

OPEN DATA SCIENCE CONFERENCE

London | September 19 - 22 2018



@ODSC

#ODSC 2018 - London

20.09.2018

The Path to Deep Learning with Tensorflow + Keras

Juliet Moreiro

Technical Evangelist (Microsoft)

Pablo Doval

Data Pontifex (Plain Concepts)



Juliet Moreiro

TECHNICAL EVANGELIST @ MICROSOFT

Cloud evangelist, you'll never guess my cat's name.

@julietsvq



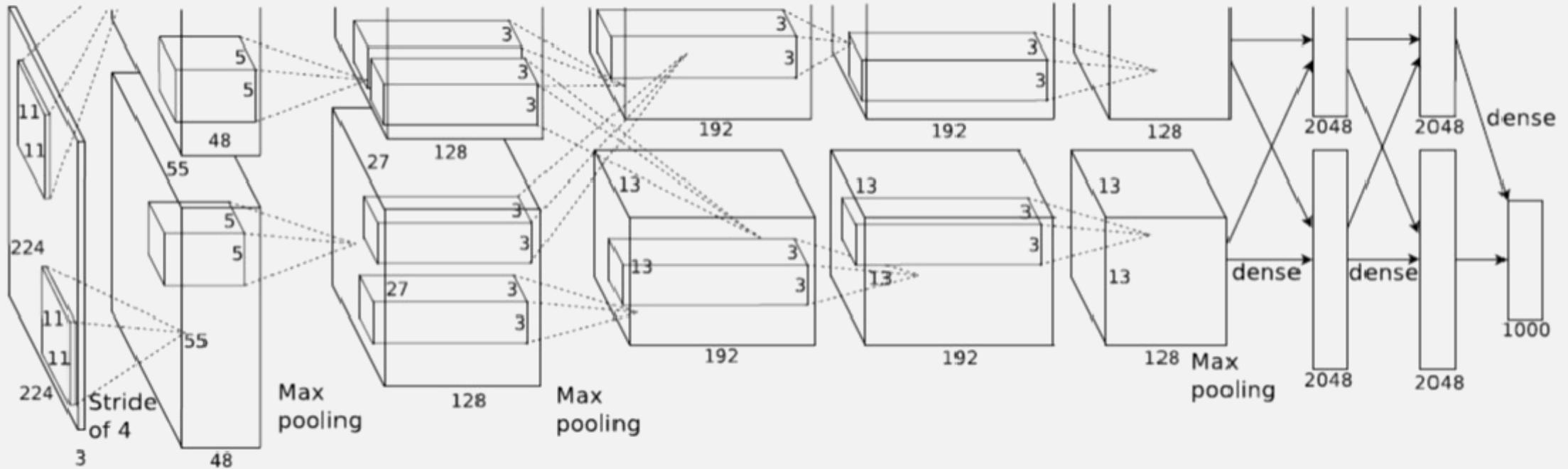
Pablo Doval

DATA PONTIFEX

I work with code and data, but don't tell my mom; she thinks I'm a piano player in a whorehouse.

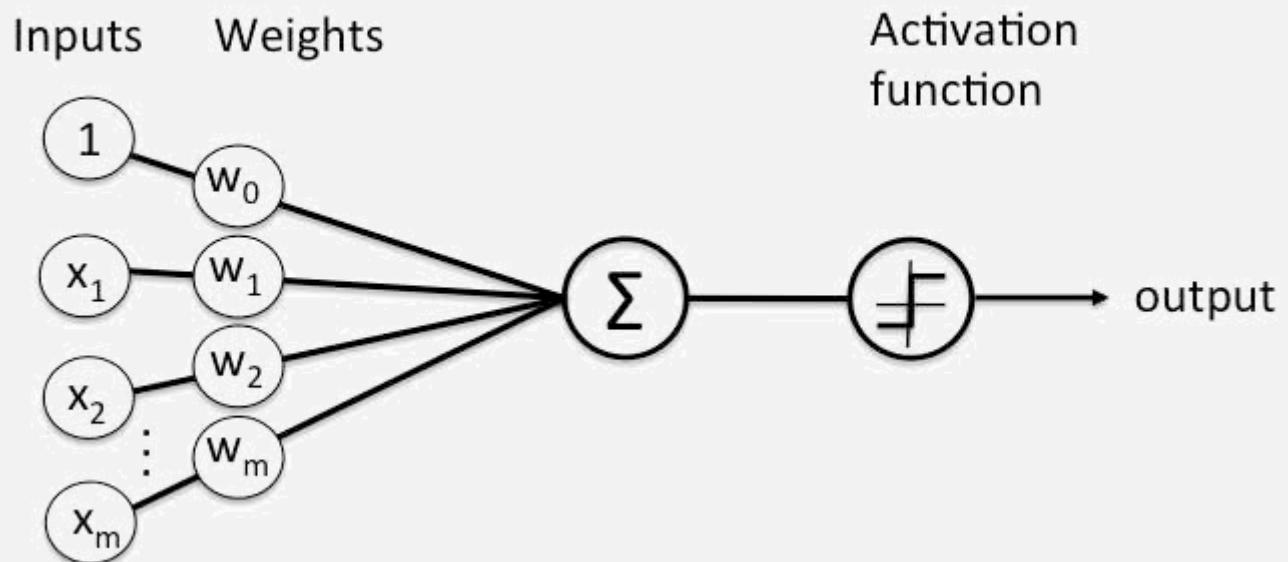
@PabloDoval

WHAT IS THIS THING?

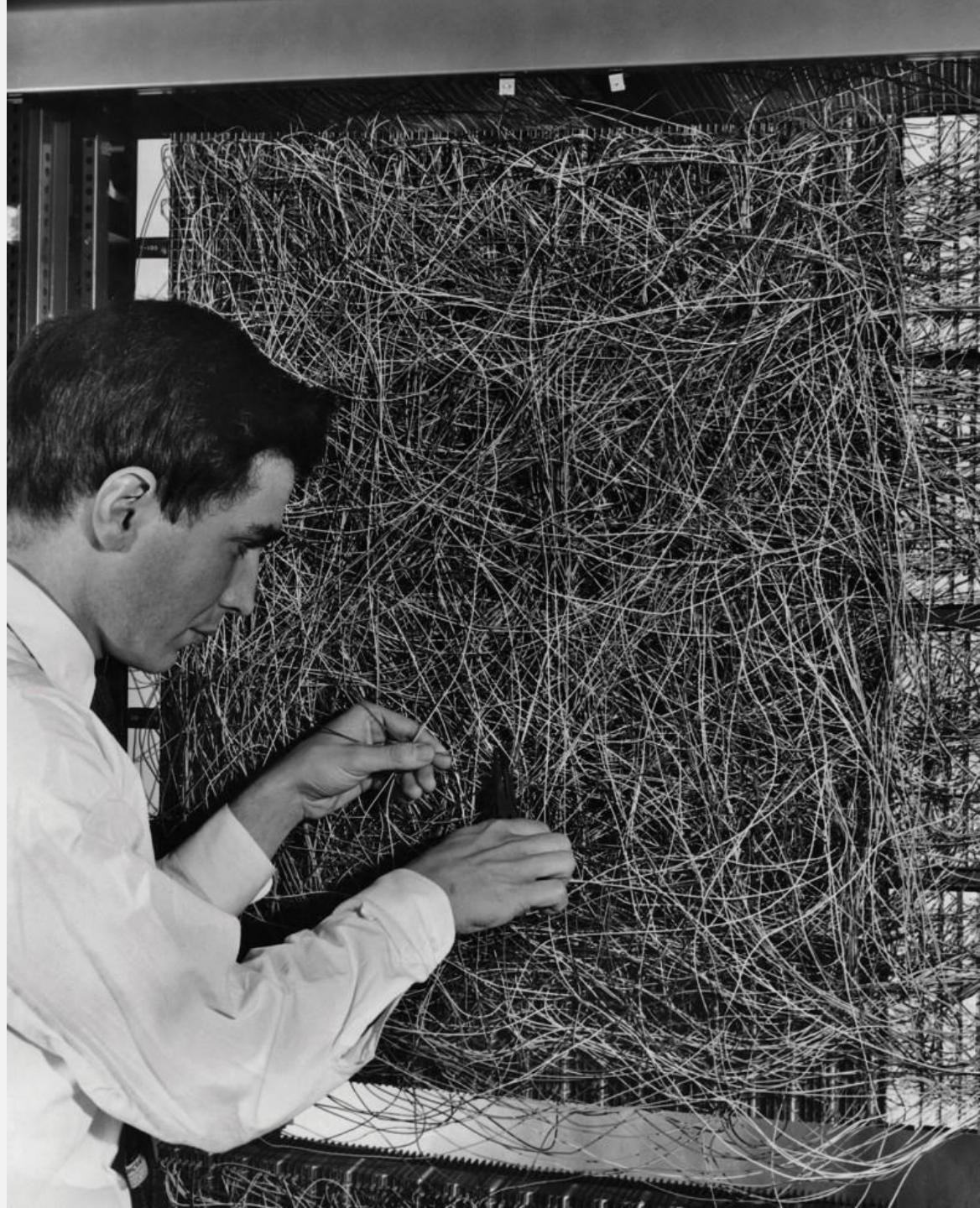
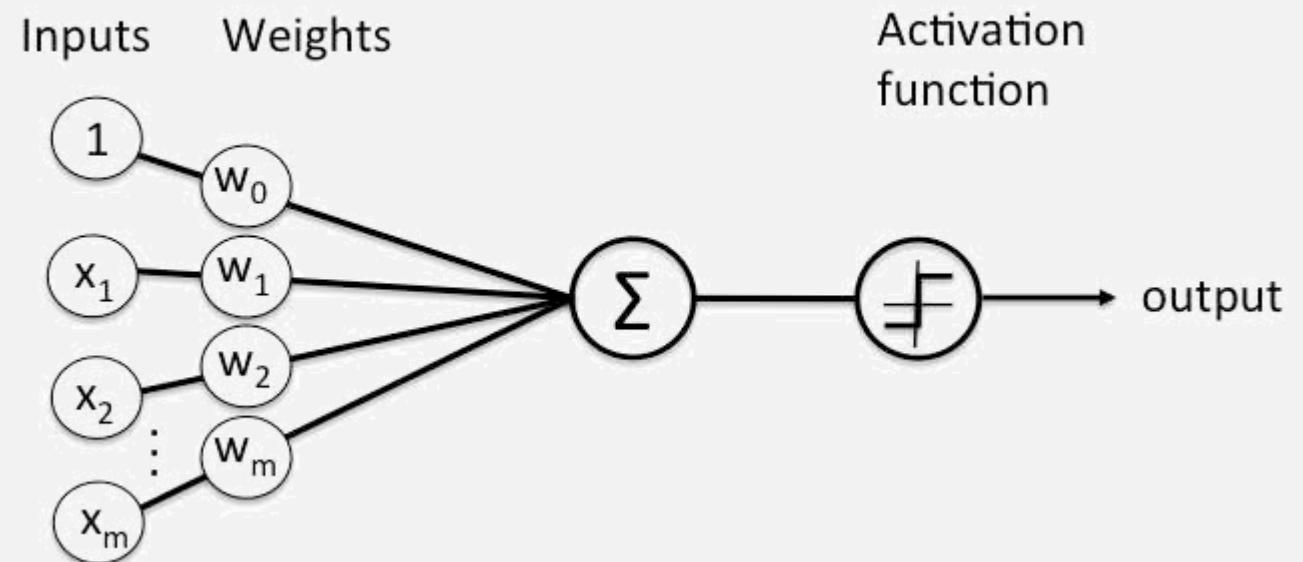




PERCEPTRON



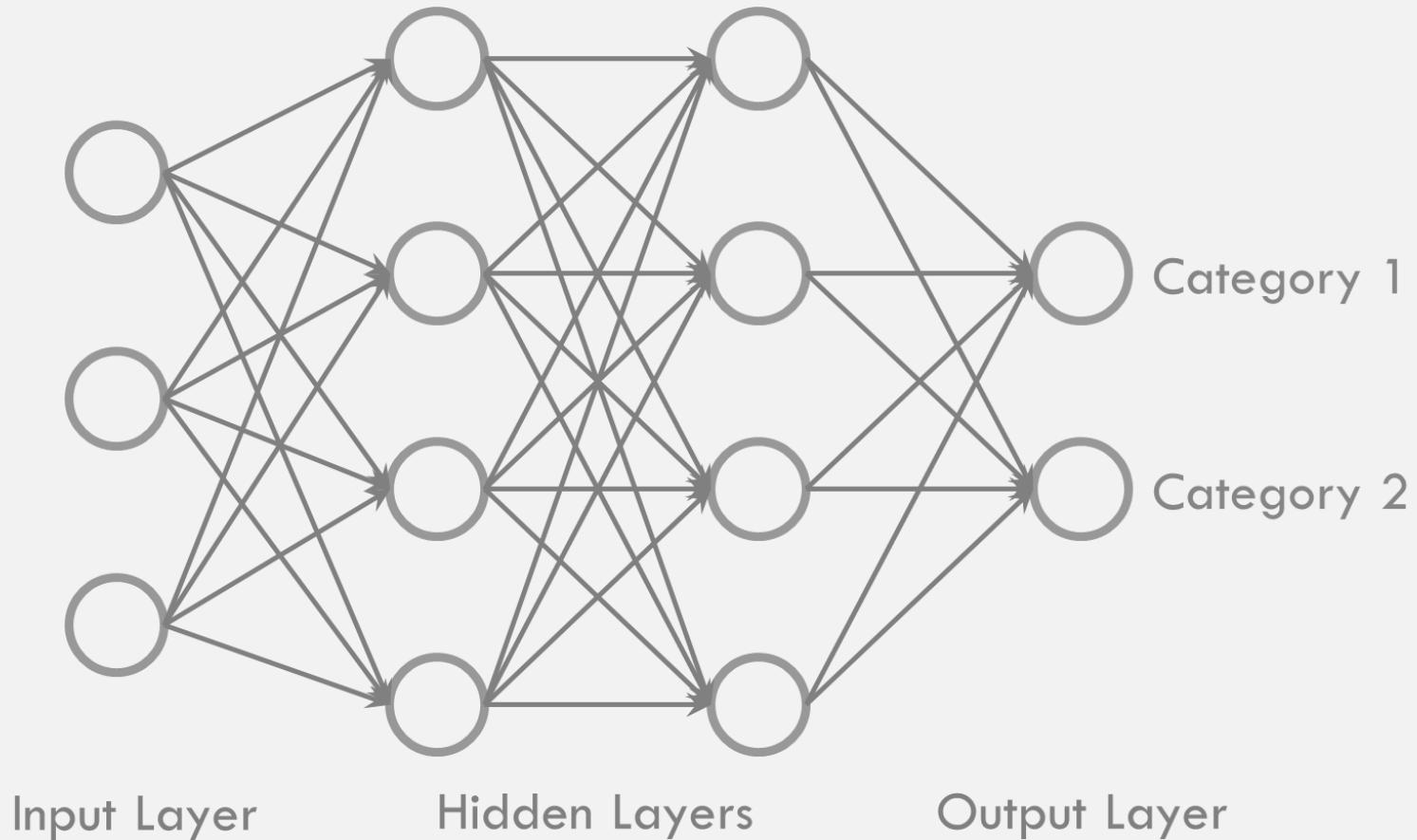
PERCEPTRON



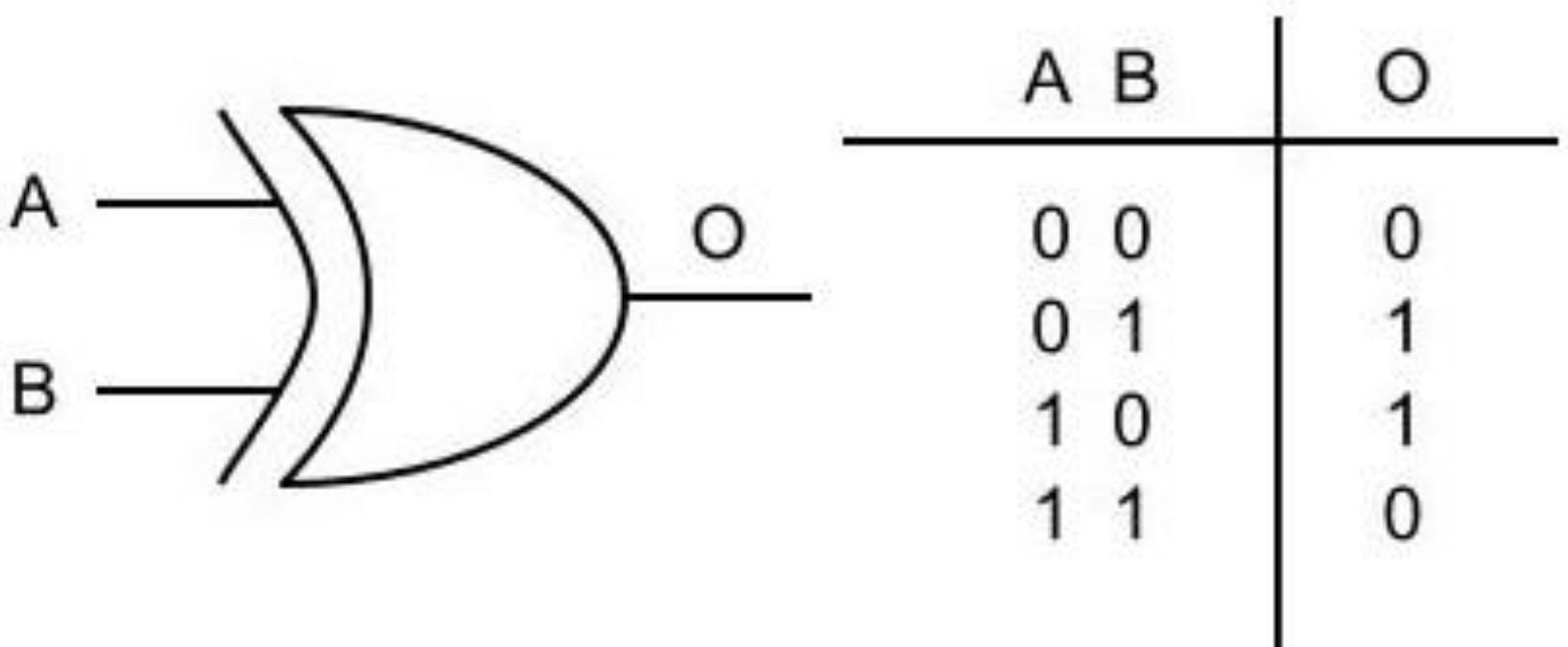
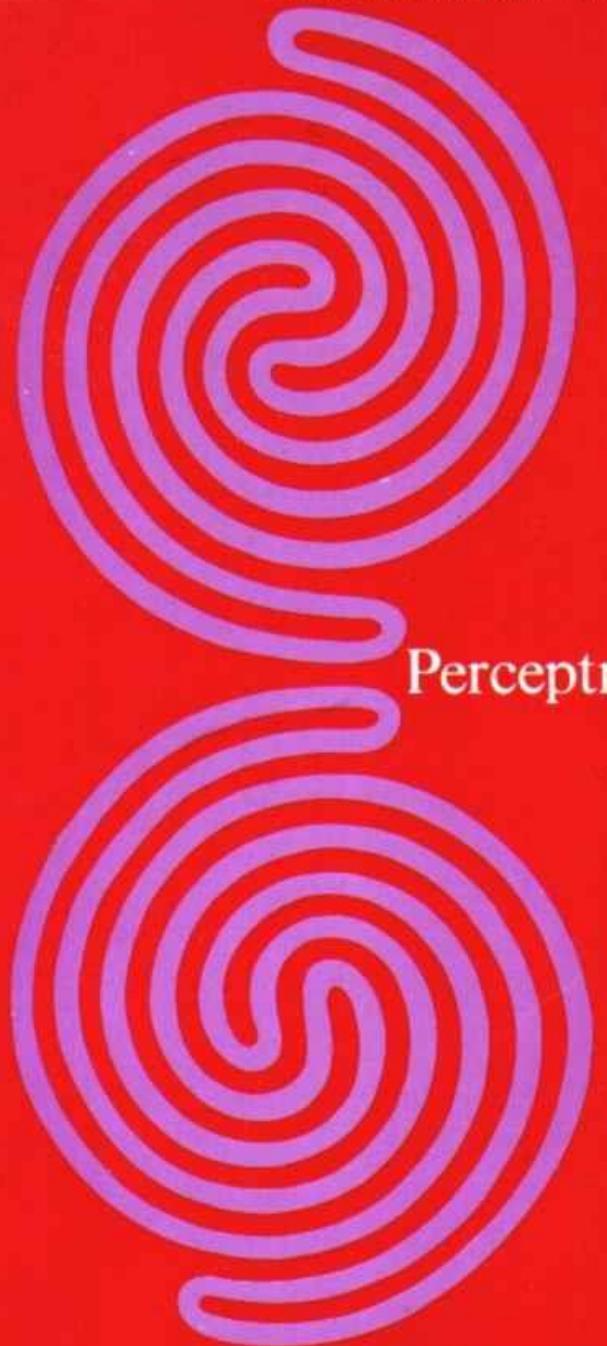
ARTIFICIAL INTELLIGENCE



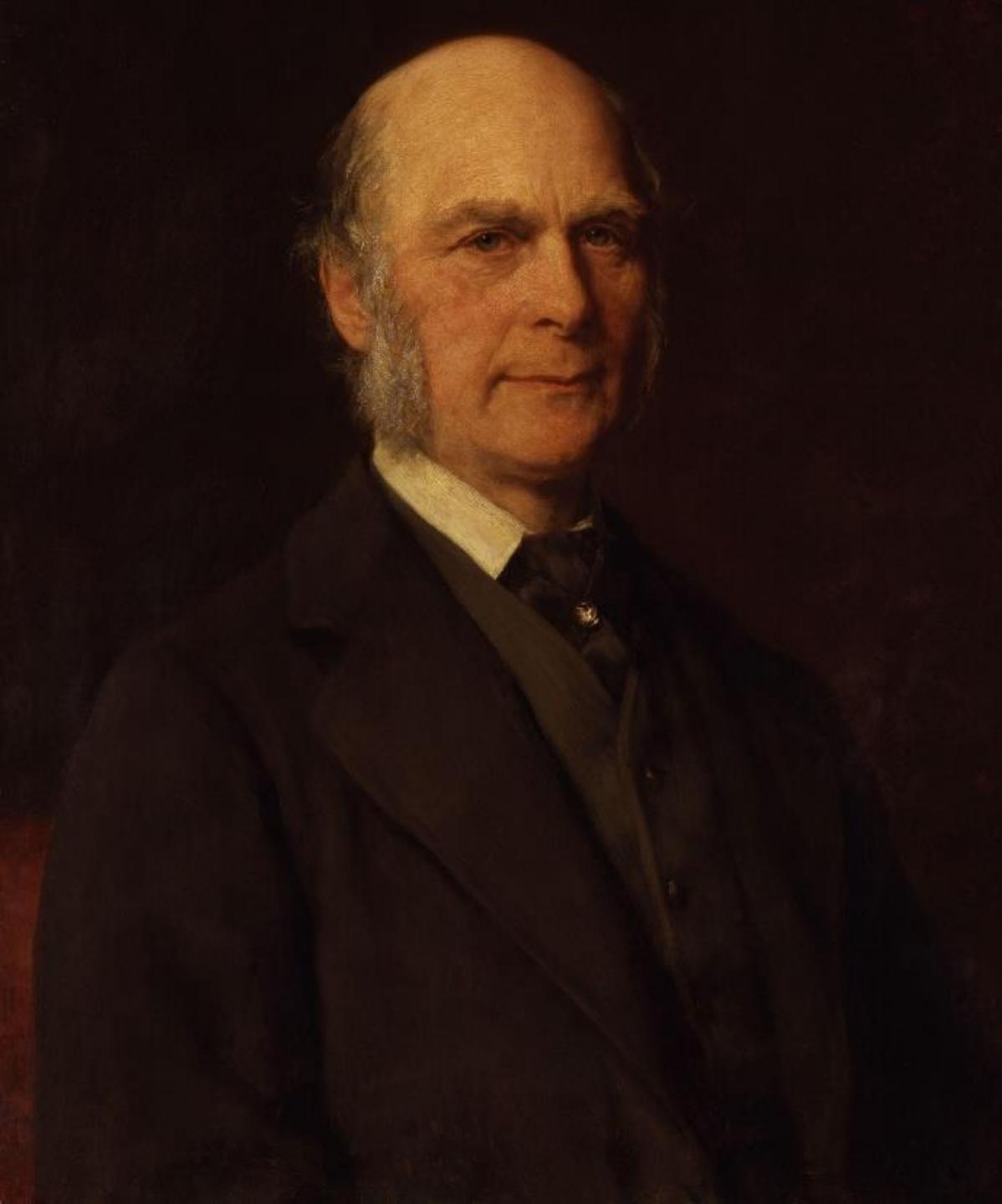
MULTILAYER PERCEPTRON



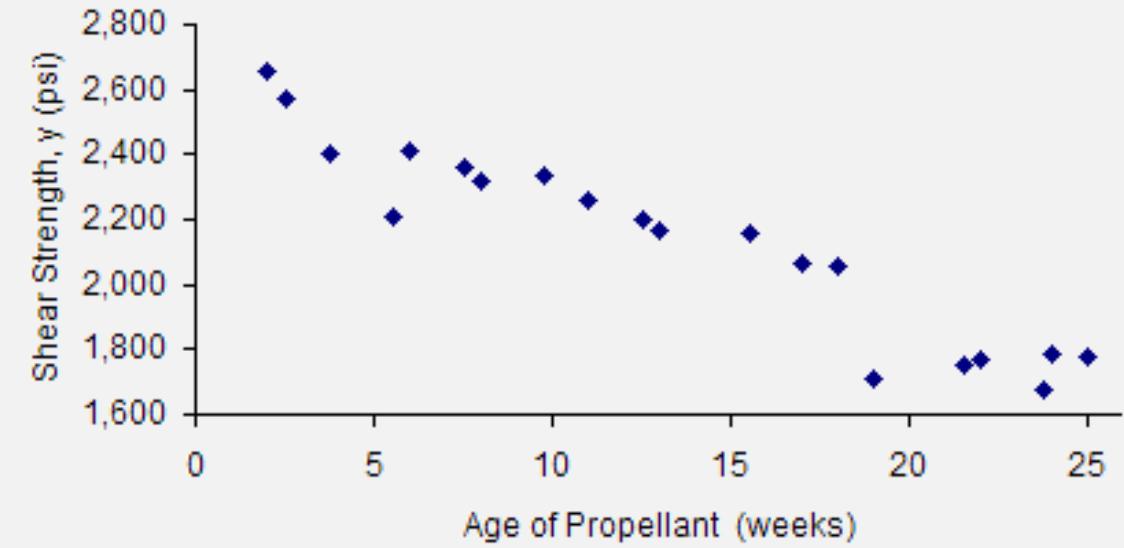
WINTER
IS COMING



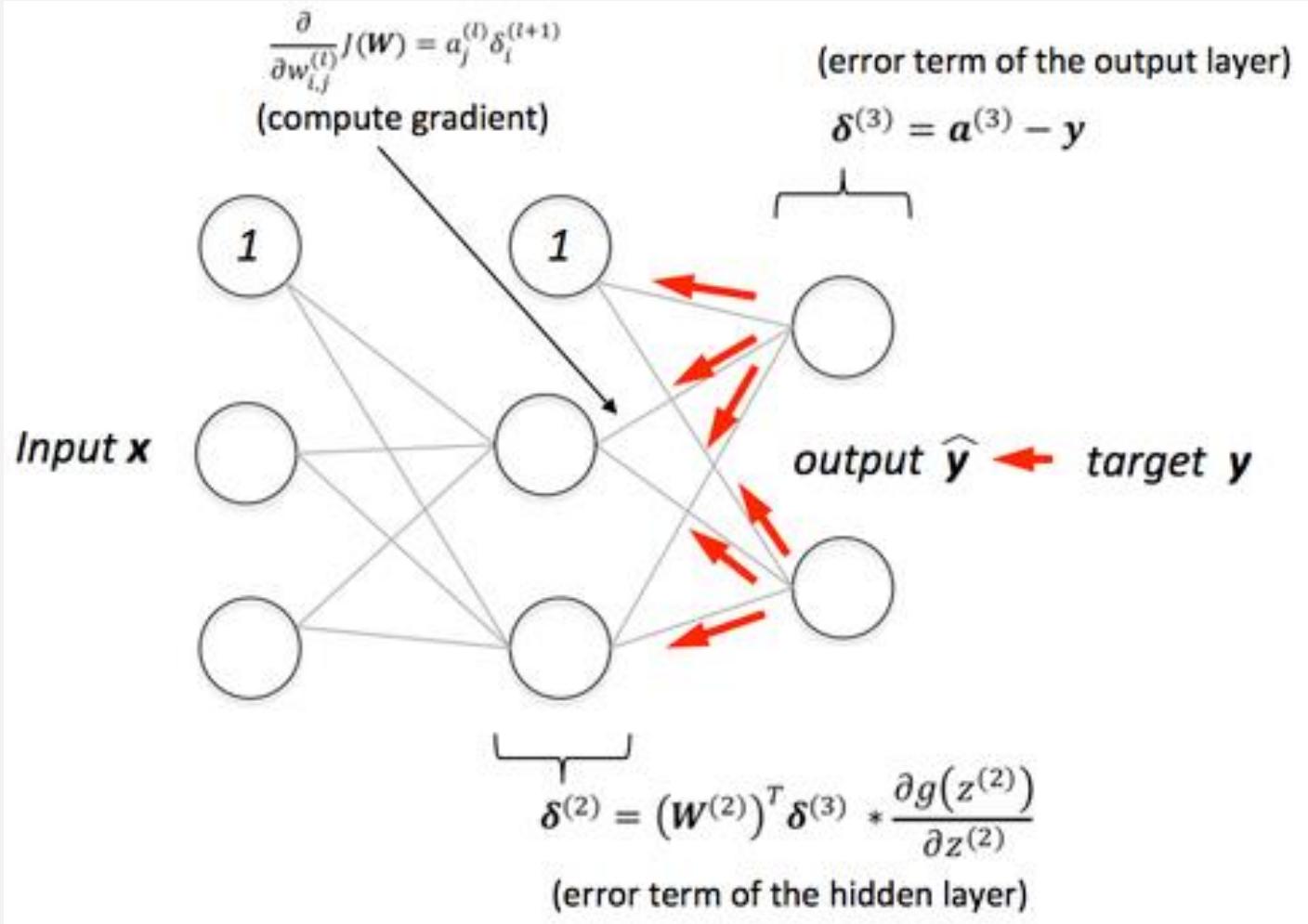
EXPERT SYSTEMS?



MACHINE LEARNING

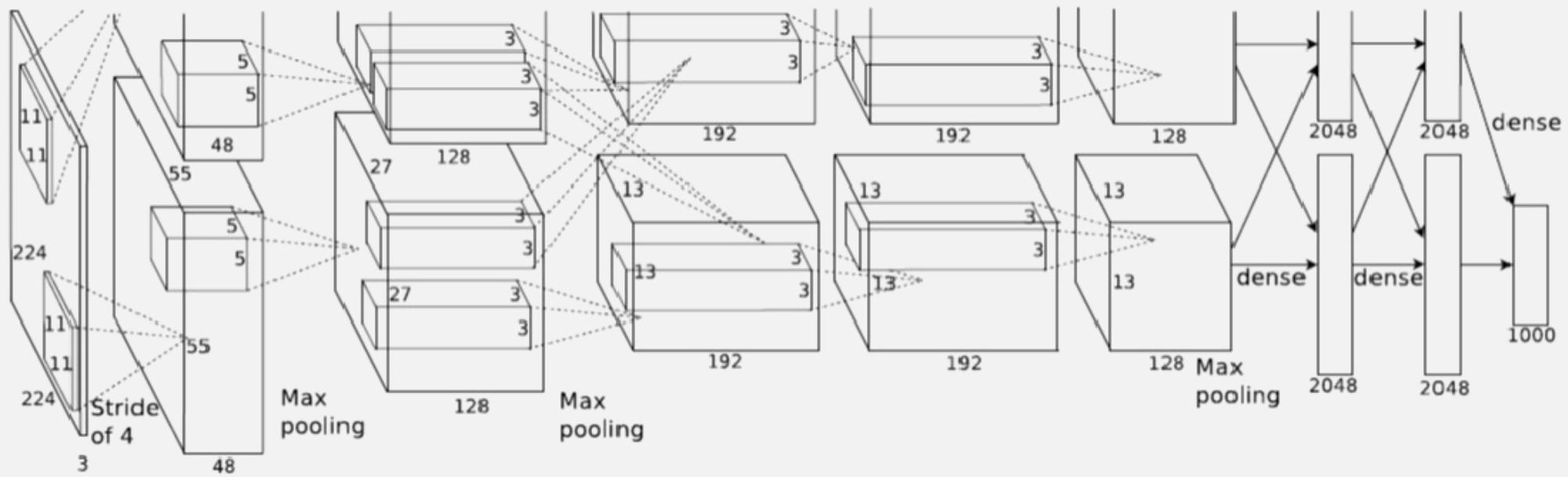


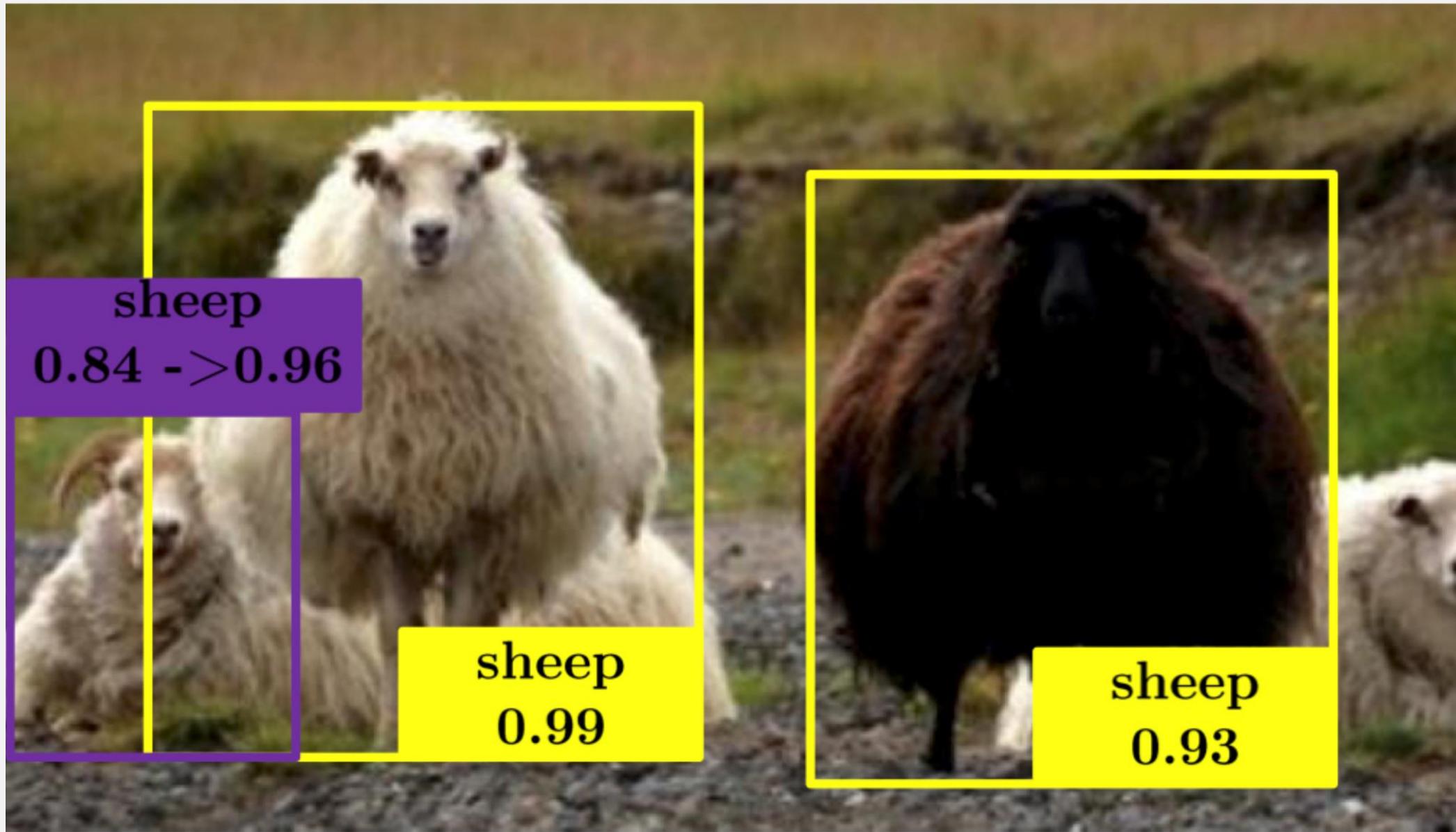
BACKPROPAGATION



DEEP LEARNING

A Machine Learning technique











LAB 0: Setup

CHECKLIST

- CUDA 9
 - >= 5.2
 - Python 3.6
 - VS Code (or Atom, ...)
 - Jupyter
- <https://github.com/PabloDoval/odsc18-london>
-
- The diagram illustrates the checklist items grouped into four categories: CUDA, Anaconda, IDE, and Repository. Each category is preceded by a black text label and followed by a blue curly brace that spans the list of items under it. A horizontal blue line is positioned above the IDE and Repository sections.
- CUDA
 - Anaconda
 - IDE
 - Repository

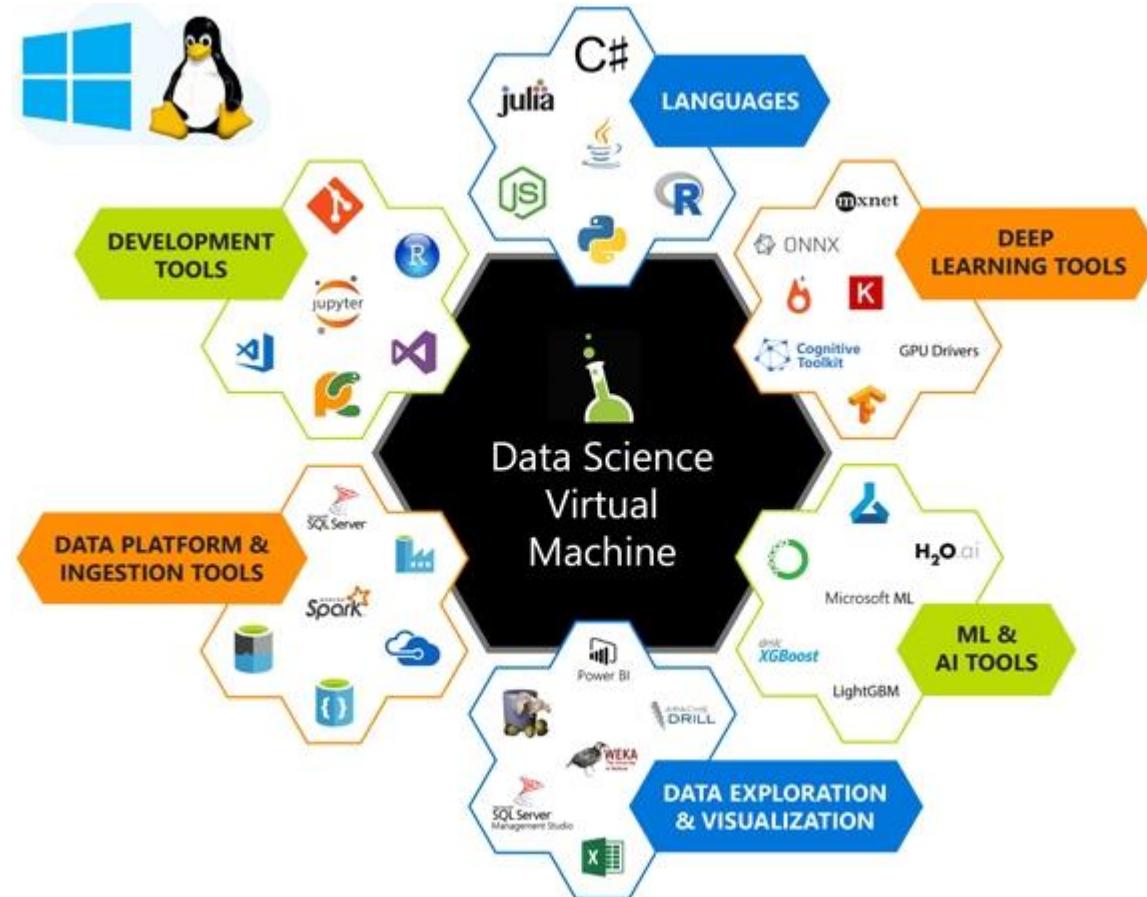
ODSC 2018 Workshop - Deep Learning

Initial Setup

1. First of all, please do install Anaconda 5.2 in your machine
 - Windows version [here](#).
 - Linux version [here](#).
 - MacOS X version [here](#).
2. Go to the `.\Scripts` folder and run the script that will perform the environment setup:
 - `environment.bat` (if you are running Windows)
 - `./environment.sh` (for you, Linux folks)
3. Once the script finishes running, that environment will be active. The name of the environment is `odsc18-deeplearningworkshop`, in case you want to activate it manually.
4. If you plan on using GPU - and you should! - install the CUDA toolkit from [this link](#) to the NVIDIA web page.
5. Run the `main.py` to check that the environment and dependencies have been properly setup. The output will show the version of the Tensorflow and Keras distributions, as well as the number of CPUs and GPUs available in our machine.

Azure DataScience Virtual Machines

Pre-installed installed images, configured and tested with popular tools that are commonly used for data analytics, machine learning and AI.



<https://azure.microsoft.com/en-gb/services/virtual-machines/data-science-virtual-machines>

Deep Learning with Tensorflow

DEEP LEARNING

Machine Learning

Based on Neural Networks

More than one hidden layer (*Deep Models*)

Why?

- 
- Interface for expressing machine learning models
 - Implementation for executing such models
 - Handles multiple CPUs and GPUs
 - Multi-platform

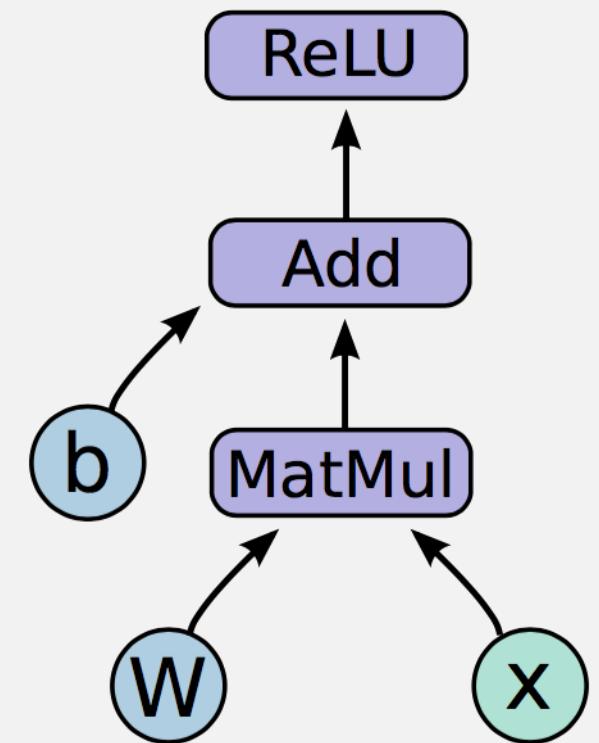
TensorFlow

TensorFlow

- Express a numeric computation as a **graph**
- Graph nodes are **operations** which have any number of inputs and outputs
- Graph edges are **tensors** which flow between nodes
- Development in **Python**

TensorFlow

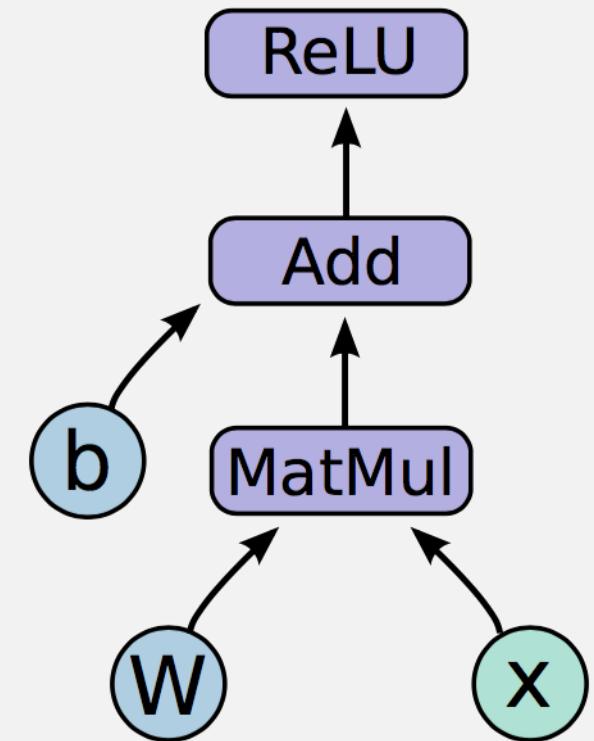
$$h_i = \text{ReLU}(Wx + b)$$



TensorFlow

$$h_i = \text{ReLU}(Wx + b)$$

- **Variables:**
 - Nodes which emit their value as output
 - State is persisted, and retained across multiple executions of the graph
 - Examples: parameters

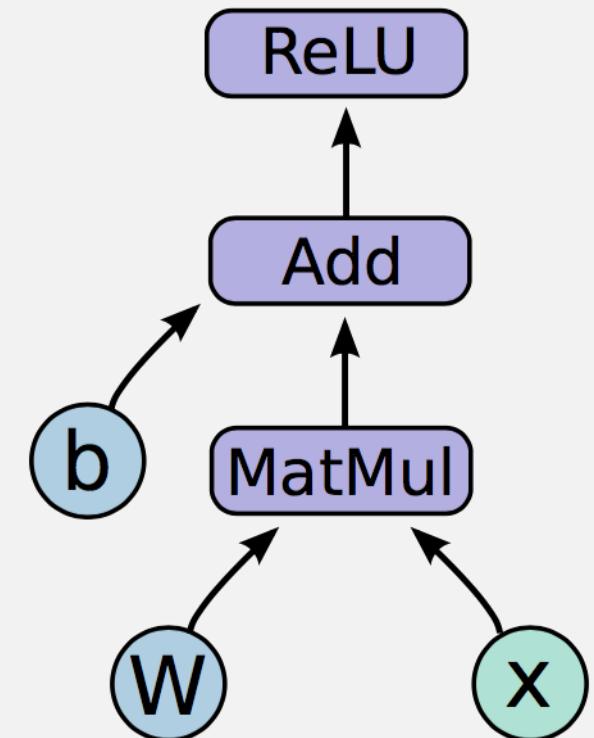


TensorFlow

$$h_i = \text{ReLU}(Wx + b)$$

- **Placeholders:**

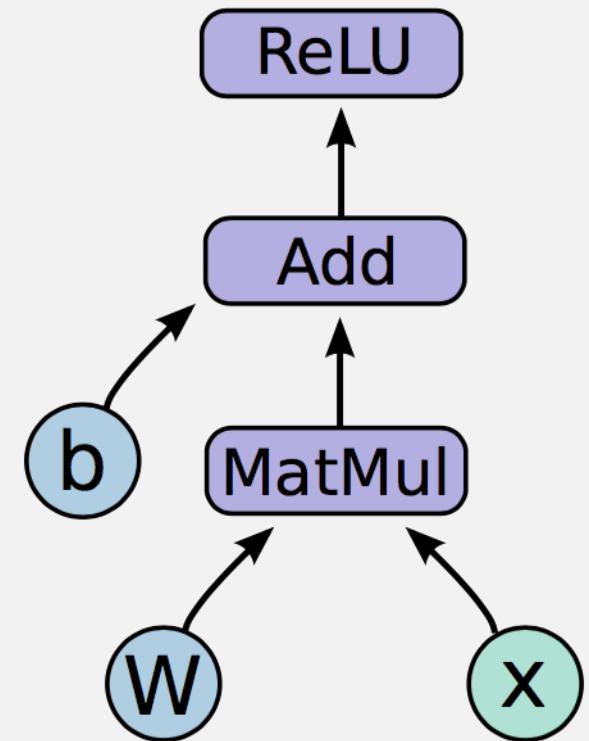
- Values are fed at execution time
- Examples: inputs, hyper-parameters



TensorFlow

$$h_i = \text{ReLU}(Wx + b)$$

- **Operations:**
 - Perform the specific operation with the tensor input stream
 - Examples: MatMul, Add, ReLU

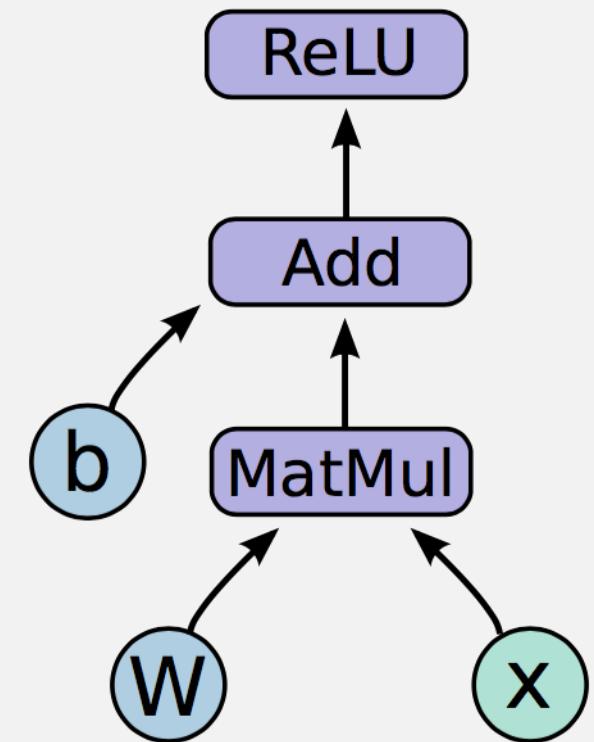


TensorFlow – Graph Code

```
import tensorflow as tf

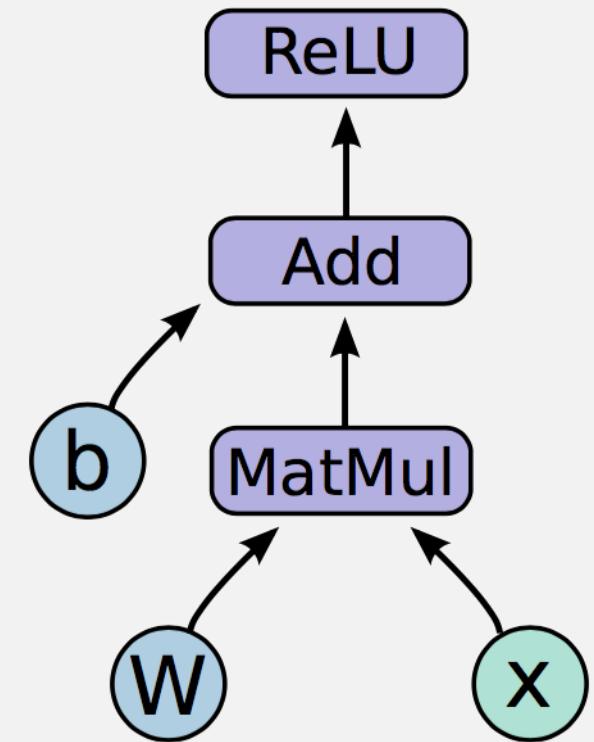
b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (None, 784))
h_i = tf.nn.relu(tf.matmul(x, W) + b)
```



TensorFlow – Graph Execution

- **Session:**
 - Once we have defined our **graph**, we need to execute it
 - We can deploy the graph to a **session**
 - These sessions can be binded to an **execution context** (GPU, CPU, ...)



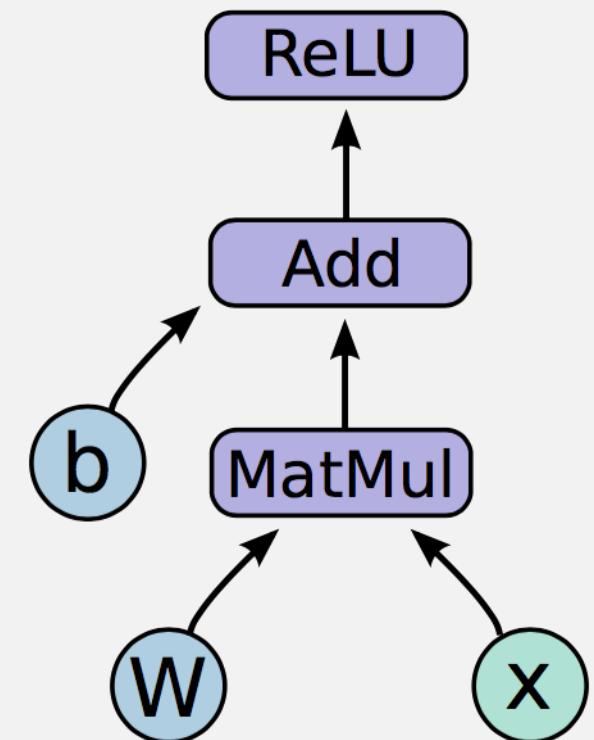
TensorFlow – Execution Code

```
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (None, 784))
h_i = tf.nn.relu(tf.matmul(x, W) + b)

sess = tf.Session()
sess.run(tf.initialize_all_variables())
sess.run(h_i, {x: np.random.random(64, 784)})
```

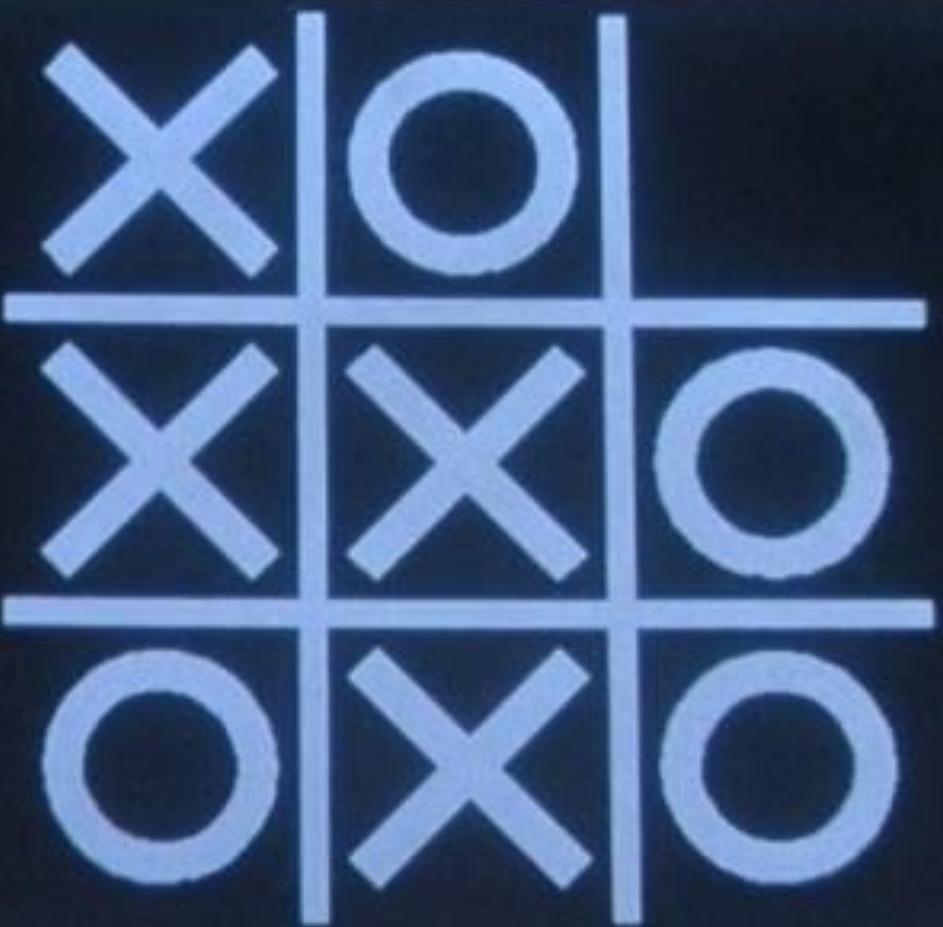


LAB 1: TensorFlow Operations

TensorFlow 2.0

- Eager Execution as default
 - No need to add `tf.enable_eager_execution()`
 - Blocks or Functions can be defined to be executed in graph mode
- Less conventions, more object-oriented and pythonic design
 - `Variable_scope` removed
- Clean up in libraries and contrib

SHALL WE PLAY A LITTLE GAME?



YLING
4
COM STS
8365

OMM STS
3603

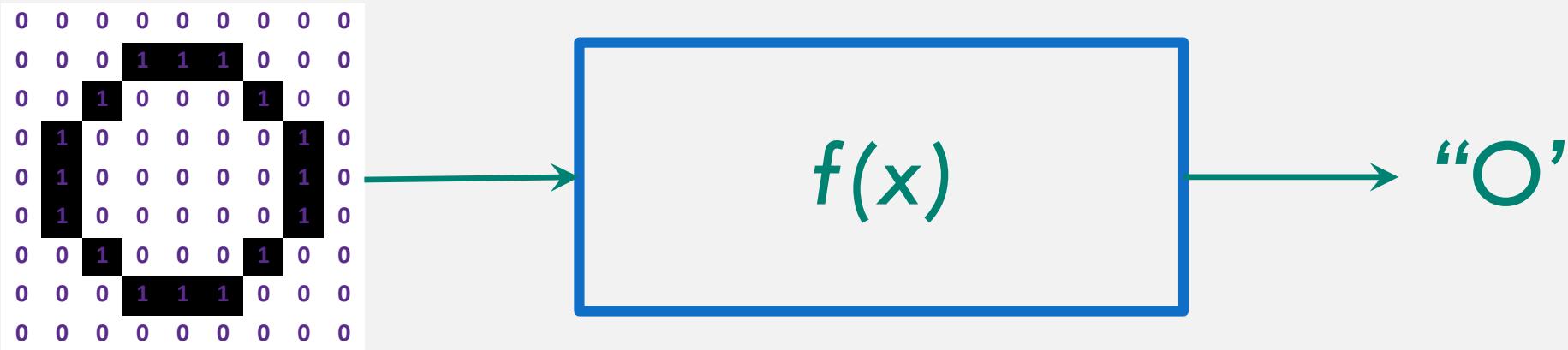
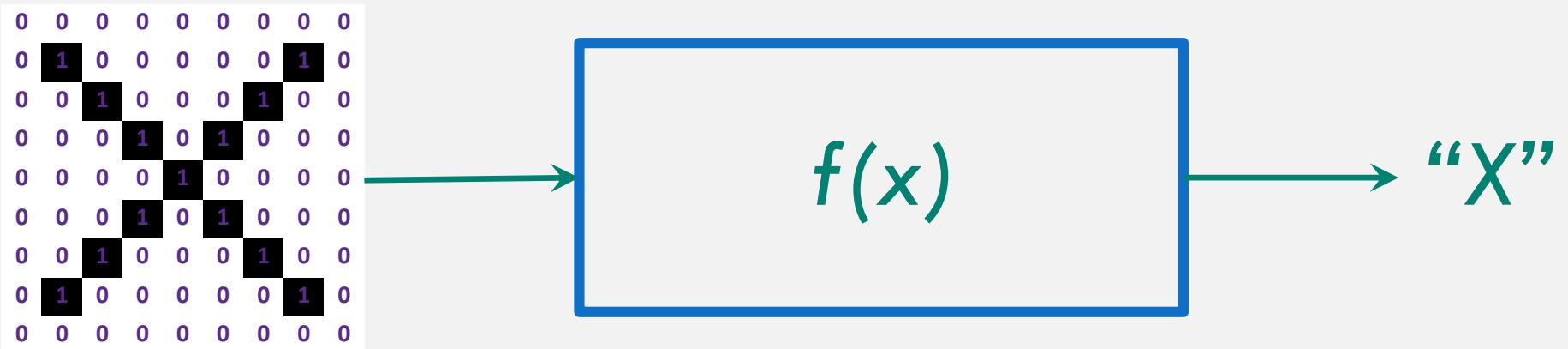
RPL STS
9071

EOT STS
4531

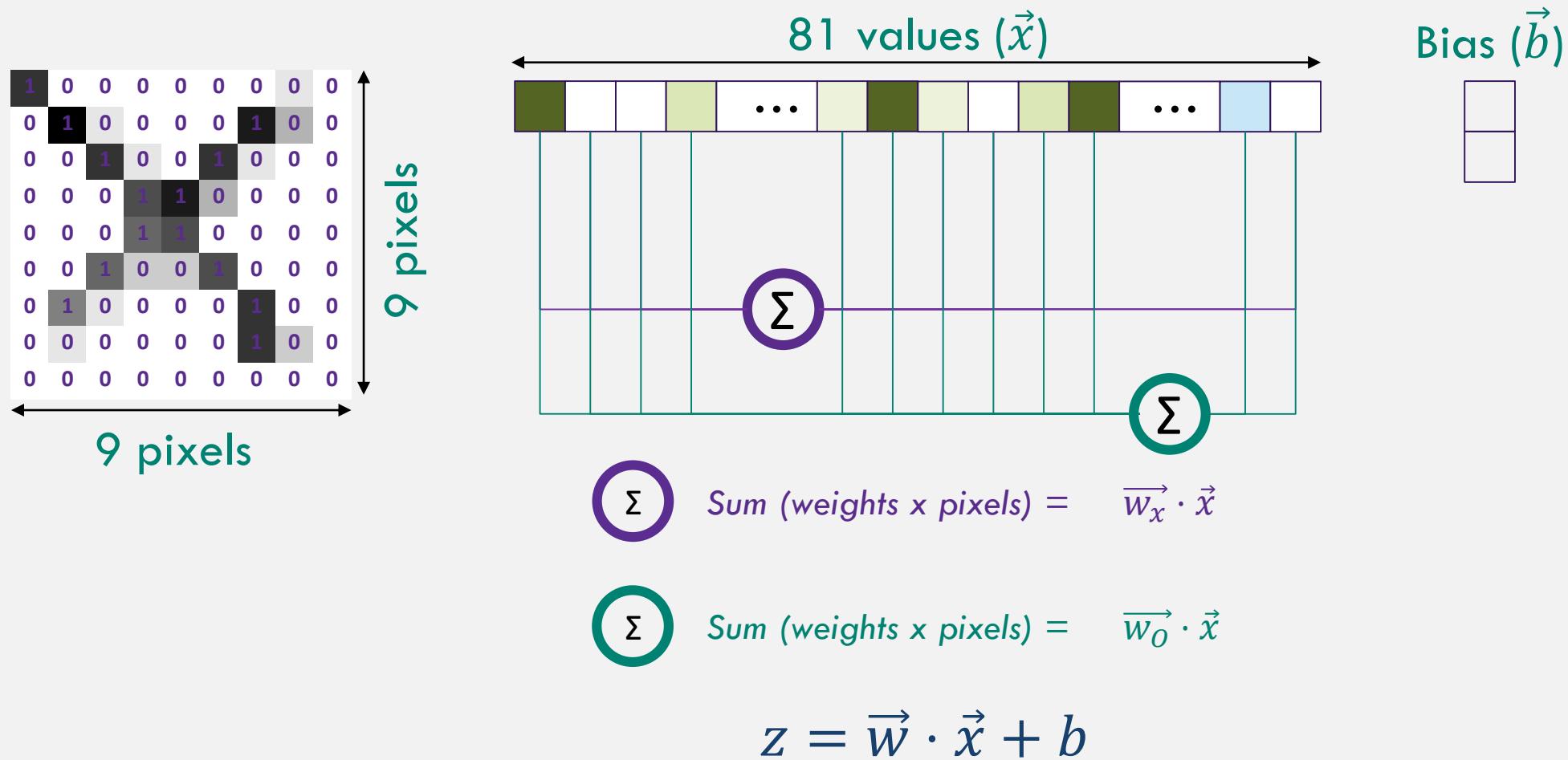
PAC STS
8838

ADL STS
6632

A SIMPLE EXAMPLE

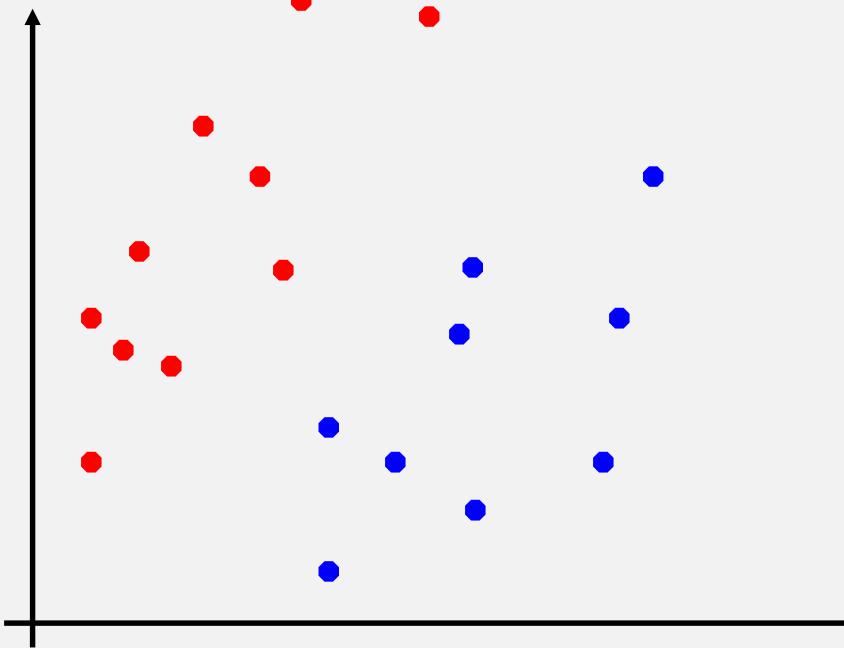


A SIMPLE EXAMPLE



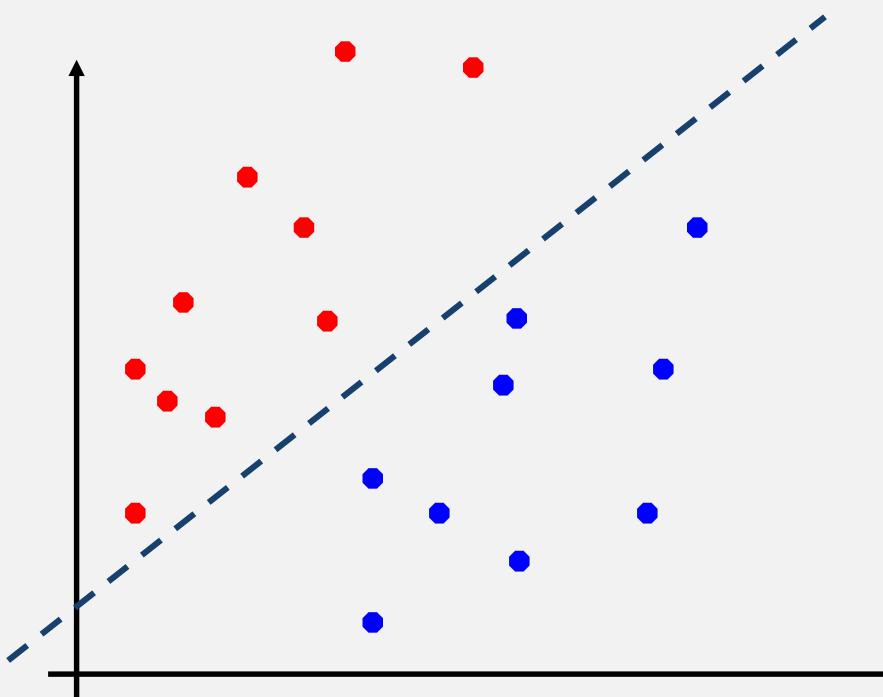
NOW THAT YOU MENTION IT...

$$z = w \cdot \vec{x} + b$$



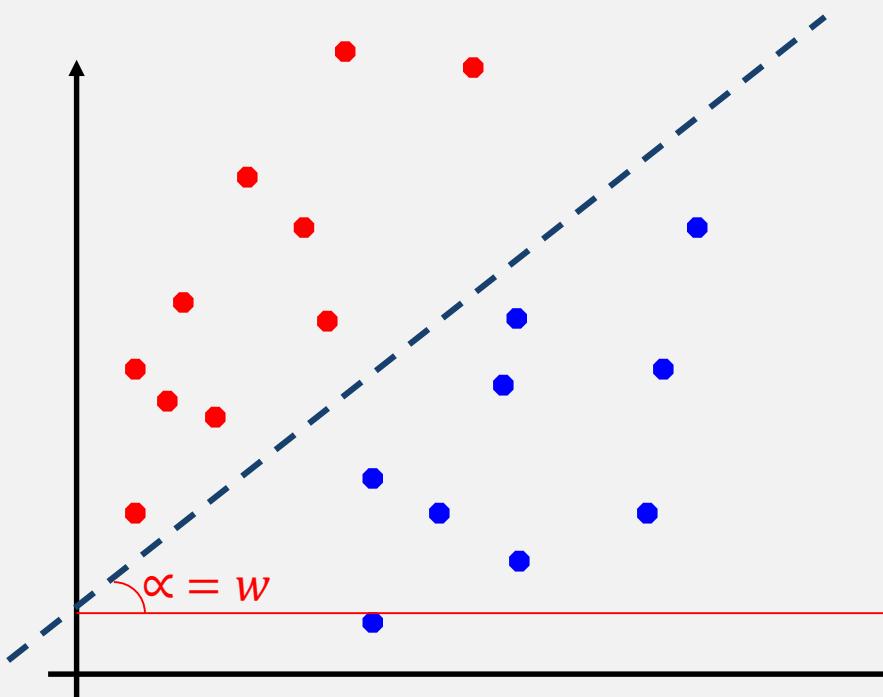
NOW THAT YOU MENTION IT...

$$z = \mathbf{w} \cdot \vec{x} + b$$

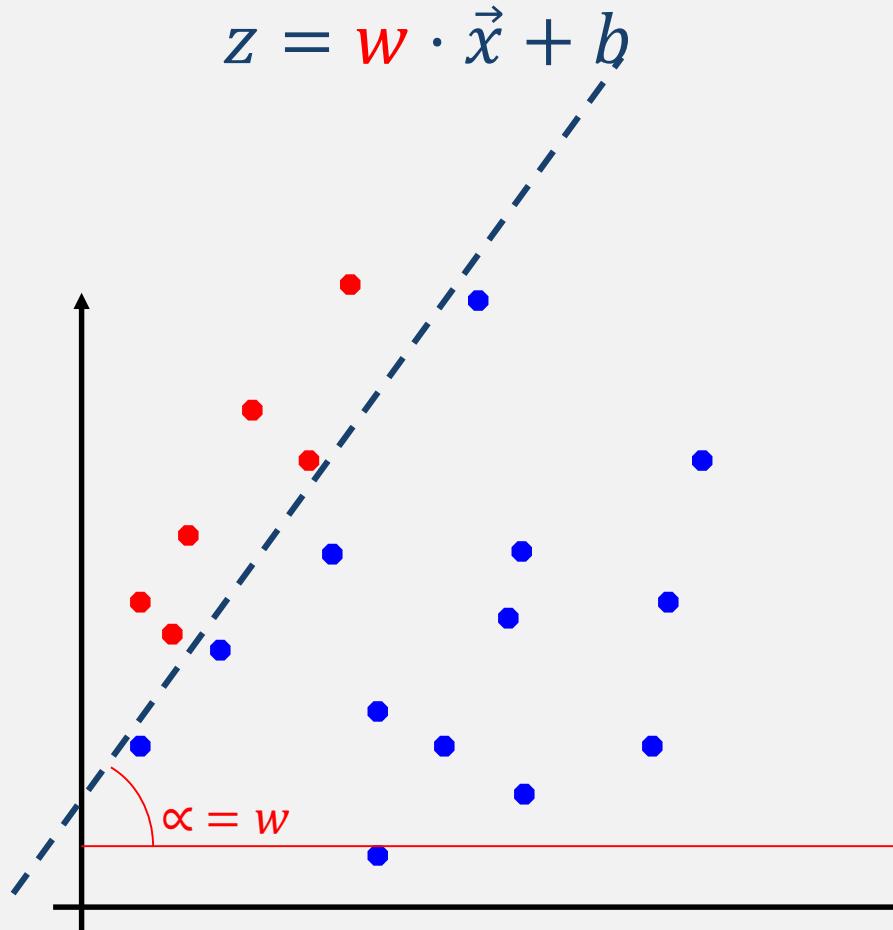


NOW THAT YOU MENTION IT...

$$z = \mathbf{w} \cdot \vec{x} + b$$

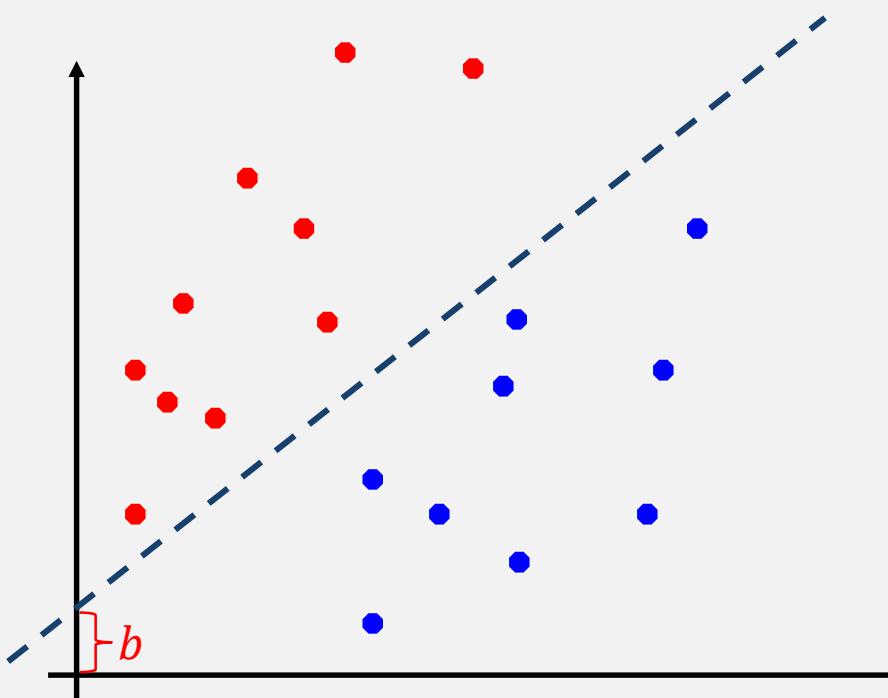


NOW THAT YOU MENTION IT...



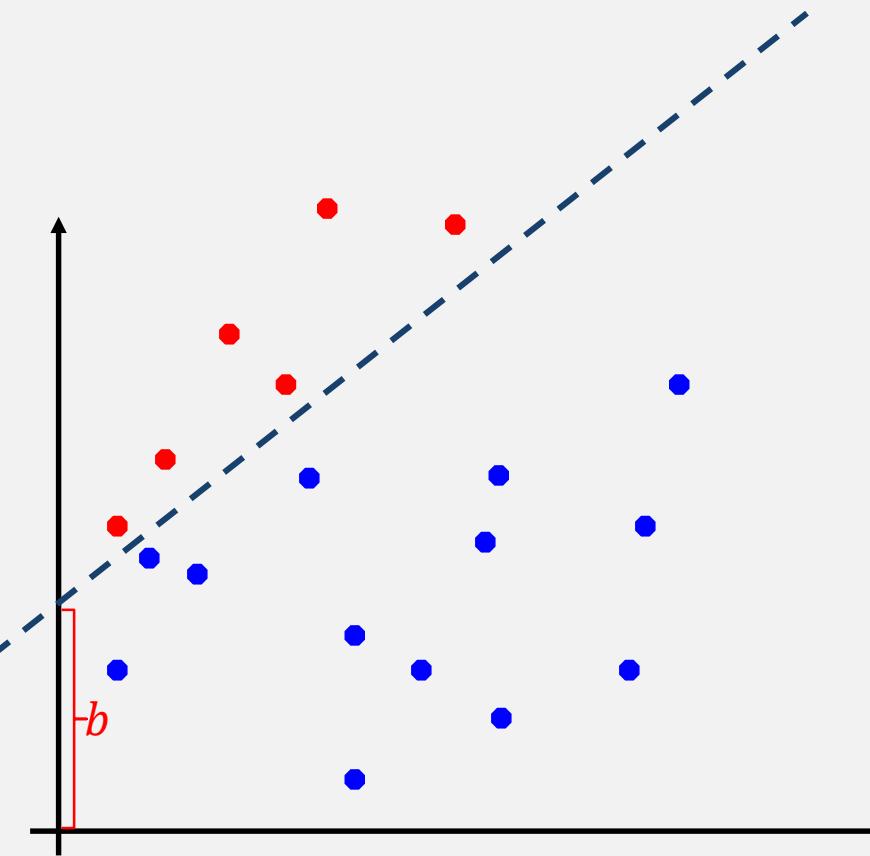
NOW THAT YOU MENTION IT...

$$z = w \cdot \vec{x} + b$$

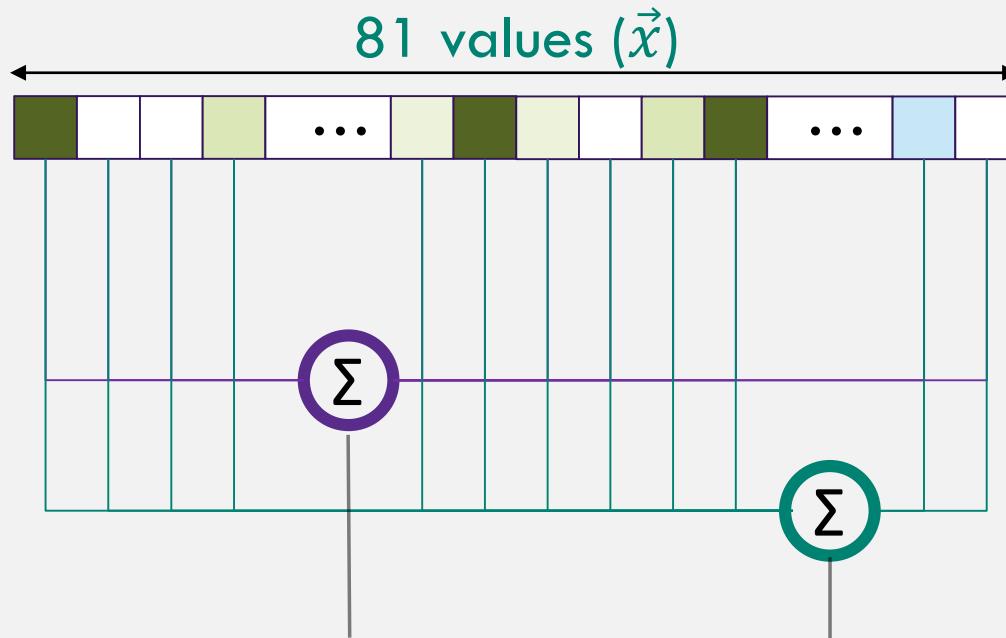
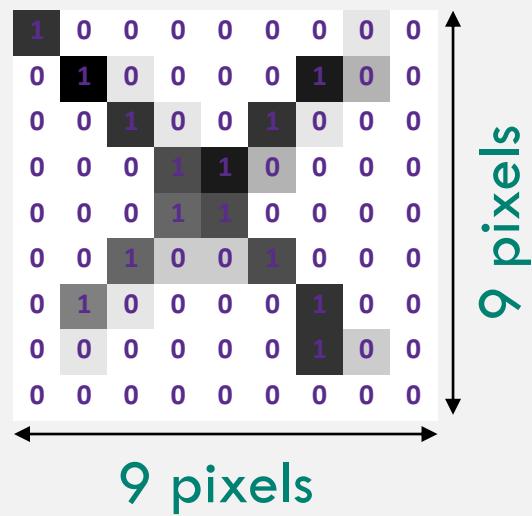


NOW THAT YOU MENTION IT...

$$z = w \cdot \vec{x} + b$$



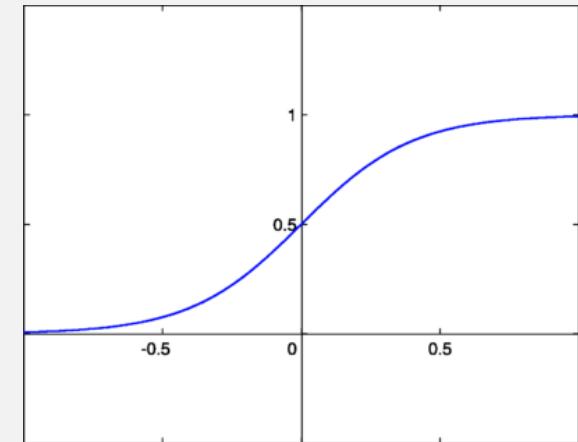
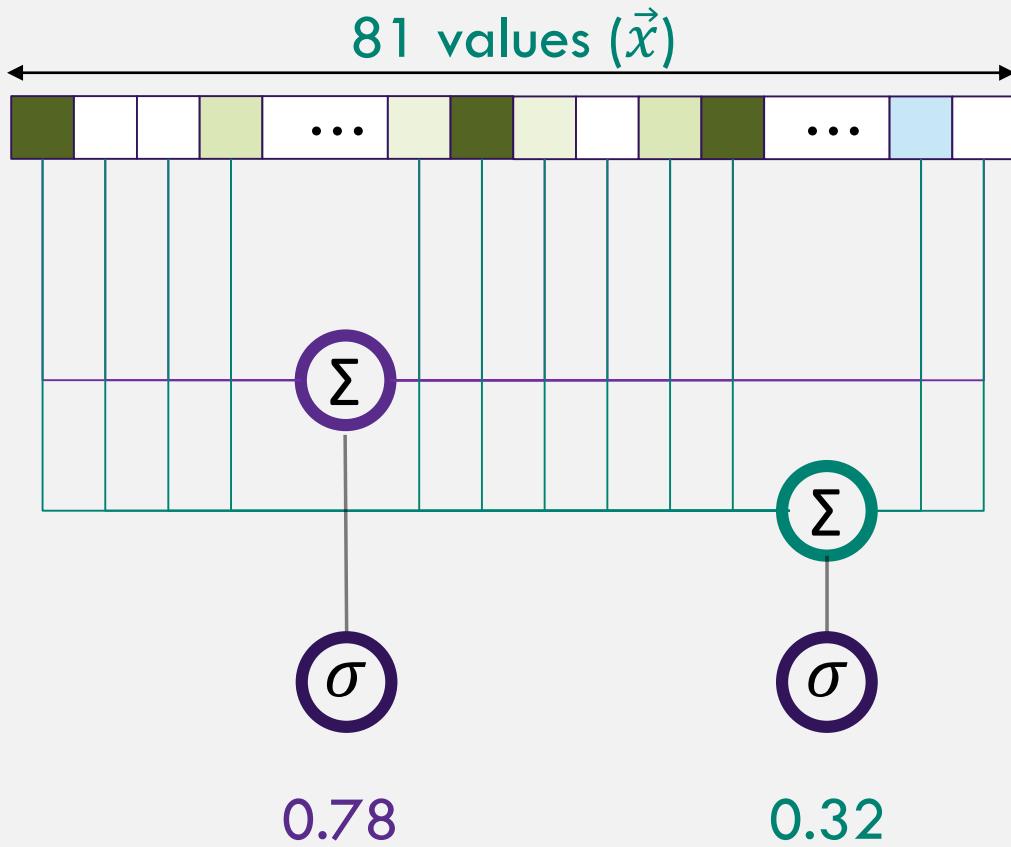
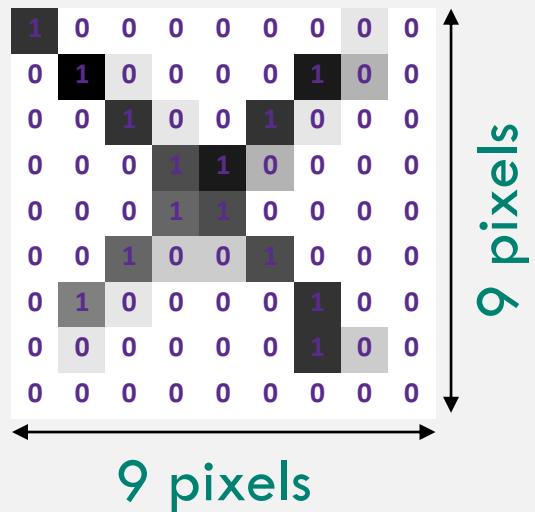
SIGMOID



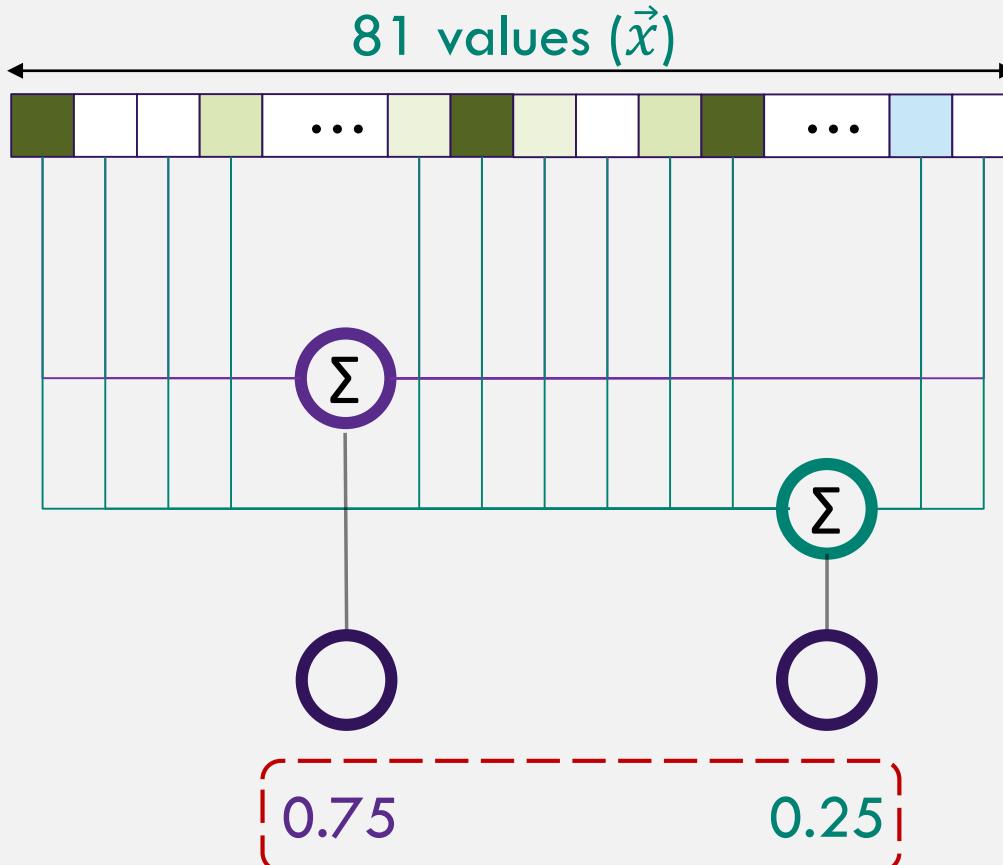
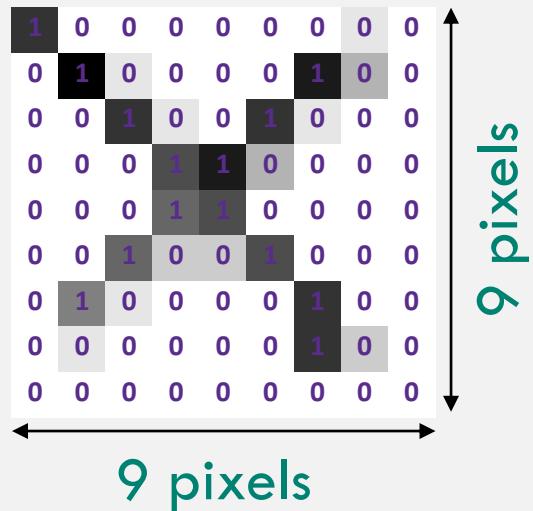
823,731.01

14,78

SIGMOID

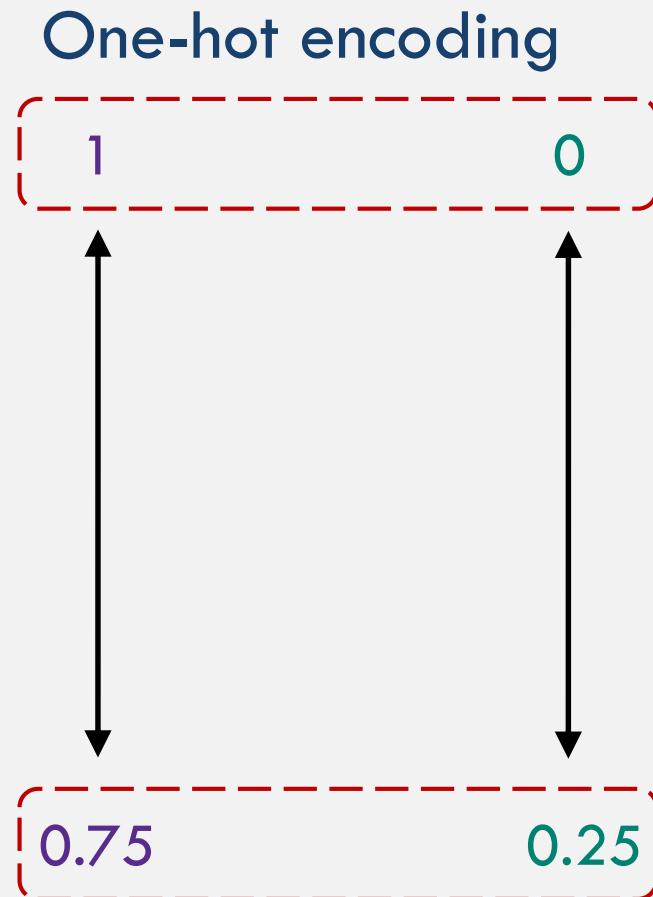
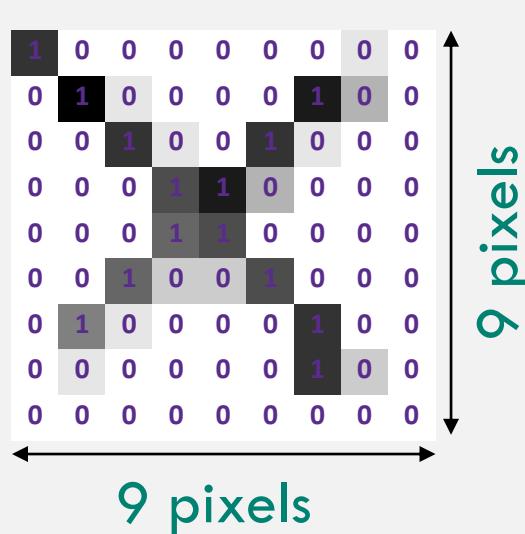


SOFTMAX

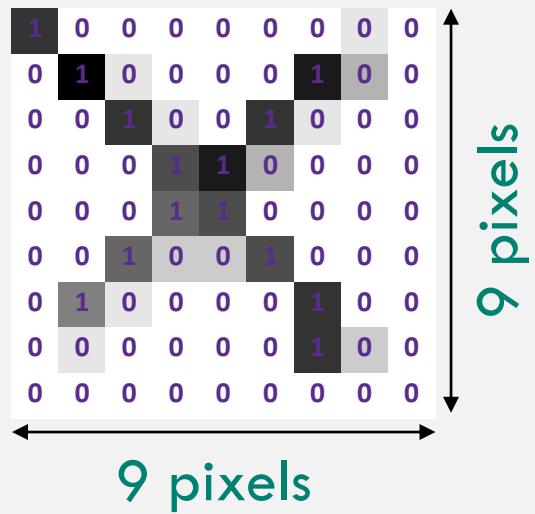


$$s(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$$

LOSS FUNCTION



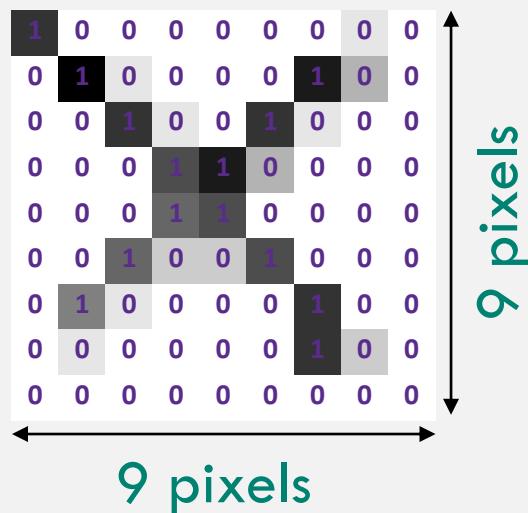
LOSS FUNCTION



$$\text{error} = \frac{1}{n} \sum_{i=1}^n (y_i - p_i)^2$$



LOSS FUNCTION

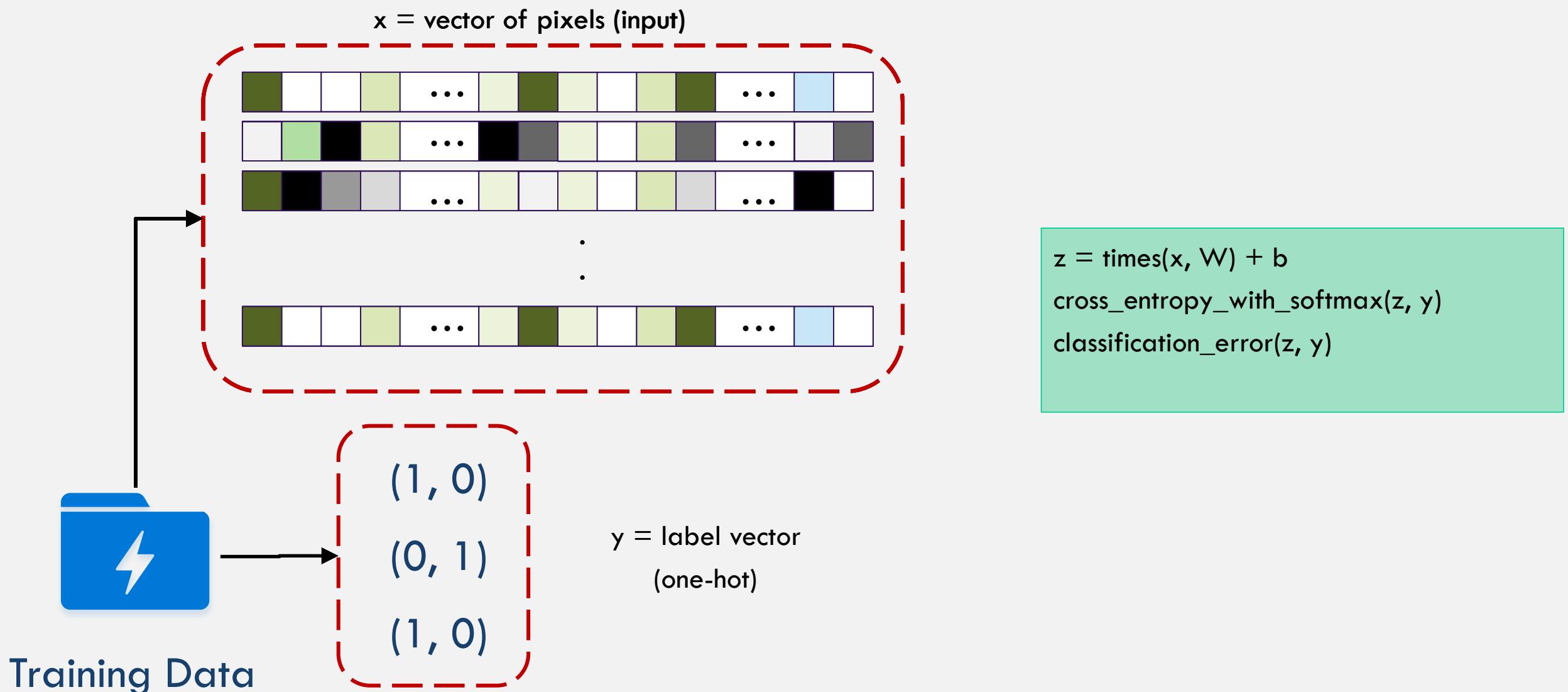


$$\text{Cross Entropy Error} = - \sum_{c=1}^M y_{0,c} \log(p_{0,c})$$

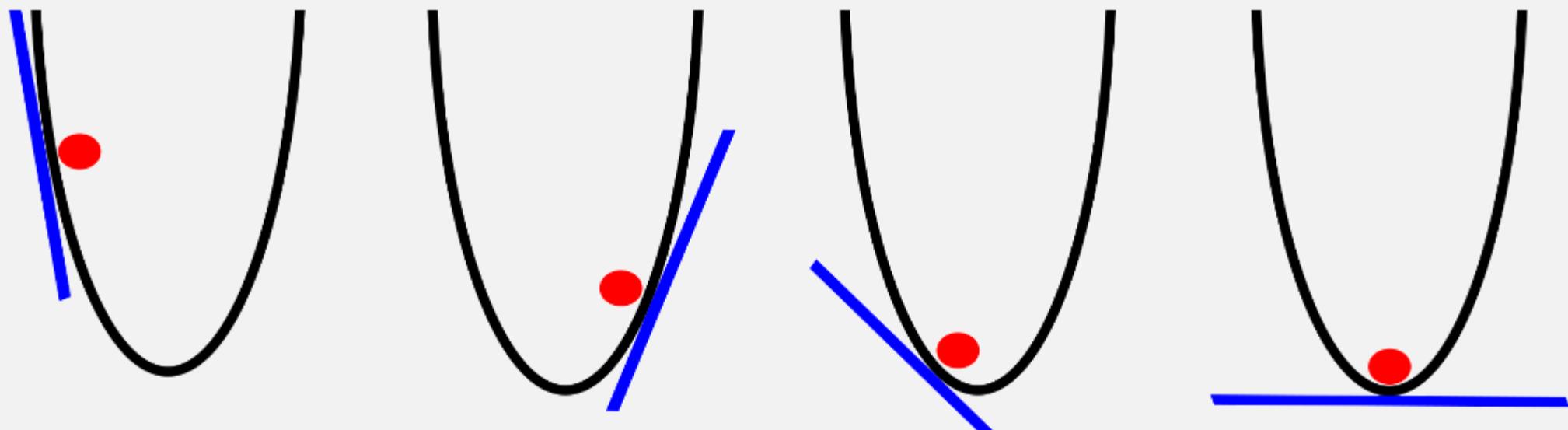
$$\text{Binary: } -(y * \log(p) + (1-y) * \log(1-p))$$



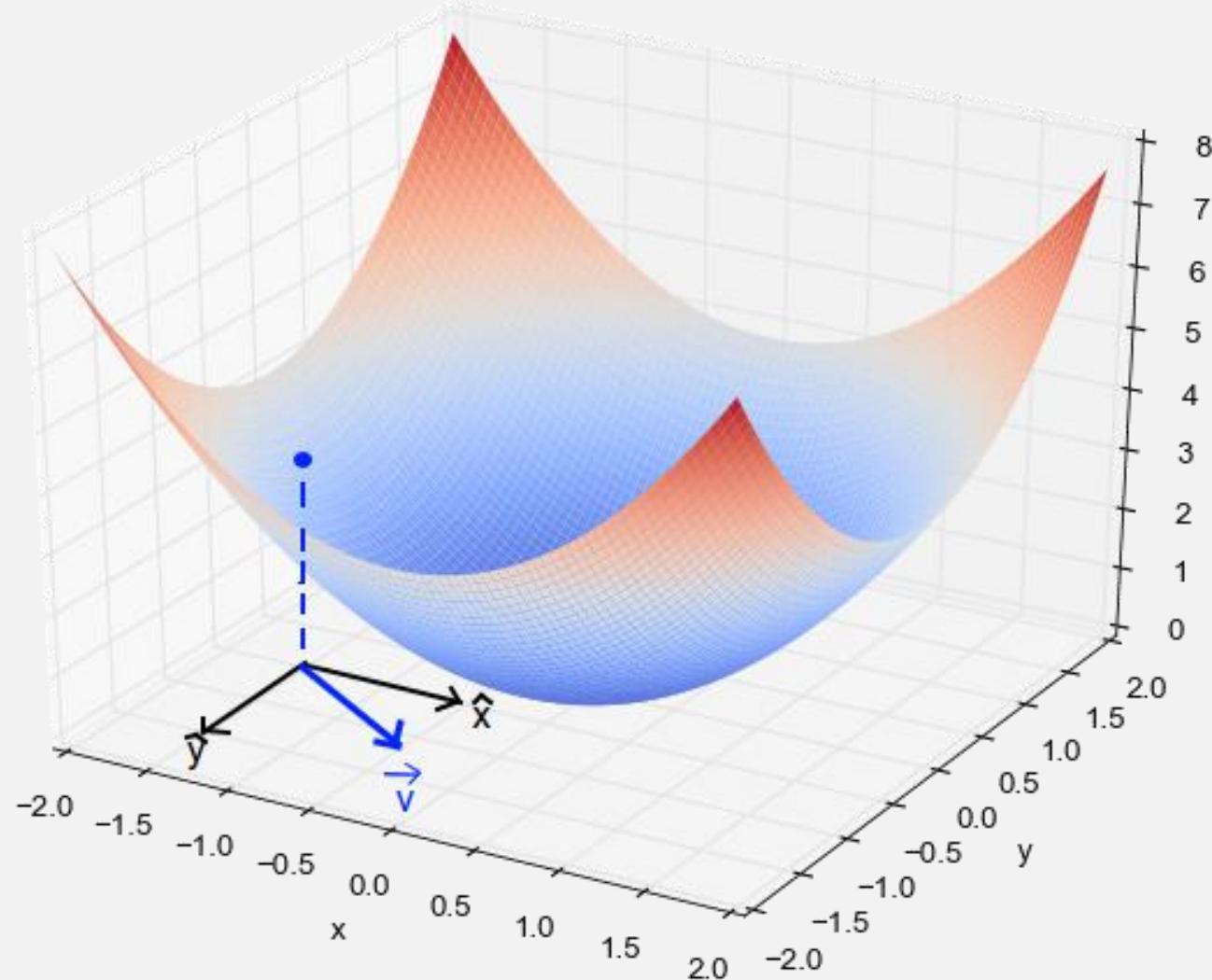
TRAINING



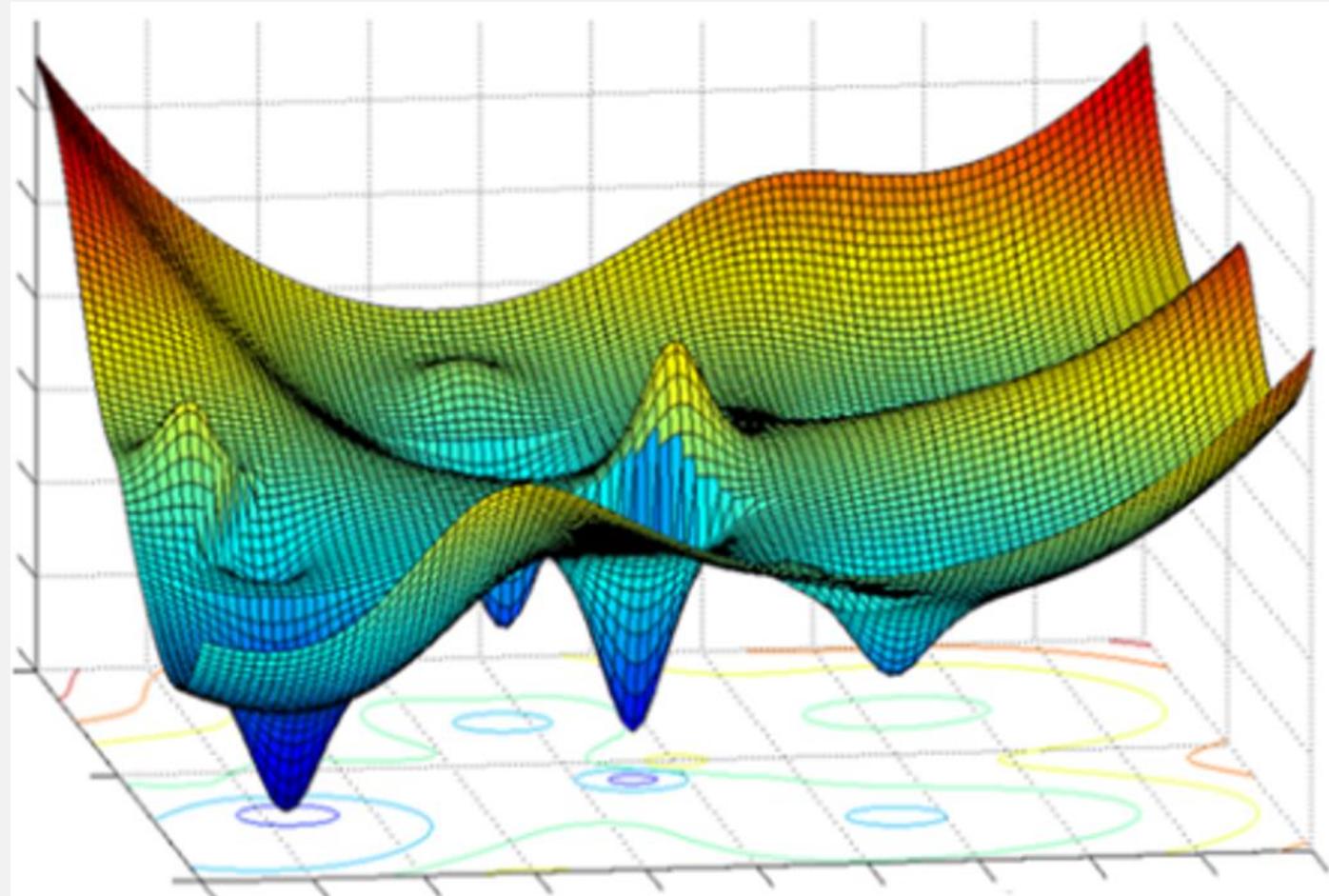
GRADIENT DESCENT



GRADIENT DESCENT



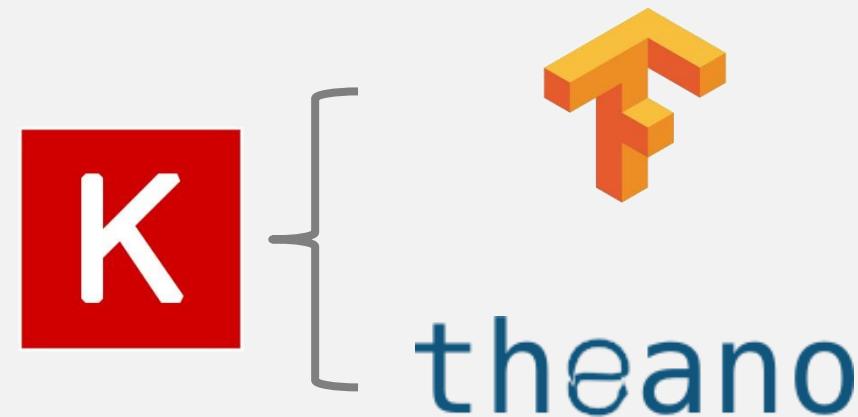
GRADIENT DESCENT



LAB 2: Logistic Regression

Keras

- Mission: to make building Deep architectures easier
- Works on top of TensorFlow or Theano
- Built by François Chollet at Google



Keras

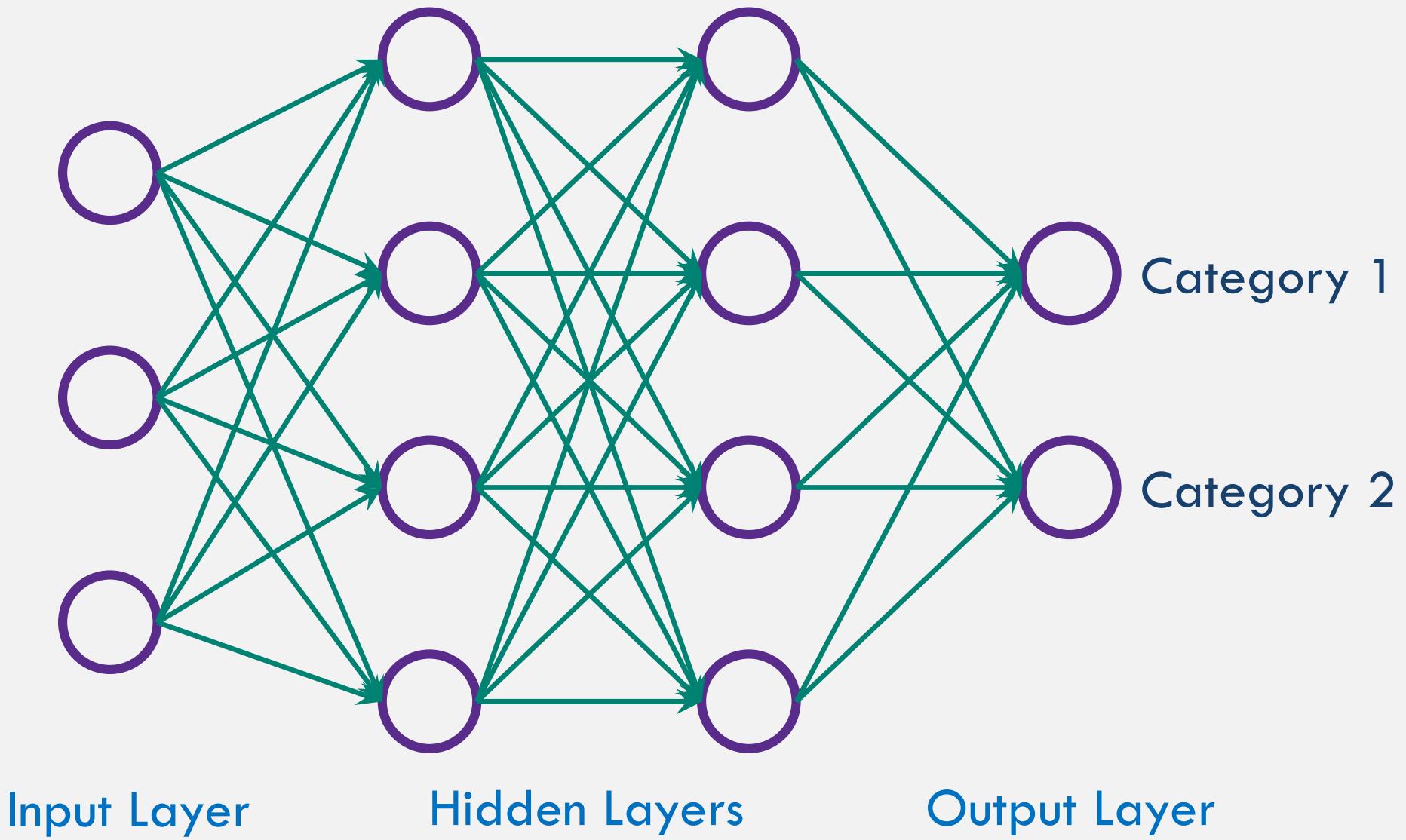
- Based on layers and their input/output tensors

```
#build model (using the softmax activation function)
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='SGD', metrics=['accuracy'])
```

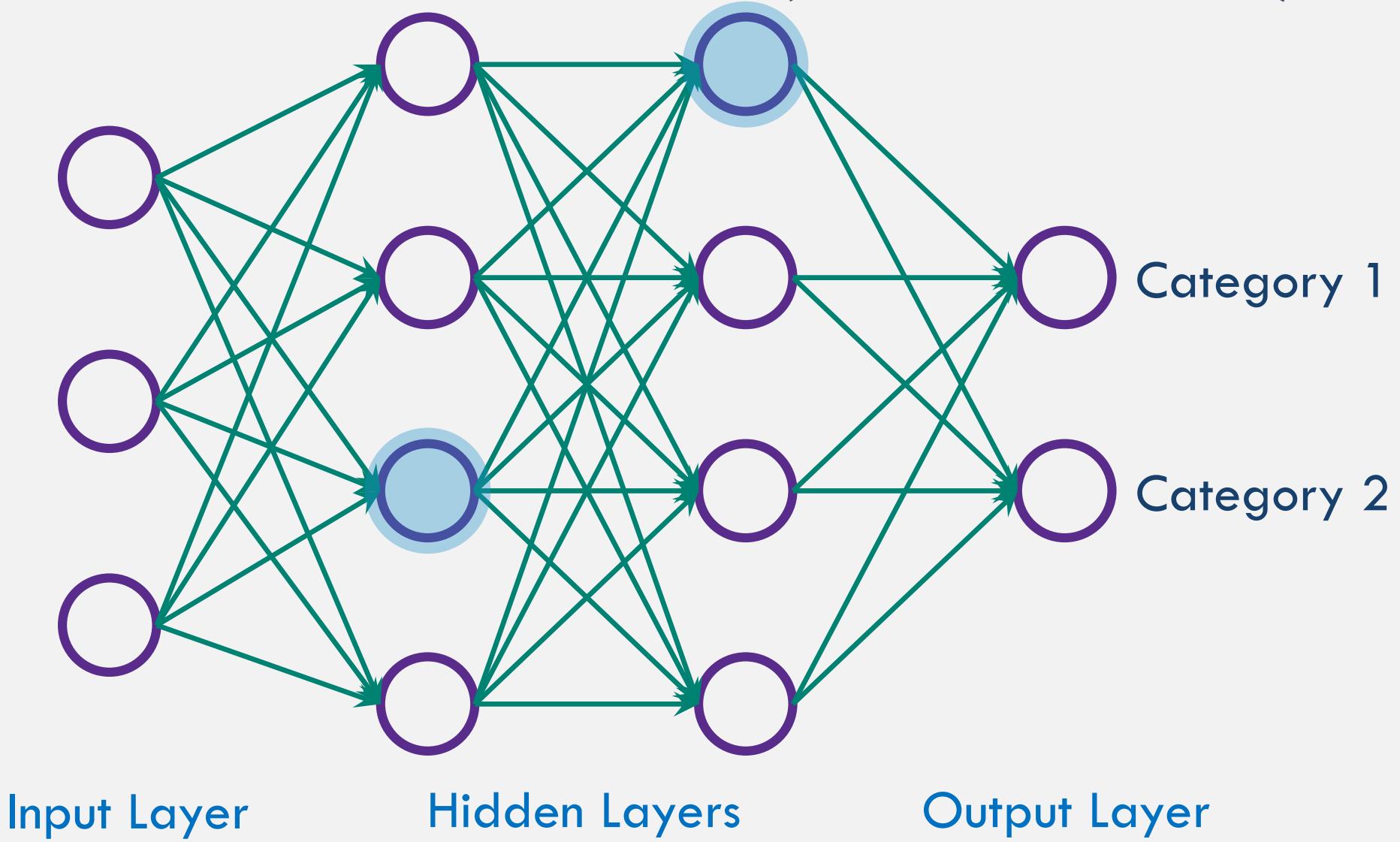
LAB 3: Logistic Regression (Keras)

Neural Networks

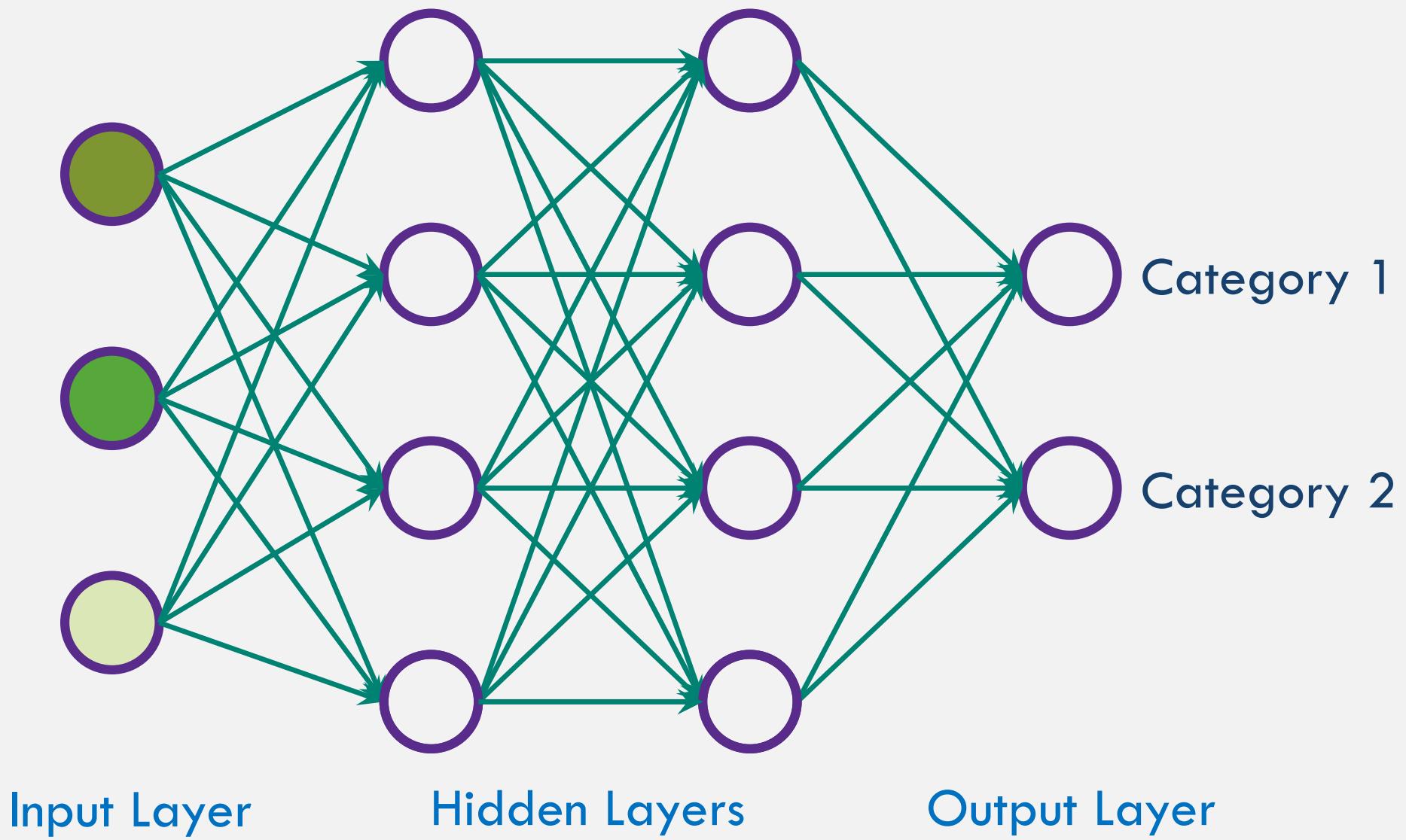
NEURAL NETWORK (CLASSIFIER)



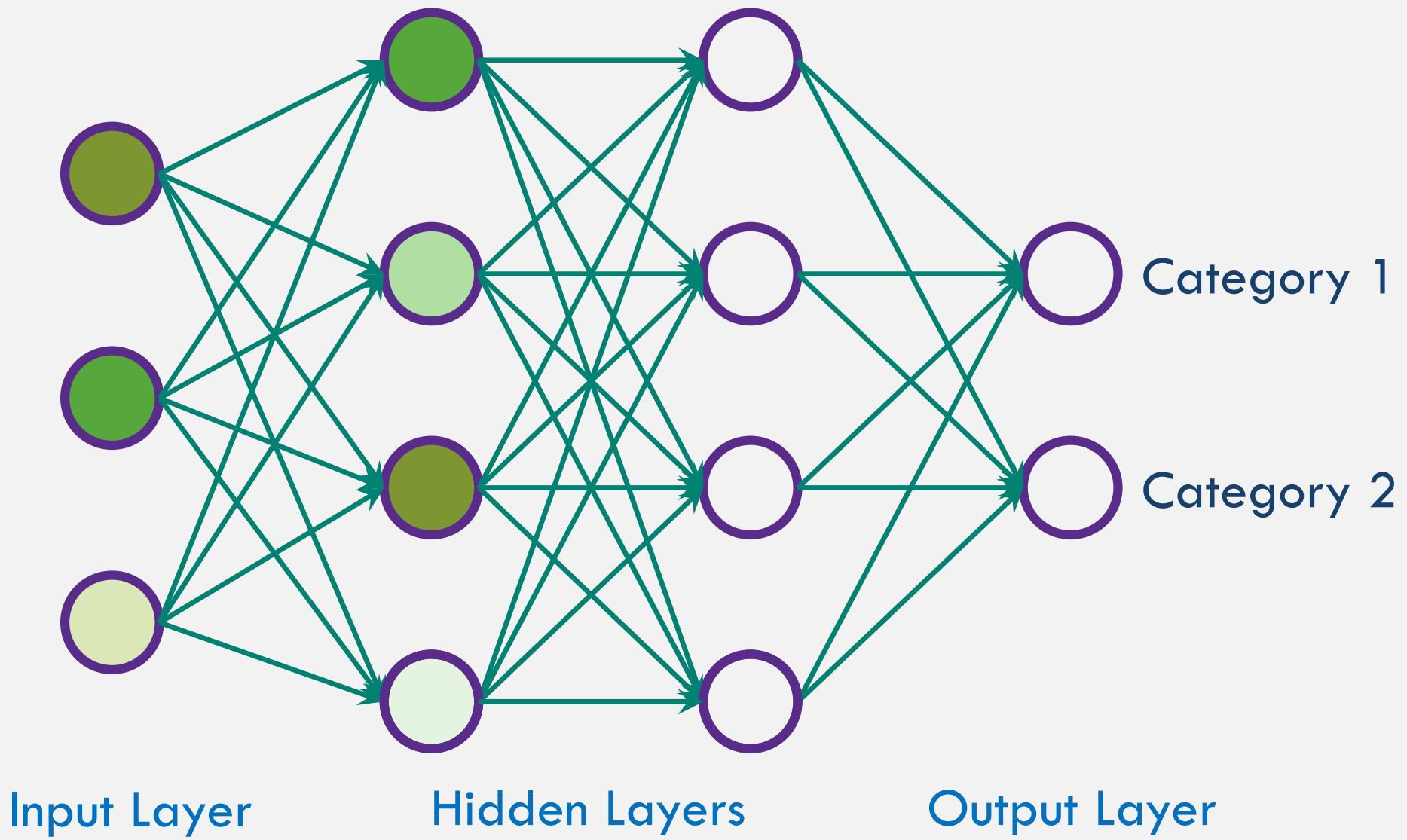
NEURAL NETWORK (CLASSIFIER)



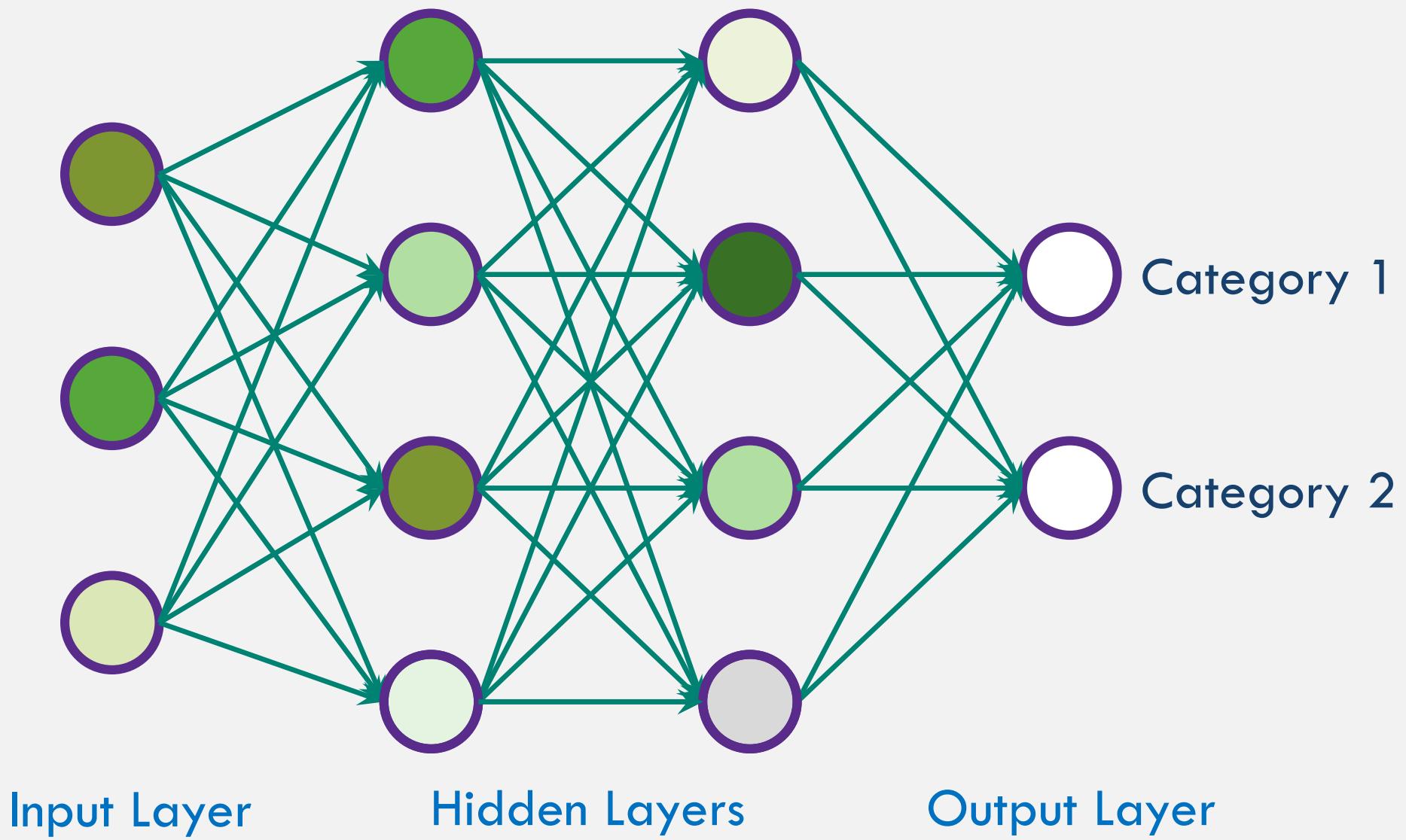
FORWARD PROPAGATION



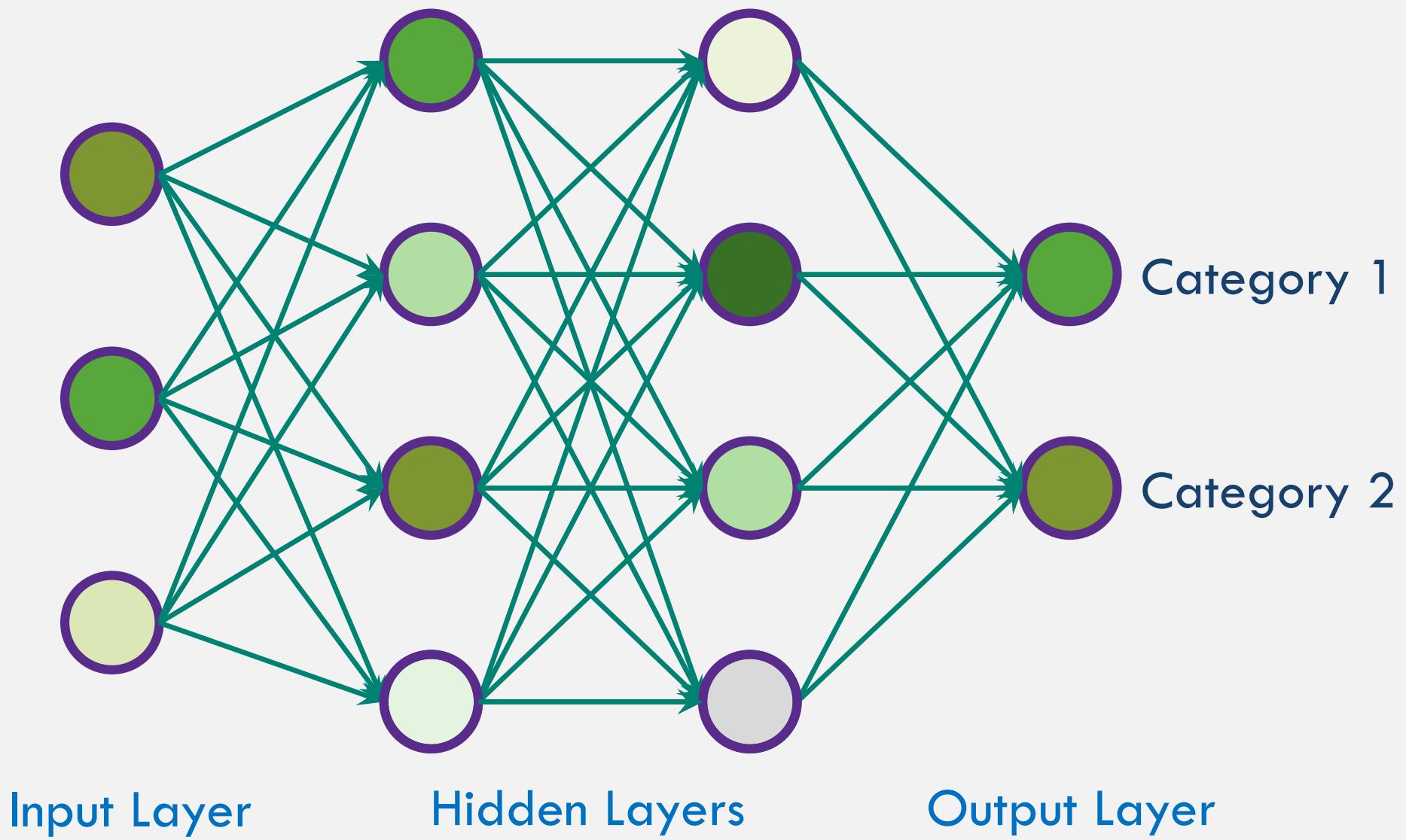
FORWARD PROPAGATION



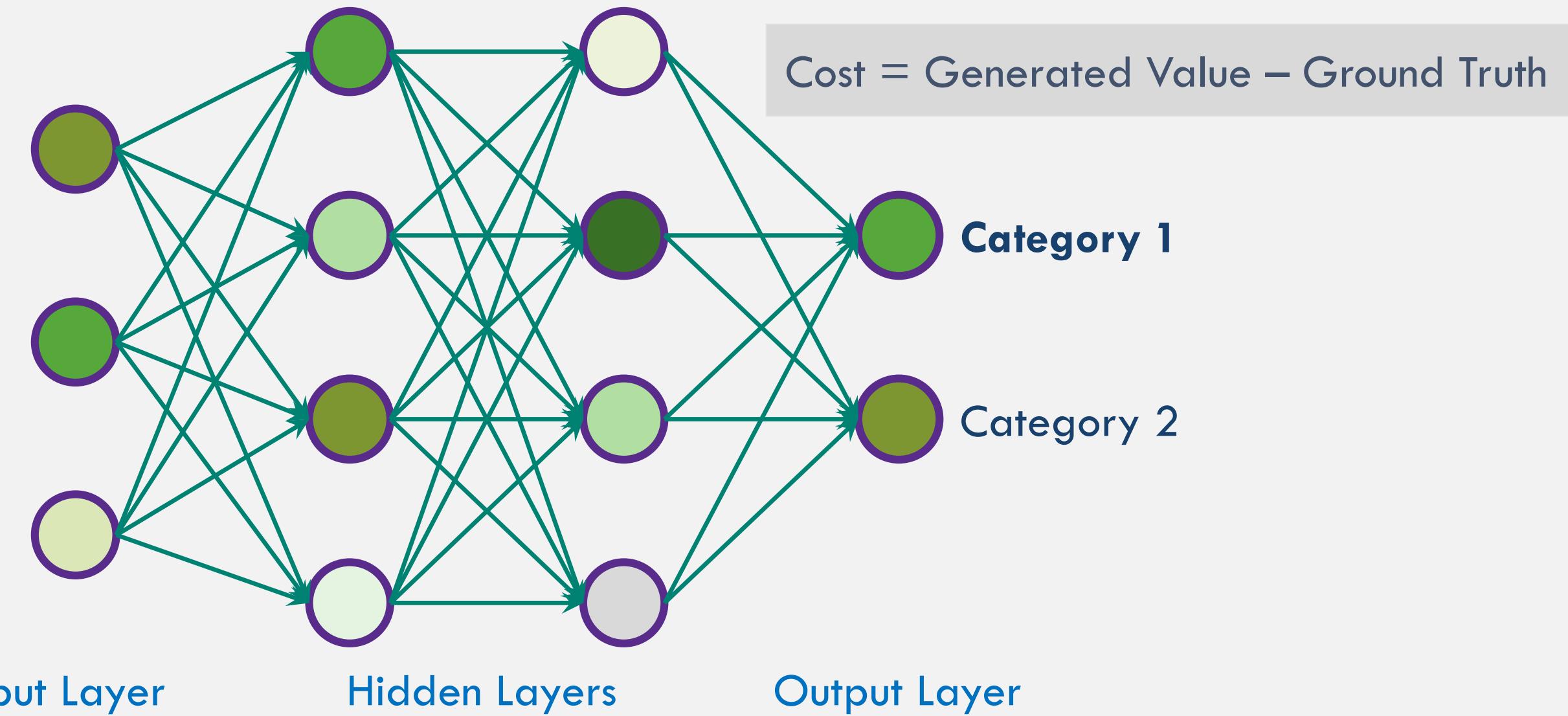
FORWARD PROPAGATION



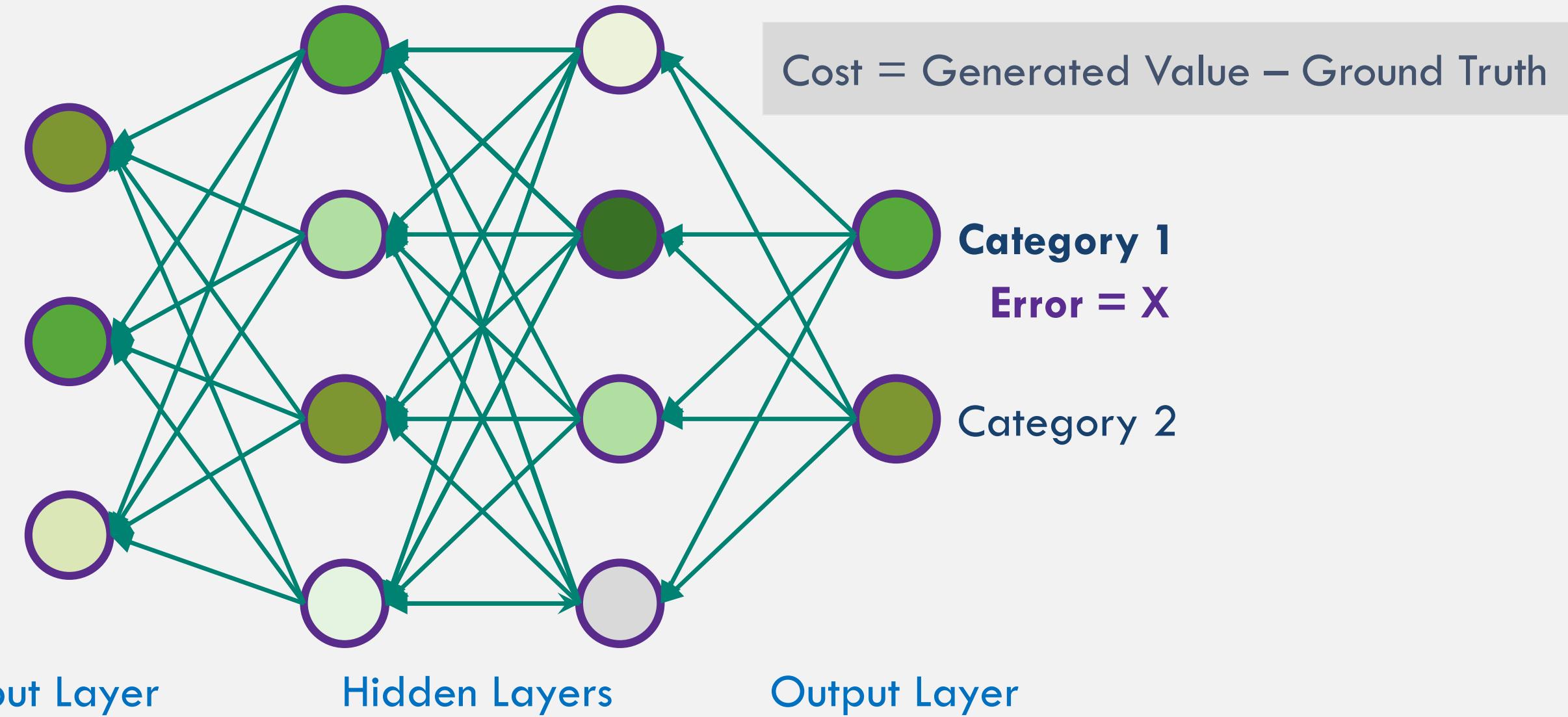
FORWARD PROPAGATION



TRAINING



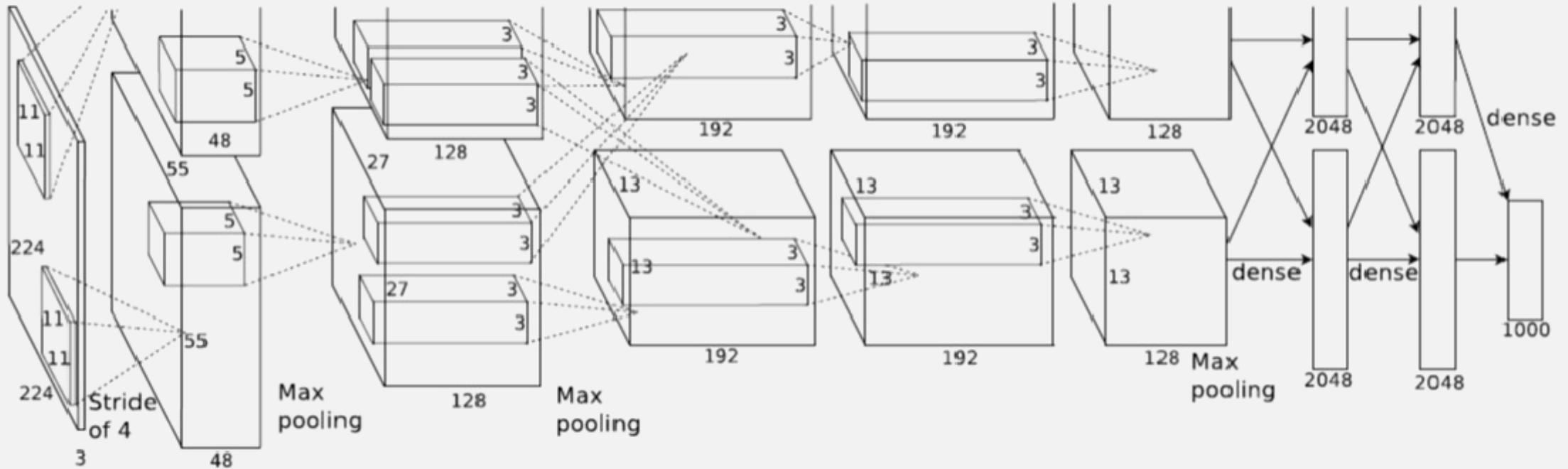
BACKPROPAGATION



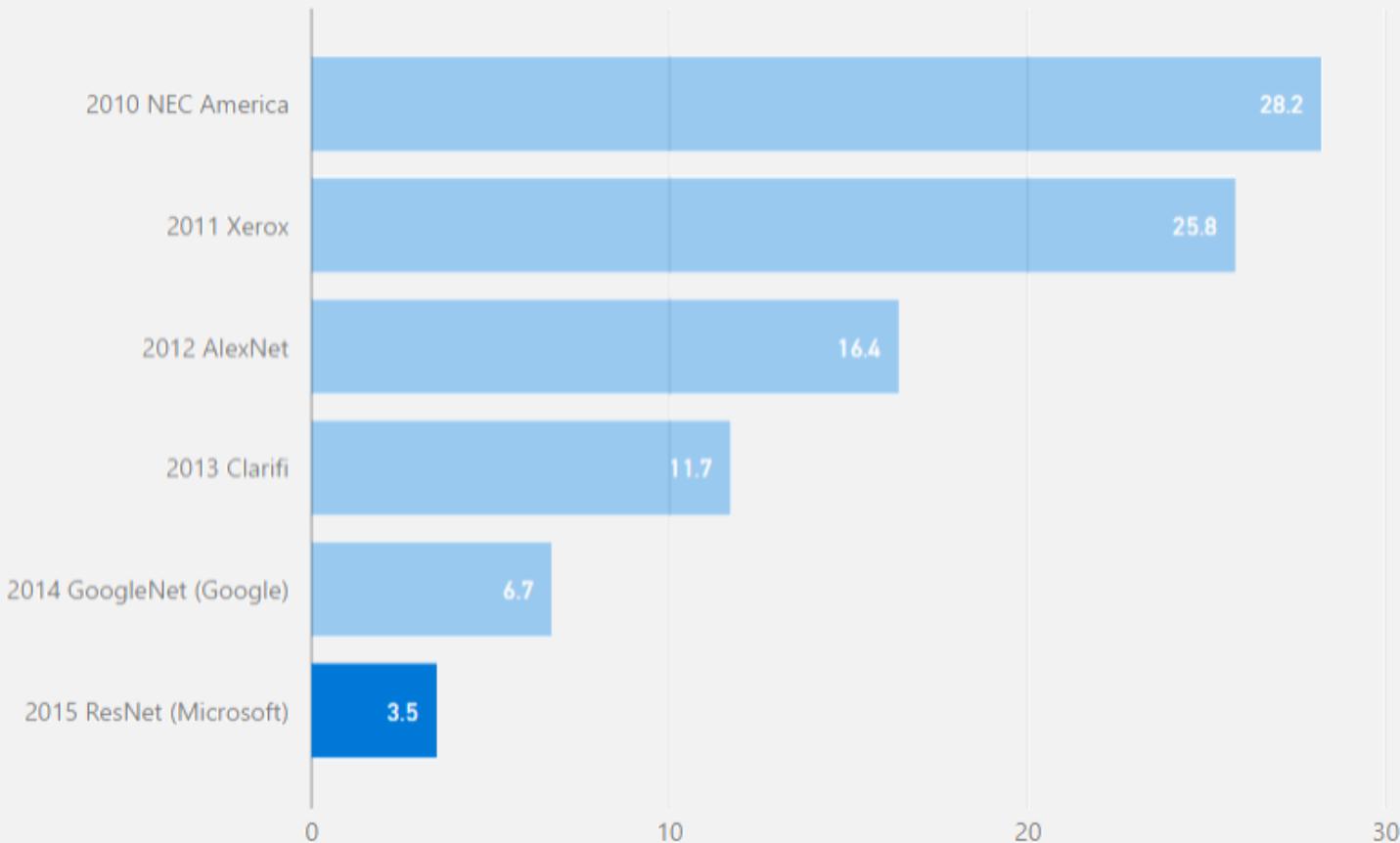
LAB 4: Multi-Layer Perceptron

Convolutional Networks

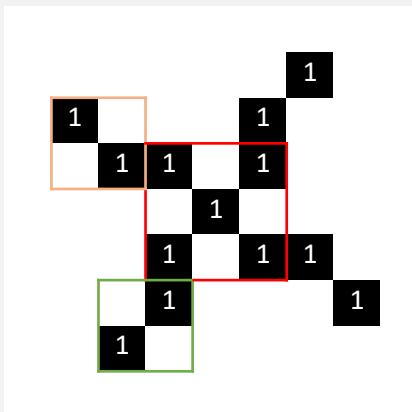
REMEMBER THIS THING?



IMAGENET CHALLENGE



FILTERING



FILTERING

...looking for common characteristics

Filter 1

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

1	-1	-1
-1	1	-1
-1	-1	1

1	1	1
1	1	1
1	1	1

1.0

Filter 2

-1	-1	1
-1	1	-1
1	-1	-1

-1	1	-1
1	1	1
-1	1	-1

0.11

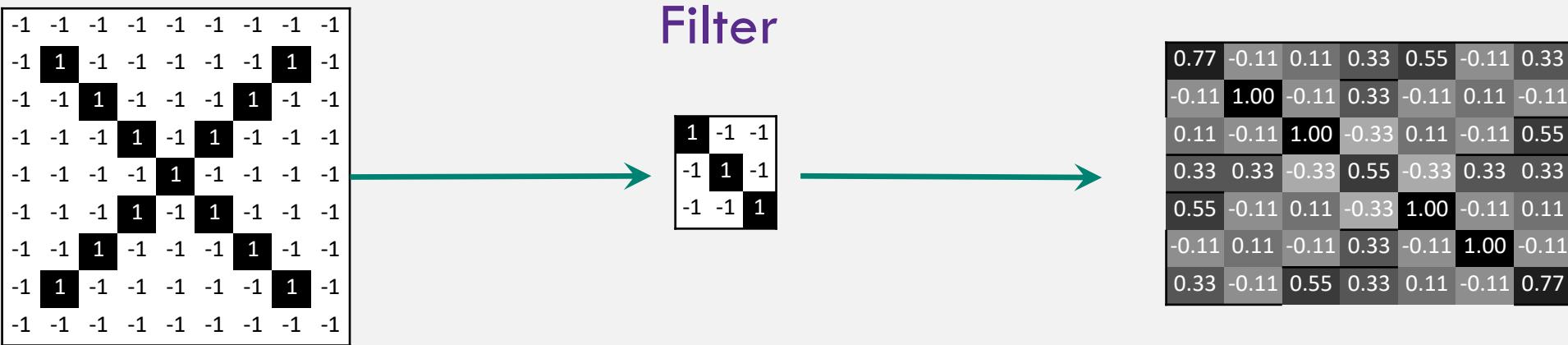
Filter 3

1	-1	1
-1	1	-1
1	-1	1

1	1	-1
1	1	1
-1	1	1

0.55

CONVOLUTION



CONVOLUTION

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

=

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

CONVOLUTION

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

=

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

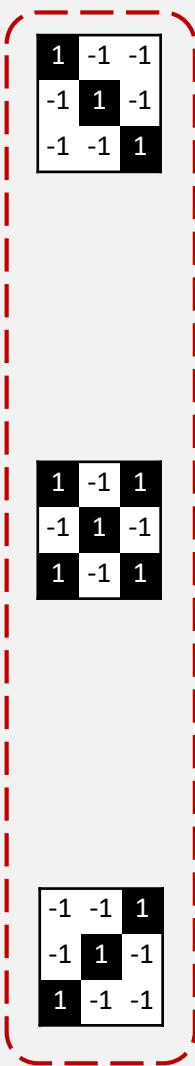


-1	-1	1
-1	1	-1
1	-1	-1

=

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



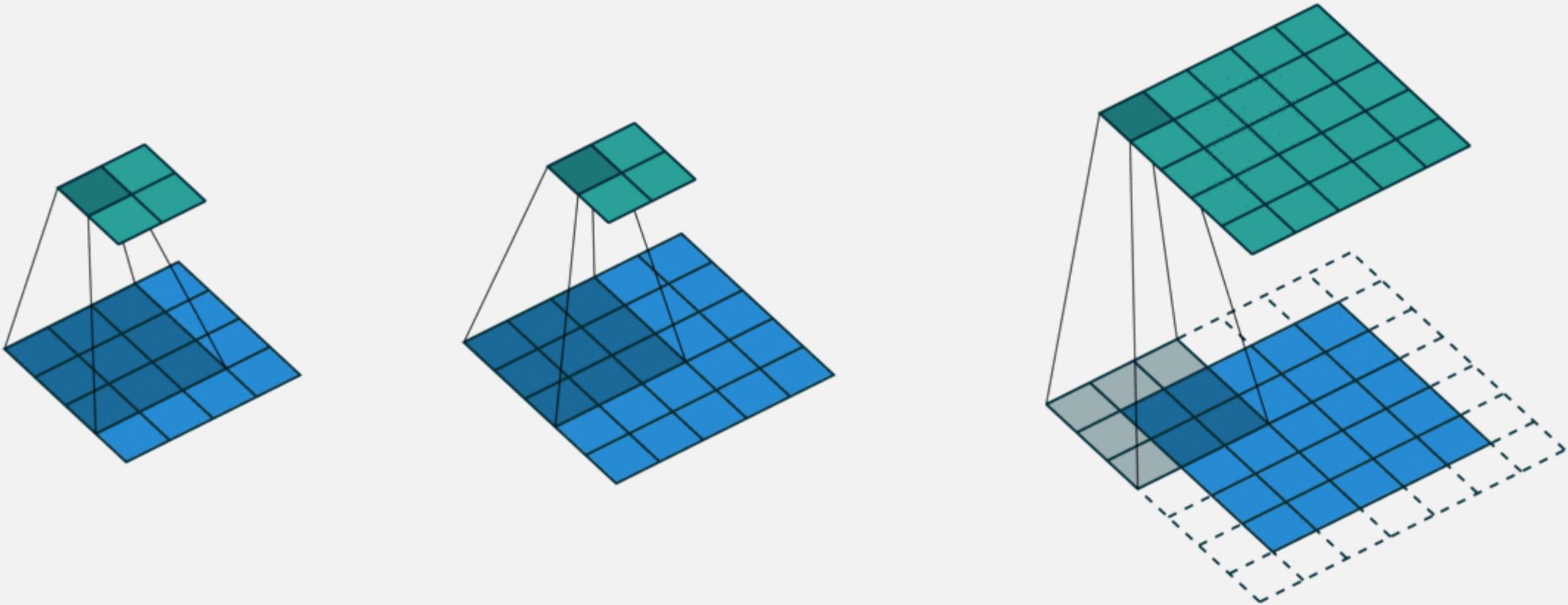
=

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

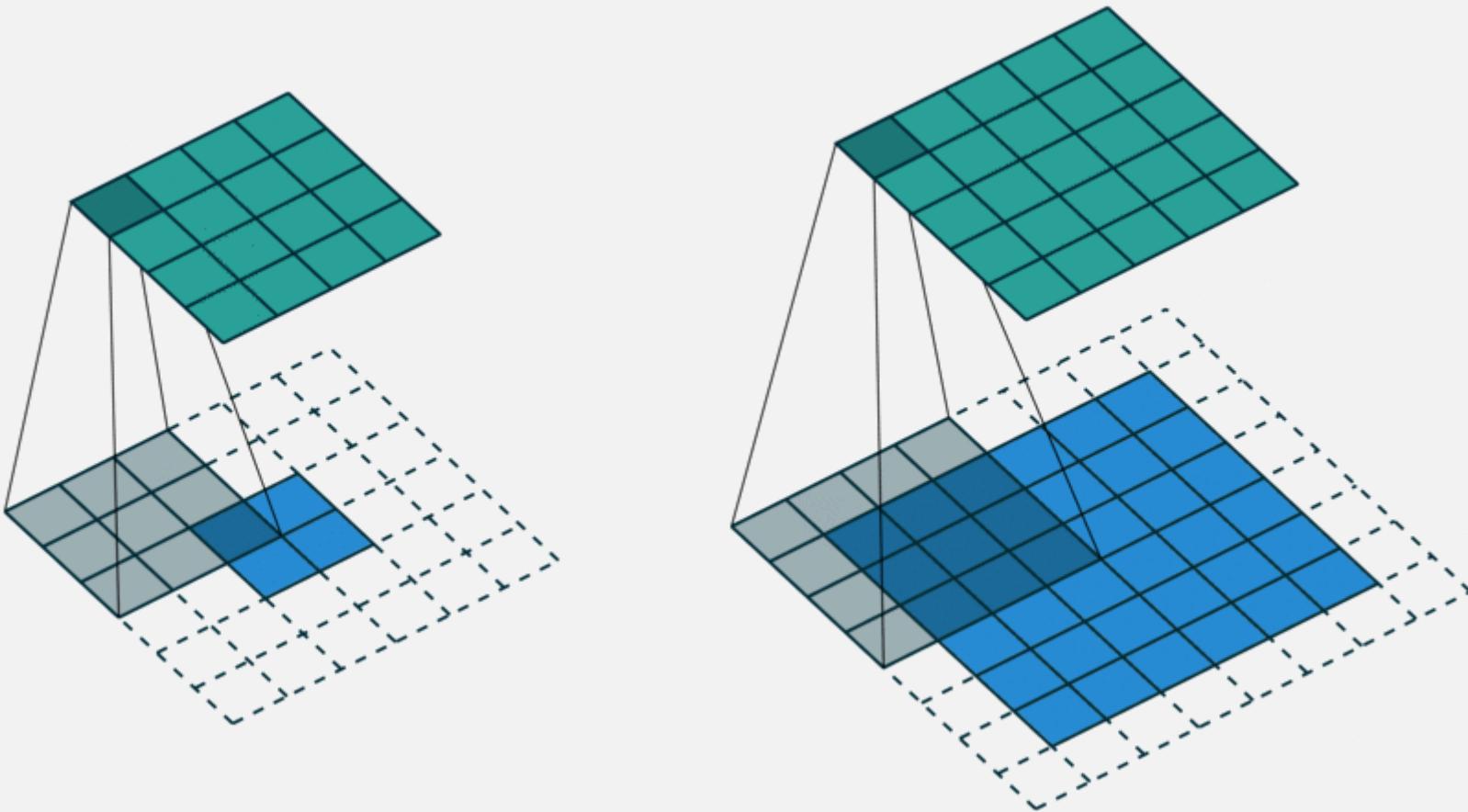
0.33	-0.55	-0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

STRIDES AND PADDING



DECONVOLUTION – TRANSPOSED CONVOLUTION



POOLING

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

POOLING

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

AVERAGE POOLING – STRIDE 1

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.39	0.22	0.17	0.28	0.11	0.06
0.22	0.45	0.22	0.00	0.00	0.11
0.17	0.22	0.22	0.00	0.00	0.28
0.28	0.00	0.00	0.22	0.22	0.17
0.11	0.00	0.00	0.22	0.45	0.22
0.06	0.11	0.28	0.17	0.22	0.39

MAX POOLING – STRIDE 1

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

1.00	1.00	0.33	0.55	0.55	0.33
1.00	1.00	1.00	0.33	0.11	0.55
0.33	1.00	1.00	0.55	0.33	0.55
0.55	0.33	0.55	1.00	1.00	0.33
0.55	0.11	0.33	1.00	1.00	1.00
0.33	0.55	0.55	0.33	1.00	1.00

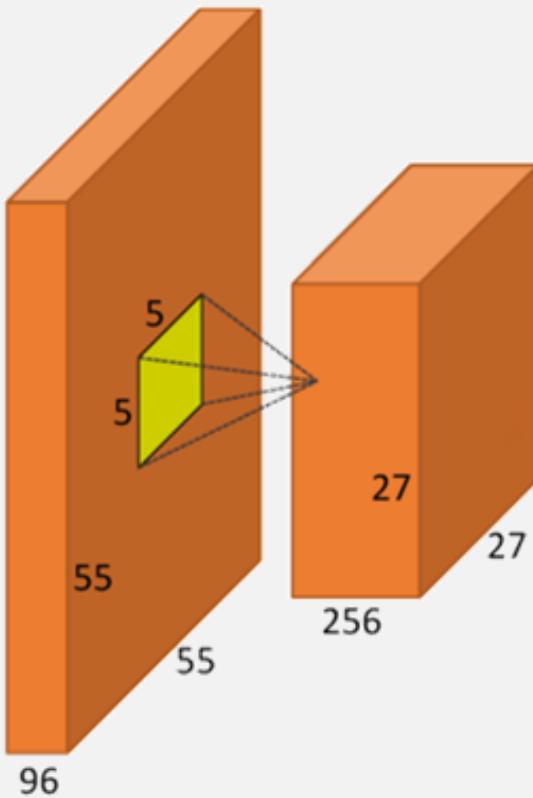
AVERAGE POOLING – STRIDE 2

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

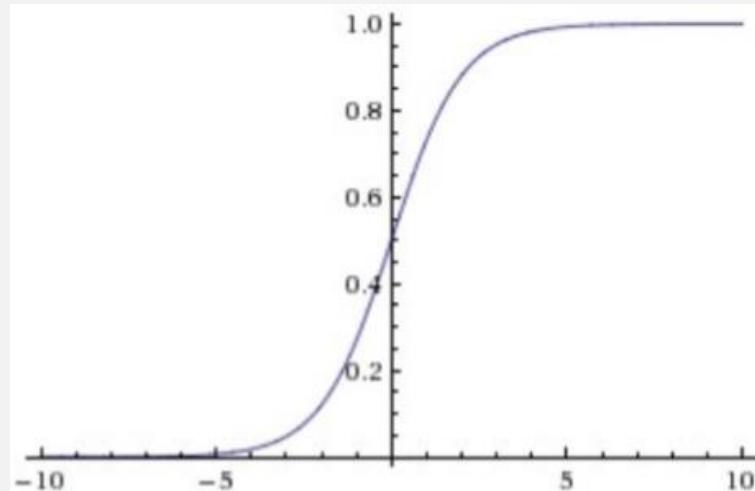
WHY POOLING?

ACTIVATION FUNCTIONS?



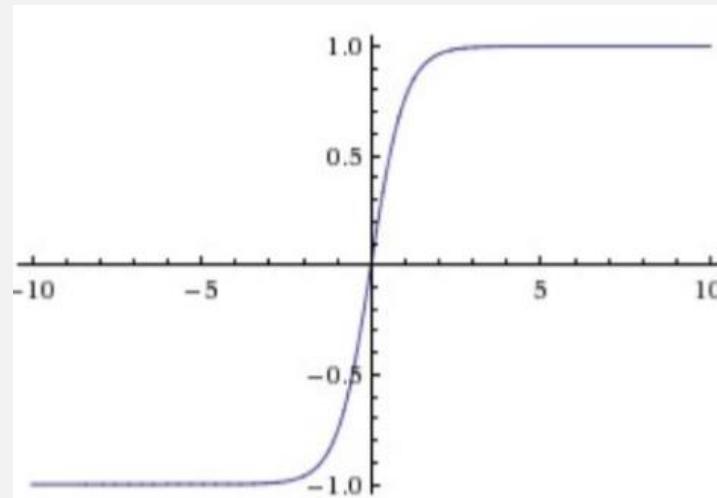
ACTIVATION FUNCTIONS?

LOGISTIC FUNCTION (“sigmoid”):



- Simple
- Drawbacks:
 - Saturates
 - Non-centered in zero
- We don't use it anymore, but we thank her for the services provided to the cause :)

ACTIVATION FUNCTIONS?

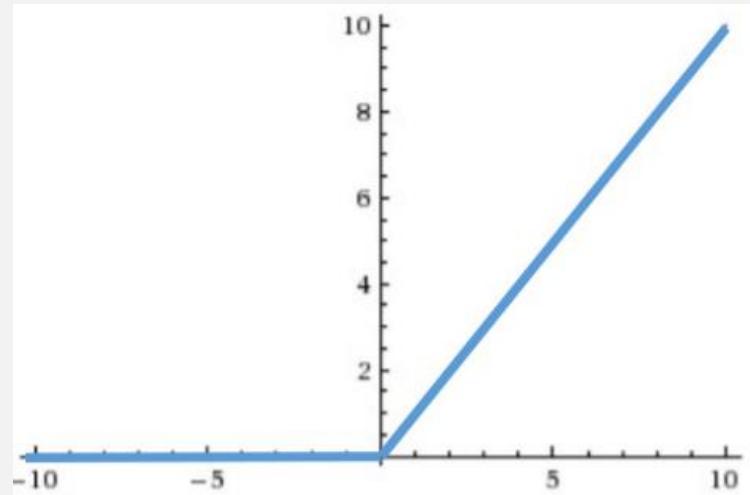


HYPERBOLIC TANGENT (\tanh):

- Solves the issue with not being centered in zero
- Still ‘killing gradients’ due to saturating behaviour
- It is always better than the logistic function

ACTIVATION FUNCTIONS?

RECTIFIED LINEAR UNITS (ReLU):

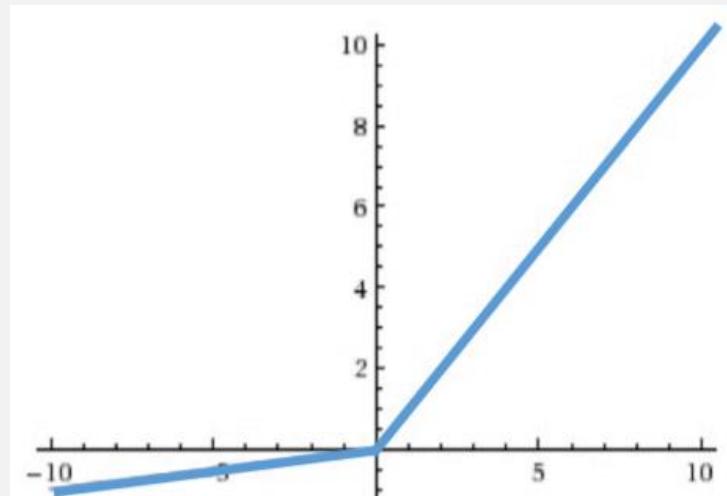


- Optimization for the SGD convergence
- More simple from a computational point of view
- Fragile

ACTIVATION FUNCTIONS?

LEAKY ReLU:

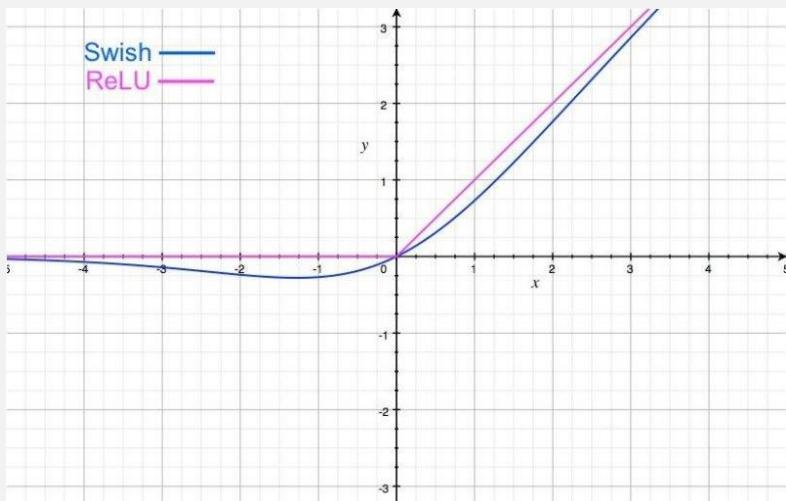
- More stable than traditional ReLU
- Particular case of MaxOut



ACTIVATION FUNCTIONS?

SWISH:

- Self-Gating activation function
- Better performance than ReLU



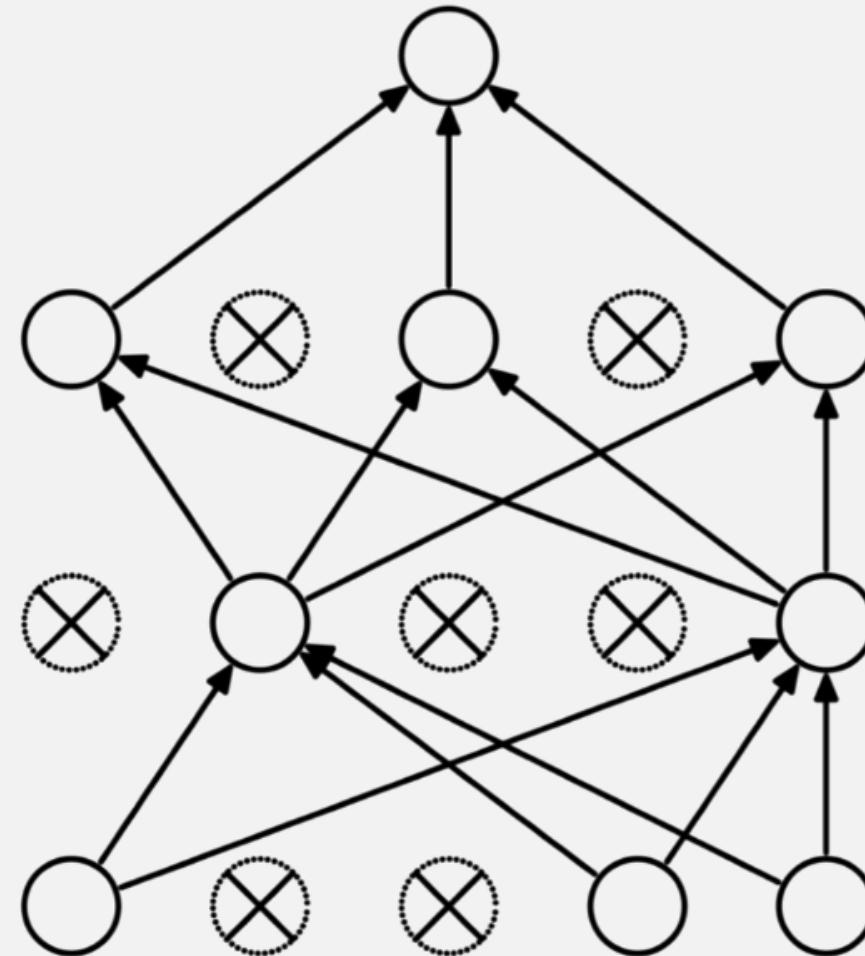
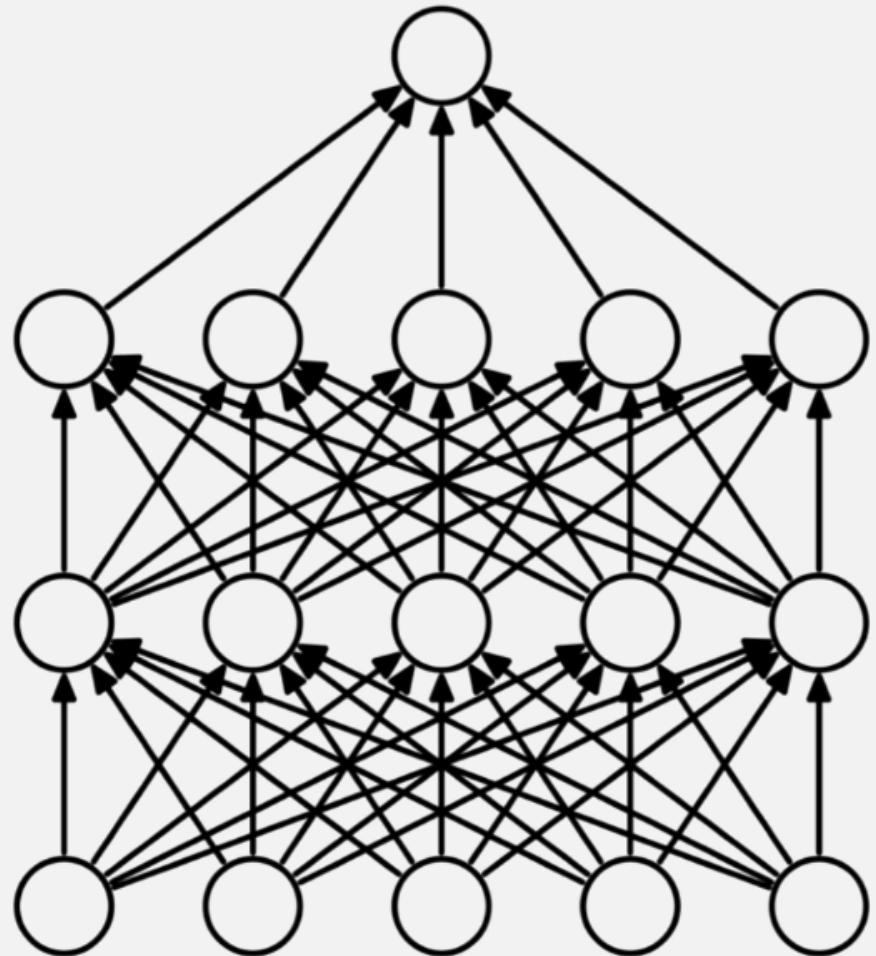
RECTIFIED LINEAR UNITS (ReLU)

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

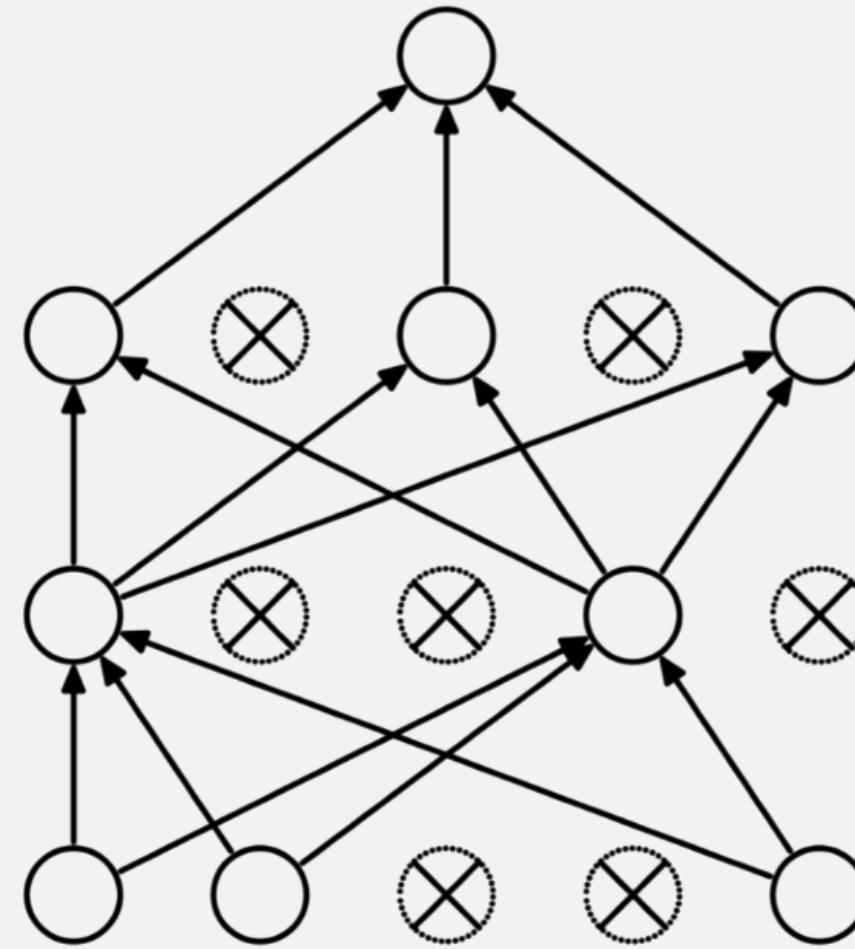
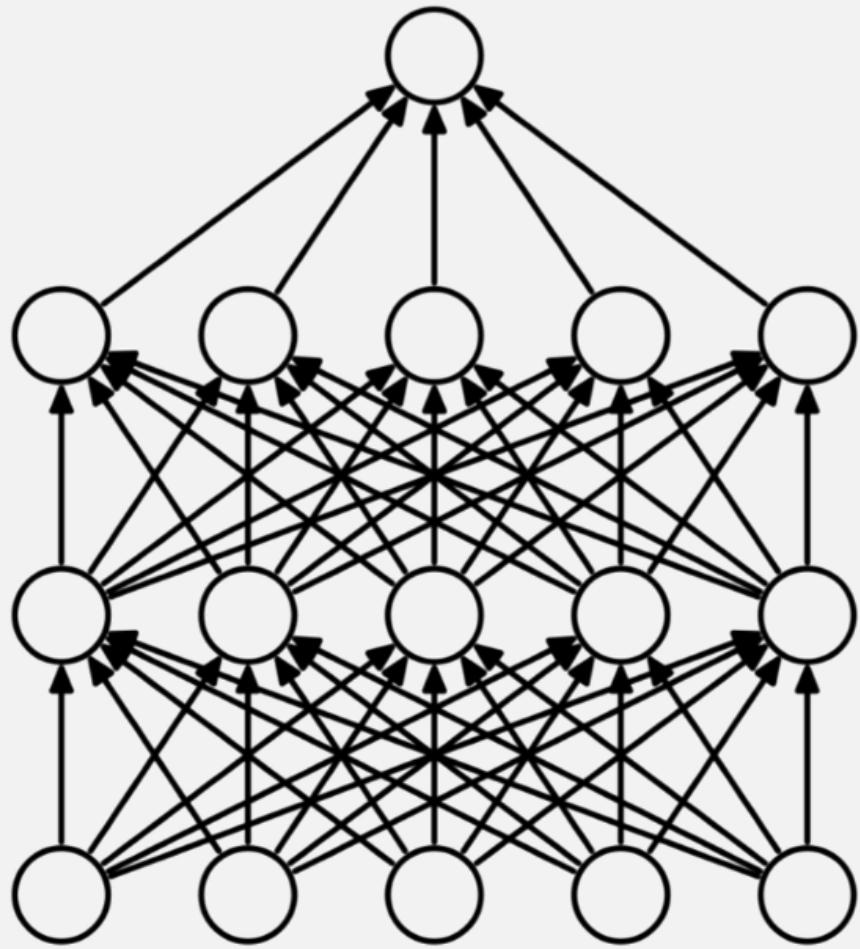


0.77	0.00	0.11	0.33	0.55	0.00	0.33
0.00	1.00	0.00	0.33	0.00	0.11	0.00
0.11	0.00	1.00	0.00	0.11	0.00	0.55
0.33	0.33	0.00	0.55	0.00	0.33	0.33
0.55	0.00	0.11	0.00	1.00	0.00	0.11
0.00	0.11	0.00	0.33	0.00	1.00	0.00
0.33	0.00	0.55	0.33	0.11	0.00	0.77

DROPOUT



DROPOUT



WHAT IS MISSING?

Filter 1 Filter 2 Filter 3

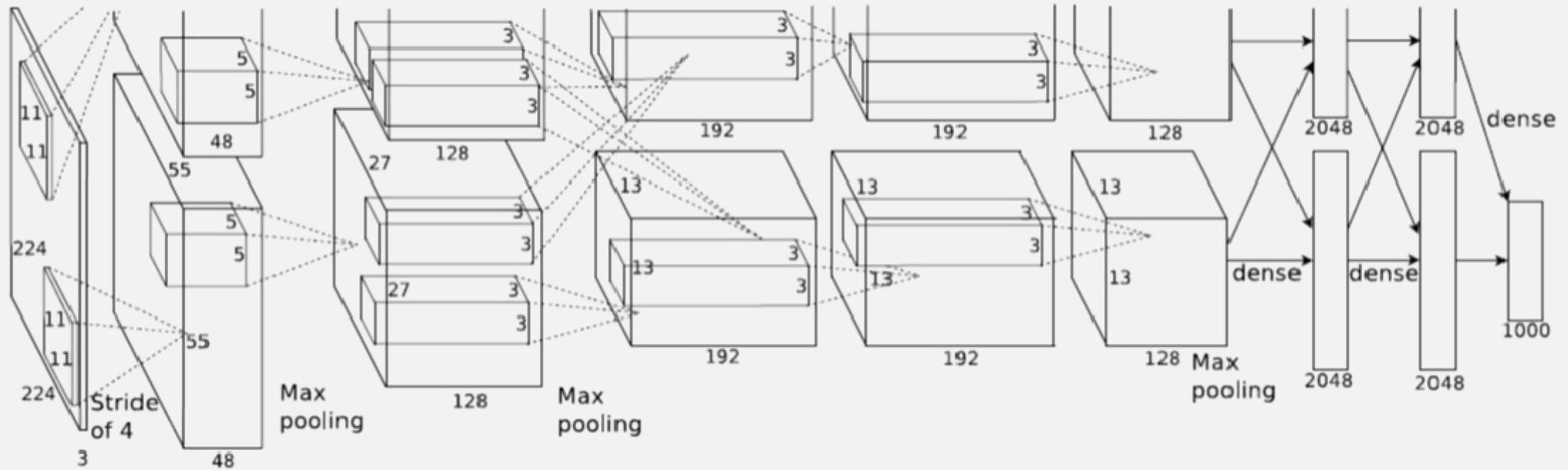
$$\begin{matrix} 1 & -1 & -1 \\ -1 & \boxed{1} & -1 \\ -1 & -1 & 1 \end{matrix}$$

$$\begin{matrix} -1 & -1 & 1 \\ -1 & \boxed{1} & -1 \\ 1 & -1 & -1 \end{matrix}$$

$$\begin{matrix} 1 & -1 & 1 \\ -1 & \boxed{1} & -1 \\ 1 & -1 & 1 \end{matrix}$$

How have these filters been
selected/created?

AlexNet



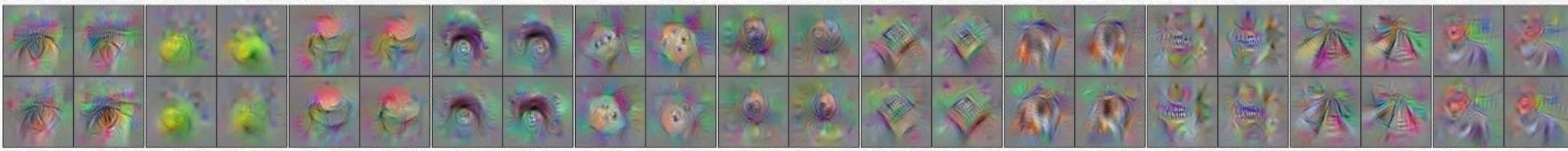
HOW A CNN SEES

Layer 2



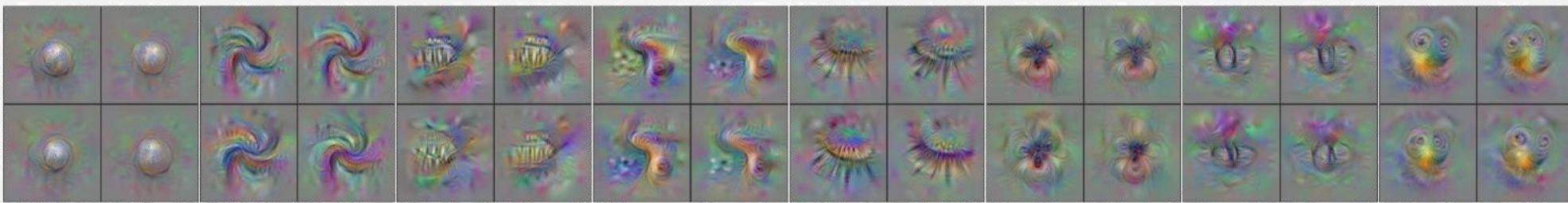
HOW A CNN SEES

Layer 3



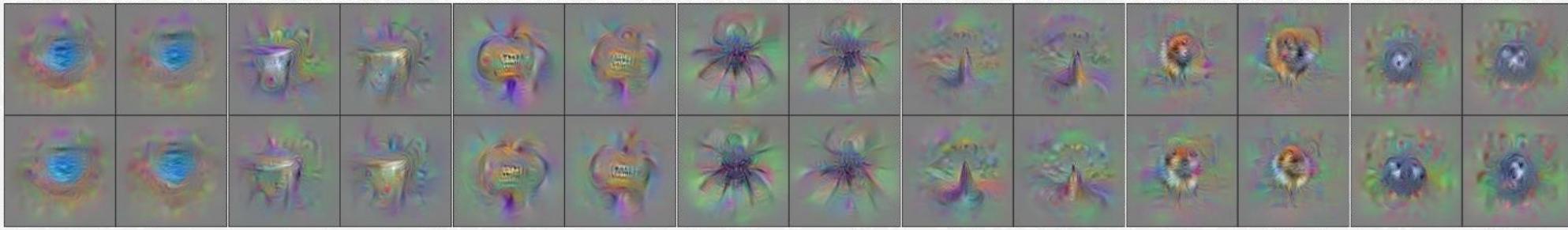
HOW A CNN SEES

Layer 4



HOW A CNN SEES

Layer 5



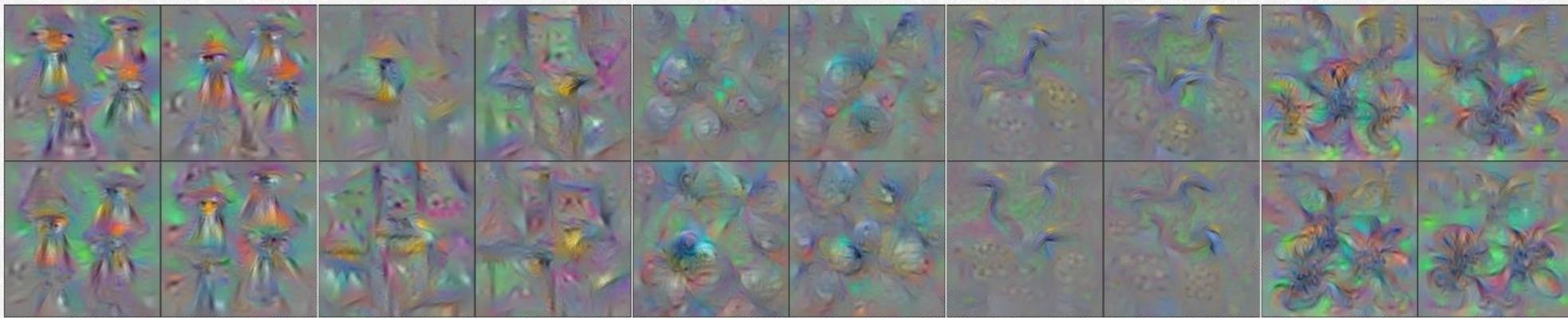
HOW A CNN SEES

Layer 6



HOW A CNN SEES

Layer 7



HOW A CNN SEES

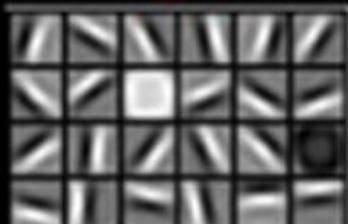
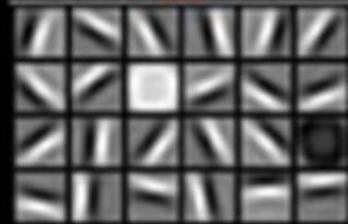
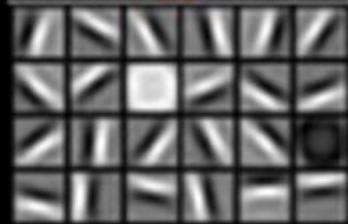
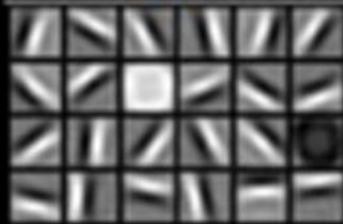
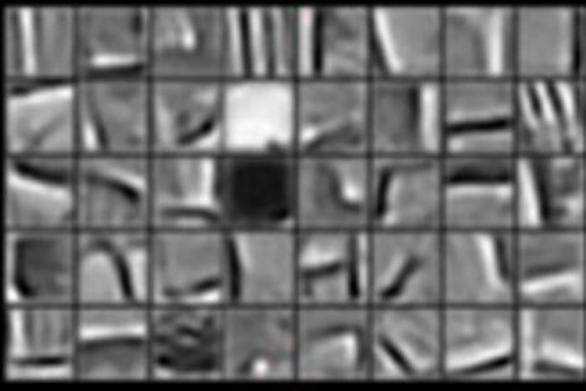
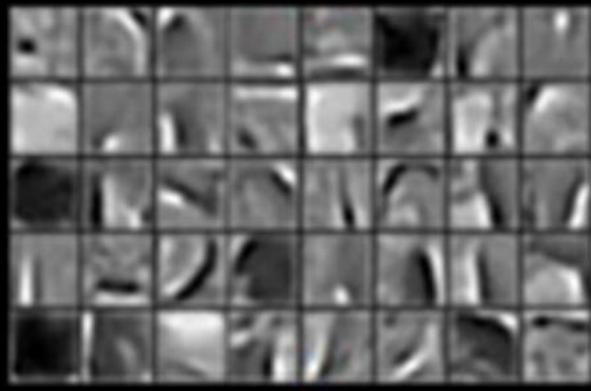
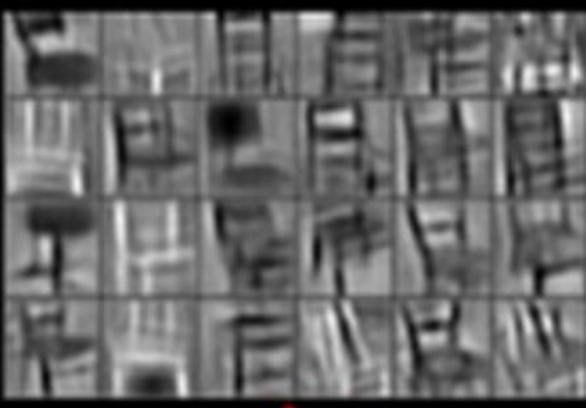


Faces

Cars

Elephants

Chairs

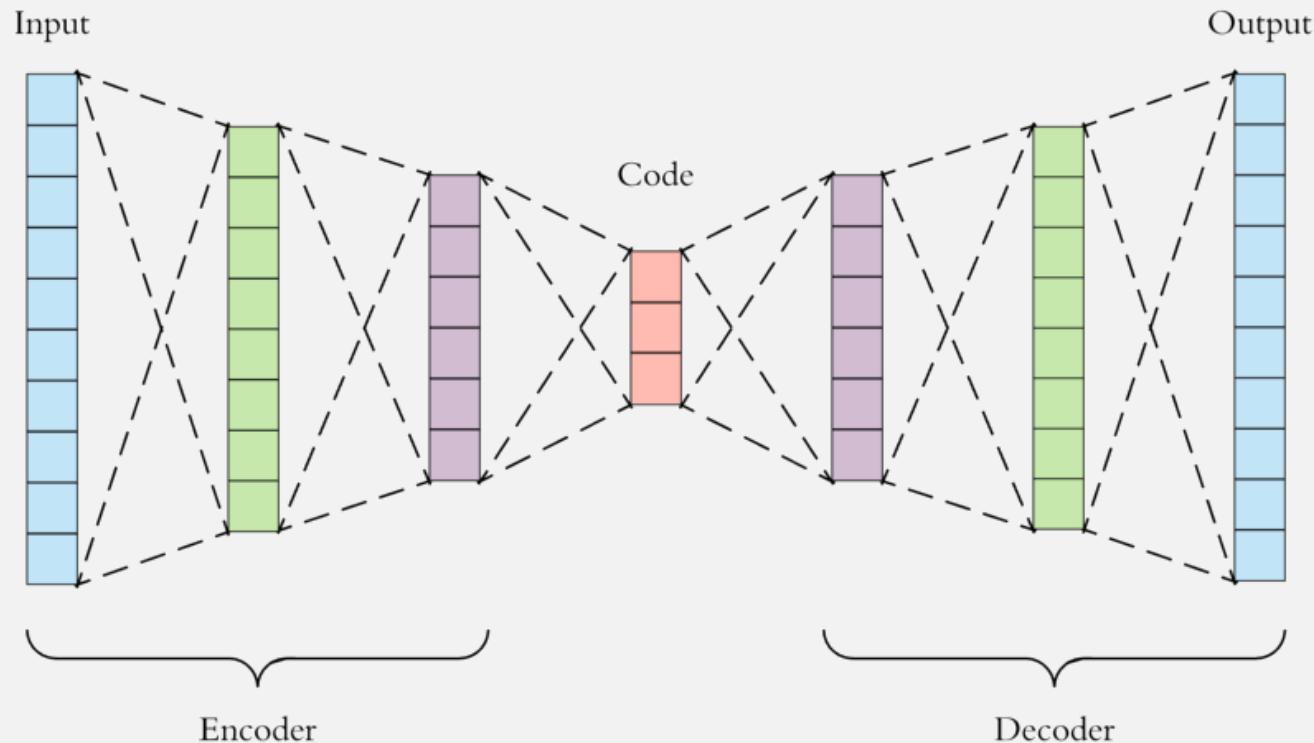


LAB 5: Convolutional Networks

Autoencoders

BASIC AUTOENCODER

$$\hat{x} = h_{W,b} \approx x$$

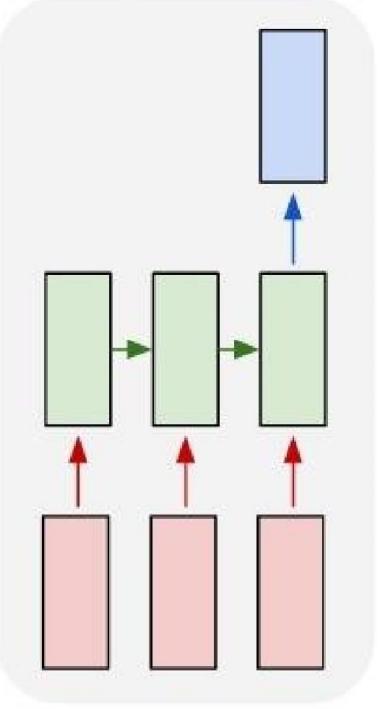


LAB 6: Autoencoder

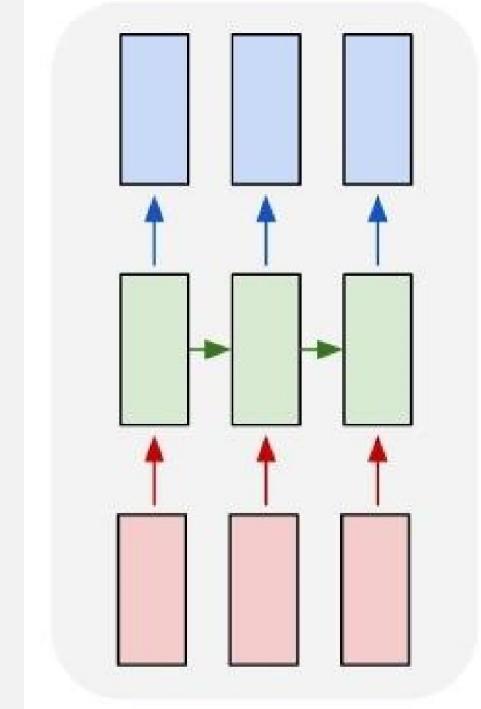
Recurrent Neural Networks

HANDLING SEQUENCES

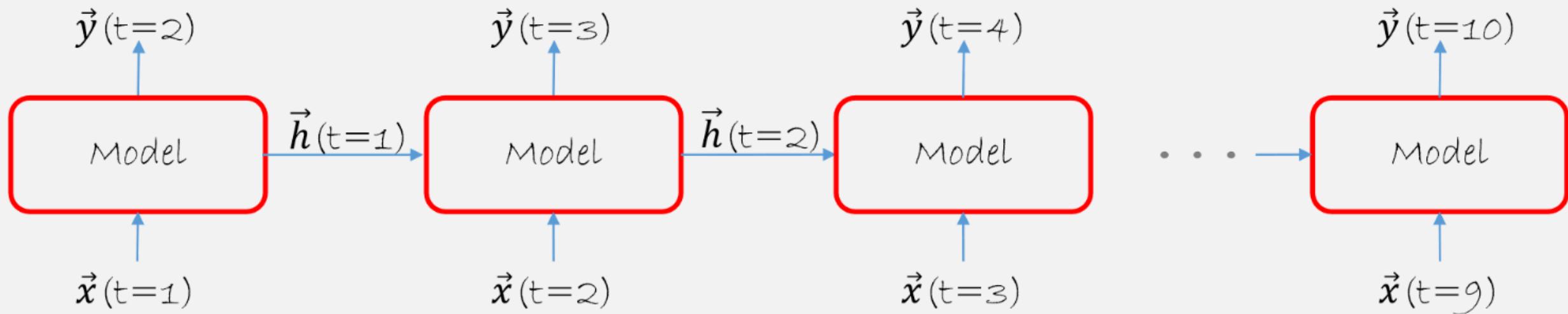
many to one



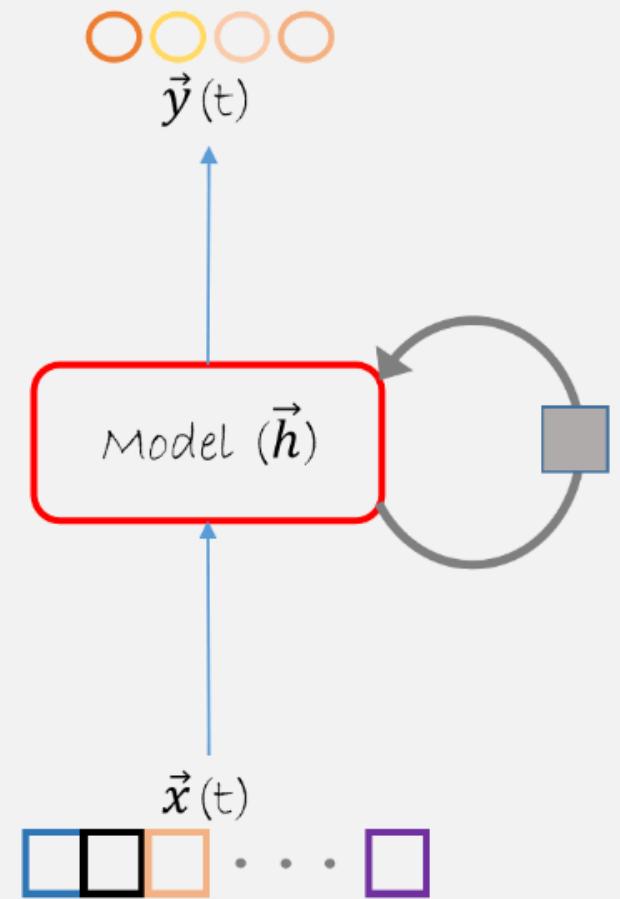
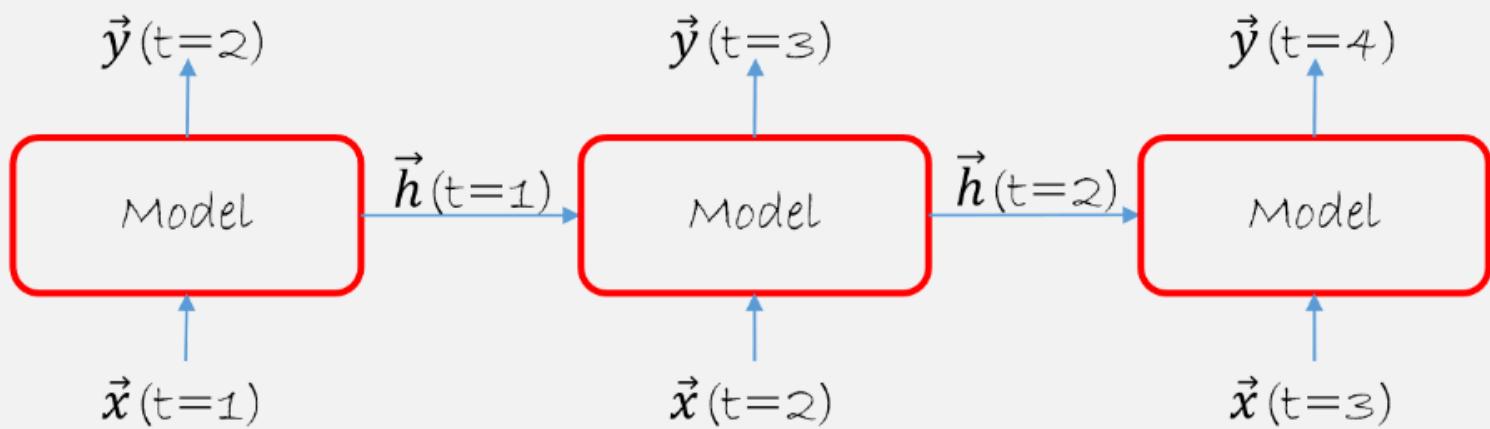
many to many

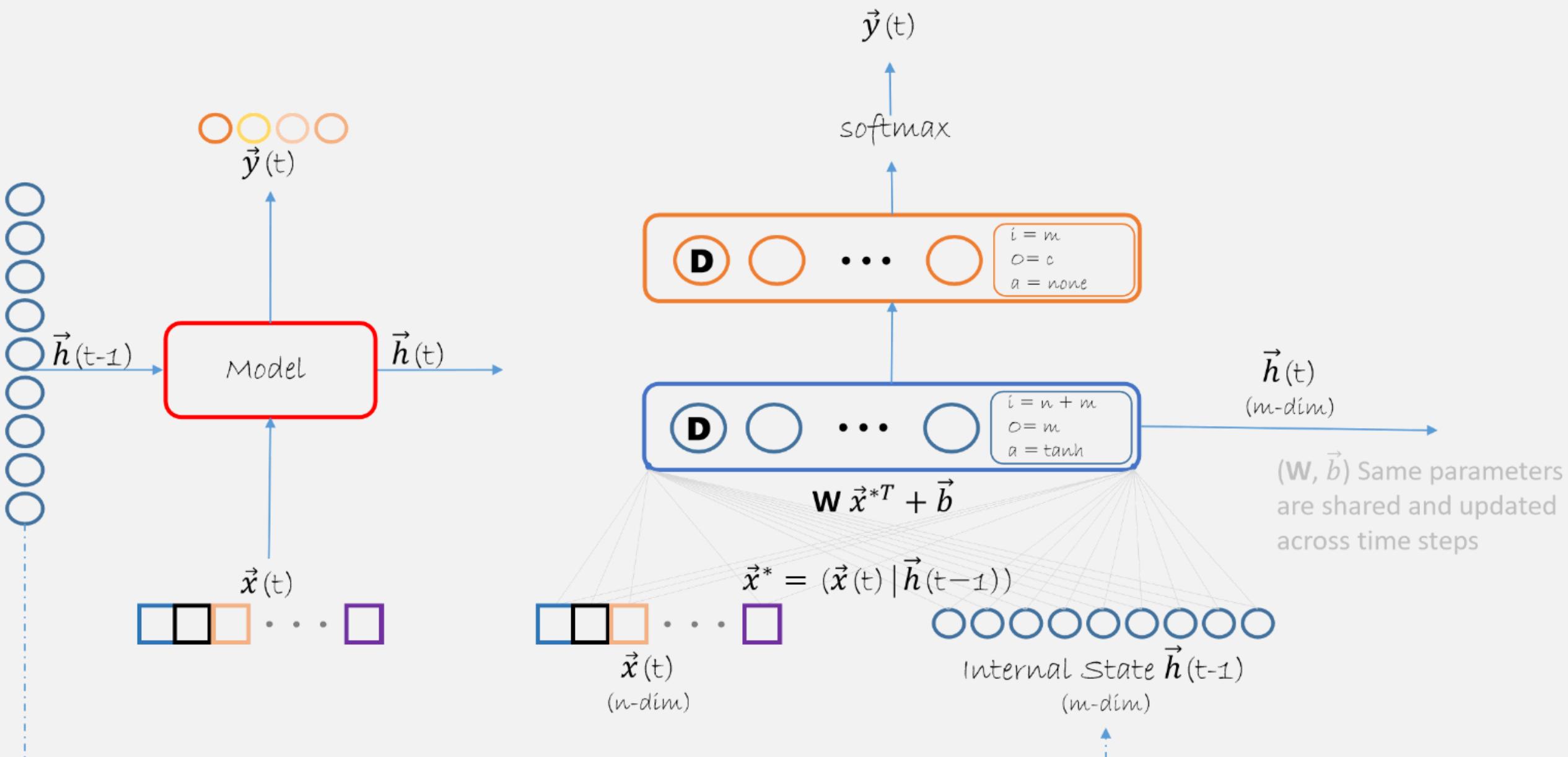


TRACKING THE HISTORY

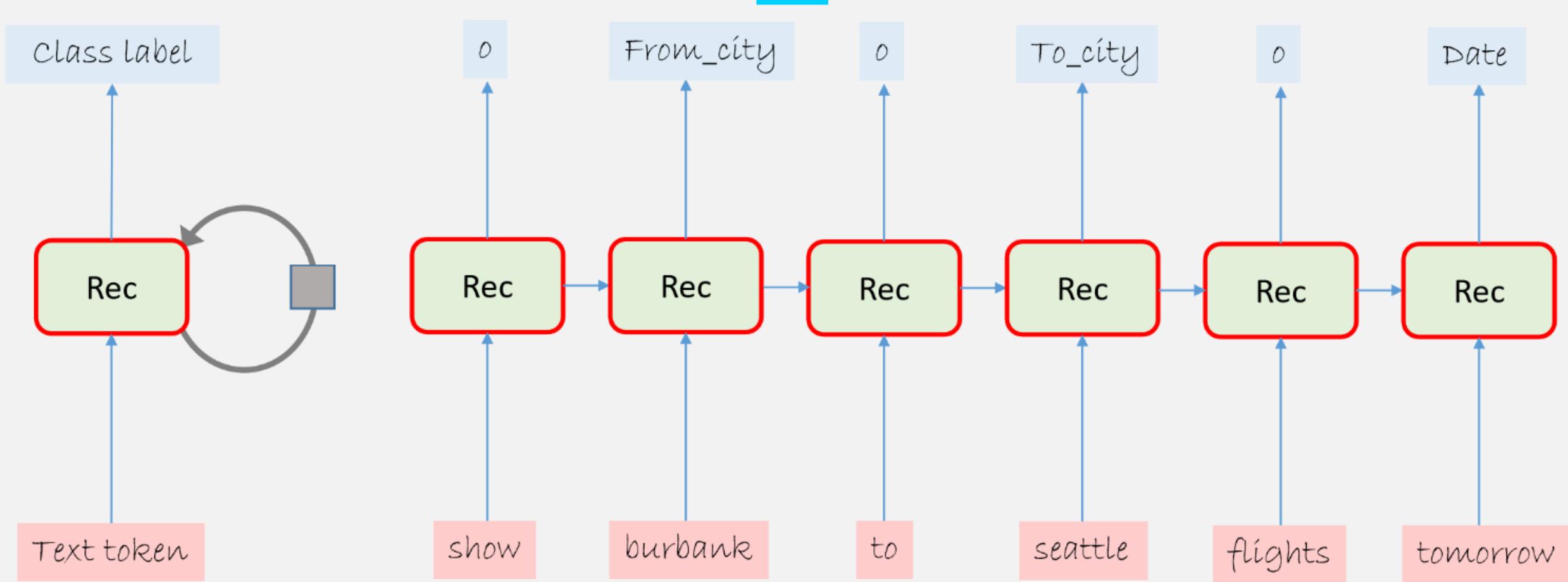


RECURRENCE

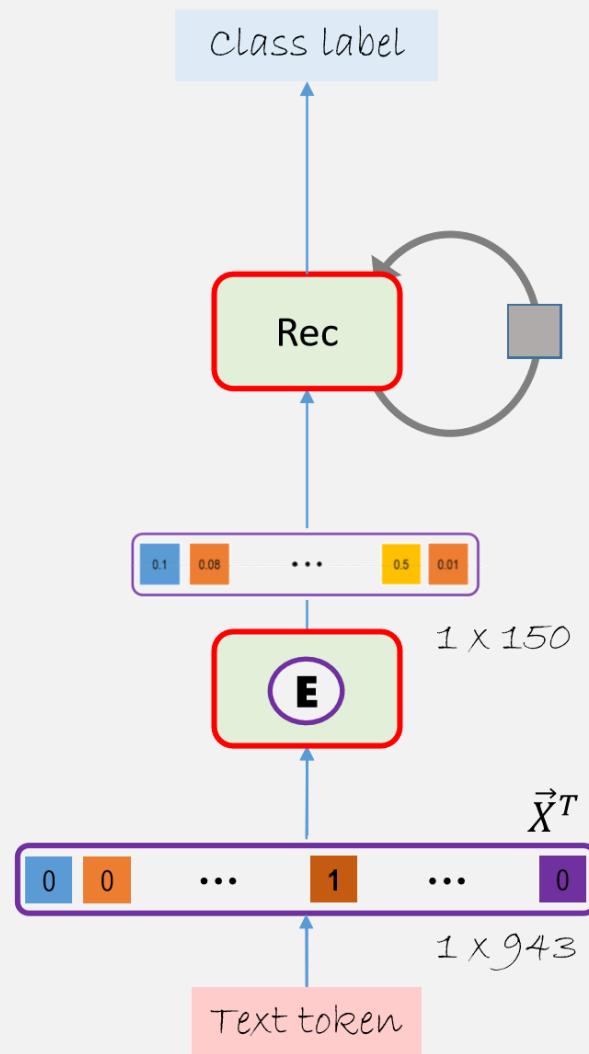


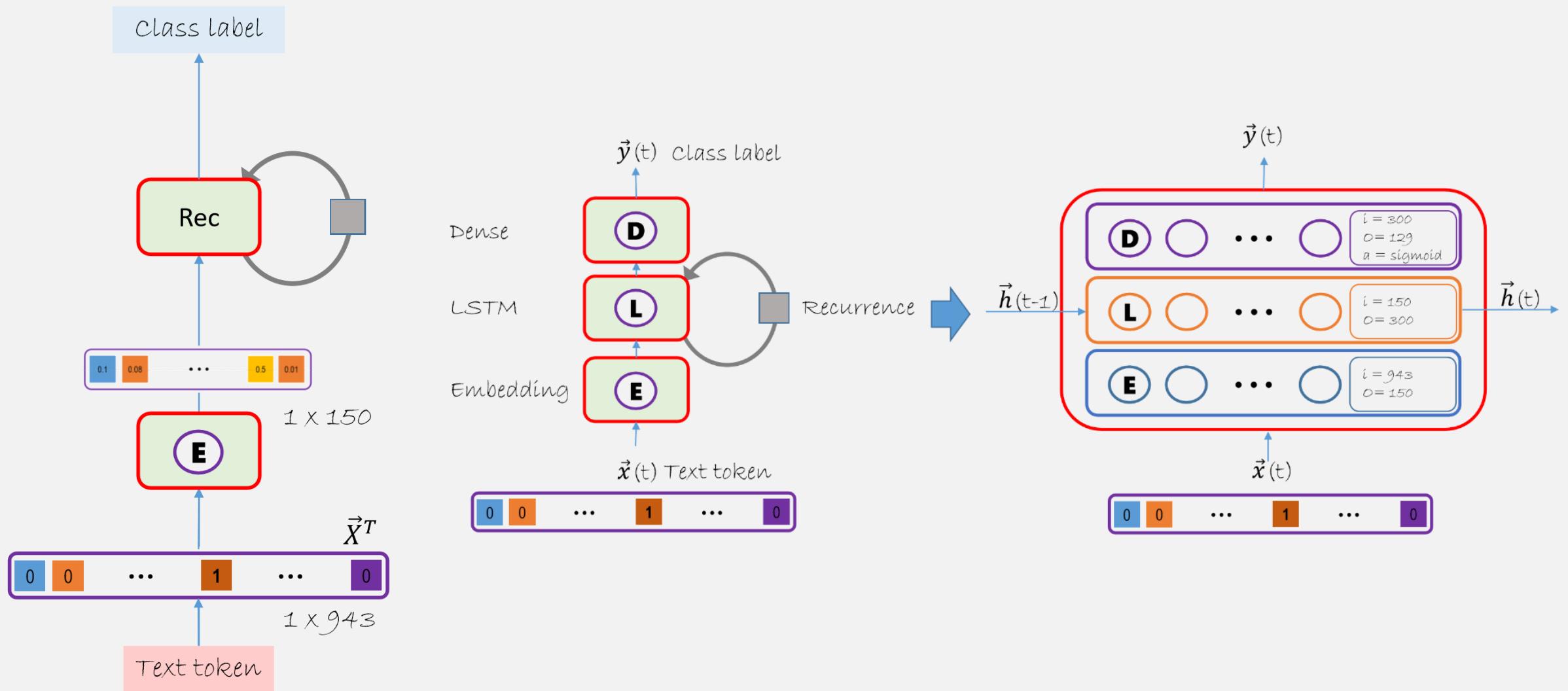


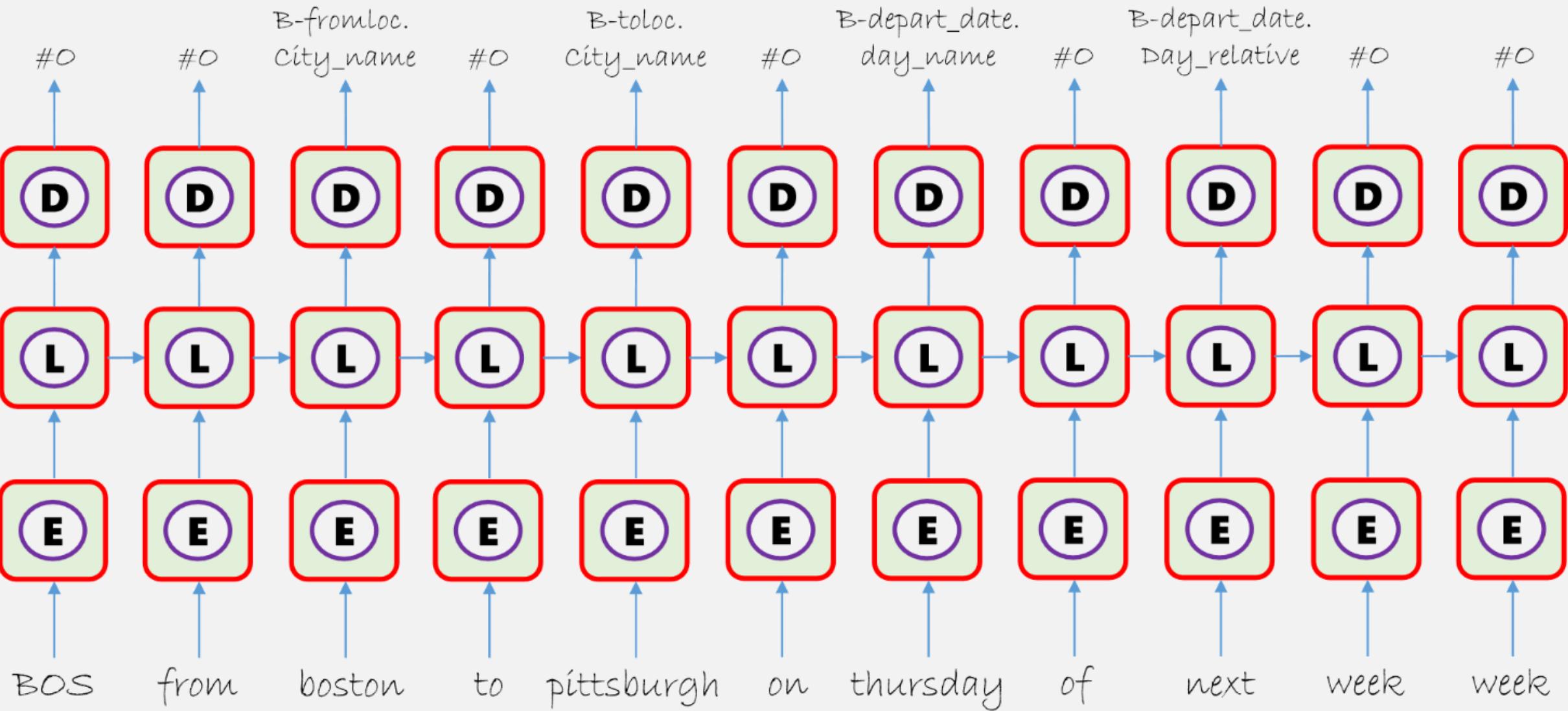
LONG SHORT TERM MEMORY



Sequence Id	Input Word (sample)	Word Index (in vocabulary) S0	Word Label	Label Index (S2)
19	# BOS	178:1	# O	128:1
19	# please	688:1	# O	128:1
19	# give	449:1	# O	128:1
19	# me	581:1	# O	128:1
19	# the	827:1	# O	128:1
19	# flights	429:1	# O	128:1
19	# from	444:1	# O	128:1
19	# boston	266:1	# B-fromloc.city_name	48:1
19	# to	851:1	# O	128:1
19	# pittsburgh	682:1	# B-toloc.city_name	78:1
19	# on	654:1	# O	128:1
19	# thursday	845:1	# B-depart_date.day_name	26:1
19	# of	646:1	# O	128:1
19	# next	621:1	# B-depart_date.date_relative	25:1
19	# week	910:1	# O	128:1
19	# EOS	179:1	# O	128:1







Some Advanced Ideas

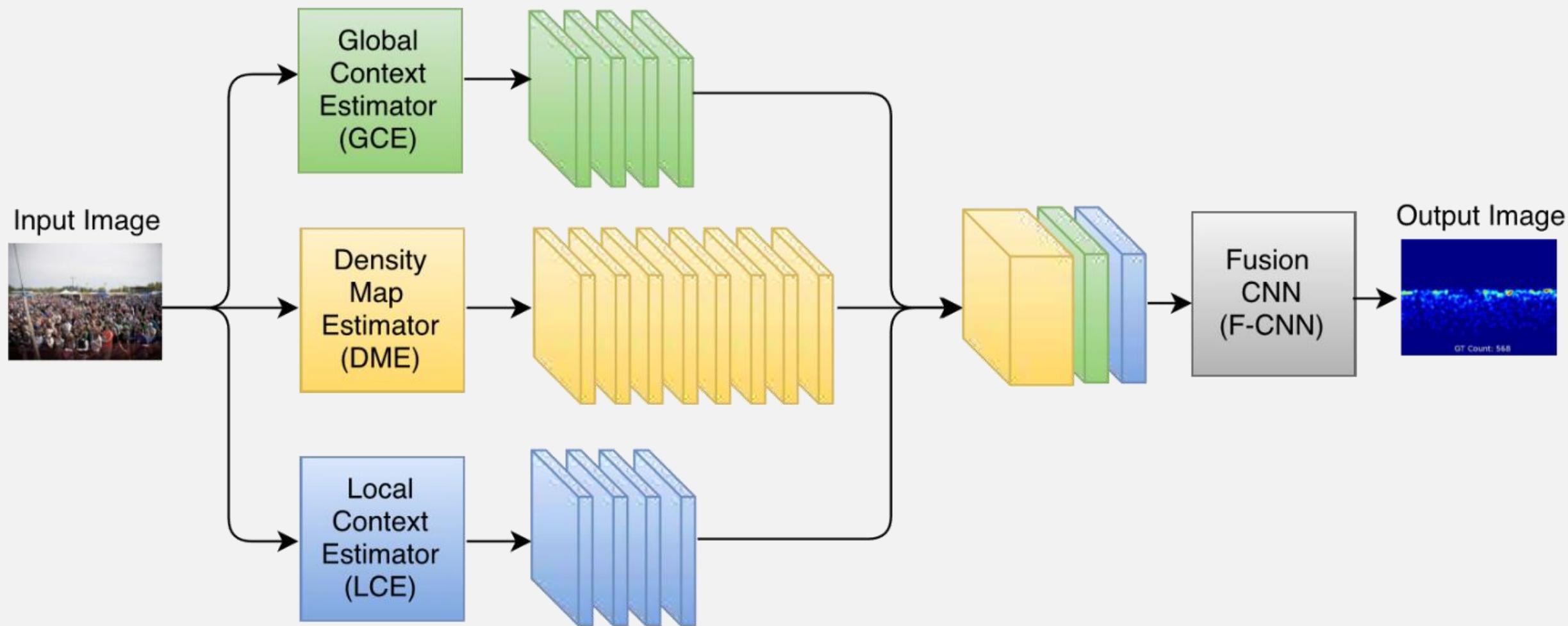
Counting people in a multitude

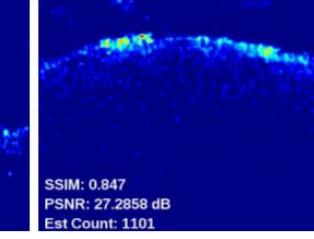
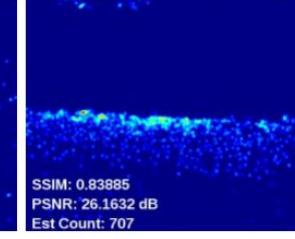
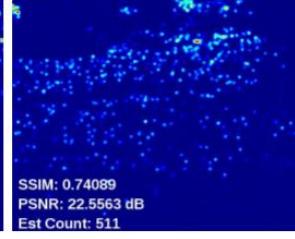
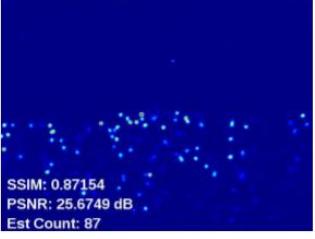
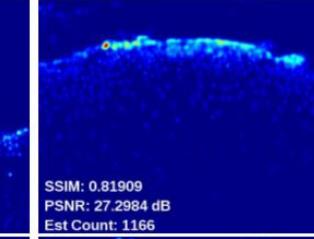
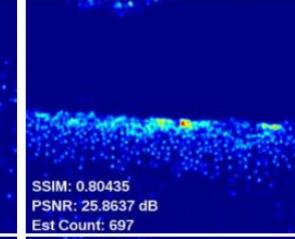
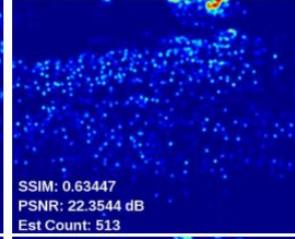
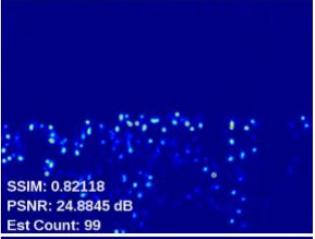
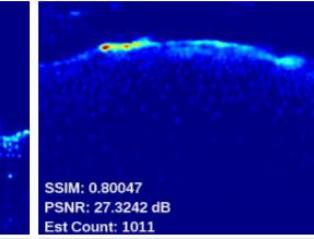
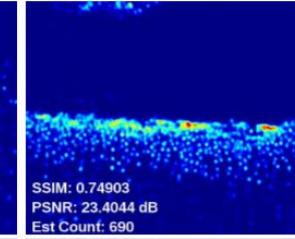
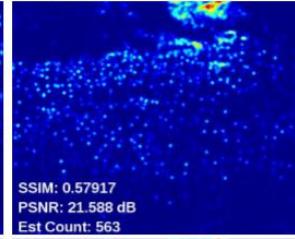
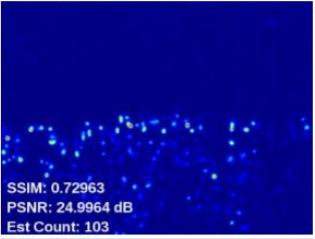
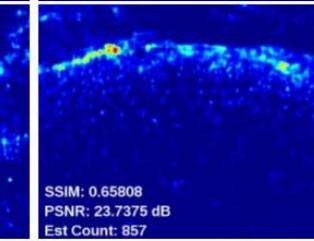
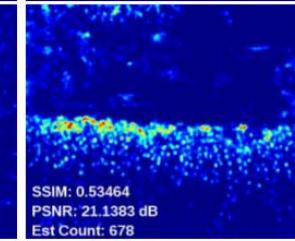
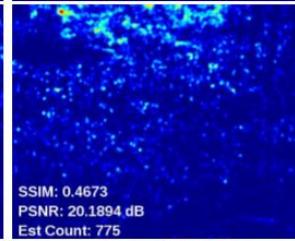
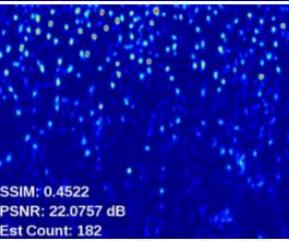
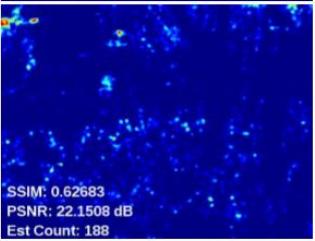
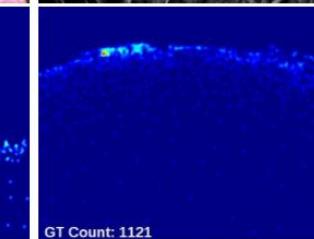
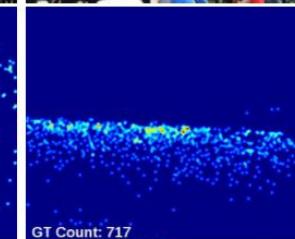
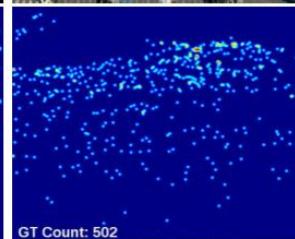
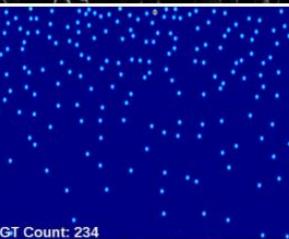
CONTEXTUAL PYRAMID CNN (CP-CNN)

A new method named Contextual Pyramid CNN (CP-CNN) is proposed here to generate density maps and influx estimations, by explicitly incorporating global and local context information. Composed of four modules: Global Context Estimator (GCE), Local Context Estimator (LCE), Density Map Estimator (DME) and a Fusion-CNN (F-CNN) convolutional network.

Vishwanath A. Sindagi, Vishal M. Patel; The IEEE International Conference on Computer Vision (ICCV), 2017, pp. 1861-1870









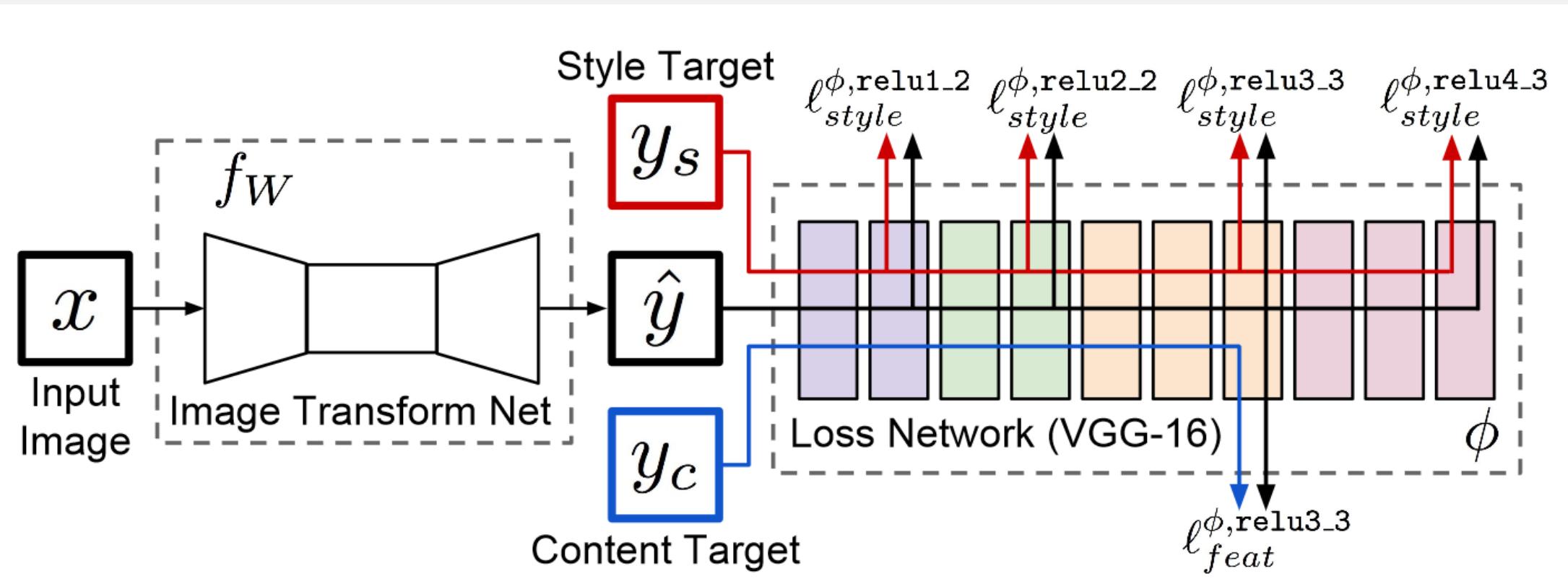
Transferring style between images

CONVOLUTIONAL NEURAL NETWORKS

By using a perceptual loss functions based on high-level features extracted from pretrained networks, networks for image transformation tasks can be trained, and by fine tuning the loss function different features can be kept for the source image and the style image.

Justin Johnson, Alexandre Alahi, Li Fei-Fei; Perceptual Losses for Real-Time Style Transfer and Super-Resolution, 2016









Using GANs to drive design decisions

GENERATIVE ADVERSARIAL NETWORKS

By taking advantage of Generational Adversarial Networks, synthetic images based on the training data can be generated. Including an external array of features, the generated images can be tailored to a specific set of requirements.

Jaime Deverall, Jiwoo Lee, Miguel Ayala; Using Generative Adversarial Networks to Design Shoes

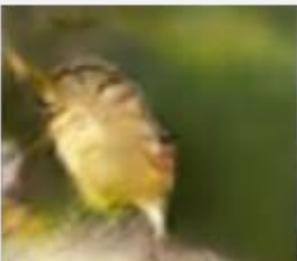


Text
description

This bird is blue with white and has a very short beak



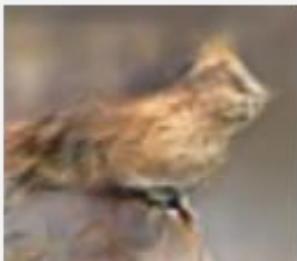
This bird has wings that are brown and has a yellow belly



A white bird with a black crown and yellow beak



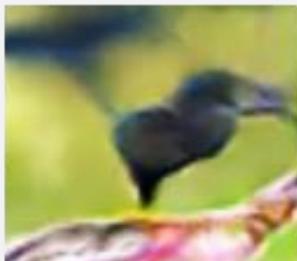
This bird is white, black, and brown in color, with a brown beak



The bird has small beak, with reddish brown crown and gray belly



This is a small, black bird with a white breast and white on the wingbars.



This bird is white black and yellow in color, with a short black beak

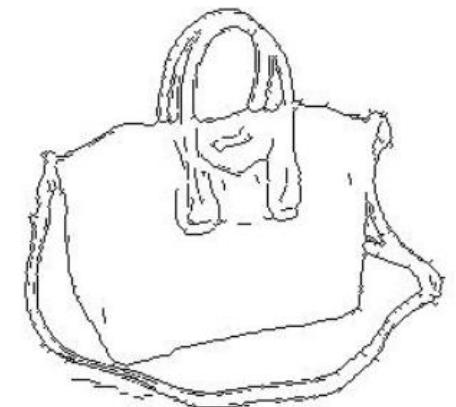
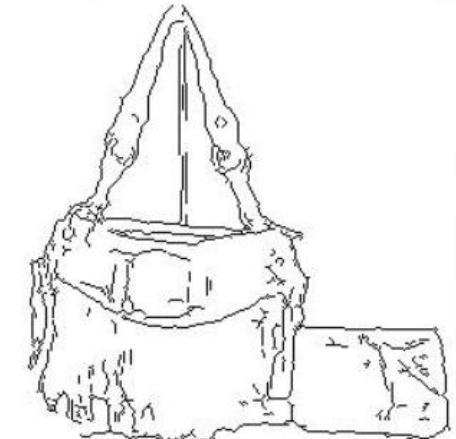
Input



Ground truth



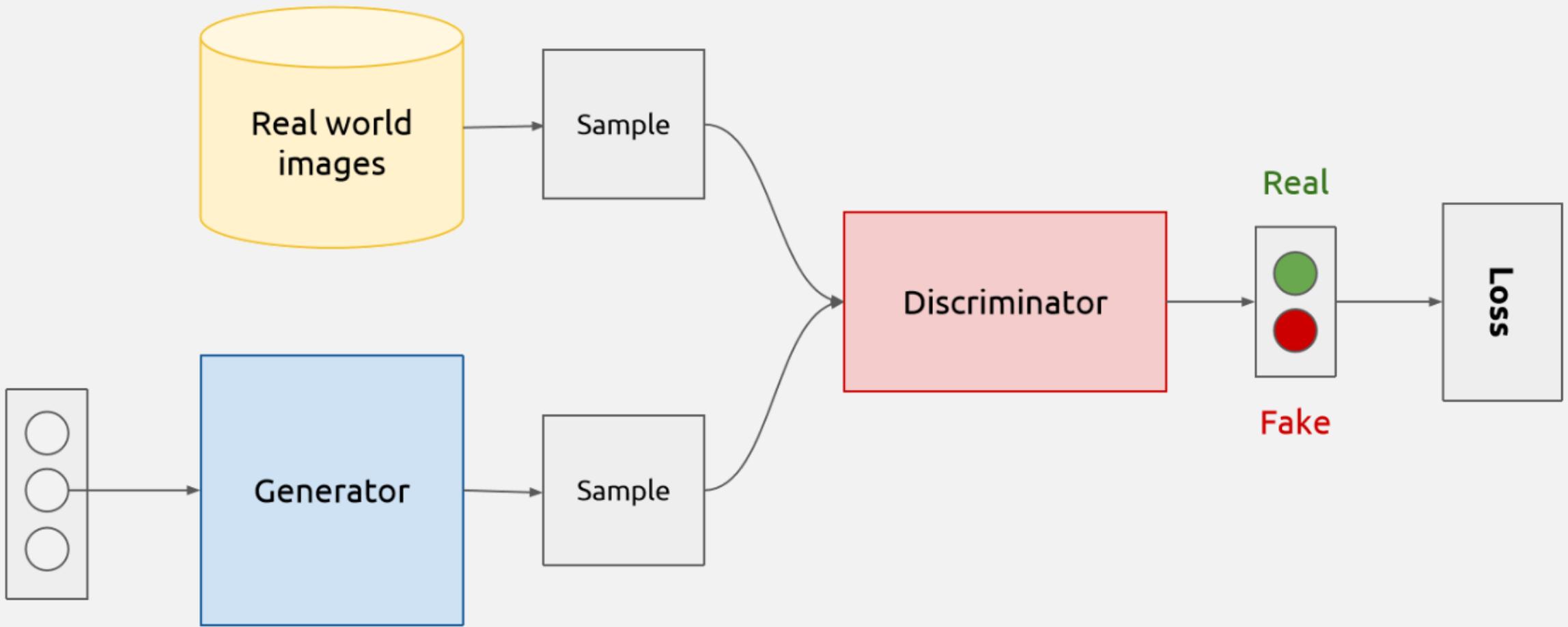
Output





plain concepts

Latent random variable



DEEP LEARNING - MOVING FAST!

- Swish Activation Function
- Capsule Networks
- ...

SOME INTERESTING PAPERS

“ImageNet Classification with Deep Convolutional Neural Networks”

[Krizhevsky, Sutskever, Hinton]

<http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf>

“Visualizing and Understanding Convolutional Networks”

[Zeiler, Fergus]

arXiv: 1311.2901v3

“Optimization Methods for Large-Scale Machine Learning”

[Bottou, Curtis, Nocedal]

arXiv: 1606.04838v2

SOME INTERESTING PAPERS

“Swish: A Self-Gated Activation Function”

[Ramachandran, Zoph, Quoc V.]

arXiv: 1710.05941v2

“Dynamic Routing Between Capsules”

[Sabour, Frosst, Hinton]

arXiv: 1710.09829