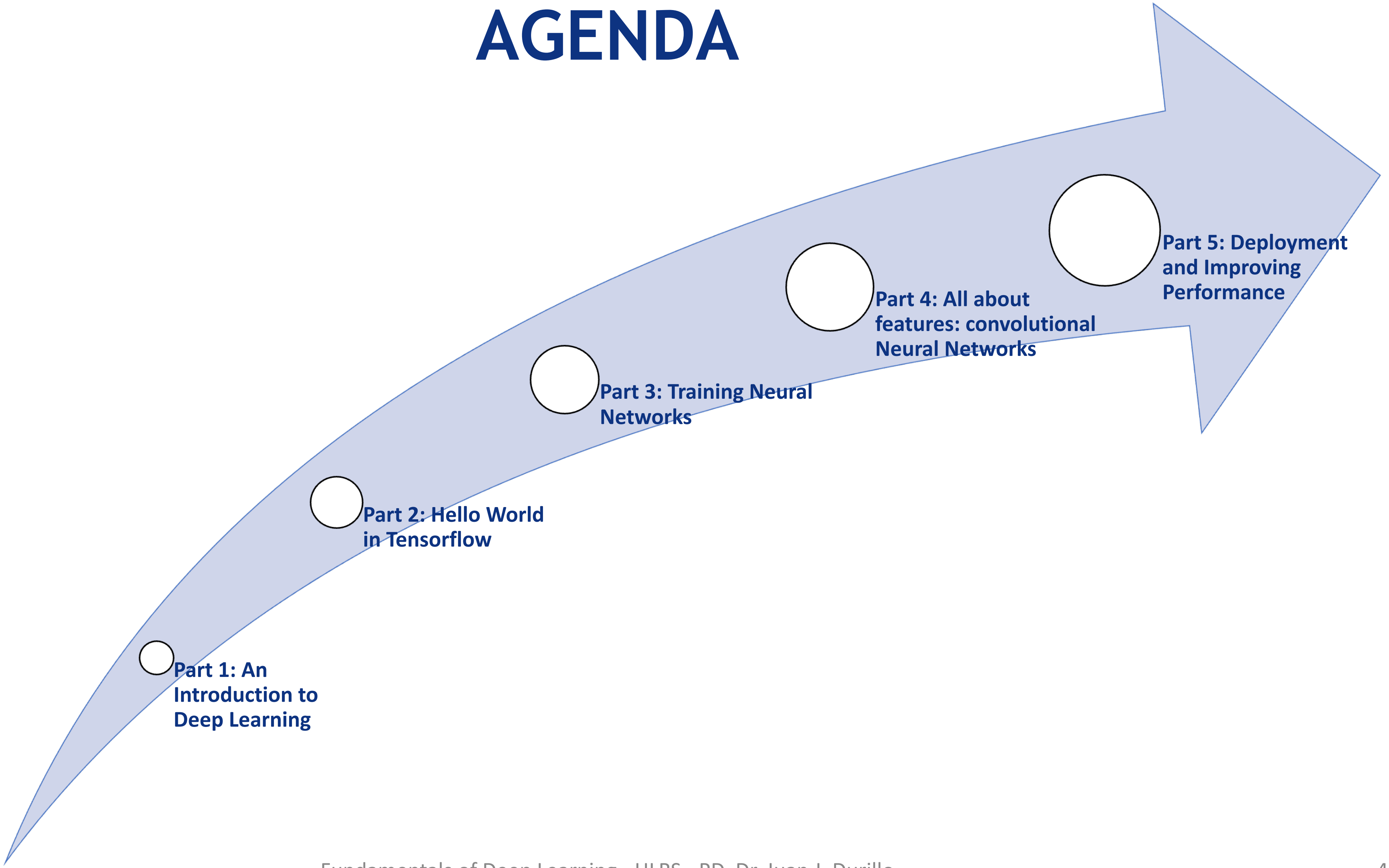
FUNDAMENTALS OF DEEP LEARNING

A BIT ABOUT ME

- PD Dr. **Juan Jo**sé Durillo Barrionuevo
  - Since 2018 I am full time researcher at Leibniz Supercomputing Centre
  - Since 2019 Nvidia University Ambassador for
    - Fundamentals of Deep Learning
    - Data parallelism, how to train in multiple GPUs
    - Transformer based applications of NLP
  - Since 2022 I am visiting lecturer at Technical University Muenchen
    - Next Generation AI Hardware
  - Email: durillo@lrz.de
- My path to here:
  - 2011 – 2017 Assistant Professor University of Innsbruck
    - Artificial Intelligence for compiler and software orchestrator
  - 2011 PhD at the University of Málaga
    - Artificial Intelligence for multi-objective optimization problems
      - focus on Nature Inspired computing

- Get you up and on your feet quickly
- Build a foundation to tackle a deep learning project right away
- We won’t cover the whole field, but we’ll get a great head start
- Foundation from which to read articles, follow tutorials, take further classes

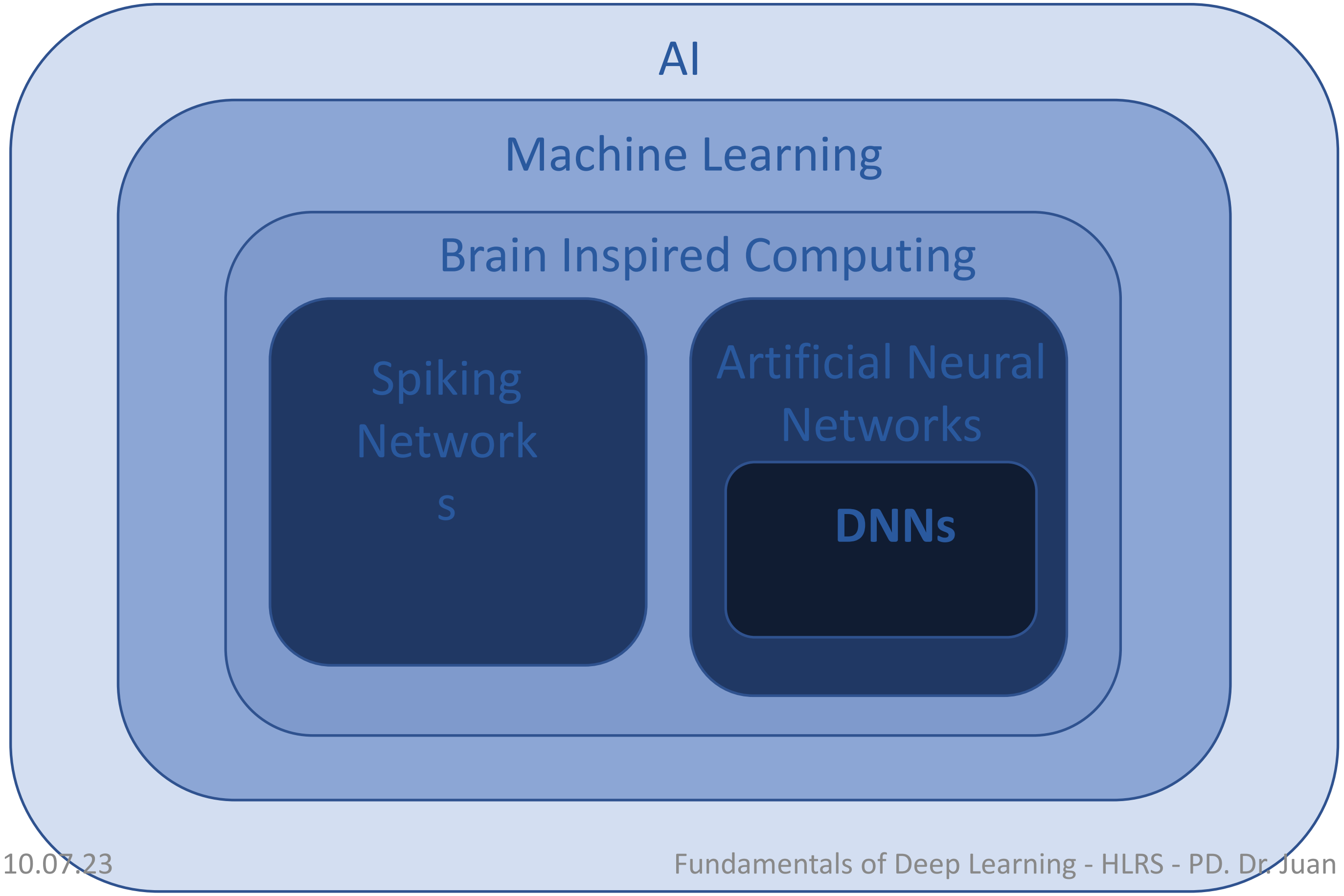
AGENDA


Part 1: An Introduction to Deep Learning
Part 2: Hello World in Tensorflow
Part 3: Training Neural Networks
Part 4: All about features: convolutional Neural Networks
Part 5: Deployment and Improving Performance

MOTIVATION

- DNN (Deep Neural Networks) are the foundation of many AI applications
  - Speech Recognition, Text Generation, Image Processing, Autonomous-driving cars, Cancer Detection, Playing complex games, Generation of software based on textual descriptions
- Even exceeded human-level accuracy in some domains
- DNNs are not for free and they have associated a high computational complexity in both:
  - Inference and Training

DNNS IN THE AI CONTEXT



AI: Science and Engineering field related to the creation of machine that have the ability of achieving goals like humans do

ML: […] without having these goals explicitly programmed

BIC: […] having the model of how the brain works as inspiration

Spiking: […] network of neuros that fires up with input and time dependences

ANN: […] network of neuros that fires up depending on input

DNN: […] network of neuros that fires up depending on input, with a certain deepness of layers

A LONG WAY UNTIL TODAY

EARLY ON, GENERALIZED INTELLIGENCE LOOKED POSSIBLE

TURNED OUT TO BE HARDER THAN EXPECTED

Early Neural Networks

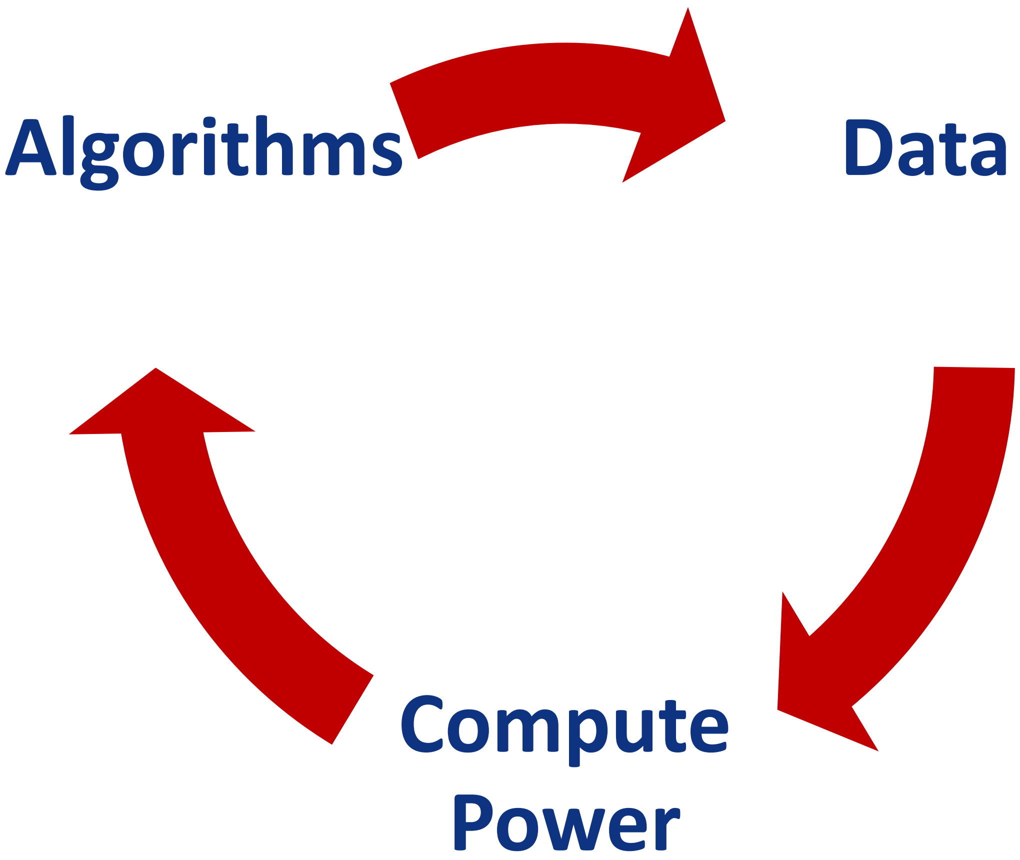
- Inspired by biology
- Created in the 1950’s
- Outclassed by Von Neumann Architecture

Expert Systems

- Rule based code
- Complex
- Require expertise on the problem
- Large set of fixed, hard to understand by all, set of rules

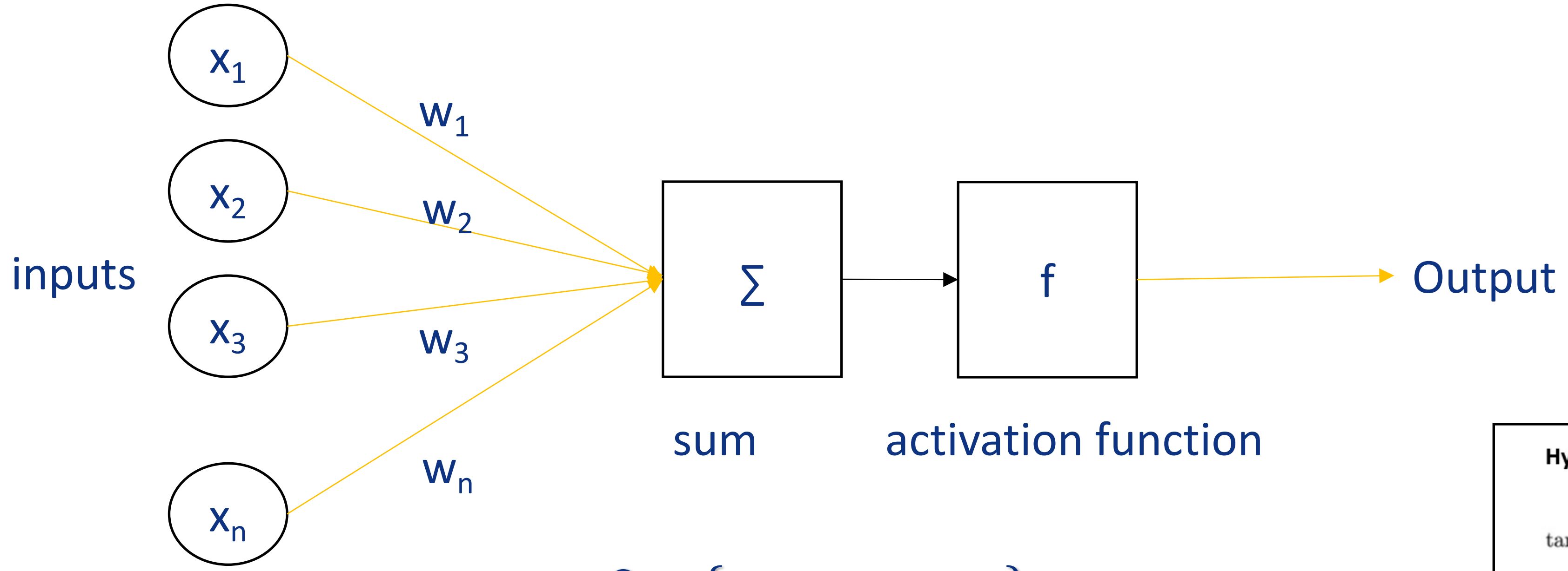
EXPERT SYSTEMS

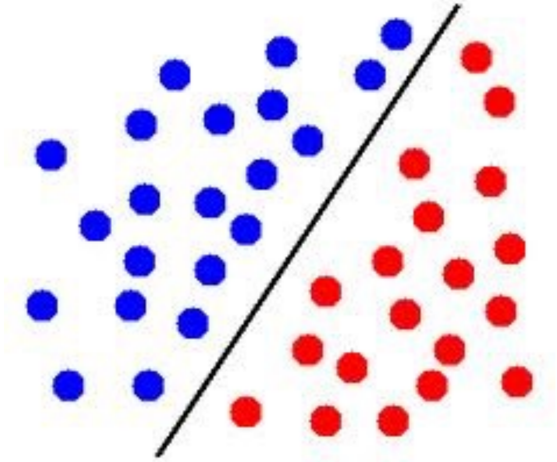
DEEP LEARNING REVOLUTION



WHAT IS DEEP LEARNING?

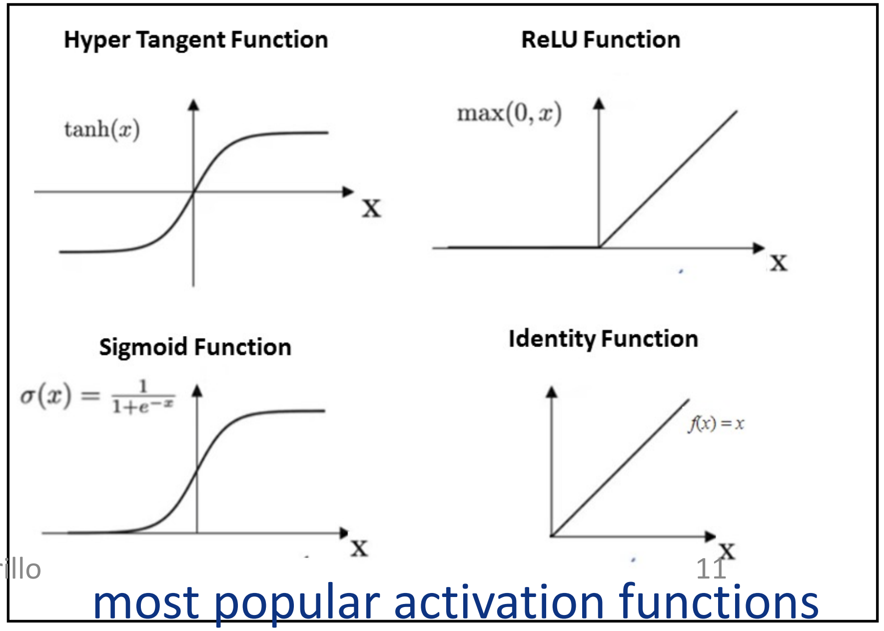
SINGLE NEURON



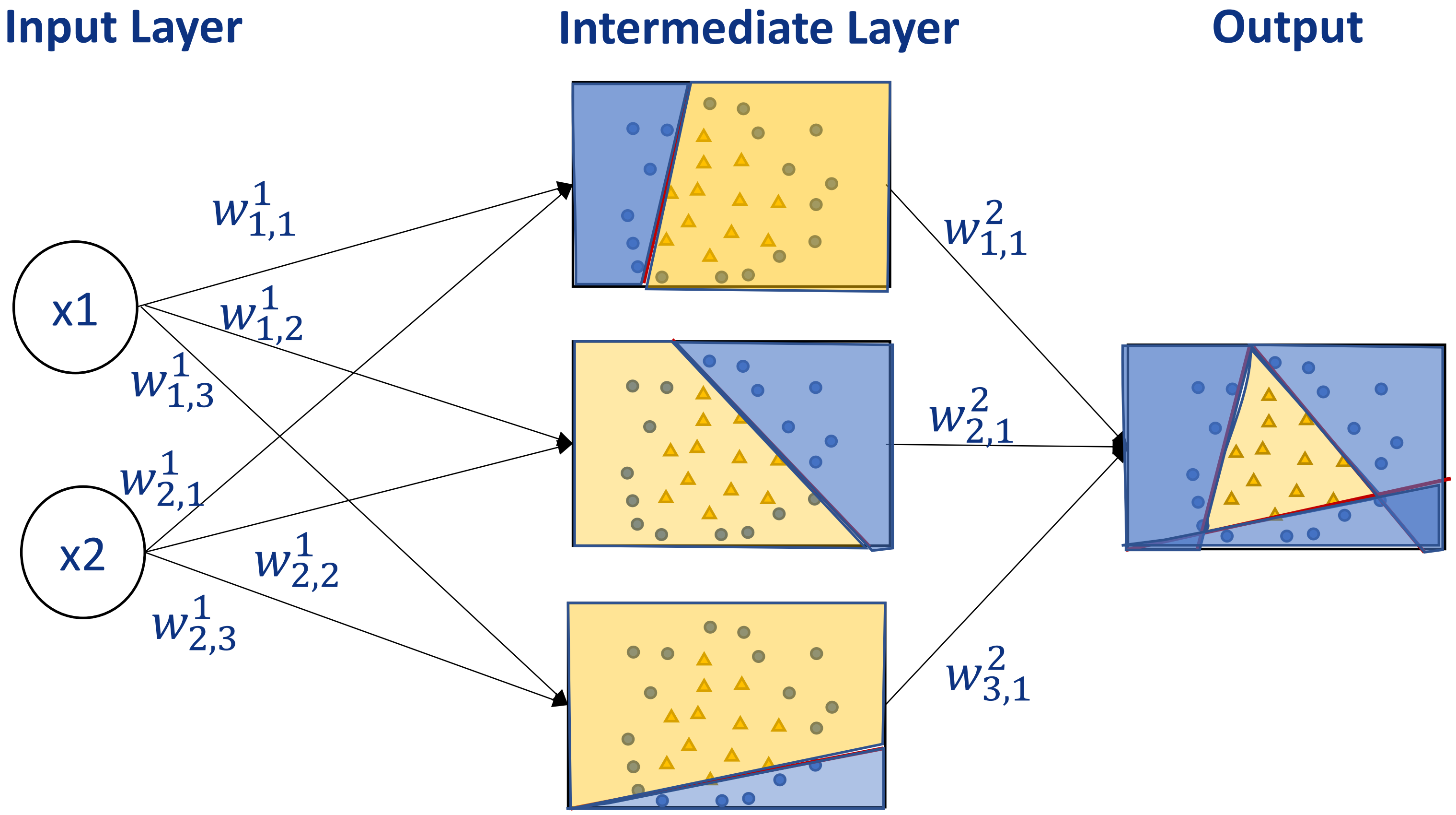


$$\Theta = \{w_1, w_2 \dots, w_n\}$$

Single artificial neurons work well for linearly separable datasets (indeed output is the activation effect on a linear combination of the input)



NEURAL NETWORK

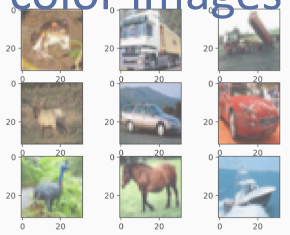


$$\Theta = \{w_{1,1}^{1}, w_{1,2}^{1}, w_{1,3}^{1}, w_{2,1}^{1}, w_{2,2}^{1}, w_{2,3}^{1}, w_{1,1}^{2}, w_{2,1}^{2}, w_{2,3}^{2}\}$$

- Works well even when the data is not linearly separable

SUPERVISED LEARNING

X: 32 x 32 color images



Υ : labels

{truck, car, horse, bird, boat}

Example (CIFAR10 dataset)

- Data domain Z: $X \times \Upsilon$

  X → domain of the input data

  Υ → set of labels (knowledge)
- Data Distribution is a probability distribution over a data domain
- Training set $z_1, \ldots, z_n$ from Z assumed to be drawn from the Data Distribution D
- Validation set $v_1, \ldots, v_m$ from Z also assumed to be drawn from D
- A machine learning model is a function that given a set of parameters Θ and z from Z produces a prediction
- The prediction quality is measured by a differentiable non-negative scalar-valued loss function, that we denote $\ell(\Theta; z)$

(SUPERVISED) LEARNING

- Given Θ we can define the expected loss as: $L(\Theta) = \mathbb{E}_{z\sim D}[\ell(\Theta; z)]$
- Given D, ℓ, and a model with parameter set Θ, we can define learning as:

  "The task of finding parameters Θ that achieve low values of the expected loss, while we are given access to only n training examples"
- The mentioned task before is commonly referred to as *training*
- Empirical average loss given a subset of the training data set $S(z_1, \ldots, z_n)$ as:

$$\hat{L}(\Theta) = \frac{1}{n}\sum_{t=1}^{n}[\ell(\Theta; z_t)]$$

- Usually a proxy function, easier to understand by humans, is used for describing how well the training is performed (e.g., accuracy)

(SUPERVISED) LEARNING

- The dominant algorithms for training neural networks are based on mini-batch stochastic gradient descent (SGD)

- Given an initial point Θ_0 SGD attempt to decrease $\hat{L}$ via the sequence of iterates

$$\Theta_t \longleftarrow \Theta_{t-1} - n_t g(\Theta_{t-1}; B_t)$$

$$g(\Theta; B) = \frac{1}{|B|} \sum_{z \epsilon B} \nabla \ell(\Theta; z)$$

B_t: random subset of training examples

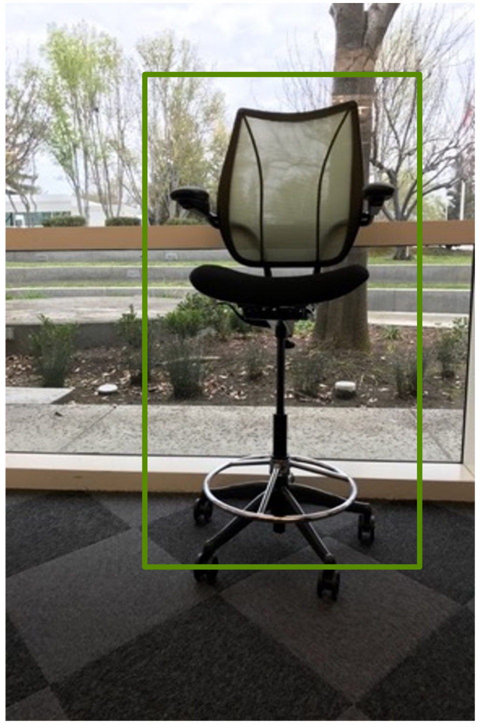
n_t: positive scalar (learning rate)

epoch: update the weights after going over all training set

COMPUTER VISION EXAMPLES

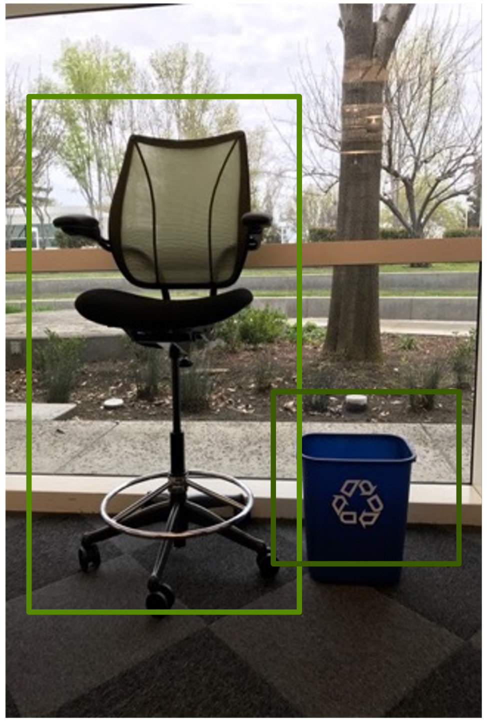
predicting the type or class of an object in an image

Image Classification



predicting the type or class on an object in an image and draw a bounding box around it

Image Classification + Localization



predicting the location of objects in an image via bounding boxes and the classes of the located objects

Object Detection

predicting the class to which each pixel in the image belongs to

Image Segmentation

INPUT REPRESENTATION

28 x 28
= 784 pixels

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	55	97	157	156	187	255	81	0	0	0	0	0
0	0	0	0	0	0	0	0	5	20	58	20	96	140	215	215	251	254	254	254	254	254	163	0	0	0	0	0
0	0	0	0	0	0	0	0	157	254	254	254	254	254	254	254	254	254	254	254	254	241	100	0	0	0	0	0
0	0	0	0	0	0	0	0	155	117	184	214	214	155	117	49	19	19	141	254	241	158	23	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	173	254	246	70	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	90	247	254	219	68	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	8	43	203	254	250	121	15	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	43	146	254	254	229	111	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4	137	245	254	254	238	40	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	133	254	254	254	254	254	235	106	23	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	55	196	196	158	196	223	254	254	216	23	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	70	245	253	99	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	197	254	134	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	200	254	134	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	88	246	253	99	0	0	0	0	0	0	0	0	0
0	0	0	37	59	18	0	0	0	0	0	0	0	0	67	254	254	170	0	0	0	0	0	0	0	0	0	0
0	0	0	128	254	159	0	0	0	0	0	11	20	140	248	254	204	9	0	0	0	0	0	0	0	0	0	0
0	0	1	156	254	229	86	79	79	161	176	218	254	254	254	168	6	0	0	0	0	0	0	0	0	0	0	0
0	0	0	29	203	254	254	254	255	254	254	254	237	95	19	6	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	5	89	155	155	156	111	58	58	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

image

```
dict=['EOS','a','my','sleeps','on','dog','cat','the','bed','floor']
```

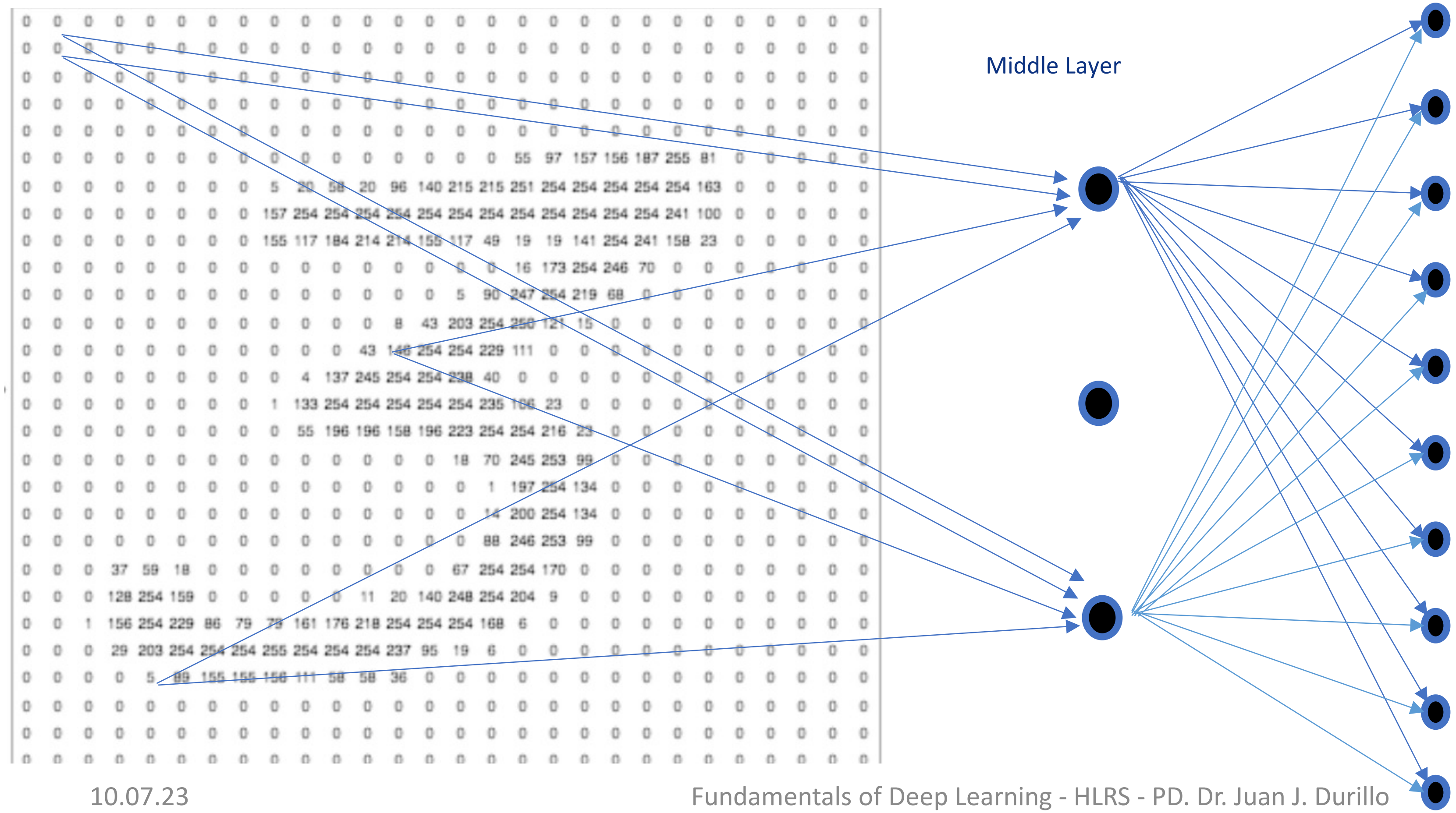
sentence = `['a', 'dog', 'sleeps', 'on', 'the', 'floor', 'EOS']`

```
[[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

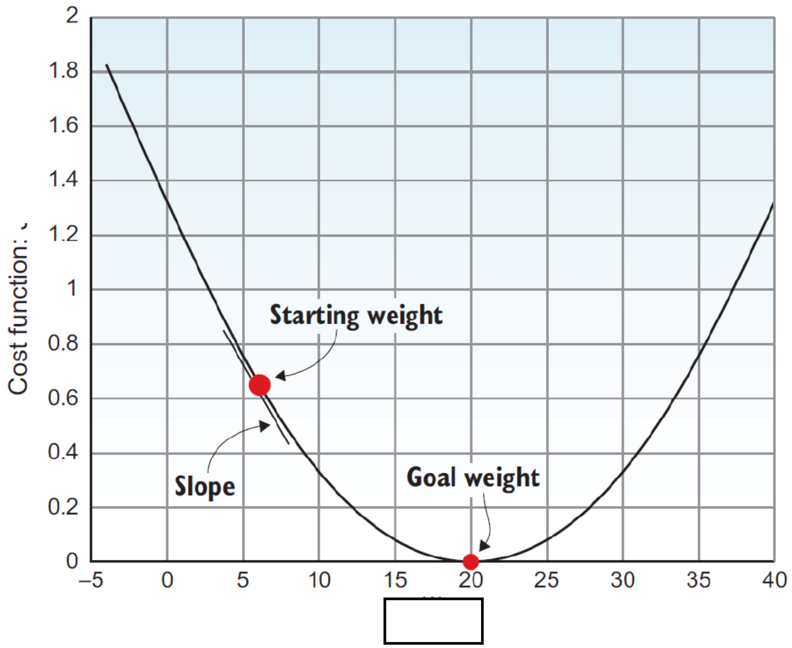
language

NEURAL NETWORKS FOR IMAGE CLASSIFICATION

Fully Connected Neural Network

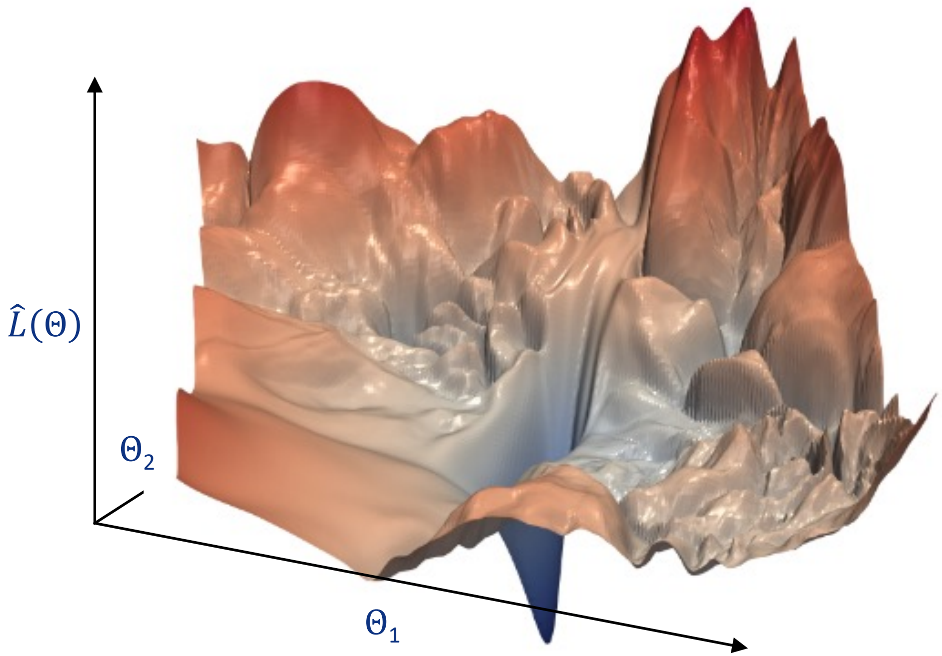


TRAINING NEURAL NETWORKS



Θ

main idea



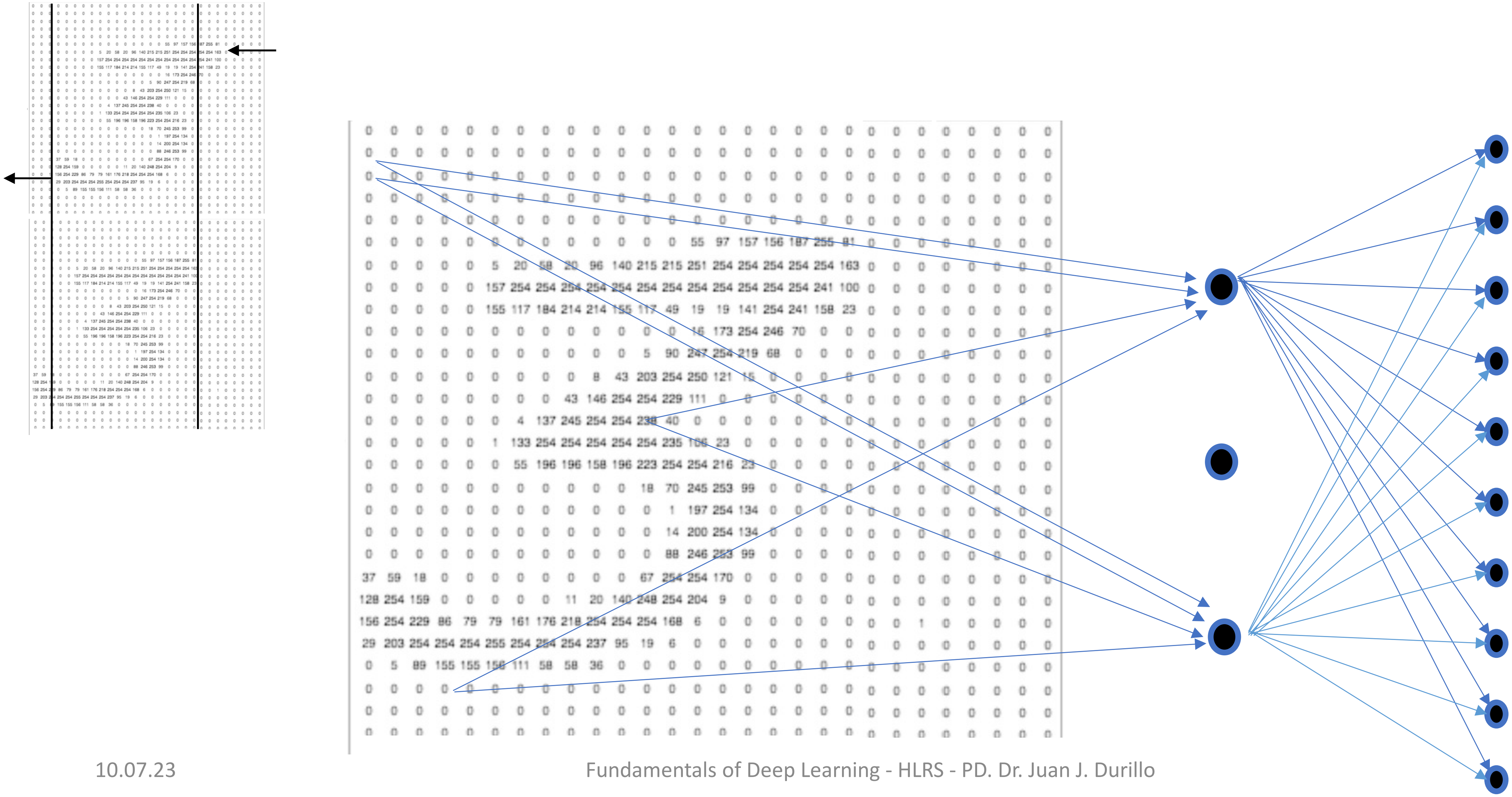
how the surface looks like in reality

Stochastic Gradient Descent

$$\Theta_t \leftarrow \Theta_{t-1} - \eta_t g(\Theta_{t-1}; B_t)$$

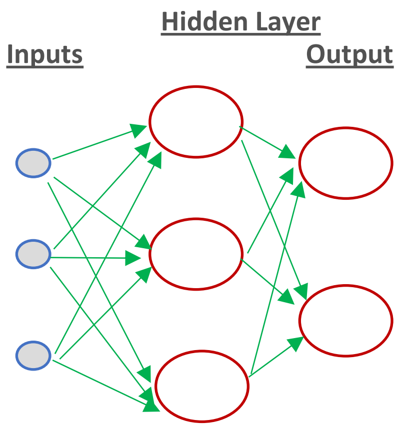
$$g(\Theta; B) = \frac{1}{|B|} \sum_{z \epsilon B} \nabla \ell(\Theta; z)$$

NEURAL NETWORKS FOR IMAGE CLASSIFICATION

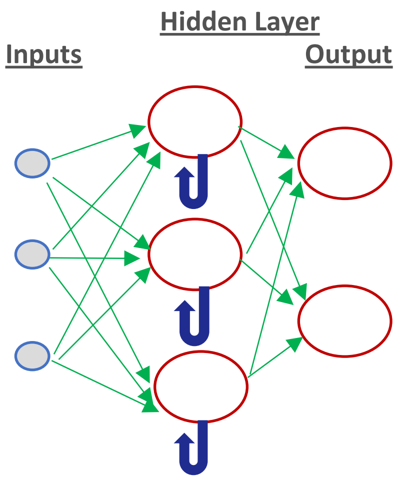


ARTIFICIAL NEURAL NETWORKS

- Layers of neurons connected to each other
  - Different types depending on input to output connections attributes
    - pattern
      - all-to-all as in fully connected networks
      - sparse some-to-some as in convolutional networks
    - weights
    - potentially unconstrained
    - might also be shared among different connections in the layer
- Variety of shapes and sizes depending on the application
- Deep Neural Networks characterized by several hidden layers (these between the input and output ones)
  - many authors set the threshold in 3

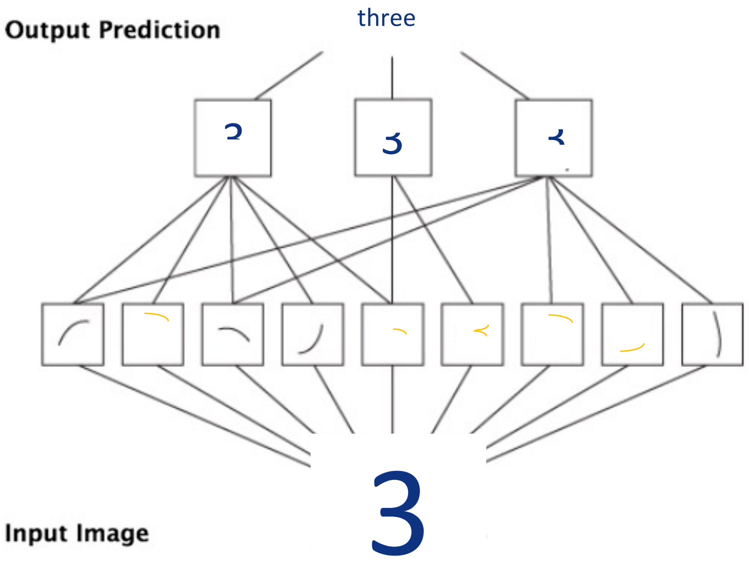


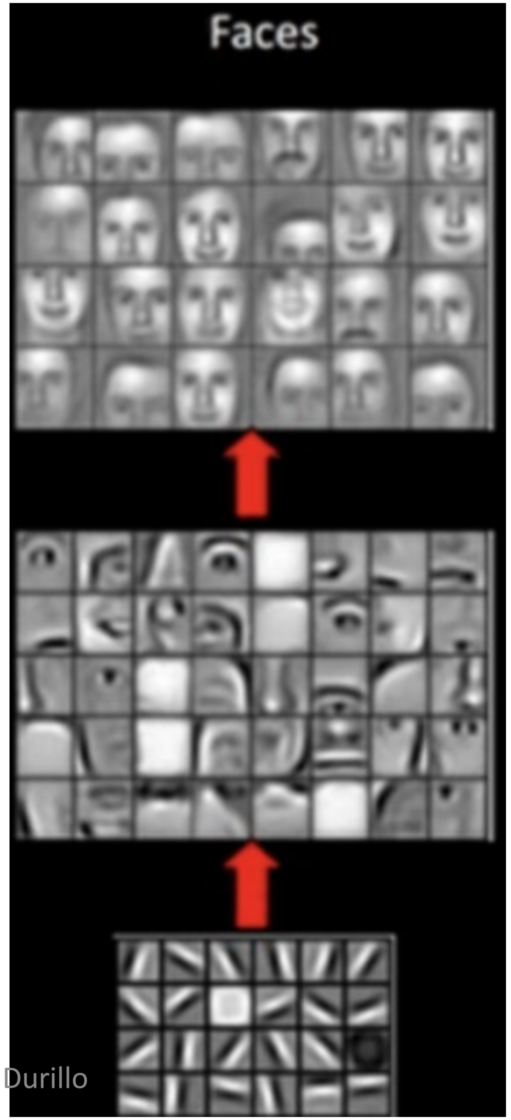
Feed-forward Neural Network
- Sequence of operations
- No additional memory requirements (only weights)
- Same output for same input



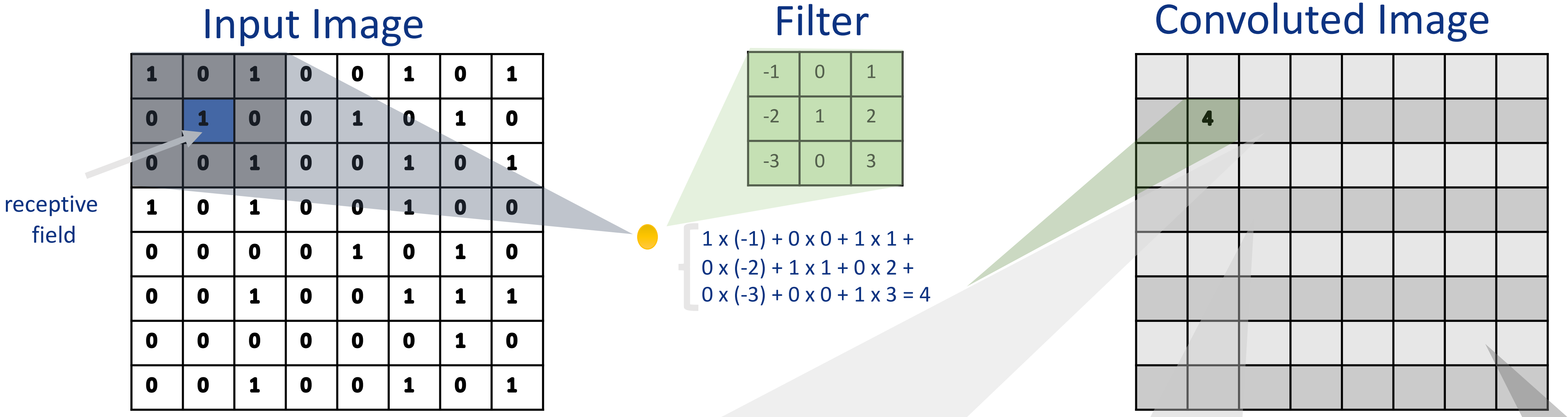
Recurrent Neural Network
- Limits to batch selection
- Additional memory requirements for neuron state
- Output for same input depends on state
- Hard to parallelize

NO MORE FEATURE ENGINEERING





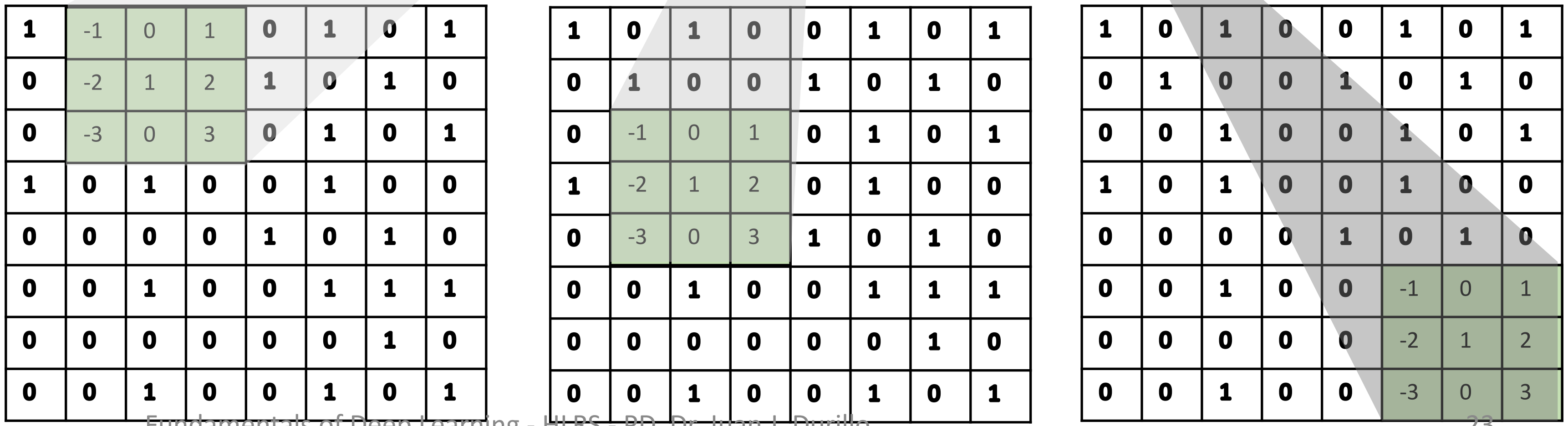
LEARNING FEATURES FROM DATA: CONVOLUTIONS



Filter is convoluted with all the pixels of the image

How many units the filter moves horizontally or vertically is called **stride** and can be different in both dimensions

The stride defines the size of the convoluted image



*The London skyline image is designed by Freepik

FILTERS

Input Image:

Can we get only vertical lines out of this picture?

1	**0**	**-1**

filter 1

1	**0**	**-1**
1	**0**	**-1**
1	**0**	**-1**

filter 2

try the code yourself (in octave)!

```
 I=imread(<path-to-image>);
GRAY=rgb2gray(I)
FILTER=[ 1 0 -1; 1 0 -1; 1 0 -1]; % filter 2
CONVOLUTED=conv2(GREY,FILTER);
Imwrite(CONVOLUTED, <path-to-result>);
```

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

filter 3

CONVOLUTIONAL LAYER - CONV

CONV computation (tensor notation)

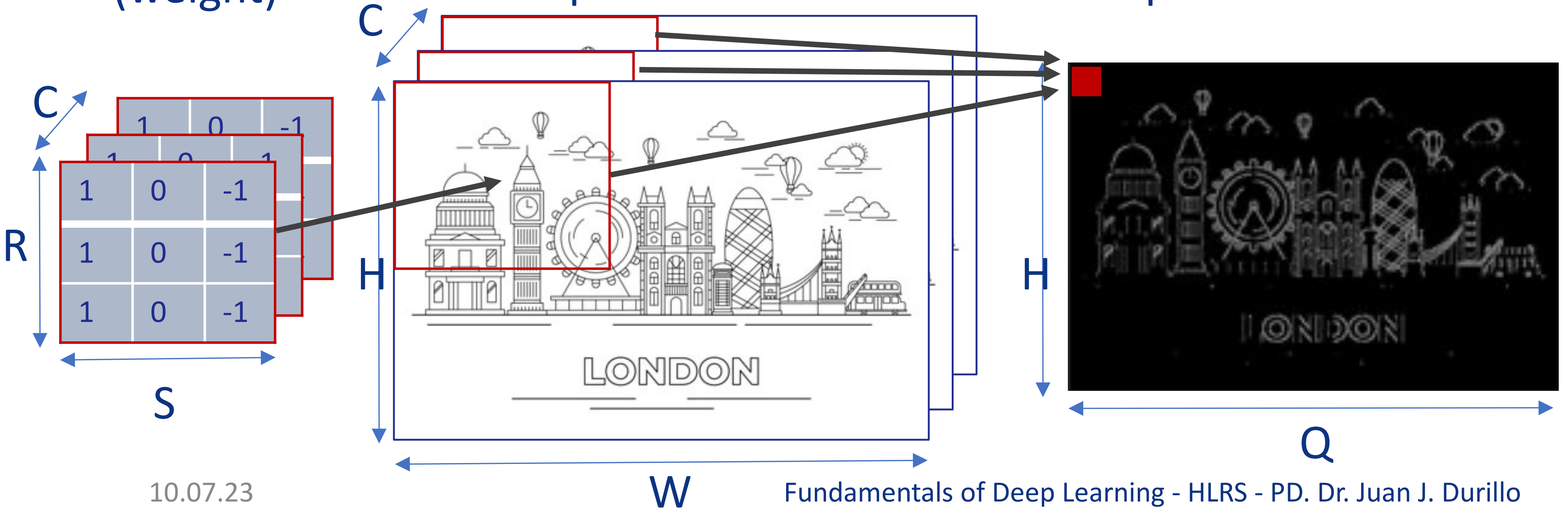
U is a stride of the input feature map

$$O_{nmpq} = \left(\Sigma_{crs}\, I_{nc(Up+r)(Uq+s)}\ x\ F_{mcrs} \right) + \ b_m$$

Filters (weight)

Input Feature Maps

Output Feature Map

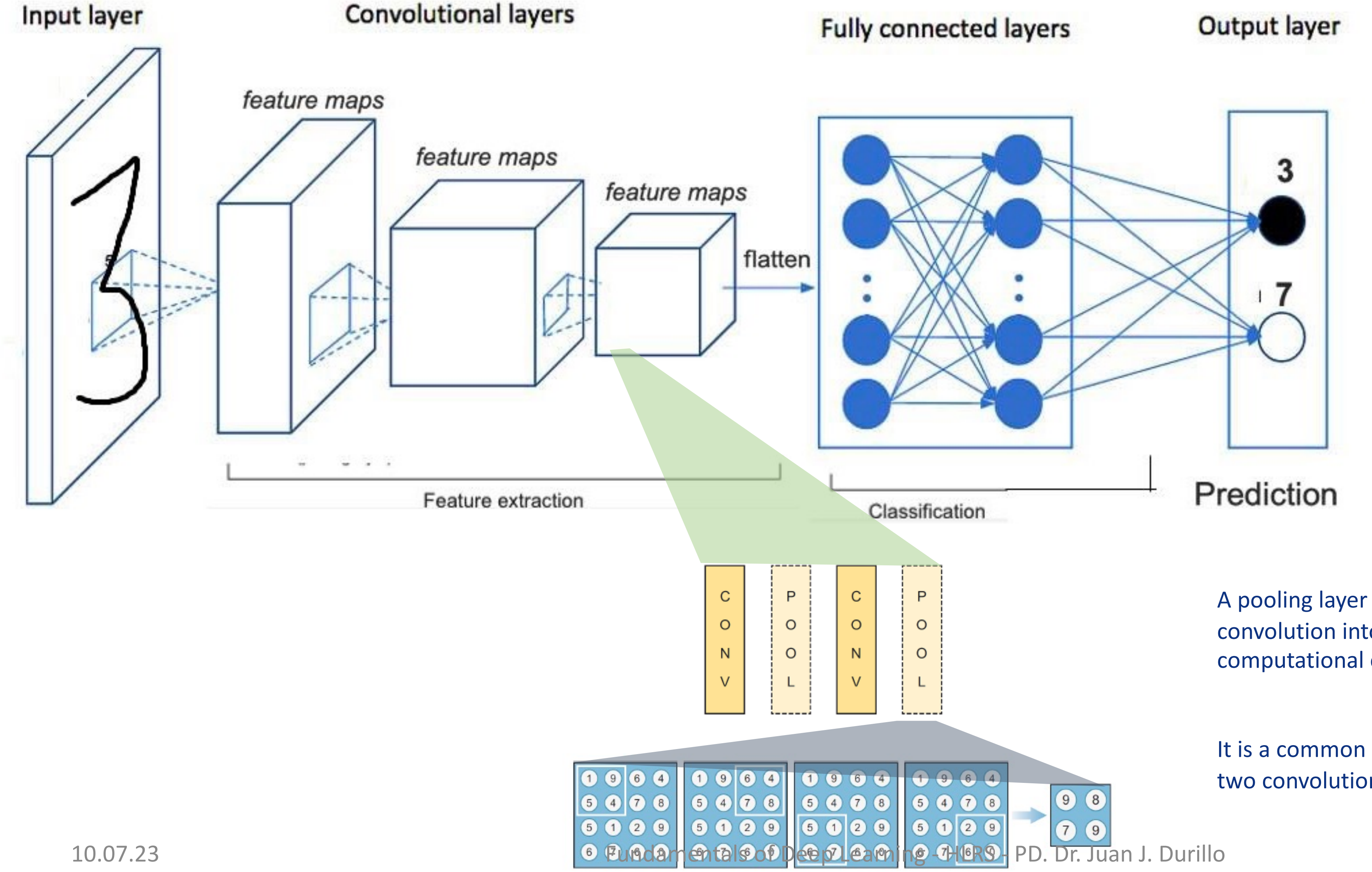


Shape Parameter	Description
M	# of filters (in the example not shown as is 1) It determines the # of output feature maps
C	# of channels of filters and input feature map
H/W	Input feature map spatial height and width
R/S	Filter spatial height and weight
P/Q	Output feature map heigh/width

```
int outputFMSize  = outputFMHeight * outputFMWidth;
int kernelSize     = k * k;
int inputFMWidth  = computeInputFMWidth(outputFMWidth,k);
int inputFMSize    = inputFMWidth * inputFMWidth;

for (int m = 0; m < M; m++)
  for (int n = 0; n < C; n++)
     for (int r = 0; r < H; r++)
       for (int c = 0; c < W; c++)
         for (int i = 0; i < R; i++)
           for (int j = 0; j < S; j++)
           {
            int outputIndex = m * (P*Q) + r * outputFMWidth + c;
            int kernelIndex = m * n * kernelSize + n * kernelSize + i * k + j;
            int inputIndex  = n * inputFMSize + (r+i) * inputFMWidth + c + j;
            output[outputIndex] += kernels[kernelIndex]*input[inputIndex];
           }
```

CONVOLUTIONAL NEURAL NETWORKS (CNN)



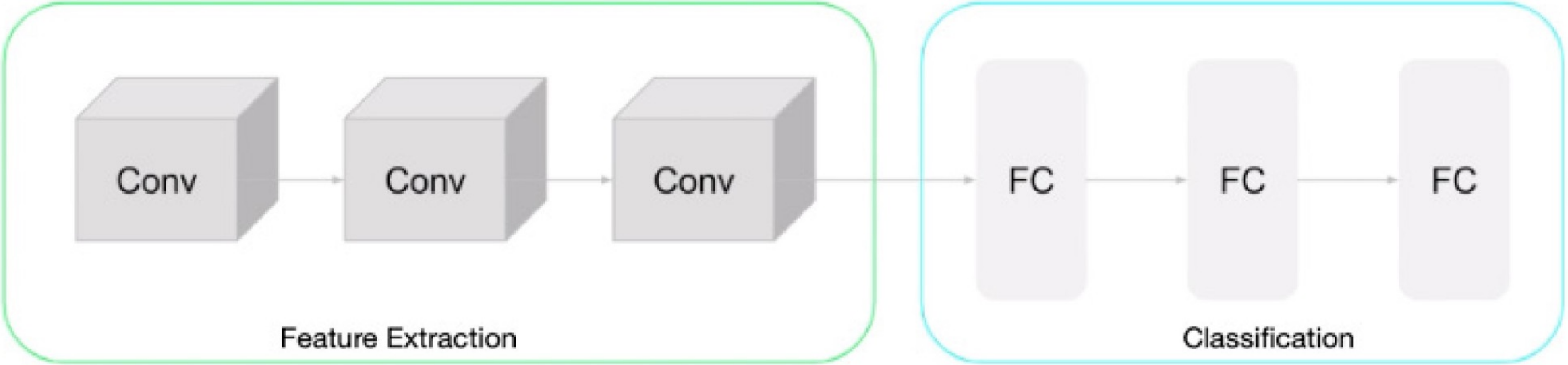
A pooling layer down sample the feature maps produced by a convolution into smaller number of parameters to reduce the computational complexity.

It is a common practice to add pooling layers after each one or two convolutions layers in the CNN architecture.

CNN ARCHITECTURE: A COMMON PATTERN AND ITS INFLUENCE

Convolutional Layers

Classification Layer

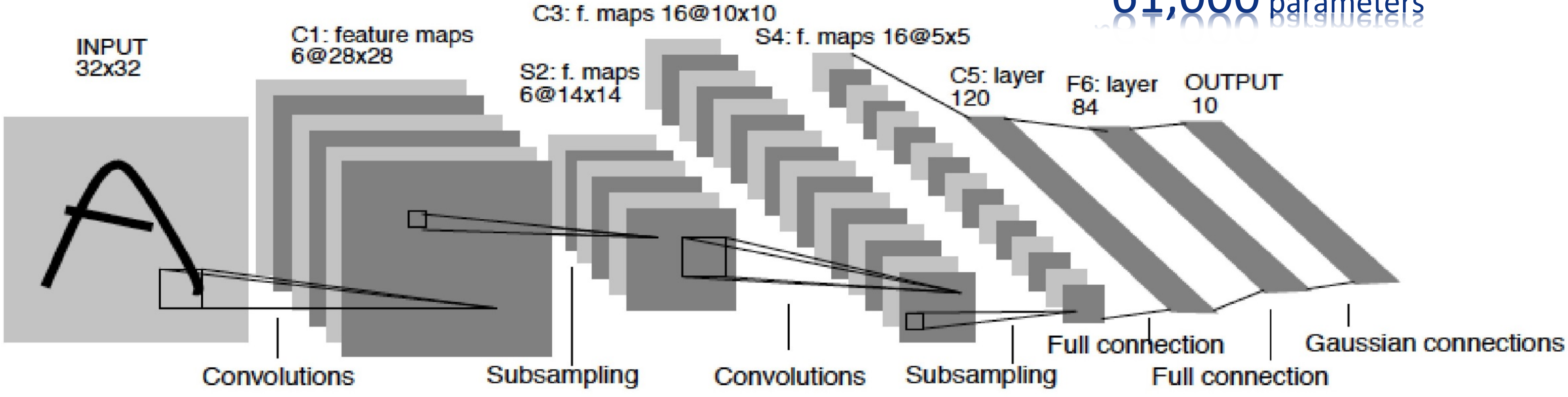


The execution time required during a forward pass through a neural network is bounded from below by the number of floating point operations (FLOPs).

This FLOP count depends on the deep neural network architecture and the amount of data.

LENET ARCHITECTURE

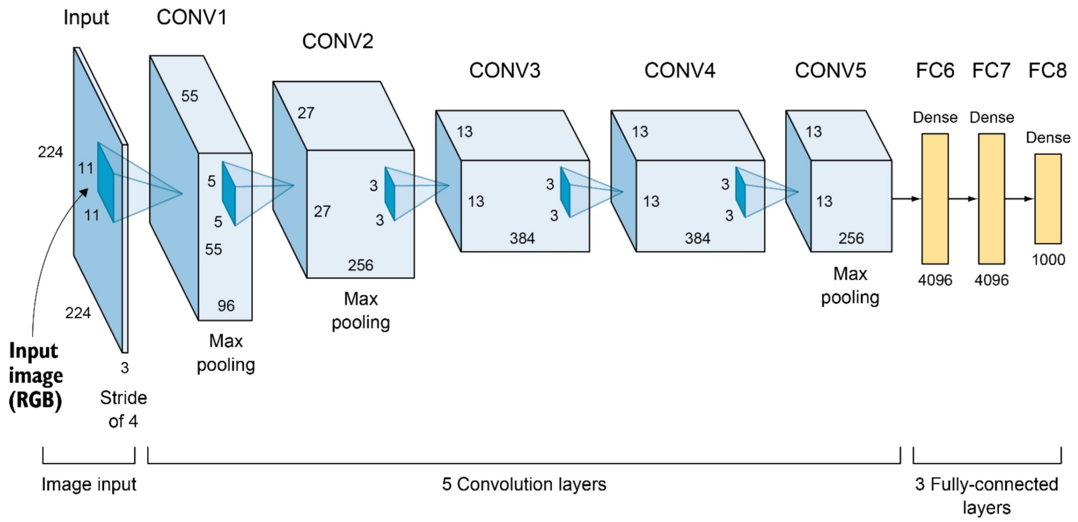
~61,000 parameters



Architecture summary :

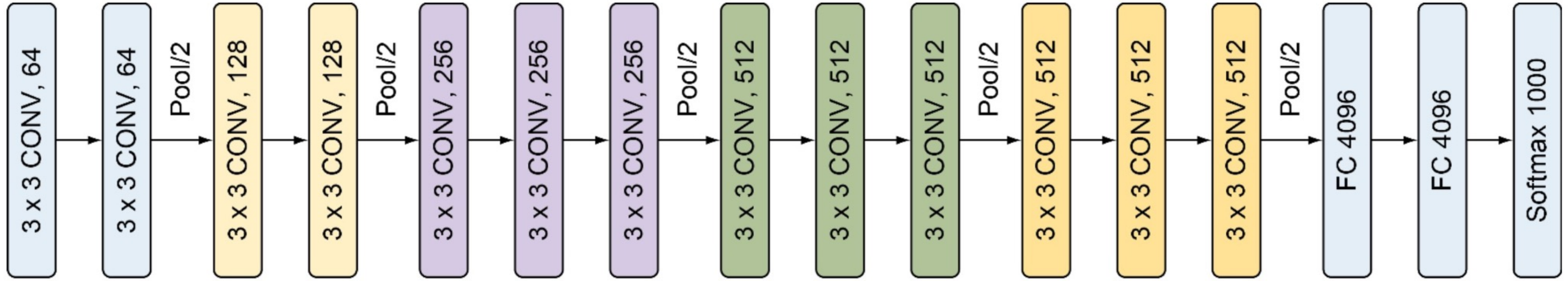
- 3 convolutional layers filters in all the layers equal to 5x5
  (layer 1 depth = 6, layer 2 depth = 16, layer 3 depth = 120)
- As activation function the tanh function is used

ALEXNET AND VGG ARCHITECTURES



~60,000,000 parameters

AlexNet

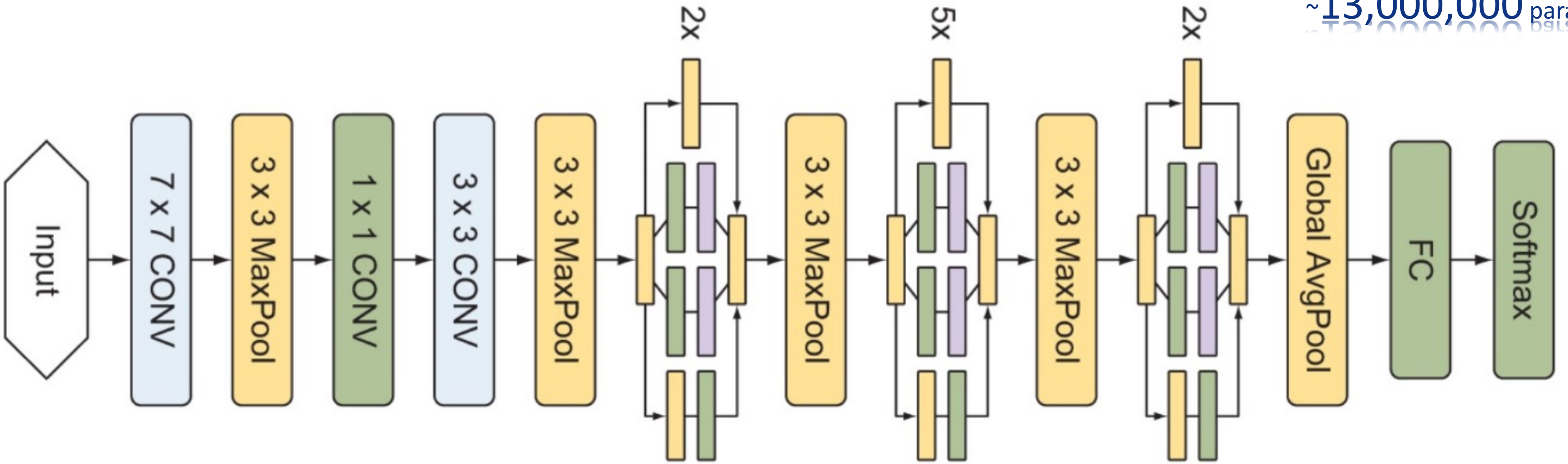


~138,000,000 parameters

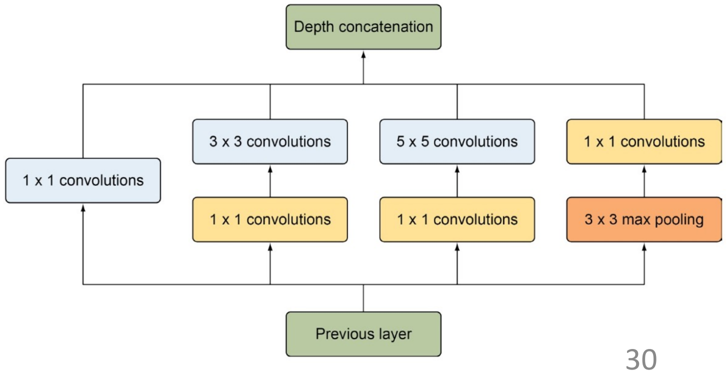
VGG16

GOOGLENET

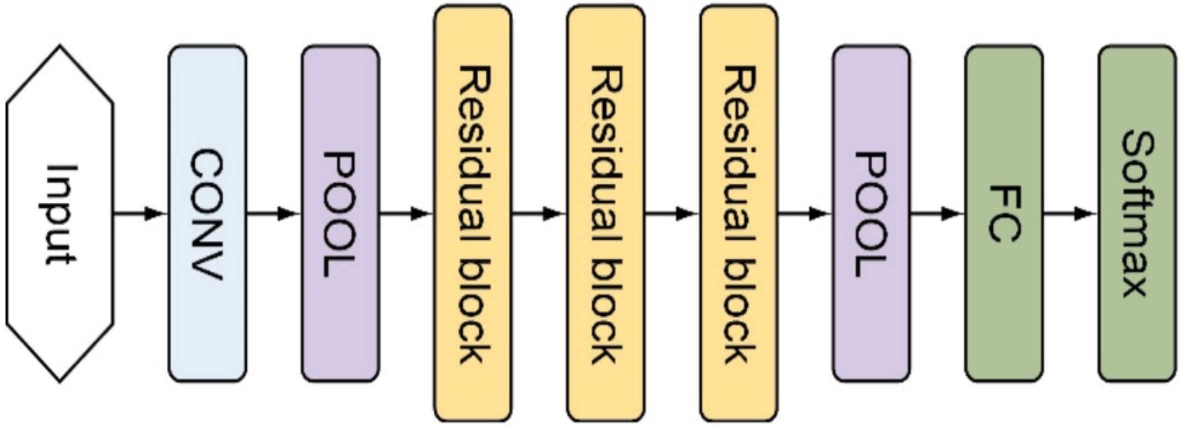
~13,000,000 parameters

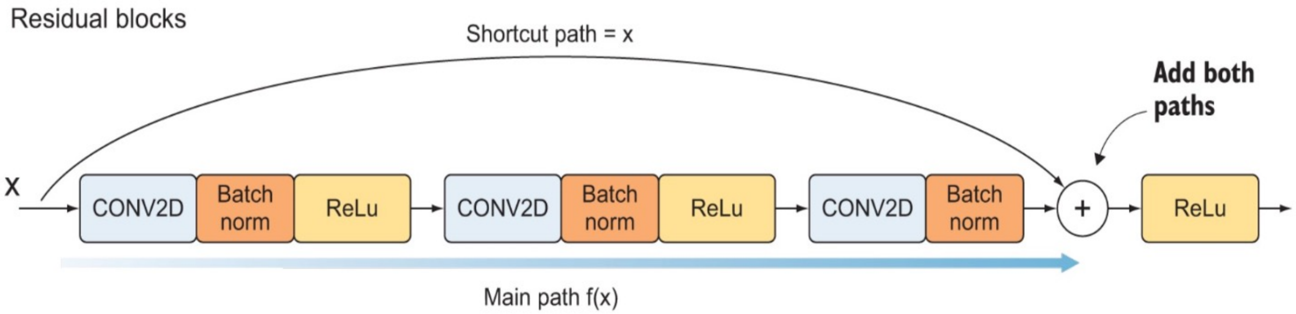


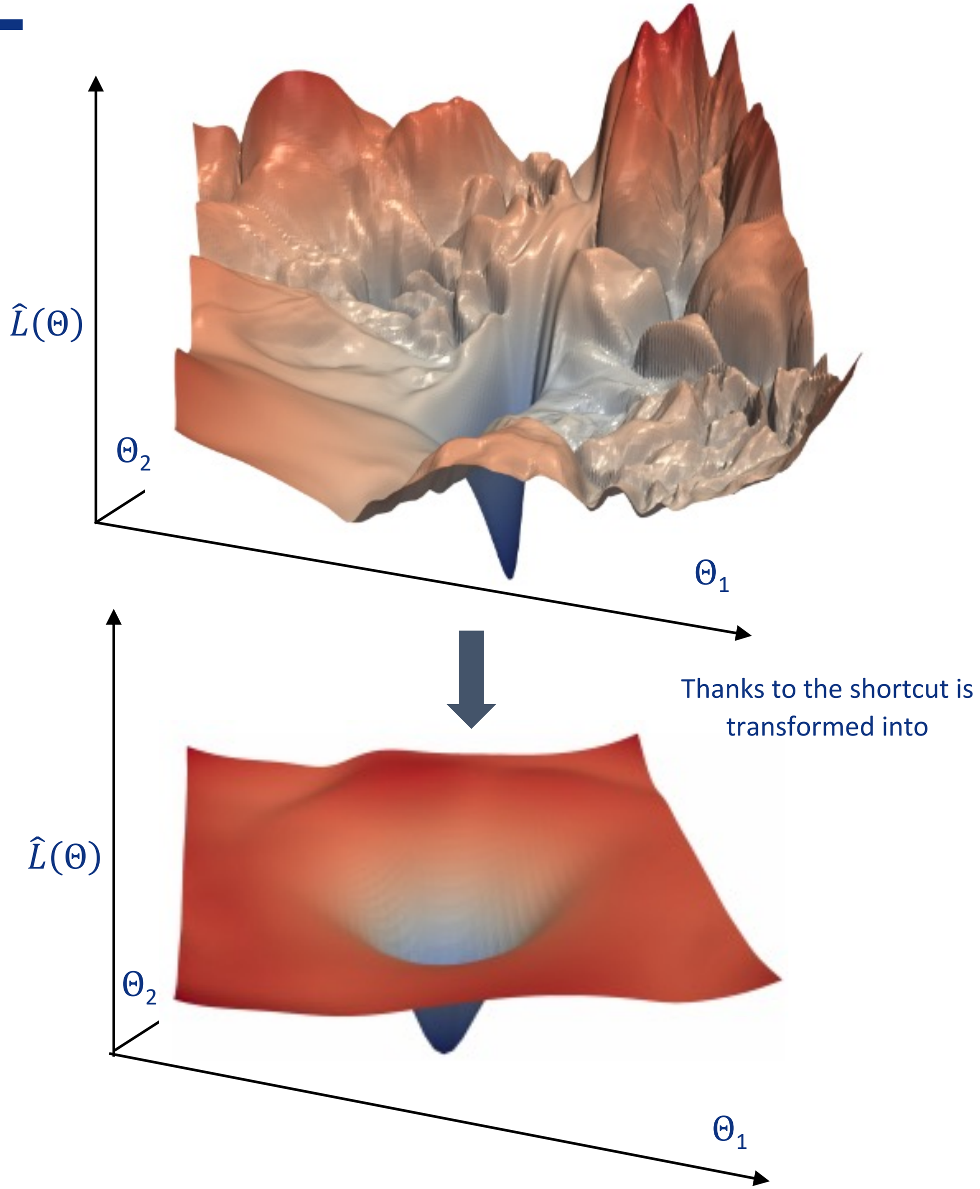
- What is the best kernel size for each layer?
- Concatenating filters instead of stacking them for reducing computational expenses



RESTNET

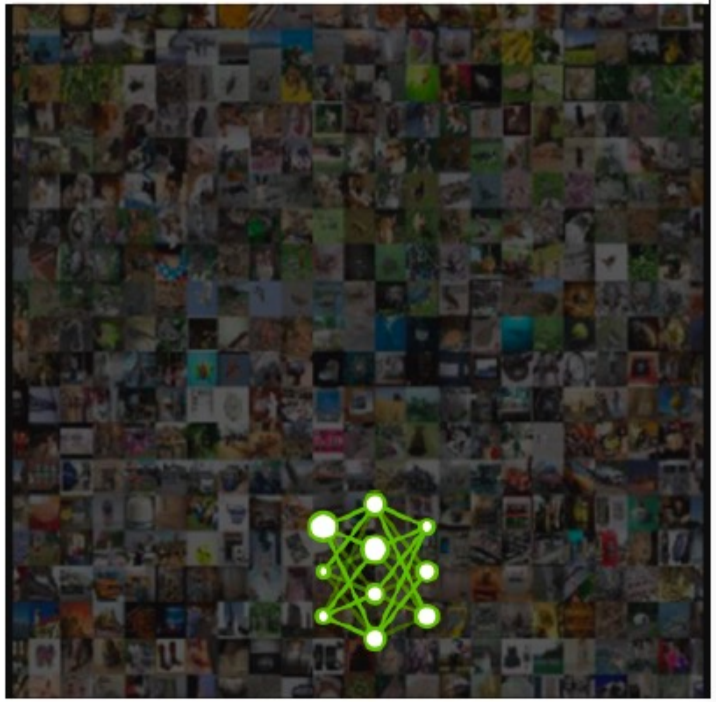

Input
CONV
POOL
Residual block
Residual block
Residual block
POOL
FC
Softmax


Residual blocks
Shortcut path = x
Add both paths
x
CONV2D
Batch norm
ReLu
CONV2D
Batch norm
ReLu
CONV2D
Batch norm
+
ReLu
Main path f(x)


$\hat{L}(\Theta)$
Θ_2
Θ_1
Thanks to the shortcut is transformed into
$\hat{L}(\Theta)$
Θ_2
Θ_1

INCREASING COMPLEXITY

7 Exaflops
60 Million Parameters



2015 - Microsoft ResNet
Superhuman Image Recognition

20 Exaflops
300 Million Parameters

2016 - Baidu Deep Speech 2
Superhuman Voice Recognition

100 Exaflops
8700 Million Parameters

2017 - Google Neural Machine Translation
Near Human Language Translation

CONVOLUTIONAL DNN: CNNS

Metrics	LeNet	AlexNet	Overfeat Fast	VGG 16	Google LeNet	ResNet 50
Top- 5 error	N/A	16.4	14.2	7.4	6.7	5.3
Input Size	28x28	227x227	231x231	224x224	224x224	224x224
# CONV layers	2	5	5	13	57	53
Filter Sizes	5	3,5,11	2,5,11	3	1,3,5,7	1,3,7
# Weights	2.6 k	2.3 M	16 M	14.7 M	6.0 M	23.5 M
# filters	20,50	94-384	96-1024	64-512	16-384	64-2048
MACs	283 k	666 M	2.67 G	15.3 G	1.43 G	3.86 G
# FC layers	2	3	3	3	1	1
# Weights	58 k	58.6 M	130 M	124 M	1 M	2 M
MACs	58 K	58.6 M	130 M	124 M	1 M	2 M
Total weights	60k	61 M	146 M	138 M	7 M	25.5 M
Total MACs	341 k	724 M	2.8 G	15.5 G	1.43 G	3.9 G

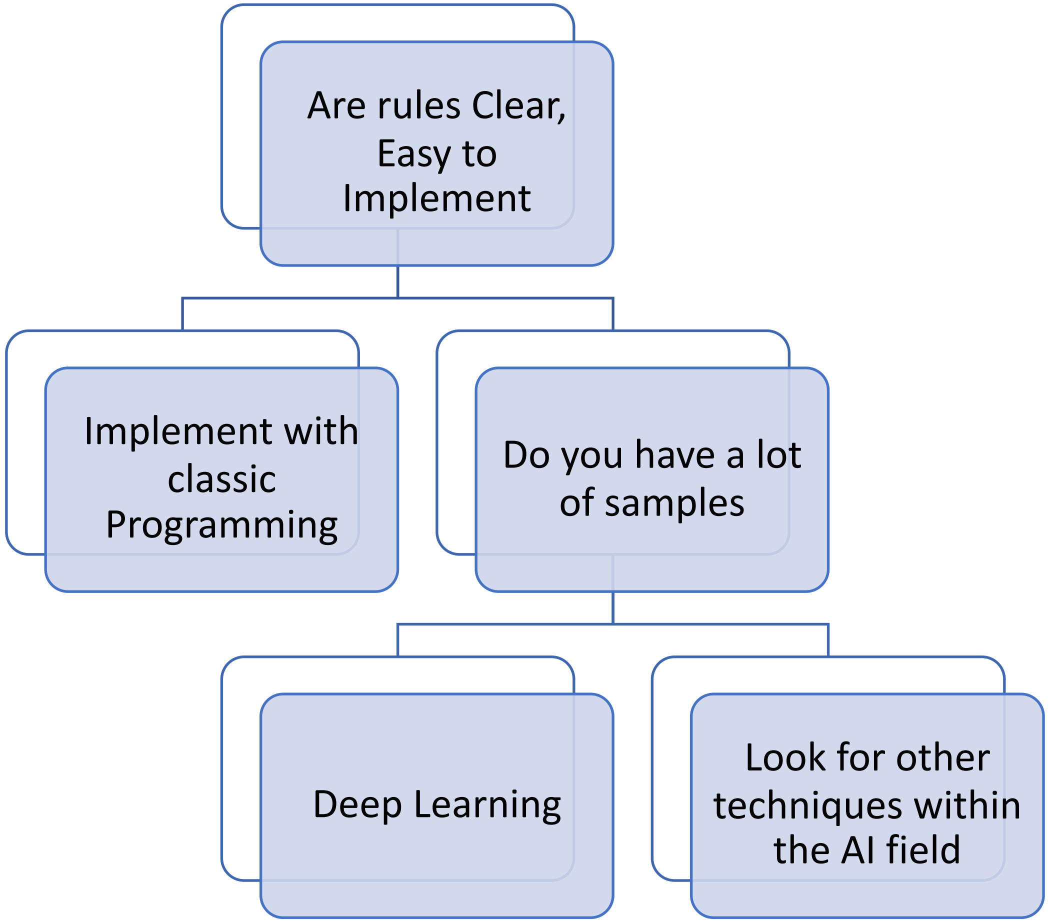
SYNCHRONIZATION POINT

- Brief introduction to Deep Learning with emphasis in Deep Convolutional Neural Networks
- Review of basic concepts: from perceptron to the learning task
- Debrief of most important concepts of neural network architectures

Expert Systems
Define a set of rules for classification
Program those rules into the computer
Feed it examples, and the program uses the rules to classify

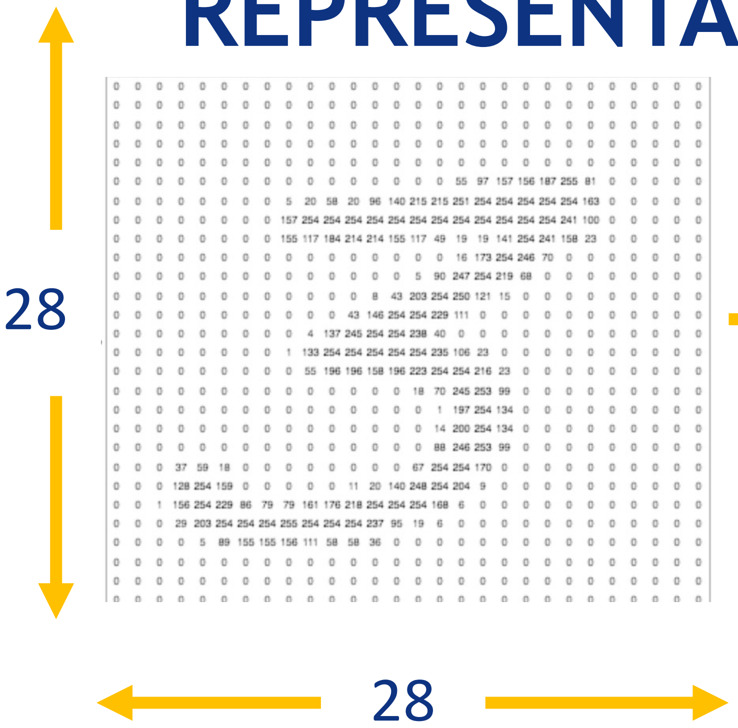
Deep Learning
Show model the examples with the answer of how to classify
Model takes guesses, we tell it if it's right or not
Model learns to correctly categorize as it's training. The system learns the rules on its own

WHEN TO CHOOSE DEEP LEARNING



FIRST EXERCISE: HELLO WORLD

REPRESENTATION (FORMAT, PREPARATION, …)



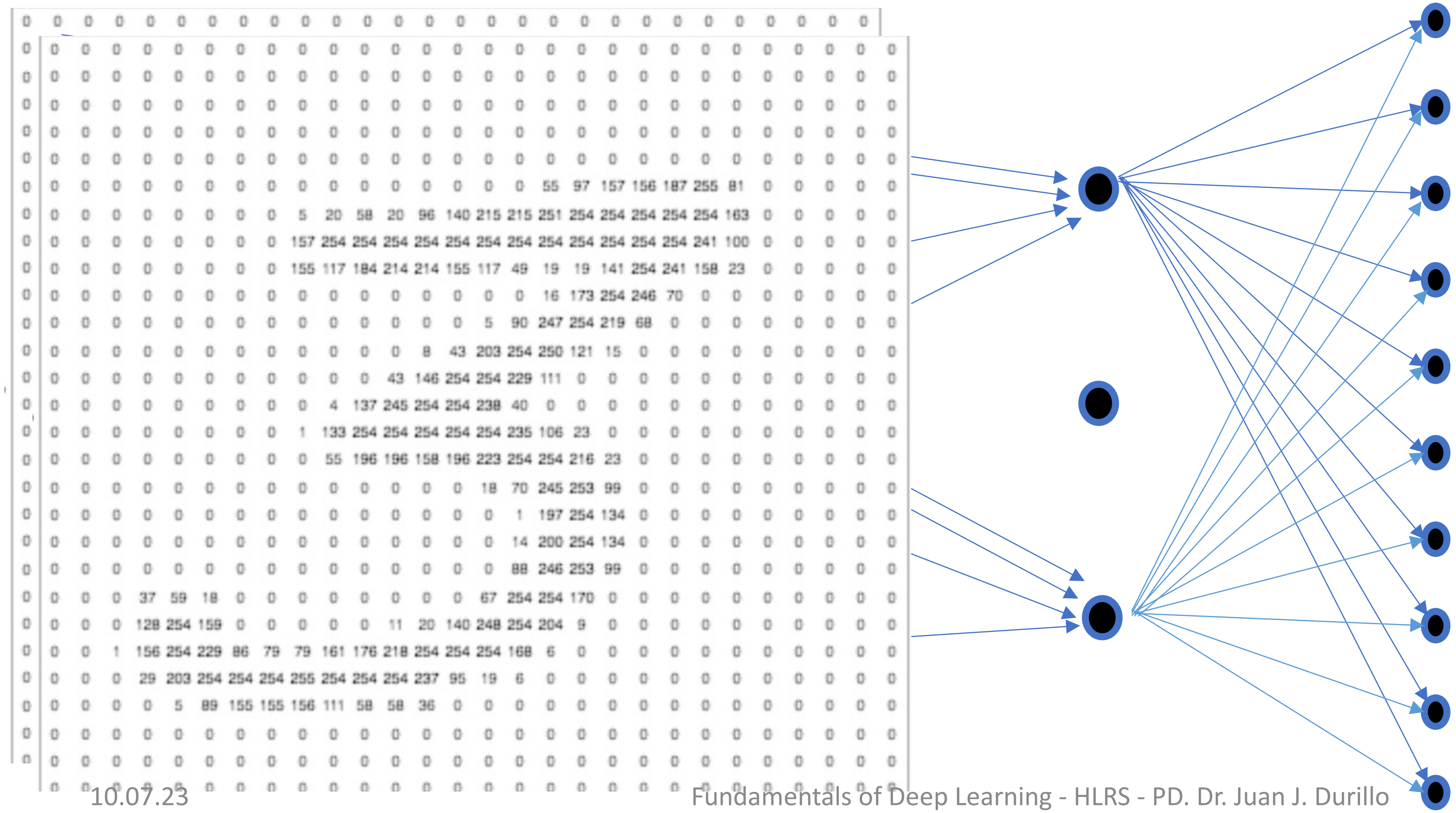
[0,0,0,…, 55, 97, 157, , 156, 187, 255, 81, 0, 0, …]

Inputs and Targets

0 → [1,0,0,0,0,0,0,0,0,0]

1 → [0,1,0,0,0,0,0,0,0,0]

NEURAL NETWORKS



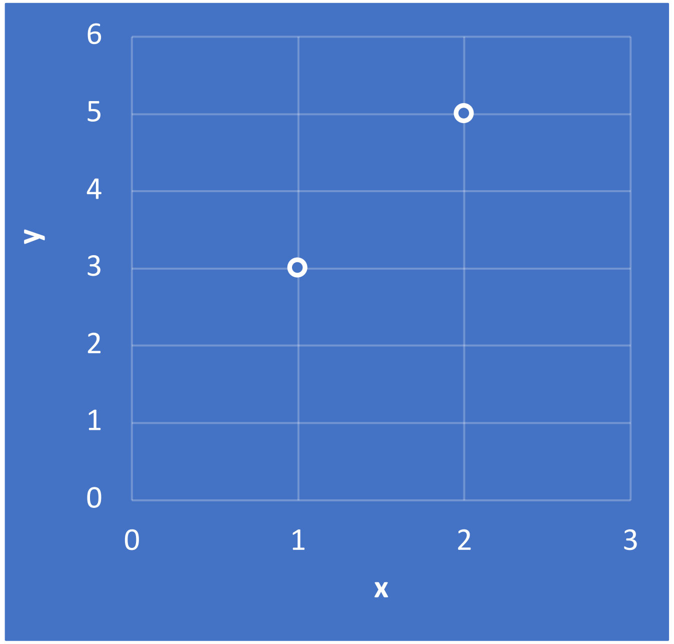
How many layers?
What size?
What activation Functions?
What loss function?

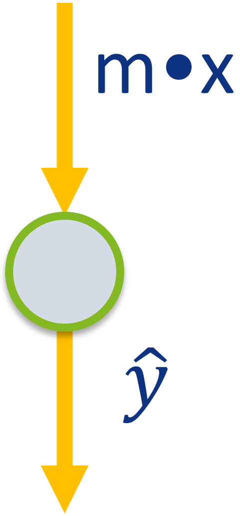
UNDERSTANDING TRAINING: A SIMPLER MODEL

A SIMPLER MODEL

$$y = mx + b$$

x	y
1	3
2	5





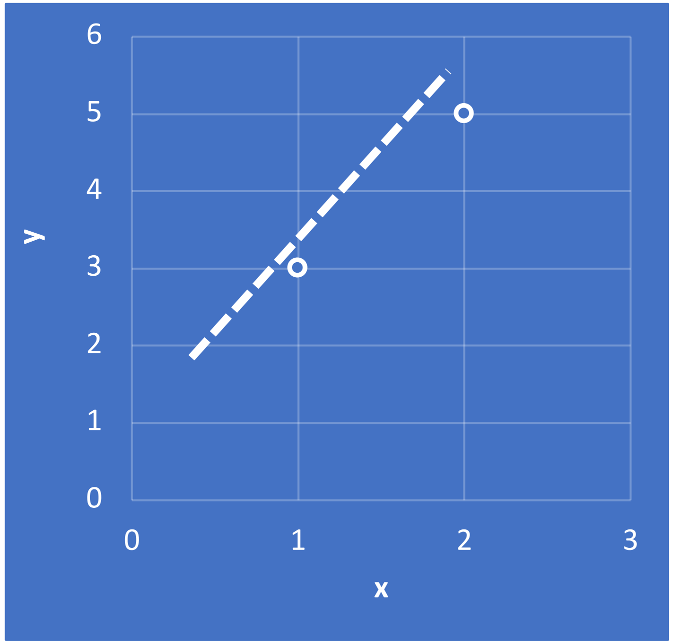
$$m = ?$$

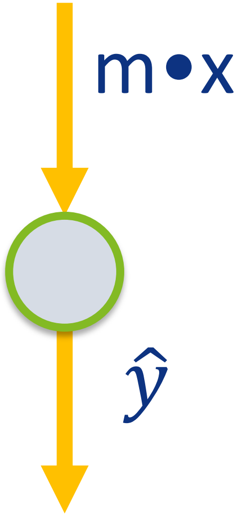
$$b = ?$$

A SIMPLER MODEL

$$y = mx + b$$

x	y
1	3
2	5





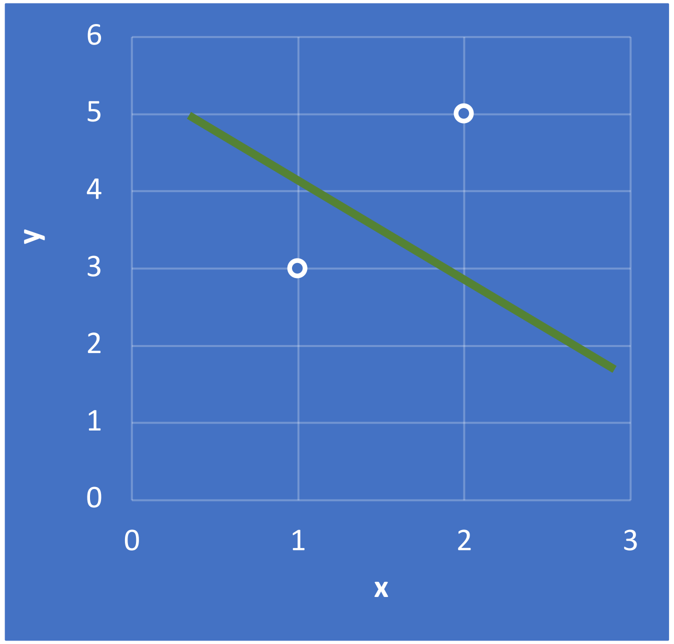
$m = ?$

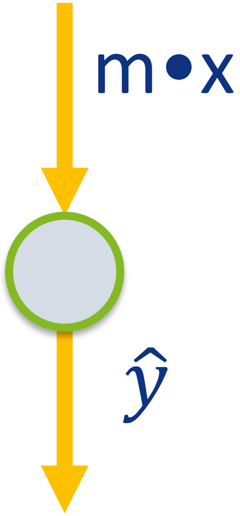
$b = ?$

A SIMPLER MODEL

$$y = mx + b$$

x	y	$\hat{y}$
1	3	4
2	5	3





Start Random

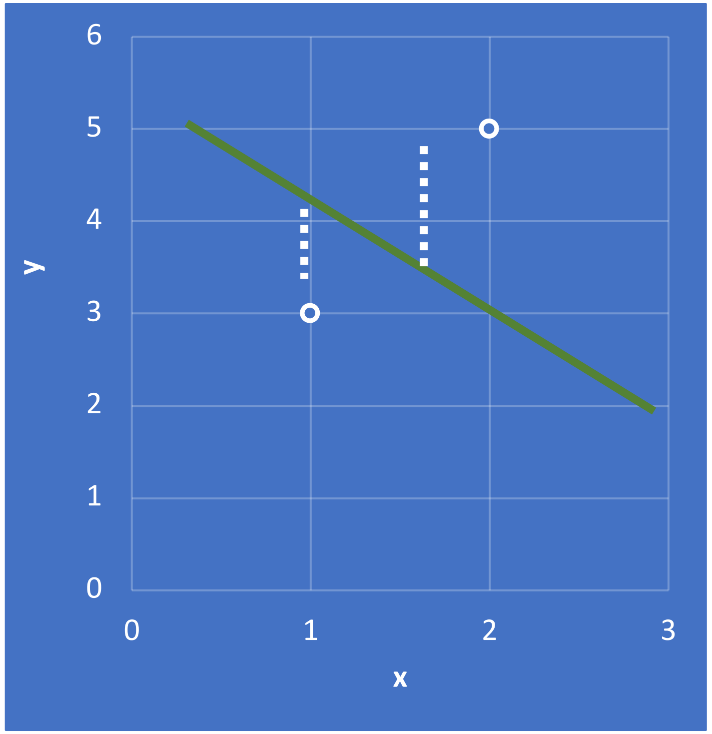
$$m = -1$$

$$b = 5$$

A SIMPLER MODEL

$$y = mx + b$$

x	y	$\hat{y}$	err^2
1	3	4	1
2	5	3	4
		MSE =	2.5
		RMSE =	1.6



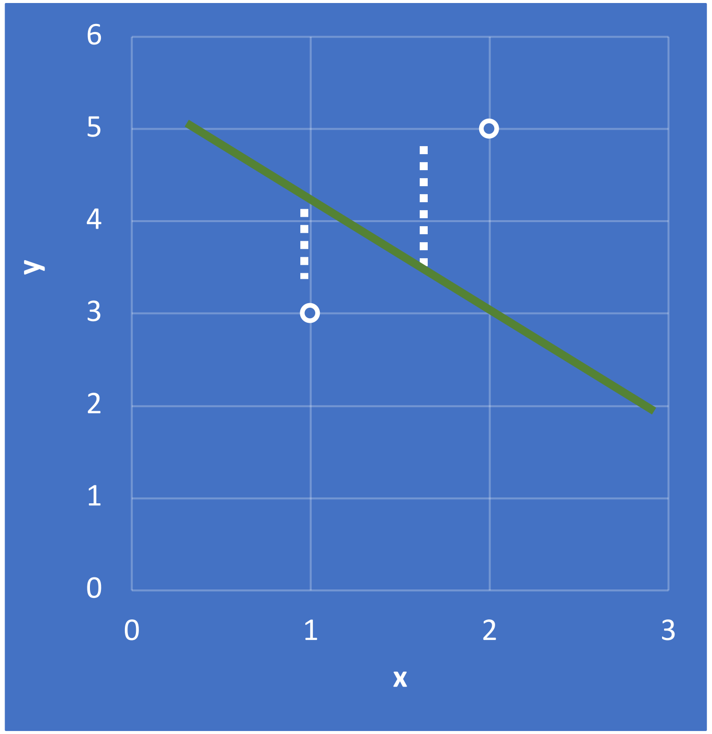
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

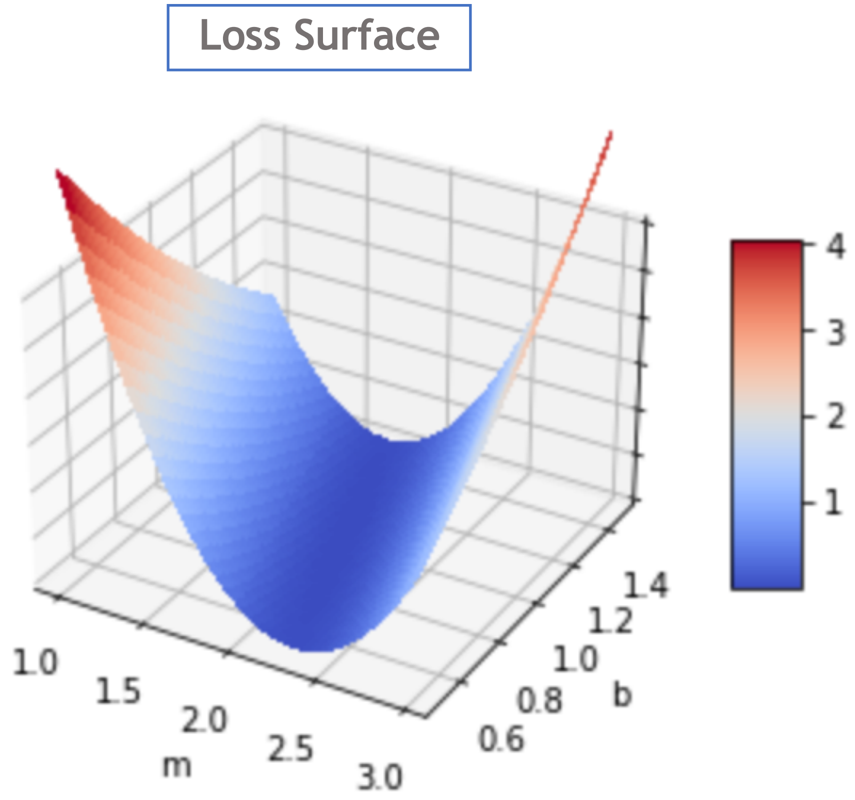
A SIMPLER MODEL

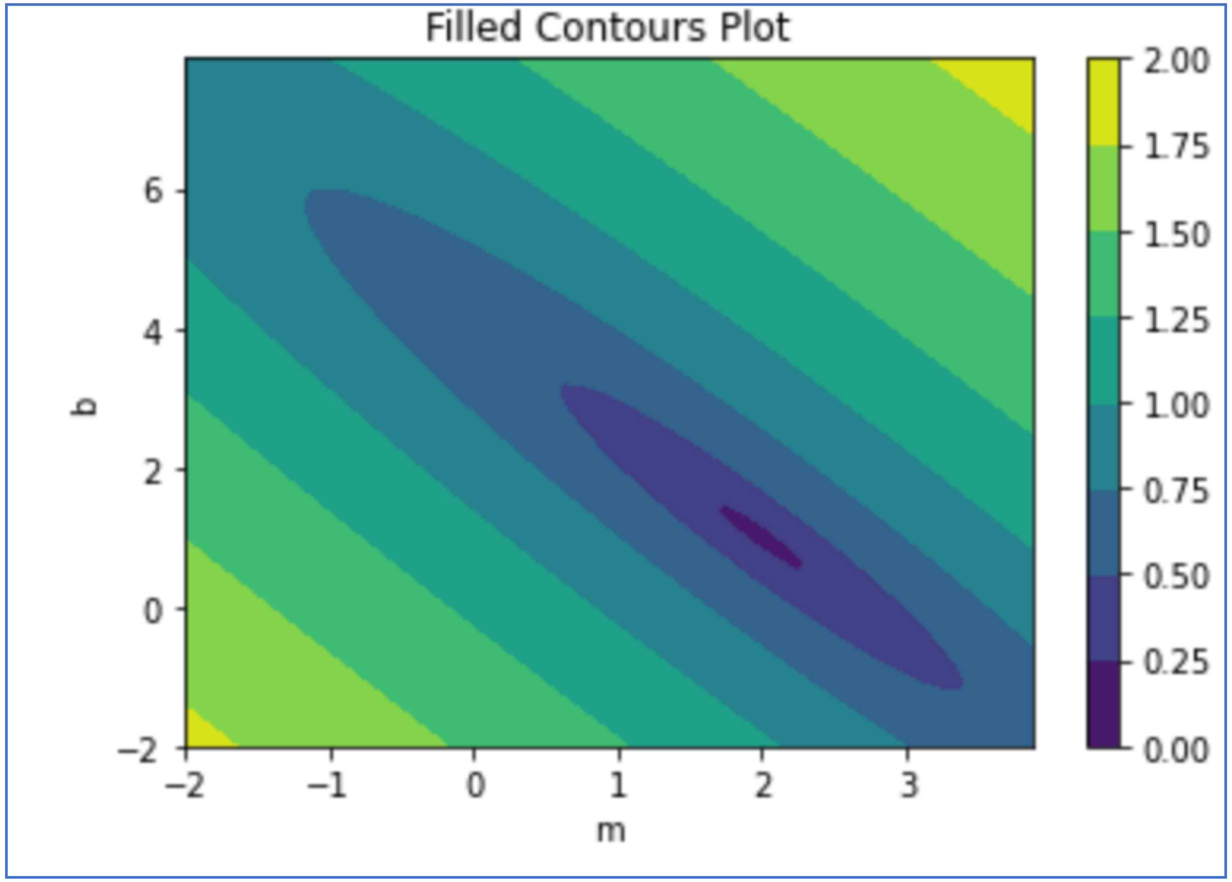
$$y = mx + b$$

x	y	$\hat{y}$	err^2
1	3	4	1
2	5	3	4
		MSE =	2.5
		RMSE =	1.6

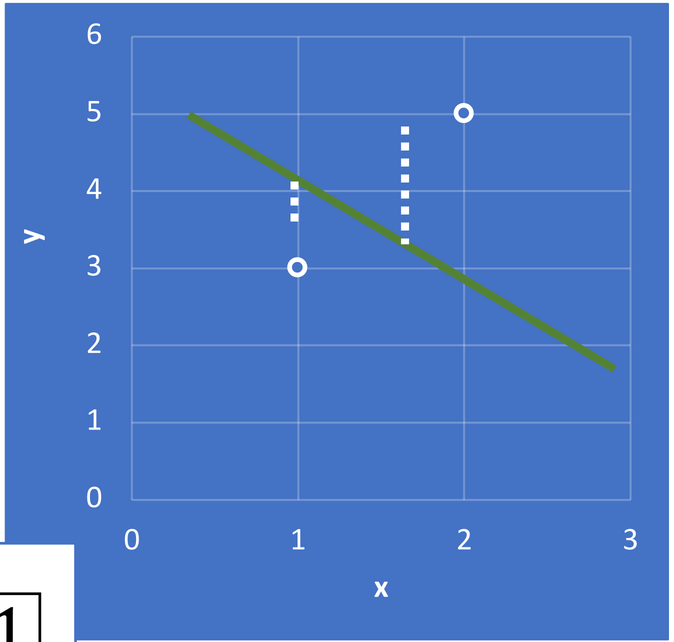


THE LOSS CURVE



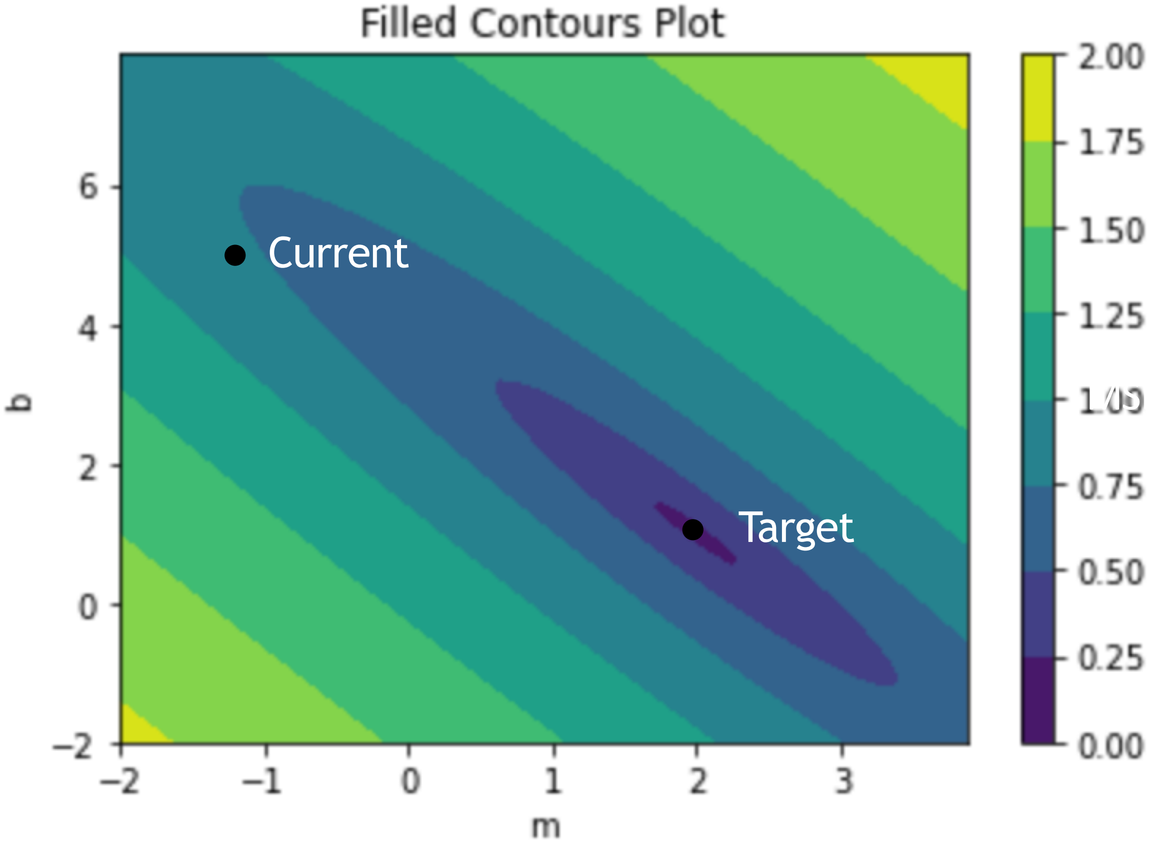


THE LOSS CURVE

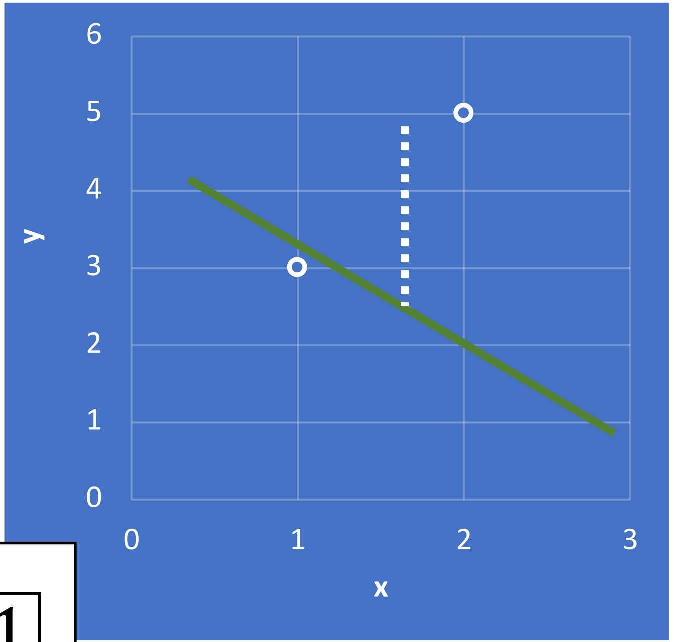


$$\boxed{m = -1}$$

$$\boxed{\mathrm{b} = 5}$$

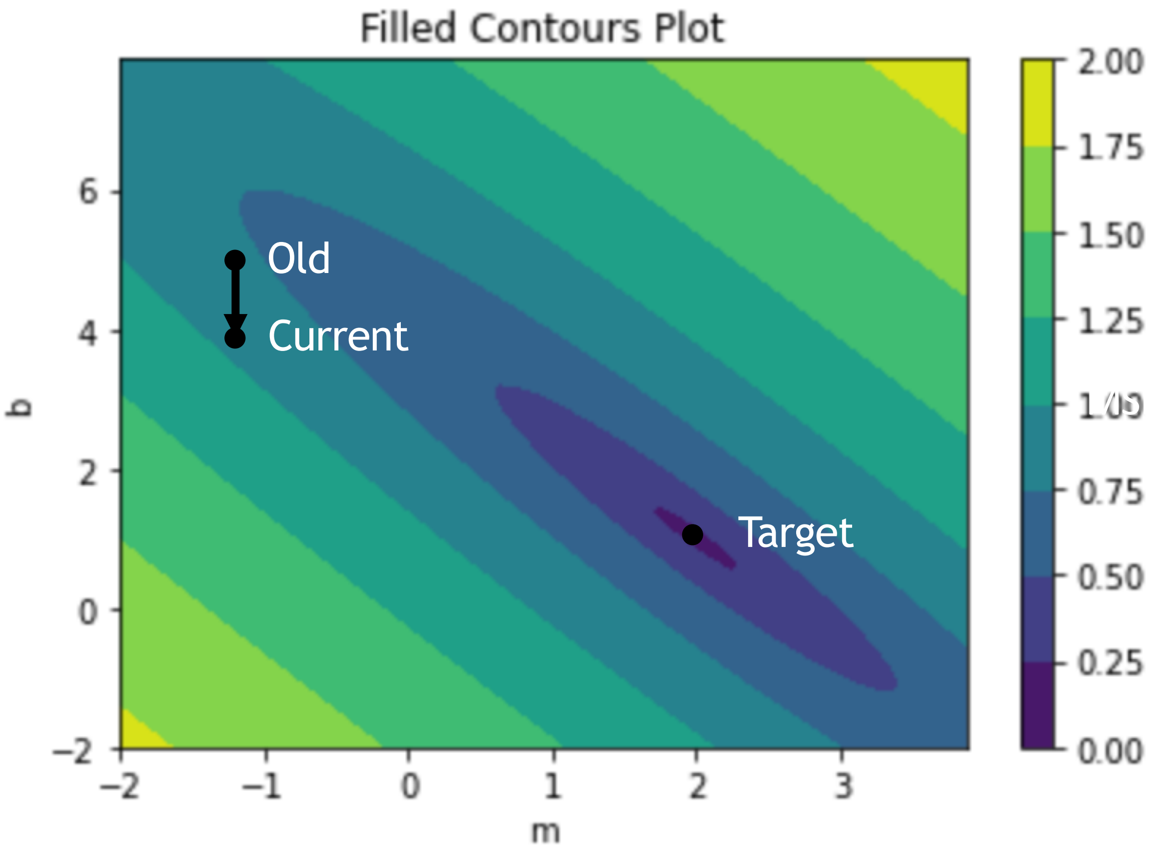


THE LOSS CURVE

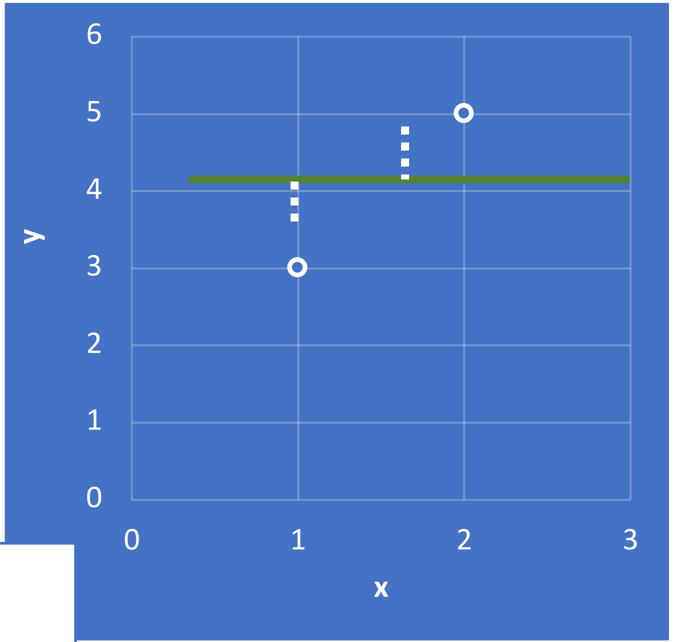


$$\boxed{m = -1}$$

$$\boxed{b = 4}$$

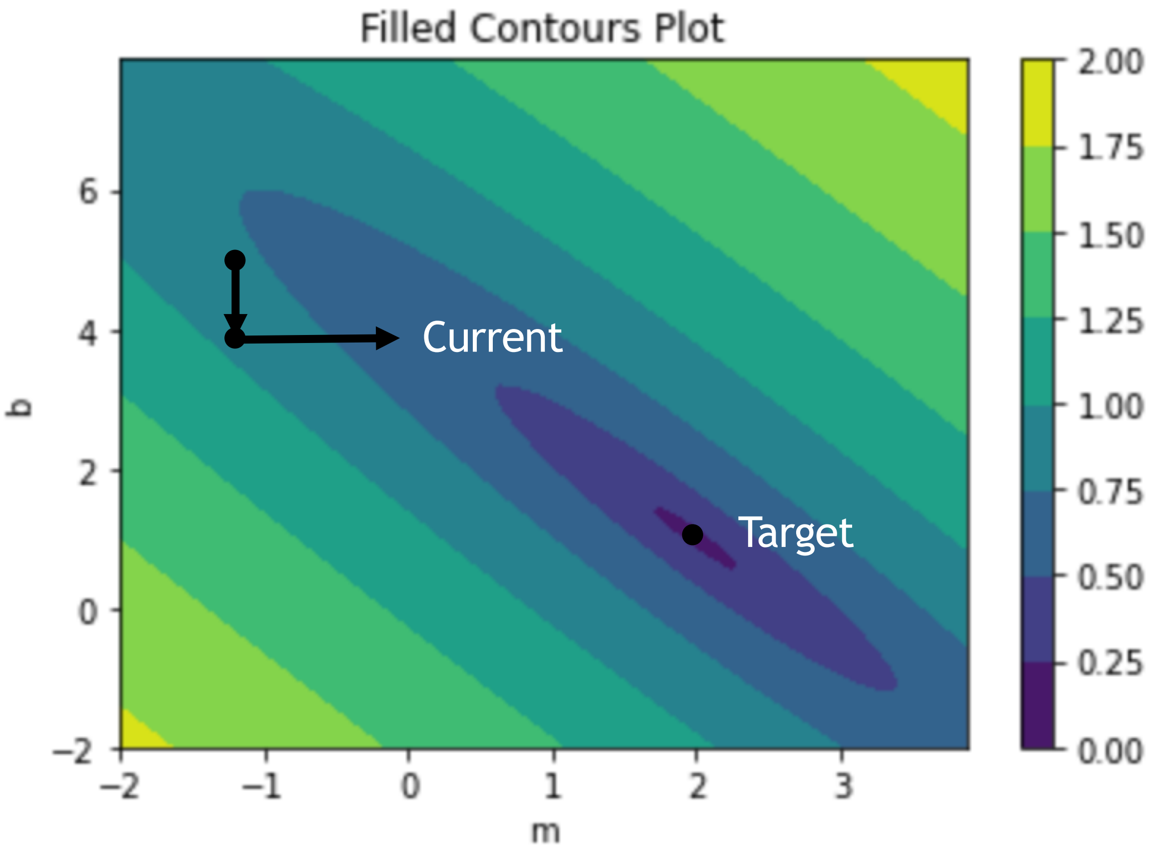


THE LOSS CURVE



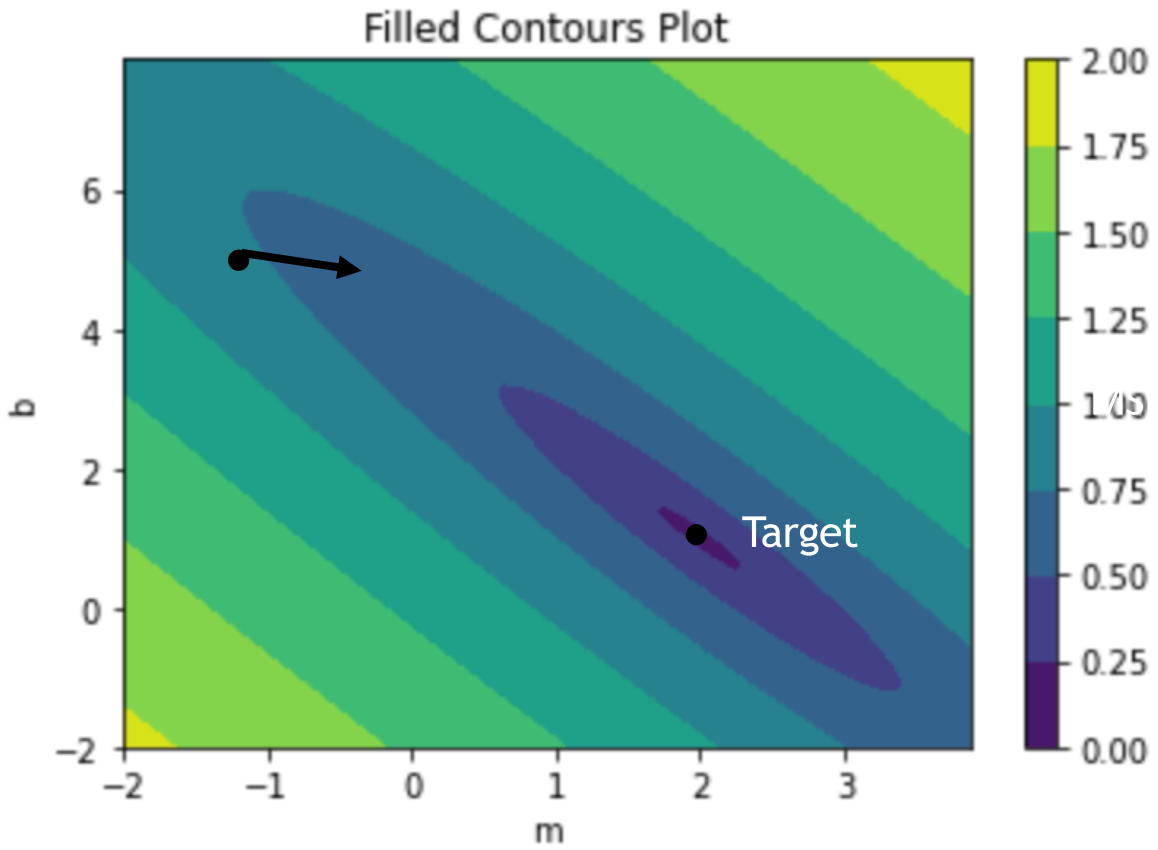
$\boxed{m = 0}$

$\boxed{b = 4}$



THE LOSS CURVE

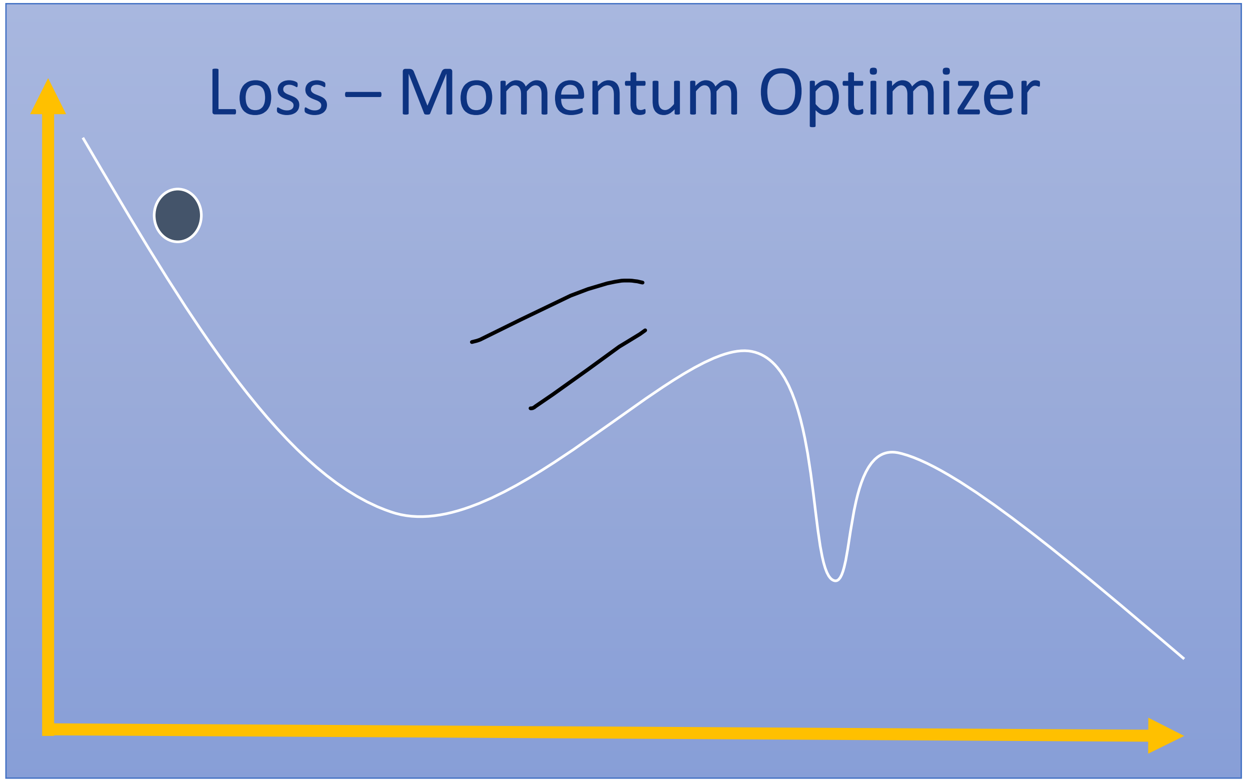
The Gradient	Which direction loss decreases the most
λ: The learning rate	How far to travel
Epoch	A model update with the full dataset
Batch	A sample of the full dataset
Step	An update to the weight parameters



THE LOSS CURVE

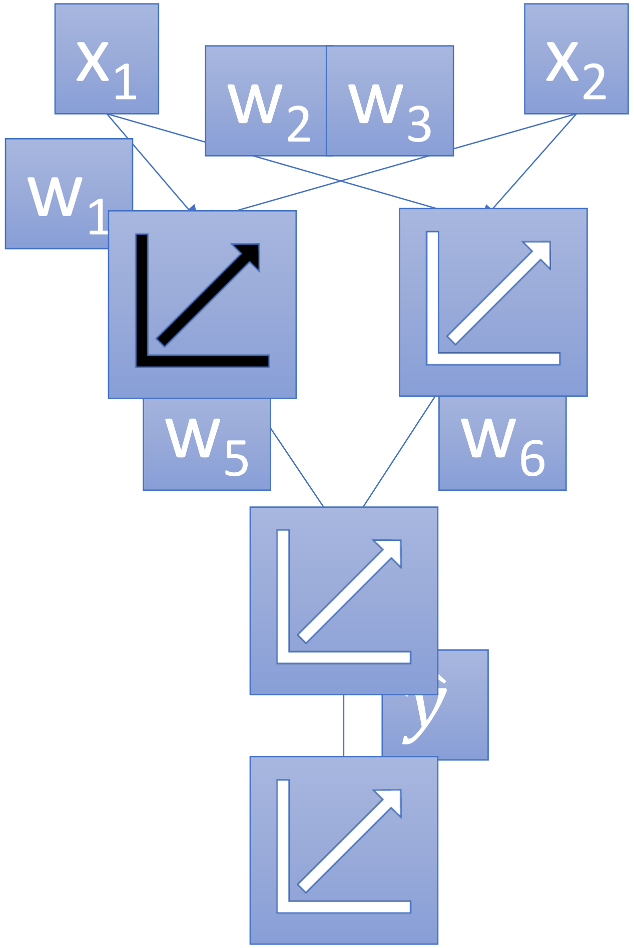
The Gradient	Which direction loss decreases the most
λ: The learning rate	How far to travel
Epoch	A model update with the full dataset
Batch	A sample of the full dataset
Step	An update to the weight parameters

OPTIMIZERS



- Adam
- Adagrad
- RMSprop
- SGD

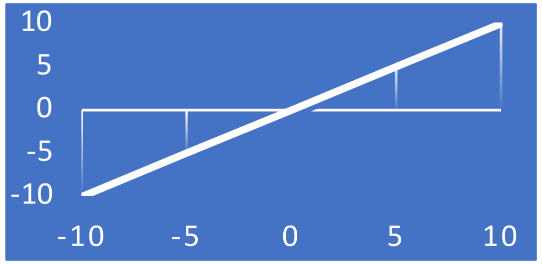
BUILDING A NETWORK



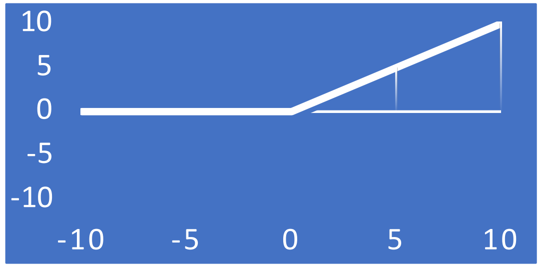
- Neurons organized in layers
- Connections between layers
  - 1 to all, some to some
  - If all regressions are linear, then output will also be a linear regression
- Activation functions

ACTIVATION FUNCTIONS

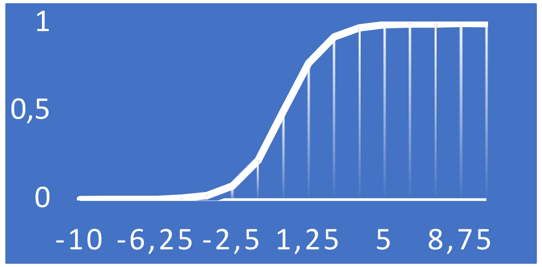
Linear



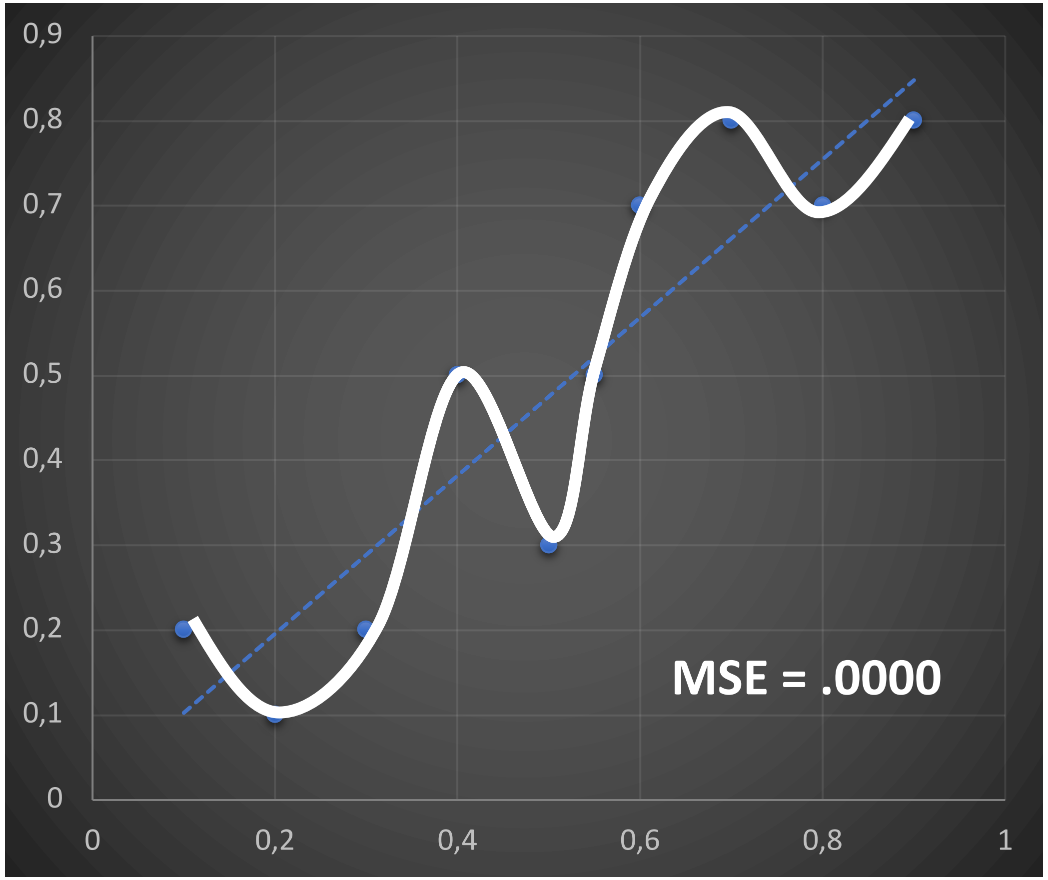
ReLu

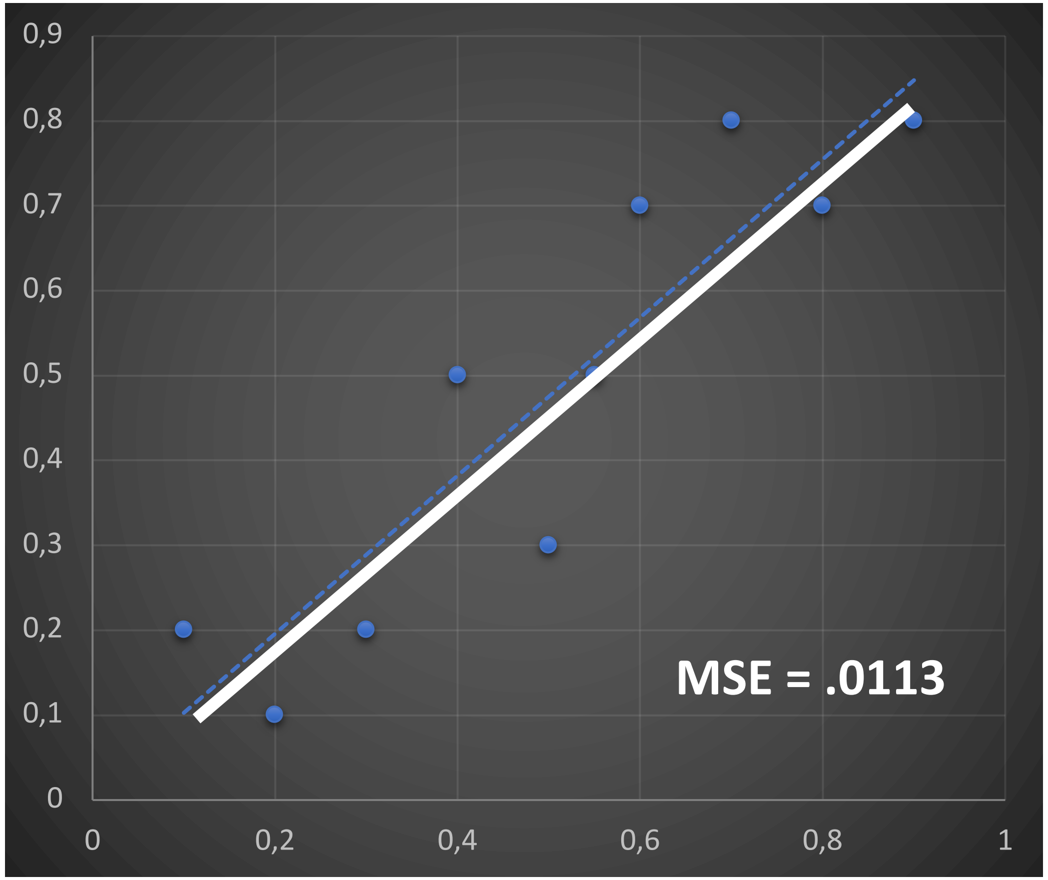


Sigmoid

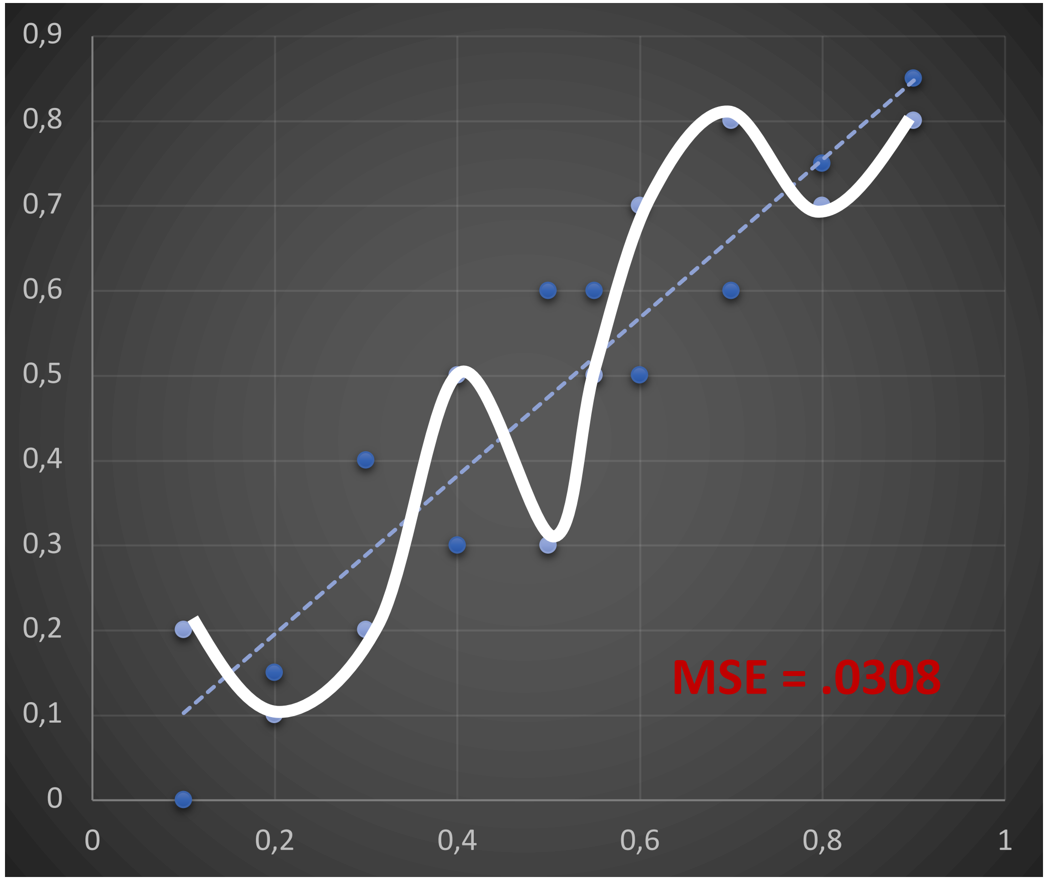


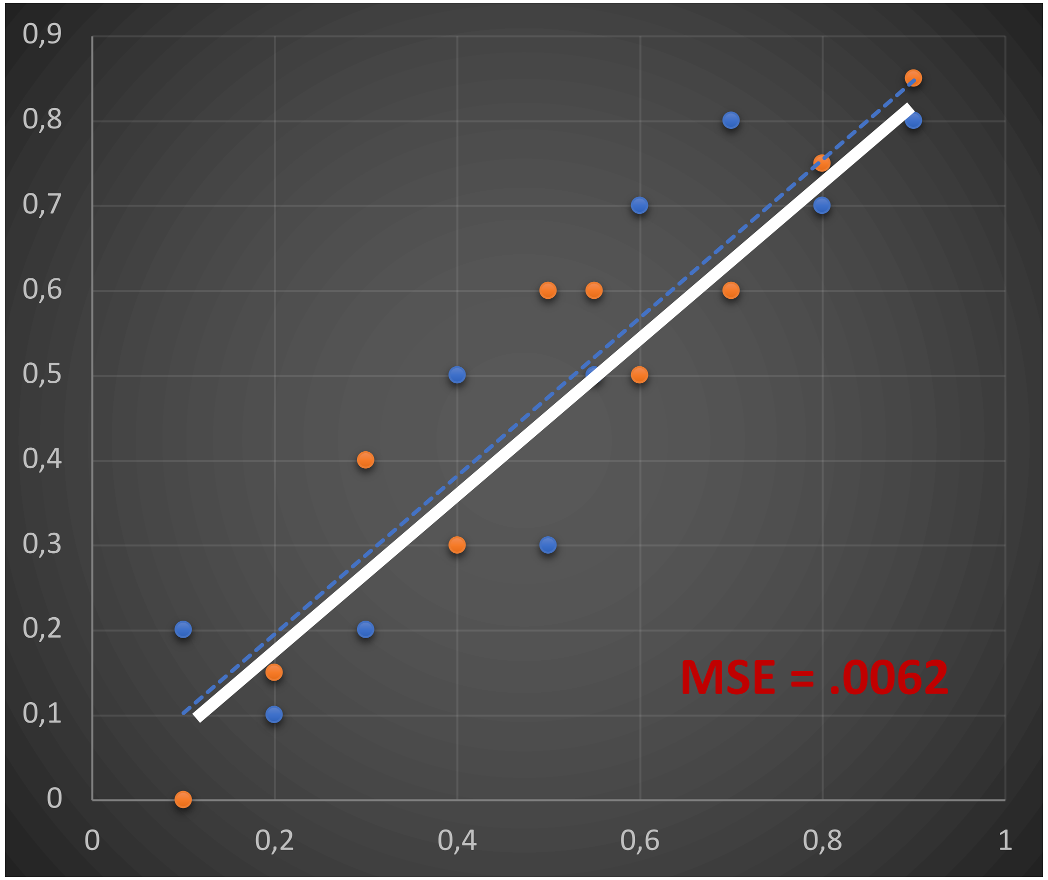
OVERFITTING



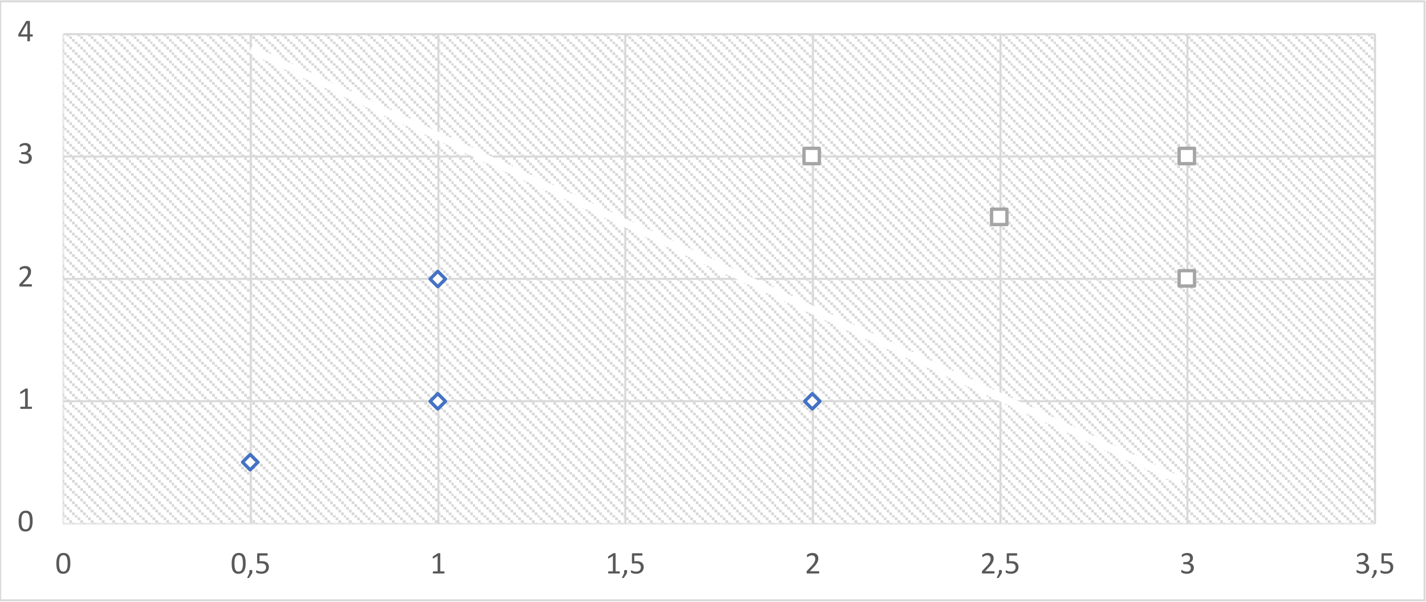


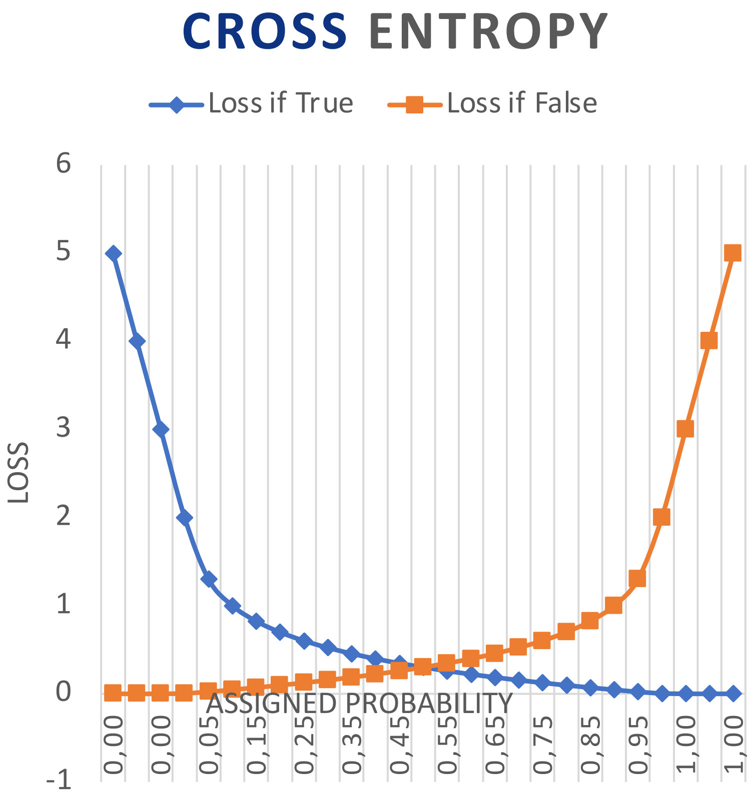
OVERFITTING





CROSS ENTROPY

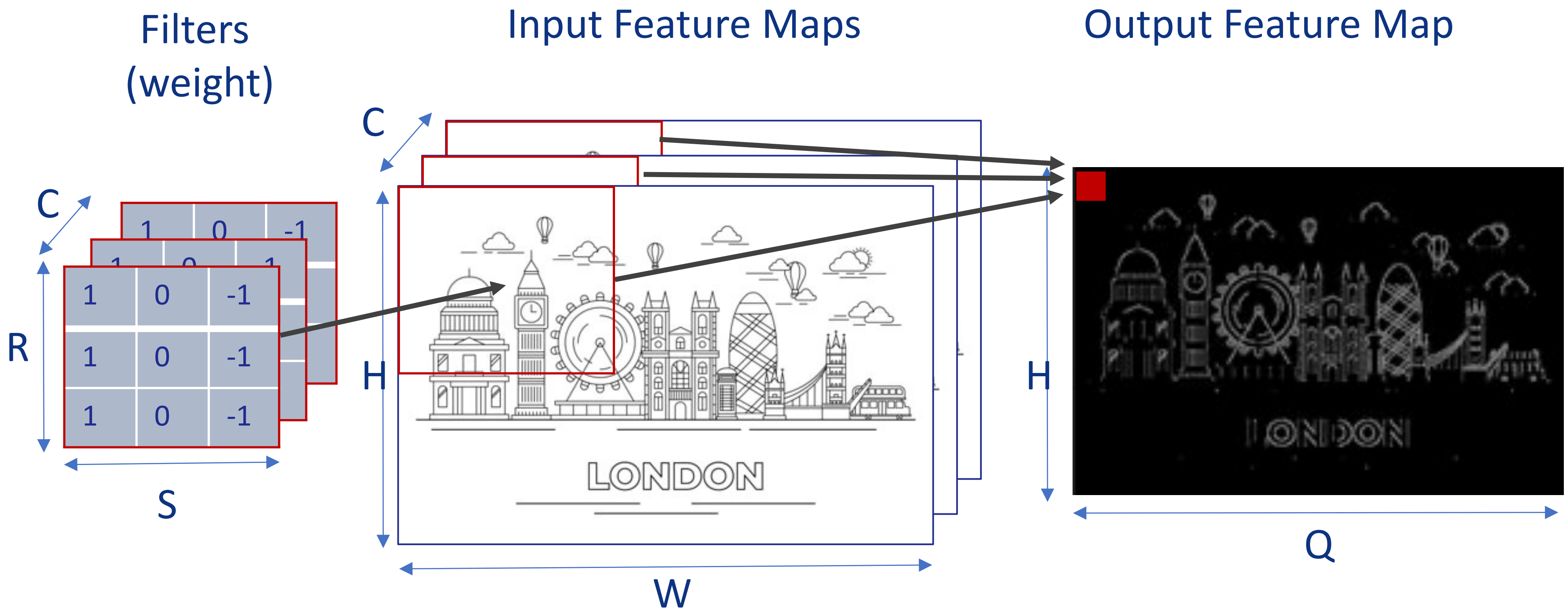




SECOND EXERCISE: RECOGNIZING HANDWRITTEN NUMBERS

KERNELS AND CONVOLUTIONS

KERNELS AND CONVOLUTION



KERNELS AND CONVOLUTION

Filters (weight)

.06	.13	.06
.13	.25	.13
.06	.13	.06

$*$

Input Feature Maps

1	**0**	**1**	**1**	**0**	**1**
0	1	0	0	1	0
0	1	1	1	1	0
0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1

$=$

Output Feature Map

.56	.57	.57	.56
.7	.82	.82	.7
.69	.95	.95	.69
.64	.69	.69	.64

Stride 1

1	0	1	1	0	1
0	1	0	0	1	0
0	1	1	1	1	0

→

.56	.57	.57	.56

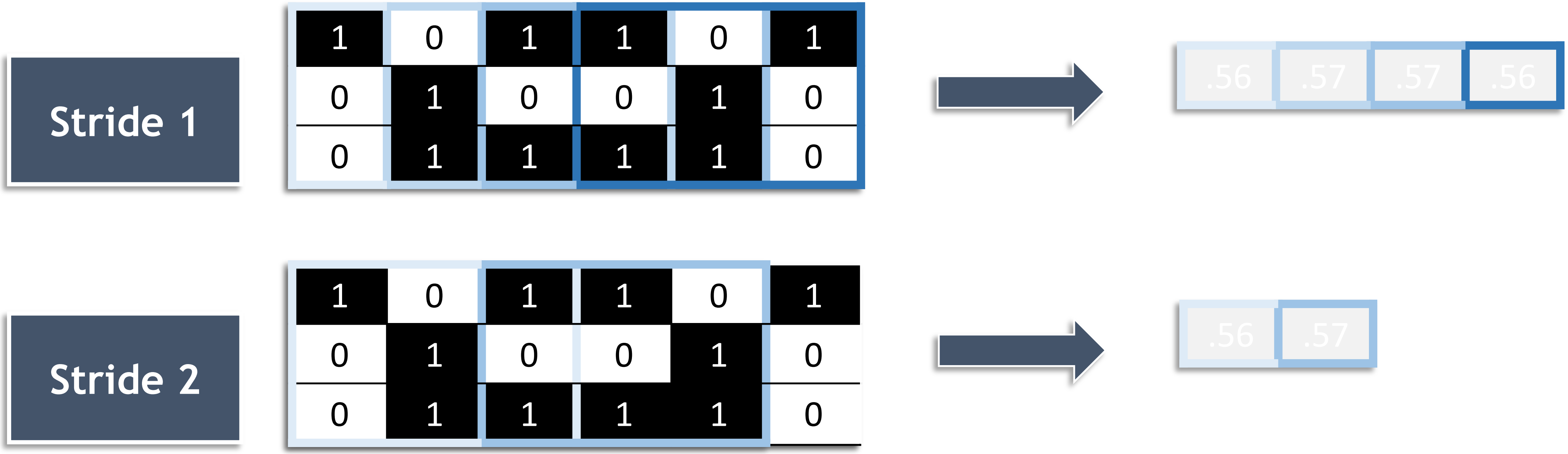
Stride 2

1	0	1	1	0	1
0	1	0	0	1	0
0	1	1	1	1	0

→

.56	.57

STRIDE


Stride 1
1 0 1 1 0 1
0 1 0 0 1 0
0 1 1 1 1 0
.56 .57 .57 .56
Stride 2
1 0 1 1 0 1
0 1 0 0 1 0
0 1 1 1 1 0
.56 .57

PADDING

Original Image

1	0	1	1	0	1
0	1	0	0	1	0
0	1	1	1	1	0
0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1

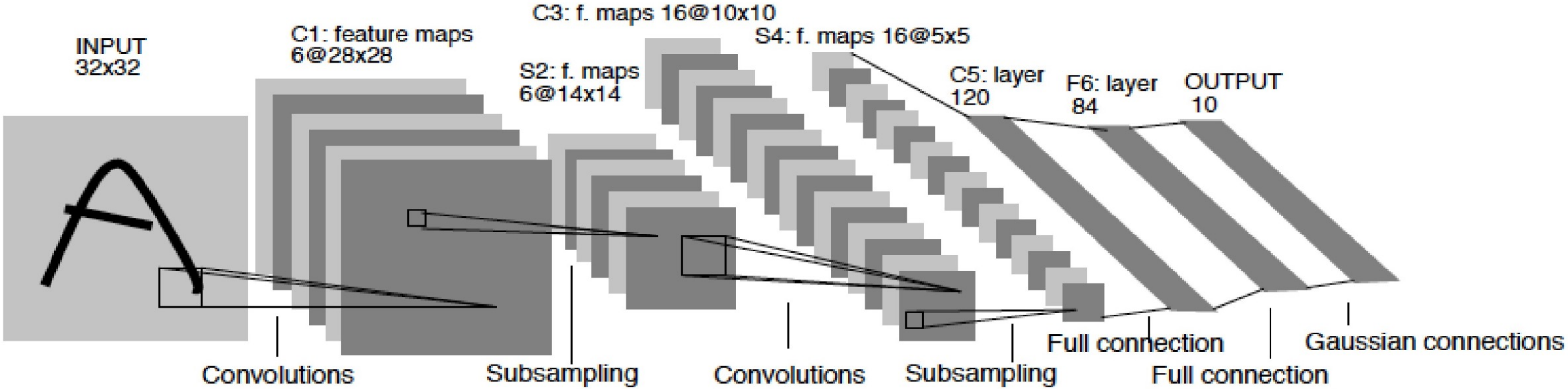
Zero Padding

0	0	0	0	0	0	0	0
0	1	0	1	1	0	1	0
0	0	1	0	0	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0

Mirror Padding

1	1	0	1	1	0	1	1
1	1	0	1	1	0	1	1
0	0	1	0	0	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
1	1	0	1	1	0	1	1
1	1	1	0	0	1	1	1
1	1	1	0	0	1	1	1

TIPYCAL CNN ARCHITECTURE

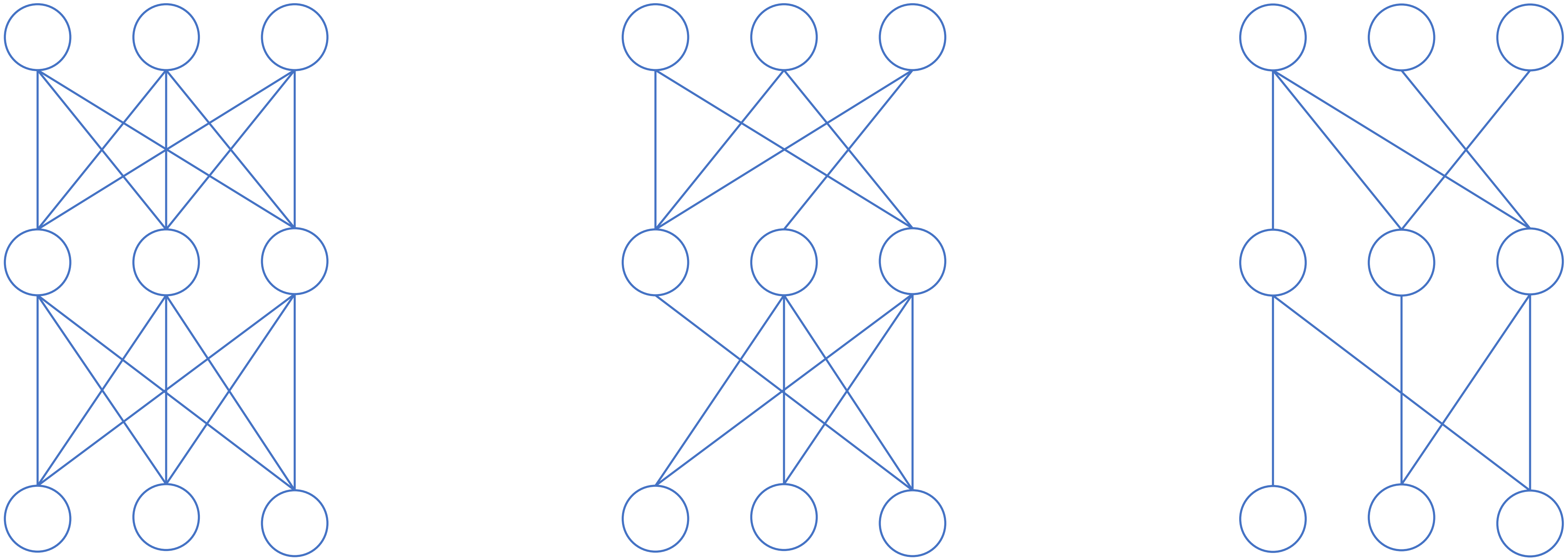


WHAT ARE THESE SUB-SAMPLINGS?

110	256	153	67
12	89	88	43
10	15	50	55
23	9	49	23

256	153
23	55

DROPOUT

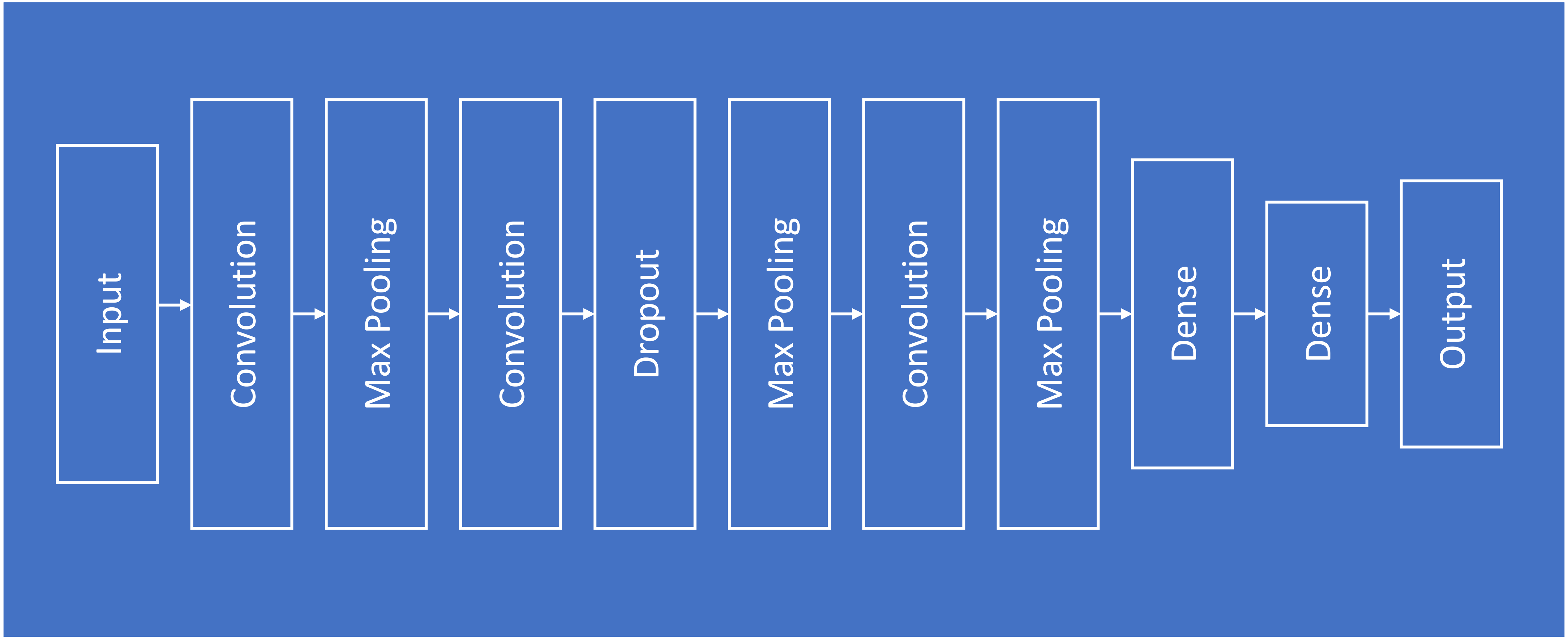


rate = 0

rate = .2

rate = .4

WHOLE ARCHITECTURE

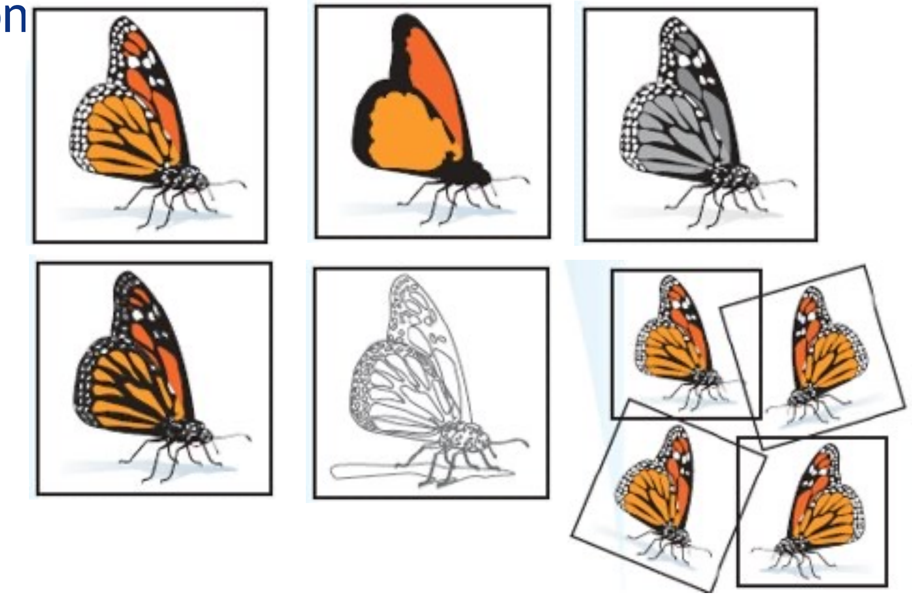


EXERCISE 3:CNN FOR A MORE COMPLEX PROBLEM

MODEL AND DATA

- CNN increased validation accuracy
- Still seeing training accuracy higher than validation
- Clean data provides better examples
- Dataset variety helps the model generalize
- A technique helping in this case is data augmentation
  - Color change
  - Image flipping
  - Rotation
  - Zooming
  - Brightness

Domain dependent



MODEL DEPLOYMENT AND TRANSFER LEARNING

PRE-TRAINED MODELS

TensorFlow Hub

Keras

NVIDIA NGC

PYTORCH
HUB

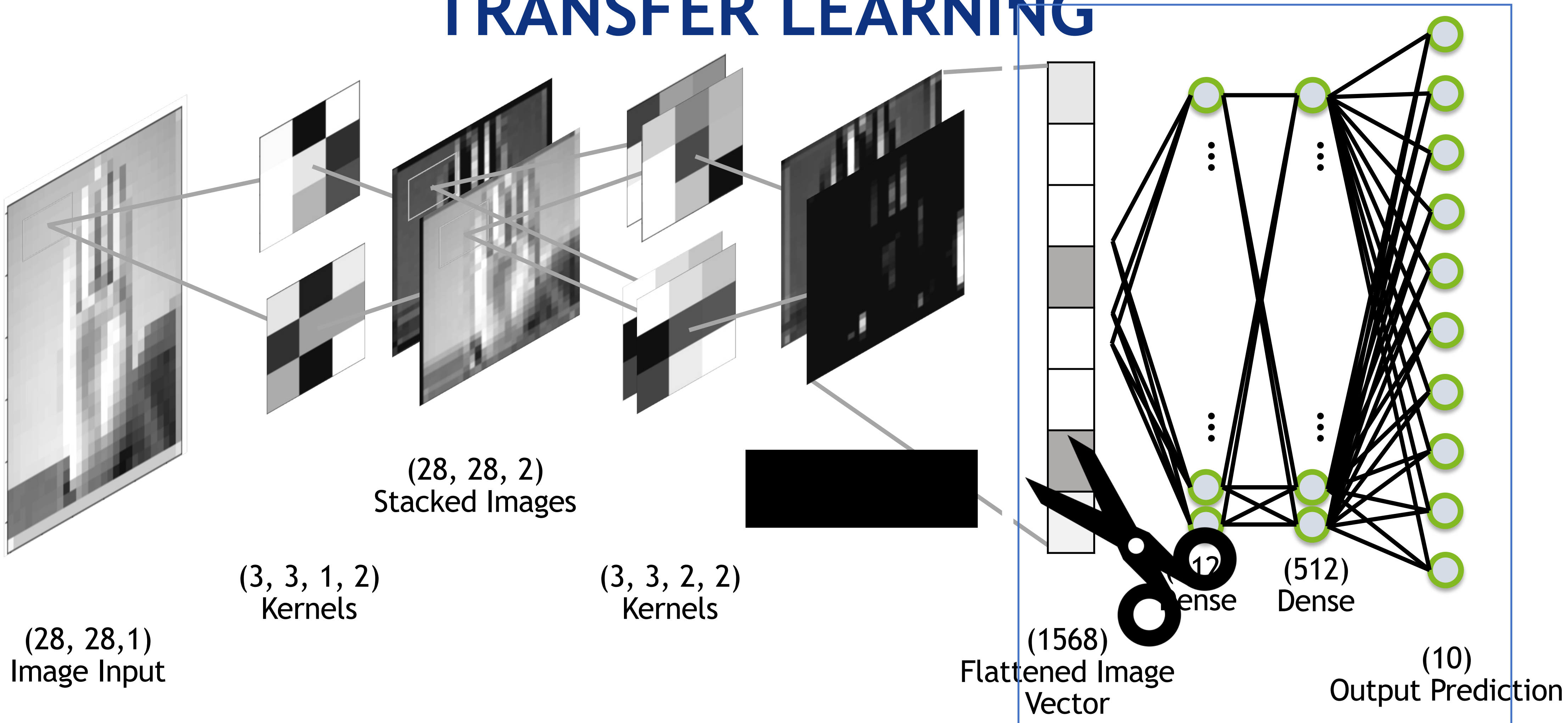
PRE-TRAINED MODELS

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan[*] **& Andrew Zisserman**[+]
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

NET

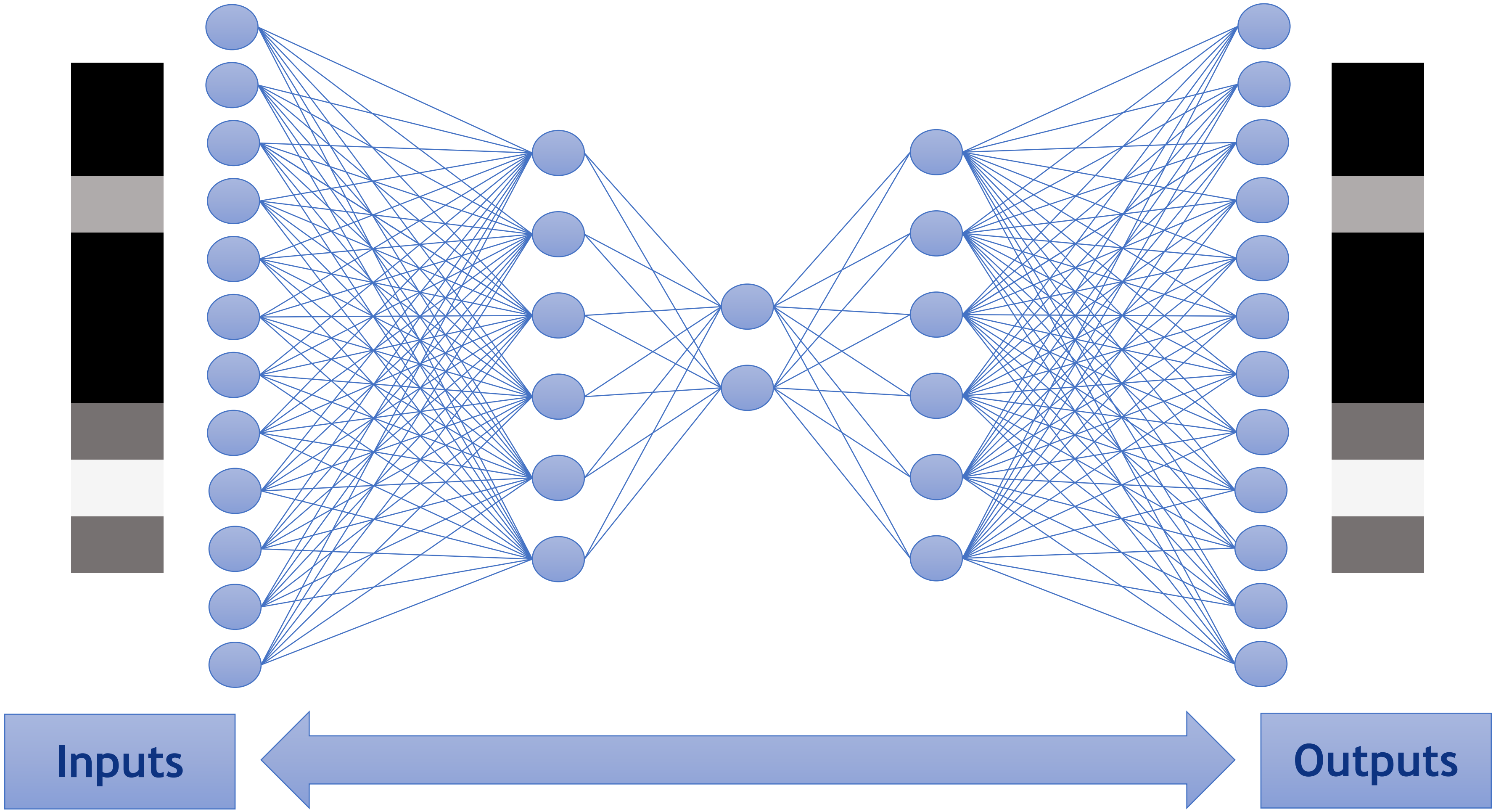
TRANSFER LEARNING



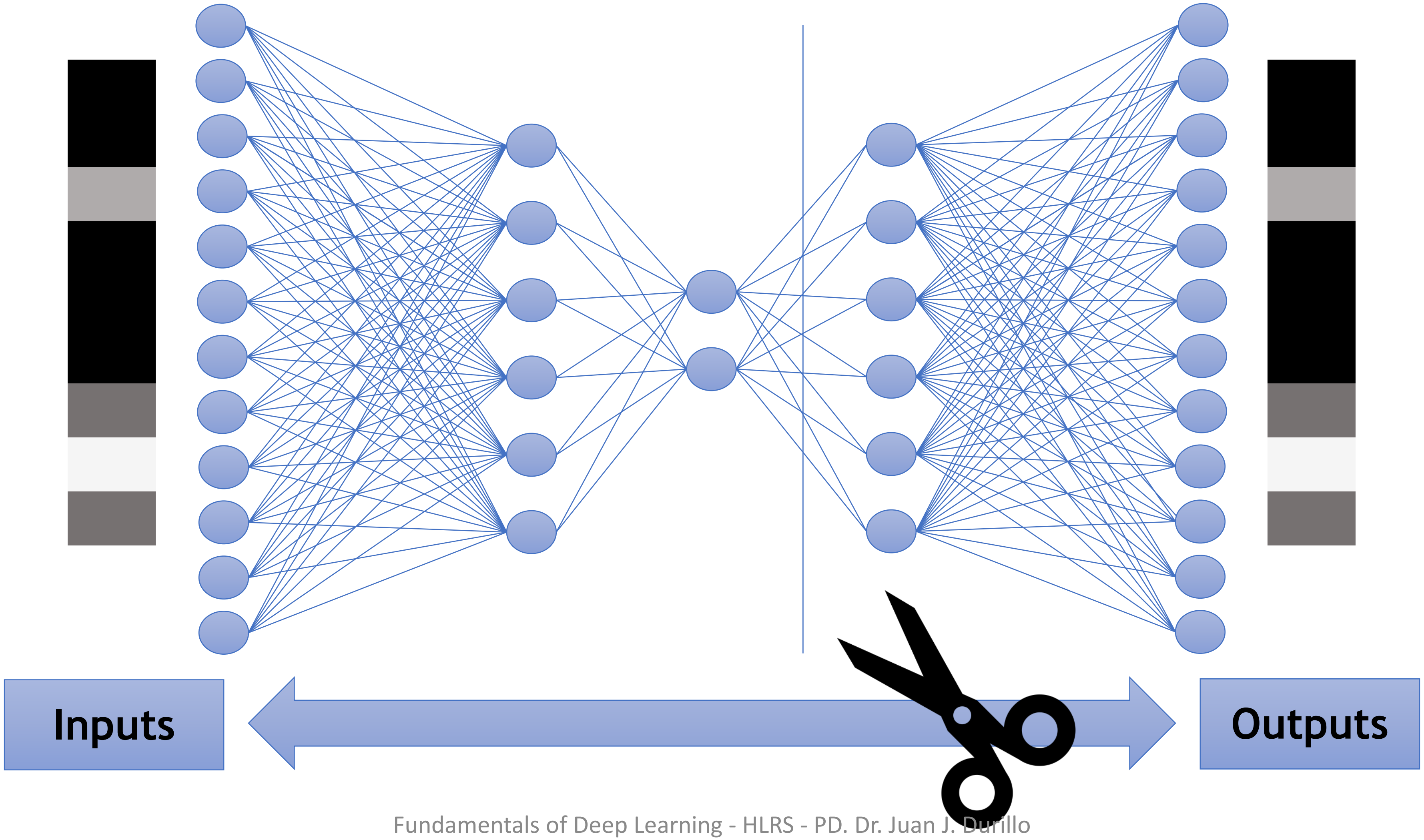
EXERCISE 4: DEPLOYMENT AND ADVANCED ARCHITECTURES

AUTOENCODERS

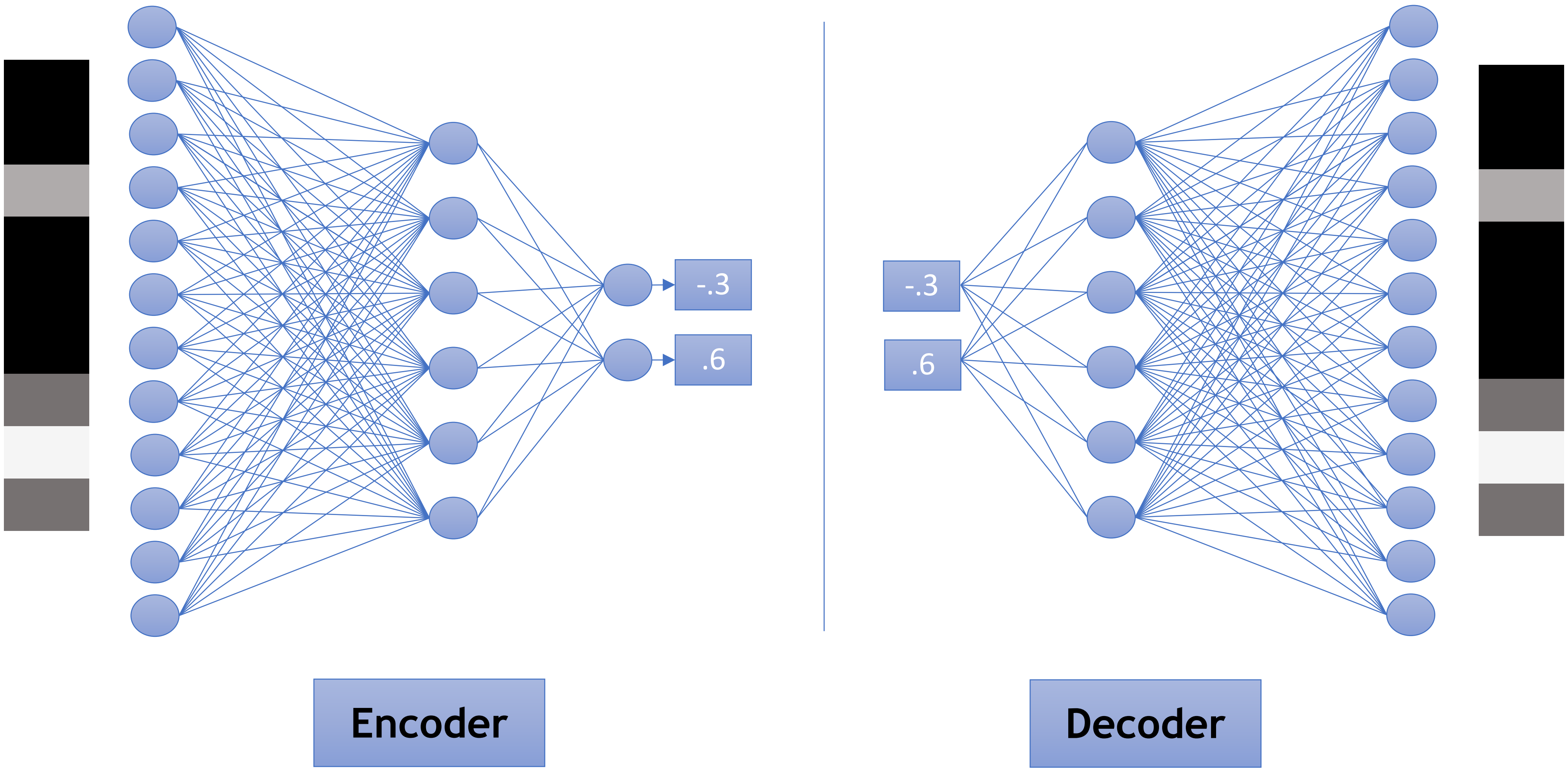
AUTOENCODERS



AUTOENCODERS



AUTOENCODERS



EXERCISE 5:GENERATING HANDWRITTEN NUMBERS

SOME WORDS ON NLP

EVOLUTION OF NLP TOWARD TRANSFORMERS

- Last 20 years a profound change in NLP
  - Experienced different paradigms and finally entered an era mostly dominated by Transformer architectures
  - Transformers started with the help of various neural–based encoder-decoder like approaches and gradually evolved towards attention-based encoder-decoder type architectures

LEVERAGING EMBEDDINGS

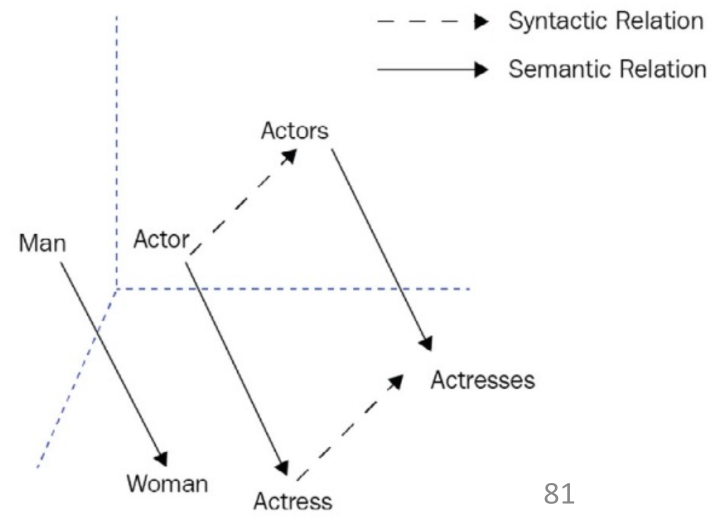
Argued that count-based models can be better than neural models

Leverages both global and local statistics of a corpus

Performs well in syntactic and semantic tasks

Embedding offsets between terms help to apply vector-oriented reasoning

- Learn embeddings based on word-word co-occurrence statistics
- Product of two words embedding should be proportional to their co-occurrence frequency

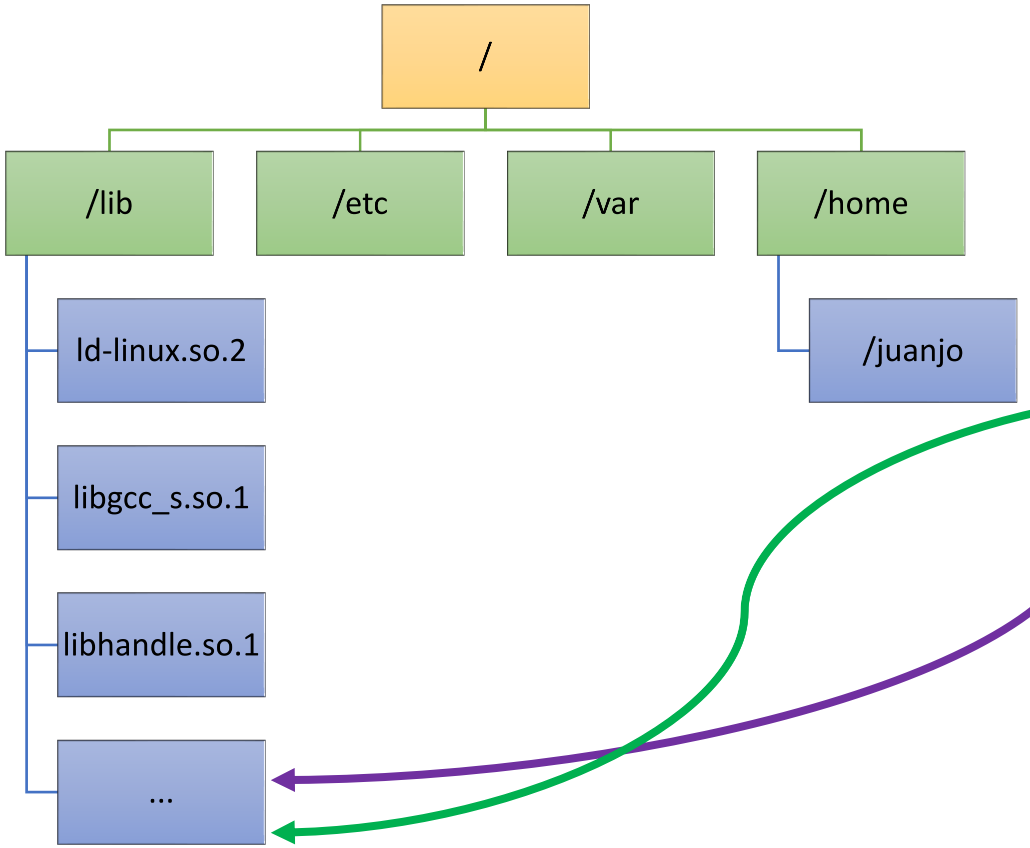


CLOSING THOUGHTS AND QUESTIONS

APPENDIX: UNDERSTANDING CONTAINERS

UNDERSTANDING CONTAINERS

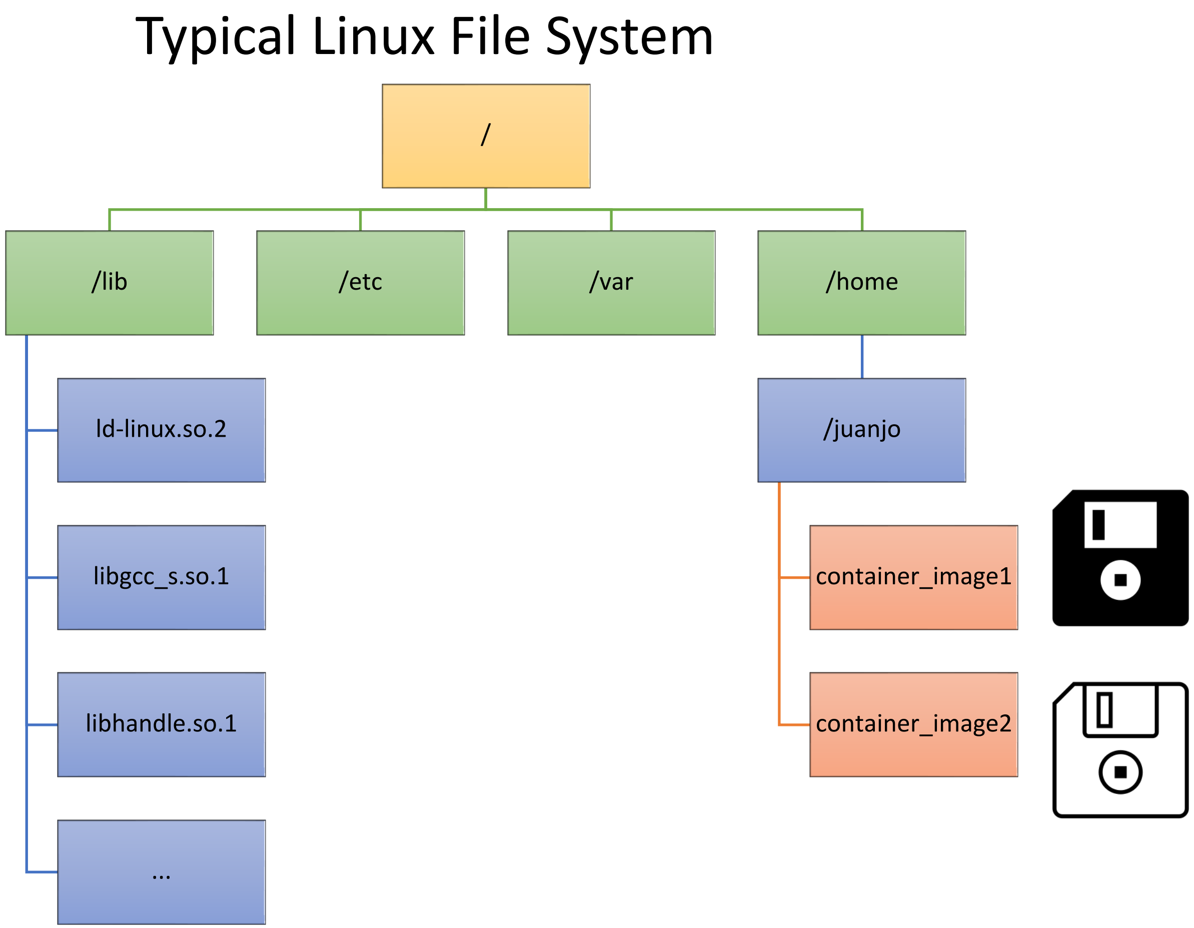
Typical Linux File System



Program In Execution in Linux

```
$ ldd /bin/program
linux-vdso.so.1 (0x00007fff204e8000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f411cc6a000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f411ca78000)
libpcre2-8.so.0 => /usr/lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007f411c9e8000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f411c9e2000)
/lib64/ld-linux-x86-64.so.2 (0x00007f411ccd0000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f411c9bf000)
```

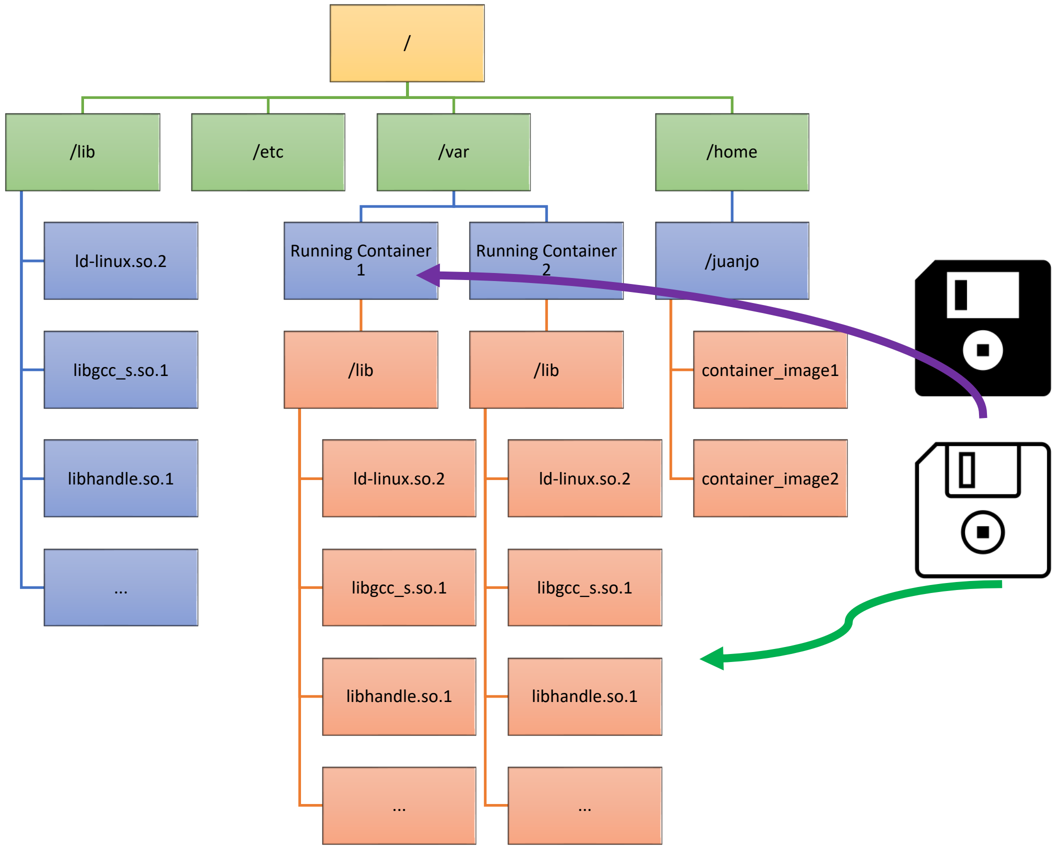
UNDERSTANDING CONTAINERS: CONTAINER IMAGES



- Typically, a single compressed file
  - It contains a complete Linux File System + Metadata
- Different container technologies might:
  - use different formats
    - e.g., OCI format is **a specification for container images based on the Docker Image Manifest Version 2, Schema 2 format**
  - hide images to users
- Are meant to be static
- Not to be confused with a docker file

UNDERSTANDING CONTAINERS: CONTAINER

Typical Linux File System

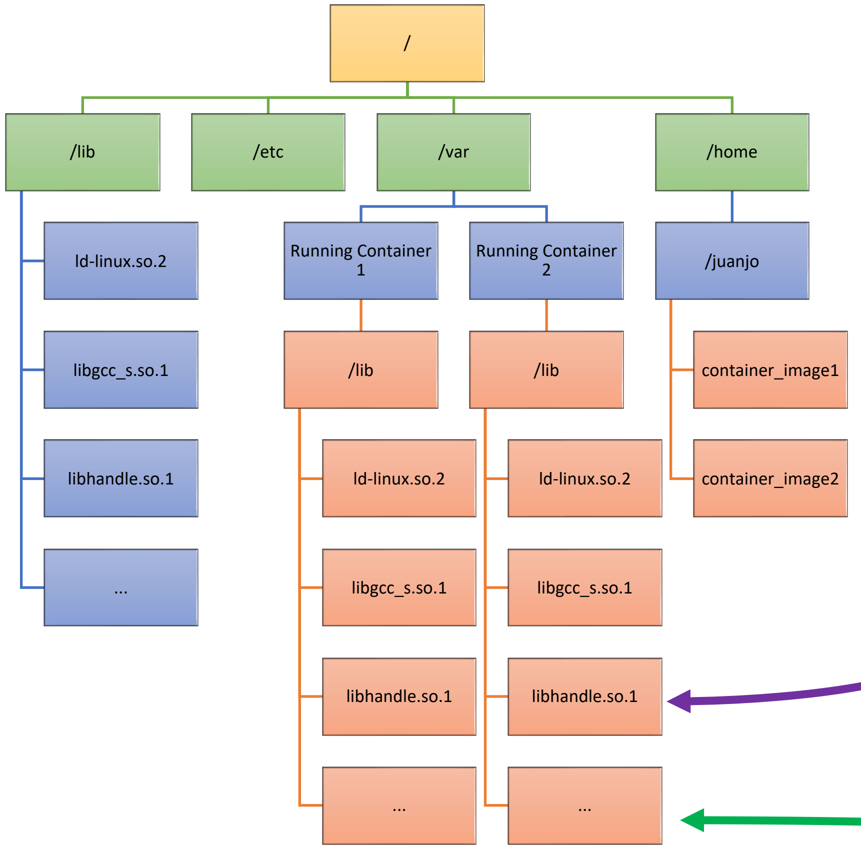


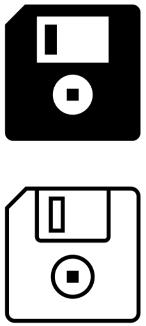
- A running instance of a container image
  - A complete Linux File System within a Linux File System
    - Libraries might be different (versions)
    - Provided programs might be different
- Specific program in charge of unpacking the image and storing it within the proper folder
  - Docker, Udocker, Podman, Enroot, etc.
- More than one container can
  - Exist at any point in time
  - Be generated from a single image

UNDERSTANDING CONTAINERS: CONTAINER

- It is possible to "run a process within a container"
  - Confine the process to the content of the container File System
  - Specific program in charge for confining and running the process within the container
    - docker run/start, enroot start

Typical Linux File System





```
$ ldd /bin/program
linux-vdso.so.1 (0x00007fff204e8000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1
(0x00007f411cc6a000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
(0x00007f411ca78000)
libpcre2-8.so.0 => /usr/lib/x86_64-linux-gnu/libpcre2-8.so.0
(0x00007f411c9e8000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2
(0x00007f411c9e2000)
/lib64/ld-linux-x86-64.so.2 (0x00007f411ccd0000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
(0x00007f411c9bf000)
```

Process run within Running Container 2