

# Introduction to Deep Learning with PyTorch

Jian Tao

jtao@tamu.edu

HPRC Short Course

11/13/2020



Texas A&M Engineering  
Experiment Station

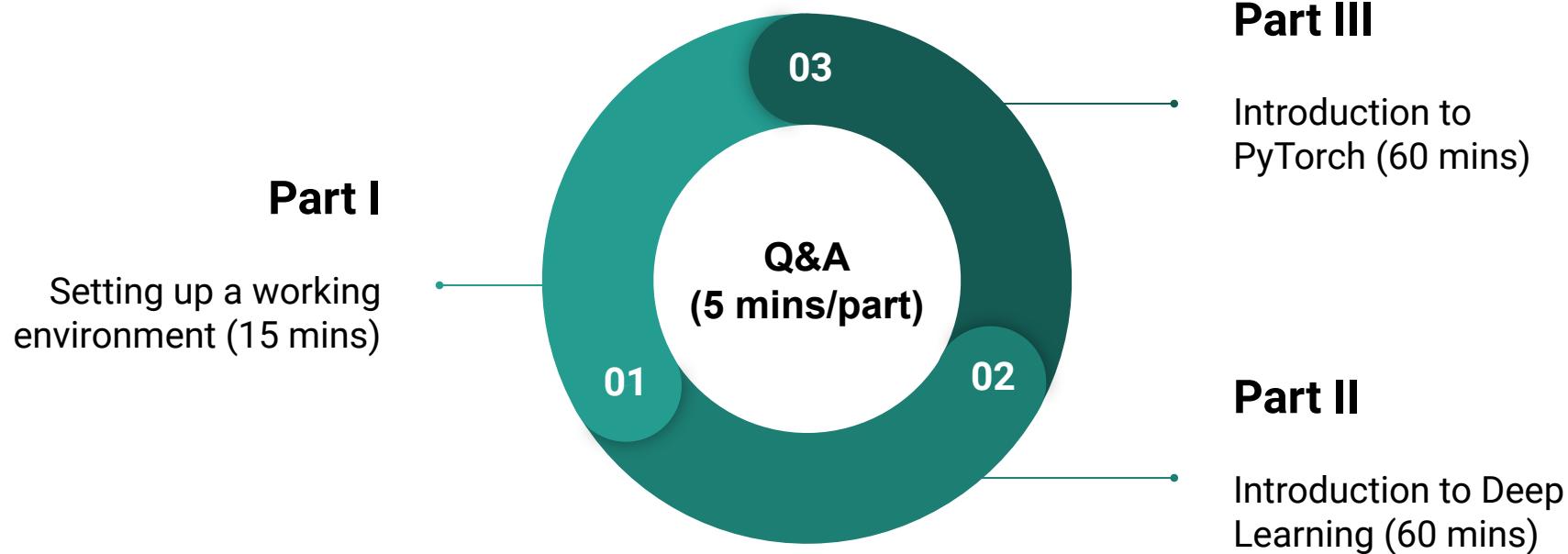


High Performance  
Research Computing  
DIVISION OF RESEARCH

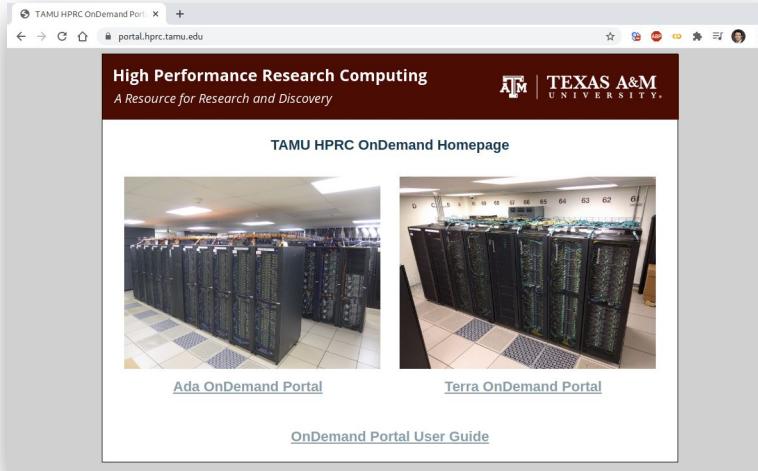


TEXAS A&M  
Institute of  
Data Science

# Introduction to Deep Learning with PyTorch



# Part I. Working Environment



HPRC Portal

# Login HPRC Portal (Terra)

TAMU HPRC OnDemand Port... x +

portal.hprc.tamu.edu

High Performance Research Computing  
*A Resource for Research and Discovery*

TEXAS A&M  
UNIVERSITY.

TAMU HPRC OnDemand Homepage



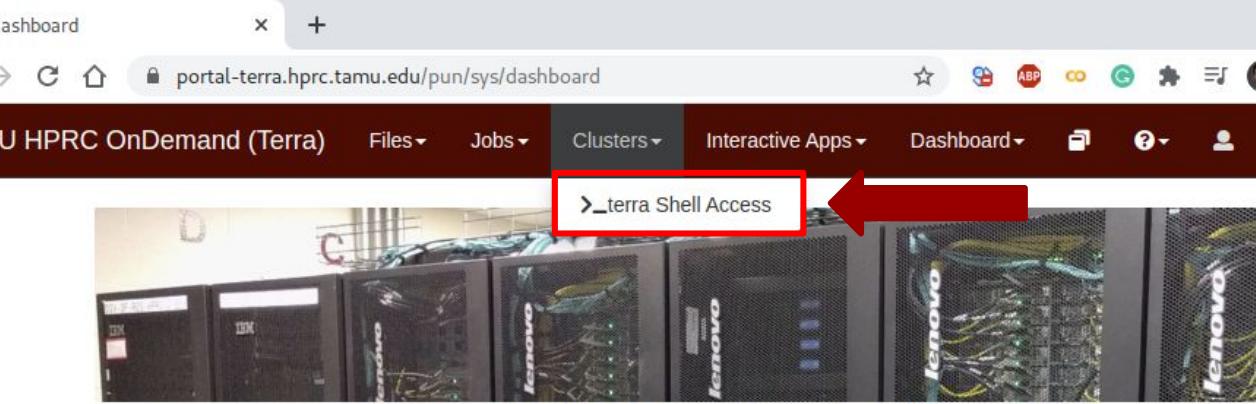
[Ada OnDemand Portal](#)

[Terra OnDemand Portal](#)

[OnDemand Portal User Guide](#)

A red rectangular box highlights the "Terra OnDemand Portal" link. A large red arrow points from the right side towards this highlighted link.

# Terra Shell Access - I



OnDemand provides an integrated, single access point for all of your HPC resources.

Message of the Day

---

**IMPORTANT POLICY INFORMATION**

- Unauthorized use of HPRC resources is prohibited and subject to criminal prosecution.
- Use of HPRC resources in violation of United States export control laws and regulations is prohibited. Current HPRC staff members are US citizens and legal residents.
- Sharing HPRC account and password information is in violation of State Law. Any shared accounts will be DISABLED.

# Terra Shell Access - II

A screenshot of a web browser window titled "Shell". The address bar shows the URL `portal-terra.hprc.tamu.edu/pun/sys/shell/ssh/terra.tamu.edu`. The main content area displays a terminal session. The session starts with a security notice:

```
*****  
This computer system and the data herein are available only for authorized  
purposes by authorized users: use for any other purpose is prohibited and may  
result in administrative/disciplinary actions or criminal prosecution against  
the user. Usage may be subject to security testing and monitoring to ensure  
compliance with the policies of Texas A&M University, Texas A&M University  
System, and the State of Texas. There is no expectation of privacy on this  
system except as otherwise provided by applicable privacy laws. Users should  
refer to Texas A&M University Standard Administrative Procedure 29.01.03.M0.02,  
Rules for Responsible Computing, for guidance on the appropriate use of Texas  
A&M University information resources.  
*****
```

Following the notice, the terminal prompts for a password:

```
Password:  
Duo two-factor login for jtao
```

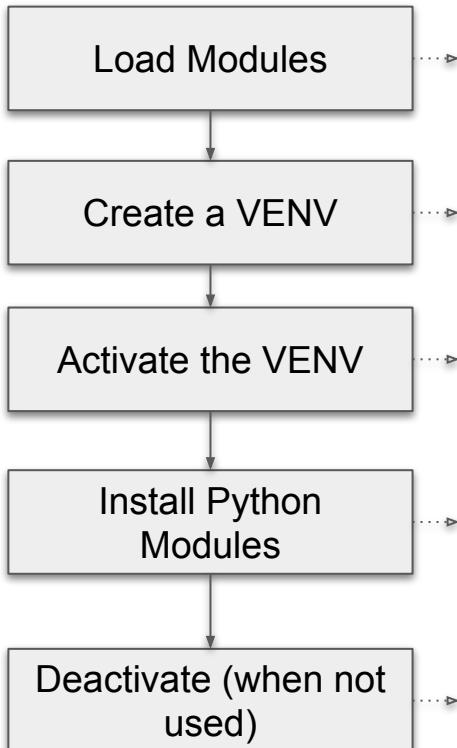
It then asks for a passcode or option:

```
Enter a passcode or select one of the following options:  
1. Duo Push to iPhone (iOS)  
2. Duo Push to iPad (iOS)
```

The user has entered option 1:

```
Passcode or option (1-2): 1
```

# Python Virtual Environment (VENV)



```
# clean up and load Anaconda
cd $SCRATCH
module purge
module load Python/3.7.4-GCCcore-8.3.0

# create a Python virtual environment
python -m venv mylab

# activate the virtual environment
source mylab/bin/activate

# install required package to be used in the portal
pip install --upgrade pip setuptools
pip install jupyterlab torch torchvision
pip install pandas scikit-plot tensorboard

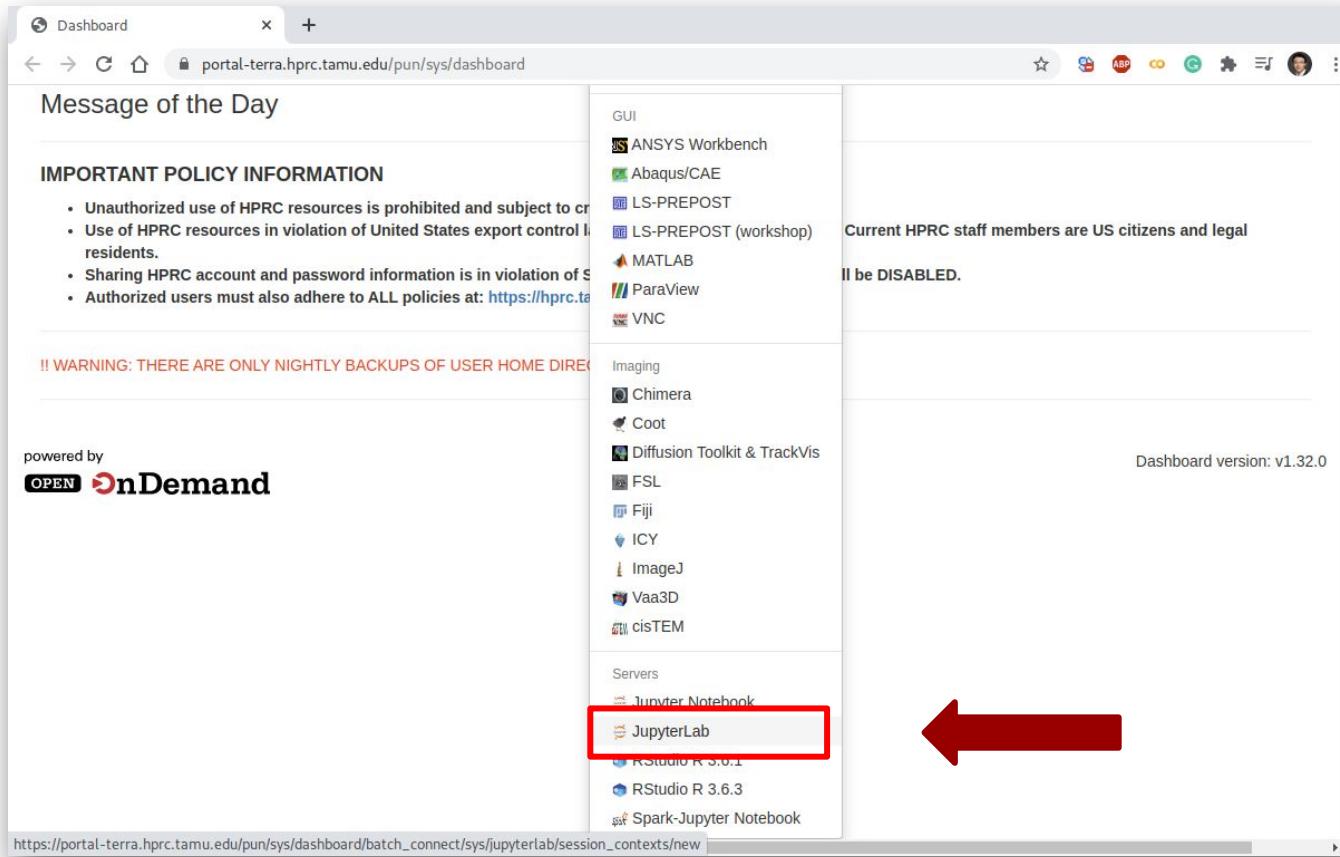
# deactivate the virtual environment
# source deactivate
```

# Check out Exercises

The screenshot shows a GitHub repository page for the user 'jtao' named 'shortcourses'. The 'Code' tab is selected. On the left, there's a sidebar with a 'master' branch dropdown, a file list ('jtao Create README.md', 'Intro\_to\_Julia'), and an 'Add files via' button. On the right, there's an 'About' section with a description: 'No description, website, or topics provided.' Below it are links for 'Clone' (with options for HTTPS, SSH, and GitHub CLI) and a 'Readme' file. A modal window is open over the page, containing the command: '# git clone (check out) the Jupyter notebooks for the short courses git clone https://github.com/jtao/shortcourses.git'.

```
# git clone (check out) the Jupyter notebooks for the short courses
git clone https://github.com/jtao/shortcourses.git
```

# Go to JupyterLab Page



# Set Virtual Environment

The screenshot shows a web browser window titled "JupyterLab" at the URL [portal-terra.hprc.tamu.edu/pun/sys/dashboard/batch\\_connect/sys/jupyterlab/session\\_contexts/new](https://portal-terra.hprc.tamu.edu/pun/sys/dashboard/batch_connect/sys/jupyterlab/session_contexts/new). The page is part of the "TAMU HPRC OnDemand (Terra)" interface. The main content area is titled "JupyterLab" and describes launching a JupyterLab server on the Terra cluster. A dropdown menu under "Module" is set to "Python/3.7.4-GCCcore-8.3.0". Below it, a note states "Anaconda/3-x.x.x is Python3". A red box highlights the "Optional Conda Environment to be activated" input field, which contains the path "/scratch/user/jtao/mylab/bin/activate". A large red arrow points from this field towards the terminal command below.

```
# enter the full path of the activate command of your virtualenv  
/scratch/user/YOURNETID/mylab/bin/activate
```

GUI

ANSYS Workbench

Abaqus/CAE

Number of hours

1

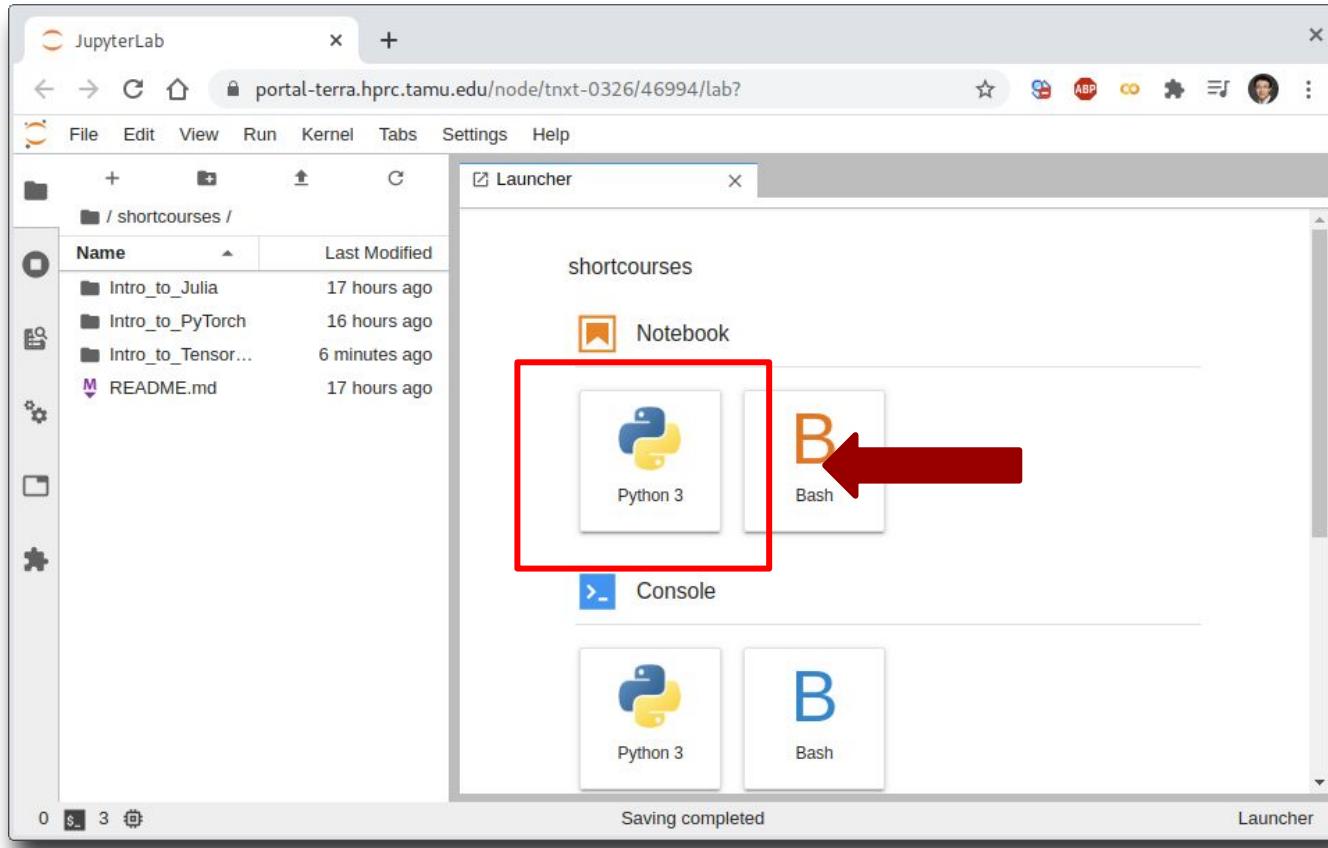
# Connect to JupyterLab

The screenshot shows a web browser window for the TAMU HPRC OnDemand (Terra) portal. The URL is `portal-terra.hprc.tamu.edu/pun/sys/dashboard/batch_connect/sessions`. The page displays a message "Session was successfully deleted." in a green box. Below it, a JupyterLab session is listed with the ID 6119424. The session details are as follows:

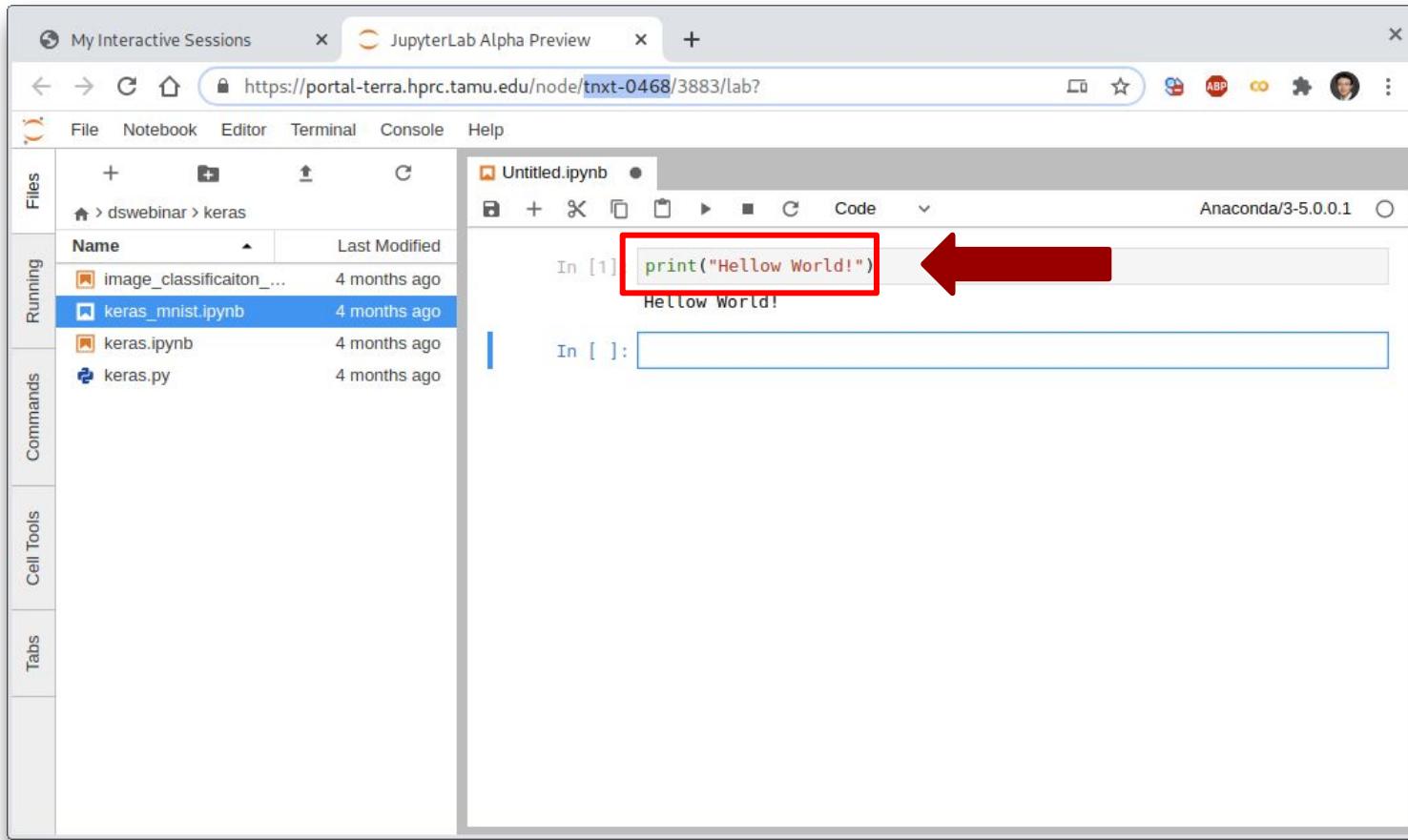
- Host:** tnxt-0468
- Created at:** 2020-11-12 01:49:27 CST
- Time Remaining:** 56 minutes
- Session ID:** 5cbf368d-1a3a-4154-8689-ac13dcd1cdde
- Status:** 1 node | 1 core | Running

A red box highlights the "Connect to JupyterLab" button, and a large red arrow points to it from the bottom right.

# Create a Jupyter Notebook



# Test JupyterLab



# Part II. Deep Learning

***Deep Learning***

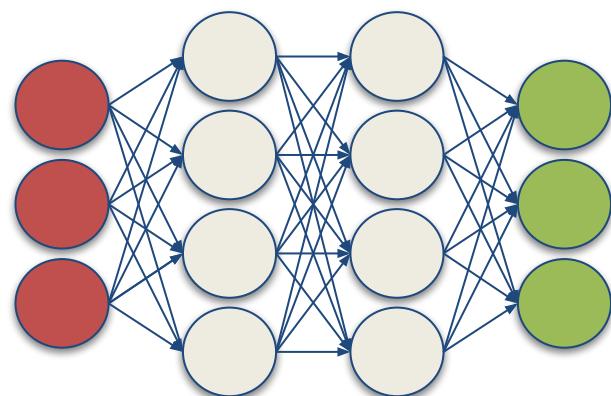
by Ian Goodfellow, Yoshua Bengio, and Aaron Courville

<http://www.deeplearningbook.org/>

***Animation of Neuron Networks***

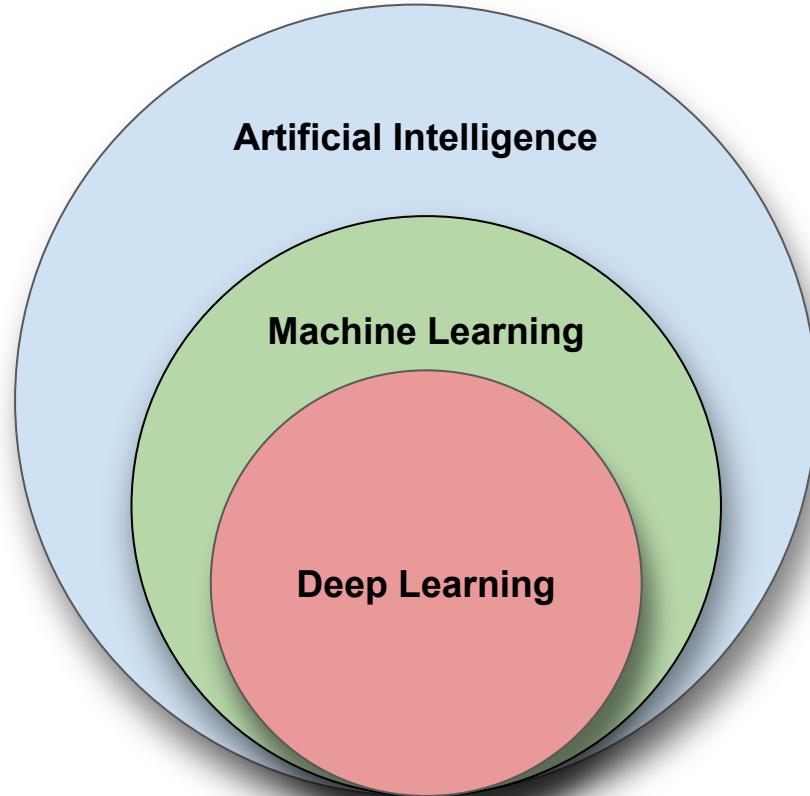
by Grant Sanderson

<https://www.3blue1brown.com/>

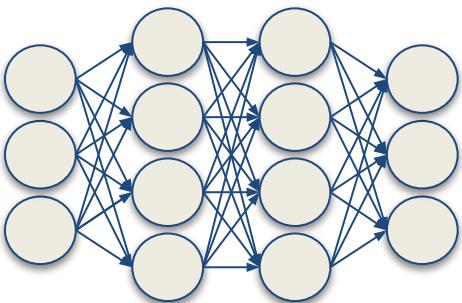
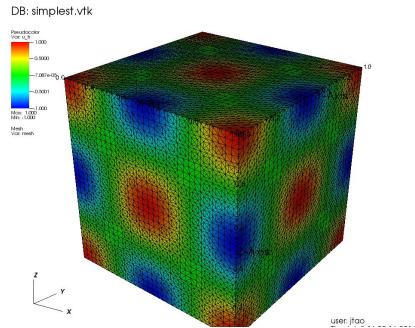


# Relationship of AI, ML and DL

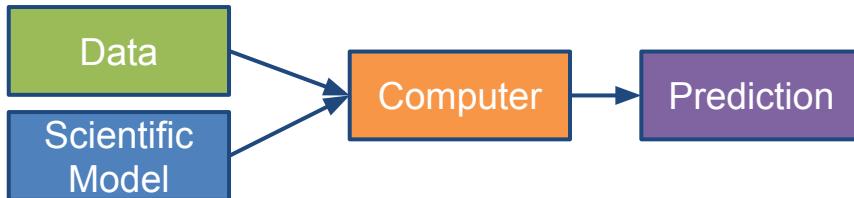
- **Artificial Intelligence (AI)** is anything about man-made intelligence exhibited by machines.
- **Machine Learning (ML)** is an approach to achieve **AI**.
- **Deep Learning (DL)** is one technique to implement **ML**.



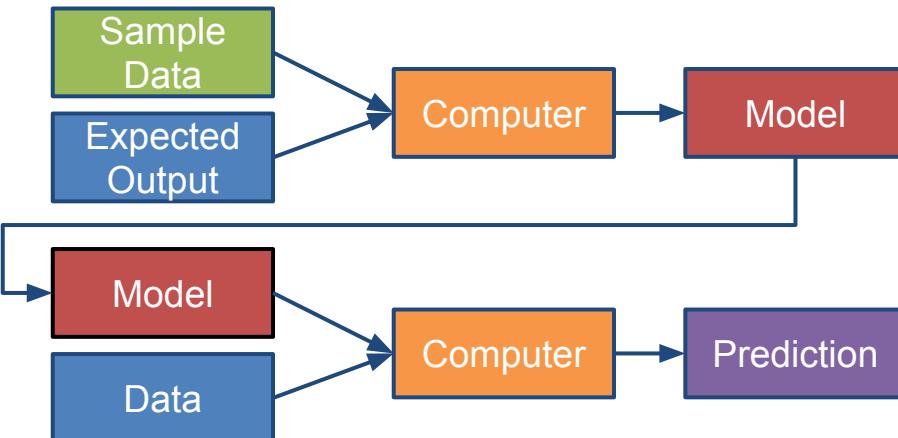
# Machine Learning



## Traditional Modeling

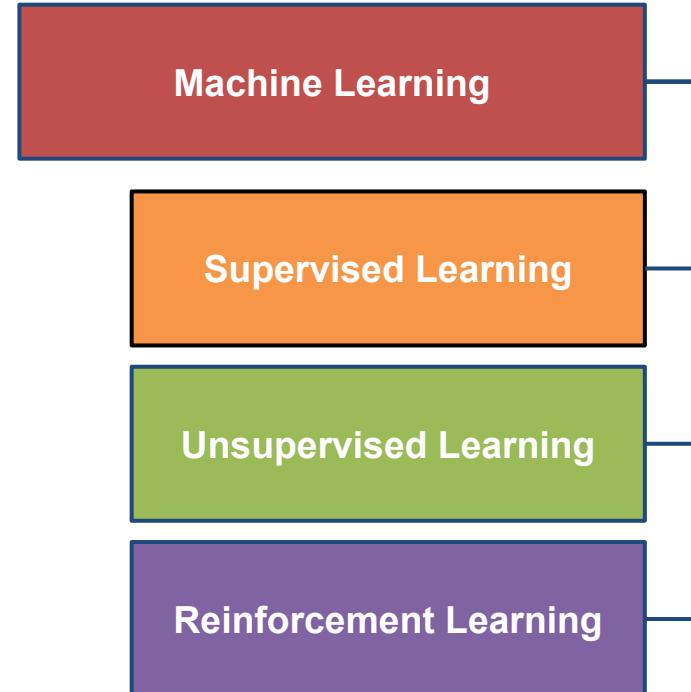


## Machine Learning (Supervised Learning)



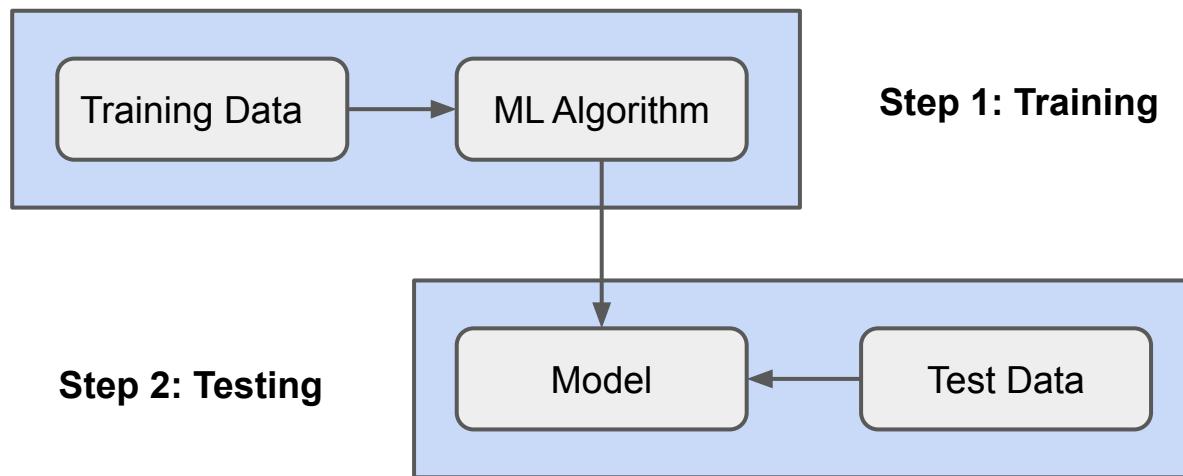
# Types of ML Algorithms

- **Supervised Learning**
  - trained with labeled data; including regression and classification problems
- **Unsupervised Learning**
  - trained with unlabeled data; clustering and association rule learning problems.
- **Reinforcement Learning**
  - no training data; stochastic Markov decision process; robotics and self-driving cars.



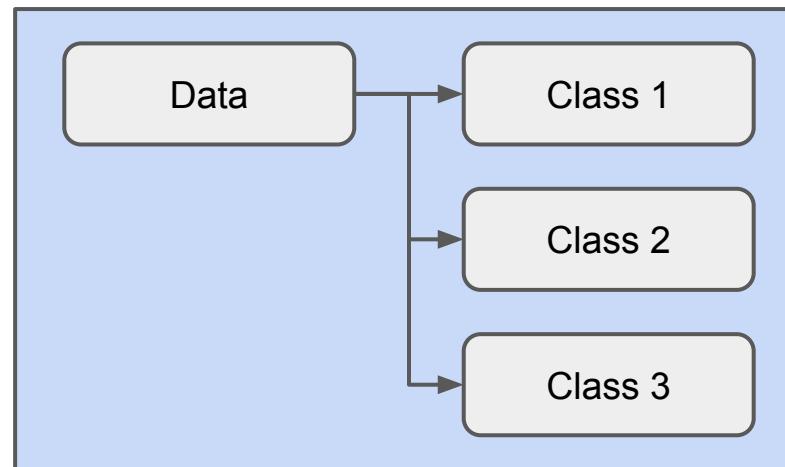
# Supervised Learning

When both input variables - X and output variables - Y are known, one can approximate the mapping function from X to Y.



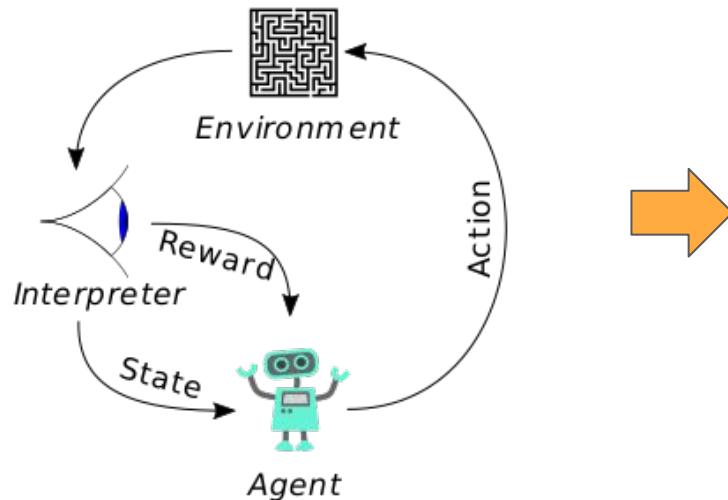
# Unsupervised Learning

When only input variables - X are known and the training data is neither classified nor labeled. It is usually used for clustering problems.

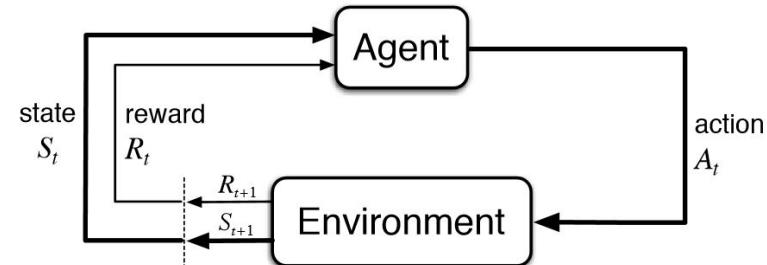


# Reinforcement Learning

When the input variables are only available via interacting with the environment, reinforcement learning can be used to train an "agent".



(Image Credit: Wikipedia.org)



(Image Credit: deeplearning4j.org)

# Why Deep Learning?

- Limitations of traditional machine learning algorithms
  - not good at handling high dimensional data.
  - difficult to do feature extraction and object recognition.
- Advantages of deep learning
  - DL is computationally expensive, but it is capable of handling high dimensional data.
  - feature extraction is done automatically.

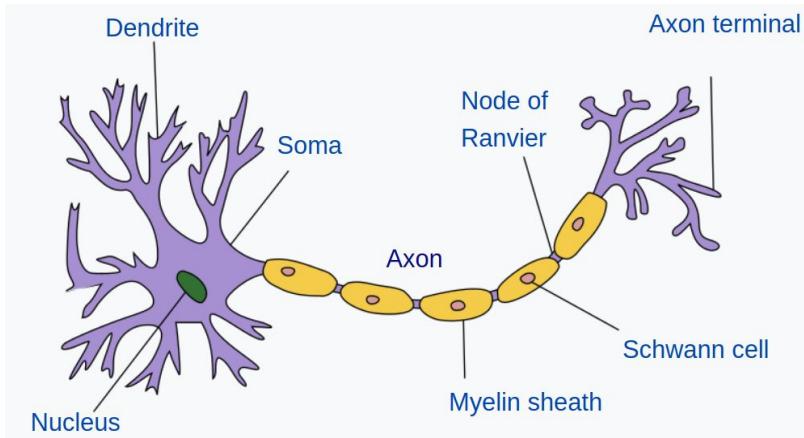
# What is Deep Learning?

Deep learning is a class of machine learning algorithms that:

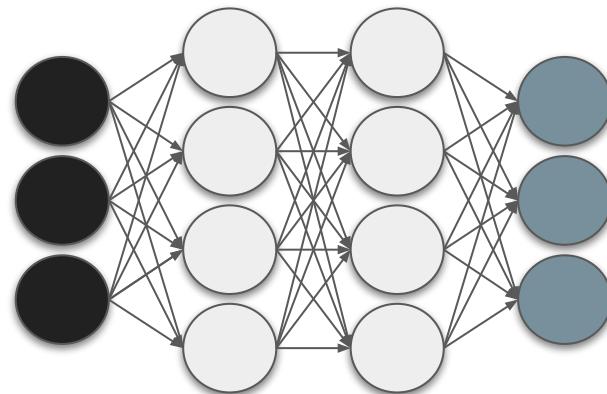
- use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.
- learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manners.
- learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

(Source: Wikipedia)

# Artificial Neural Network

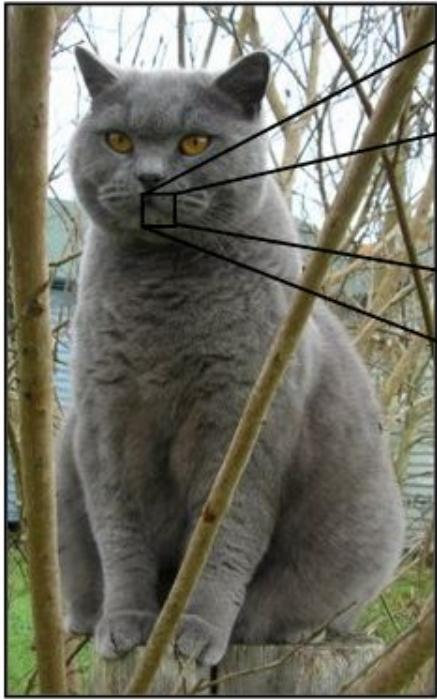


Input      Hidden Layers      Output



(Image Credit: Wikipedia)

# Inputs and Outputs

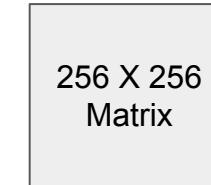


05	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	00	
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	08	49	13	36	65
52	70	95	23	04	60	11	42	69	51	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	63	03	69	41	92	36	54	22	40	40	28	66	33	13	80
24	47	34	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
52	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	50	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	32	08	03	97	35	99	16	07	97	57	32	16	26	26	79	35	27	98	66
59	34	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	55	36	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	37	03	92	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	49	55	51	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	67	48

What the computer sees

image classification →

82% cat  
15% dog  
2% hat  
1% mug



DL model

4-Element Vector

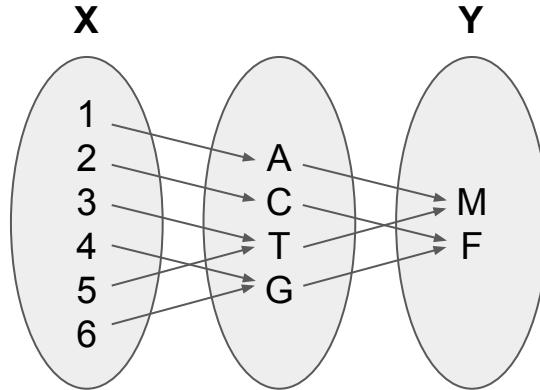
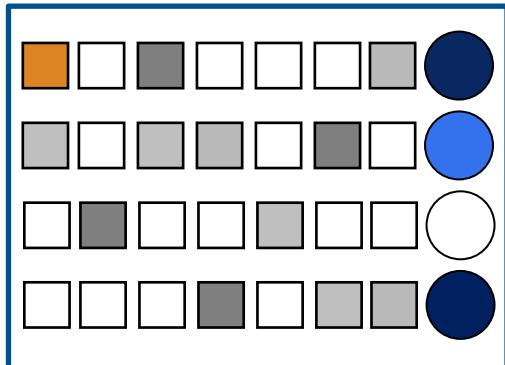


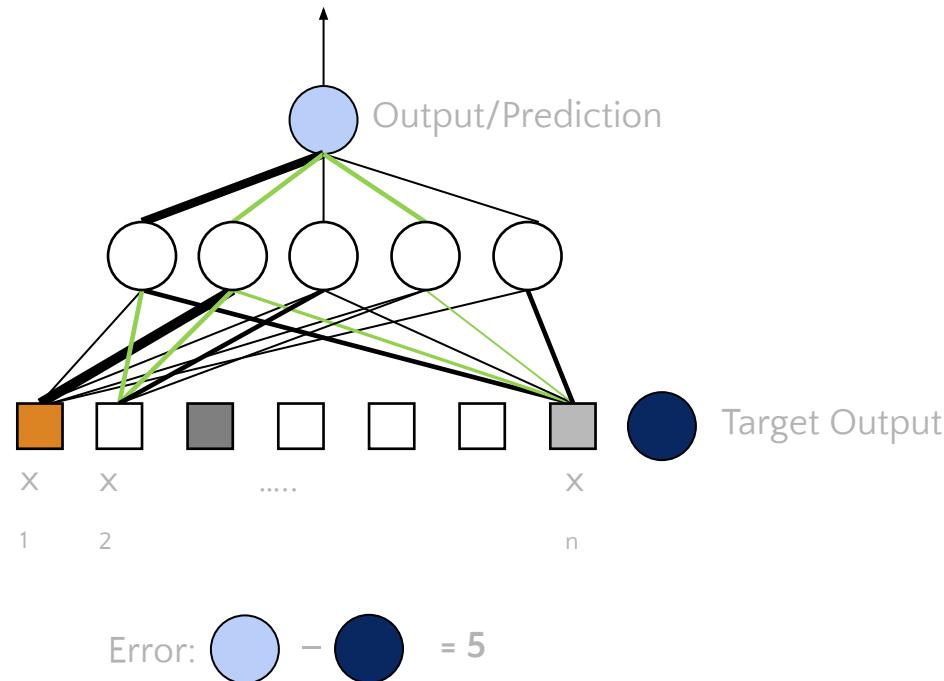
Image from the [Stanford CS231 Course](#)

With deep learning, we are searching for a **surjective** (or **onto**) function  $f$  from a set  $X$  to a set  $Y$ .

Dataset

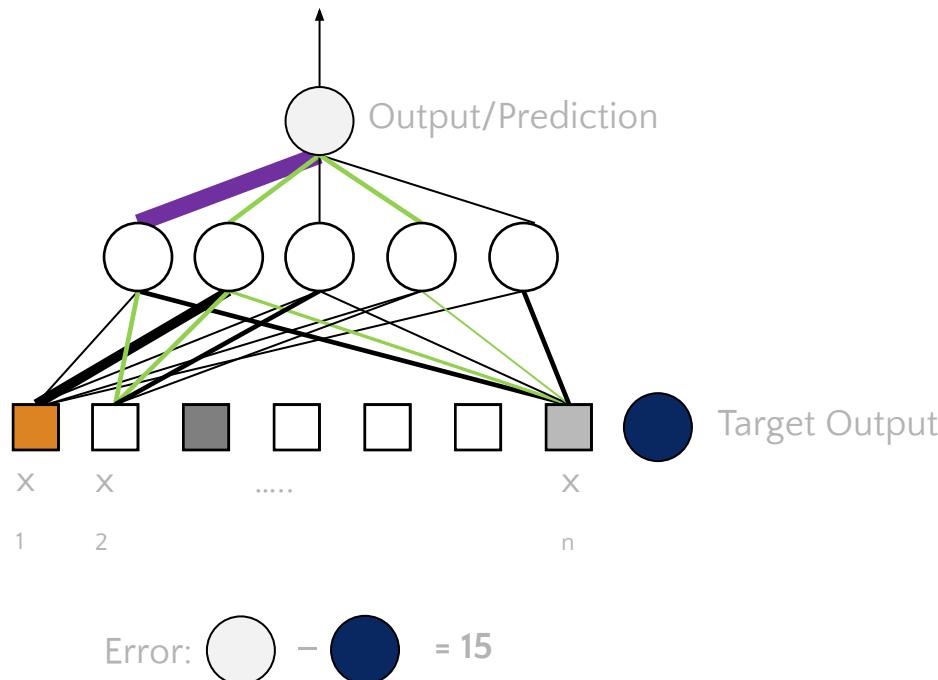


# Learning Principle



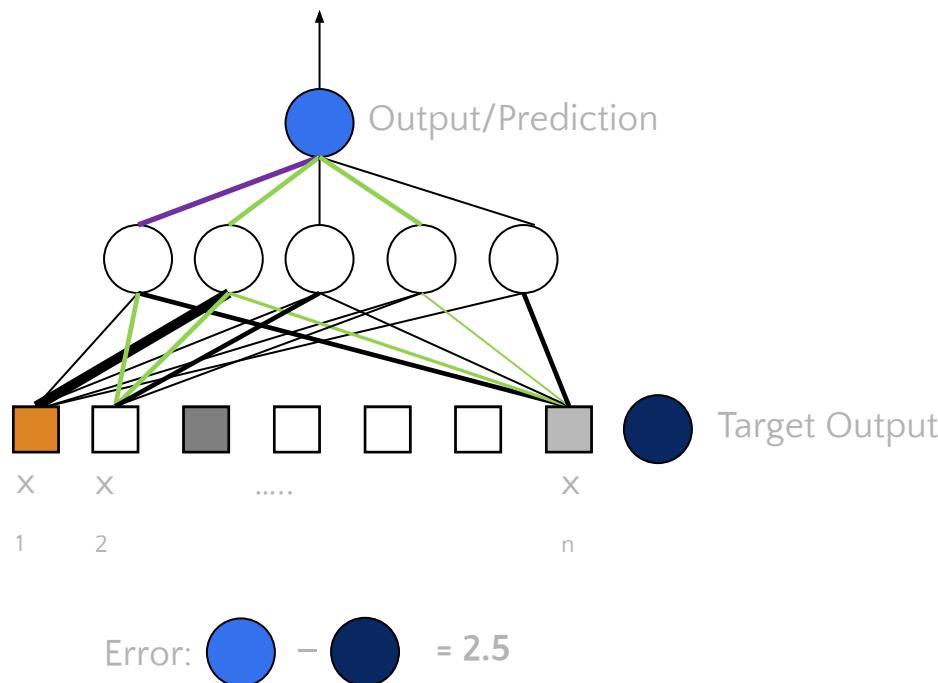
(Image Credit: NVIDIA Deep Learning Institute)

# Learning Principle



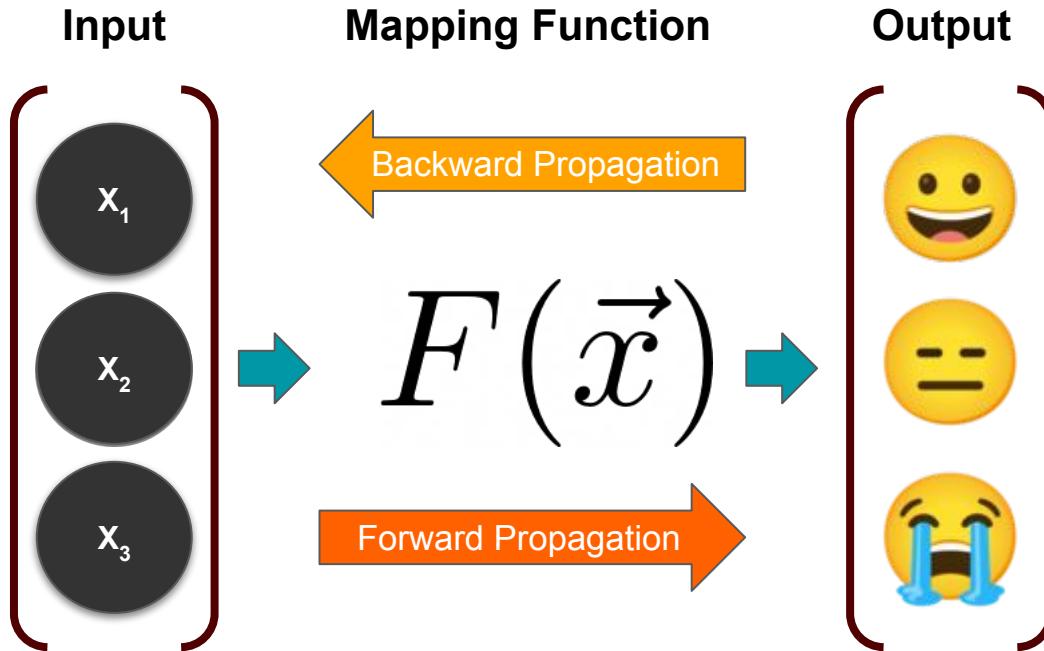
(Image Credit: NVIDIA Deep Learning Institute)

# Learning Principle



(Image Credit: NVIDIA Deep Learning Institute)

# Deep Neural Network as a Nonlinear Function



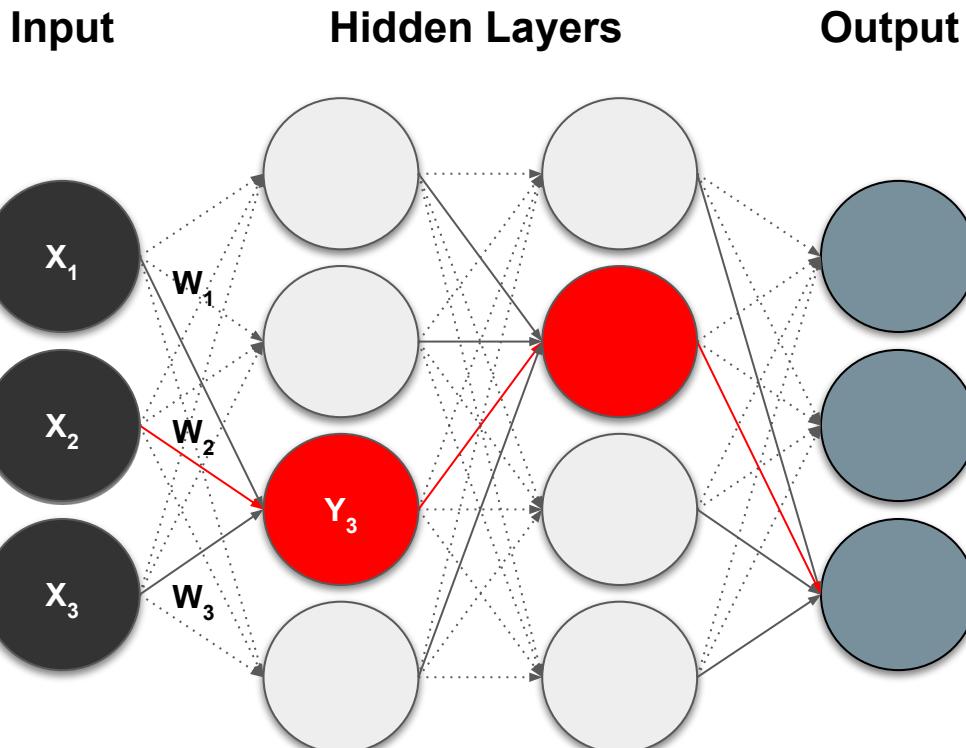
- **Training:** given **input** and **output**, find best-fit  $F$
- **Inference:** given **input** and  $F$ , predict **output**

# Supervised Deep Learning with Neural Networks

From one layer to the next

$$Y_j = f \left( \sum_i W_i X_i + b_i \right)$$

f is the activation function,  
W<sub>i</sub> is the weight, and b<sub>i</sub> is  
the bias.



# Training - Minimizing the Loss

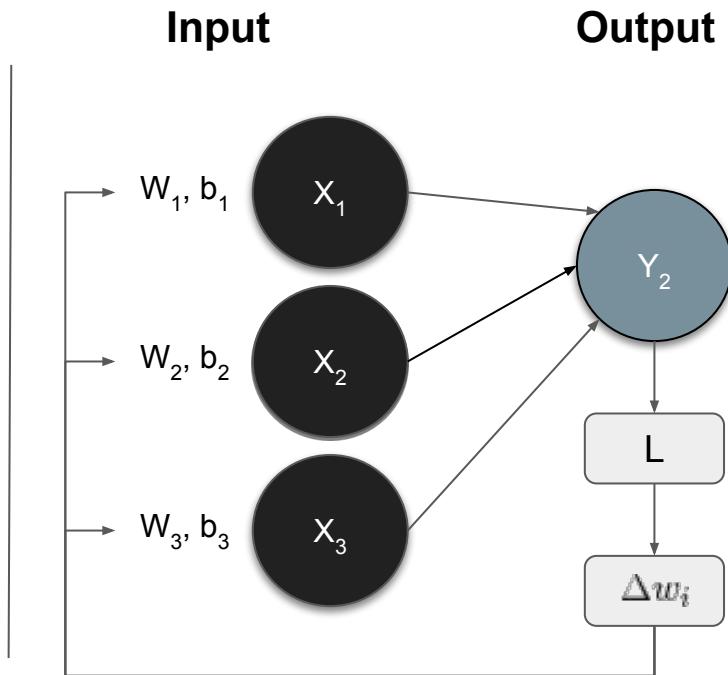
The loss function with regard to weights and biases can be defined as

$$L(\mathbf{w}, \mathbf{b}) = \frac{1}{2} \sum_i (\mathbf{Y}(\mathbf{X}, \mathbf{w}, \mathbf{b}) - \mathbf{Y}'(\mathbf{X}, \mathbf{w}, \mathbf{b}))^2$$

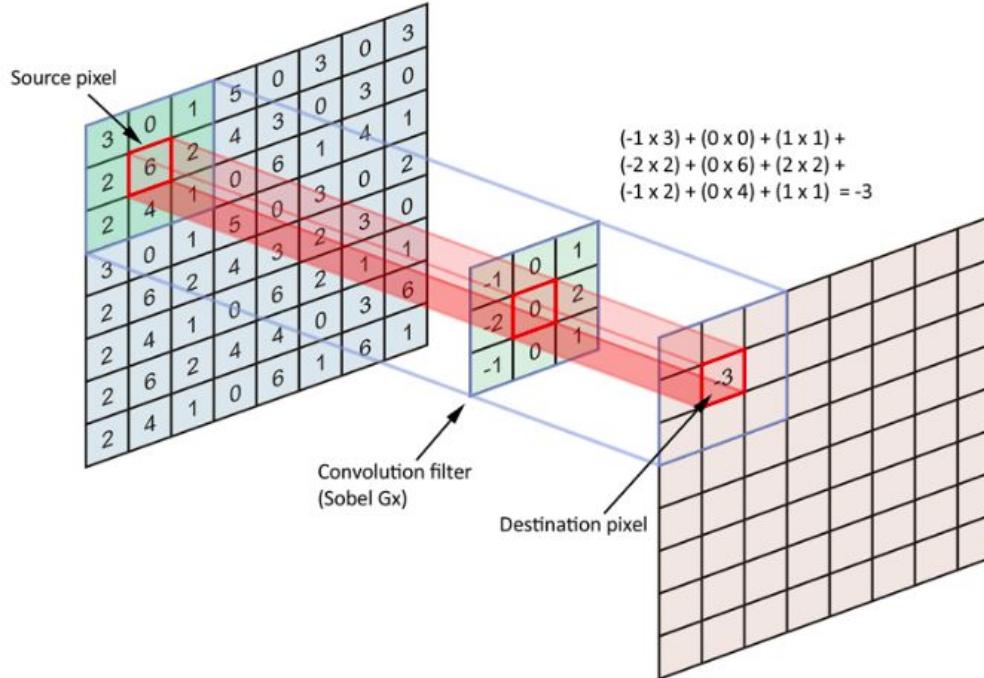
The weight update is computed by moving a step to the opposite direction of the cost gradient.

$$\Delta w_i = -\alpha \frac{\partial L}{\partial w_i}$$

Iterate until  $L$  stops decreasing.



# Convolution in 2D



(Image Credit: [Applied Deep Learning | Arden Dertat](#))

# Convolution Kernel

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

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

(Image Credit: [Applied Deep Learning | Arden Dertat](#))

# Convolution on Image

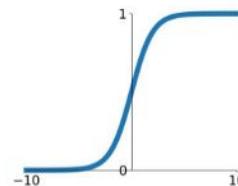


Image Credit: [Deep Learning Methods for Vision | CVPR 2012 Tutorial](#)

# Activation Functions

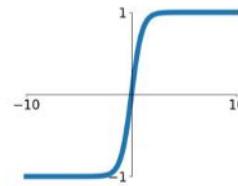
**Sigmoid**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



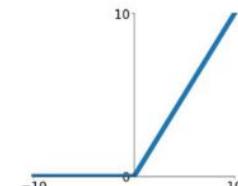
**tanh**

$$\tanh(x)$$



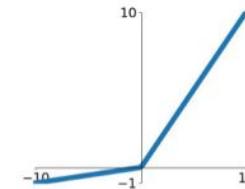
**ReLU**

$$\max(0, x)$$



**Leaky ReLU**

$$\max(0.1x, x)$$



**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

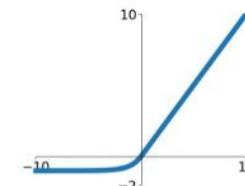


Image Credit: [towardsdatascience.com](https://towardsdatascience.com/)

# Introducing Non Linearity (ReLU)

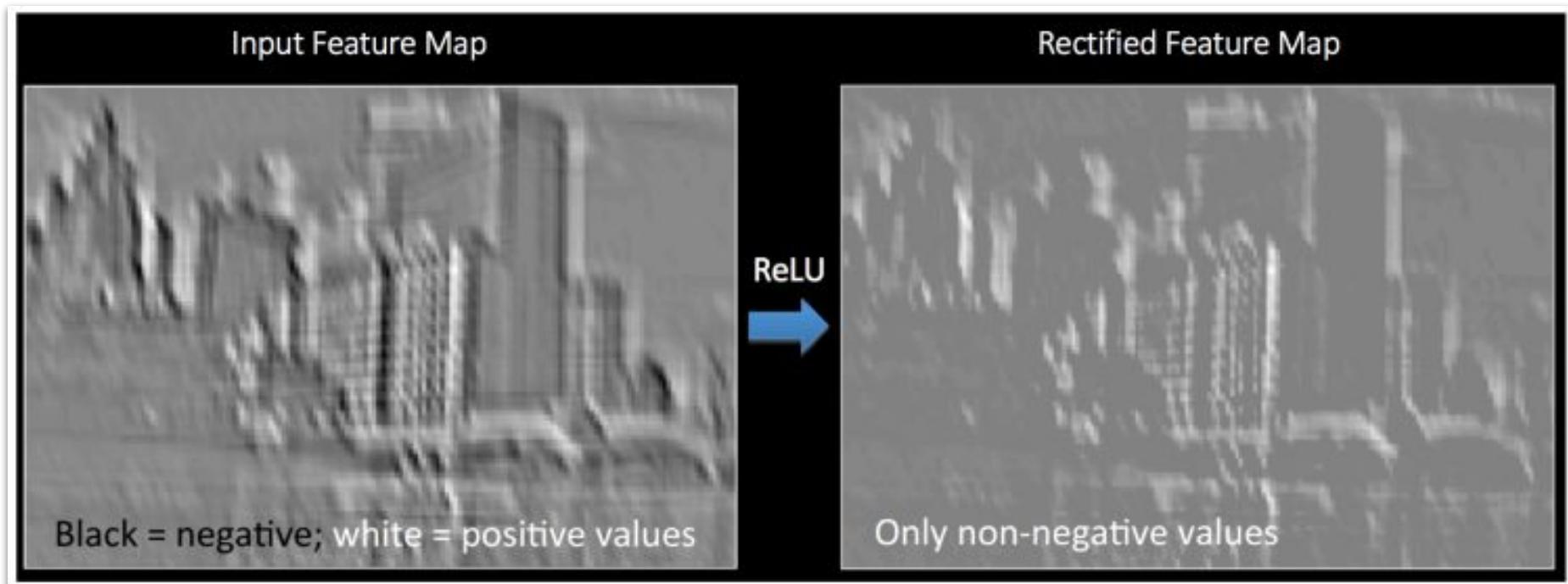
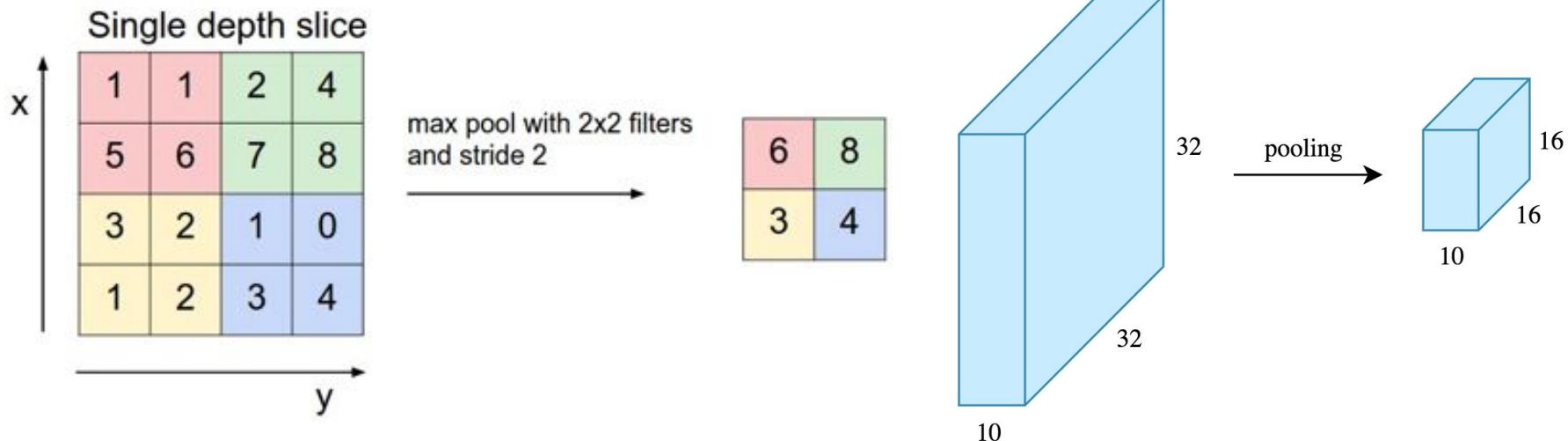


Image Credit: [Deep Learning Methods for Vision | CVPR 2012 Tutorial](#)

# Max Pooling



(Image Credit: [Applied Deep Learning | Arden Dertat](#))

# Pooling - Max-Pooling and Sum-Pooling

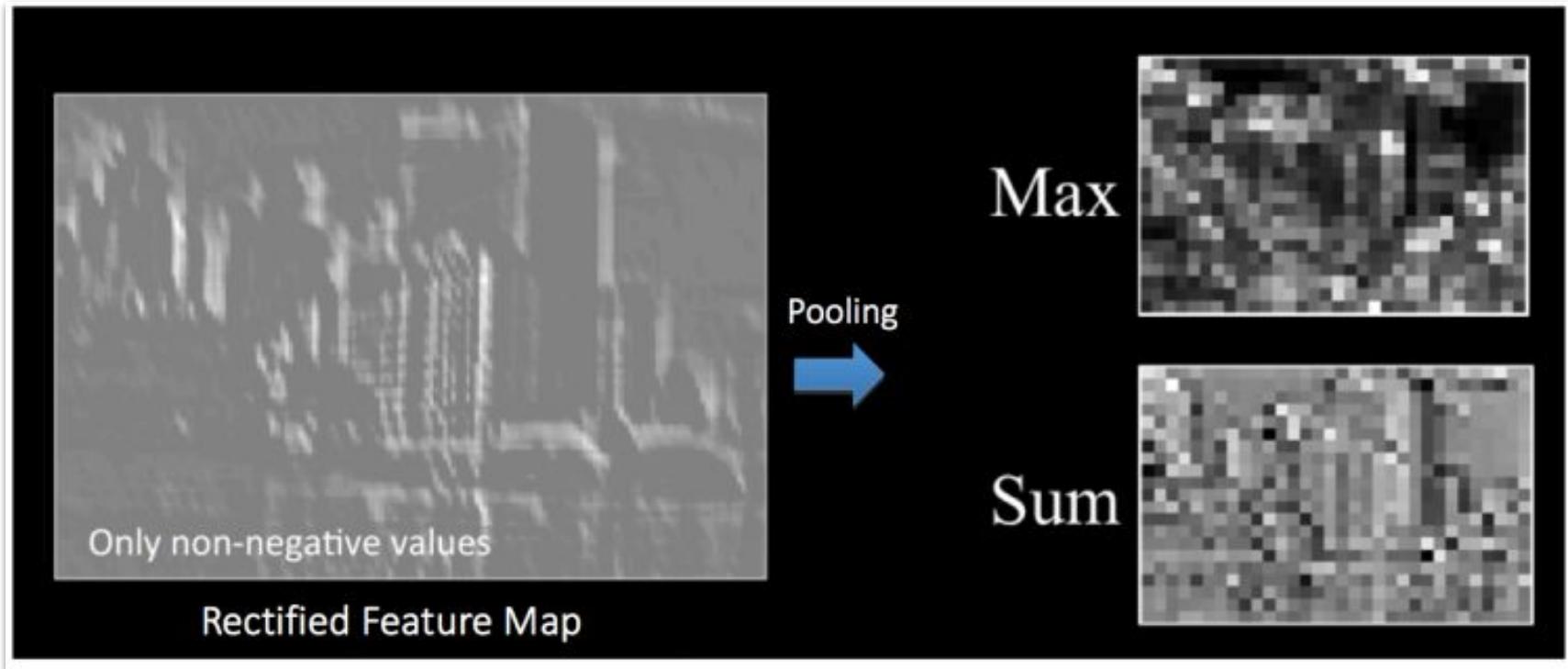
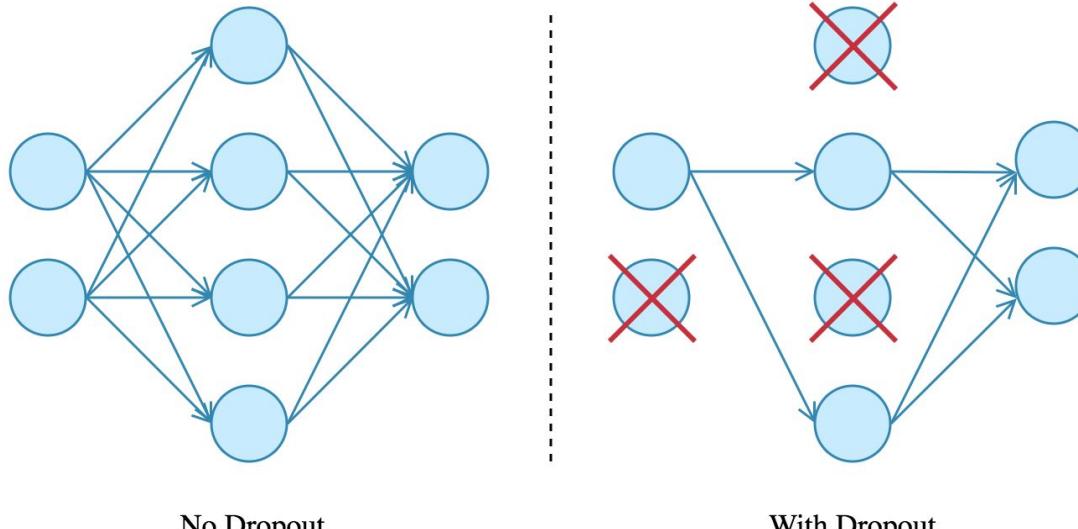


Image Credit: [Deep Learning Methods for Vision | CVPR 2012 Tutorial](#)

# CNN Implementation - Drop Out

Dropout is used to prevent overfitting. A neuron is temporarily “dropped” or disabled with probability  $P$  during training.

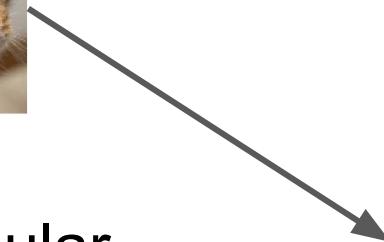


No Dropout

With Dropout

(Image Credit: [Applied Deep Learning | Arden Dertat](#))

# CNN Implementation - Data Augmentation (DA)

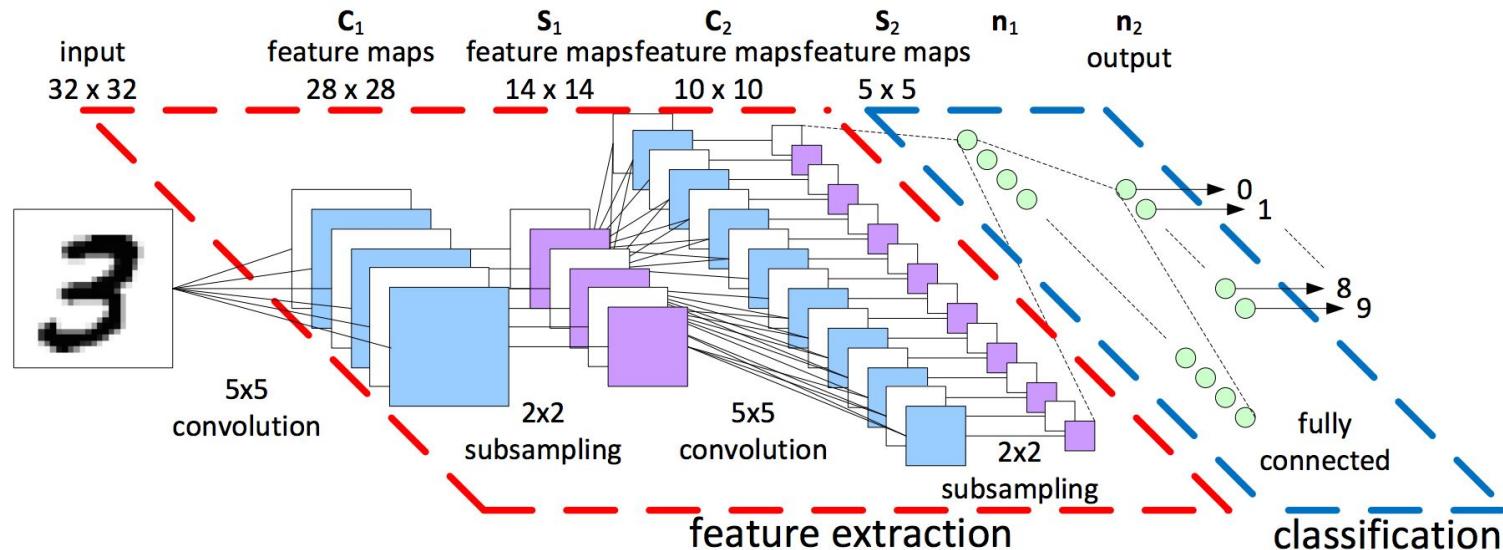


DA helps to popular  
artificial training  
instances from the  
existing train data sets.

(Image Credit: [Applied Deep Learning | Arden Dertat](#))

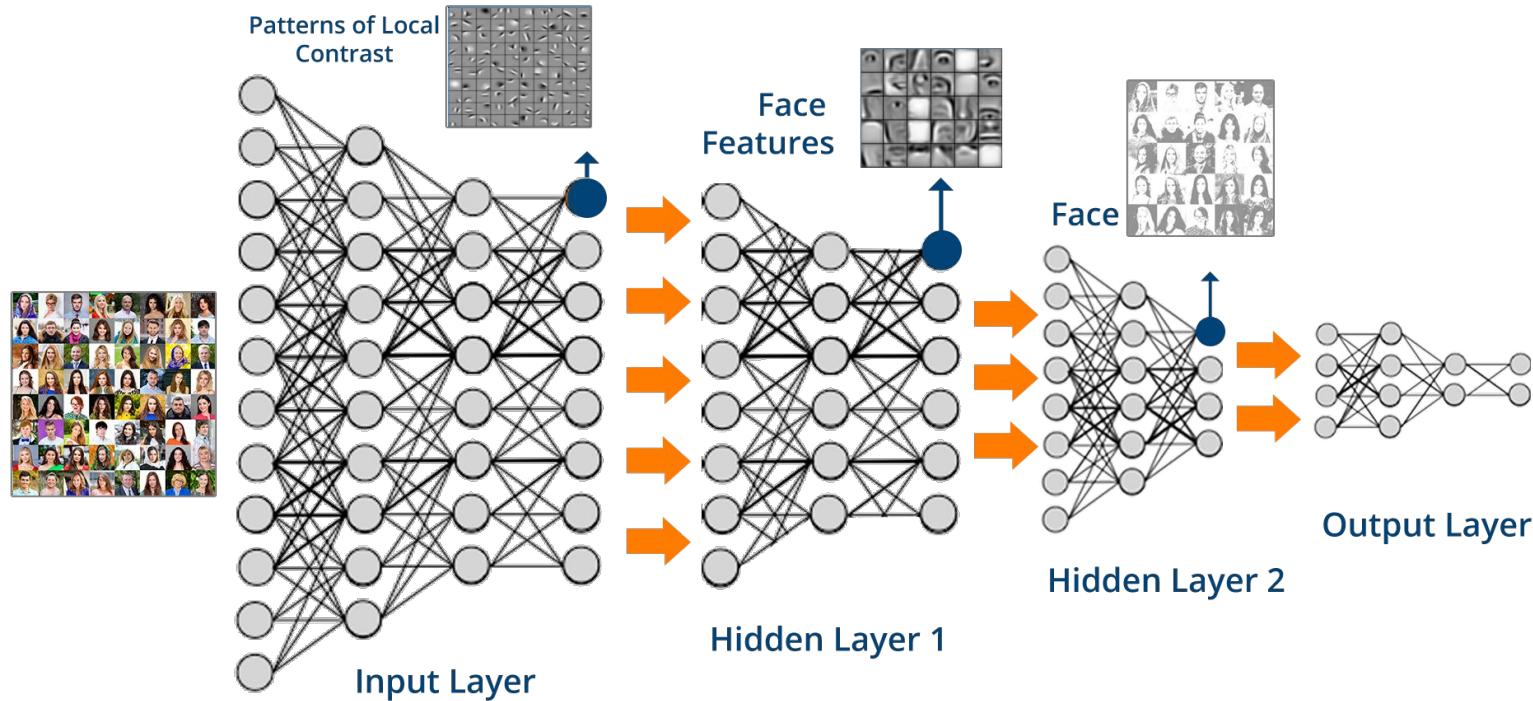
# Convolutional Neural Networks

A convolutional neural network (**CNN**, or **ConvNet**) is a class of deep, feed-forward artificial neural networks that explicitly assumes that the inputs are images, which allows us to encode certain properties into the architecture.



LeNet-5 Architecture (Image Credit: <https://becominghuman.ai>)

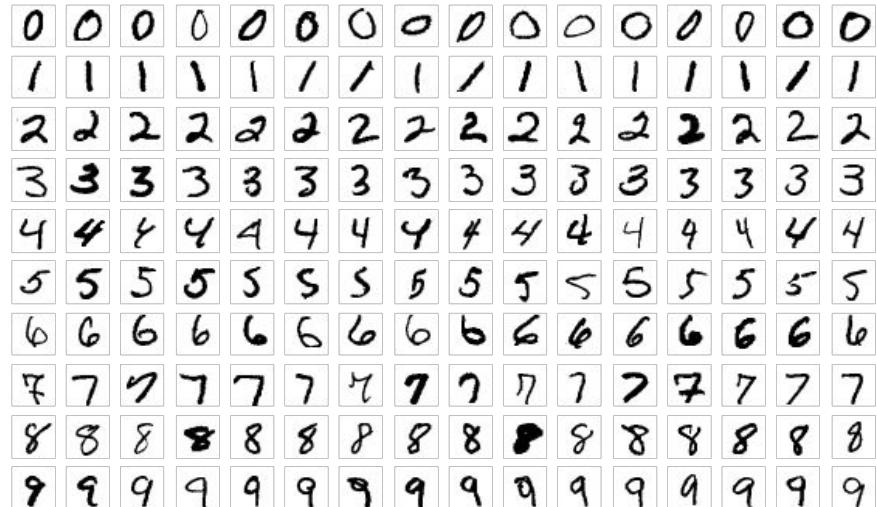
# Deep Learning for Facial Recognition



(Image Credit: [www.edureka.co](http://www.edureka.co))

# MNIST - Introduction

- **MNIST** (Mixed National Institute of Standards and Technology) is a database for handwritten digits, distributed by Yann Lecun.
- 60,000 examples, and a test set of 10,000 examples.
- 28x28 pixels each.
- Widely used for research and educational purposes.



(Image Credit: Wikipedia)

# MNIST - CNN Visualization



(Image Credit: <http://scs.ryerson.ca/~aharley/vis/>)

# **Hands-on Session #1**

## **A Simple Deep Learning Example with**

### **PyTorch - First Glance**



# Part III. Introduction to PyTorch

*PyTorch website:*  
<https://pytorch.org/>

*Deep Learning with PyTorch:*  
<https://pytorch.org/tutorials/>



# A Brief History of PyTorch

PyTorch is an open source machine learning library based on the Torch library, which was first released by Ronan Collobert, Koray Kavukcuoglu, and Clement Farabet in Oct 2002.

- The first official release of PyTorch was by Facebook's AI Research lab (FAIR) in Oct 2016.
- Version 1.0 that integrated both Caffe2 and ONNX was release in May 2018.
- The latest release is version 1.4.0, as of Feb 13 2020.

# Overview of PyTorch

**PyTorch** is an open-source machine learning library written in Python, C++ and CUDA. PyTorch provides two high-level features:

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU)
- Deep neural networks built on a tape-based autodiff system

In a layman's term, PyTorch is a fancy version of NumPy that runs on GPUs and comes with a lot of machine learning functionalities.

# TensorFlow, Keras, and PyTorch



**TensorFlow** is an end-to-end open source **platform** for machine learning. It has a comprehensive, flexible ecosystem to build and deploy ML powered applications.

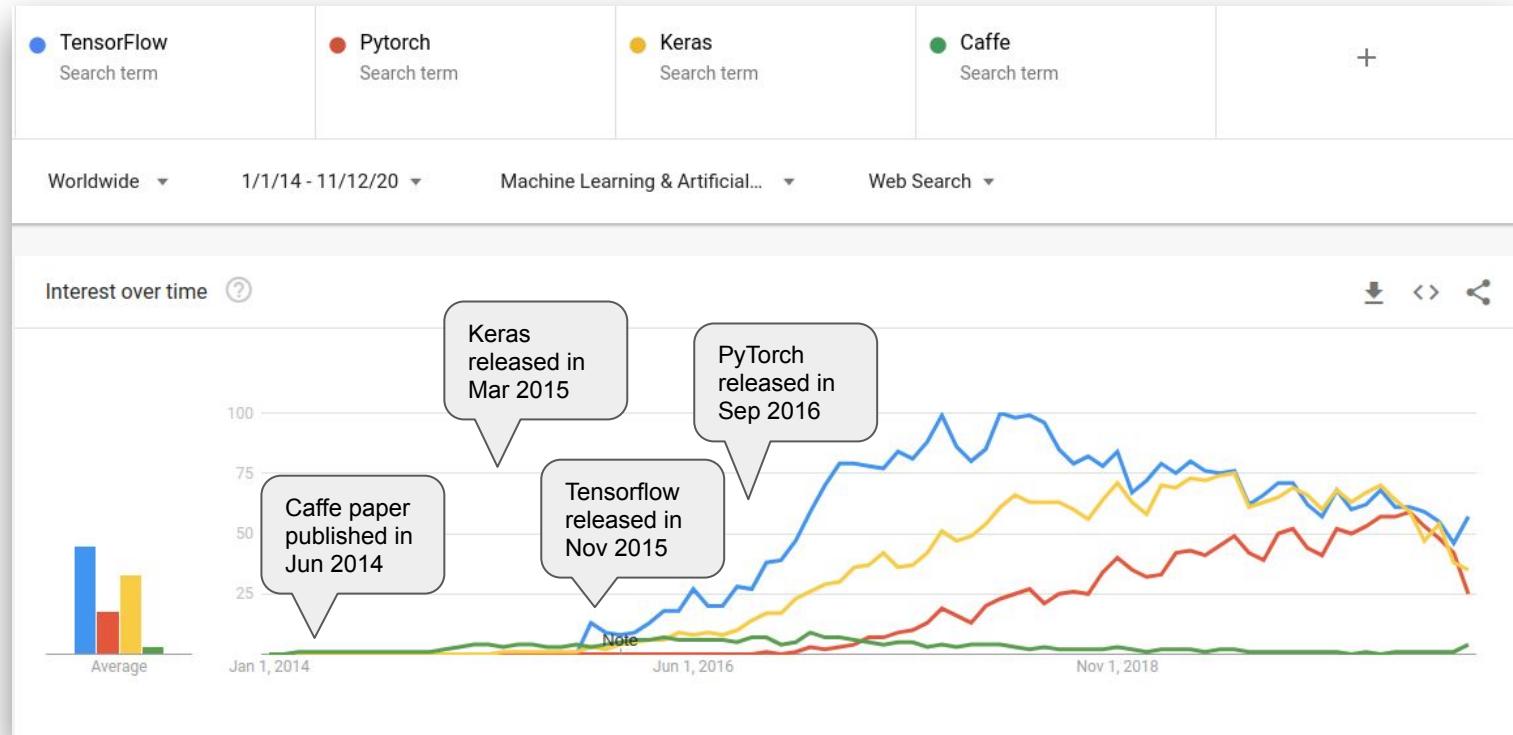


**Keras** is a high-level neural networks **API**, written in Python and capable of running on top of *TensorFlow*, *CNTK*, or *Theano*. It was developed with a focus on enabling fast experimentation.



**PyTorch** is an open source machine learning **framework** that accelerates the path from research prototyping to production deployment.

# Google Trends for Popular ML Frameworks



(Image Credit: <https://trends.google.com/>)

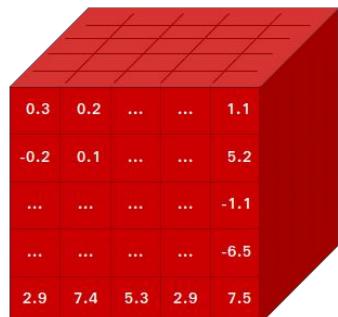
# Major Components of PyTorch

Components	Description
<code>torch</code>	a <i>Tensor library like NumPy</i> , with strong <i>GPU</i> support
<code>torch.autograd</code>	a <i>tape-based automatic differentiation library</i> that supports all differentiable Tensor operations in torch
<code>torch.jit</code>	a <i>compilation stack (TorchScript)</i> to create serializable and optimizable models from PyTorch code
<code>torch.nn</code>	a <i>neural networks library</i> deeply integrated with autograd designed for maximum flexibility
<code>torch.multiprocessing</code>	<i>Python multiprocessing</i> , but with magical memory sharing of torch Tensors across processes. Useful for data loading and Hogwild training
<code>torch.utils</code>	<i>DataLoader and other utility functions</i> for convenience

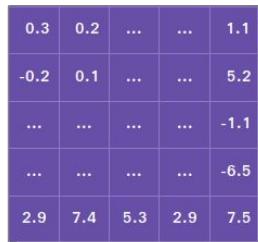
# A Powerful Tensor Library - torch

- A PyTorch tensor is an n-dimensional array that can live on either the CPU or GPU. A tensor has a static type, a rank, and a shape.

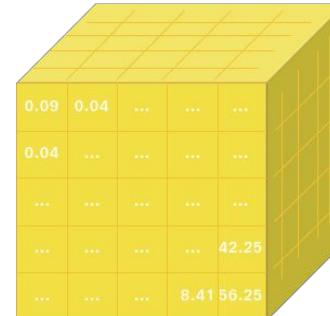
Name	Rank	Tensor
Scalar	0	[5]
Vector	1	[1 2 3]
Matrix	2	[[1 2 3 4], [5 6 7 8]]
Tensor	3	...



\*



=



(Image Credit: pytorch.org)

# Tensors on CPU and GPU - torch

```
x = torch.randn(1)
# check if a CUDA device is available
if torch.cuda.is_available():

    # a CUDA device object
    device = torch.device("cuda")

    # directly create y
    x = x.to(device)
    y = torch.ones_like(x, device=device)

    z = x + y
    print(z)
    print(z.to("cpu", torch.double))
```



# Tape-Based AutoGrad - torch.autograd

- **torch.autograd** is central to all neural networks in PyTorch.
- The **autograd** package provides automatic differentiation for all operations on Tensors.
- Use "**requires\_grad=True**" to keep traction operations on a Tensor.

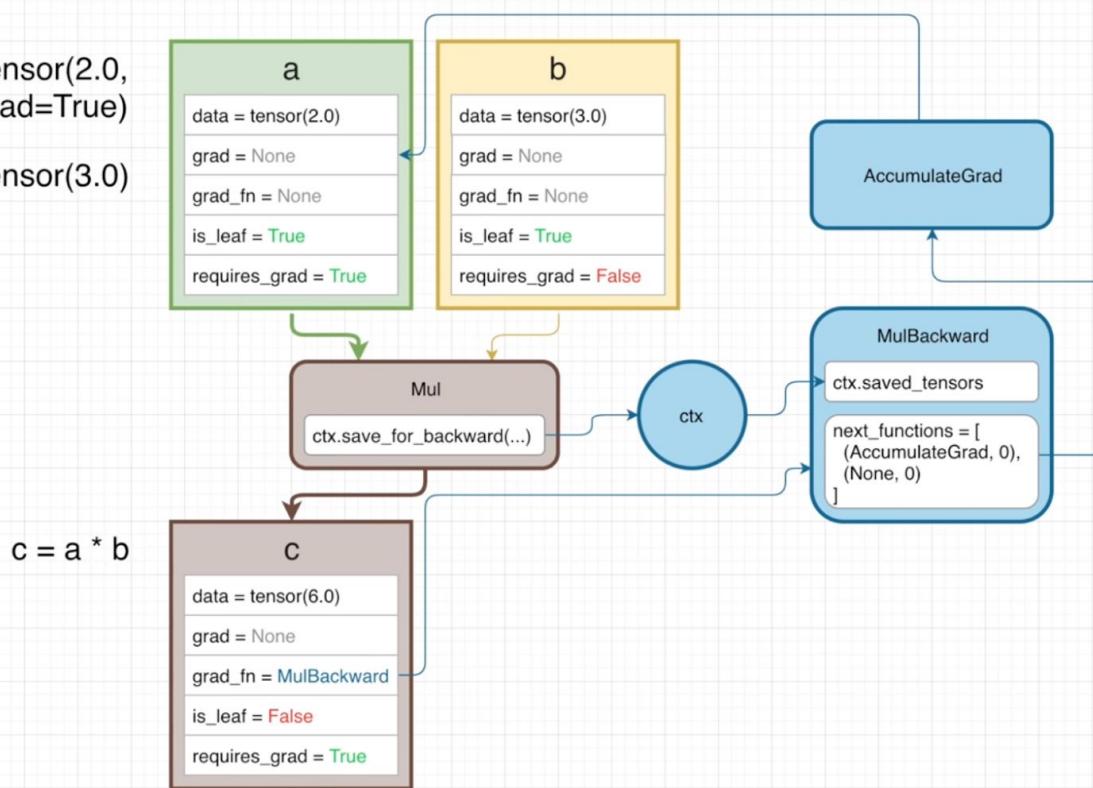
```
# x = tensor([[1., 1.],
             [1., 1.]], requires_grad=True)
x = torch.ones(2, 2, requires_grad=True)

# y = tensor([[3., 3.],
             [3., 3.]], grad_fn=<AddBackward0>)
y = x + 2
```

# Tape-Based AutoGrad - torch.autograd

```
a = torch.tensor(2.0,  
                requires_grad=True)
```

```
b = torch.tensor(3.0)
```



- PyTorch uses and replays a "**tape recorder**" to build neural networks.
- The official name of the method is called **reverse-mode auto-differentiation**.
- The dependent variable is fixed and the derivative is computed with respect to each sub-expression recursively.
- The method requires extra storage to save intermediate states.

# Dynamic Graph with PyTorch

A graph is created on the fly



```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```



(Image Credit: pytorch.org)

# Neural Network - torch.nn

- **torch.nn** depends on **autograd** to define models and differentiate them.
- An **nn.Module** contains layers, and a method **forward(input)** that returns the output.

```
import torch
import torch.nn as nn

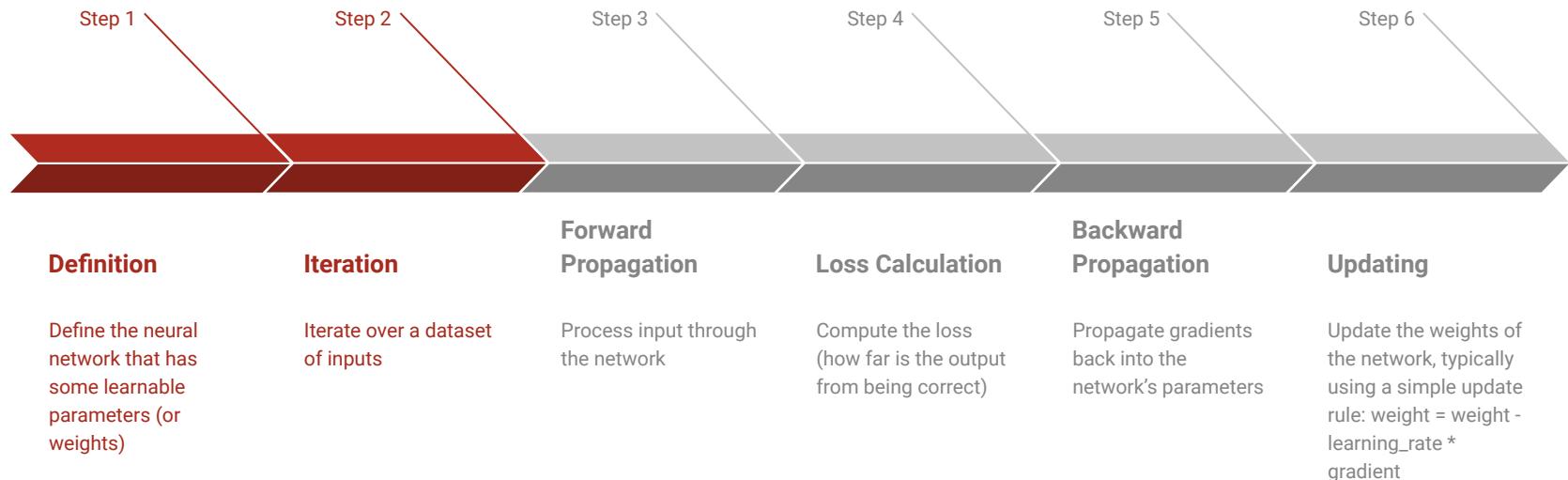
# define a neural network model
class Net(nn.Module):

    def __init__(self, param):
        super(Net, self).__init__()
        self.param = param

    def forward(self, x):
        return x * self.param

net = Net(torch.Tensor([3, 4, 5]))
print(net)
```

# Procedure to Train a Neural Network - Given a Data Set



# Train a Neural Network - torch.nn

- **Define** the neural network that has some learnable parameters.
- **Iterate** over a dataset of inputs
- **Process** input through the network
- **Compute the loss** (how far is the output from being correct)
- **Propagate gradients back** into the network's parameters
- **Update the weights** of the network.

```
import torch.optim as optim

# Net is a predefined nn model
net = Net(torch.Tensor([3, 4, 5]))
output = net(input)

# define a dummy target
target = torch.randn(10)
target = target.view(1, -1)
criterion = nn.MSELoss()
loss = criterion(output, target)

# use one of the update rules such as SGD,
Nesterov-SGD, Adam, RMSProp, etc
optimizer = optim.SGD(net.parameters(),
lr=0.01)

# zero the gradient buffers
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

# Preparing Datasets for PyTorch

In order to train a decent deep neural network model with PyTorch, the input data sets needs to be **cleaned, balanced, transformed, scaled, and splitted.**

- Balance the classes. Unbalanced classes will interfere with training.
- Transform the categorical variables into one-hot encoded variables.
- Extract the X (variables) and y (targets) values for the training and testing datasets.
- Scale/normalize the variables.
- Shuffle and split the dataset into training and testing datasets

One-hot encoding

Dog	Cat	Horse
1	0	0
0	1	0
0	0	1

Numerical encoding

Dog	Cat	Horse
1	2	3

# Predefined Datasets in torchvision

The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision. The datasets include but not limited to MNIST, Fashion-MNIST, ImageNet, CIFAR, etc. They all have two common arguments:

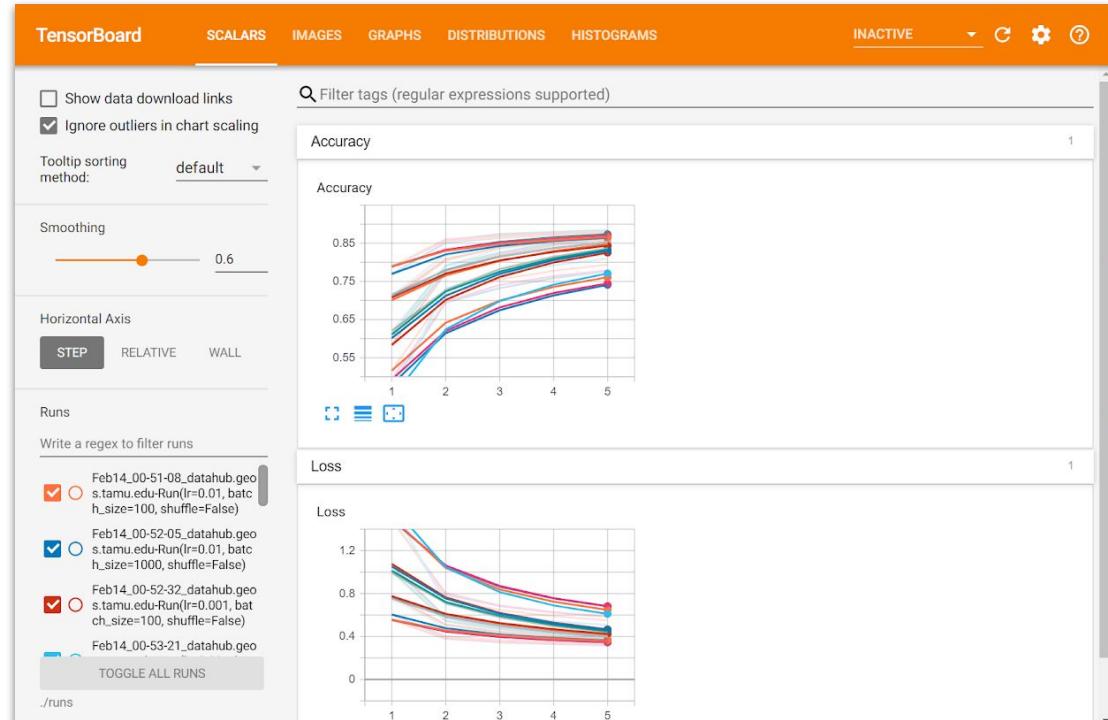
- **transform** to transform the input.
- **target\_transform** to transform the target

The datasets can all be passed to a **torch.utils.data.DataLoader**, which can load multiple samples parallelly using **torch.multiprocessing** workers.

```
from torchvision import datasets  
  
# import ImageNet data set  
imagenet_data =  
datasets.ImageNet('./imagenet')  
  
data_loader =  
torch.utils.data.DataLoader(  
    imagenet_data,  
    batch_size=4,  
    shuffle=True,  
    num_workers=args.nThreads)
```

# Monitoring Training with Tensorboard

- TensorBoard is a User Interface (UI) tools designed for TensorFlow.
- More details on TensorBoard can be found at [TensorBoard](#).
- Once you've installed TensorBoard, these utilities let you log PyTorch models and metrics into a directory for visualization within the TensorBoard UI.



# **Hands-on Session #2**

## **Getting Started with PyTorch**



# Hands-on Session #3

## Classify Fashion-MNIST with PyTorch



- Fashion-MNIST is a dataset of Zalando's article images
- consisting of a training set of 60,000 examples and a test set of 10,000 examples.
- Each example is a 28x28 grayscale image, associated with a label from 10 classes.