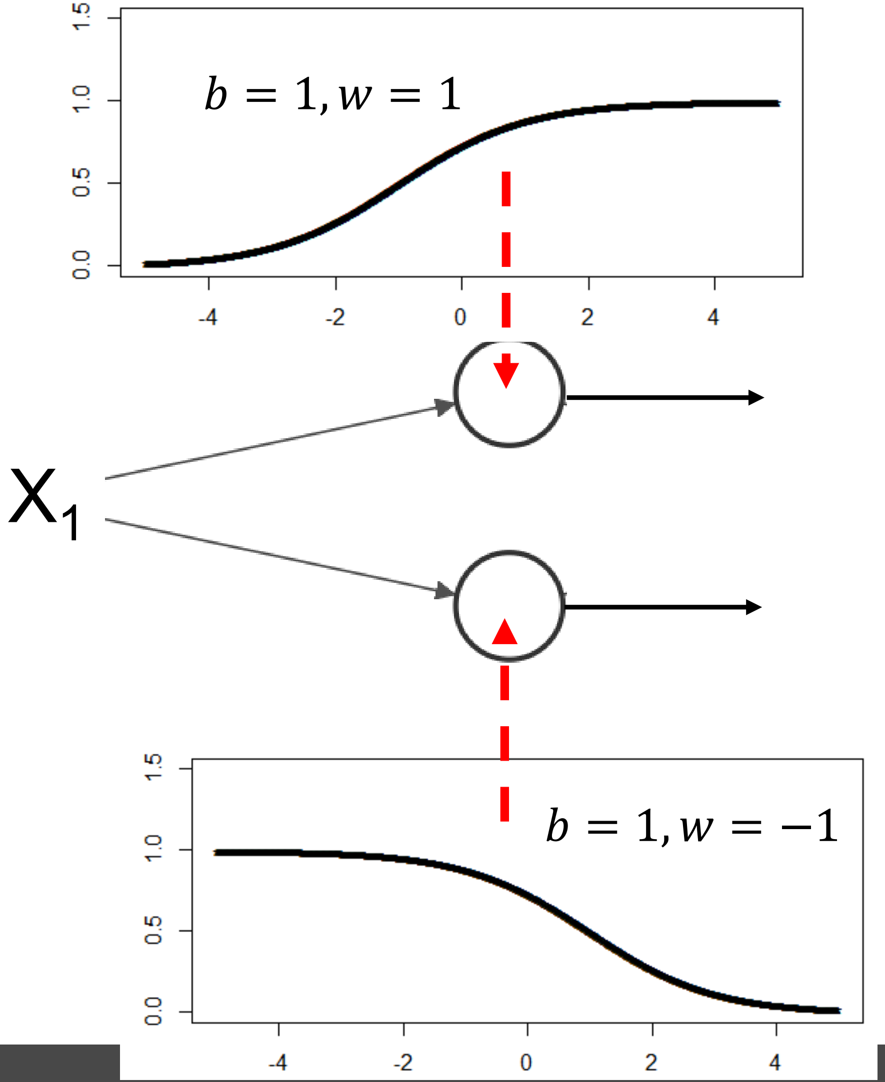
Logistic function w/various weights



Example: 1 input into 2 logistic units

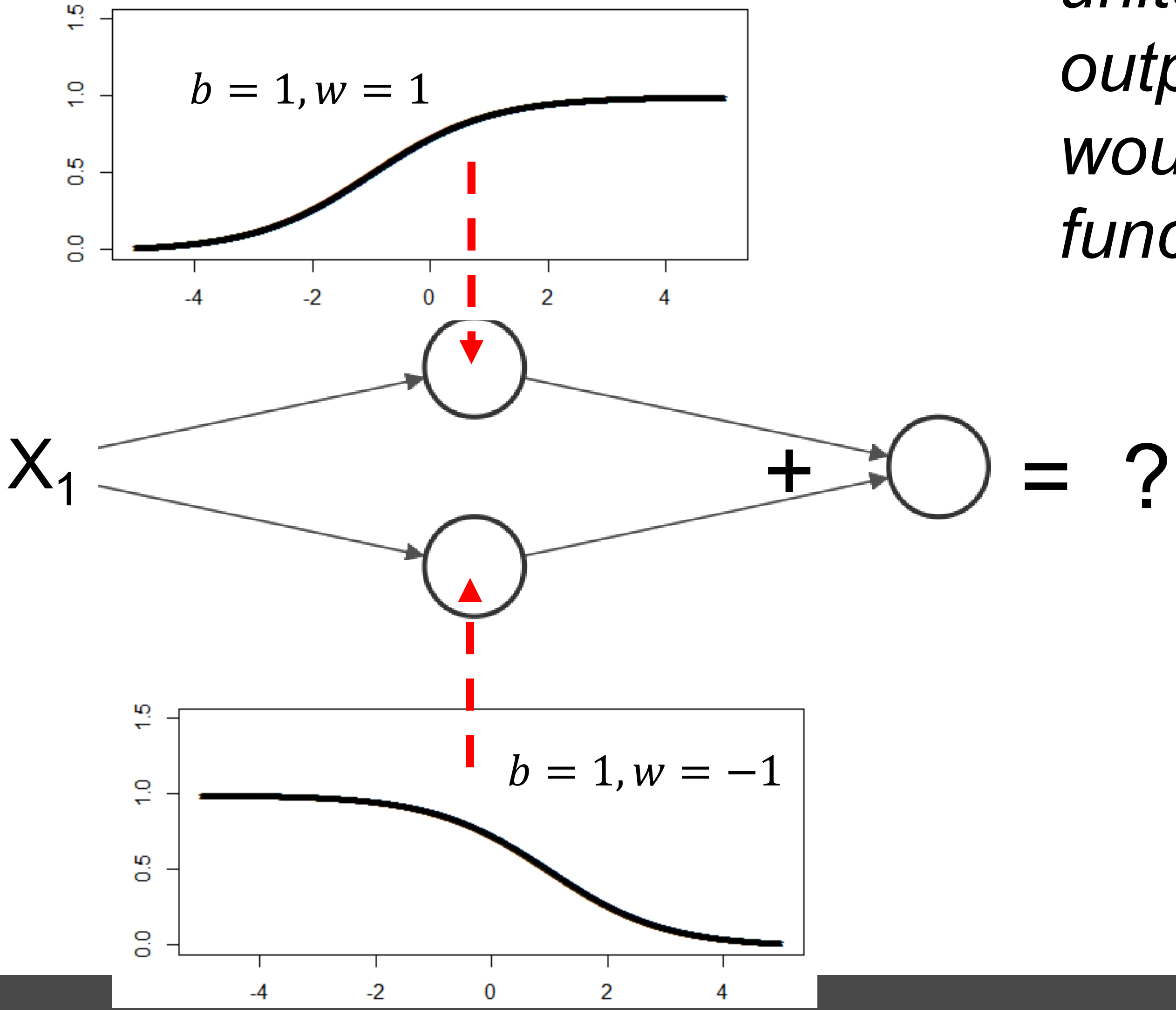


Example: 1 input into 2 logistic units with these activations



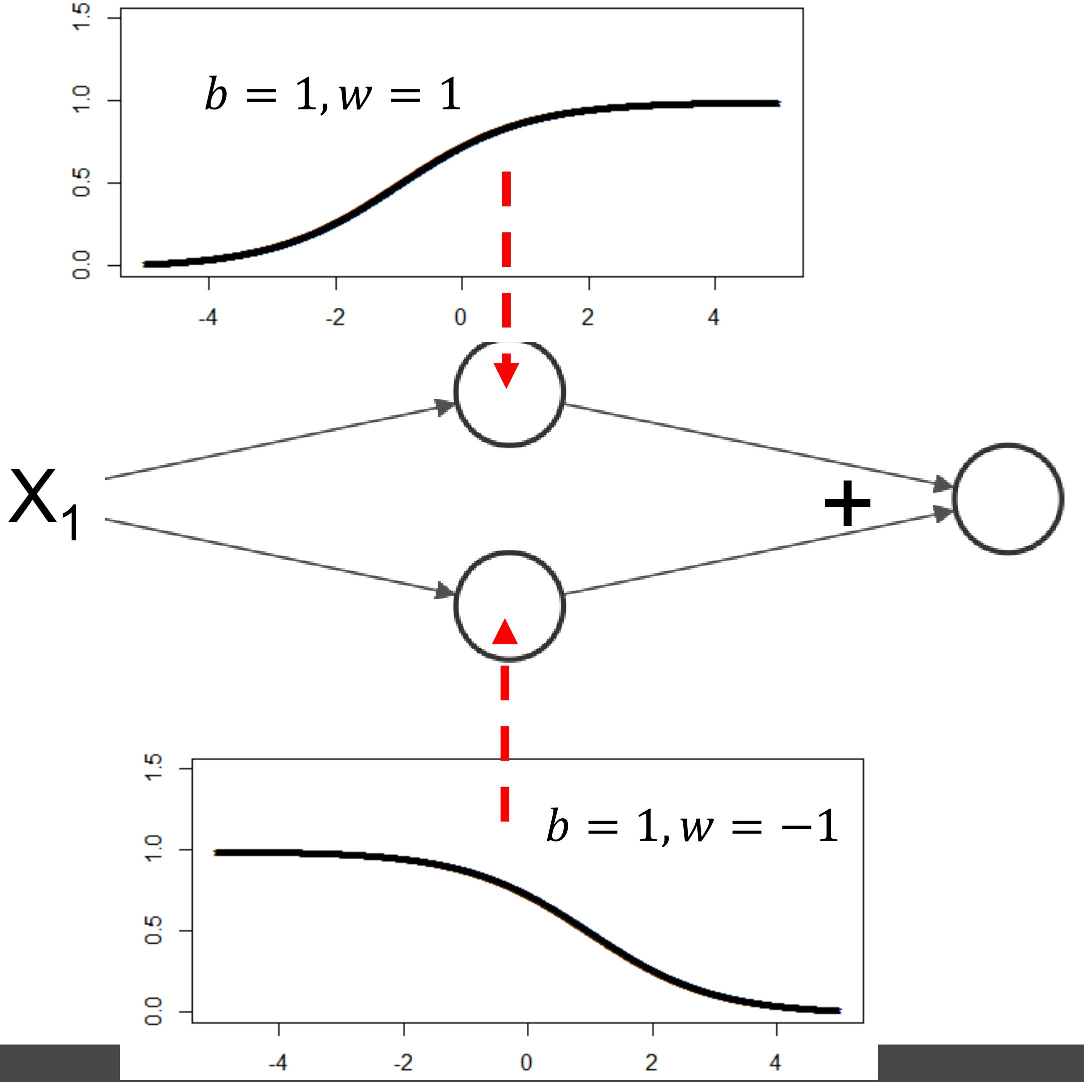
Example: 1 input into 2 logistic units with these activations

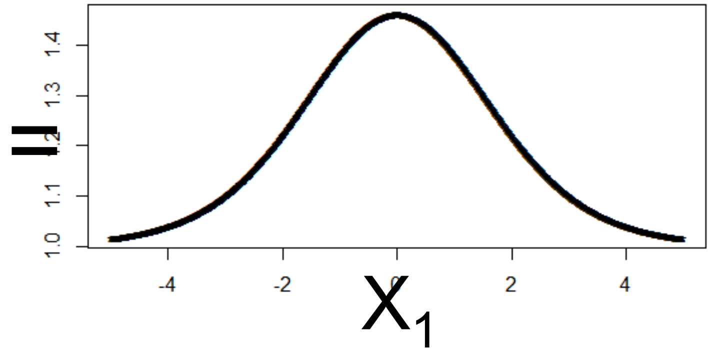
If you add these 2 units into a final output unit what would the output function look like?



Example: 1 input into 2 logistic units with these activations

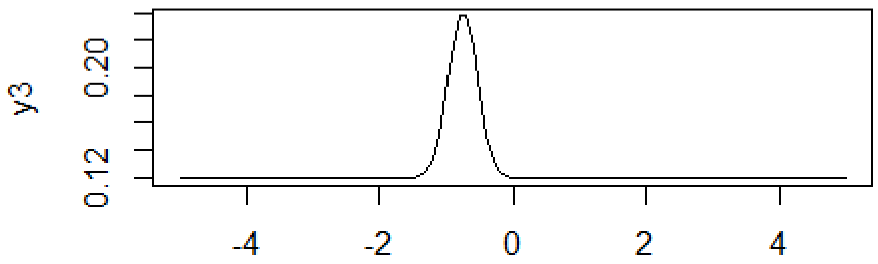
If you add these 2 units into a final output unit what would the output function look like?



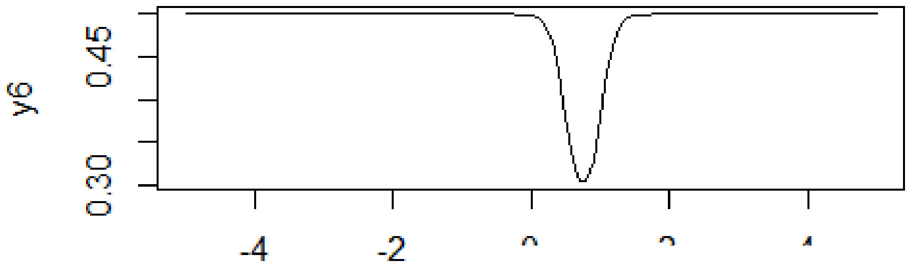


Higher level function combinations

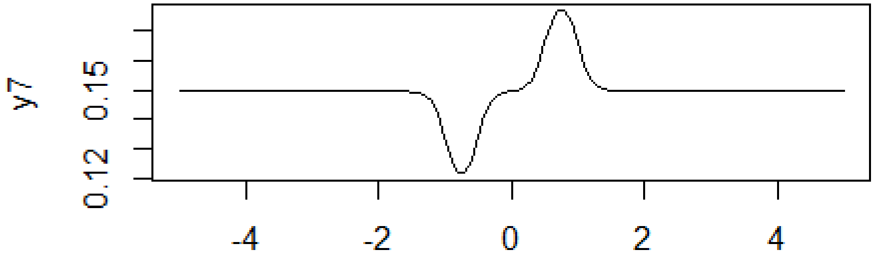
```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```




```
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")
```



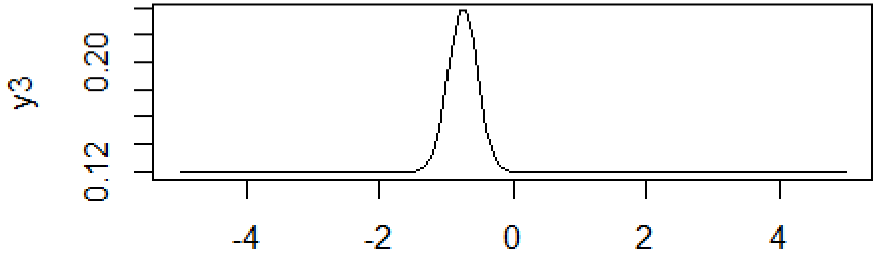

```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```



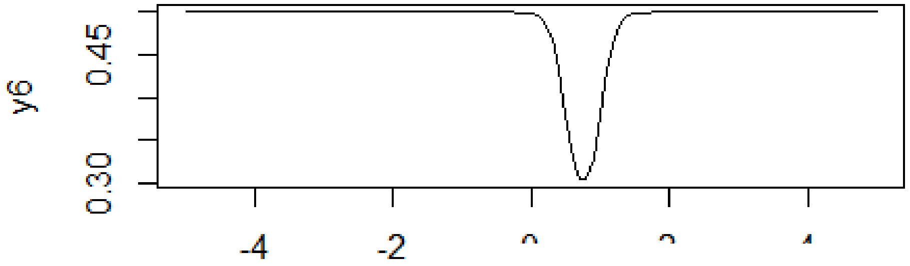
Higher level function combinations

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

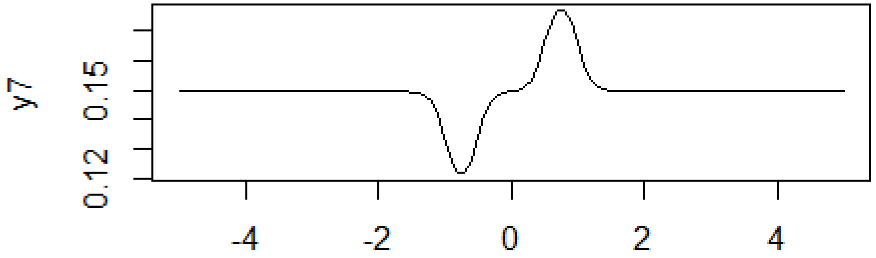
```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```




```
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")
```



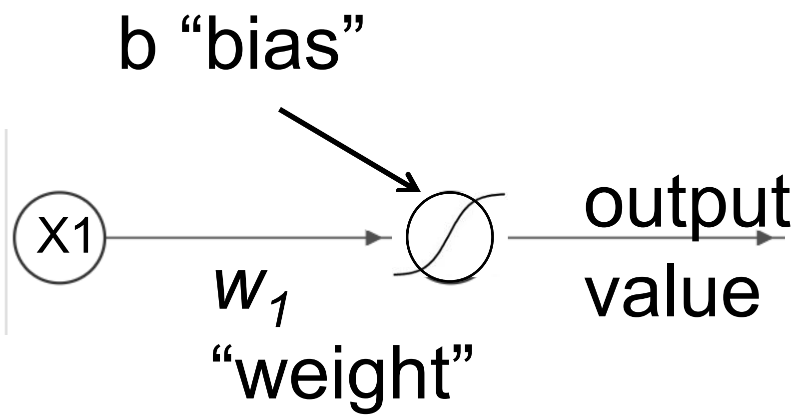
Logistic to Neural Network model

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

Draw out function as a little graph, 1 input

Logistic to Neural Network model

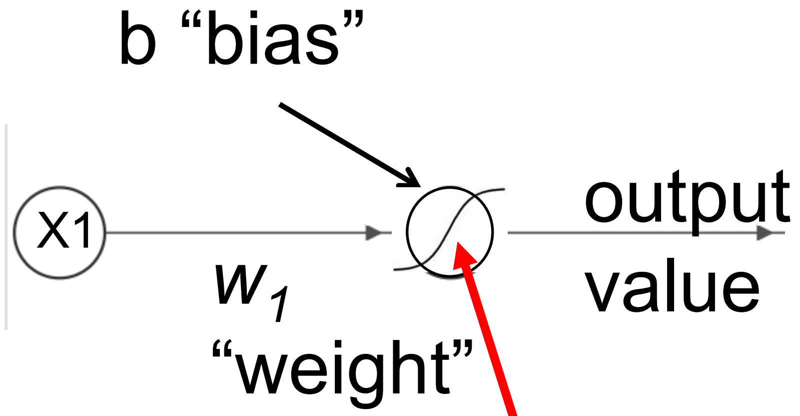
$$f(x, b, w) = \frac{\exp^{(b+w*x)}}{1+\exp^{(b+wx)}}$$



Draw out function as a little graph, 1 input

Logistic to Neural Network model

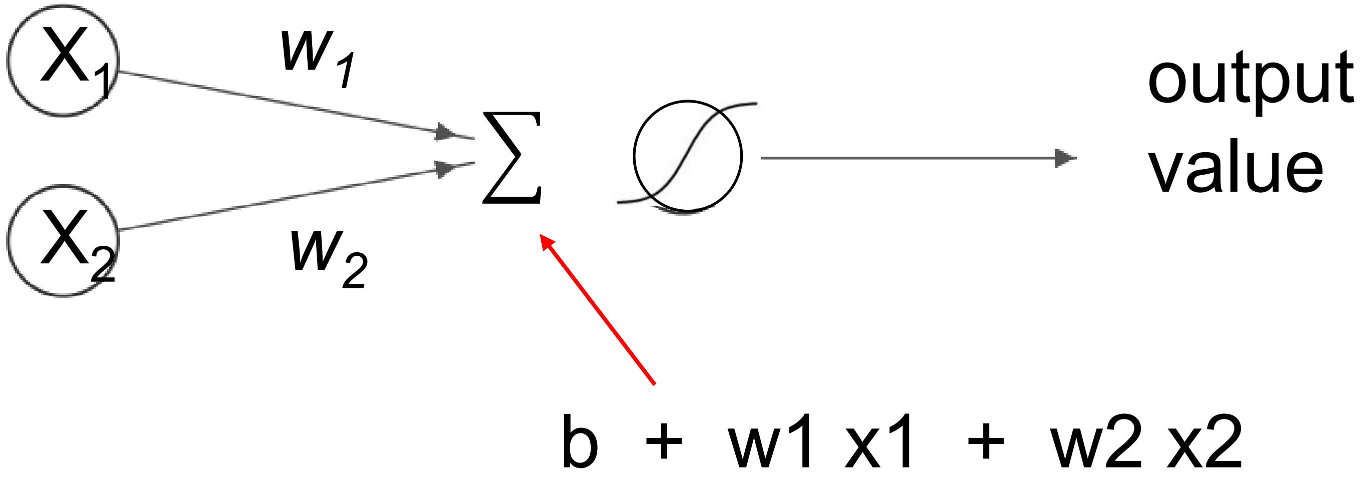
$$f(x,b,w) = \frac{\exp^{(b+w*x)}}{1+\exp^{(b+wx)}}$$



Draw out function as a little graph, 1 input

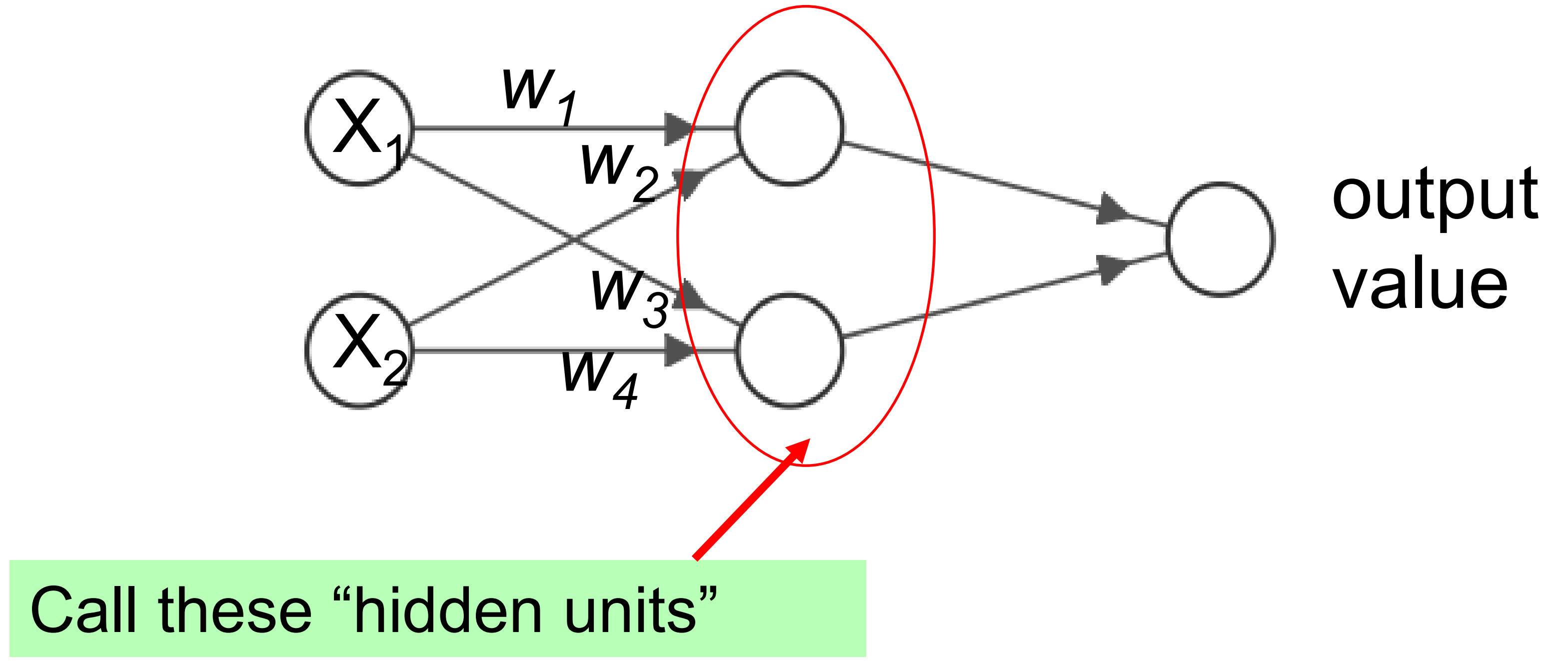
logistic function will transform input to output – call it the 'activation' function

Using 2 input units, the graph model would be:

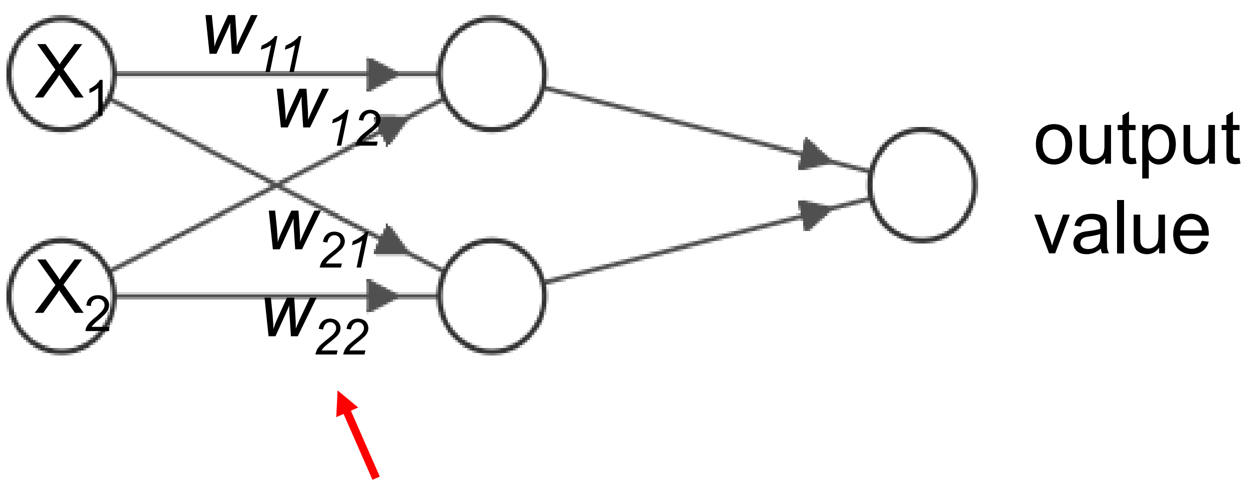


We usually don’t draw the bias.
We assume inputs*weights are summed (a dot product)

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

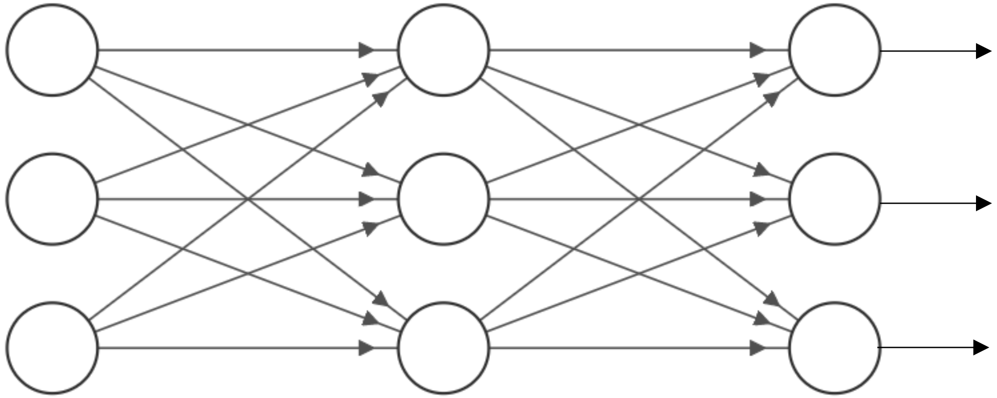


Using 2 input units, 2 intermediate units, and 1 output:



For X a Px1 vector, we set up a weight matrix W so that:
W*X = incoming activations

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

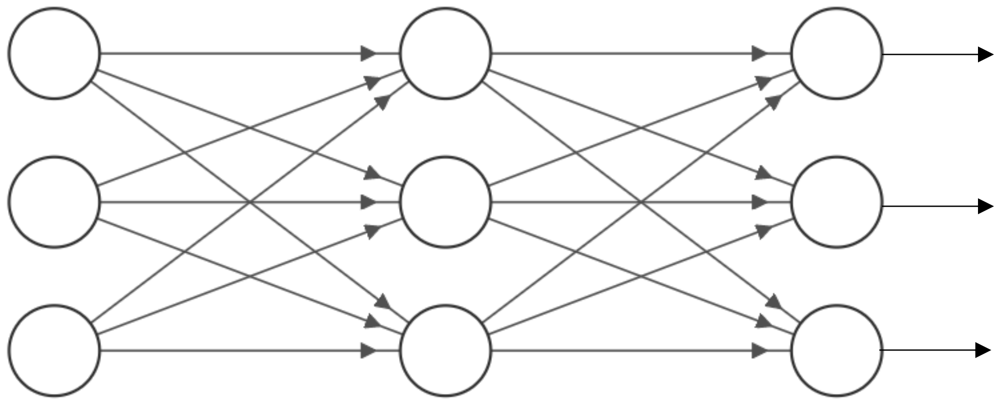


A “Multilayer Perceptron”

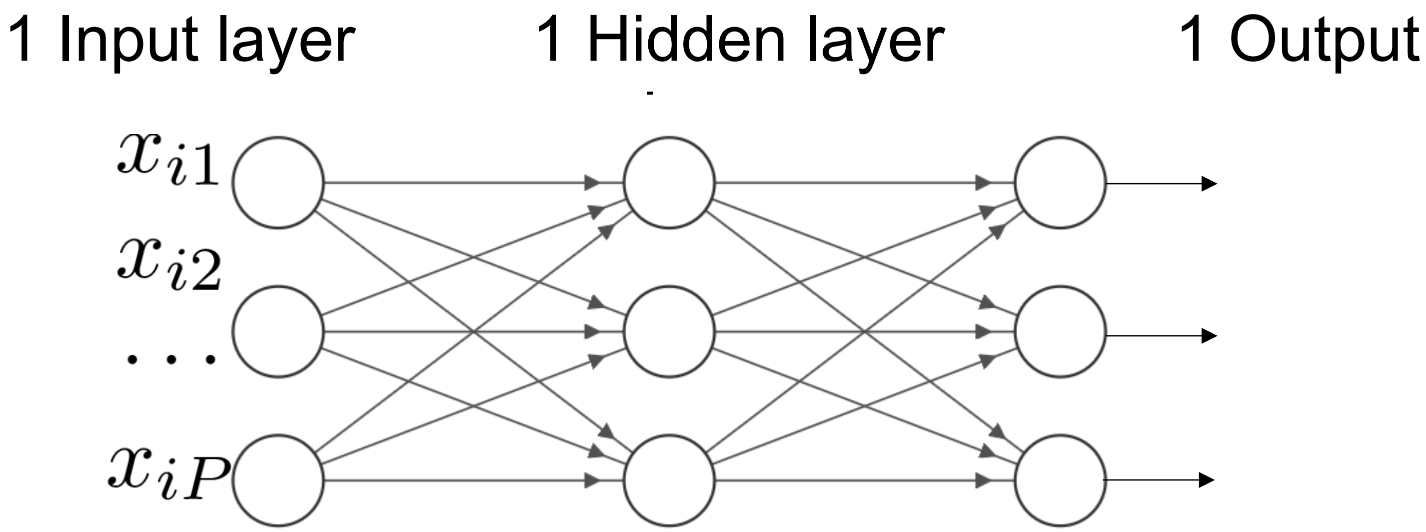
1 Input layer

1 Hidden layer

1 Output



A “Multilayer Perceptron”



P input units for each input i

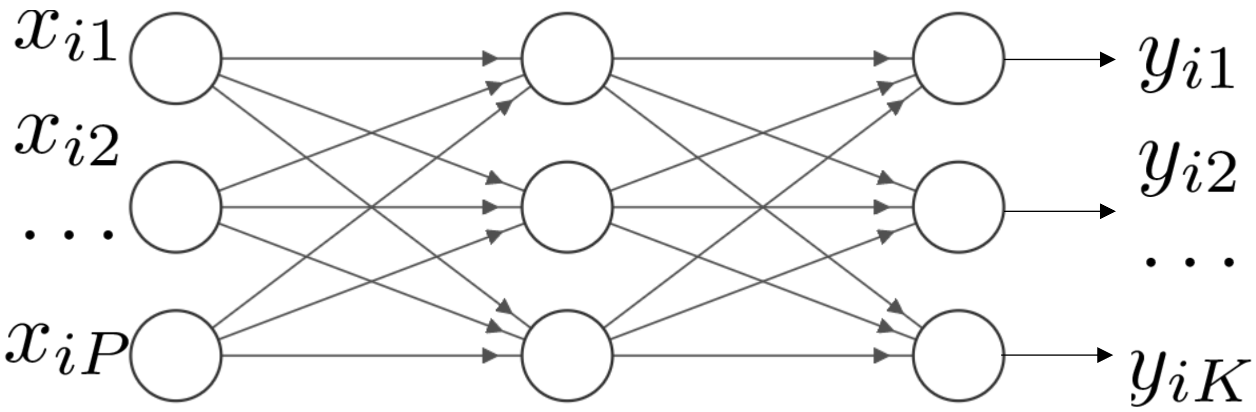
A "Multilayer Perceptron"

1 Input layer

1 Hidden layer

1 Output

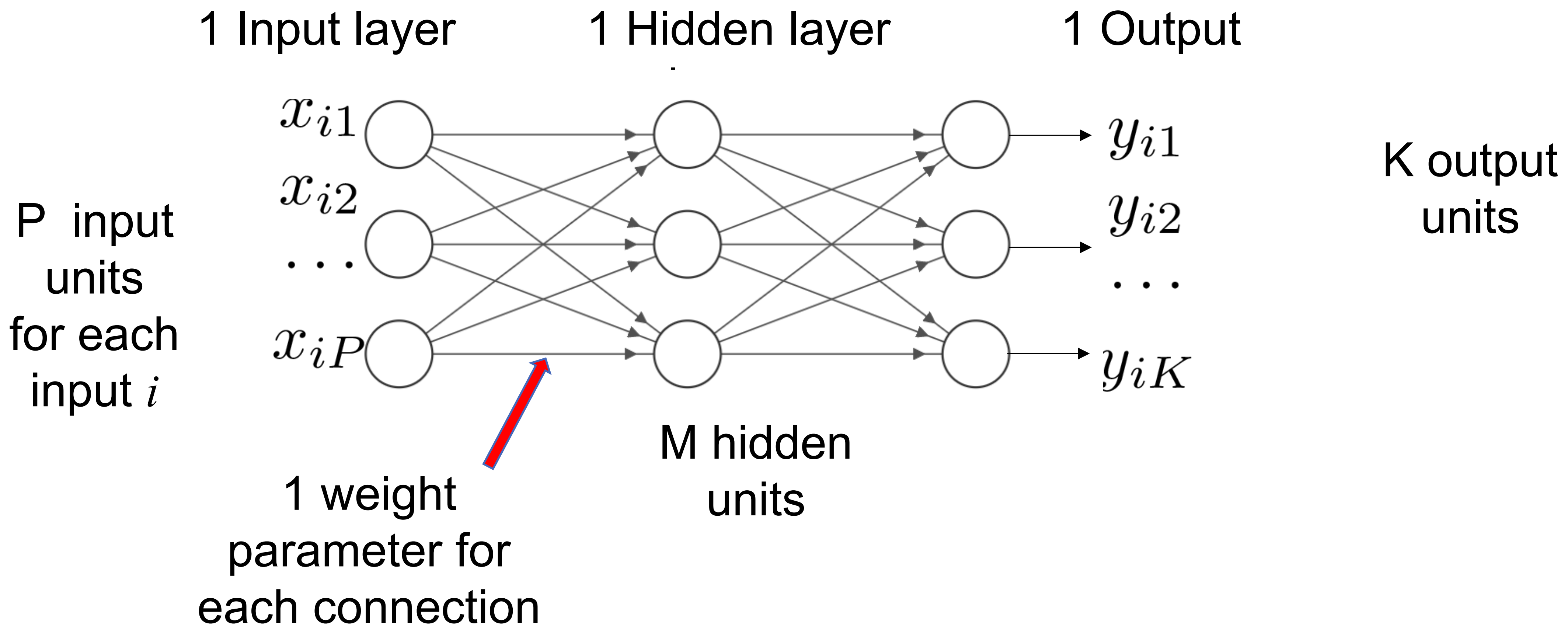
P input units for each input i



K output units

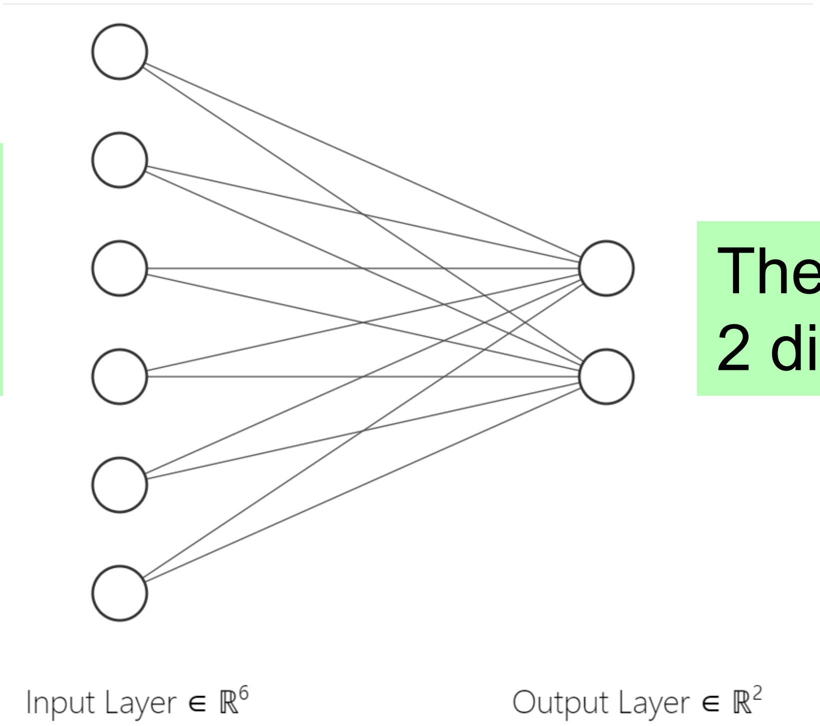
M hidden units

A “Multilayer Perceptron”



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

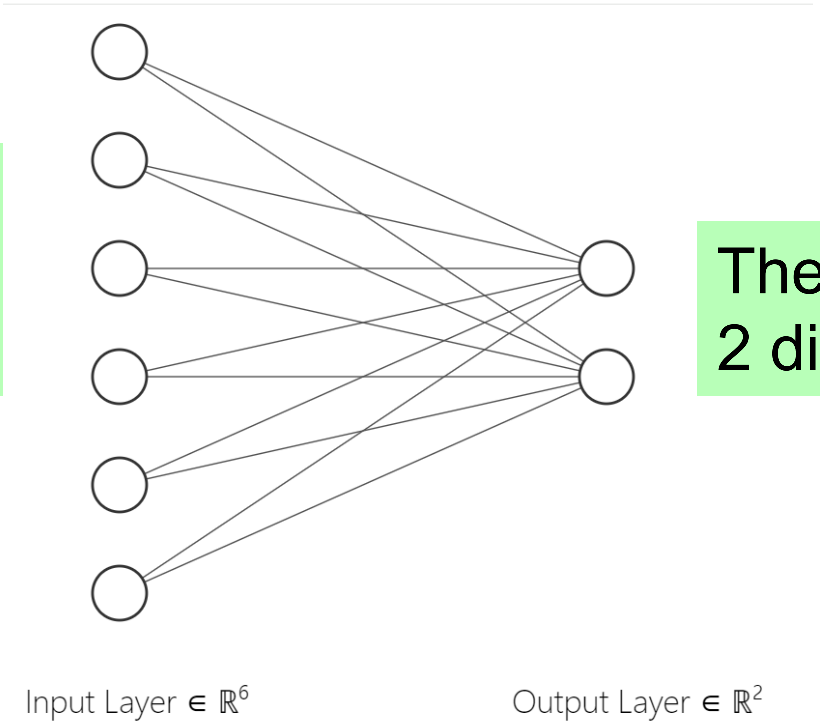
Here, the input vector has 6 values, so it’s 6 dimensions



The output vector has 2 dimensions

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

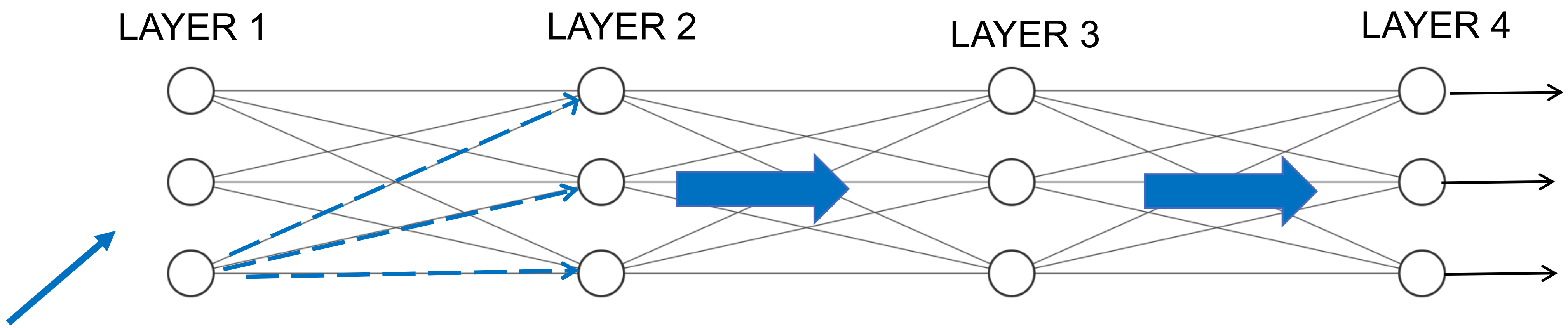
Here, the input vector has 6 values, so it’s 6 dimensions



The output vector has 2 dimensions.

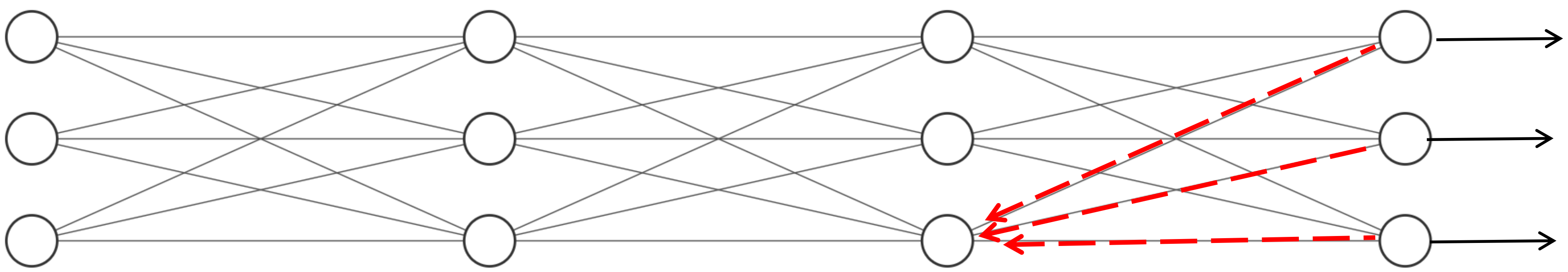
Learning the most relevant information usually means learning good embeddings!

Algorithm steps



1. FORWARD PROPAGATE AN ENTIRE BATCH OF INPUTS

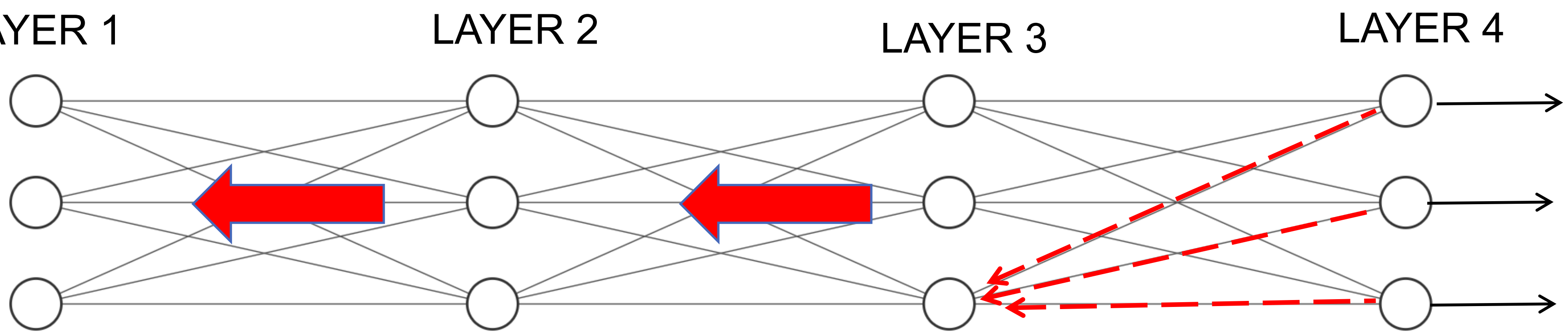
Algorithm steps



2. BACKWARD PROPAGATE ERROR FOR WHOLE BATCH 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

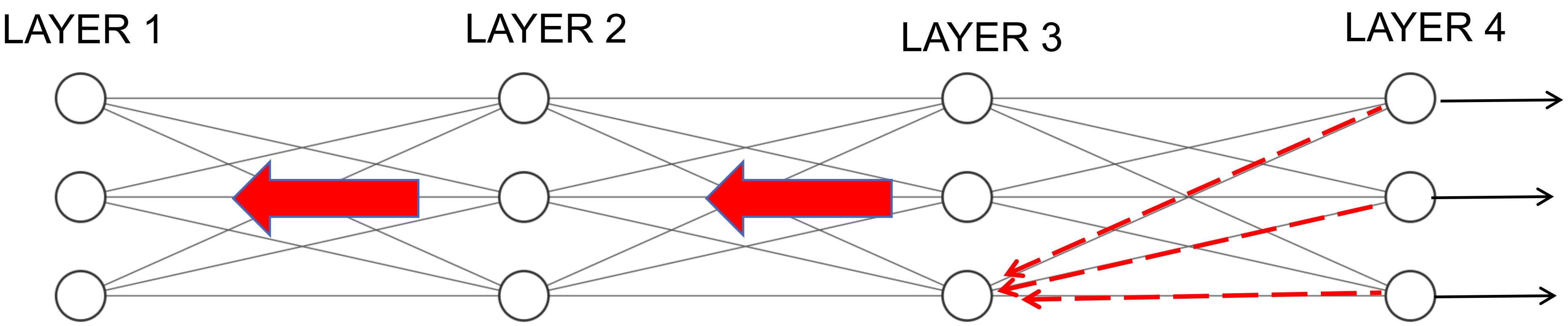


2. BACKWARD PROPAGATE ERROR FOR WHOLE BATCH USING DERIVATIVE CHAIN RULE:

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

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 FOR WHOLE BATCH USING DERIVATIVE CHAIN RULE:

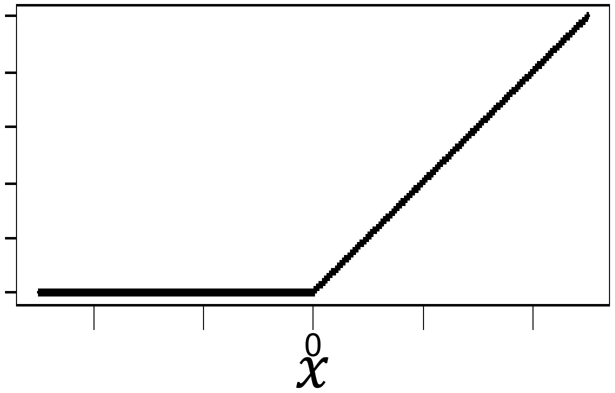
$$\frac{dE}{dw_{mp}} = \sum_{k}^{K} \frac{dE_k}{d\hat{y_k}} \frac{d\hat{y_k}}{da_k} \frac{da_k}{dh_m} \frac{dh_m}{da_m} \frac{da_m}{dw_{mp}}$$

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 (rectified linear unit)



RELU activation function

It is unscaled (bad!)

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

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

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

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: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \textit{\textbf{learning\_rate}} * \frac{dL}{dw}$

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: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \textit{\textbf{learning\_rate}} * \frac{dL}{dw}$

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

The Neural Network Algorithm [and heuristics]

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: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{learning\_rate} * \frac{dL}{dw}$

[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

- 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	if $Y \in (-\infty, +\infty)^K$	$\hat{Y} = XW$	$\hat{Y}$	Mean Squared Error (MSE)	Mean Squared Error (MSE)
Multivariate output of 0's and 1's	if $Y \in [0, 1]^K$	$\hat{Y} = \dfrac{1}{1 + exp^{-(XW)}}$	1 or 0	MSE	MSE
Binary Classification	if $Y \in \{0, 1\}$	$\hat{Y} = \dfrac{1}{1 + exp^{-(XW)}}$	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	if $Y \in \{0, 1\}^K$	$\hat{Y}_k = \dfrac{exp^{-(XW_k)}}{\sum_k exp^{-(XW_k)}}$	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’

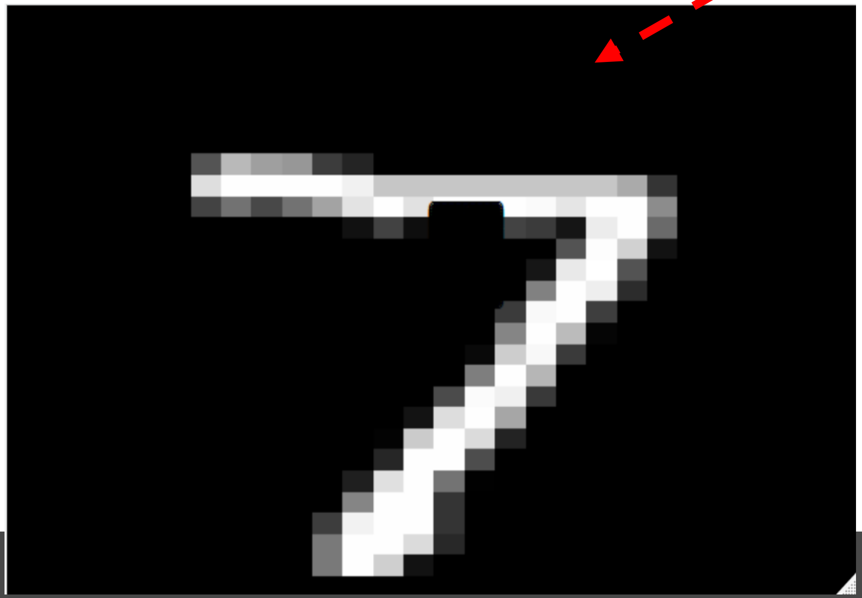
Next: Image classification of digits

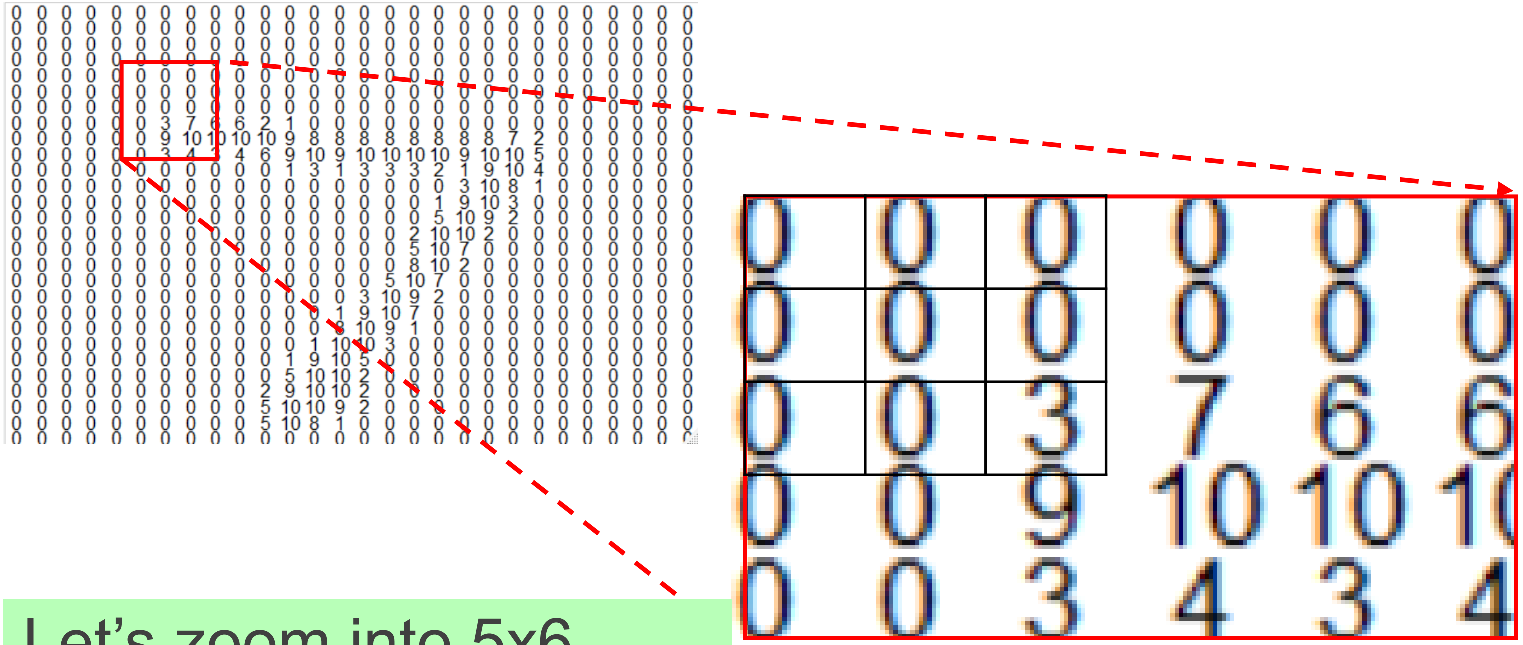
Image features

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

  (National Inst. of Standards and Technology)

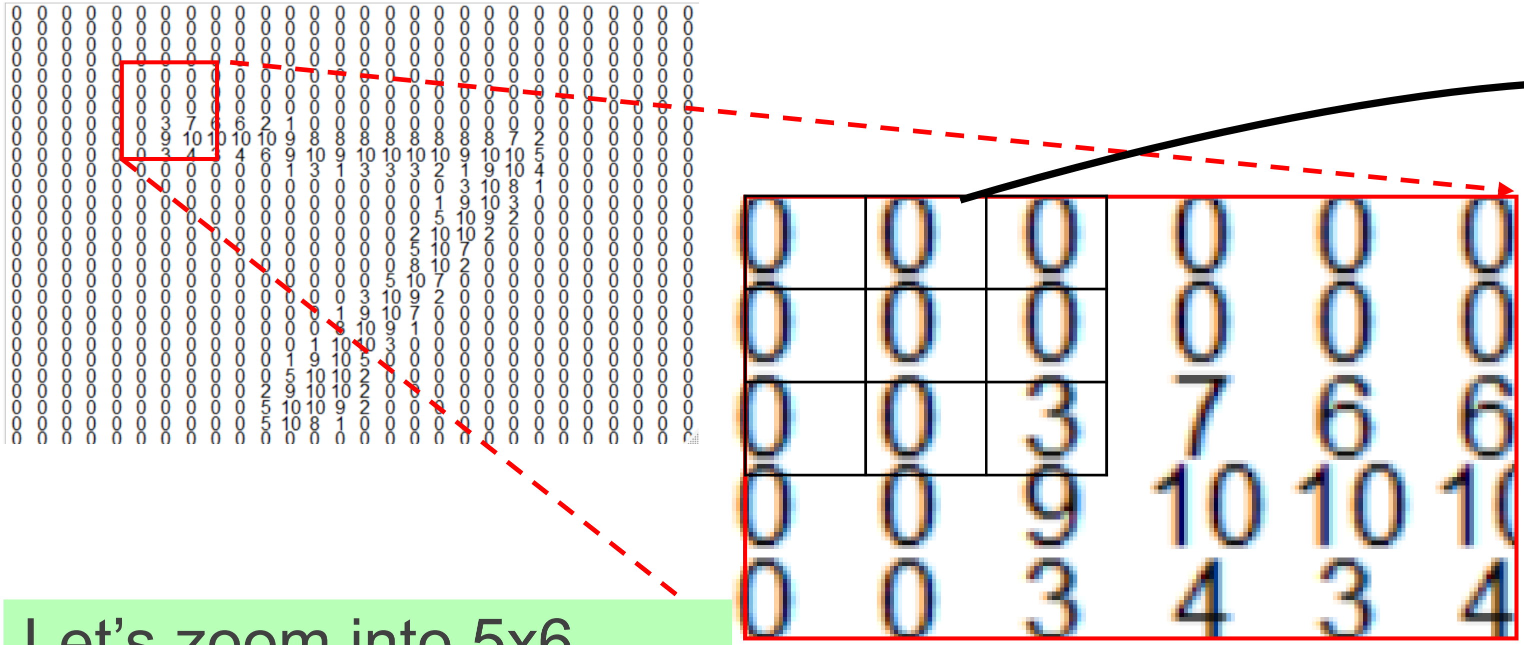
How to classify digits?





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’

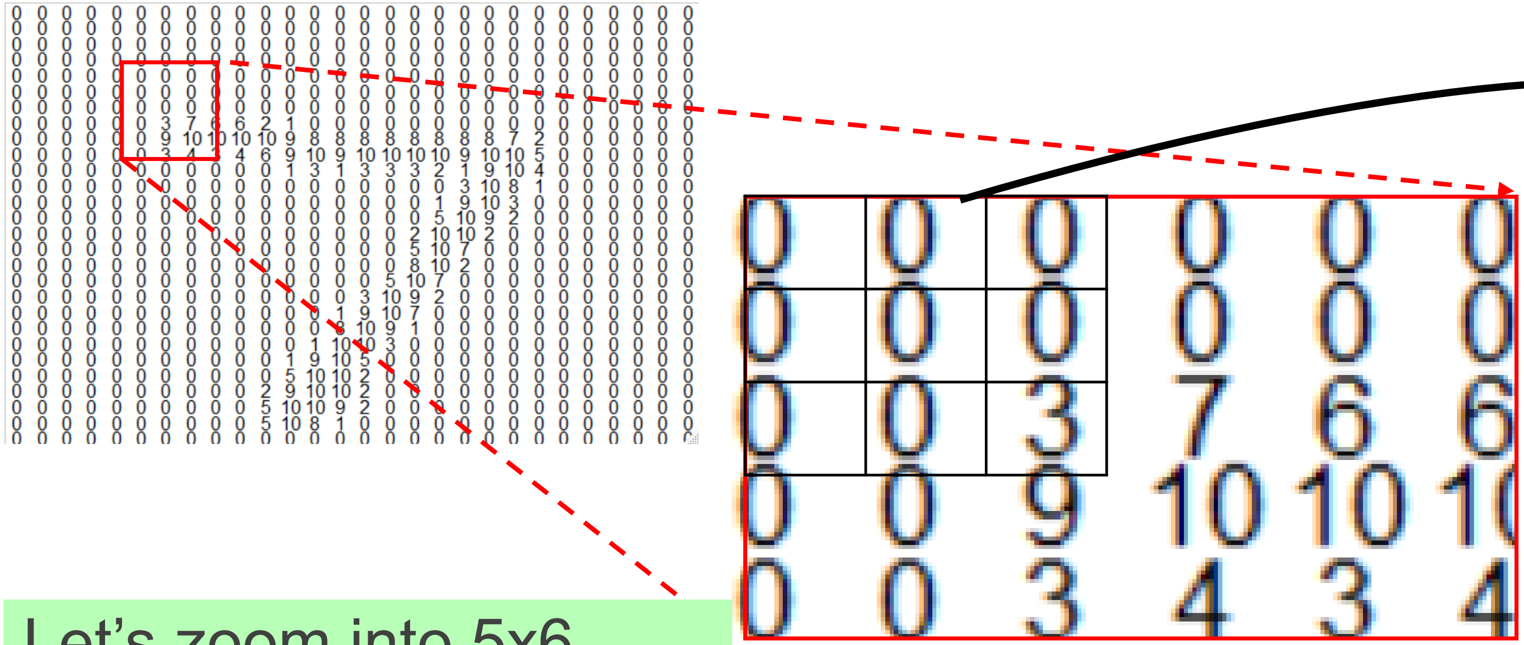
Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge

X

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

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

(our weight parameters)



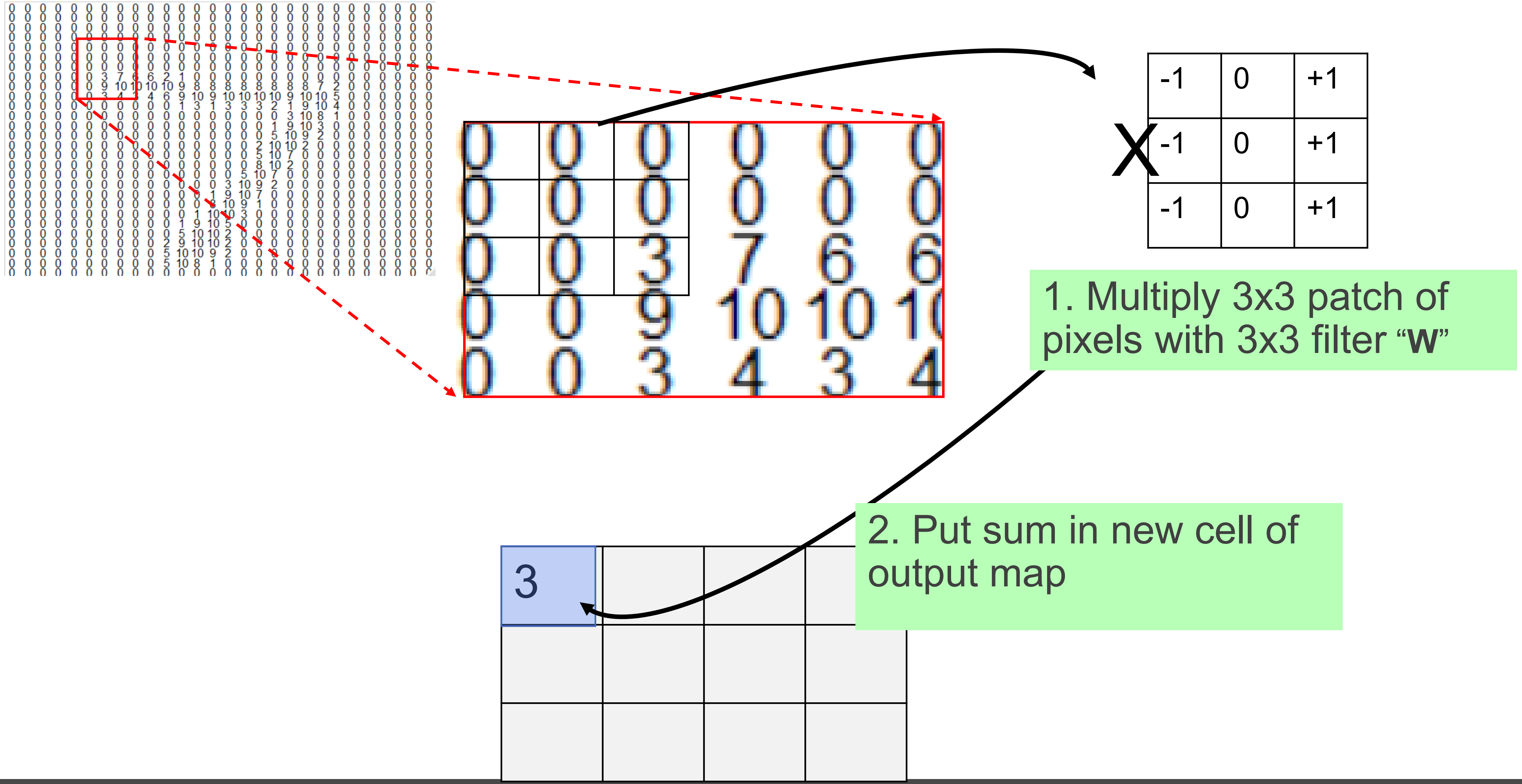
X

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

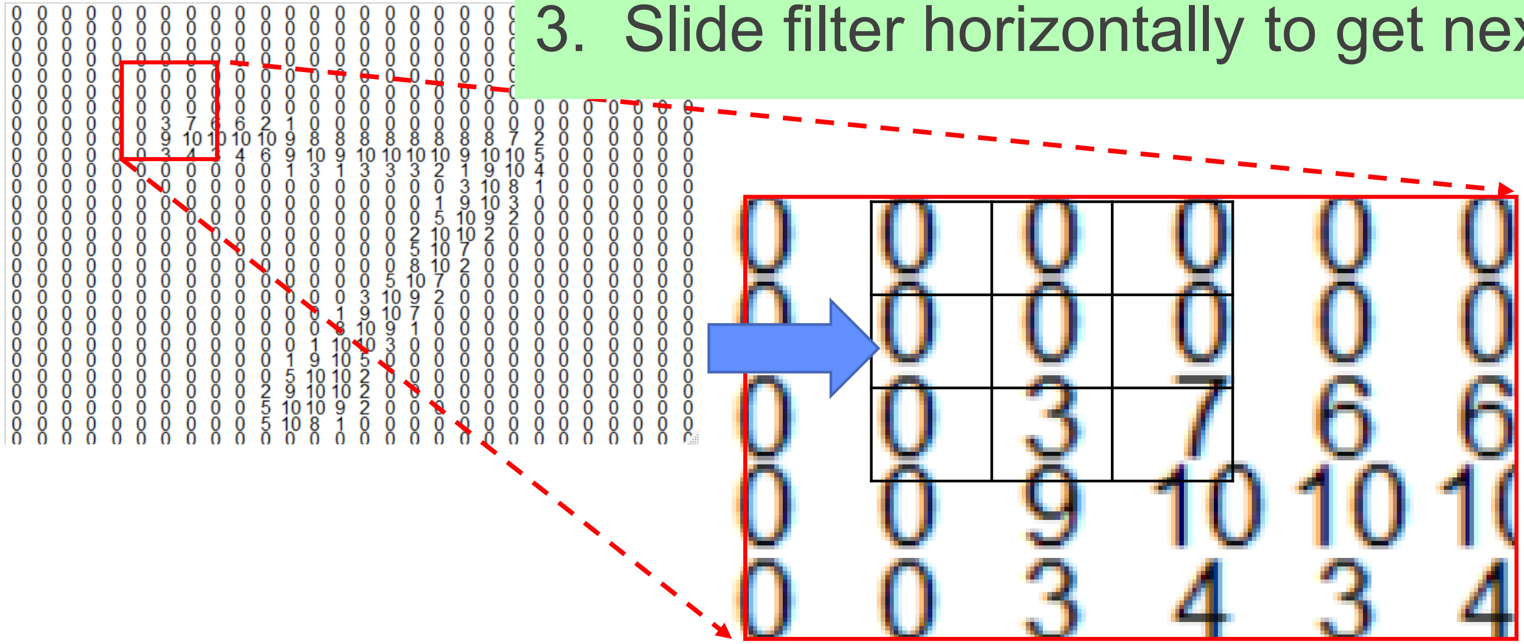
1. Multiply 3x3 patch of pixels with 3x3 filter "**W**"

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

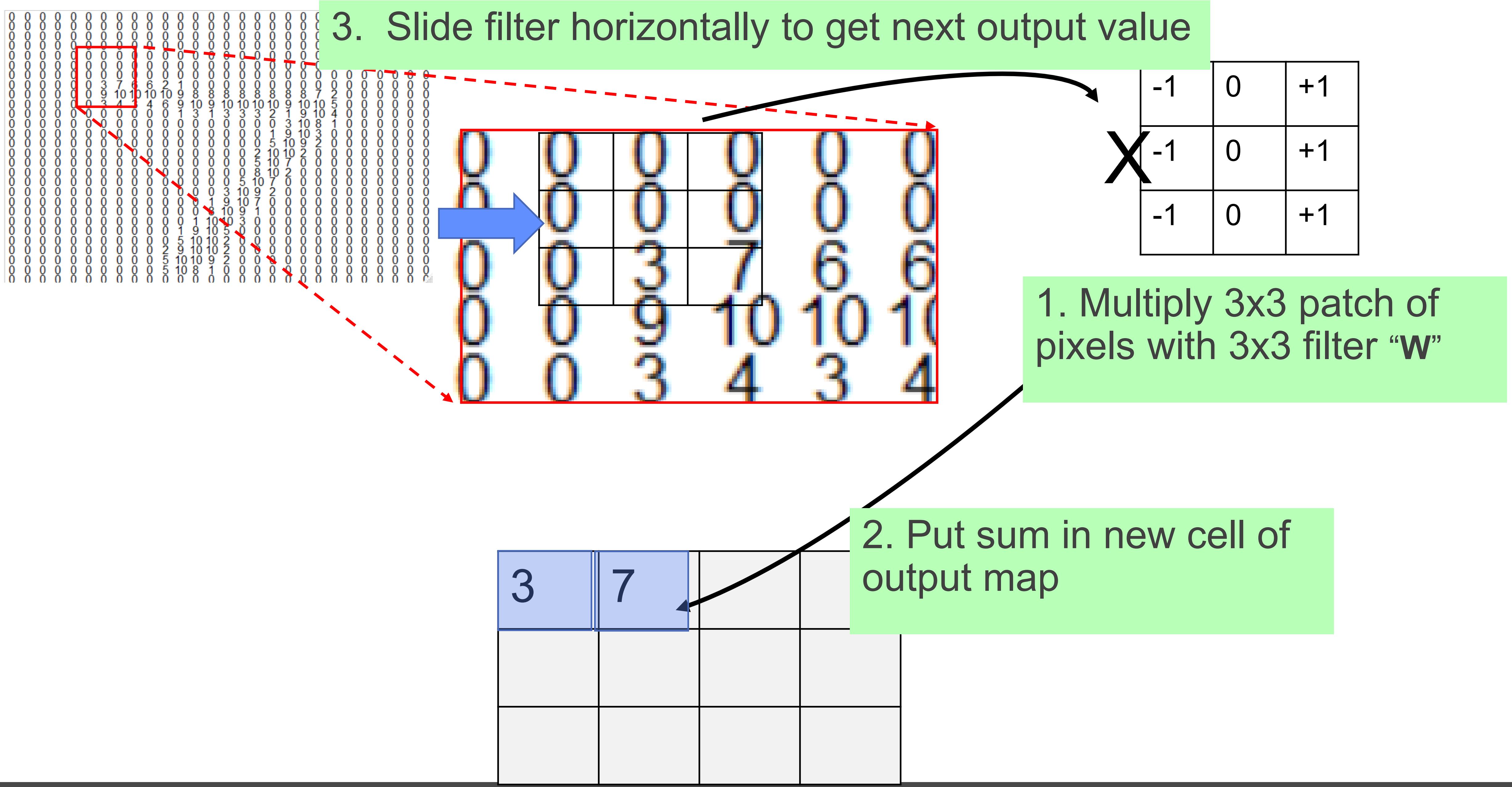
Take a 3x3 patch of pixels and apply a 'filter' template – designed to find an edge


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

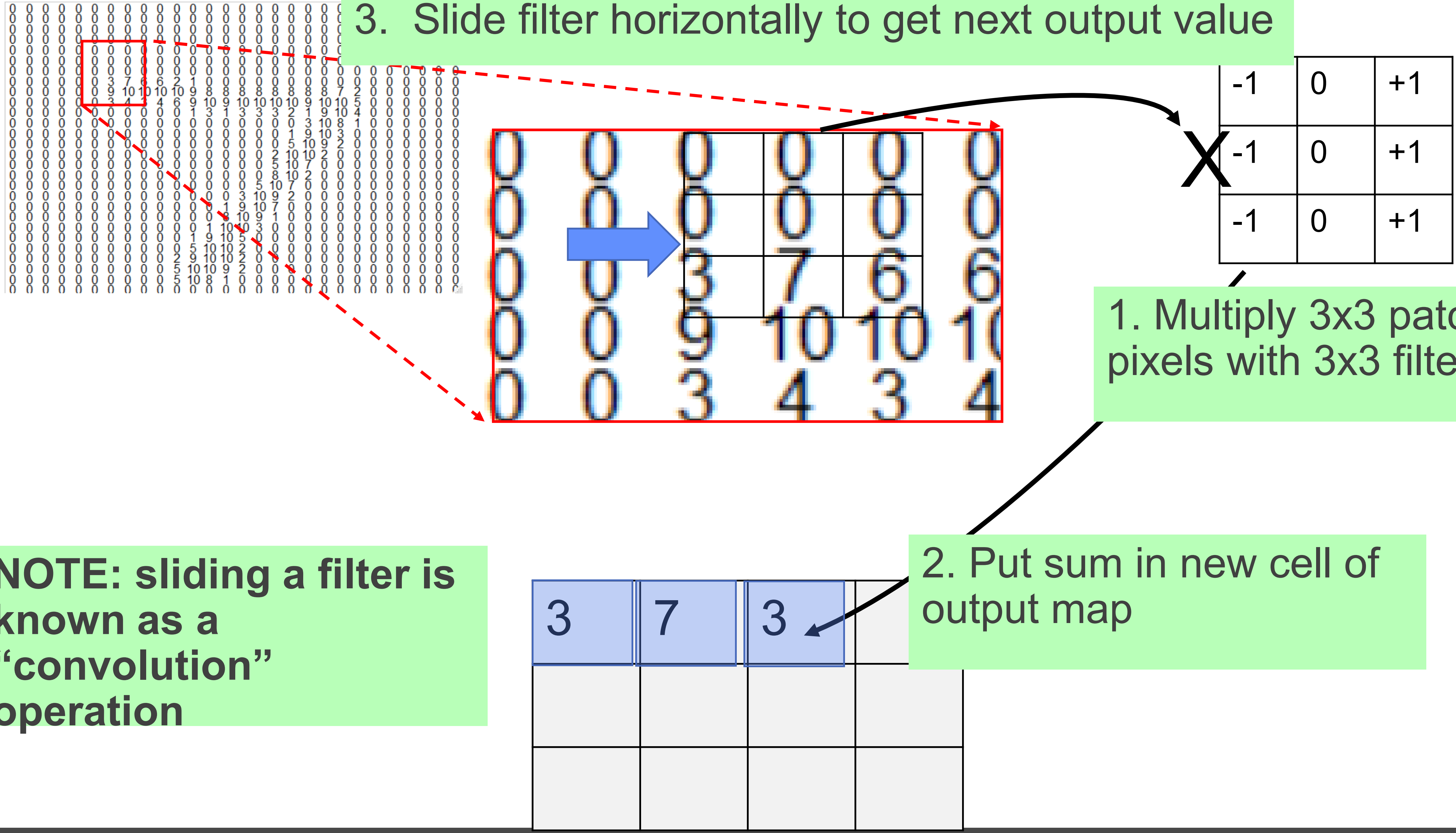
3. Slide filter horizontally to get next output value



3			


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
X
-1 0 +1
-1 0 +1
-1 0 +1
3
7

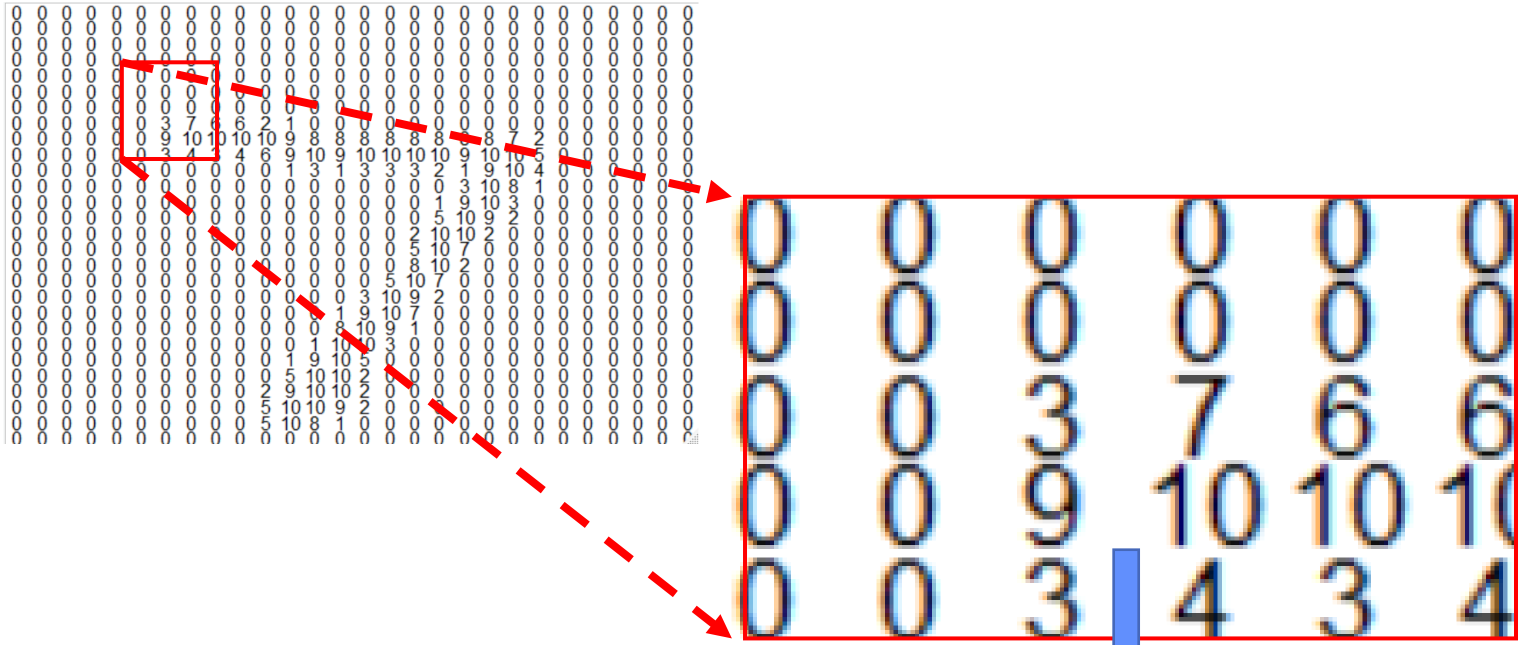
3. Slide filter horizontally to get next output value



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

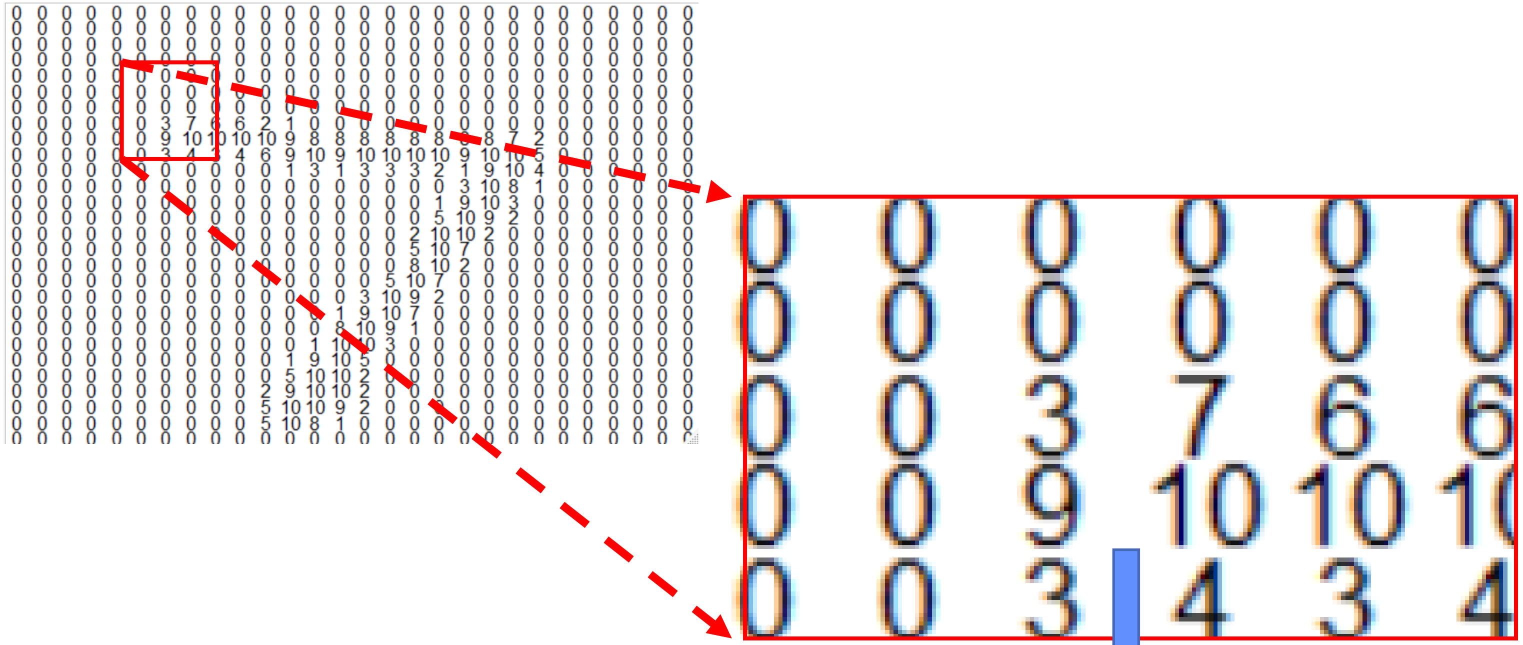
2. Put sum in new cell of output map

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



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

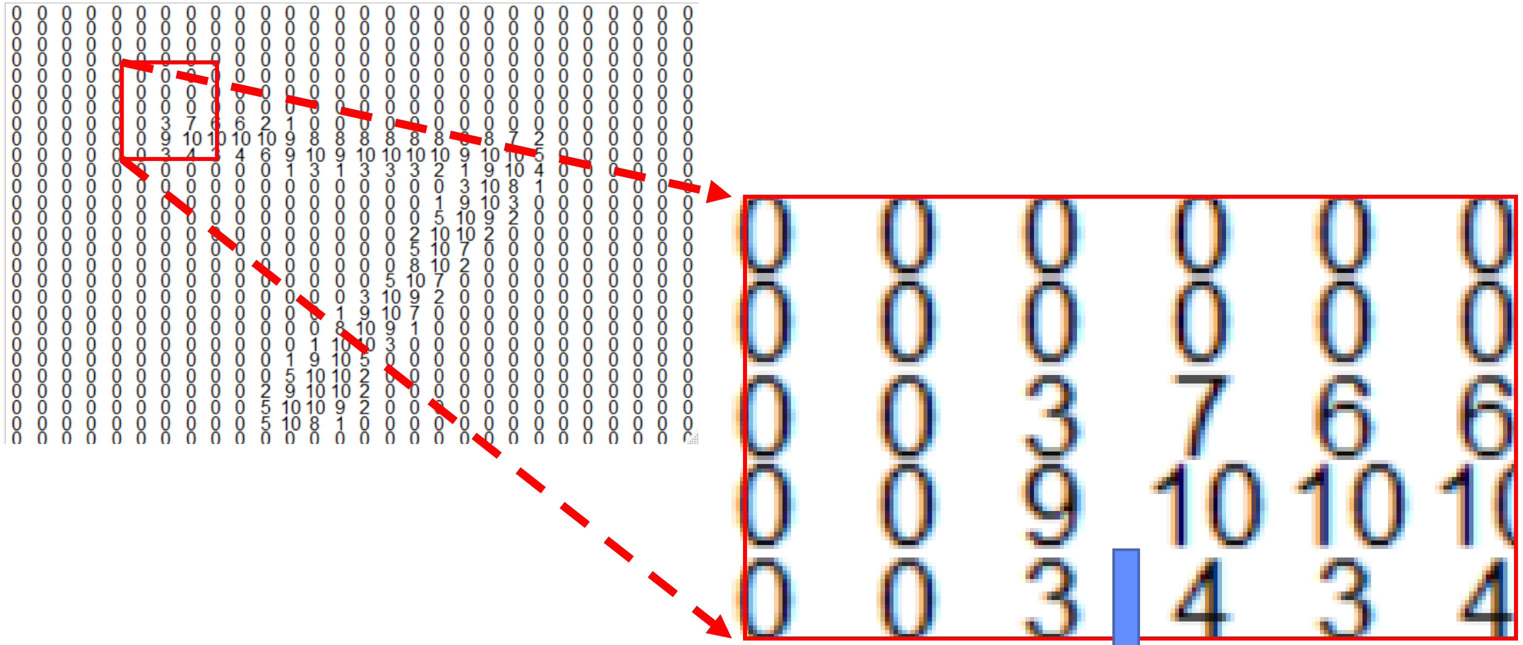
3	7	3	-1
12	17	4	-1
15	21	4	-1



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

What do the highest values in the feature map represent?

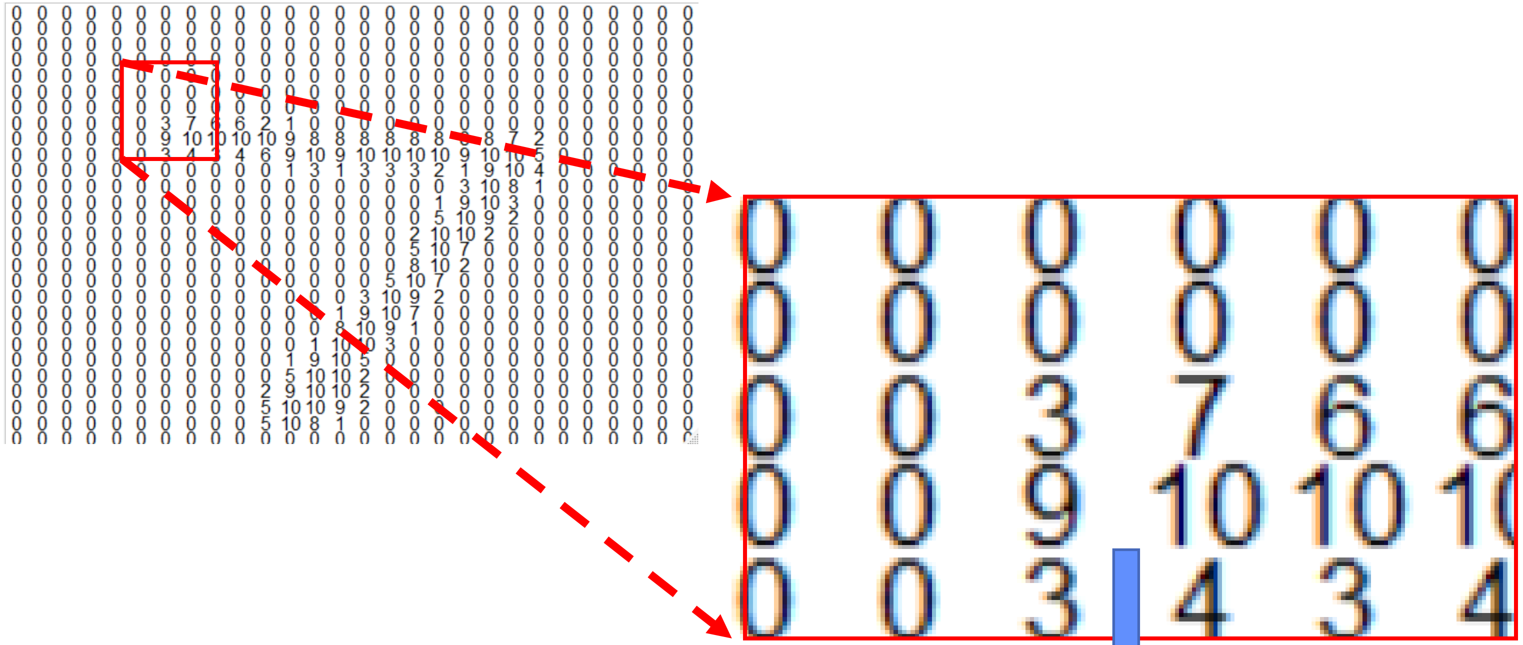
3	7	3	-1
12	17	4	-1
15	21	4	-1



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”



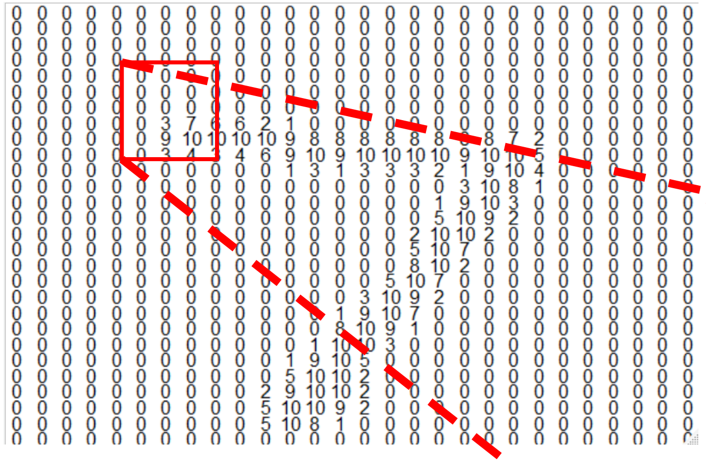
3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

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

2x2 filter has max=17

17		



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

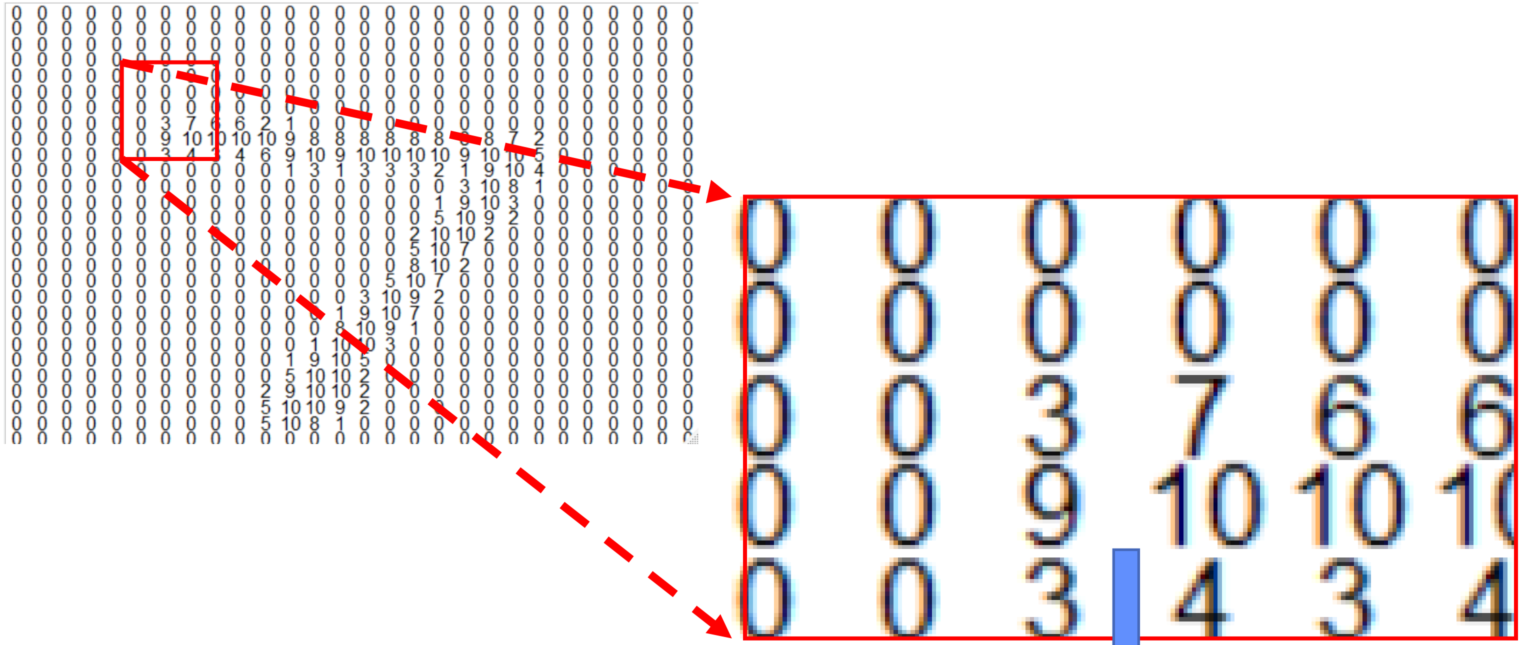
3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

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

Slide filter …

17	17	4
21	21	4

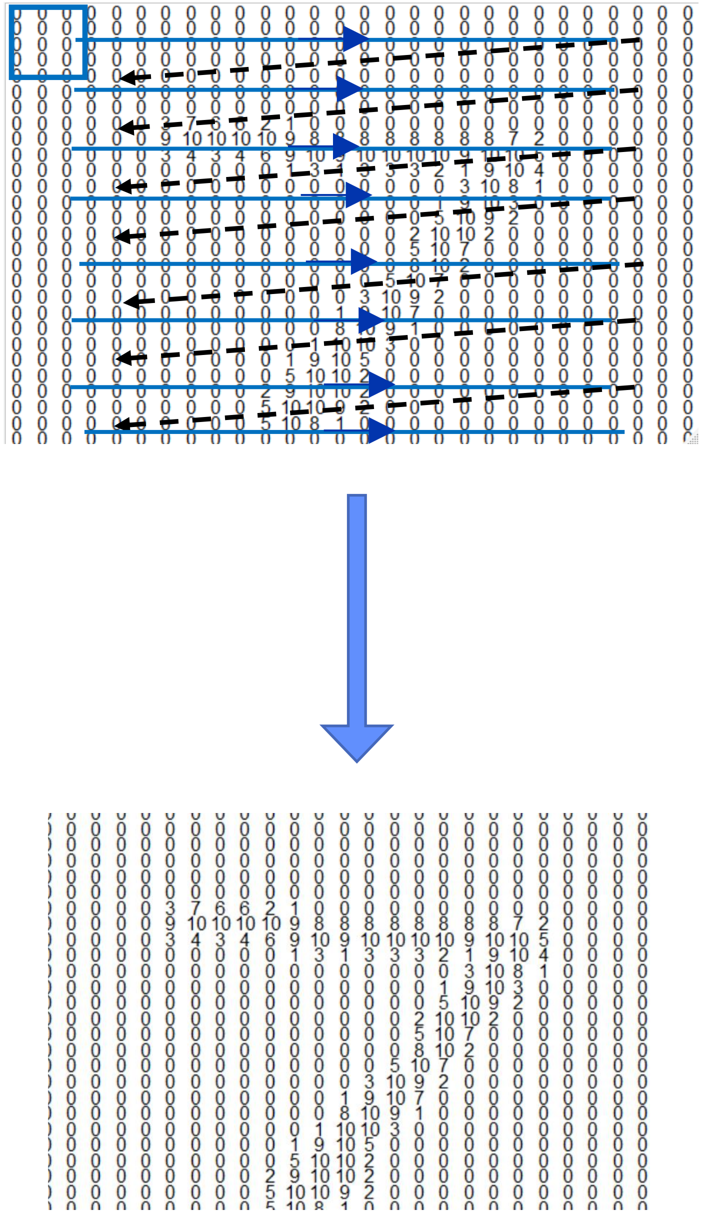


3	7	3	-1
12	17	4	-1
15	21	4	-1

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)

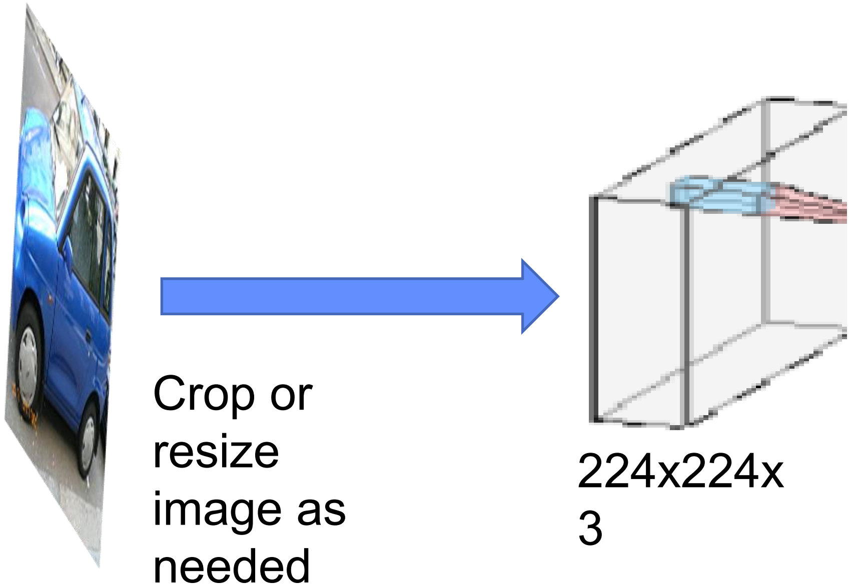
More hyperparameters:

- 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)**



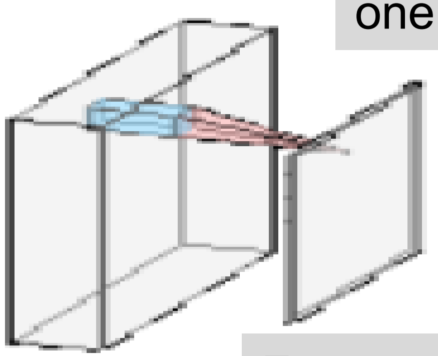
Crop or resize image as needed

224x224x3

Convolution with image

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

Crop or resize image as needed



one 3x3x3 filter

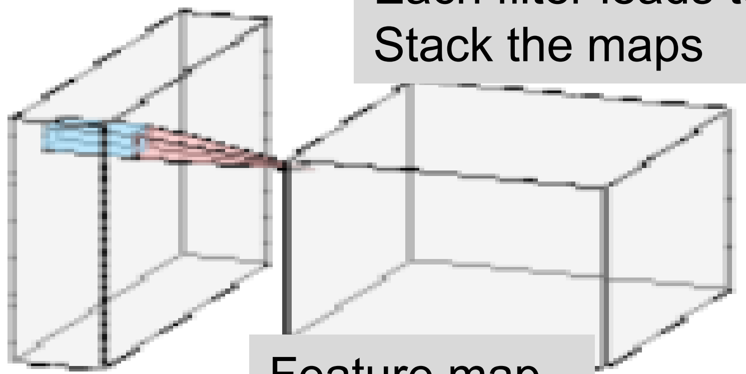
224x224x3

Feature map
Size depends on filter size and slide parameter

Convolution with image

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

Crop or resize image as needed



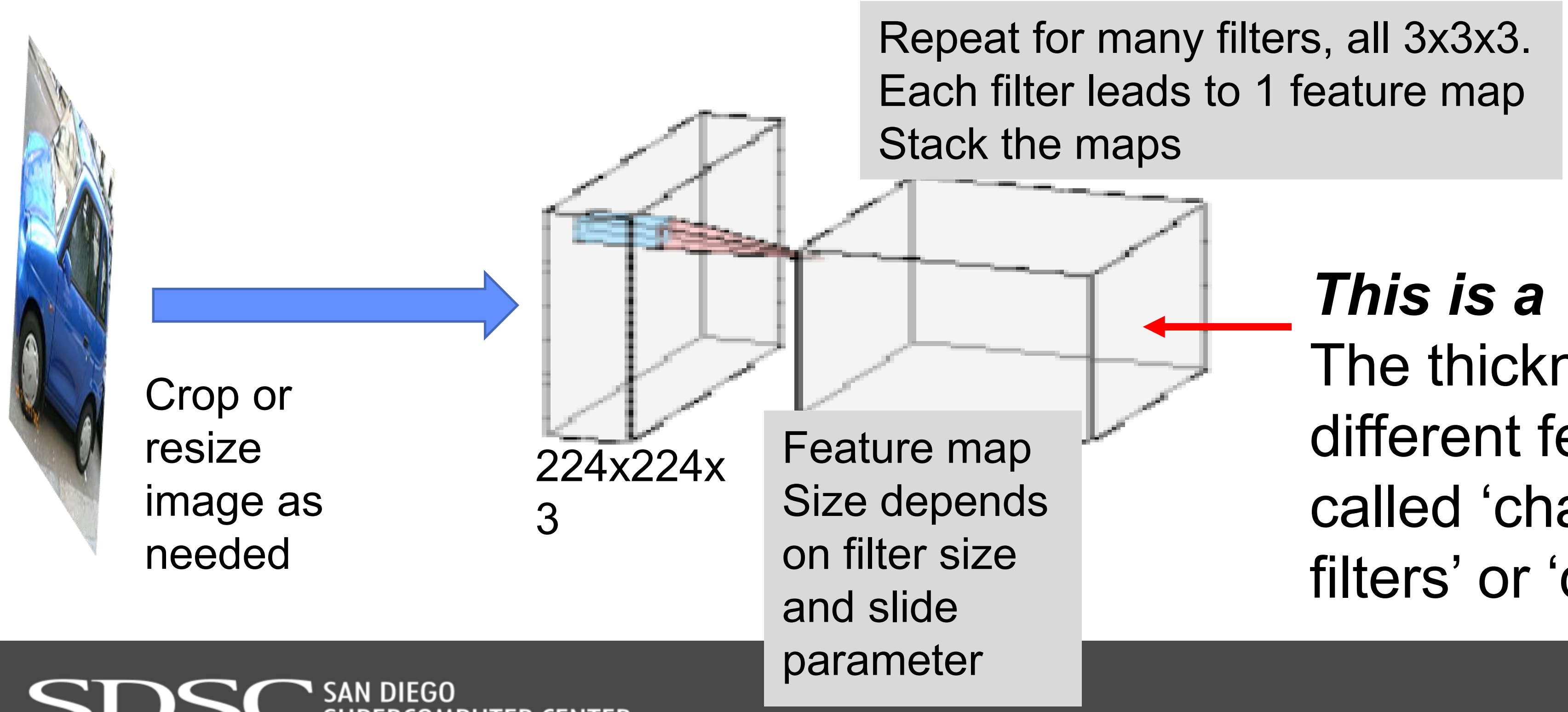
224x224x3

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

Convolution with image

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

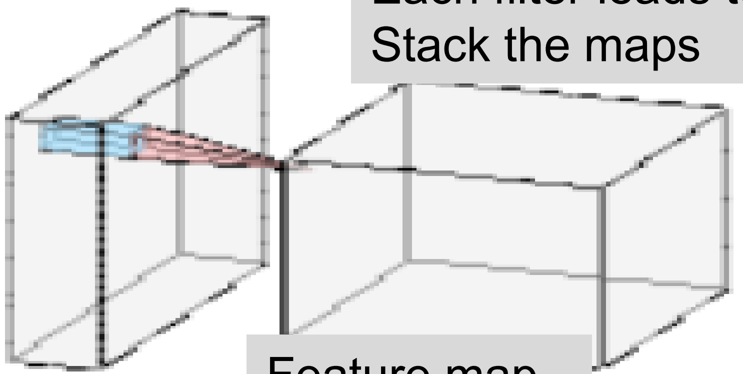


This is a "convolution layer"
The thickness is the number of different feature maps, sometimes called 'channels' or 'number of filters' or 'depth'

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

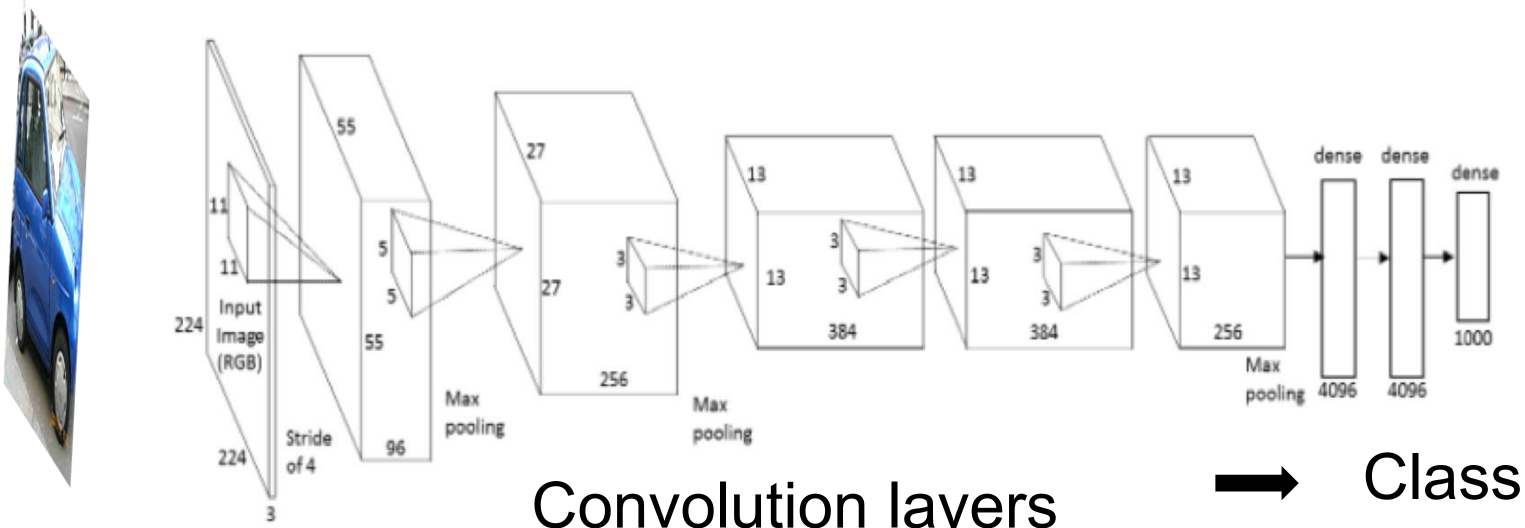
Feature map
Size depends on filter size and slide parameter

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

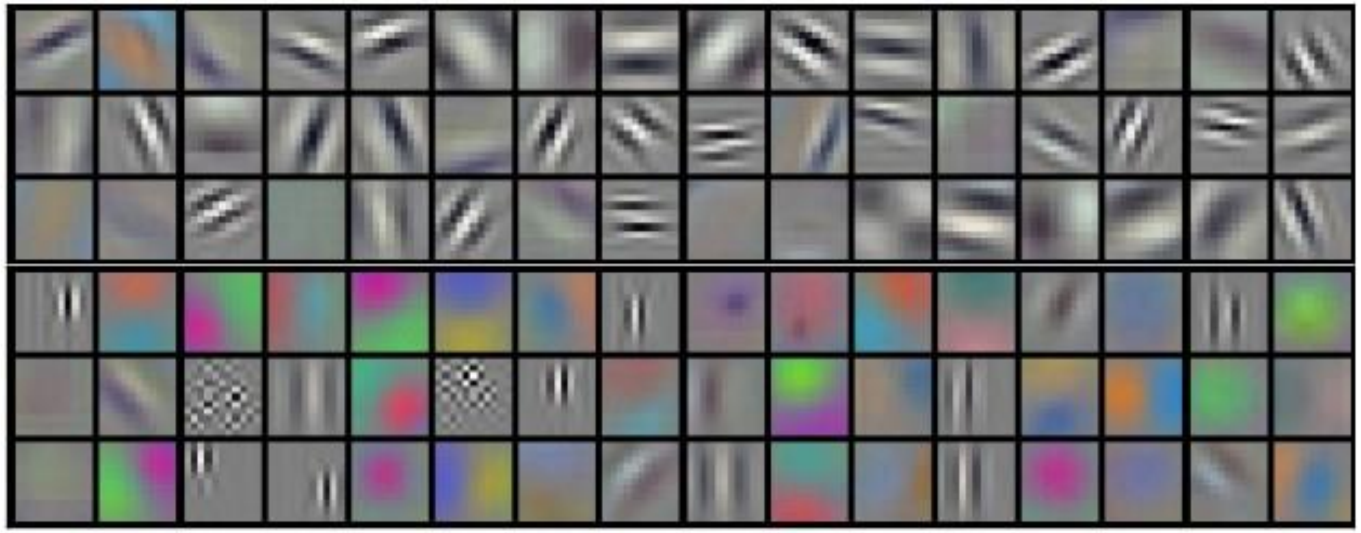
- **Large Convolution Networks – Alexnet, VGG19, ResNet, GoogLeNet, …**



Convolution layers ➞ Classification layers and output

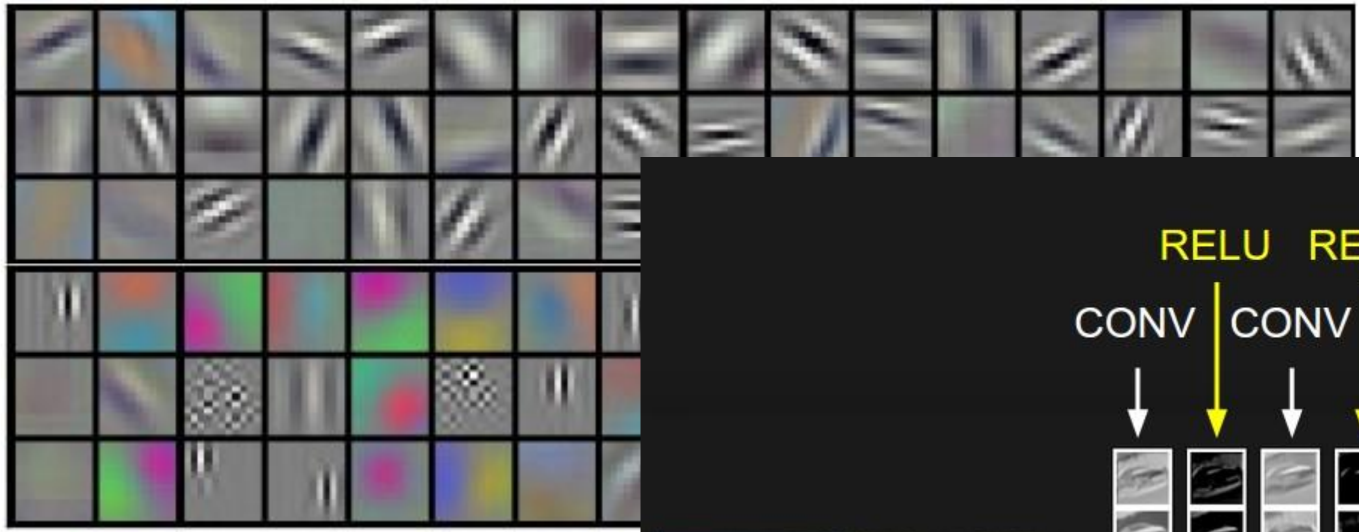
What Learned Convolutions Look Like

First convolution layer filters are simple features

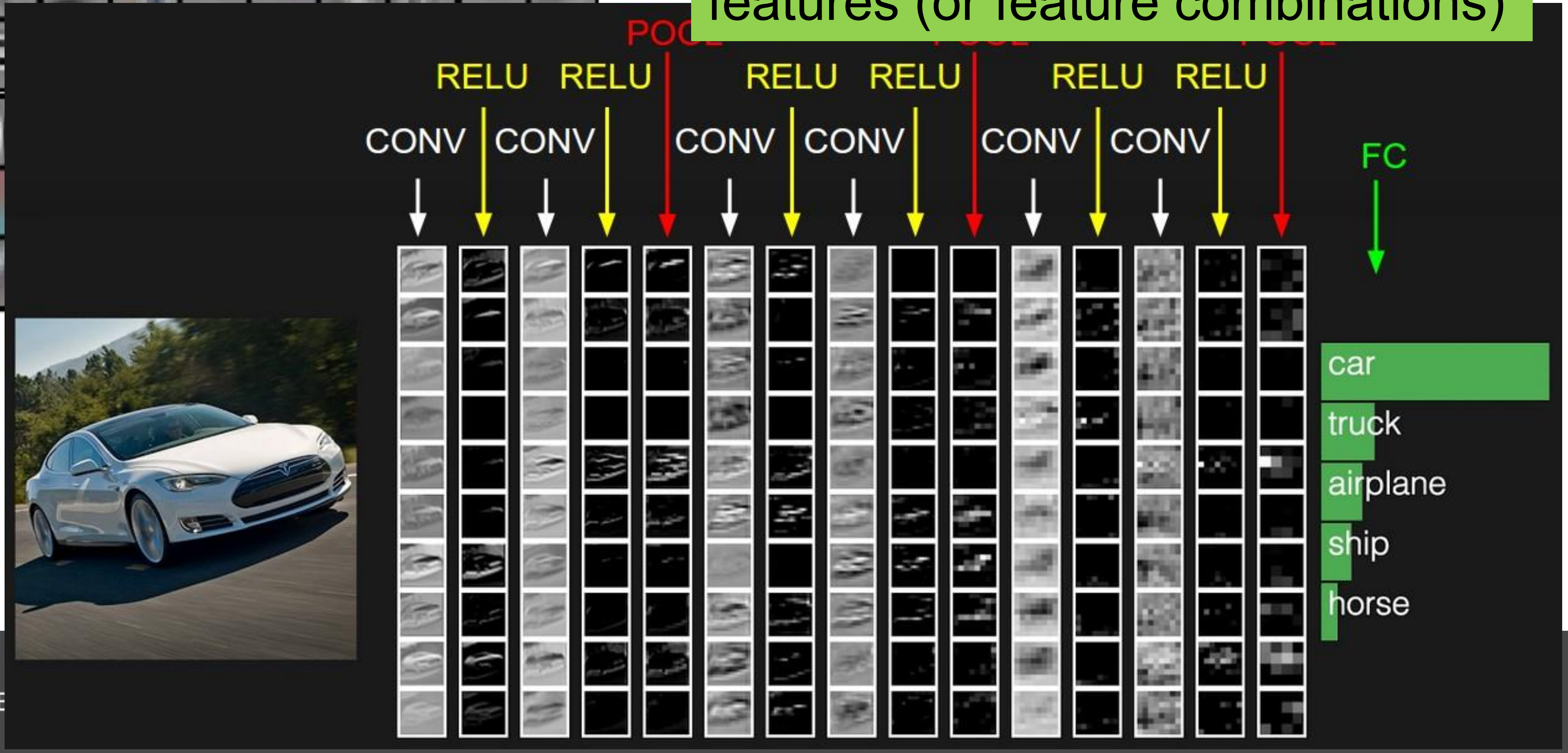


What Learned Convolutions

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

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

- **Next, notebook demo**

Exercise CNN for Digit Classification

- The ‘hello world’ of CNNs
- It uses MNIST dataset and Pytorch coding
- First: Quick intro to Pytorch and CNN layers coding
- Second: We will login and start a notebook

Python code for a ‘Sequential’ neural network

```
def MyNet(numfilt):
    model=torch.nn.Sequential(
        #Conv:  input size 1 channel, output is number of filters, the
        #  actual batch of input is implicit
        # see:   https://docs.pytorch.org/docs/stable/generated/torch.n
        torch.nn.Conv2d(in_channels=1,out_channels=numfilt,
                        kernel_size=kernel_size2use,stride=1),
        torch.nn.ReLU(),
```

Add convolution layer,
relu activation,
Max pooling layer

Python code for a ‘Sequential’ neural network

```
def MyNet(numfilt):
    model=torch.nn.Sequential(
        #Conv:  input size 1 channel, output is number of filters, the
        #  actual batch of input is implicit
        # see:   https://docs.pytorch.org/docs/stable/generated/torch.n
        torch.nn.Conv2d(in_channels=1,out_channels=numfilt,
                          kernel_size=kernel_size2use,stride=1),
        torch.nn.ReLU(),
```

Add convolution layer,
relu activation,
Max pooling layer

Images are 1x28x28,
so input is 1 channel of a 2D image

Output will be given by ‘num filters’ argument

Output matrix shape will depend on filter size
(kernel size) and stride

Python code for a ‘Sequential’ neural network

```
def MyNet(numfilt):
    model=torch.nn.Sequential(
        #Conv:  input size 1 channel, output is number of filters, the
        #  actual batch of input is implicit
        # see:   https://docs.pytorch.org/docs/stable/generated/torch.n
        torch.nn.Conv2d(in_channels=1,out_channels=numfilt,
                        kernel_size=kernel_size2use,stride=1),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(3,2),
        torch.nn.Flatten(),
        torch.nn.Linear(numfilt*reduced_size*reduced_size,16), #after m
        torch.nn.ReLU(),
        torch.nn.Linear(16, 10),
        torch.nn.LogSoftmax(dim=1))
    return model
```

Add convolution layer,
relu activation,
Max pooling layer

Then flatten into a vector
for classification layers

Python code for a 'Sequential' neural network

```
def MyNet(numfilt):
    model=torch.nn.Sequential(
        #Conv:  input size 1 channel, output is number of filters, the
        #  actual batch of input is implicit
        # see:   https://docs.pytorch.org/docs/stable/generated/torch.n
        torch.nn.Conv2d(in_channels=1,out_channels=numfilt,
                        kernel_size=kernel_size2use,stride=1),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(3,2),
        torch.nn.Flatten(),
        torch.nn.Linear(numfilt*reduced_size*reduced_size,16), #after m
        torch.nn.ReLU(),
        torch.nn.Linear(16, 10),
        torch.nn.LogSoftmax(dim=1))
    return model
```

Add convolution layer,
Relu activation function
Max pooling layer

Then flatten into a vector
for classification layers

- Every layer has some input, ouput
- Network layers need input and output size (e.g. channels or hidden units)
- Not every layer/function has trainable parameters – like which one of these?

Python code for a functional (non-sequential) neural network

```
# ------------------------------------------------------------
class MyNet(torch.nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        #Conv:  input size 1 channel, output is number of filters, the
        #  actual batch of input is implicit
        # see:   https://docs.pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
        self.conv1   = torch.nn.Conv2d(in_channels=1,out_channels=numfilt,kernel_size=
        self.linear1 = torch.nn.Linear(numfilt*reduced_size*reduced_size,16) #after mo
        self.linear2 = torch.nn.Linear(16, 10)
```

Python ‘nn’ class objects are very flexible

In the initialization you define layer functions

‘MyNet’ inherits functionality from ‘nn’

Python code for a functional (non-sequential) neural network

```
# -------------------------------------------------------------
class MyNet(torch.nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        #Conv:  input size 1 channel, output is number of filters, the
        #  actual batch of input is implicit
        # see:   https://docs.pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
        self.conv1    = torch.nn.Conv2d(in_channels=1,out_channels=numfilt,kernel_size=
        self.linear1 = torch.nn.Linear(numfilt*reduced_size*reduced_size,16) #after ma
        self.linear2 = torch.nn.Linear(16, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 3, 2)

        # <<<<<<<<<<<<<<<<<--------------------
        #Uncomment this to see what the size actually is after max pooling
        #print('MYINFO  fwd, after max, x shape:',x.shape)

        x = torch.flatten(x, 1)
        x = self.linear1(x)
        x = F.relu(x)
        x = self.linear2(x)
        #not sure i need this    x = F.relu(x)
        output = F.log_softmax(x, dim=1)  #log softmax for classfcnt or binary?
        return output
```

Python 'nn' class objects are very flexible

In the initialization you define layer functions

In the forward function you indicate the inputs and lay out the model

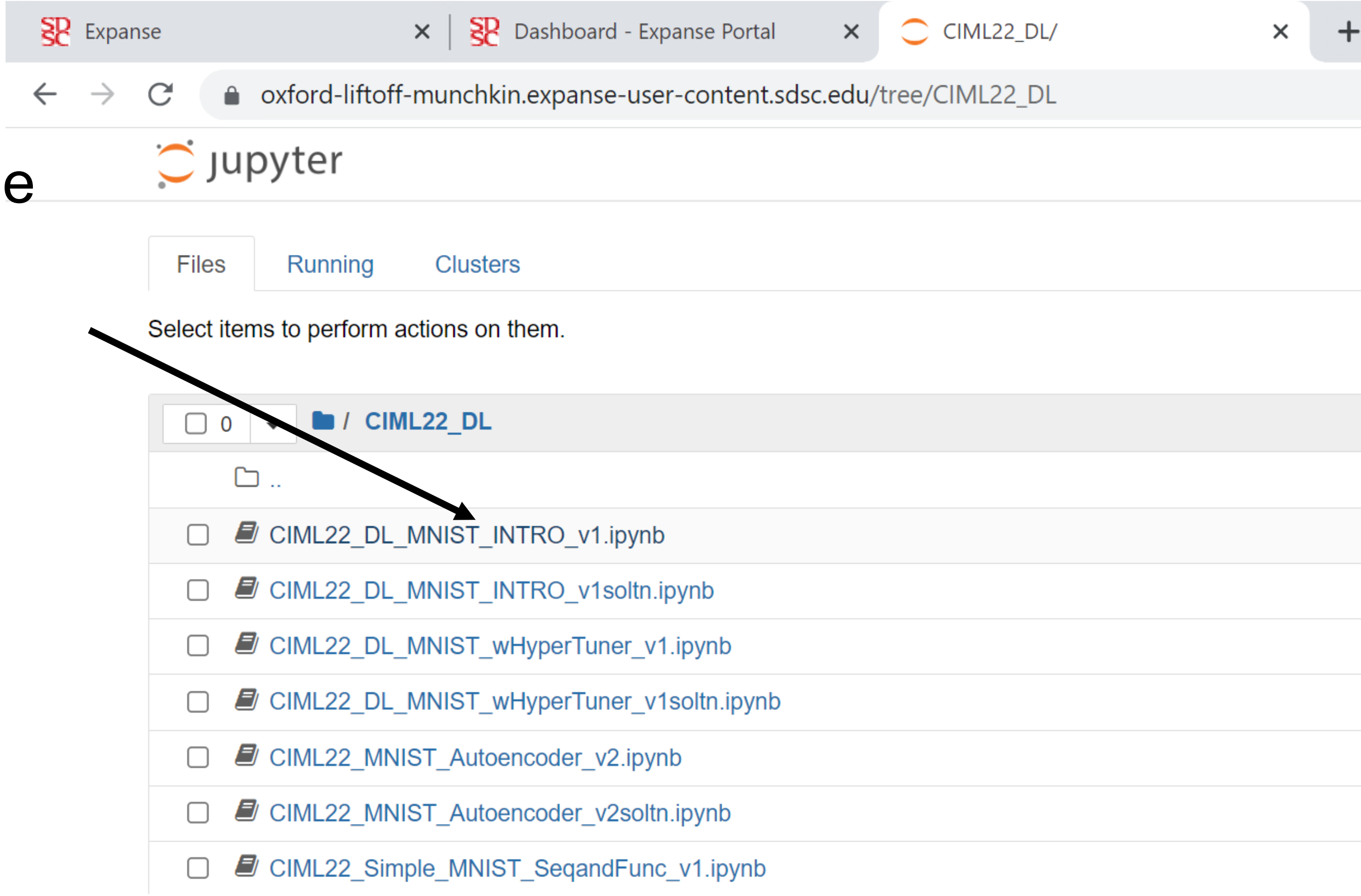
You can mix layers and torch functions in flexible ways

Pytorch coding notes:

$ jupyter-gpu-shared-pylight

In jupyter notebook session open the CIML_MNIST_Intro notebook

Follow instructions in the notebook



Notebook code overview

1. Import statements
2. Define parameters (arguments)
3. Get data and set up Pytorch dataloader

Notebook code overview

1. Import statements
2. Define parameters (arguments)
3. Get data and set up Pytorch dataloader

the loader is a Python iterator to help retrieve a batch of data at a time

```
dataset1 = datasets.MNIST(data_path, train=True, download=True,transform=transform)
dataset2 = datasets.MNIST(data_path, train=False,download=True,transform=transform)

train_loader =torch.utils.data.DataLoader(dataset1,
            batch_size =batch_size,     sampler   =None,
            num_workers=num_worker2use, pin_memory=True, drop_last=True)
test_loader = torch.utils.data.DataLoader(dataset2,
            batch_size =batch_size,     sampler   =None,
            num_workers=num_worker2use, pin_memory=True, drop_last=True)
```

Notebook code overview

1. Import statements
2. Define parameters (arguments)
3. Get data and set up Pytorch dataloader
4. Define functions to create network and do a forward pass
5. Initialize model, move to GPU device (if available)

Notebook code overview

1. Import statements
2. Define parameters (arguments)
3. Get data and set up Pytorch dataloader
4. Define functions to create network and do a forward pass
5. Initialize model, move to GPU device (if available)

A gpu node has 4 gpu devices (cards). For single device execution just set to 'current_device' .

```
use_cuda = torch.cuda.is_available()
if use_cuda:
        num_gpu = torch.cuda.device_count()
        print('INFO,  cuda, num gpu:',num_gpu)
        device     = torch.cuda.current_device()
```

Notebook code overview

1. Import statements
2. Define parameters (arguments)
3. Get data and set up Pytorch dataloader
4. Define functions to create network and do a forward pass
5. Initialize model, move to GPU device (if available)

A gpu node has 4 gpu devices (cards). For single device execution just set to 'current_device' .

```
use_cuda = torch.cuda.is_available()
if use_cuda:
        num_gpu = torch.cuda.device_count()
        print('INFO,  cuda, num gpu:',num_gpu)
        device      = torch.cuda.current_device()
```

We also move model and data to device before training

```
mymodel = MyNet().to(device)
```

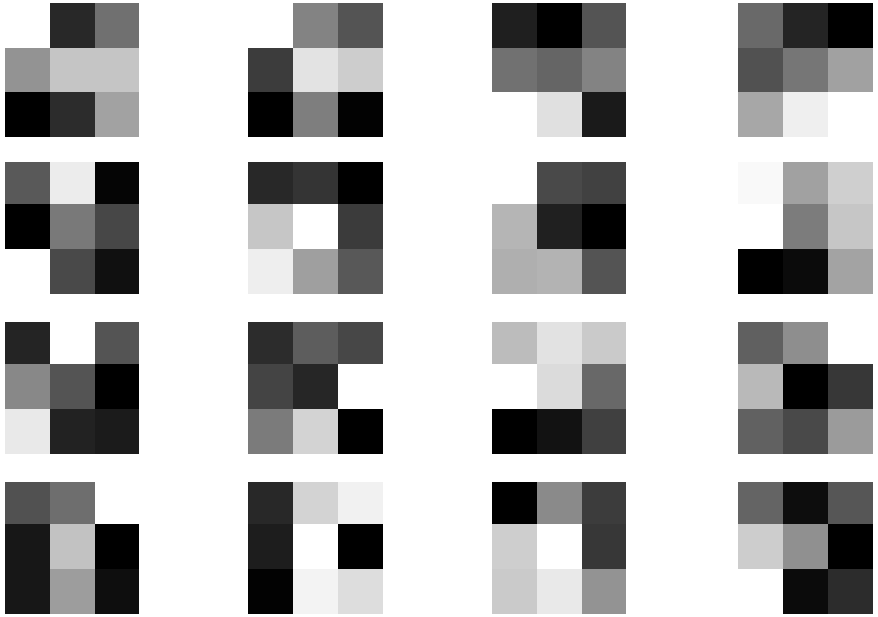
Notebook code overview

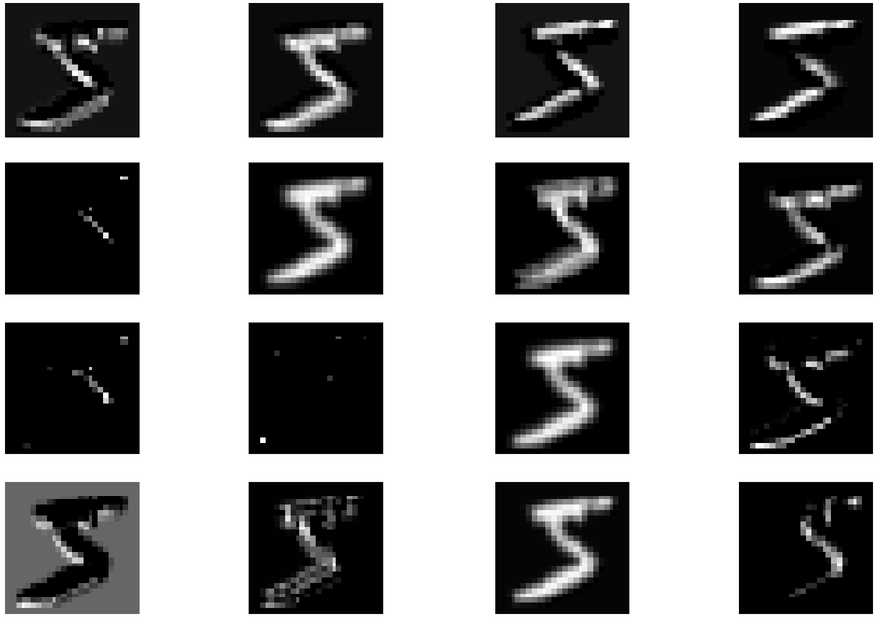
1. Import statements
2. Define parameters (arguments)
3. Get data and set up Pytorch dataloader
4. Define functions to create network and do a forward pass
5. Initialize model, move to GPU device (if available)
6. Run training loops of forward pass, get loss, run loss.backward (Pytorch keeps track of gradients)
7. Display results

Note, for testing, turn off gradients

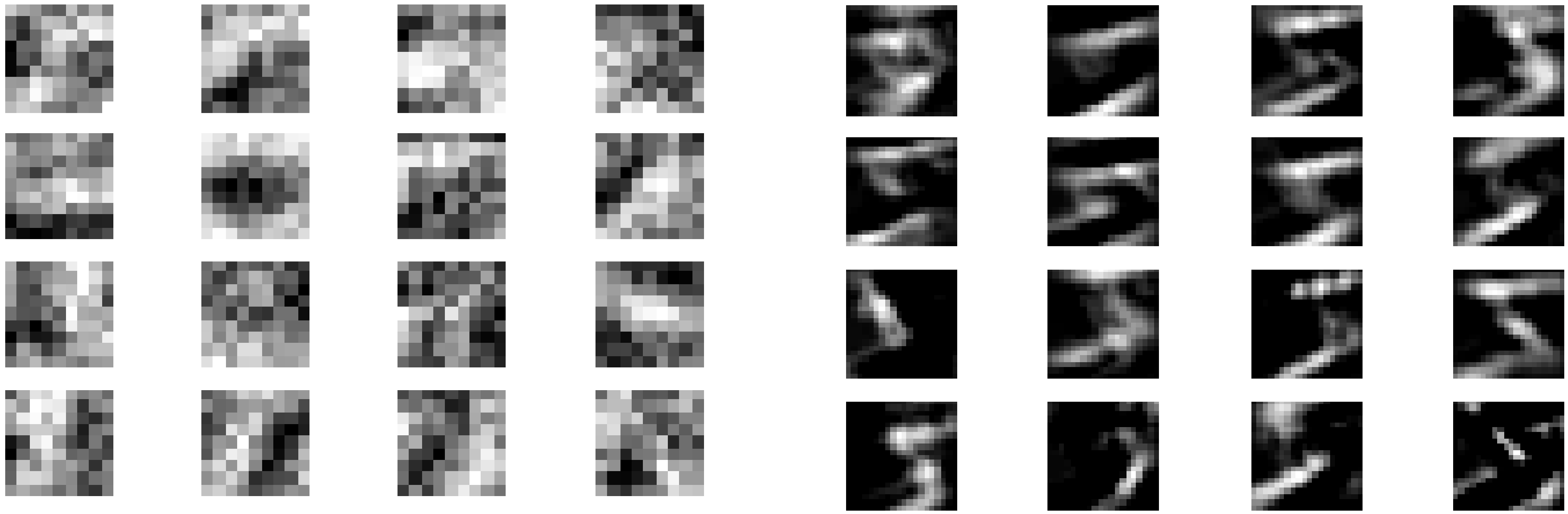
```
with torch.no_grad():
  for batch_idx, (data, target) in enumerate(test_loader):
    if batch_idx*batch_size>= max_numtest:
```

Exercise notes: 3x3 first convolution layer filter and activation

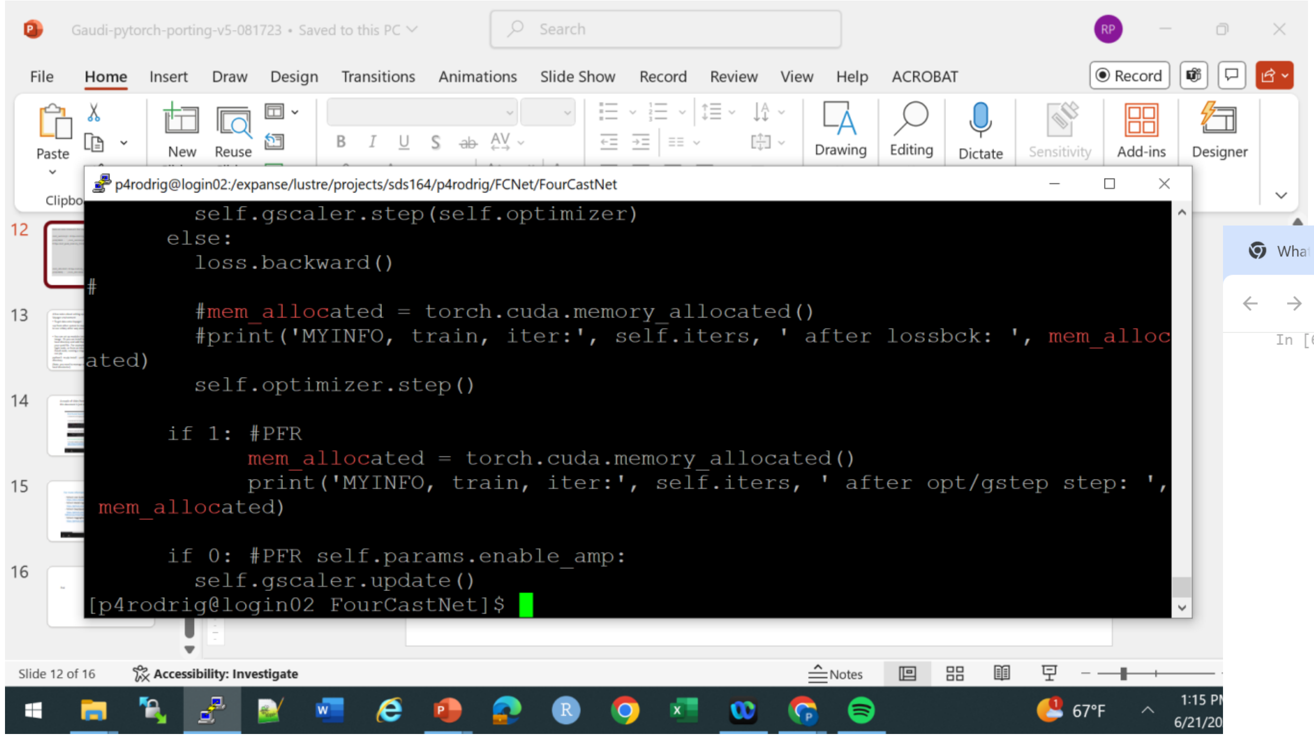


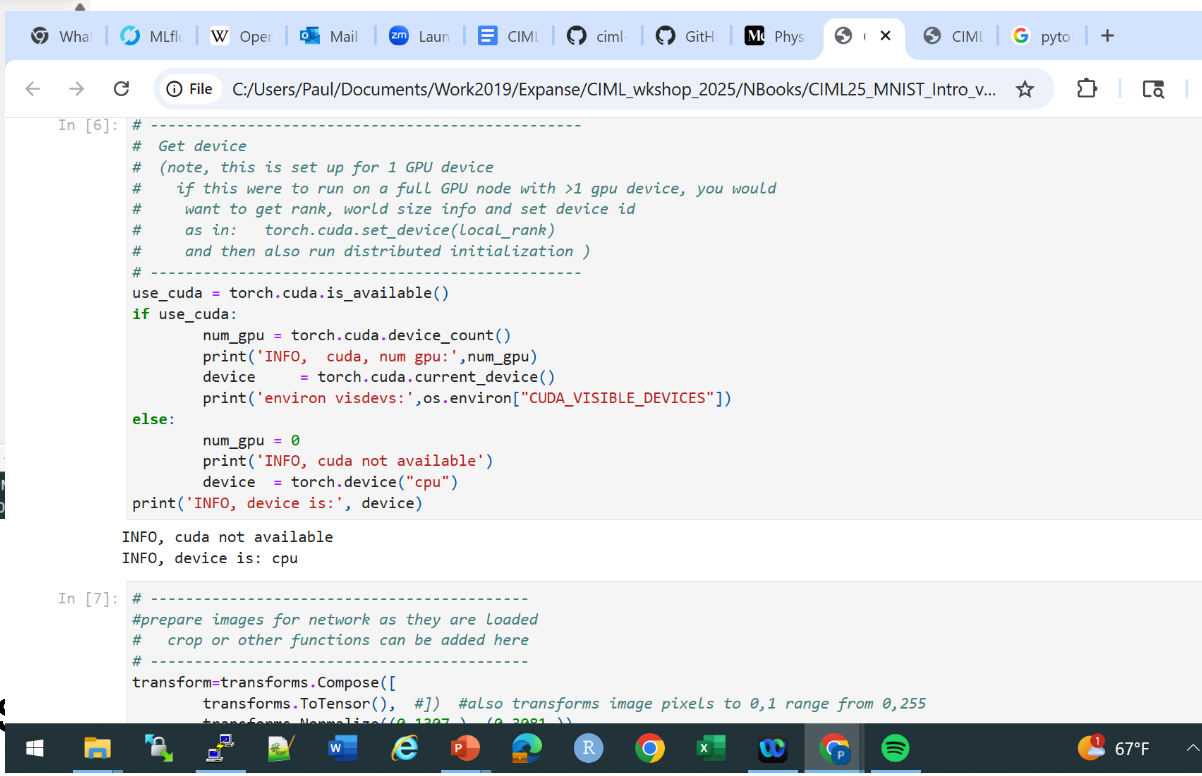


9x9 first convolution layer filter and activation



End





On a Gaudi node, all devices are known as
Nvidia-smi