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

Paul Rodriguez, PhD
(SDSC)

06/29/2023

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

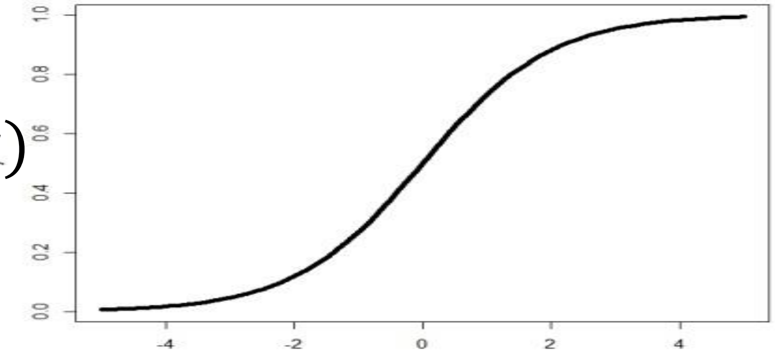
Exercise, Multinode MNIST

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

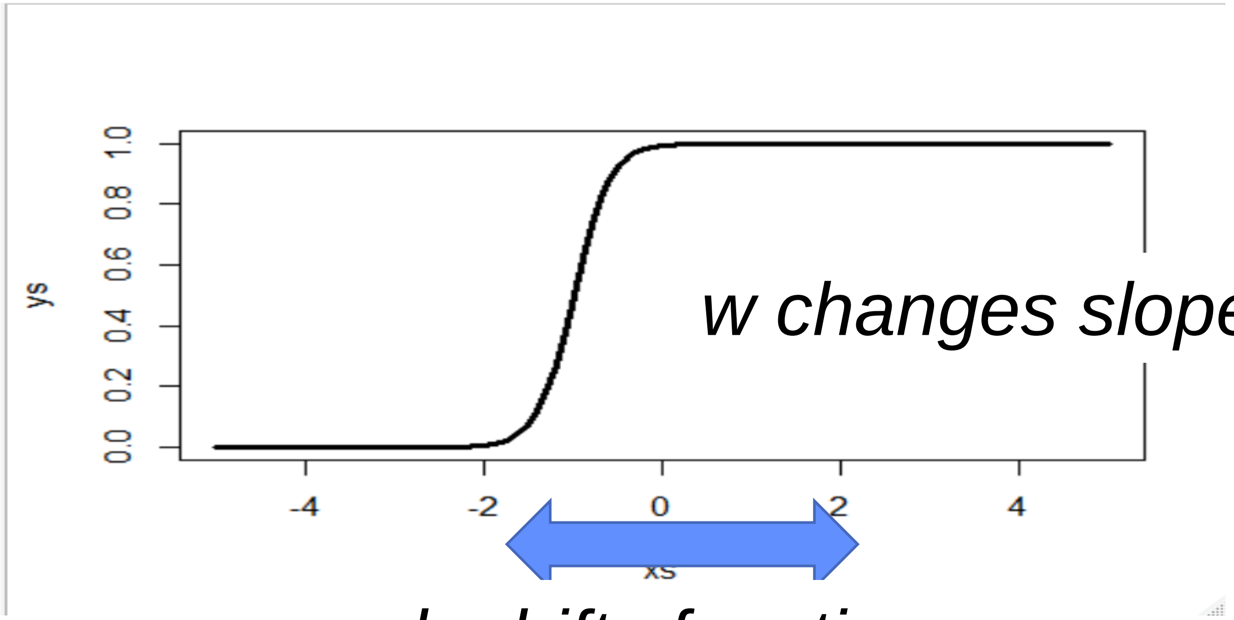
$y = f(x)$



x

Logistic Regression to Neural Network

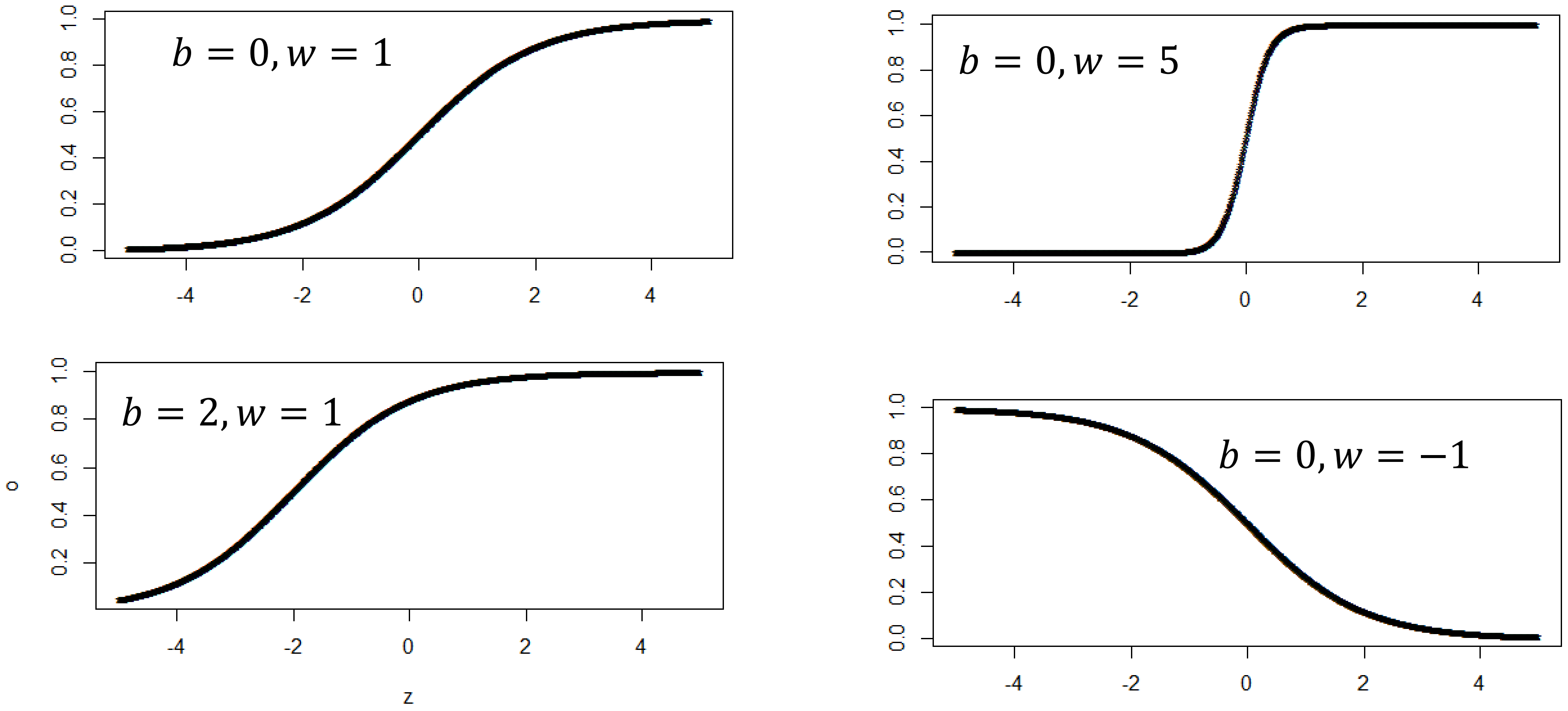
$$f(x,b,w) \quad = \quad \frac{\exp^{(b+w*x)}}{1+\exp^{(b+wx)}}$$



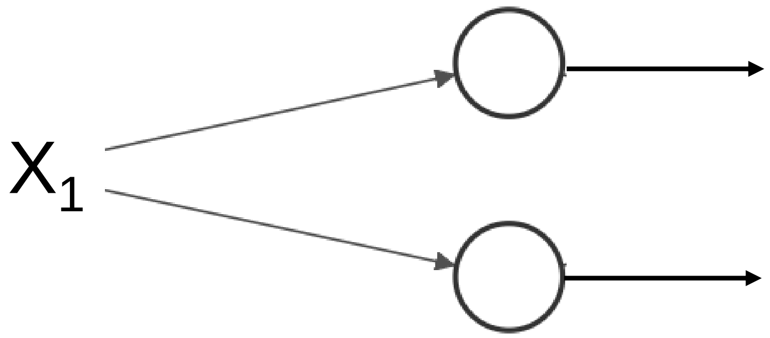
w changes slope

b shifts function

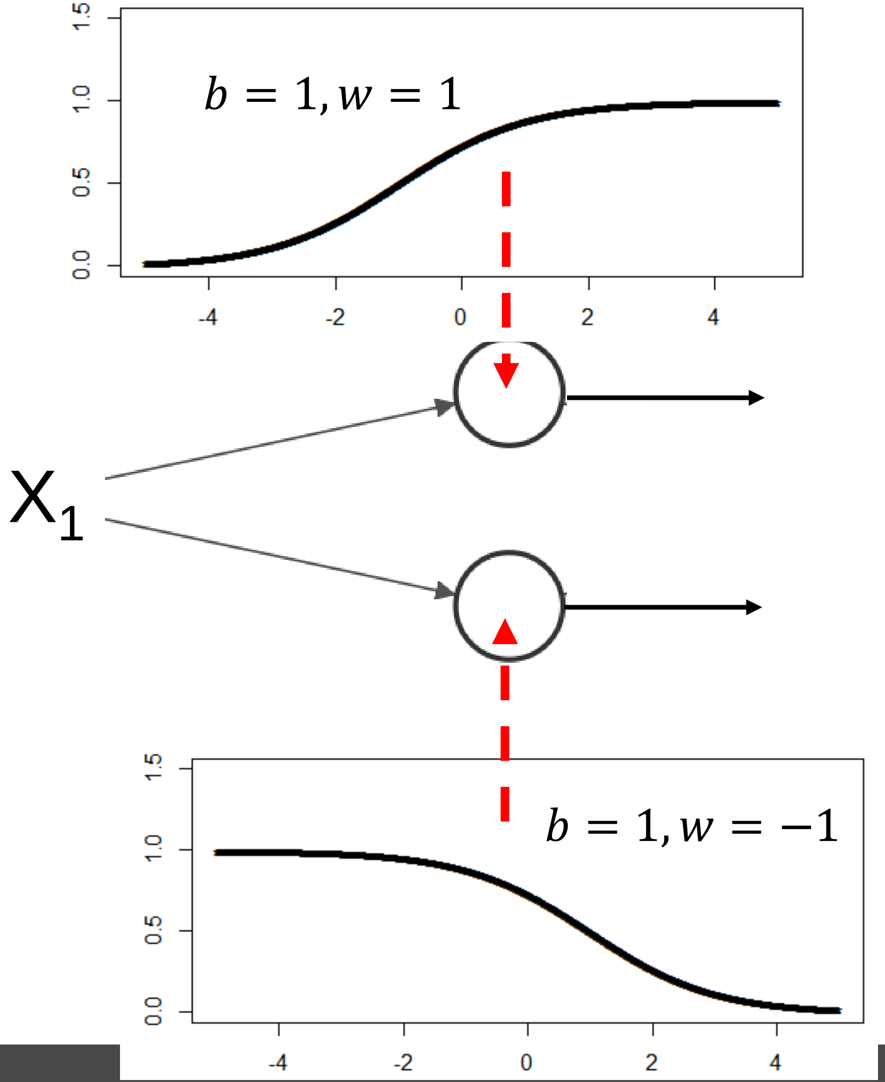
Logistic function w/various weights



Example: 1 input into 2 logistic units

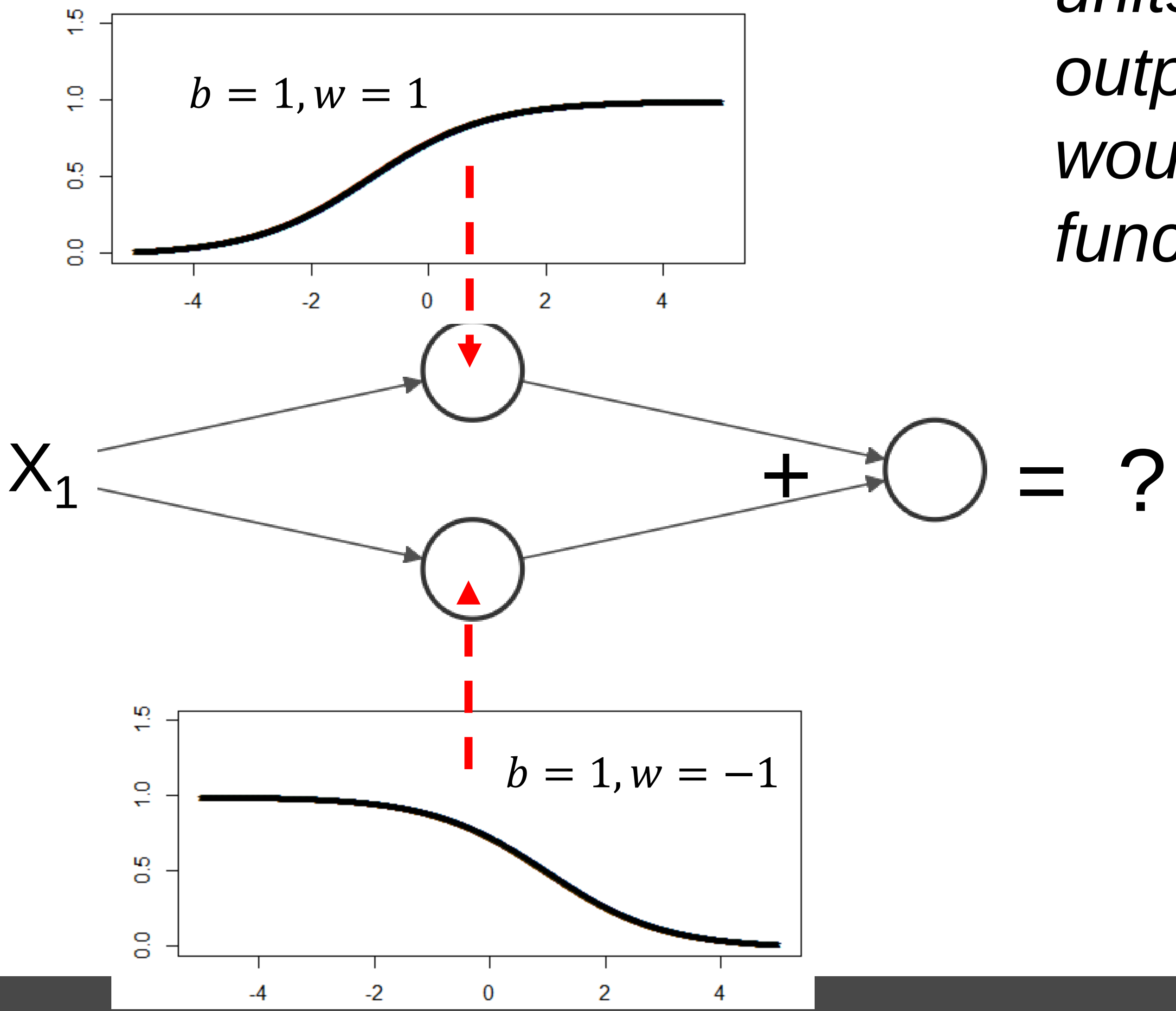


Example: 1 input into 2 logistic units with these activations



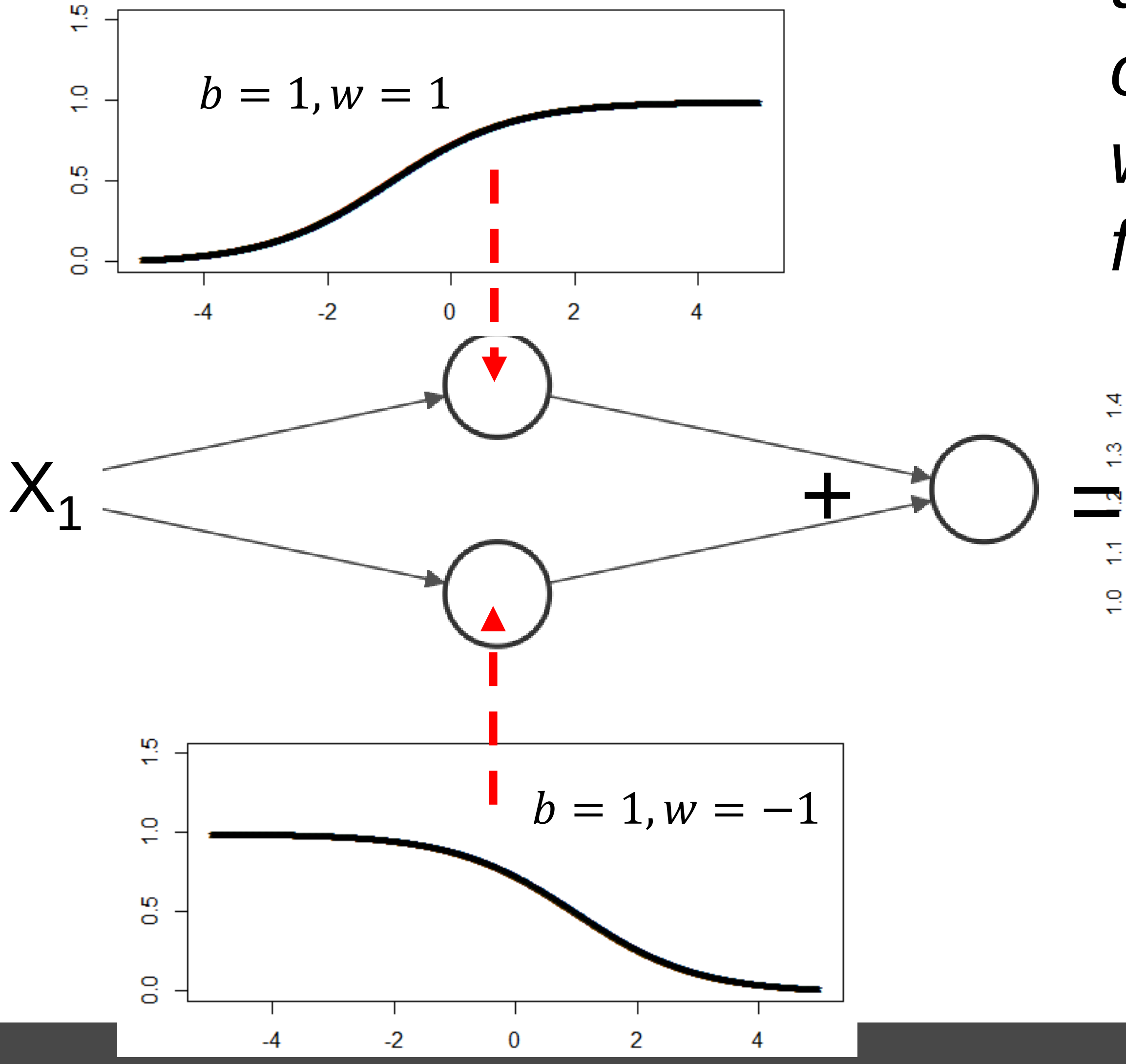
Example: 1 input into 2 logistic units with these activations

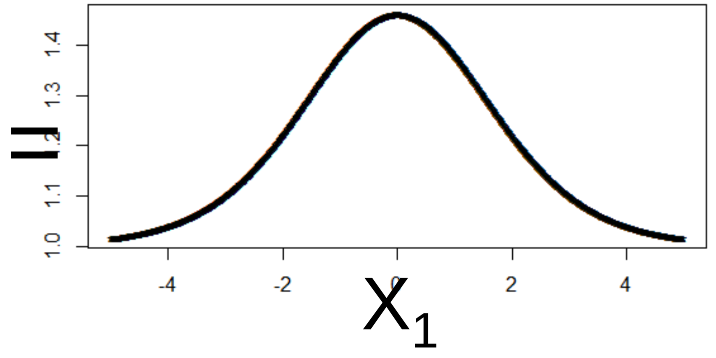
If you add these 2 units into a final output unit what would the output function look like?



Example: 1 input into 2 logistic units with these activations

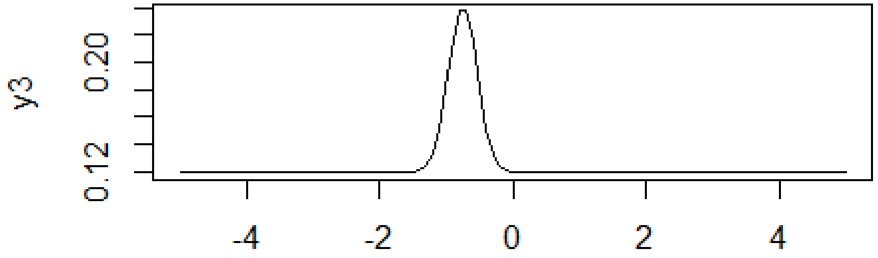
If you add these 2 units into a final output unit what would the output function look like?



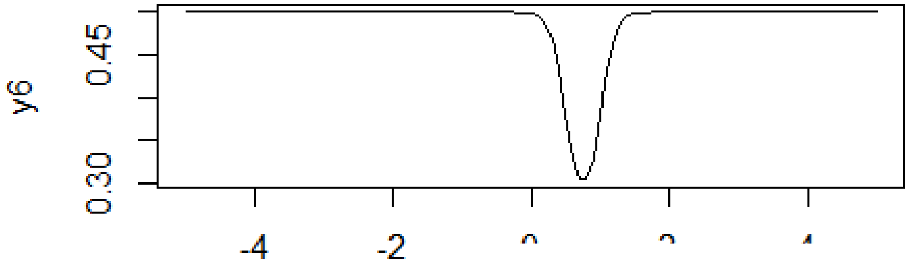


Higher level function combinations

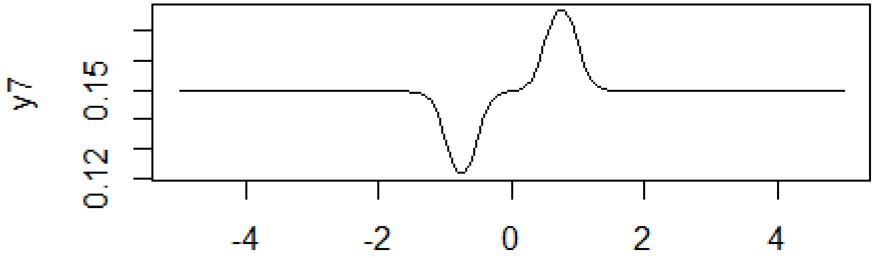
```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```




```
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")
```

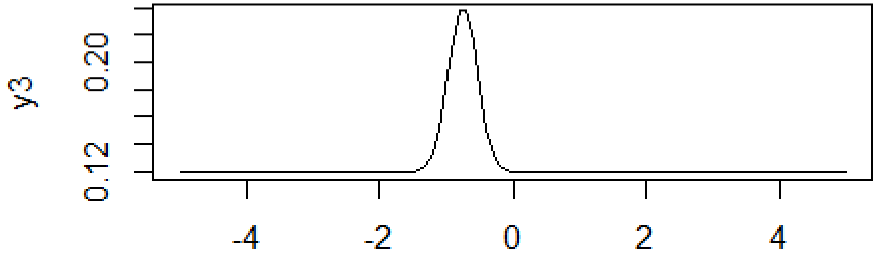



```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```



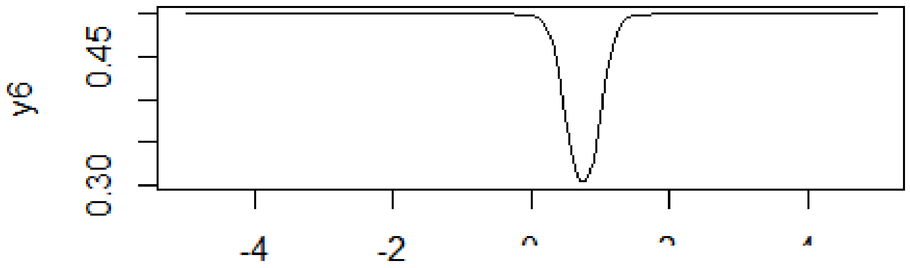
Higher level function combinations

```
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")
```

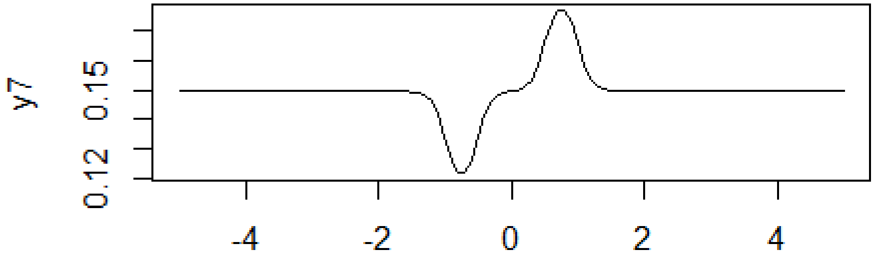


Multiple layer networks can represent any logical or real-valued functions (unbiased, but potential to overfit)

```
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")
```



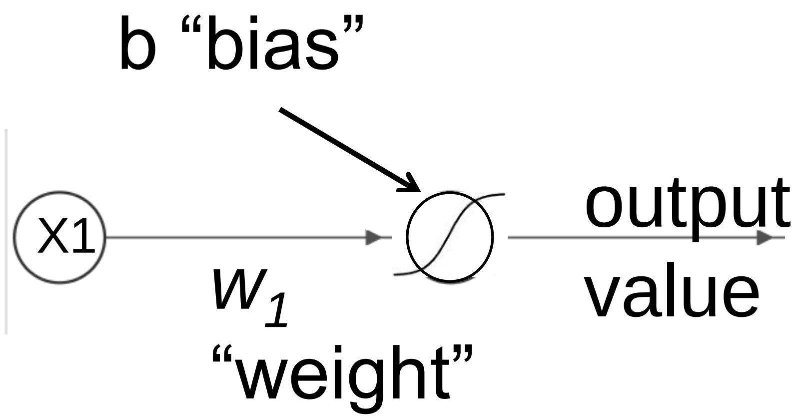
Logistic to Neural Network model

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

Draw out function as a little graph, 1 input

Logistic to Neural Network model

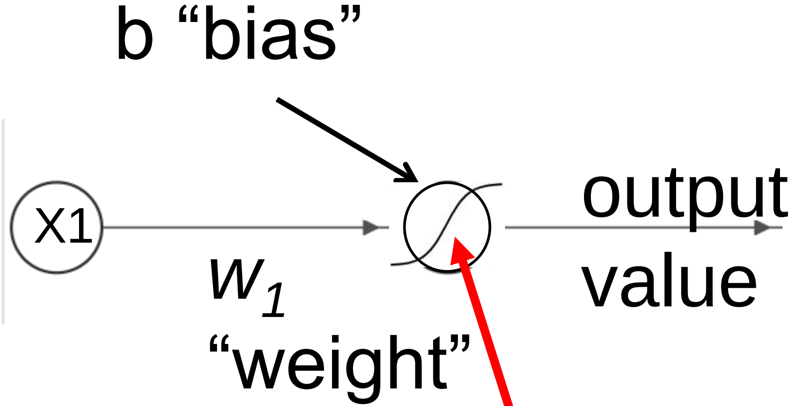
$$f(x,b,w) = \frac{\exp^{(b+w*x)}}{1+\exp^{(b+wx)}}$$



Draw out function as a little graph, 1 input

Logistic to Neural Network model

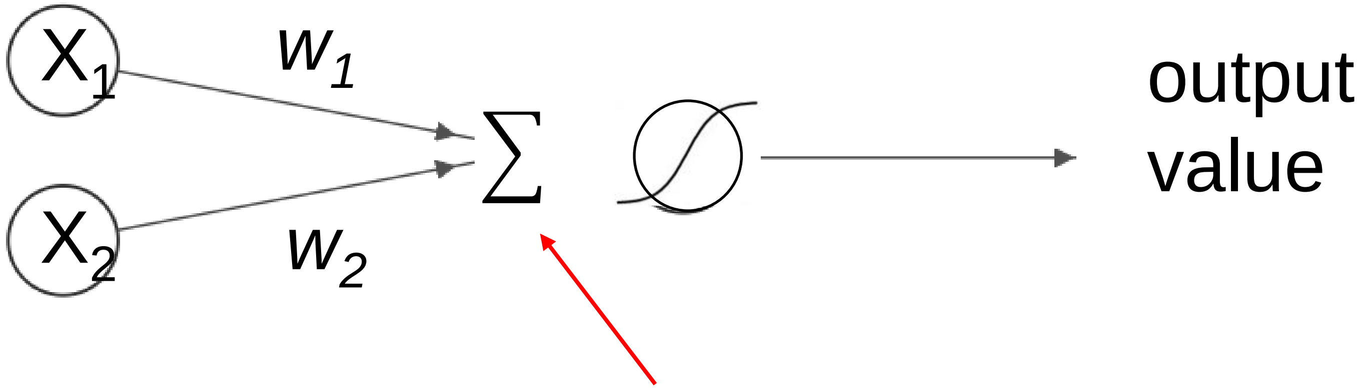
$$f(x, b, w) = \frac{\exp^{(b+w*x)}}{1+\exp^{(b+wx)}}$$



Draw out function as a little graph, 1 input

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

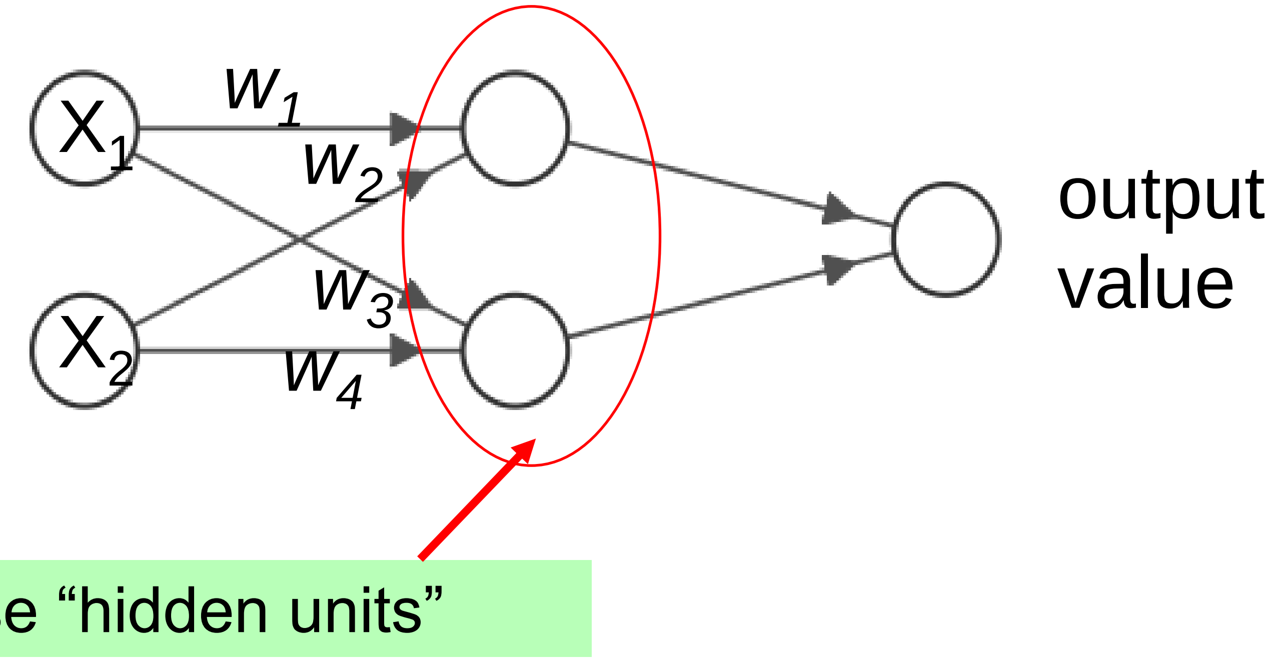
Using 2 input units, the graph model would be:



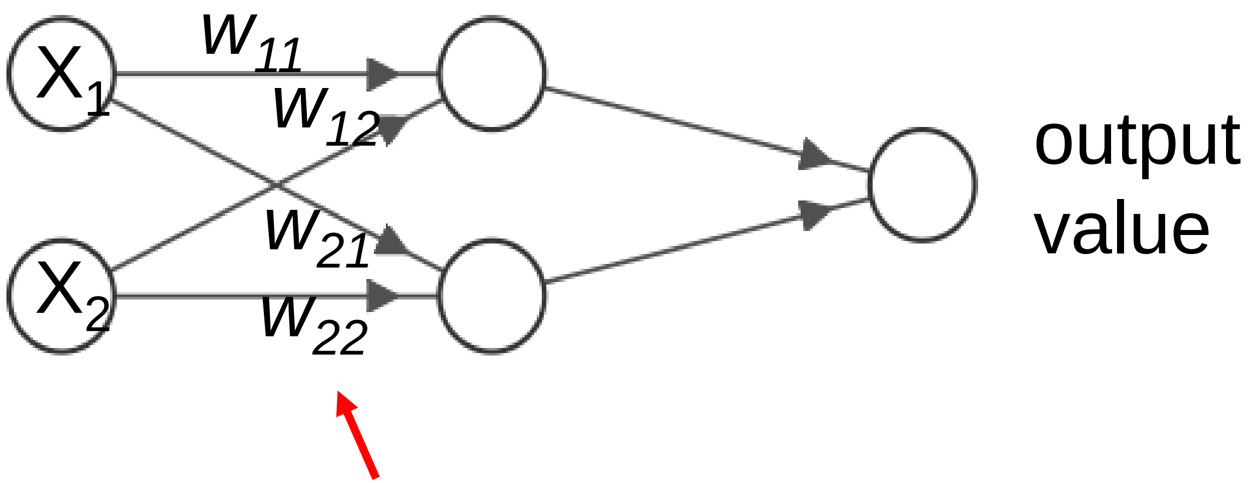
$$b + w1\ x1 + w2\ x2$$

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:

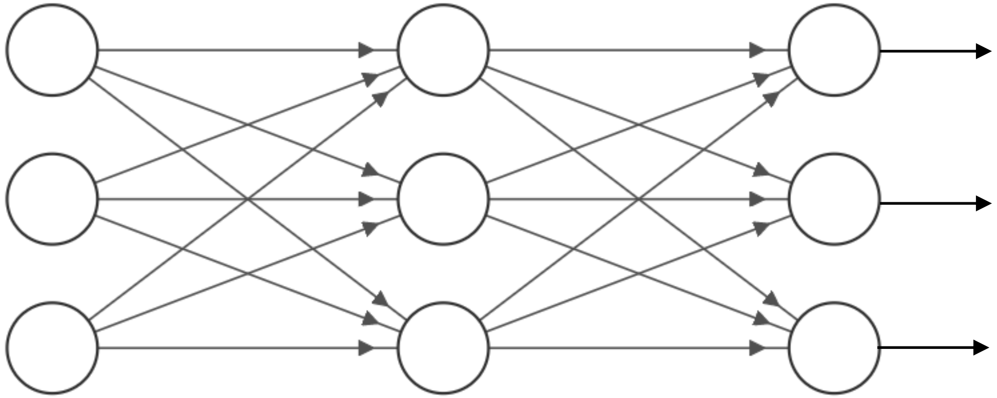


Using 2 input units, 2 intermediate units, and 1 output:



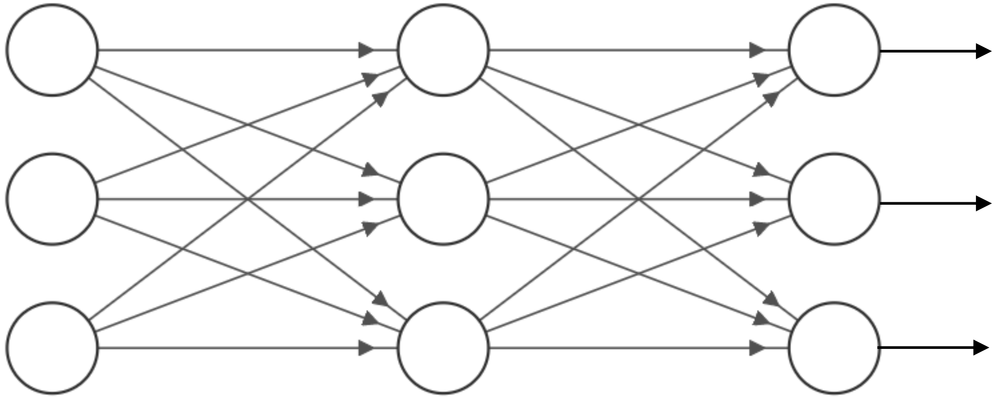
For X a Px1 vector, we can set up indices in a weight matrix so that: W*X = incoming activations from previous layer

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

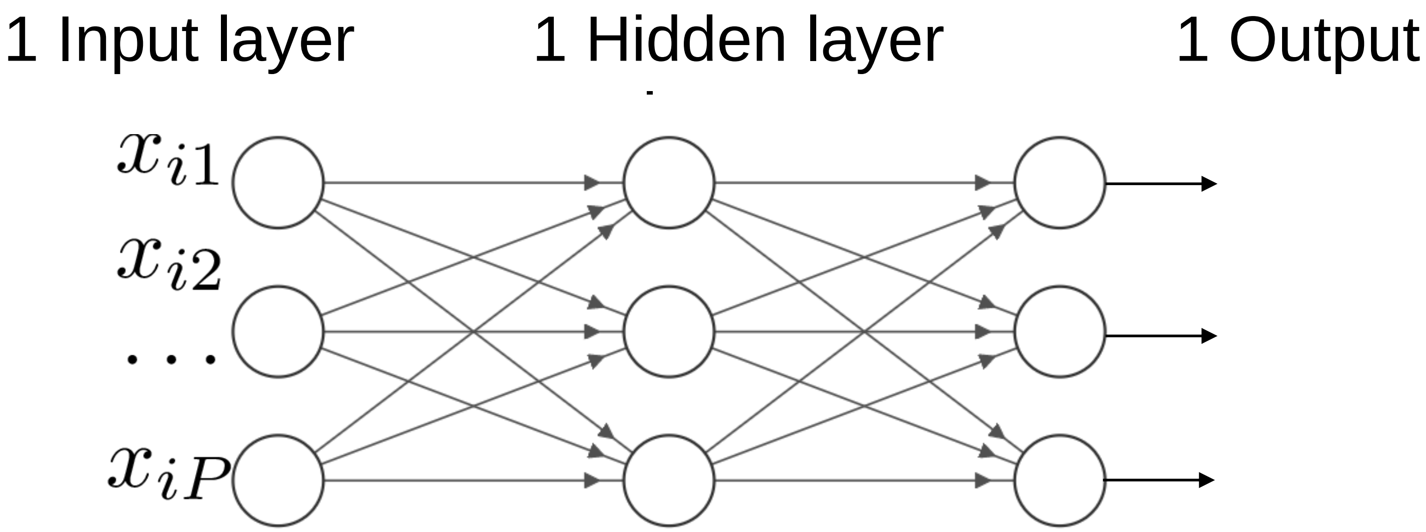


A “Multilayer Perceptron”

1 Input layer   1 Hidden layer   1 Output



A "Multilayer Perceptron"



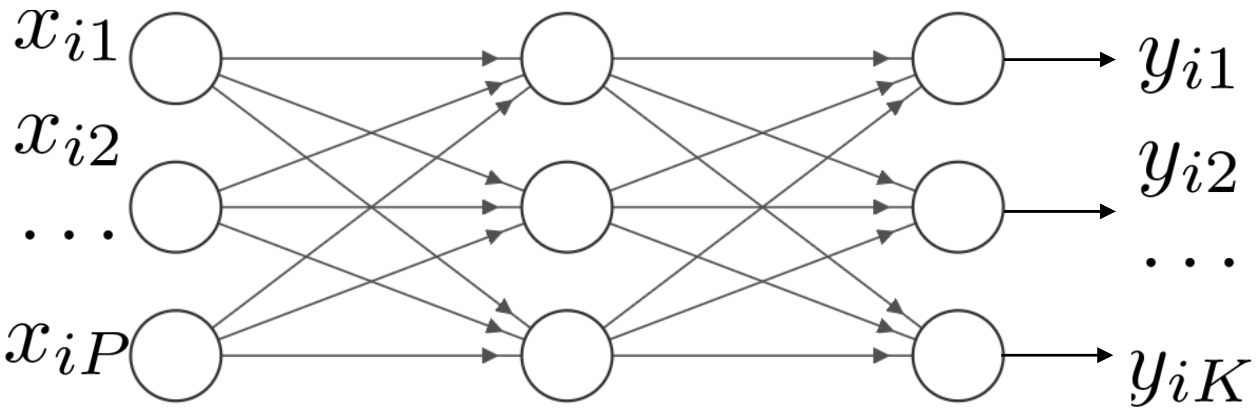
P input units for each input i

A “Multilayer Perceptron”

1 Input layer

1 Hidden layer

1 Output



P input units for each input i

M hidden units

K output units

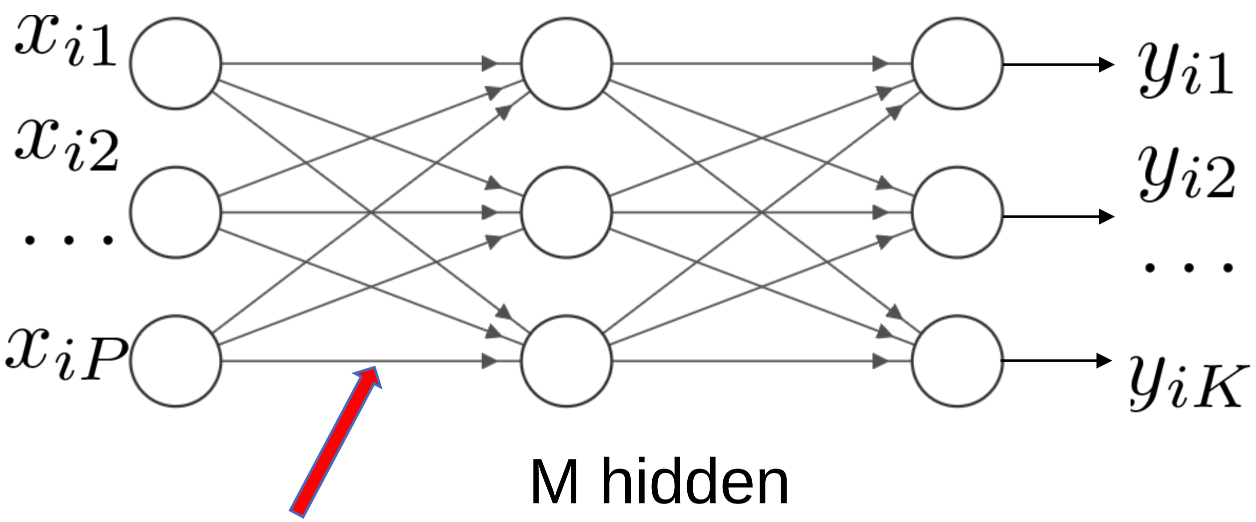
A “Multilayer Perceptron”

1 Input layer

1 Hidden layer

1 Output

P input units for each input i

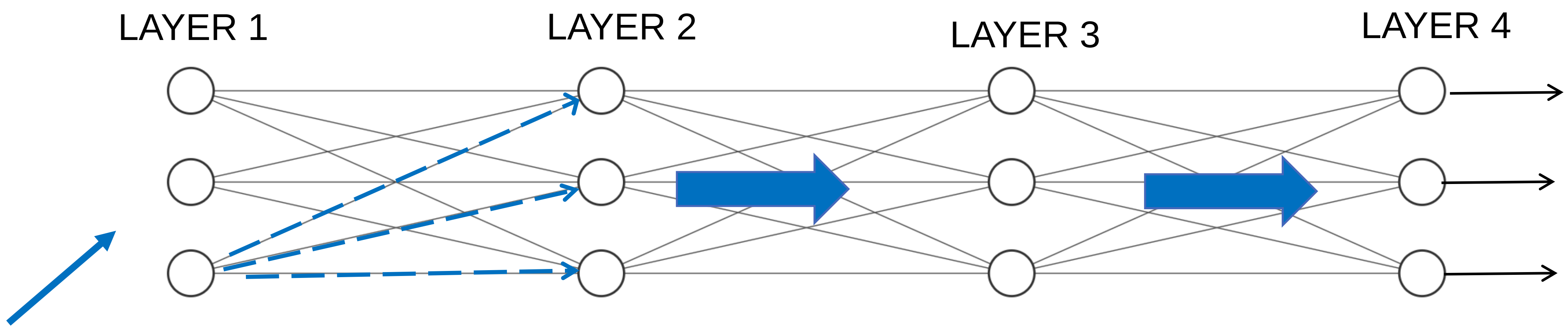


K output units

M hidden units

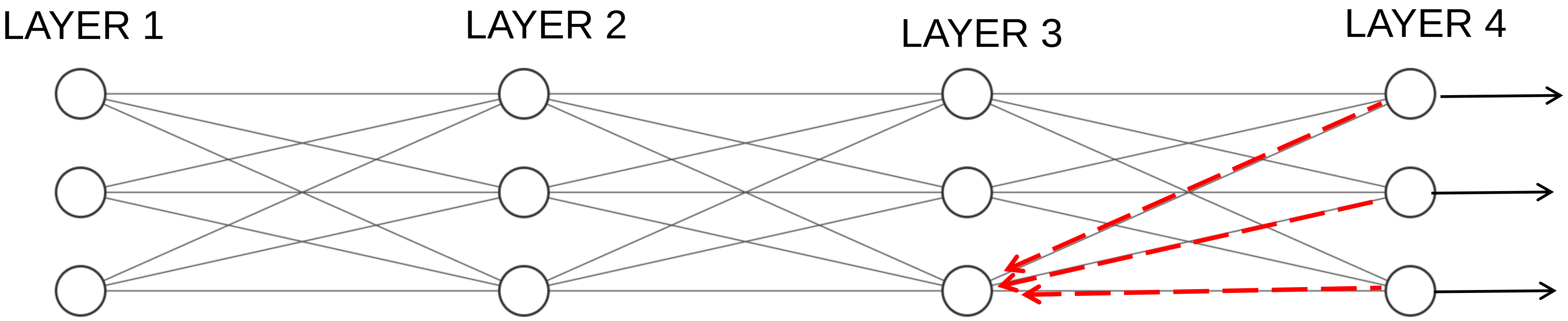
1 weight parameter for each connection

Algorithm steps



1. FORWARD PROPAGATE AN ENTIRE BATCH OF INPUTS

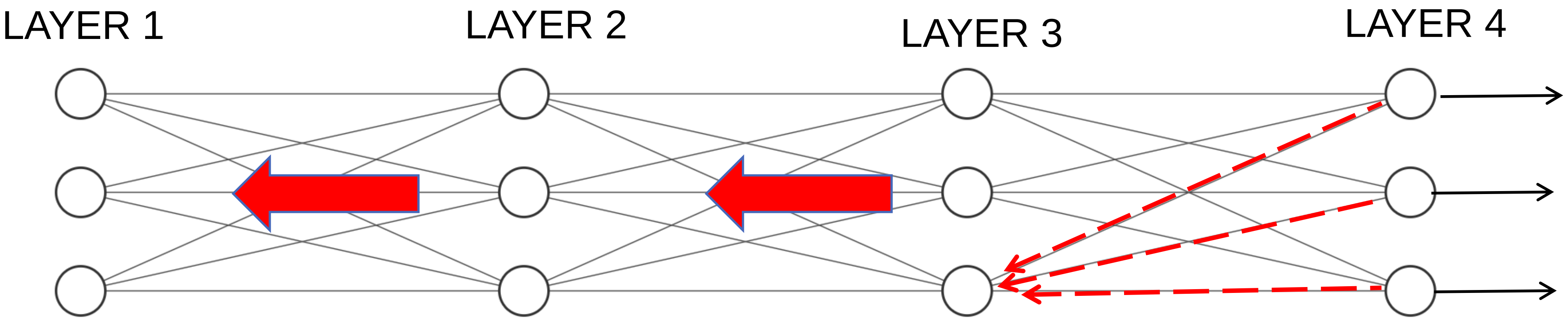
Algorithm steps



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

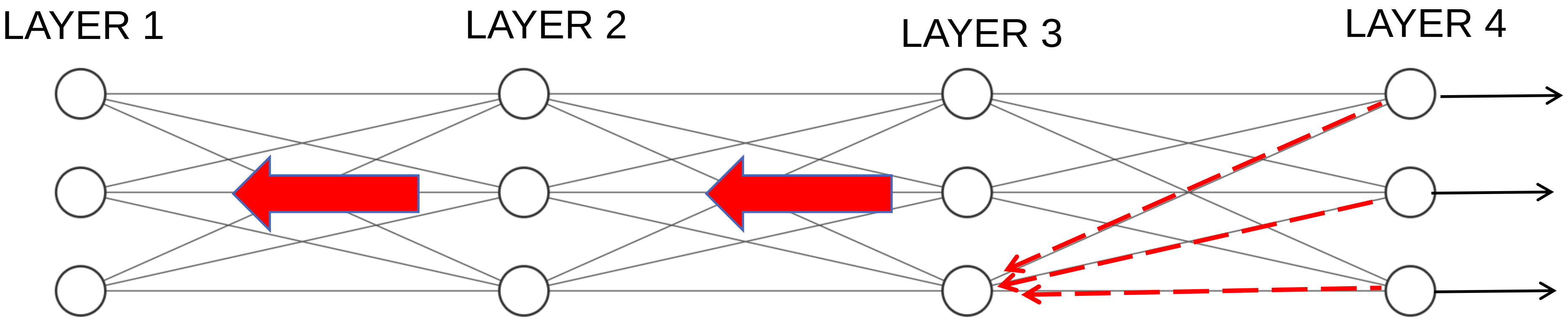


2. BACKWARD PROPAGATE ERROR FOR WHOLE BATCH USING DERIVATIVE CHAIN RULE:

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

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 …

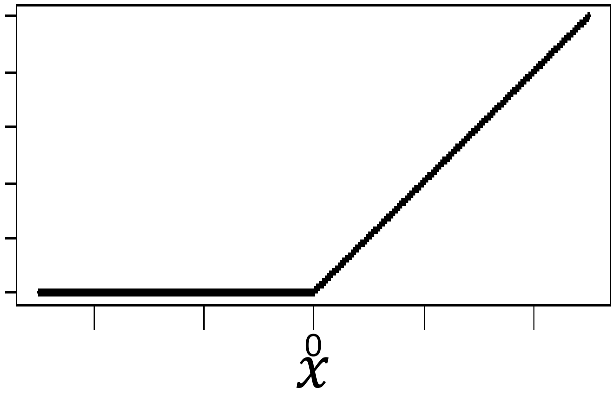
The rectified linear unit (RELU)

RELU activation function

It is unscaled (bad!)

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

RELU (rectified linear unit)



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

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

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

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: $w \leftarrow w - \; learning\_rate * \dfrac{dL}{dw}$

[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 $Y \in (-\infty, +\infty)^K$	$\hat{Y} = XW$	$\hat{Y}$	Sum Squared Error (SSE)	Mean Squared Error (MSE)
Multivariate output of 0's and 1's	if $Y \in [0,1]^K$	$\hat{Y} = \frac{1}{1+exp^{-(XW)}}$	1 or 0	SSE	MSE
Binary Classification	if $Y \in \{0,1\}$	$\hat{Y} = \frac{1}{1+exp^{-(XW)}}$	A probability given by $\hat{Y}$: $P(y=1\|x)$	Cross Entropy $L = -ylog(\hat{y}) - (1-y)(log(\hat{y}))$	Accuracy, ROC
Multiclassification	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’

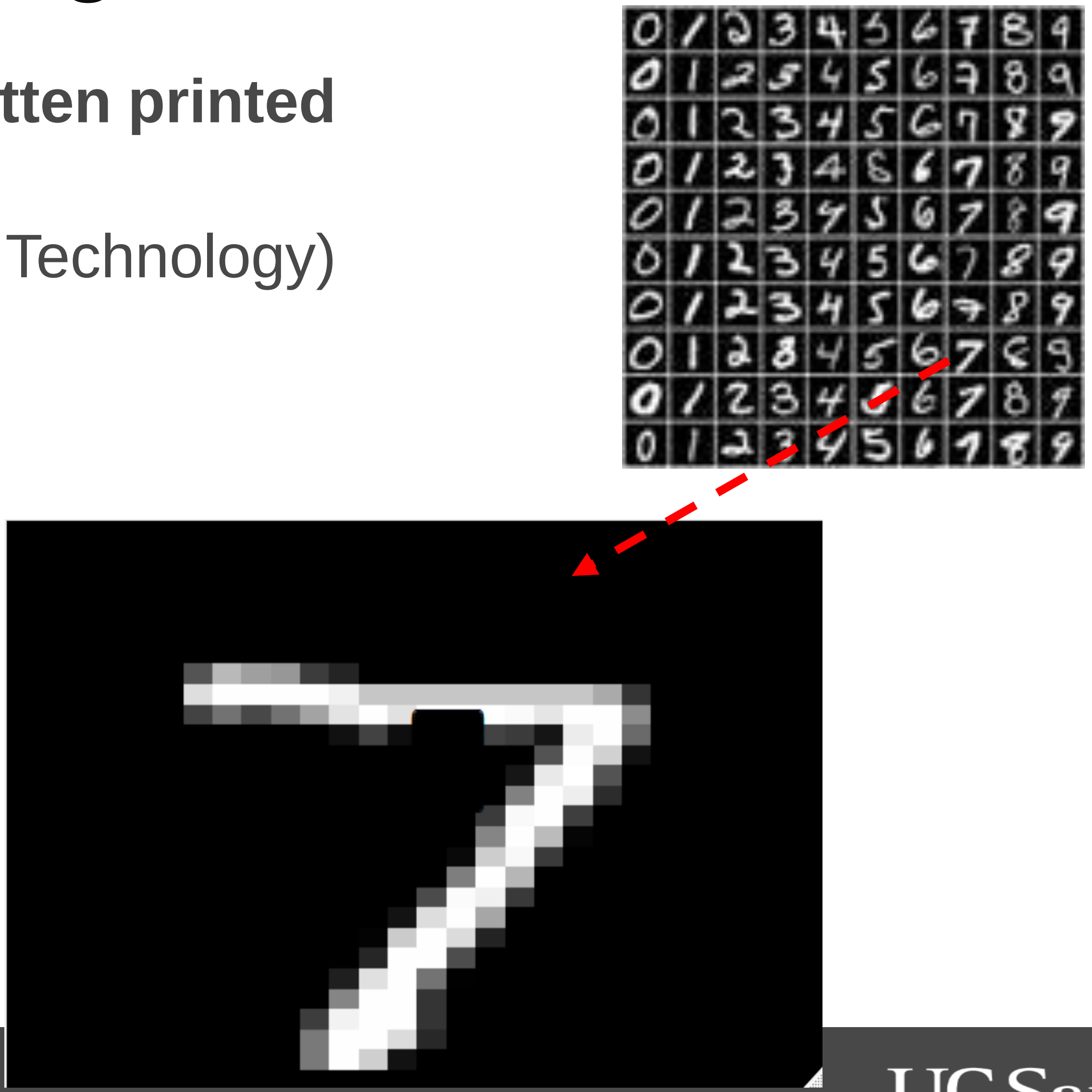
Next: Image classification of digits

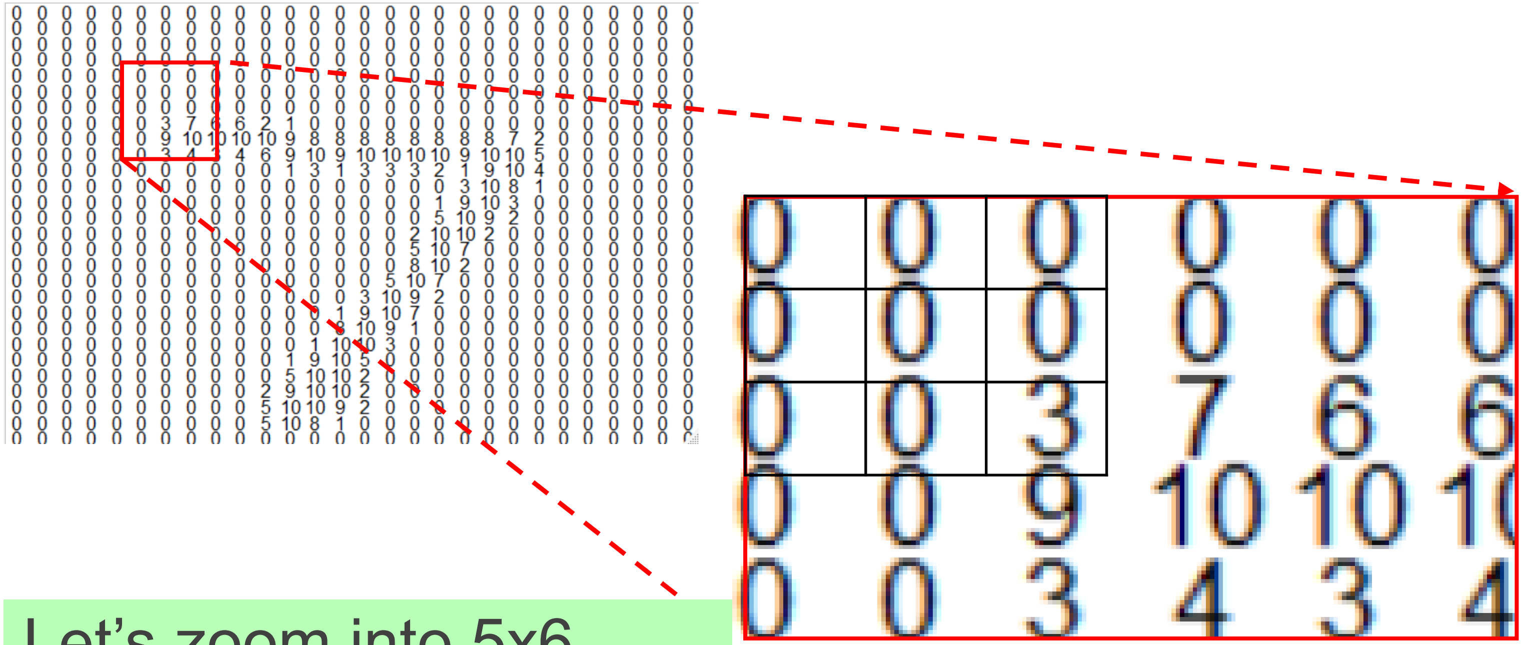
Image features

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

  (National Inst. of Standards and Technology)

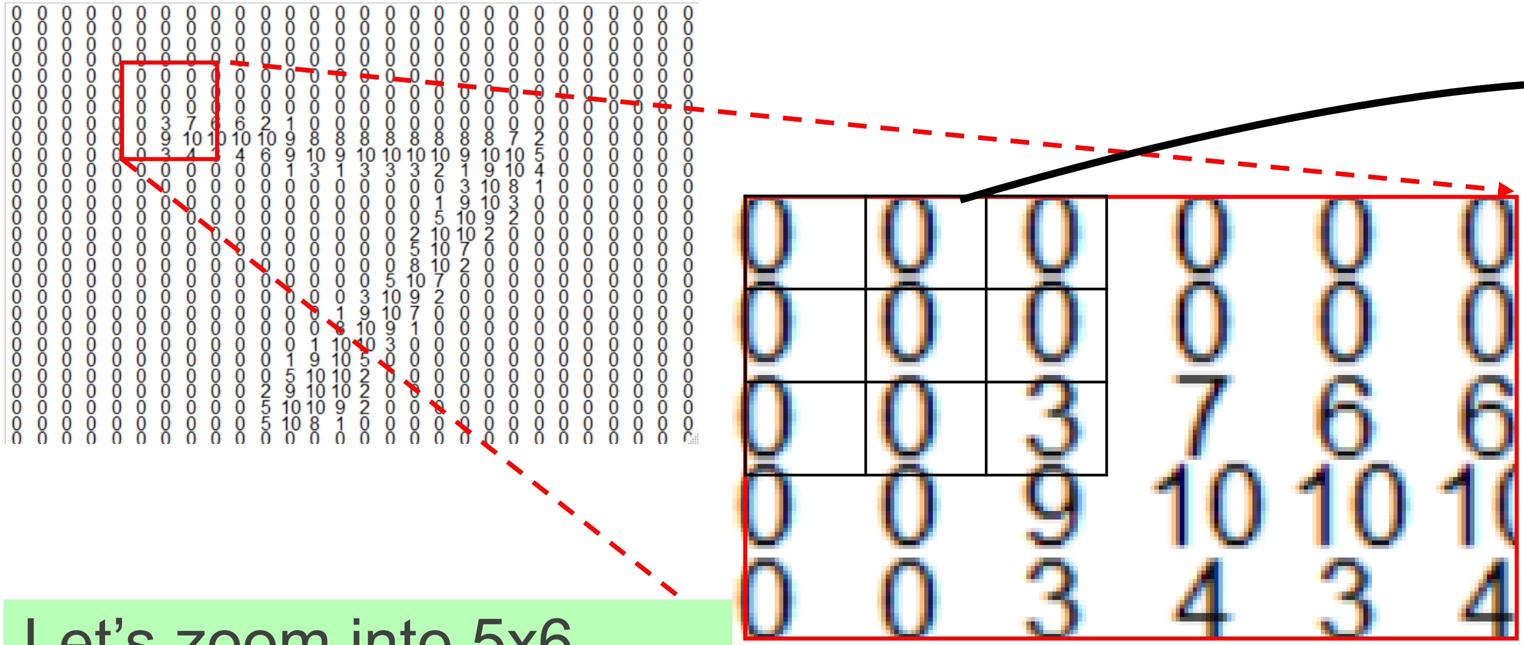
How to classify digits?





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



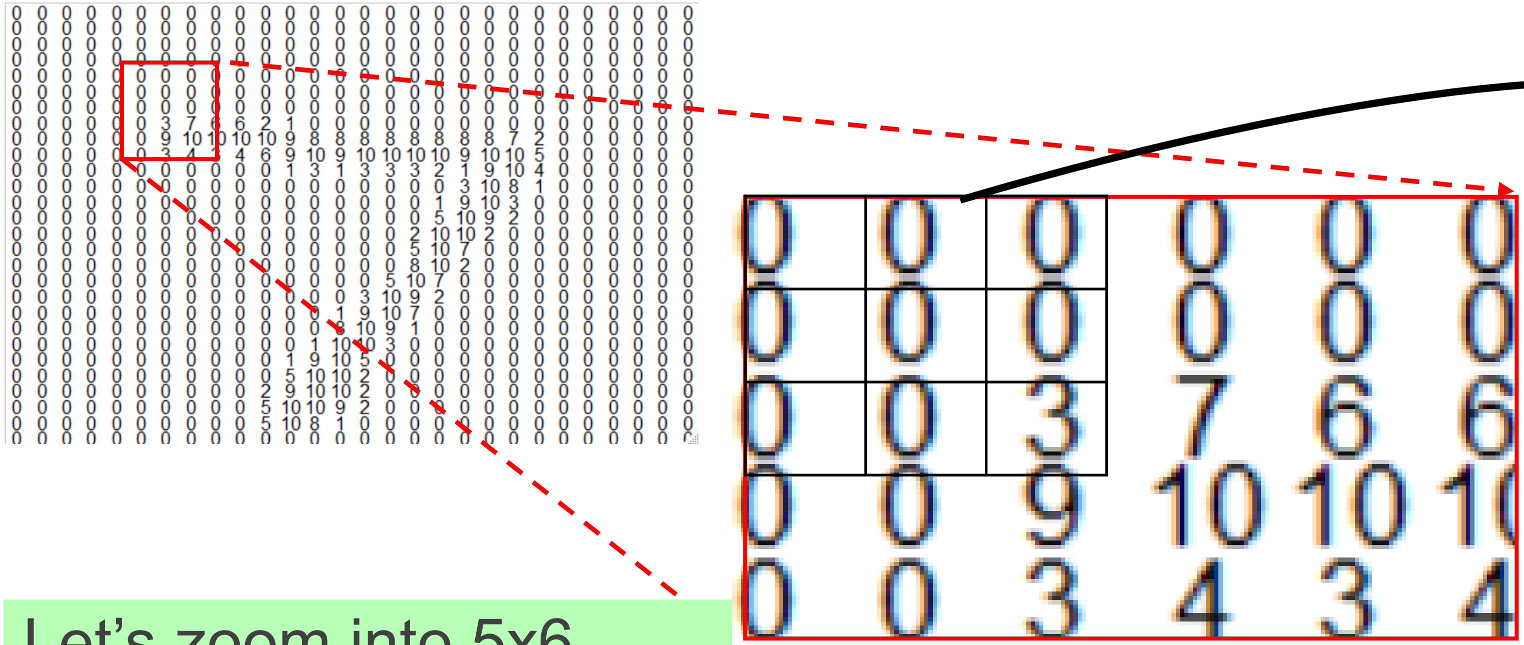
X

-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



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

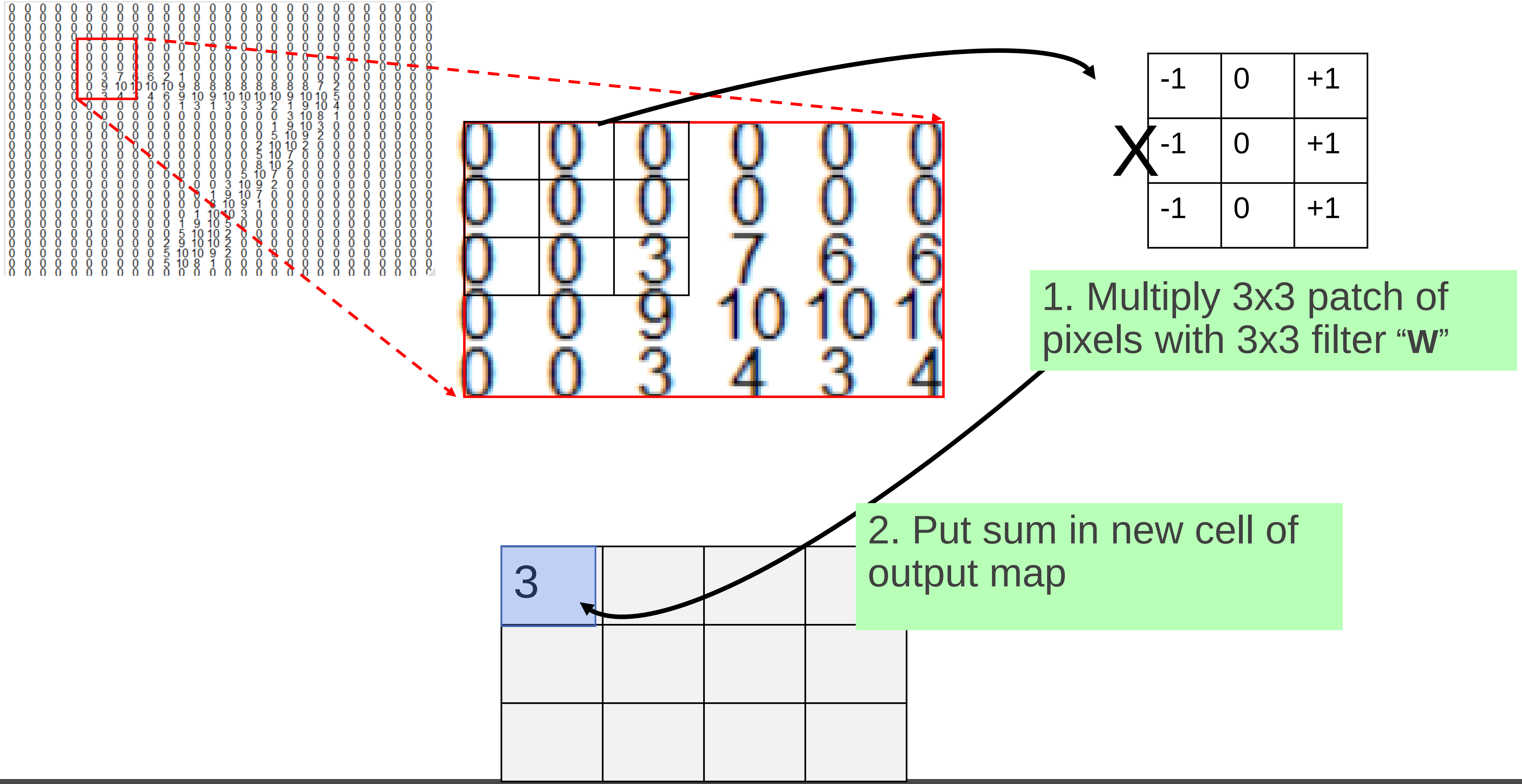
Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge

(our weight parameters)

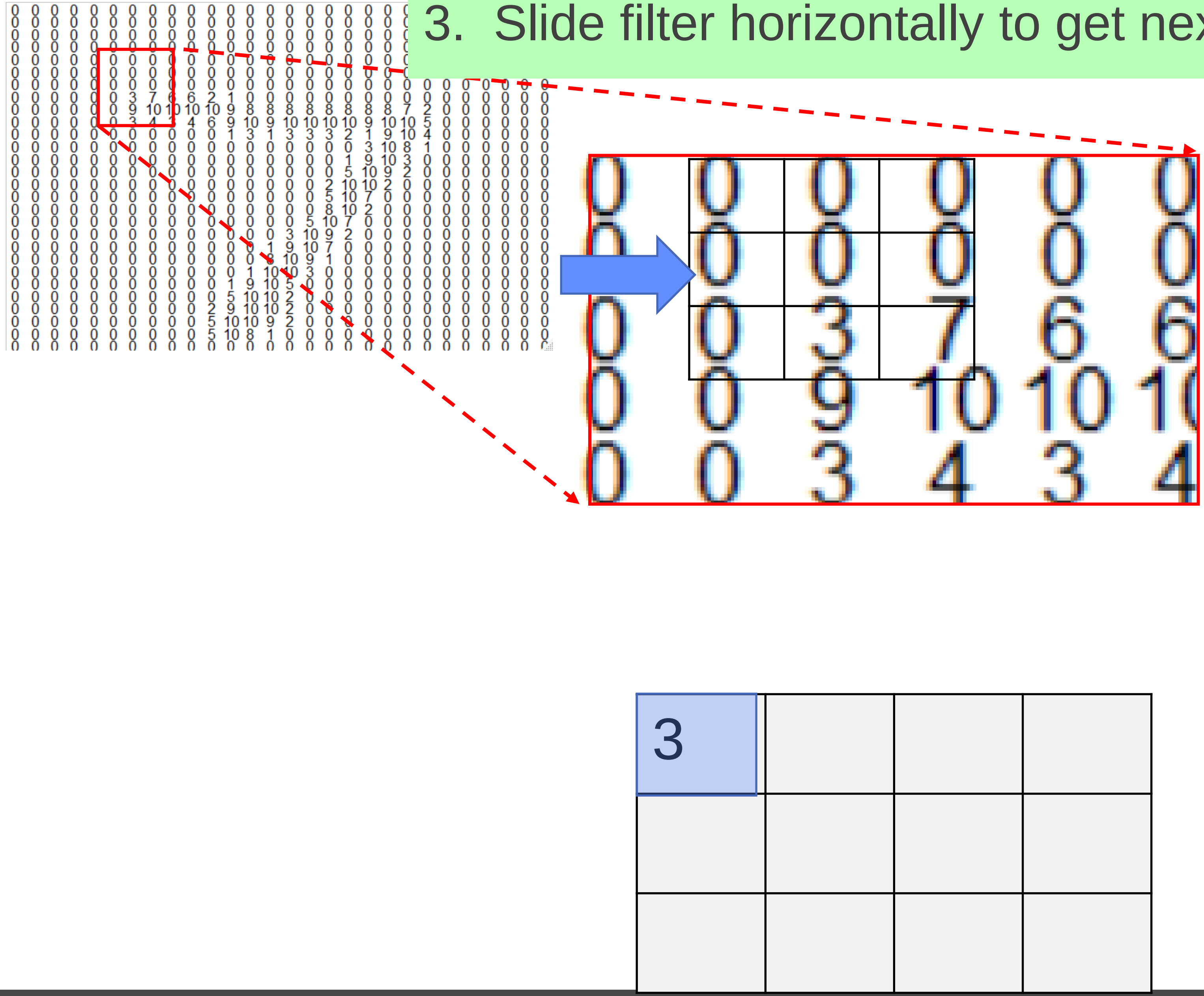
X

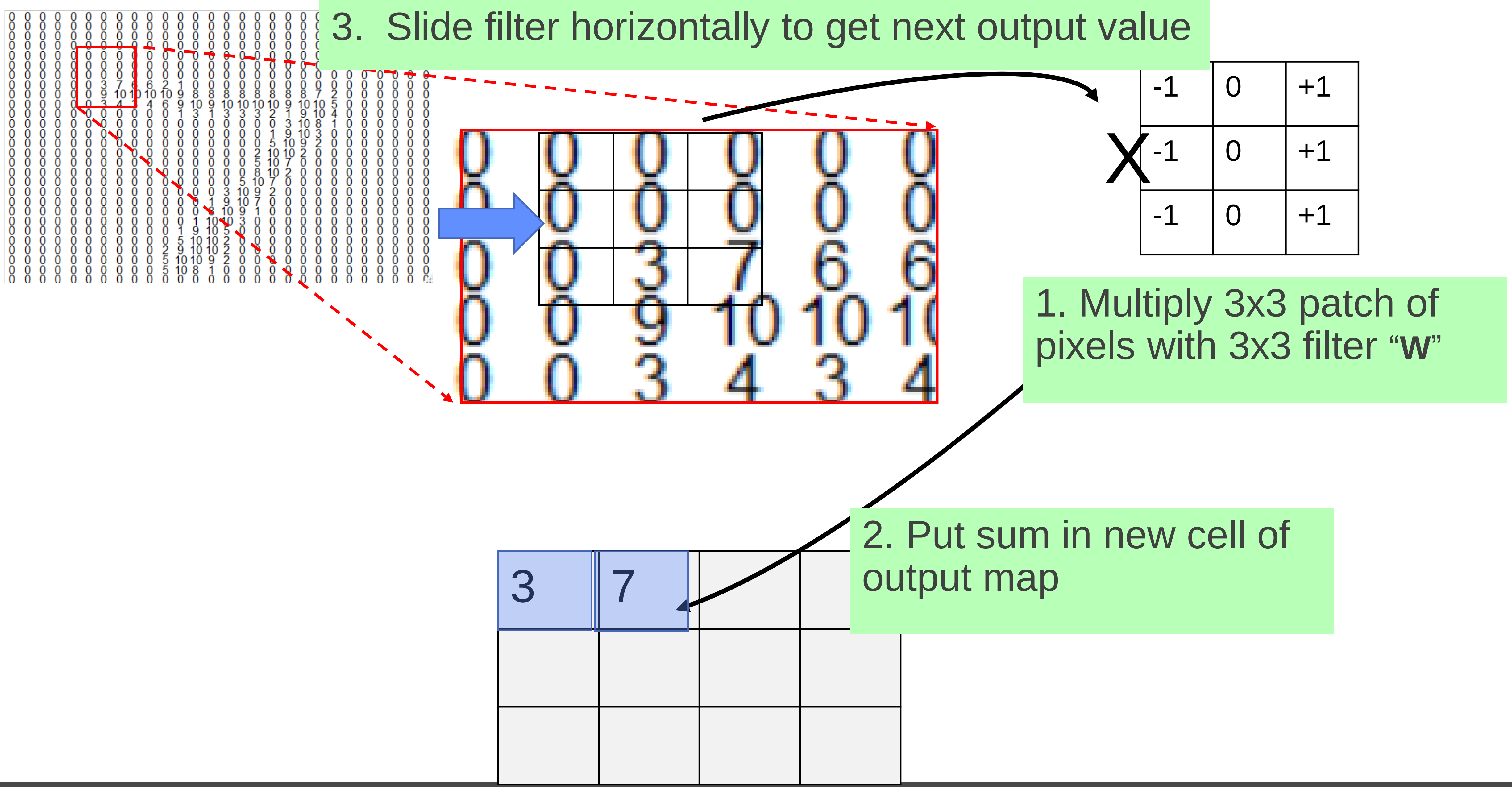
-1	0	+1
-1	0	+1
-1	0	+1

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

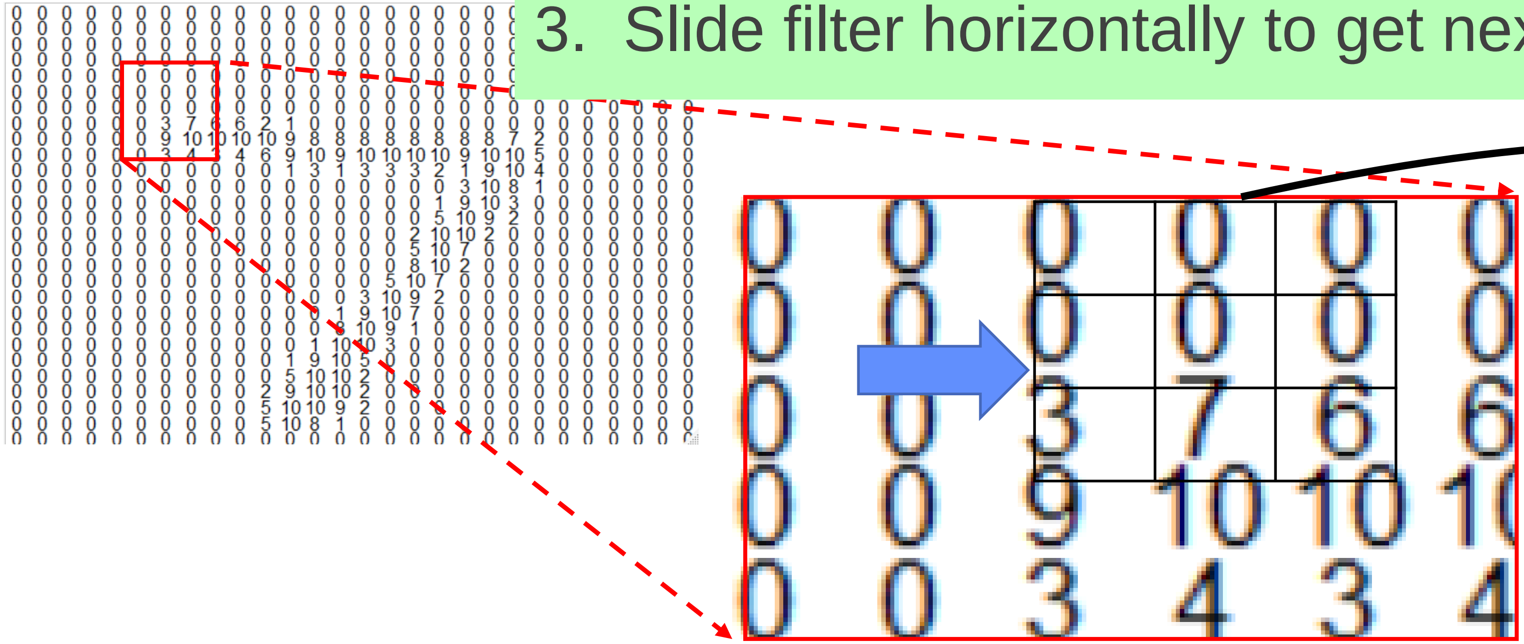

0 0 0 0 0 0
0 0 0 0 0 0
0 0 3 7 6 6
0 0 9 10 10 1(
0 0 3 4 3 4
X
-1 0 +1
-1 0 +1
-1 0 +1
1. Multiply 3x3 patch of pixels with 3x3 filter "**W**"
2. Put sum in new cell of output map
3

3. Slide filter horizontally to get next output value




3. Slide filter horizontally to get next output value
1. 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	0	+1
-1	0	+1
-1	0	+1

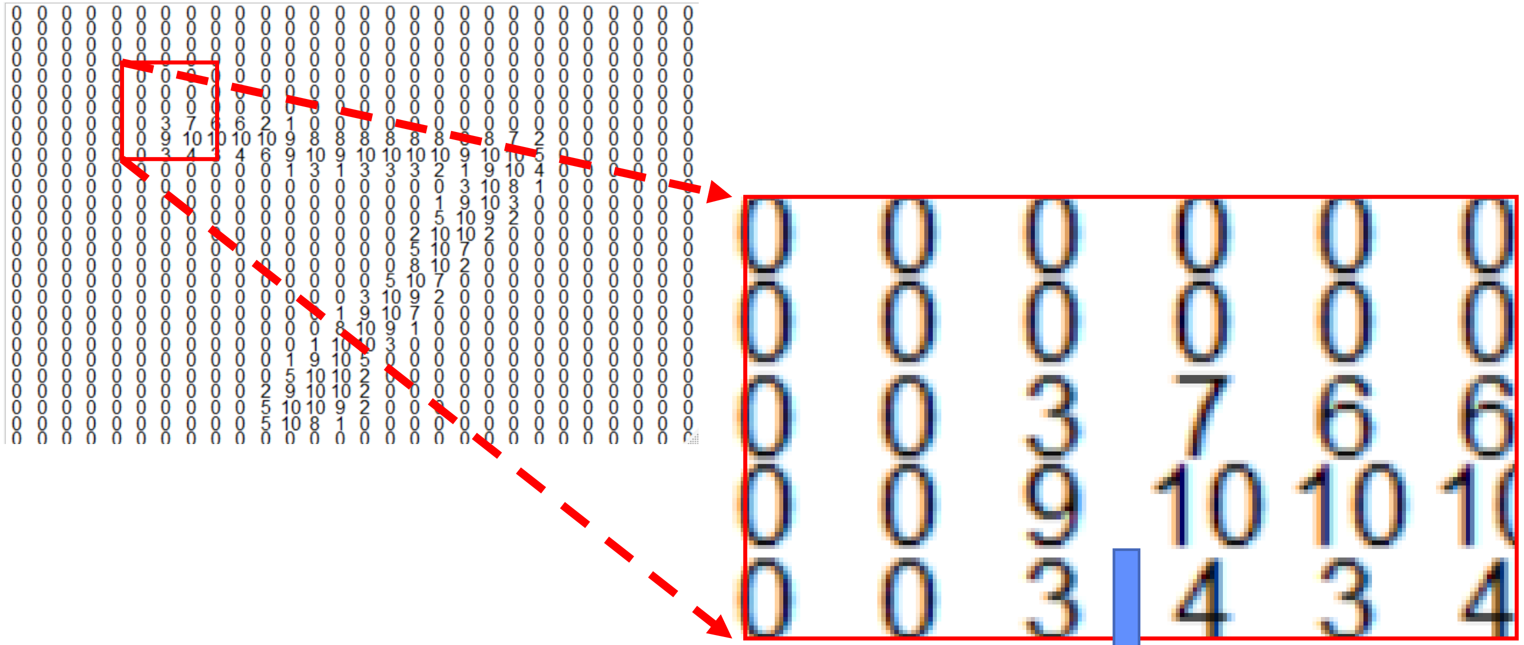
X

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

2. Put sum in new cell of output map

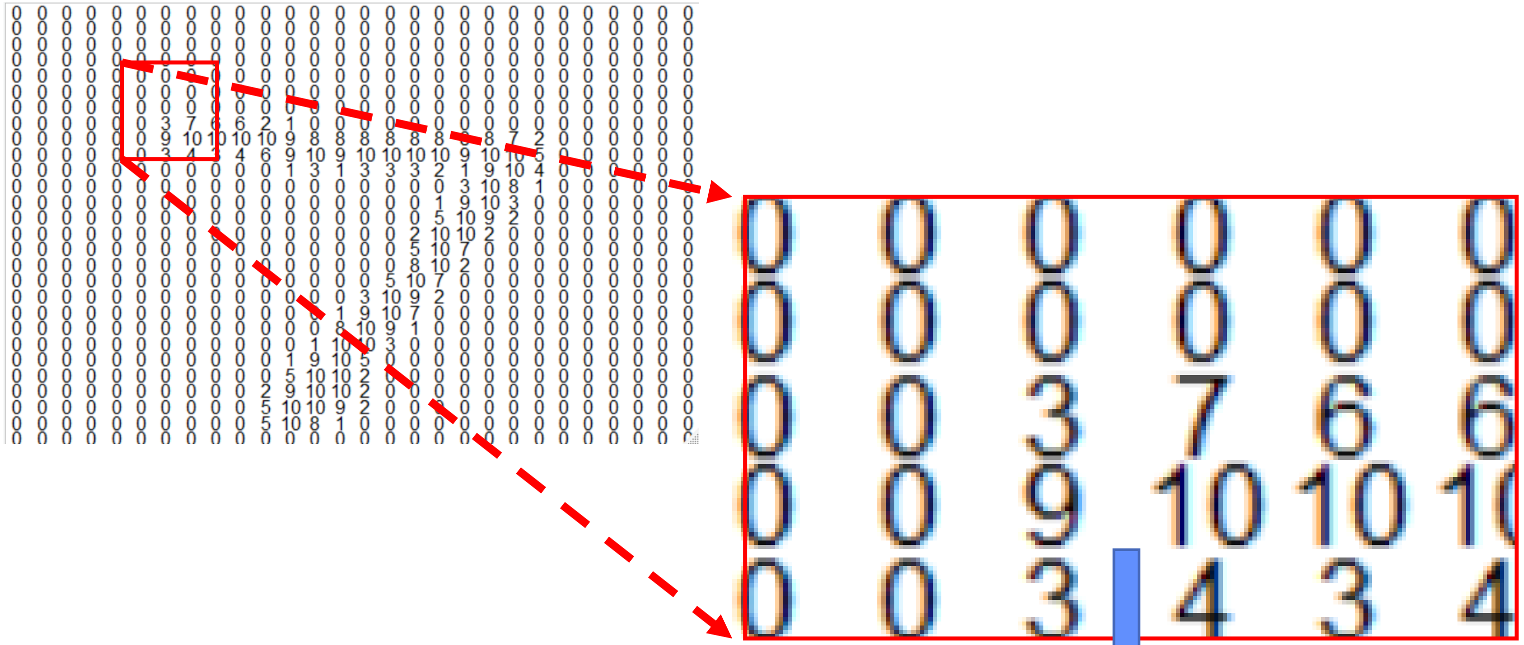
3	7	3	

NOTE: sliding a filter is known as a "convolution" operation



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

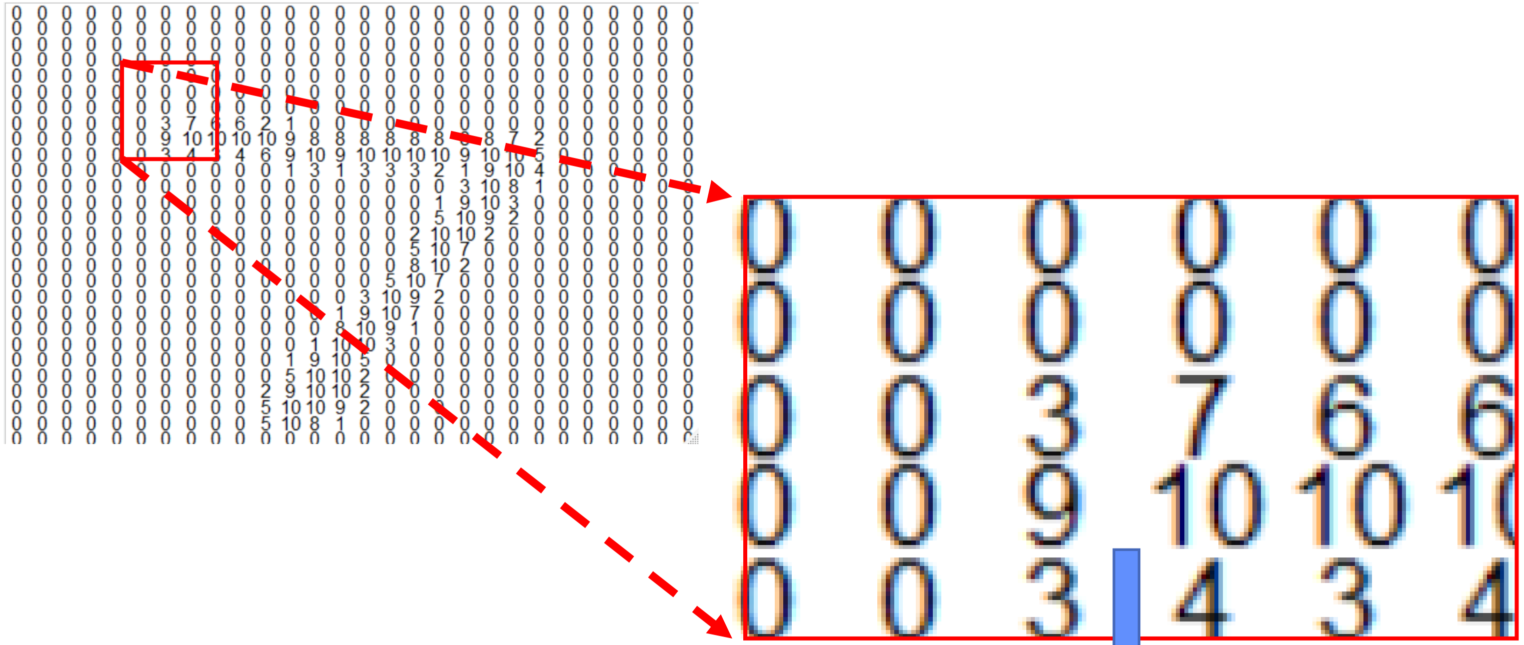
After vertical and horizontal sliding the 5x6 patch is now a 3x5 ***feature map.***



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

What do the highest values in the feature map represent?

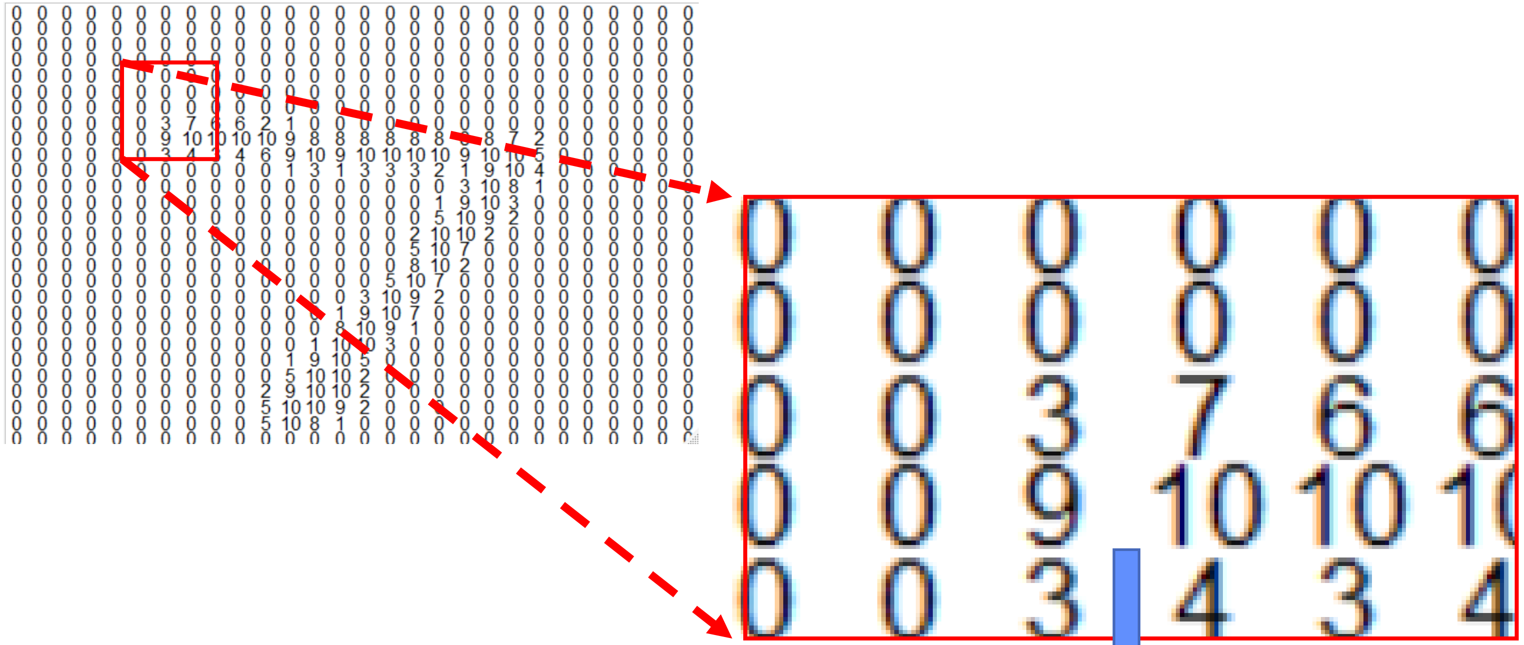
3	7	3	-1
12	17	4	-1
15	21	4	-1



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”



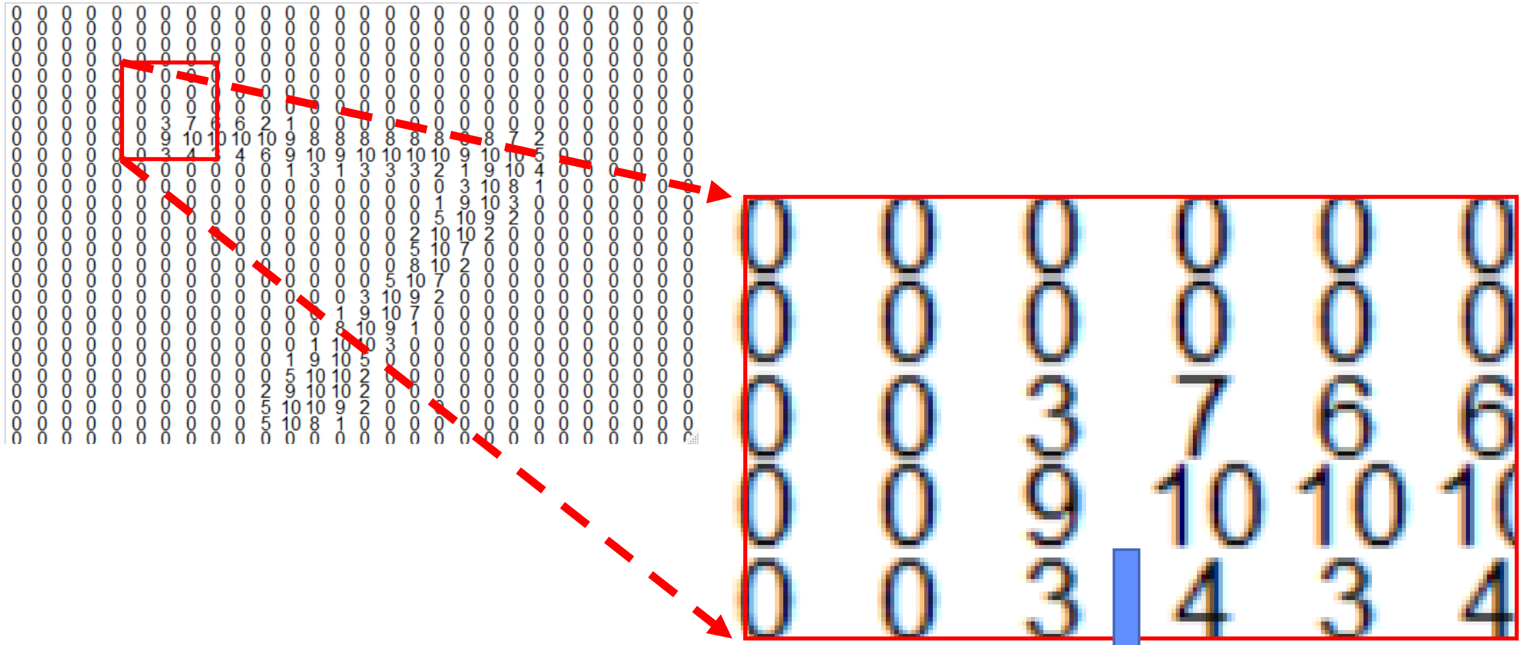
3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

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

2x2 filter has max=17

17		



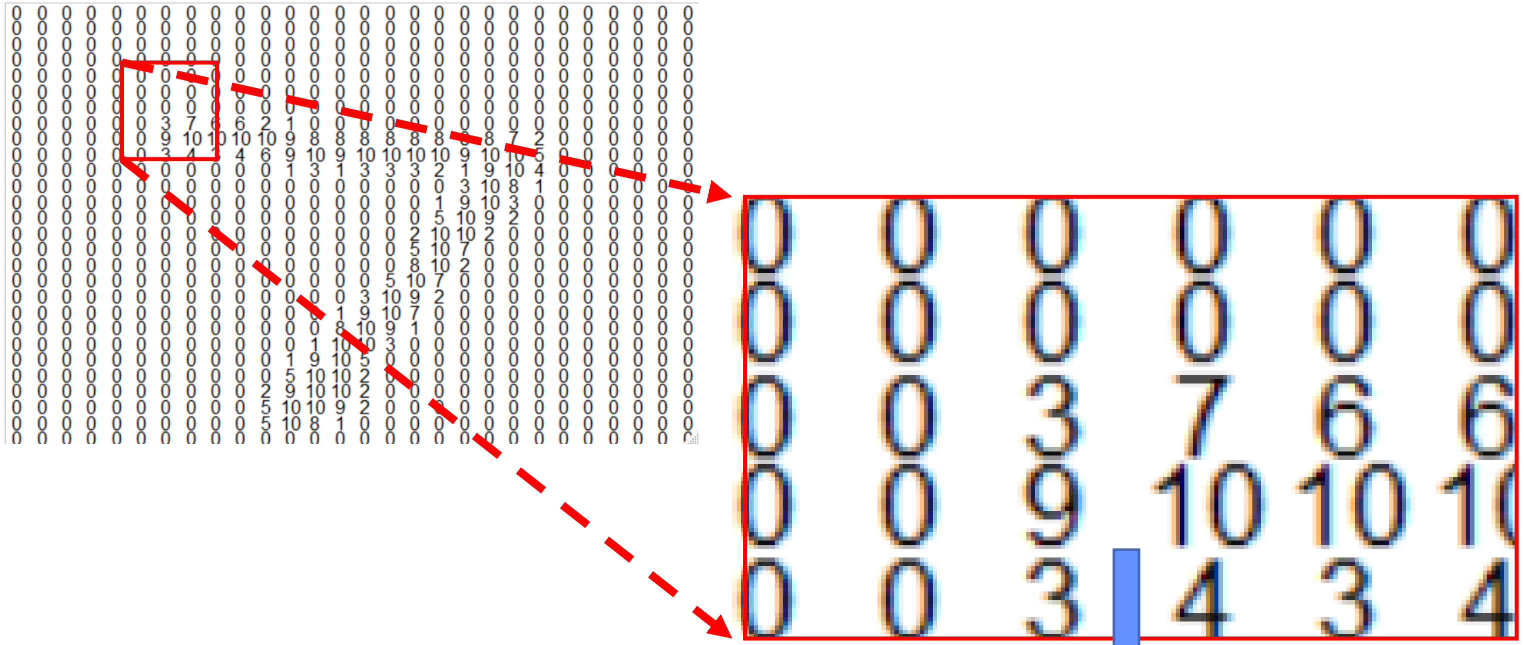
3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

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

Slide filter …

17	17	4
21	21	4

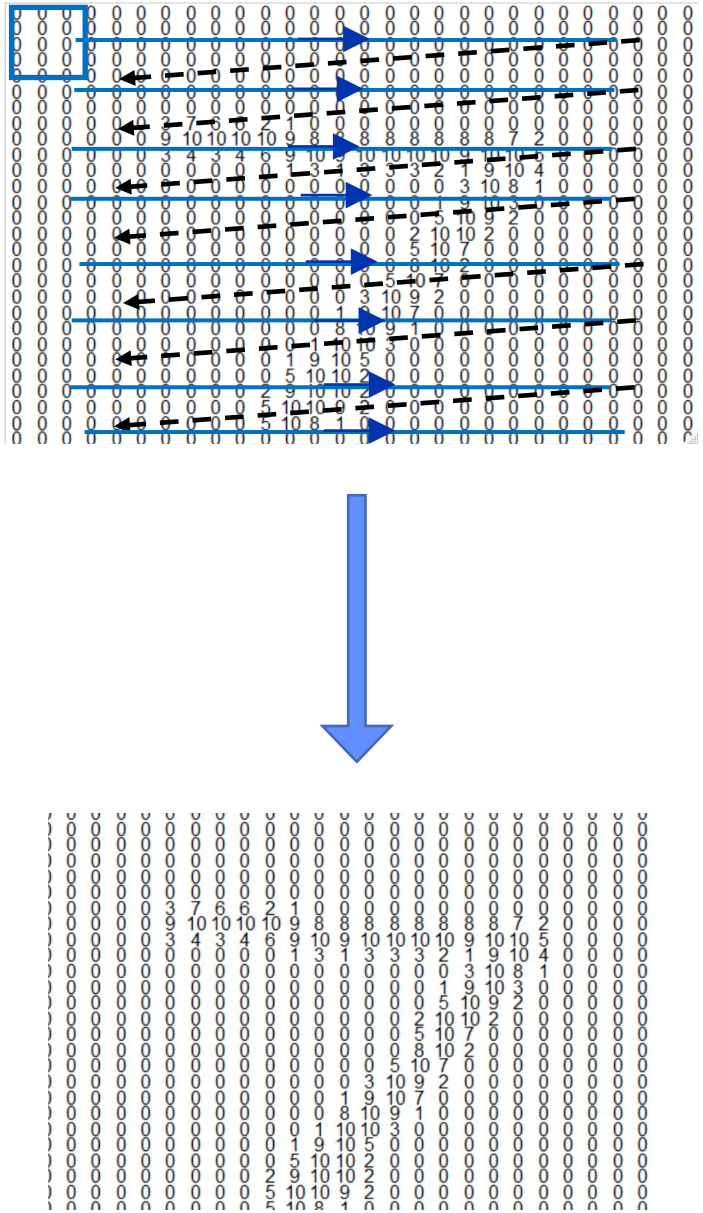


3	7	3	-1
12	17	4	-1
15	21	4	-1

*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:

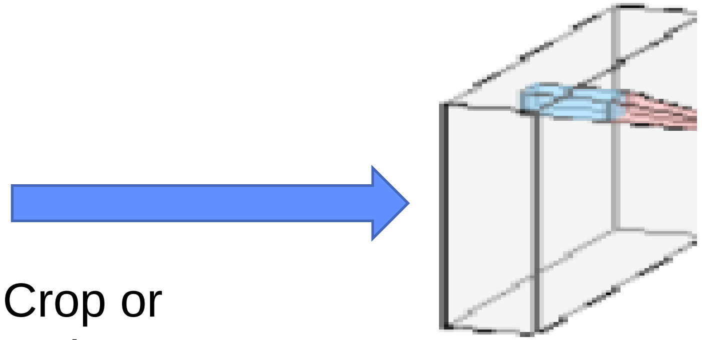
- Size of filter (smaller, like 3x3, is more general)
- Number of pixels to slide over (1 or 2 is usually fine)
- Max pooling or not (usually some pooling layers)
- Number of filters (depends on the problem!)

- **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

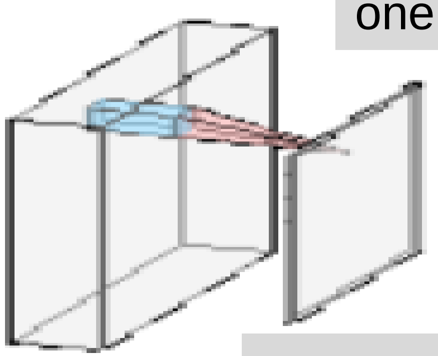


224x224x3

Convolution with image

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

Crop or resize image as needed



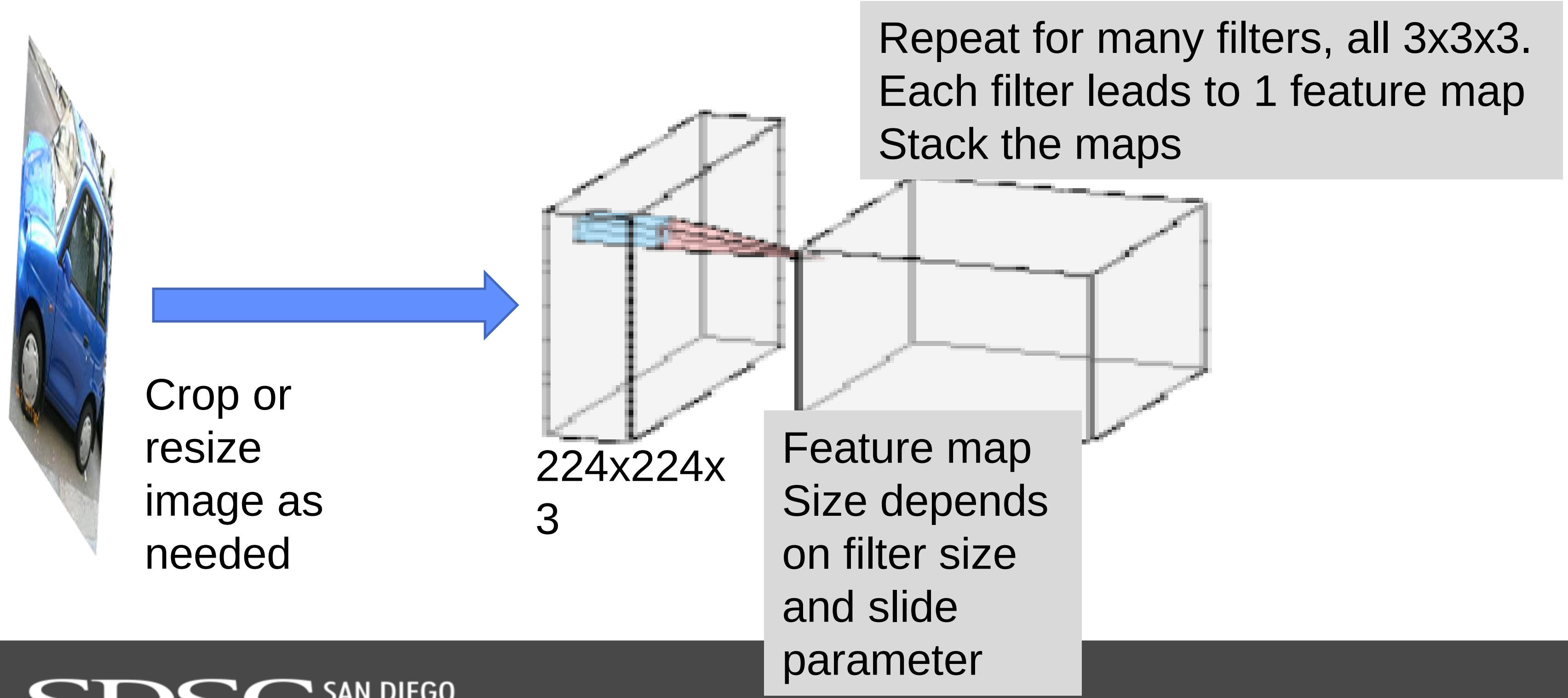
one 3x3x3 filter

224x224x3

Feature map
Size depends on filter size and slide parameter

Convolution with image

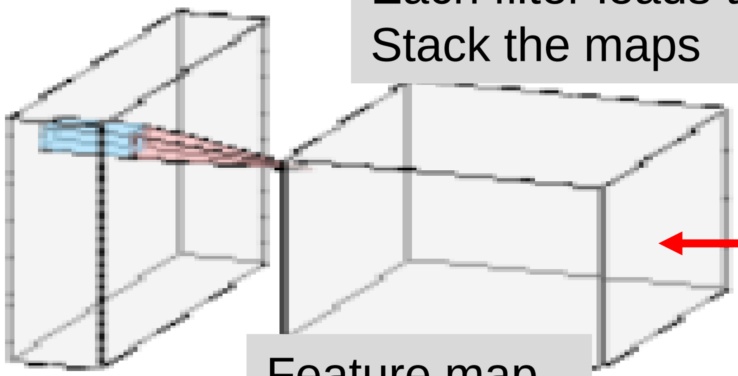
- **Make 1 layer, using HxWx3 image (3 for RGB channels)**



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

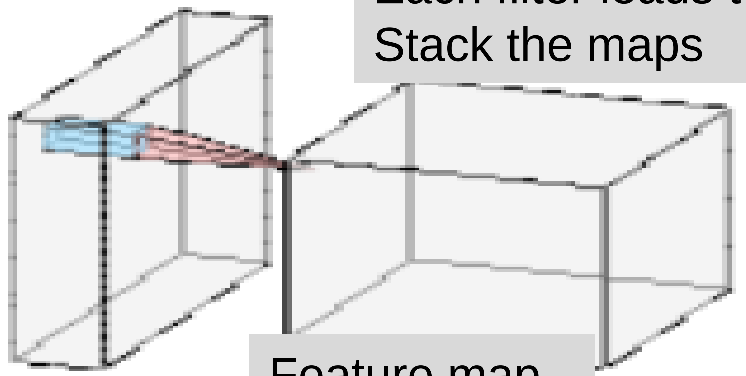
Feature map
Size depends on filter size and slide parameter

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

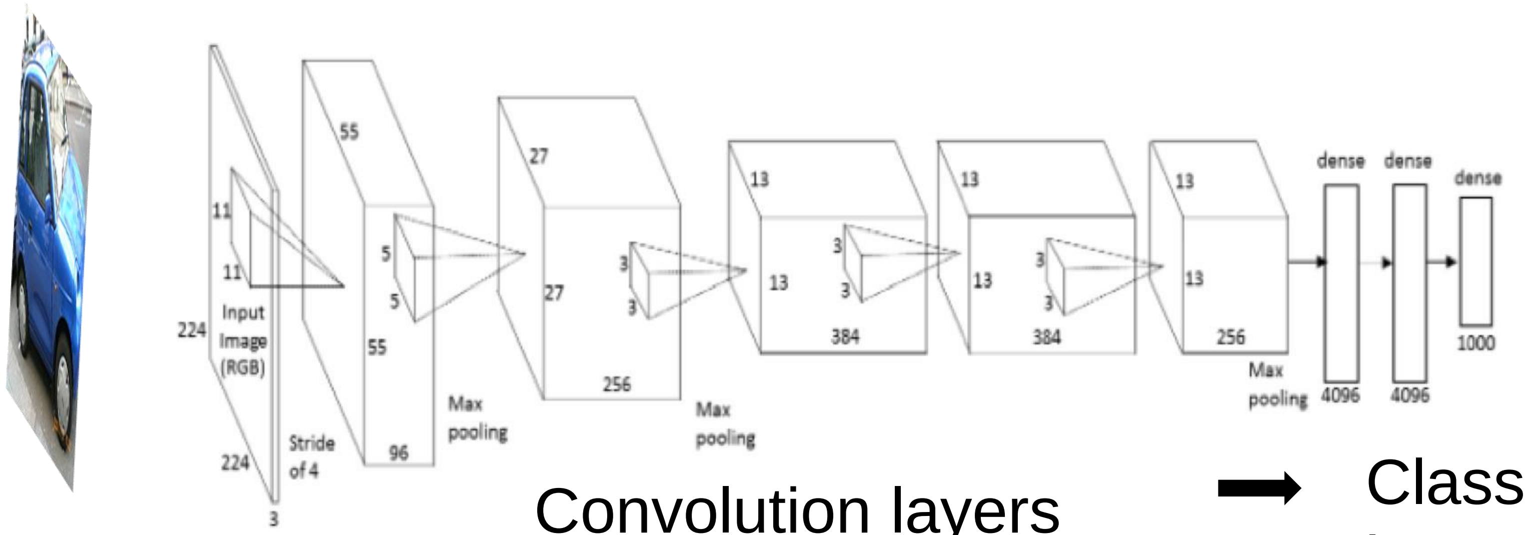
Feature map
Size depends on filter size and slide parameter

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

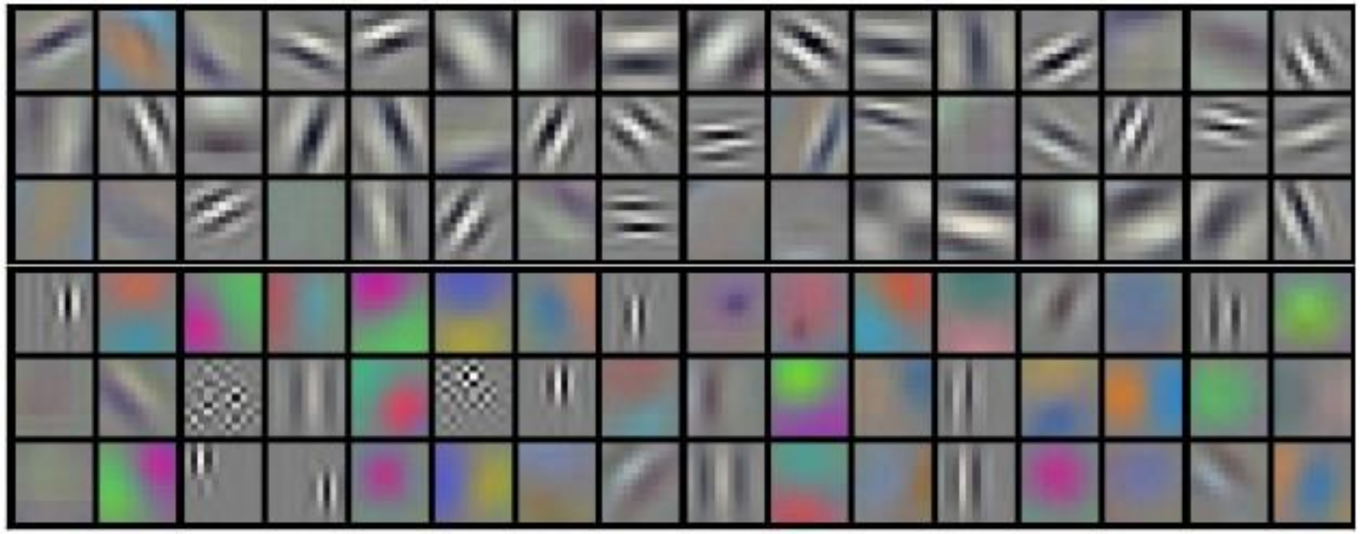
- **Large Convolution Networks – Alexnet, VGG19, ResNet, GoogLeNet, …**



Convolution layers ➡ Classification layers and output

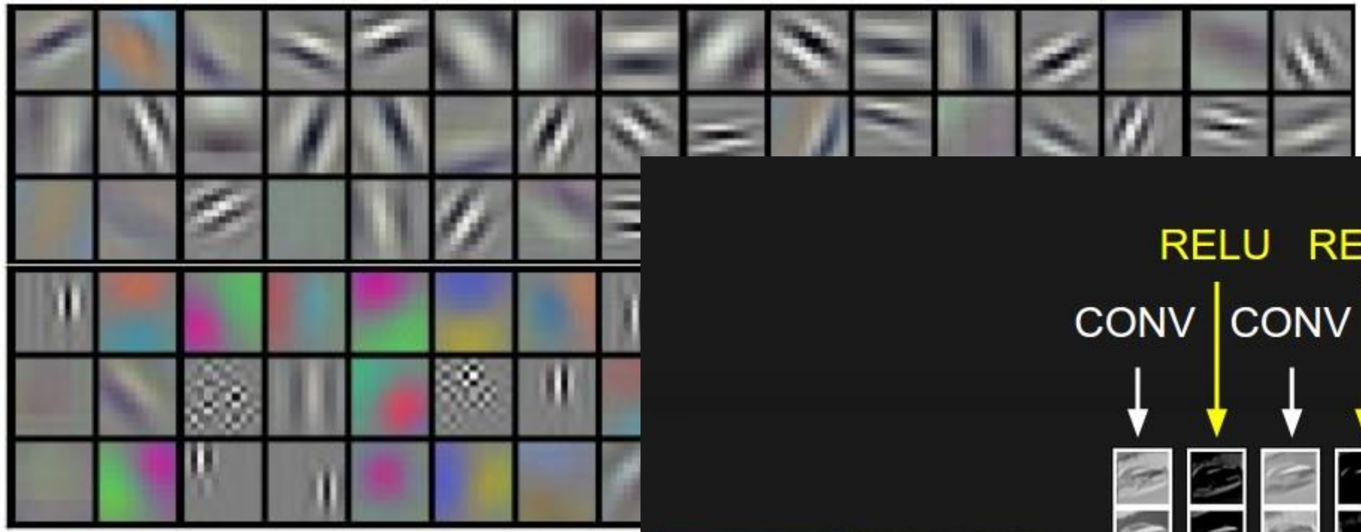
What Learned Convolutions Look Like

First convolution layer filters are simple features

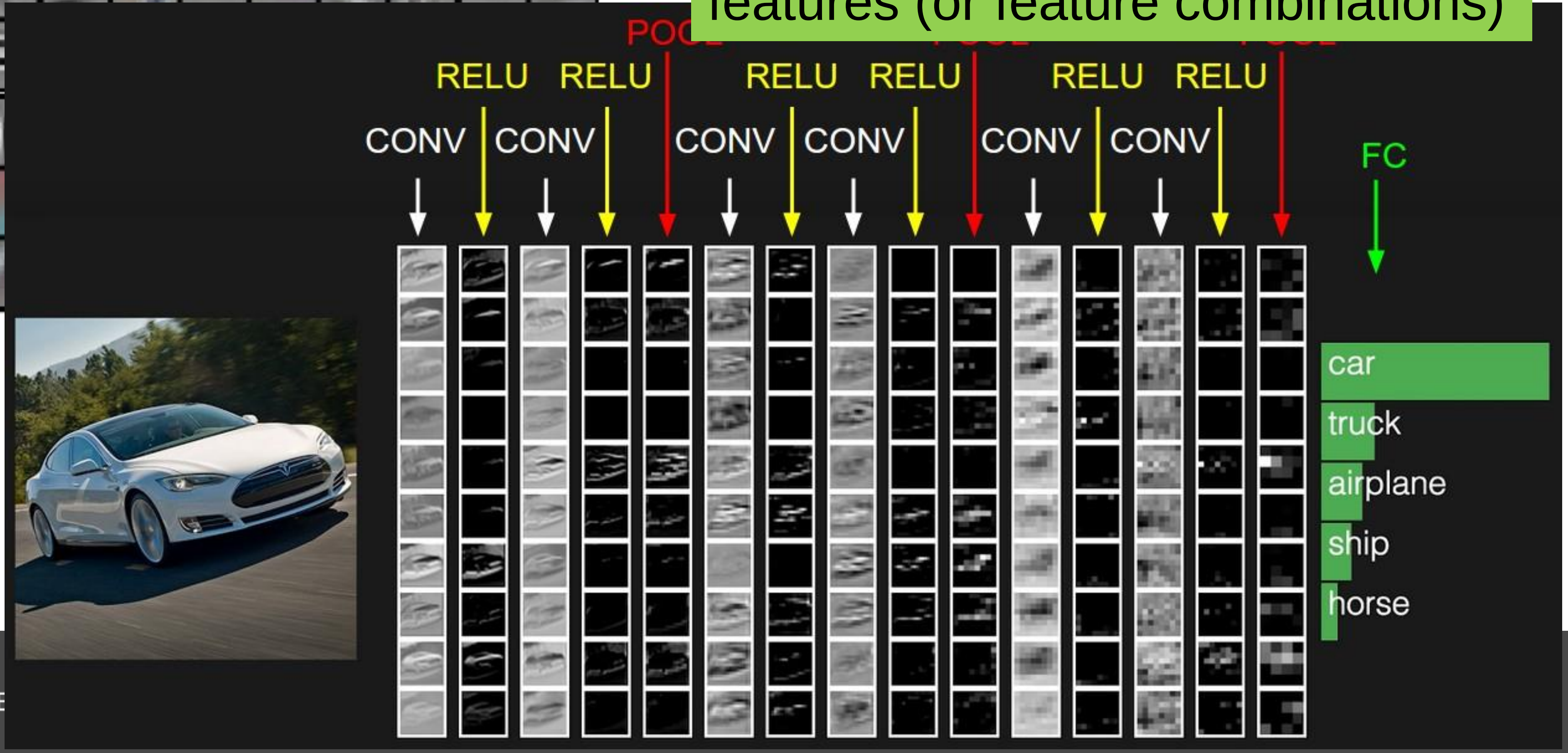


What Learned Convolutions

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

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 Keras/Tensorflow

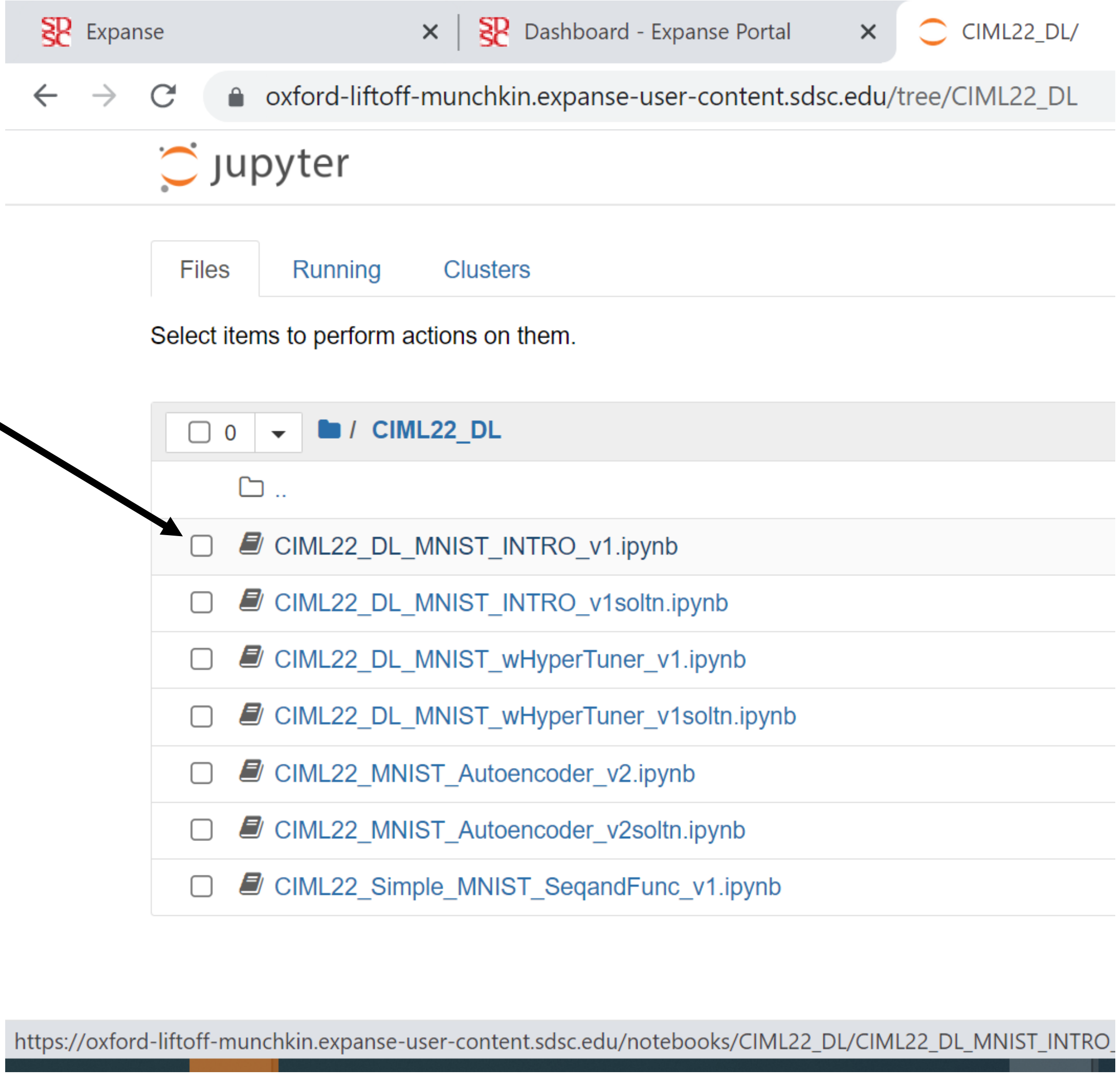
- Goal: Get familiar with Keras and CNN layers coding, and CNN solutions

- We will login and start a notebook

(see next pages for quick overview)

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 figures out the shapes
- Not every layer in Keras has trainable parameters – like which ones above?

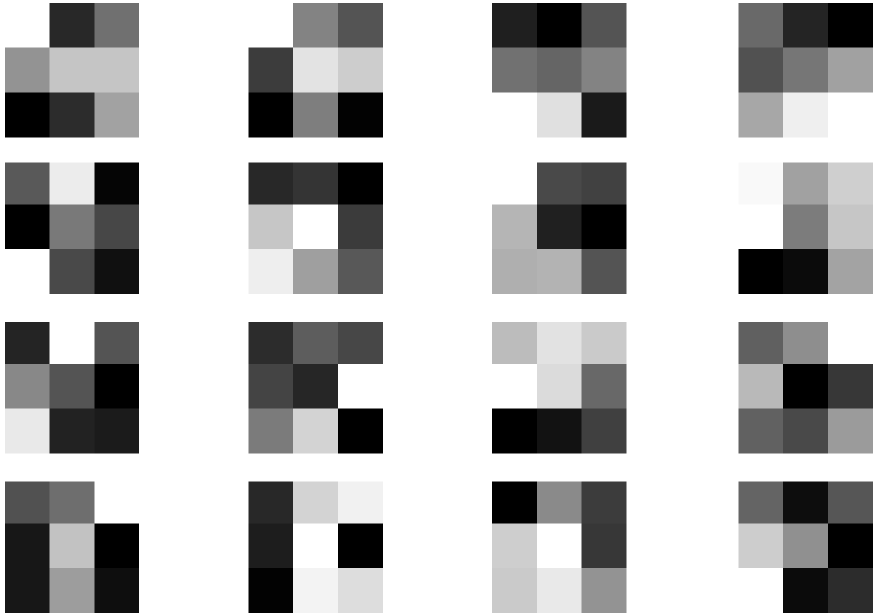
Zooming in on keras convolution layers statements

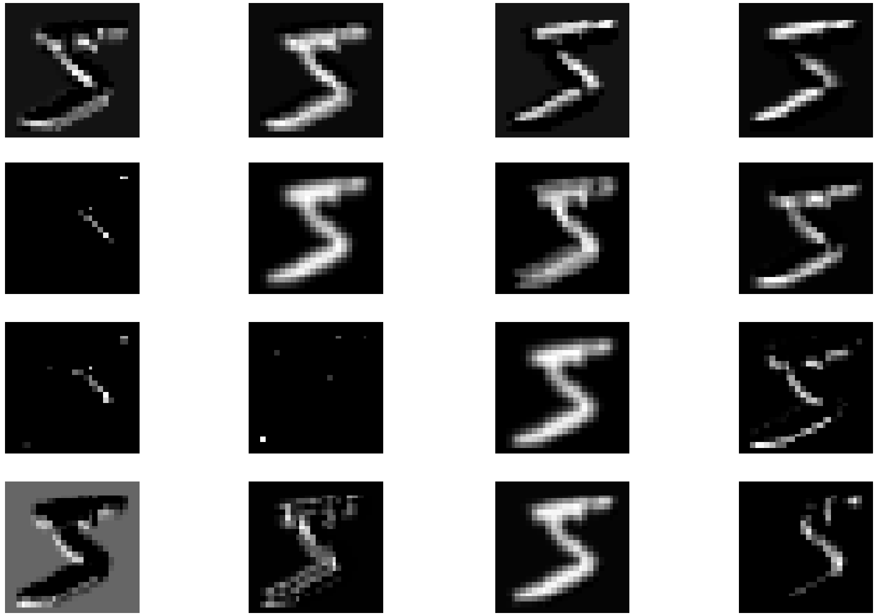
Use 16 filters, each of size 3x3

```
#  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)))
```

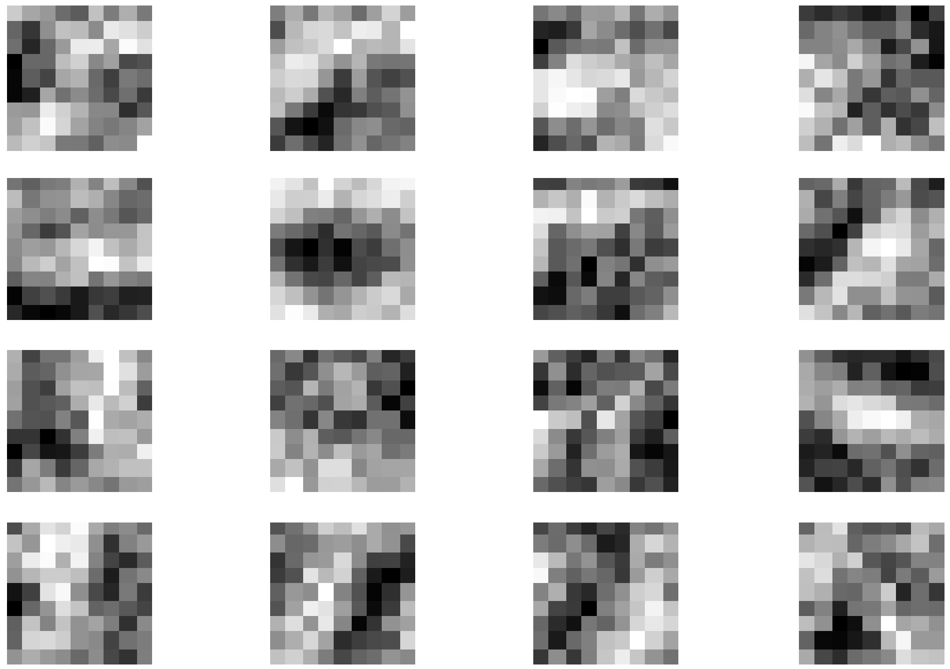
Input shape does not include number of images

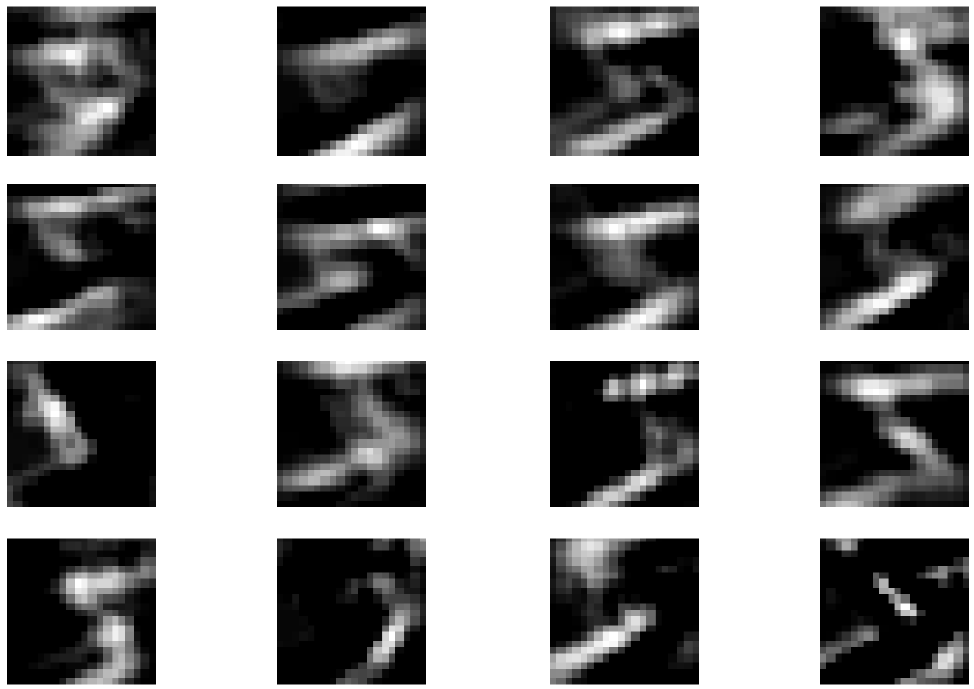
Exercise notes: 3x3 first convolution layer filter and activation





9x9 first convolution layer filter and activation





- **Some guidelines**

Things to think about for running a project

- Choosing Hyperparameters – a bit of exploration and exploitation
- Need to figure out efficient Job workflow
- On HPC, CPU work fine for many cases, you will want to use GPUs for ‘large’ models and/or large datasets.
- Model saves and/or checkpoints are available in tensorflow; tensorboard available but needs to be secure (ask for details)

Choosing Hyperparameters

- Generally:
  - architecture (layers, units, activation, filters, …)
  - algorithm (learning rate, optimizer, epochs, …)
  - efficient learning (batch size, normalization, initialization, …)

- Some options are determined by task:
  - loss function, CNN vs MLP, …

- Use what works, from related work or the latest recommendations,

Hyperparameters Search

- Can take a long time, hard to find global optimal
- Start with small data, short runs to get sense of range of good parameter values
- Easy but possibly time-consuming method:

  grid search over uniformly spaced values
- Do “exploration” then “exploitation”, ie search wide then search deep

  Keras Tuner functions can help with the wide search

Keras Hyperparameter Search Tool

- Keras Hypertuner class implements several search strategies:

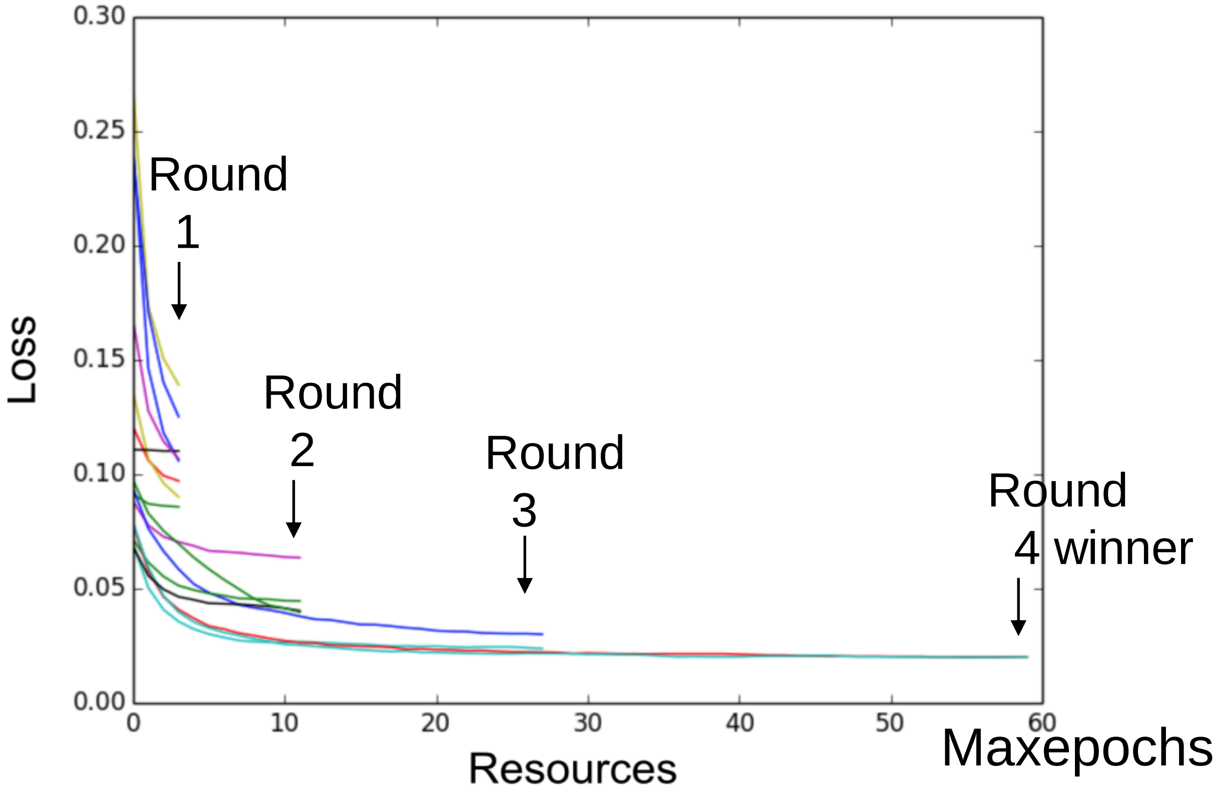
Hyperband is like a tournament competition of hyperparameter configurations, with incremental training, to weed out worse ones

RandomSearch will search randomly through the space of configurations and try to find better regions

Bayesian optimization is like function approximation to pick out next configuration

Hyperband Bracket

Each round runs several network configurations for small number of epochs
Several rounds with increasing epochs make up a bracket
Several brackets are run to end up with several possible overall winners.



Note, you could run a small grid search around hyperband winners to confirm performance

Keras Tuner code snippet

Set up function to make the model

Set up hyperparameter choices

```
#             Set up model
def build_model(numfilters,activation_choice):   #<<------add code: if yo
                                        #          list and change code t
    mymodel = keras.models.Sequential()
    mymodel.add(keras.layers.Convolution2D(numfilters,
                                        (3, 3),
                                        strides=1
```

Keras Tuner code snippet – wrap build-model with search info and tuner

Set up function to make the model

```
def build_model_hp(hp):
  hp_numfilters     = hp.Int('hpnumfilters',min_value=8,max_value=32,step=4)
  #your variable name         ^^^ the parameter name in the hp object
```

Set up hyperparameter choices

```
def build_model(numfilters,activation_choice):   #<<------add code: if yo
                                                 #          list and change code t

    mymodel = keras.models.Sequential()
    mymodel.add(keras.layers.Convolution2D(numfilters,
                                           (3, 3),
                                           strides=1
```

Define 'tuner' object; tuner search uses the model fit function

```
tuner = kt.Hyperband(build_model_hp,
                     objective  = 'val_accuracy',
                     max_epochs = num_max_epochs,
                     factor     = 3,
                     hyperband_iterations=10,
                     directory   =dirname,
                     overwrite   =True,    #overwrite directo
                     project_name='hyperbandtest',
                     executions_per_trial=5,  #to try sever
                     seed        =777)
```

Workflow and Organizing Jobs

Job Level: What makes sense to include in each job?

Model Level: run & test model for each parameter configuration

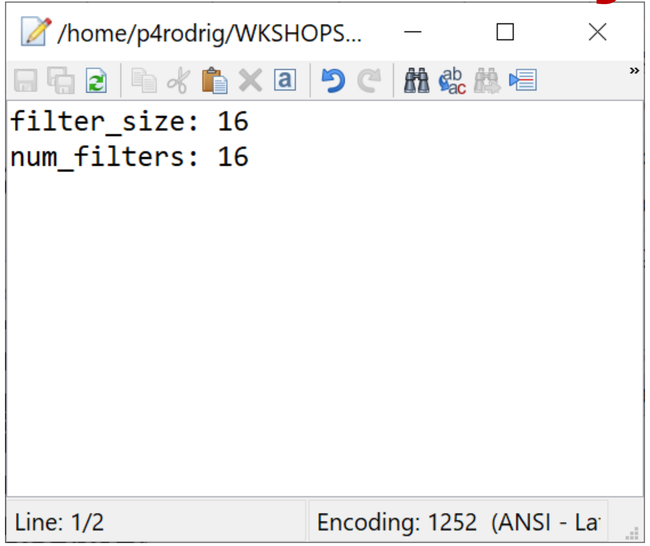
Data Level: loop through cross validation datasets (if applicable)

- **Consider how long each a model runs for 1 configuration of hyperparameters for 1 dataset**
- **Organize jobs into reasonable chunks of work**
- **For large models consider model-checkpoints**

Organizing Configurations – one way

Code snippet: using 'YAML' file to set up hyperparameter configuration

Create text file with "Parameter: Value" pairs



Read file as python dictionary

```python
import yaml

with open("./modelrun_args.yaml", "r") as f:
    my_yaml=yaml.safe_load(f)  #this returns a python dictionary

filter_size=my_yaml.get("filter_size")
num_filters=my_yaml.get("num_filters")
print('arguments, filter size:',filter_size,' num filters',num_filters)
```

Example slurm job script and execution for Expanse

```
#!/usr/bin/env bash
#SBATCH --job-name =mnist0522
#SBATCH --account=sds164
#SBATCH --partition=compute
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=128
#SBATCH --time=00:10:00
#SBATCH --output=myjoboutput.o%j.%N.out

module purge
module load singularitypro
module list

echo "filter_size: 3 " >  modelrun_args.yaml
echo "num_filters: 16 " >> modelrun_args.yaml

singularity exec --bind /expanse,/scratch  --nv \
  /cm/shared/apps/containers/singularity/tensorflow/tensorflow-latest.s
  python3 Intro_mnist_cnn2_forbatch.py > mymnist_stdoutput.txt
```

You could also modify or set up parameters; and save yaml files for each run

```
02 EXP-05192022]$
02 EXP-05192022]$ sbatch run-job-tf-compute.sbatch
 job 12502781
02 EXP-05192022]$ squeue --me
BID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
781   compute =mnist05 p4rodrig PD       0:00      1 (Priority)
813   compute galyleo- p4rodrig  R      26:03      1 exp-13-06
02 EXP-05192022]$
```

Python Notebook vs Scripts

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

1 Papermill is one tool

2 Or, you can use "*jupyter nbconvert --to script your-python.ipynb***" in the batch job.**

Also, turnoff plot display, save plots in files, and use a configuration file to pass in parameters

note on using GPU

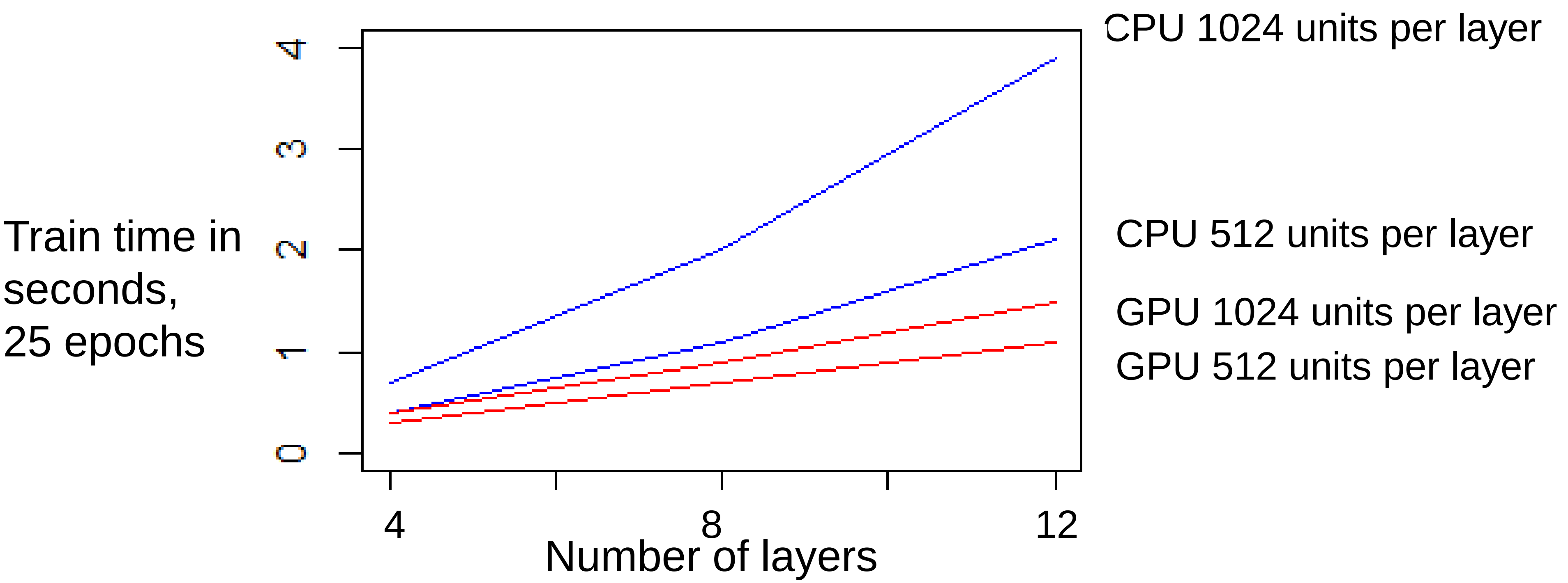
- **GPU node has multiple GPU devices**
- **By default tensforflow will run on 0th gpu device if GPU is available, otherwise it will use all CPU cores**

Code snippet to check for GPU devices

/home/p4rodrig/WKSHOPS/EXP-05192022/Intro_mnist_cnn2_forbatch.py - p4rodrig@login.expanse.sdsc.edu - Editor - W

```
physical_devices = tf.config.list_physical_devices('GPU')
n_gpus           = len(physical_devices)
print("Info,config,Num GPUs:", n_gpus)
if tf.test.gpu_device_name():
  print('Info, test,Default GPU Device:{}'.format(tf.test.gpu_device_name()))
```

GPU shared (V100) vs CPU (128 cores)
For MLP with Dense Layers, 80000x200 data matrix



GPUs faster, but you might have to wait more in job queue; also some memory limits compared to CPU, may need to use smaller batch size

Parallel DL models with multiple nodes/devices

- **The main approach to parallelize training: Data Parallel:**
  1. **Split up data (in Keras see ‘tf.data.Datasets’ API or use numpy)**
  2. **Launch script on each device**
  3. **Each device trains a copy of the model with a part of the data**
  4. **Aggregate parameter updates across model instances**

Parallel DL models with multiple nodes/devices

- **The main approach to parallelize training: Data Parallel:**
  1. **Split up data (in Keras see ‘tf.data.Dataset’ API, or use numpy)**
  2. **Launch script on each device**
  3. **Each device trains a copy of the model with a part of the data**
  4. **Aggregate parameter updates across model instances**

- **Main tools: Keras/Tensorflow ‘strategy’ or use Horovod MPI wrappers**

- **Other approaches include Model Parallel (e.g. few layers per device), or Model and Data parallel**

- **Also, using mixed precision can reduce memory footprint**

Keras/Tensorflow strategy single GPU node

Set up a 'mirror' strategy

```
mirrored_strategy = tf.distribute.MirroredStrategy(["GPU:0", "GPU:1",  "GPU:2",  "GPU:3"])
```

You also need the strategy scope around the model definition so that it can make copies

```
if (n_gpus>0):
    with mirrored_strategy.scope():
      multi_dev_model=build_model()
```

Then train as normal (use batch size multiple of 32)

Keras/Tensorflow strategy multiple GPU node

Keras also has a ‘multiworker’ strategy but it requires setting up config files with IP addresses

On HPC systems resources are shared so IP addresses are dynamic

Better to use Horovod with MPI and slurm batch job

On Expanse, for example, single node, single device execution

In slurm batch script:

singularity → python

Your python script

- Load data
- Build model
- Train

Multinode, MPI launches instances

In slurm batch script:

mpirun –n **number of tasks** singularity → python

Your python script

Load data

Build model

Train

This gets distributed across nodes and cores by slurm & mpi parameters

Multinode, mpi launches instances

In slurm batch script:

mpirun –n **number of tasks** singularity → python

Initialize back end communication, get mpi rank

Your python script

mpi rank

Load data

Build model

Train

This gets distributed across nodes and cores by slurm & mpi parameters

Multinode, mpi launches instances

In slurm batch script:

mpirun –n **number of tasks** singularity → python

Initialize back end communication, get mpi rank

'shard' or split data

Your python script

mpi rank

Load data

Local dataset

Build model

Train

This gets distributed across nodes and cores by slurm & mpi parameters

Multinode, mpi launches instances

In slurm batch script:

mpirun –n **number of tasks** singularity → python

Initialize back end communication, get mpi rank

'shard' or split data

Set up a distributed optimizer

After each batch aggregate weight updates and broadcast new values

Your python script

mpi rank

Load data

Local dataset

Build model

Dist. optmzer

Train

Rank 0 handles weight updates, checkoints

This gets distributed across nodes and cores by slurm & mpi parameters

MPI launches one instance per processor

In slurm batch script:

mpirun –n **number of tasks** singularity → python

device =GPU:0

Your python script

mpi rank

Load data

Local dataset

Build model

Dist. optmzer

Train

Rank 0 handles updates

device =GPU:0

Your python script

mpi rank

Load data

Local dataset

Build model

Dist. optmzer

Train

……

……

device =GPU:0

Your python script

mpi rank

Load data

Local dataset

Build model

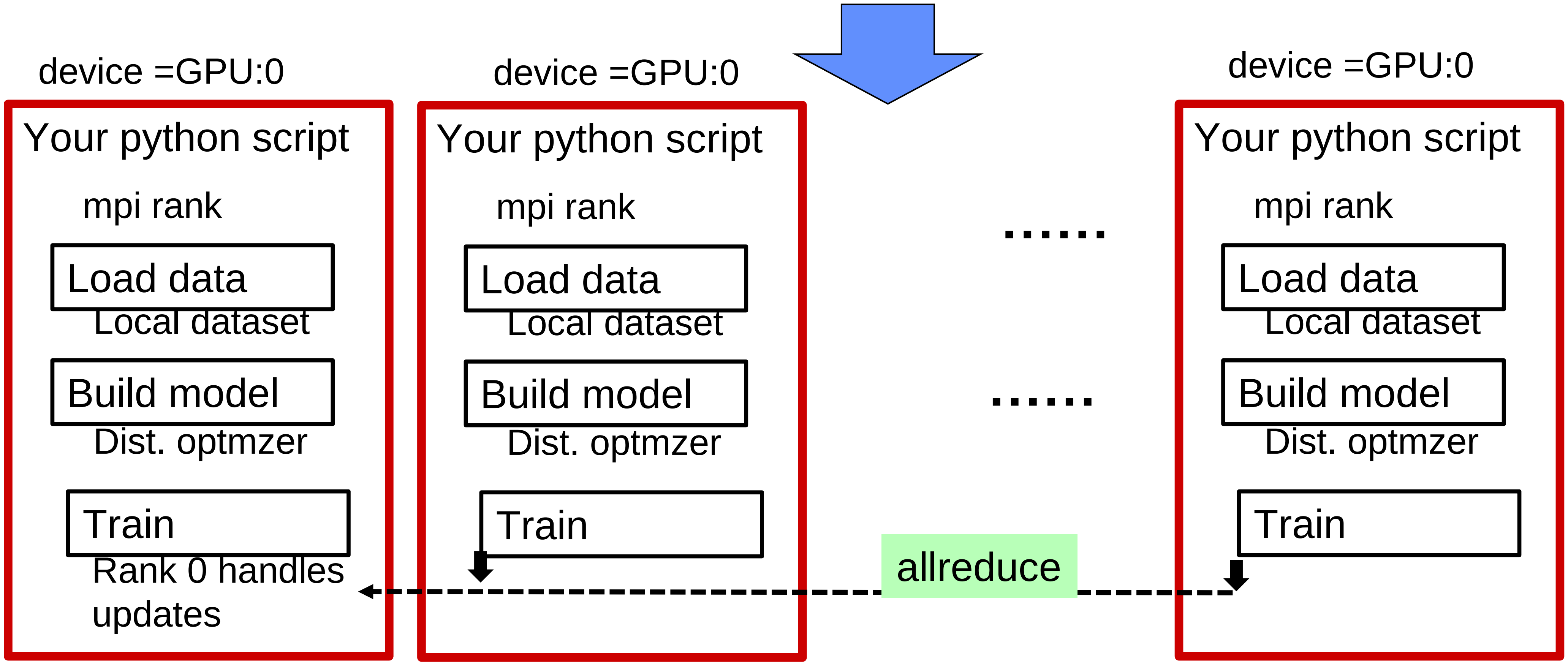
Dist. optmzer

Train

For each batch: Horovod will aggregate & share weights updates

In slurm batch script:

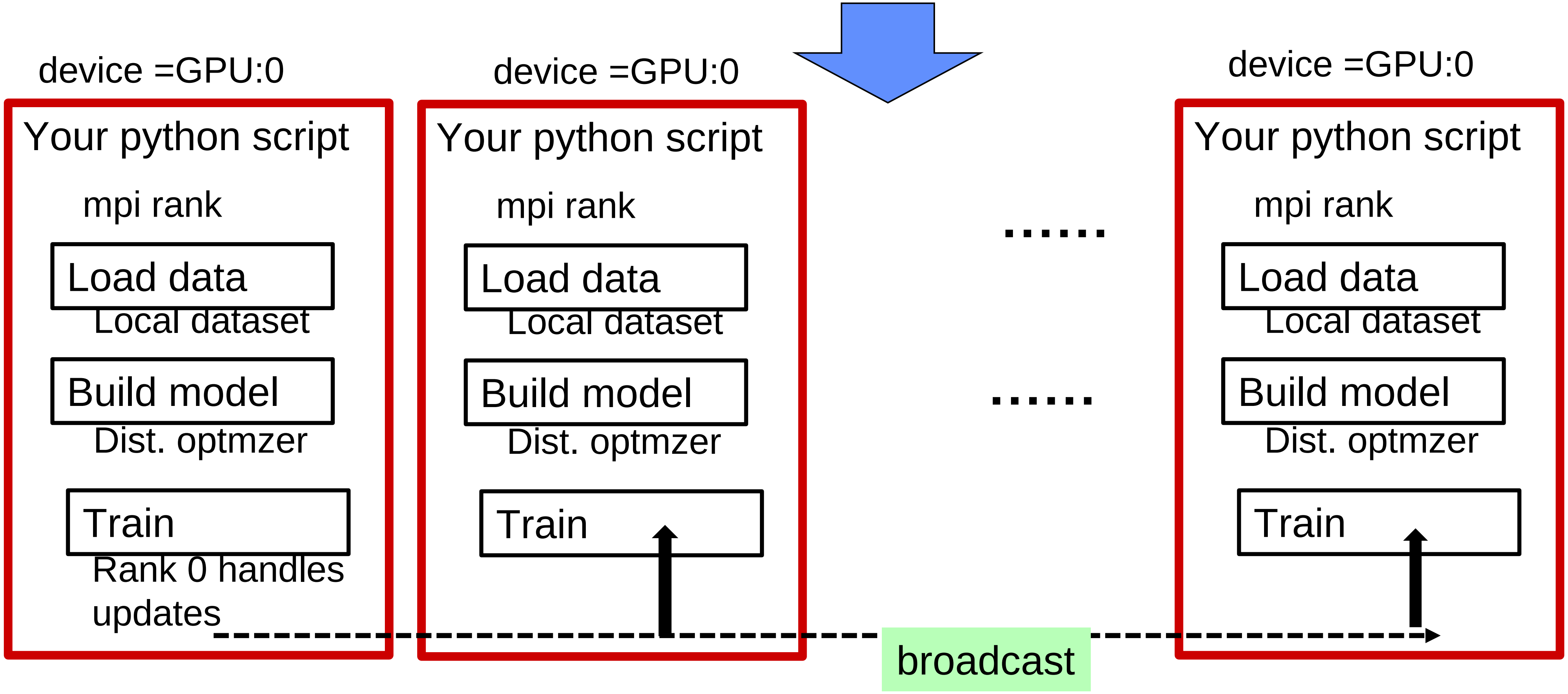
mpirun –n **number of tasks** singularity → python



For each batch: Horovod will aggregate & share weights updates

In slurm batch script:

mpirun –n **number of tasks** singularity → python



Bigger batch size better but more memory

Code snippets – Horovod functions

Not many lines of code, but becareful with sharding, batch size,
See https://horovod.readthedocs.io/en/latest/keras.html

Initialize back end communication, get mpi rank

```
import horovod.tensorflow.keras as hvd
hvd.init()
print('INFO, global rank:',hvd.rank(), ' locrank ',hvd.local_rank())
```

'shard' or split data

```
Note: be careful to get batch size in sharding and training
aligned
```

Set up a distributed optimizer

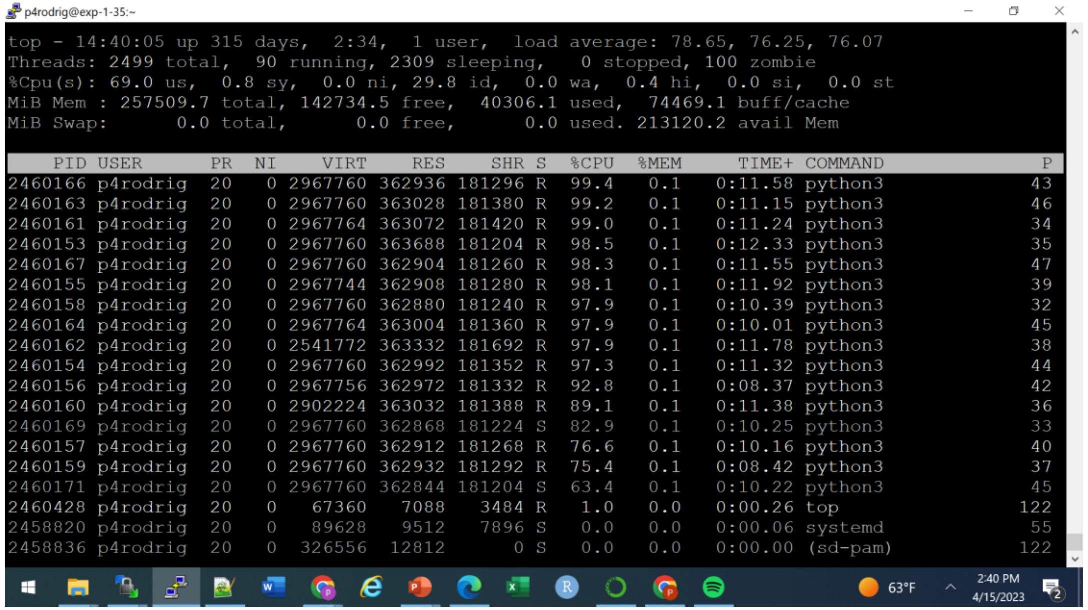
```
 optimizer2use  =
hvd.DistributedOptimizer(Adam(learning_rate=0.001*hvd.size())
```

After each batch aggregate weight updates and broadcast new values

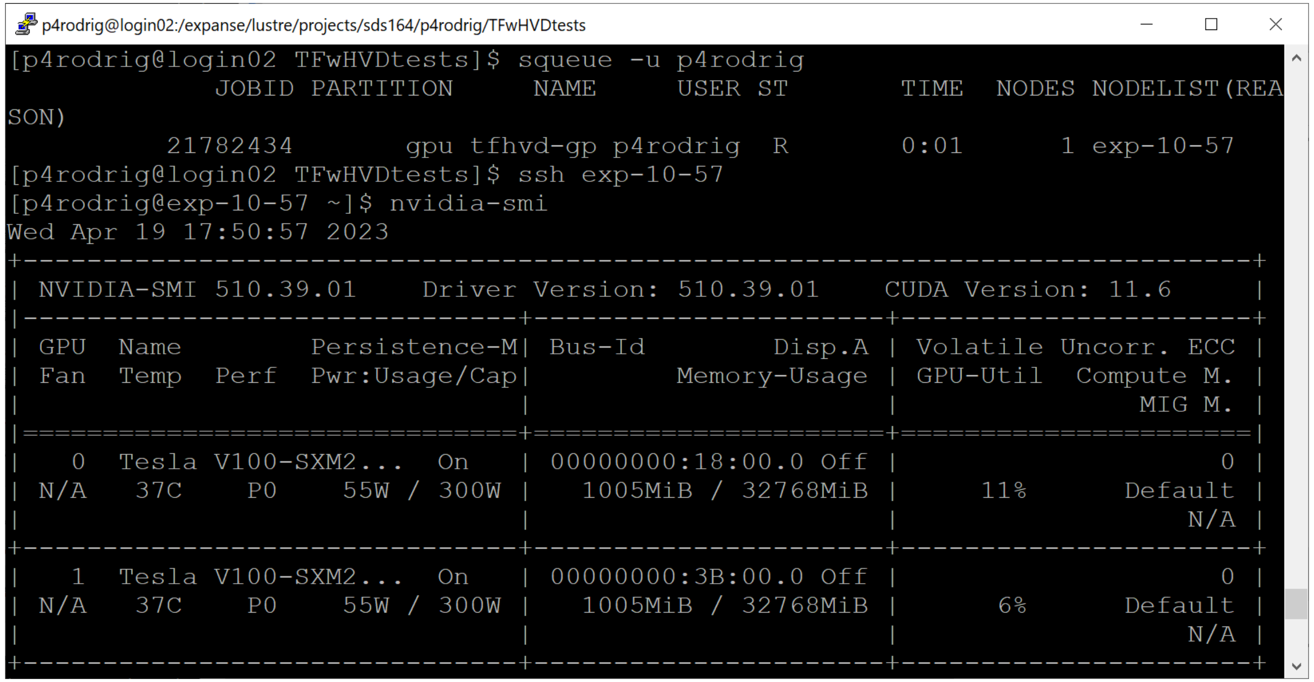
```
model.compile(optimizer    =optimizer2use,
 …
experimental_run_tf_function=False)
```

Note: on Expanse, you can run: $ squeue –u to get nodes running your job
Then you can ssh to node, run:

$ top –u userid to see processing on a CPU job



Or run: $ nvidia-smi
to see usage on GPU devices



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**