

# Introduction to Deep Learning with TensorFlow

**Jian Tao**

[jtao@tamu.edu](mailto:jtao@tamu.edu)

Spring 2020 HPRC Short Course

03/27/2020



High Performance  
Research Computing  
DIVISION OF RESEARCH



TEXAS A&M  
Institute of  
Data Science

# Schedule

- **Part I. Deep Learning (70 mins)**
- **Break (10 mins)**
- **Part II. Intro to TensorFlow (70 mins)**



# GitHub Repository for the Webinars

<https://github.com/jtao/dswebinar>

The screenshot displays the GitHub interface for the repository `jtao/dswebinar`. At the top, there is a search bar and navigation links for Pull requests, Issues, Marketplace, and Explore. Below this, the repository name is shown along with interaction buttons for Unwatch (1), Star (0), and Fork (0). A secondary navigation bar includes Code, Issues (0), Pull requests (0), Projects (0), Wiki, Security, Insights, and Settings.

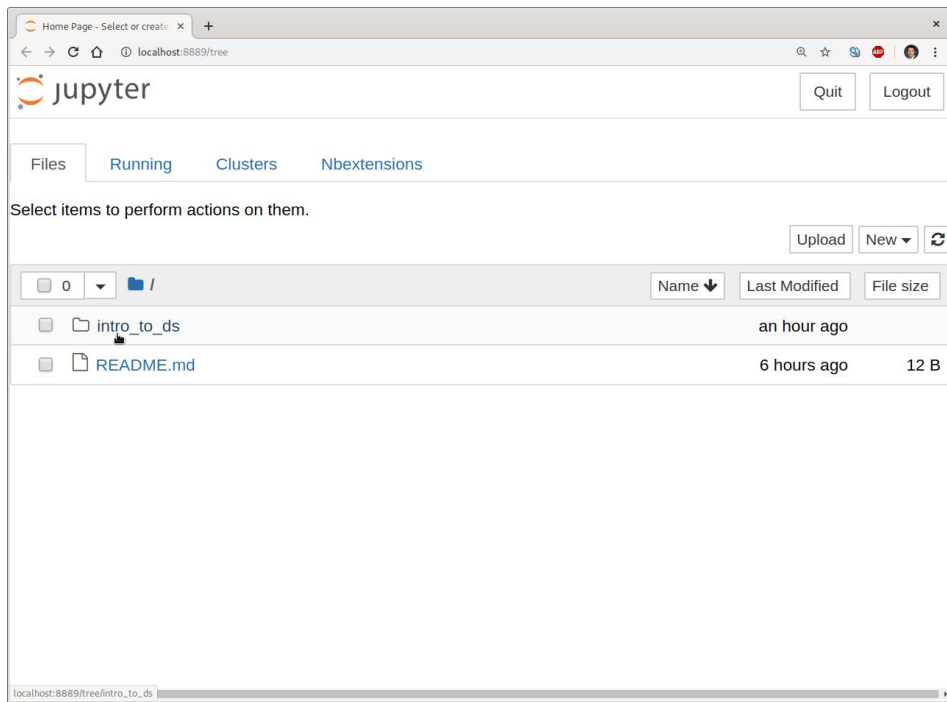
The main content area is titled "Data Science Webinars" and includes an "Edit" button. Below the title, it shows repository statistics: 15 commits, 1 branch, 0 releases, and 1 contributor. A "Clone or download" button is highlighted, which has opened a modal. The modal offers two options: "Clone with HTTPS" (with a "Use SSH" link) and "Download ZIP". The HTTPS URL provided is `https://github.com/jtao/dswebinar.git`.

The file tree below the modal shows the following structure:

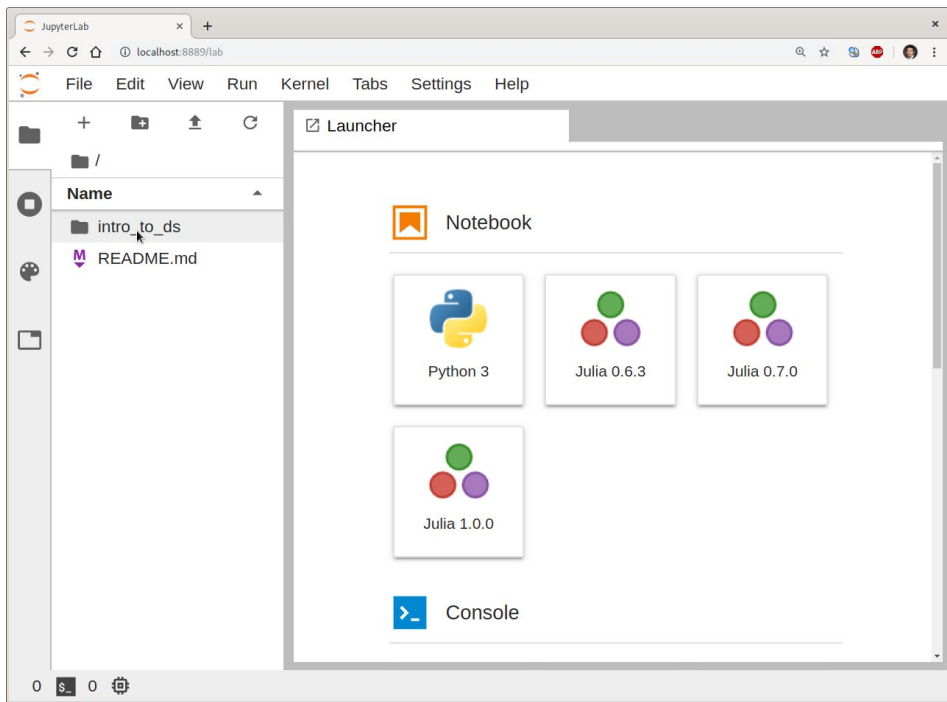
- `intro_to_ds` (removed file.s)
- `README.md` (first commit)

The repository description at the bottom reads "dswebinar". The URL at the bottom of the browser window is `https://github.com/jtao/dswebinar/archive/master.zip`.

# Jupyter Notebook and JupyterLab



Jupyter Notebook



JupyterLab

# Google Colaboratory

The image shows a browser window displaying the Google Colaboratory interface. The address bar shows the URL `colab.research.google.com/notebooks/welcome.ipynb`. The page title is "Welcome To Colaboratory". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". A sidebar on the left contains a "+ Code" button and a "Table of Contents" section with items like "Introduction", "Getting started", "More Resources", and "Machine Learning". The "File" menu is open, showing options such as "New Python 3 notebook", "New Python 2 notebook", "Open notebook...", "Upload notebook...", "Rename...", "Move to trash", "Save a copy in Drive...", "Save a copy as a GitHub Gist...", "Save a copy in GitHub...", "Save", "Save and pin revision", "Revision history", "Download .ipynb", "Download .py", "Update Drive preview", and "Print". The main content area features a "Welcome to Colaboratory!" message, a description of the environment, and a video titled "Intro to Google Colab" by Coding TensorFlow. The video thumbnail shows a man smiling and the text "Intro to Google Colab" with a play button icon. The video title is "Get started with Google Colaboratory (Coding...)" and it includes "Watch later" and "Share" buttons. The "Coding TensorFlow" logo is visible at the bottom left of the video.

# Google Colaboratory

Search GitHub user: **jtao/dswebinar**




EXAMPLES    RECENT    GOOGLE DRIVE    **GITHUB**    UPLOAD

Enter a GitHub URL or search by organization or user  Include private repos

jtao/dswebinar

Repository:  Branch:

Path

-  intro\_to\_ds/case1.1/exploration\_example.ipynb
-  intro\_to\_ds/case1.2/reproduce\_plot.ipynb
-  intro\_to\_ds/case1.3/a2z\_ds\_example.ipynb

[NEW PYTHON 3 NOTEBOOK](#)  [CANCEL](#)



# Part I. Deep Learning

## ***Deep Learning***

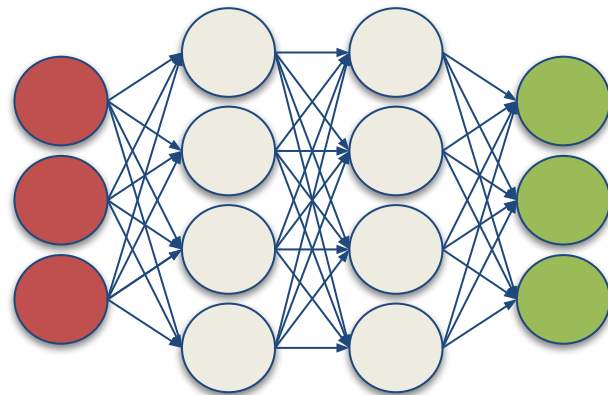
by Ian Goodfellow, Yoshua Bengio, and Aaron Courville

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

## ***Animation of Neutron Networks***

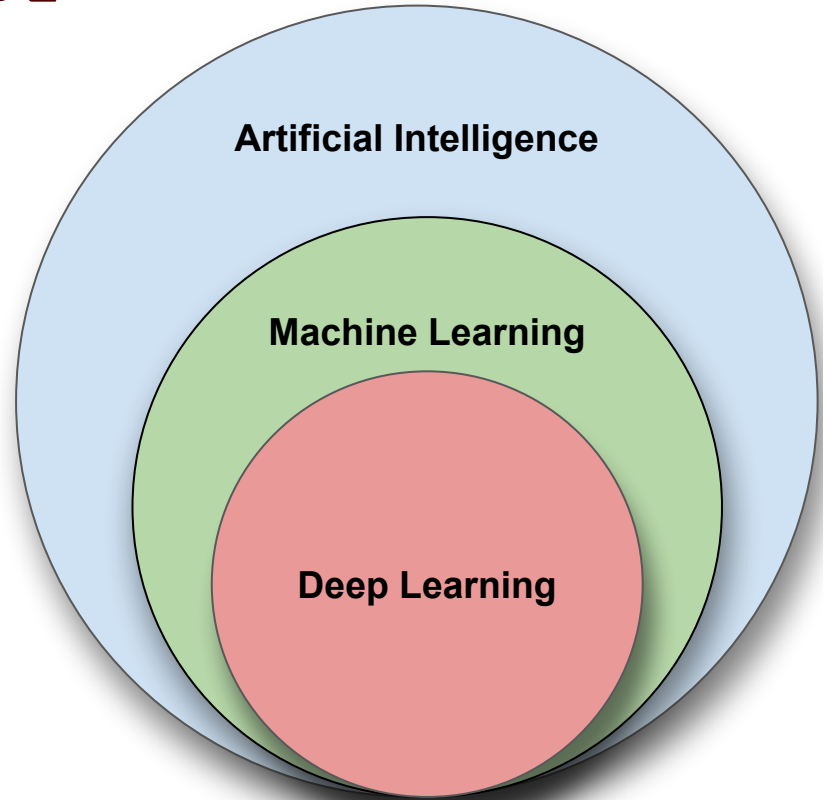
by Grant Sanderson

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



# Relationship of AI, ML, and DL

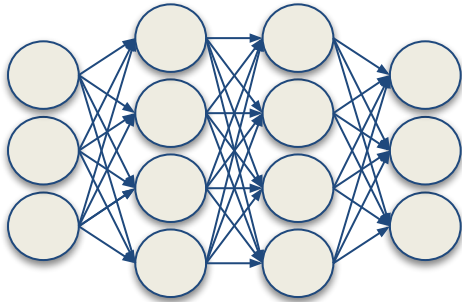
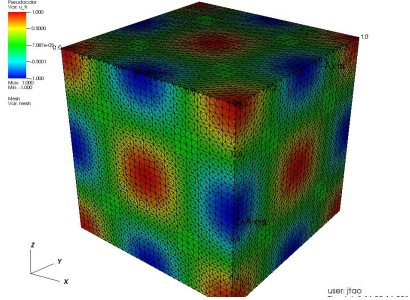
- **Artificial Intelligence (AI)** is anything about man-made intelligence exhibited by machines.
- **Machine Learning (ML)** is an approach to achieve **AI**.
- **Deep Learning (DL)** is one technique to implement **ML**.



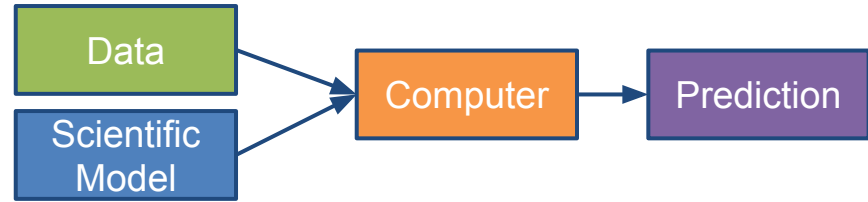


# Machine Learning

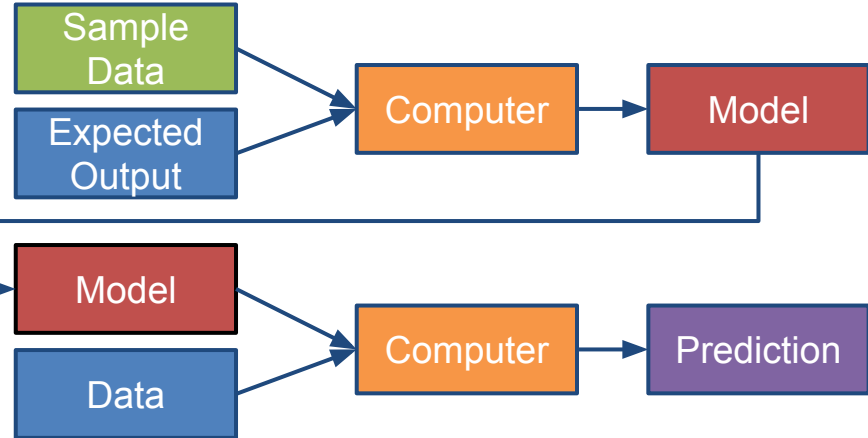
DB: simplest.vtk



## Traditional Modeling

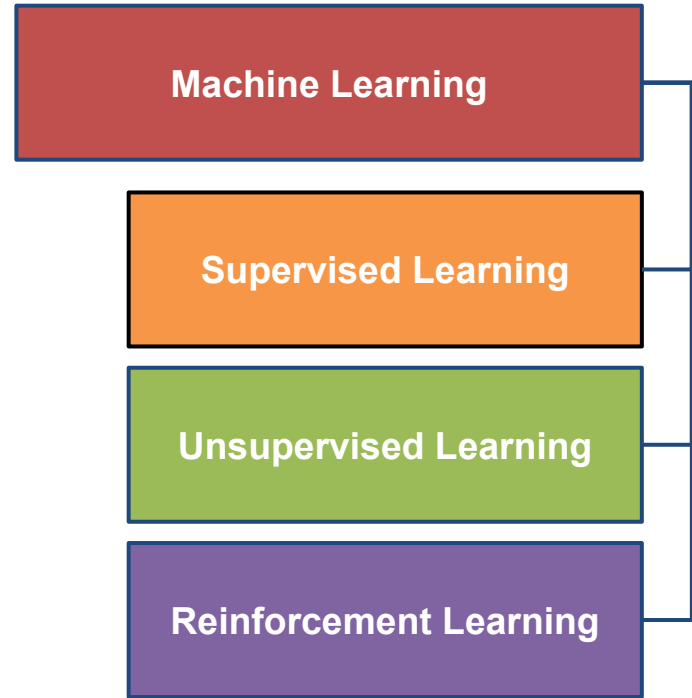


## Machine Learning (Supervised Learning)



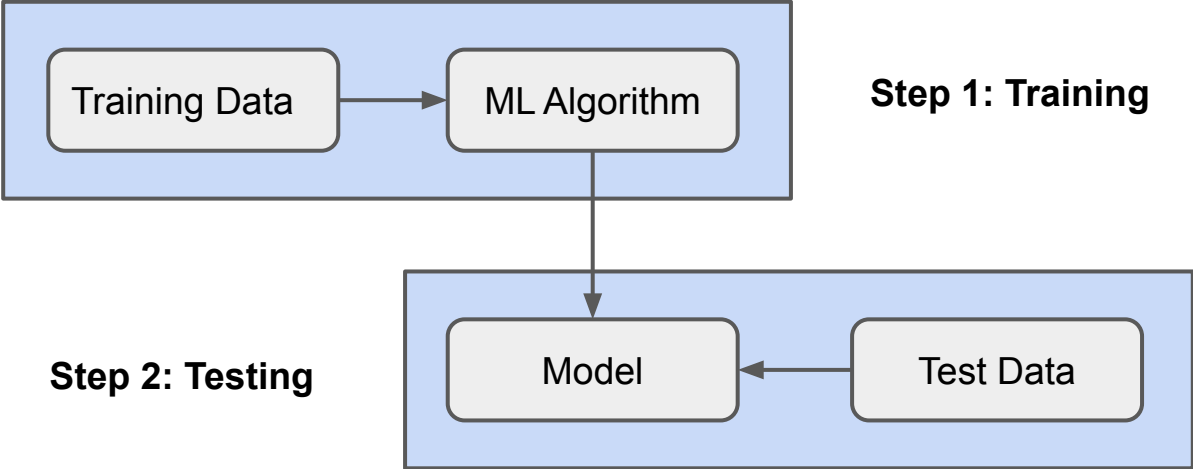
# Types of ML Algorithms

- **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.



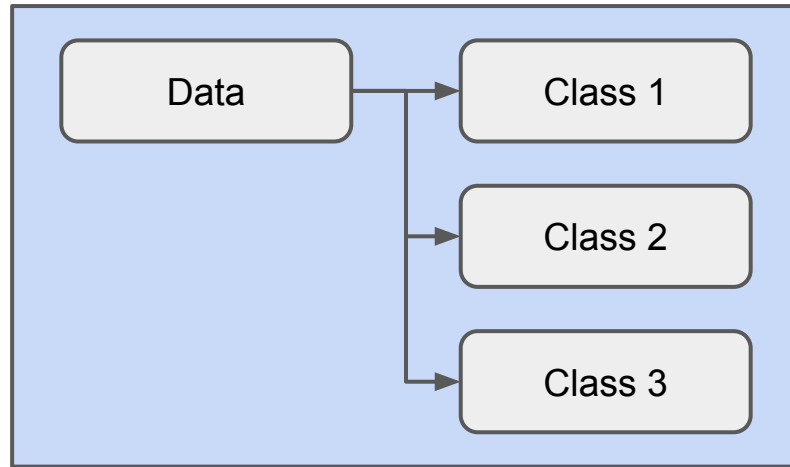
# Supervised Learning

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



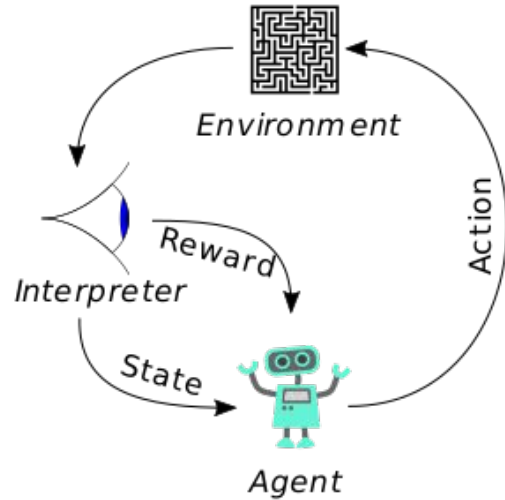
# Unsupervised Learning

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

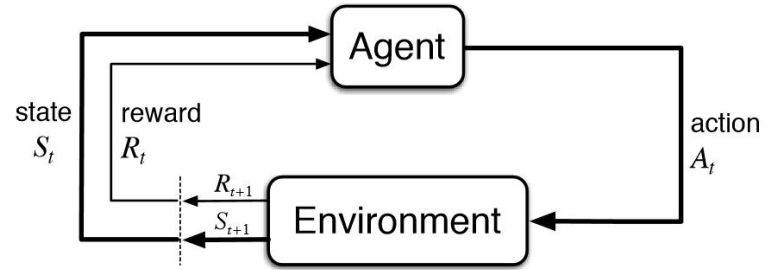


# Reinforcement Learning

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?

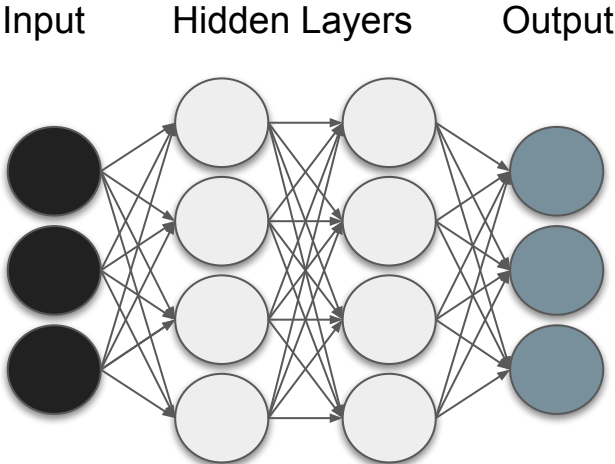
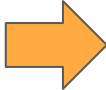
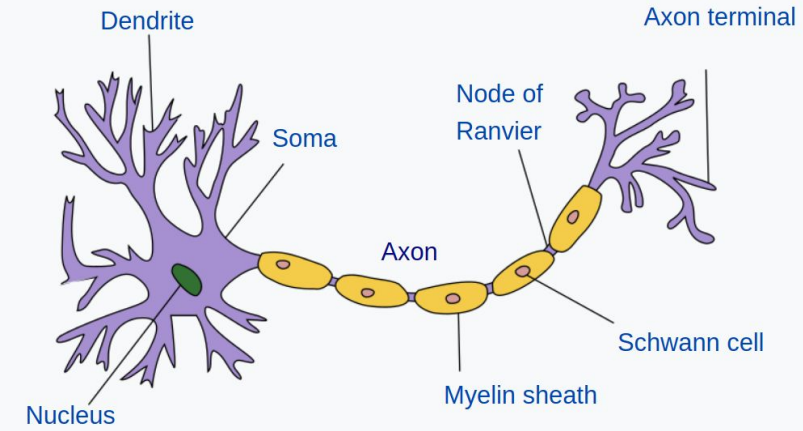
Deep learning is a class of machine learning algorithms that:

- 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)

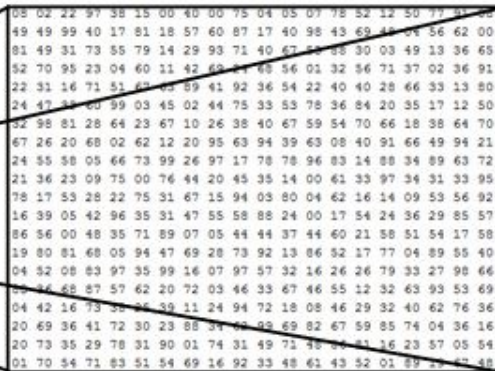


# Artificial Neural Network



(Image Credit: Wikipedia)

# Inputs and Outputs



What the computer sees

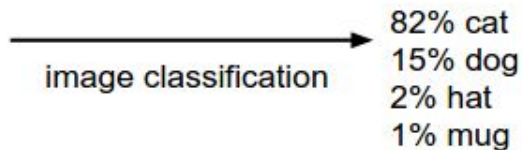
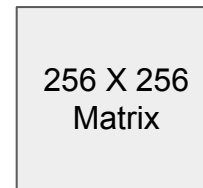
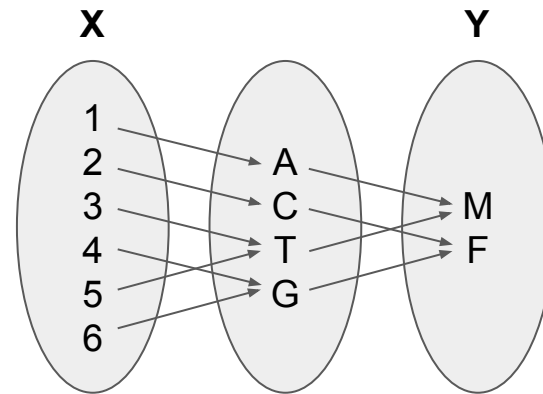


Image from the [Stanford CS231 Course](#)



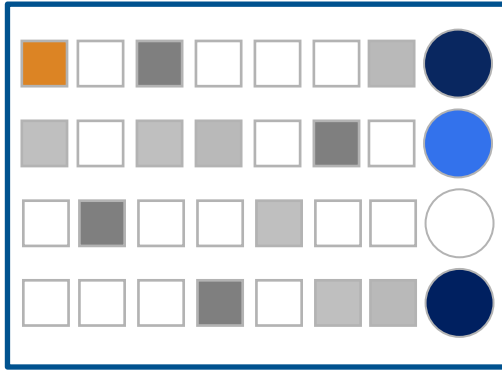
DL model

4-Element Vector

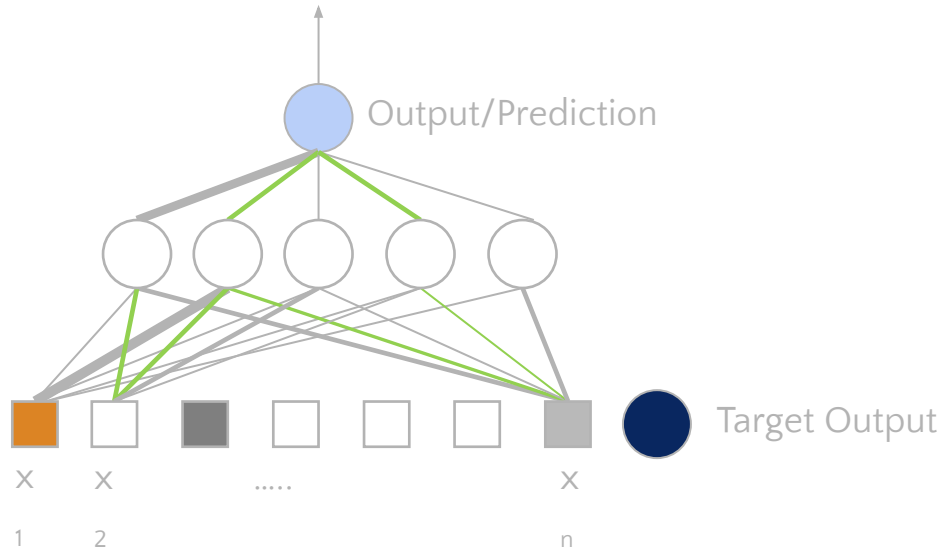


With deep learning, we are searching for a **surjective** (or **onto**) function  $f$  from a set  $X$  to a set  $Y$ .

## Dataset

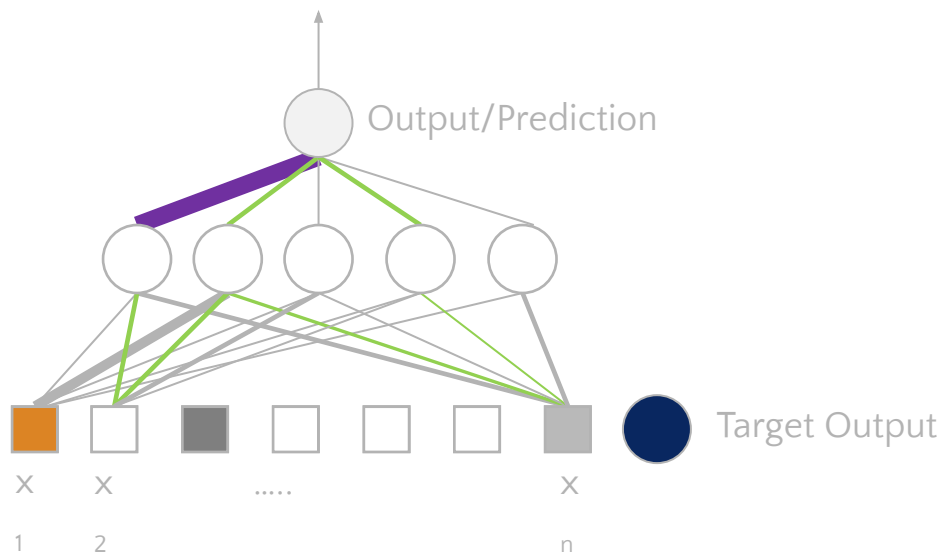


# Learning Principle



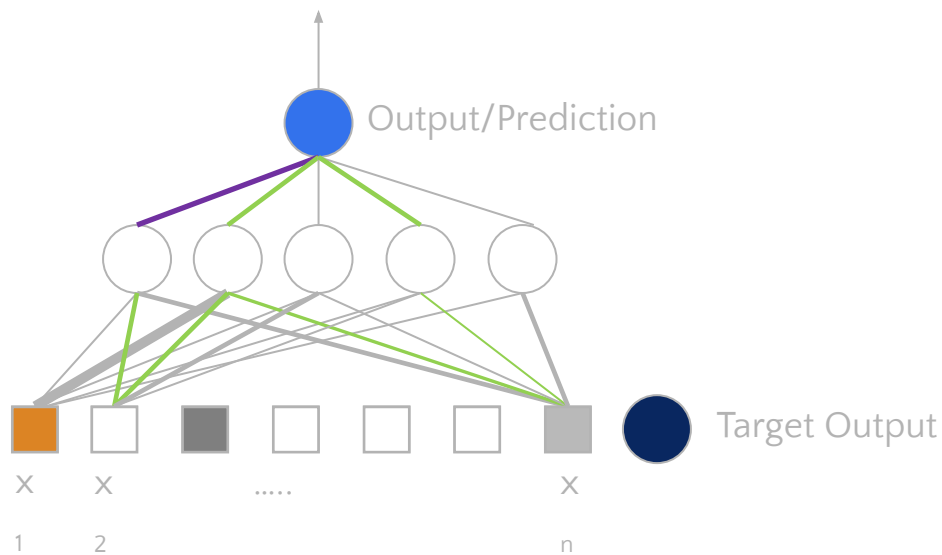
Error:  -  = 5

# Learning Principle



Error:  -  = 15

# Learning Principle



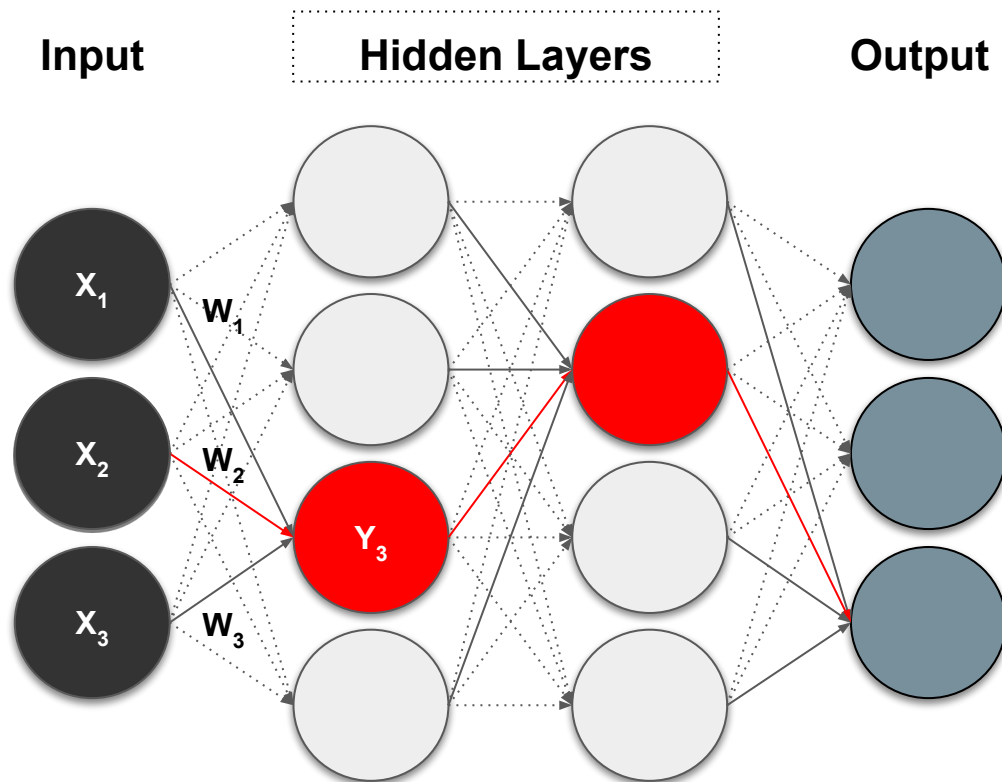
Error:  -  = 2.5

# 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,  
 $W_i$  is the weight, and  $b_i$  is  
the bias.



# Training - Minimizing the Loss

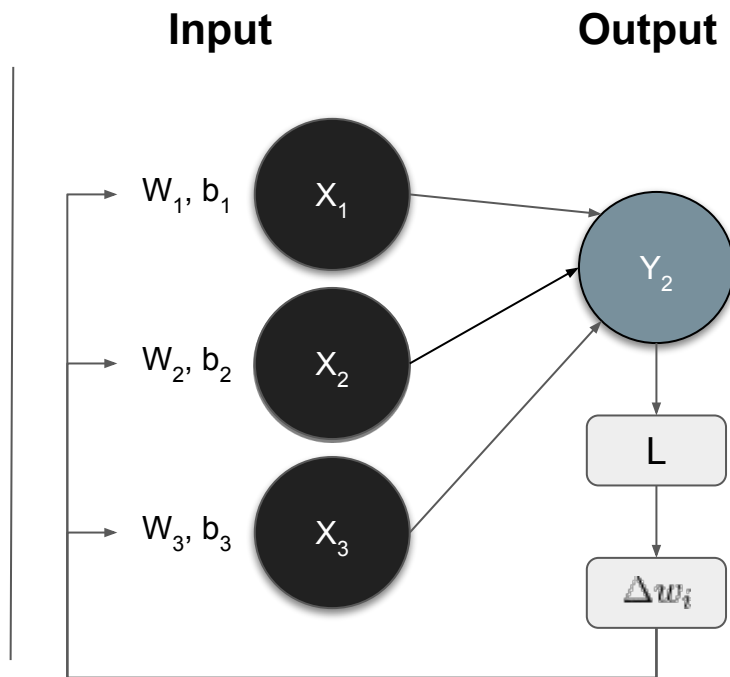
The loss function with regard to weights and biases can be defined as

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

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

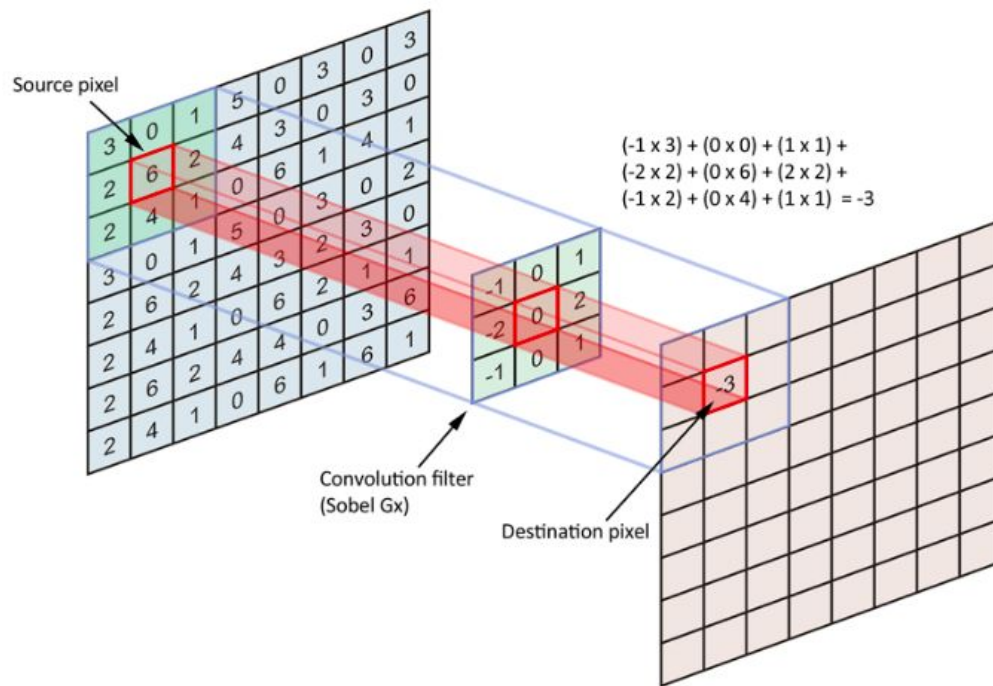
$$\Delta w_i = -\alpha \frac{\partial L}{\partial w_i}$$

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



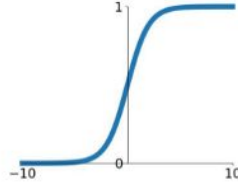
Input

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

# Activation Functions

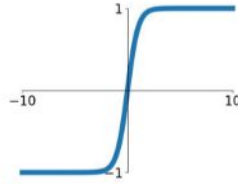
**Sigmoid**

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



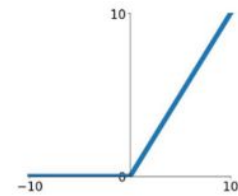
**tanh**

$$\tanh(x)$$



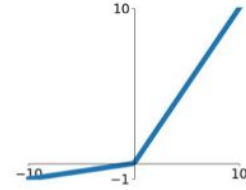
**ReLU**

$$\max(0, x)$$



**Leaky ReLU**

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

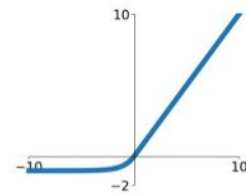


**Maxout**

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

**ELU**

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



# Introducing Non Linearity (ReLU)

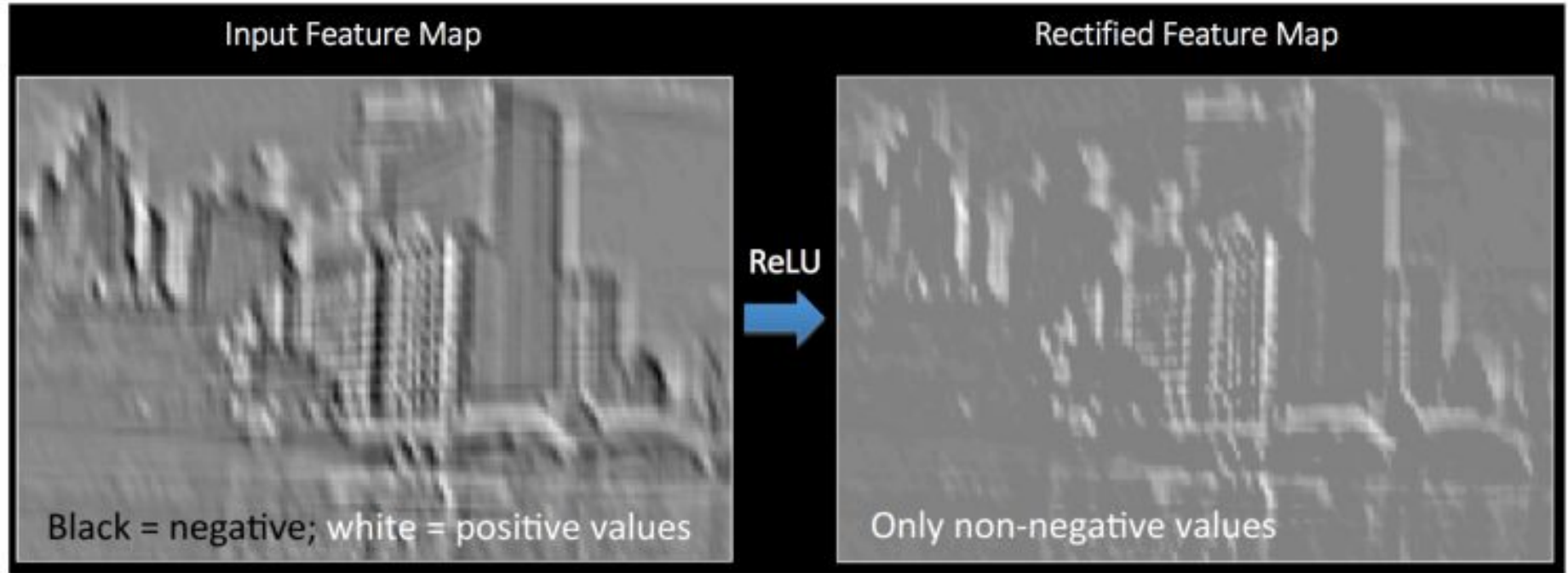
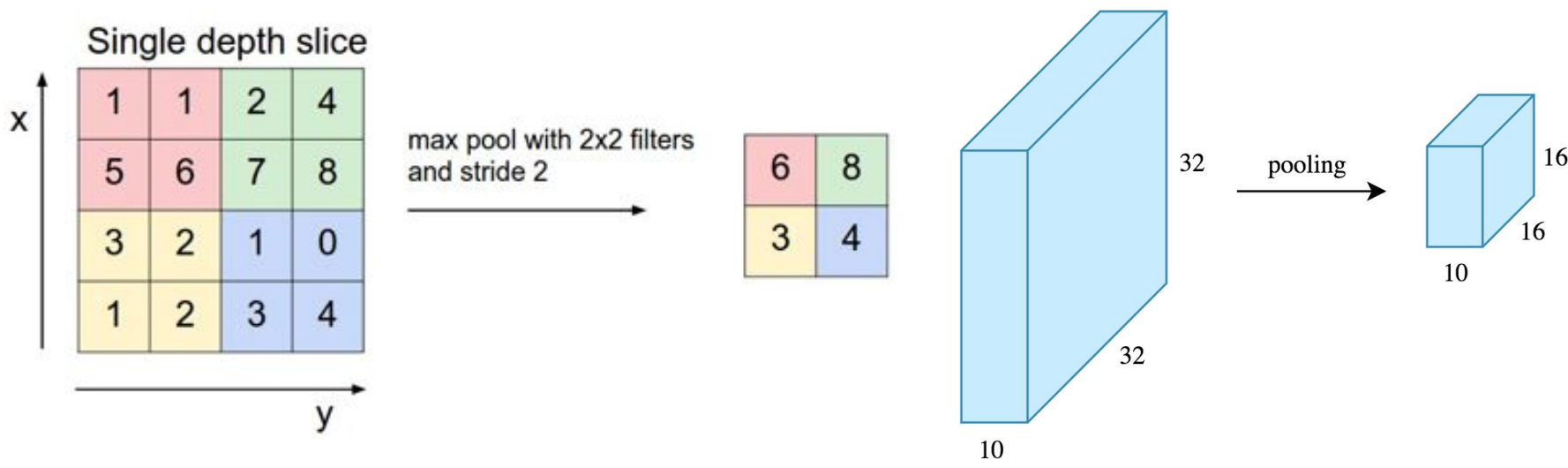


Image Credit: [Deep Learning Methods for Vision | CVPR 2012 Tutorial](#)

# Max Pooling



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

# Pooling - Max-Pooling and Sum-Pooling

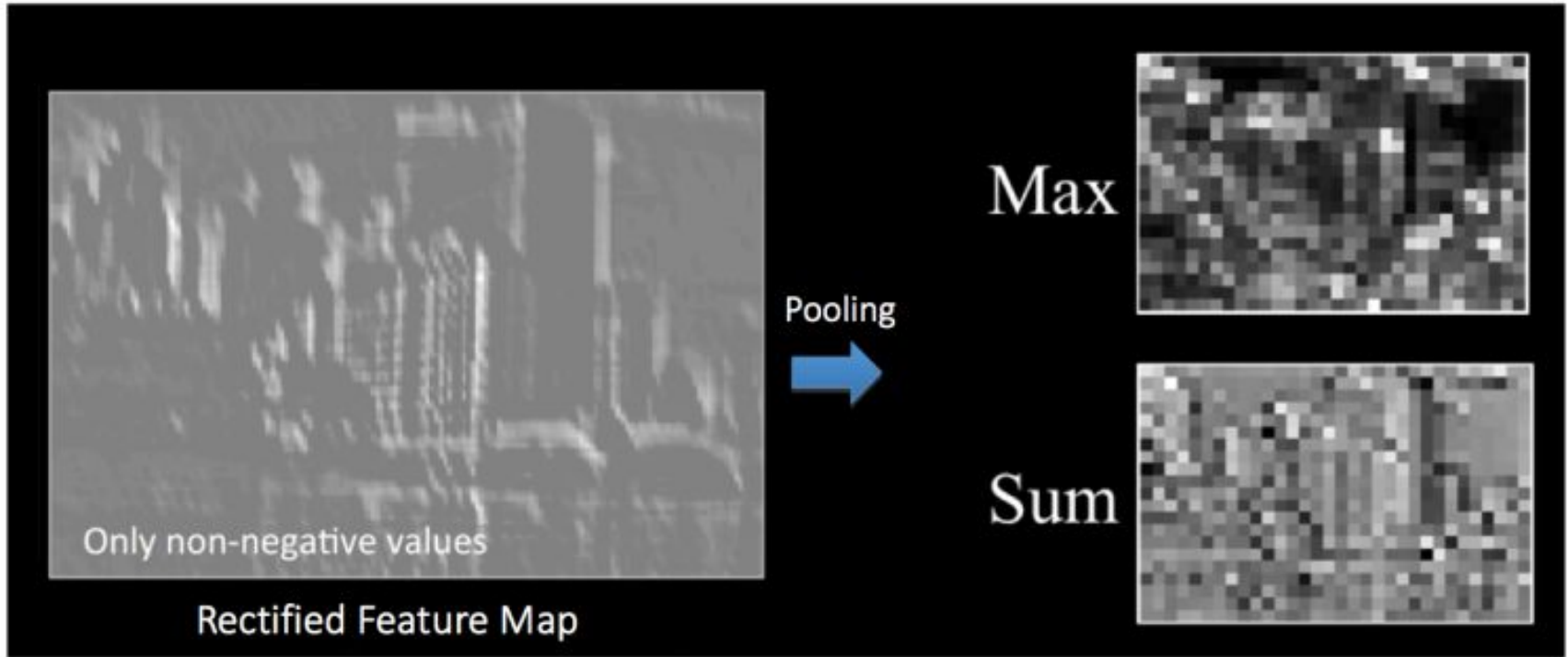
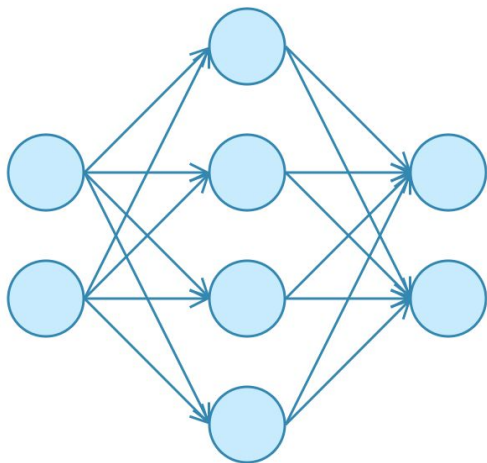


Image Credit: [Deep Learning Methods for Vision | CVPR 2012 Tutorial](#)

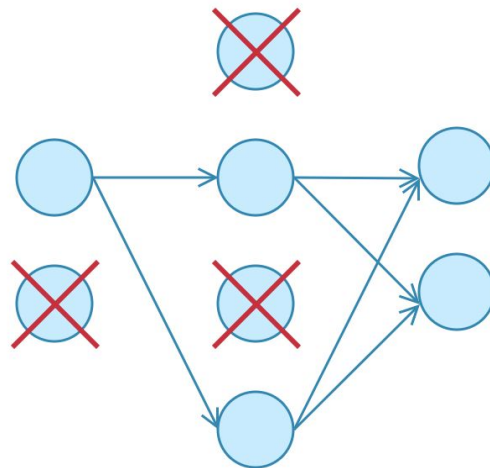


# CNN Implementation - Drop Out

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



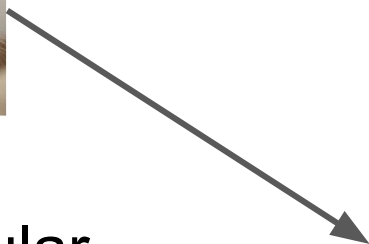
No Dropout



With Dropout

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

# CNN Implementation - Data Augmentation (DA)

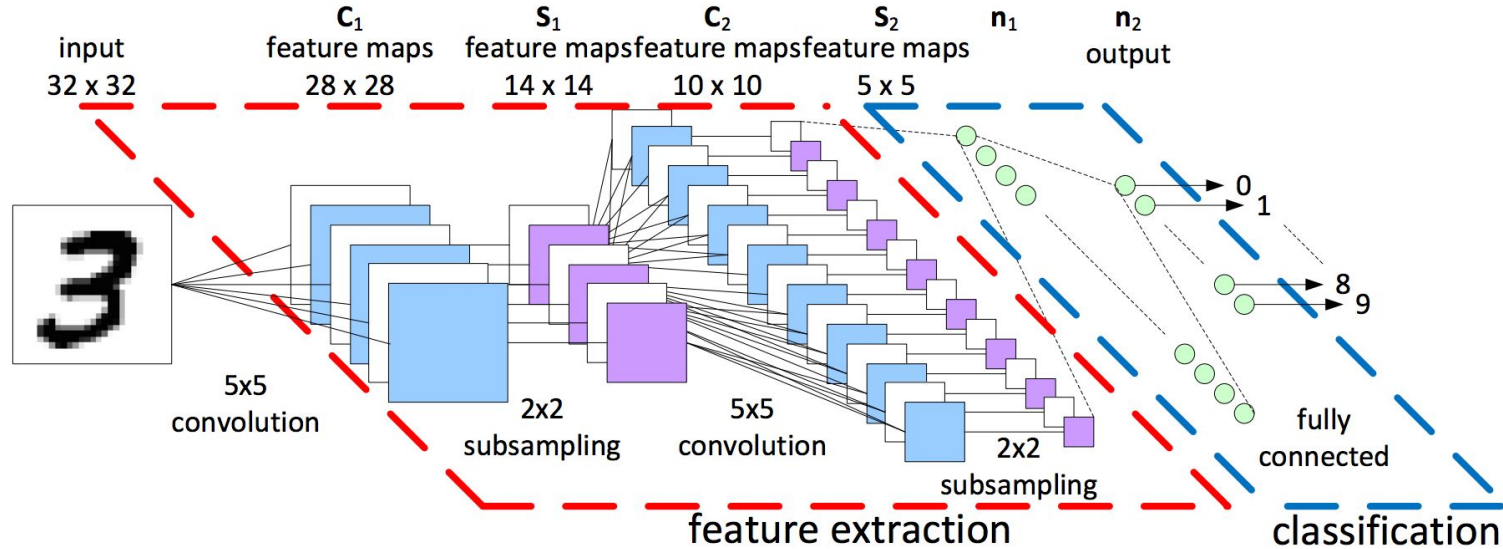


DA helps to popular artificial training instances from the existing train data sets.

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

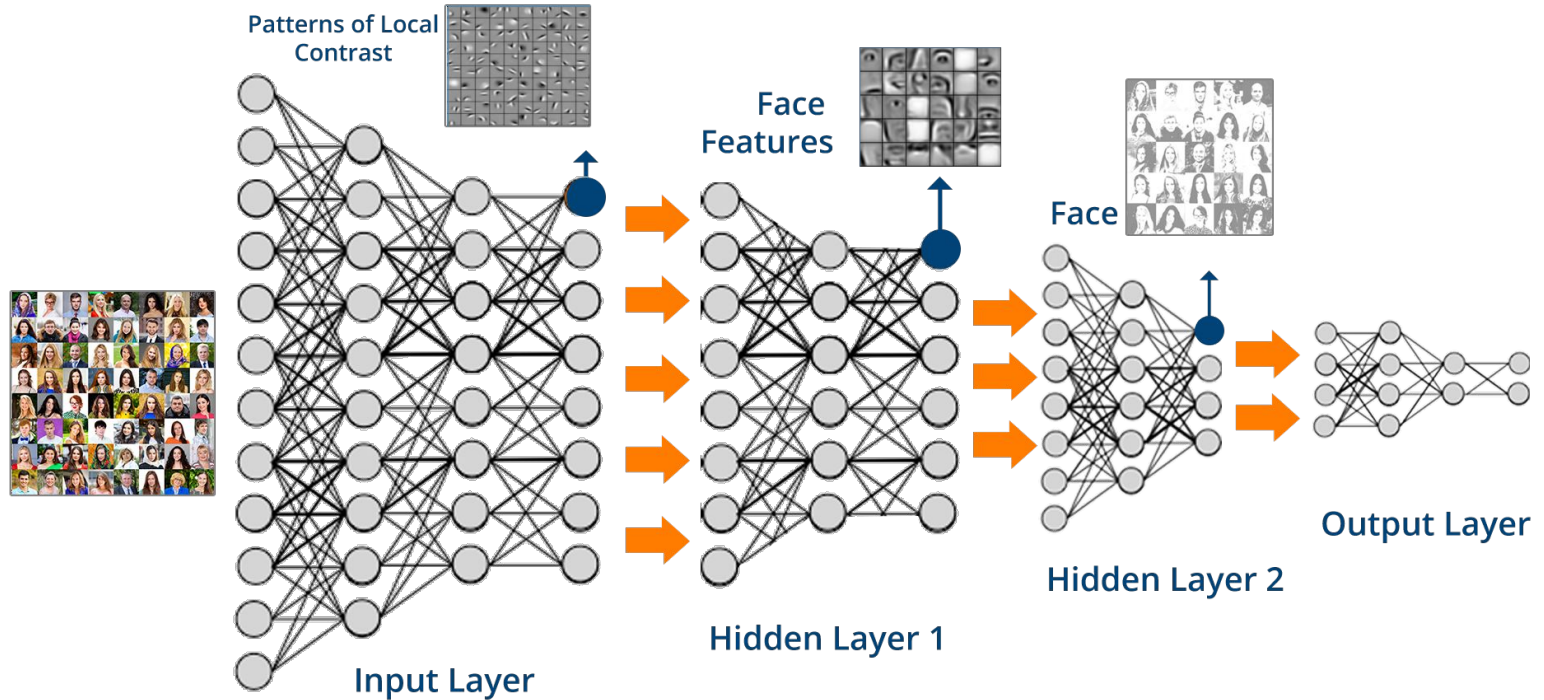
# Convolutional Neural Networks

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.



(Image Credit: <https://becominghuman.ai>)

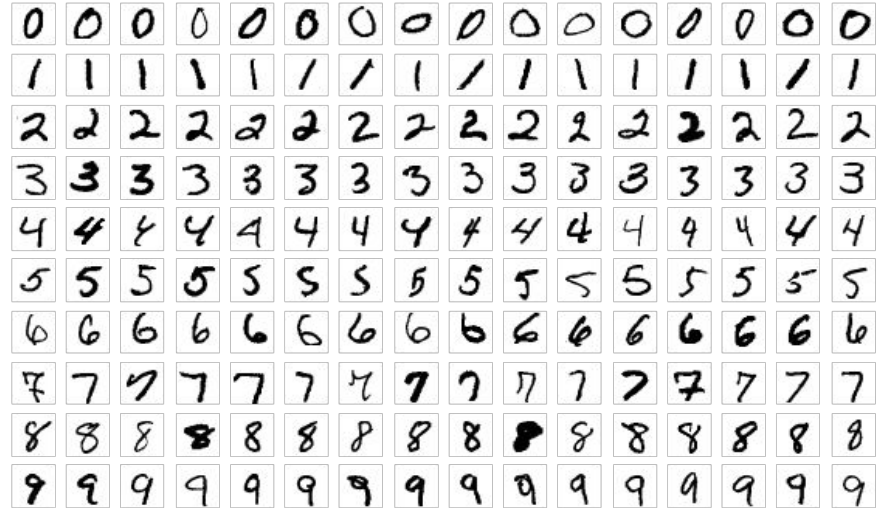
# Deep Learning for Facial Recognition



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

# MNIST - Introduction

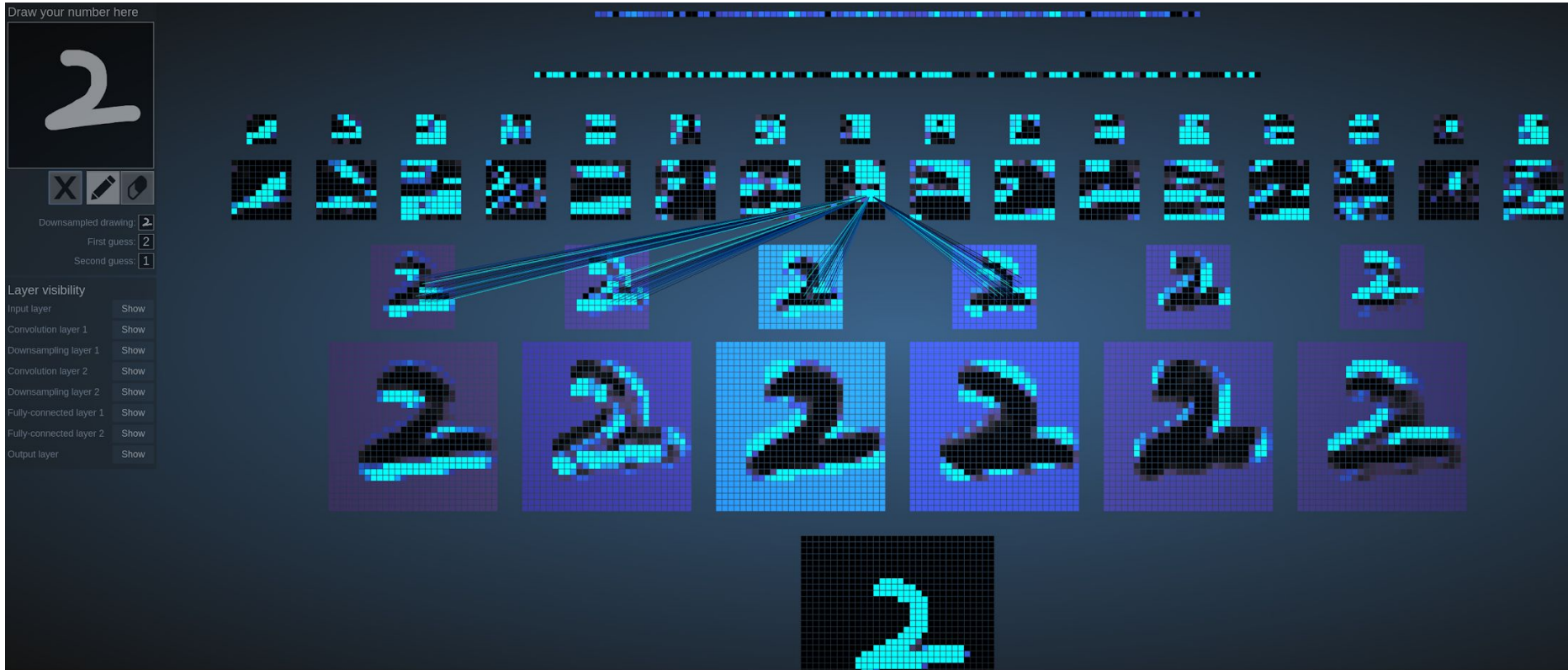
- **MNIST** (Mixed National Institute of Standards and Technology) is a database for handwritten digits, distributed by Yann Lecun.
- 60,000 examples, and a test set of 10,000 examples.
- 28x28 pixels each.
- Widely used for research and educational purposes.



(Image Credit: Wikipedia)



# MNIST - CNN Visualization



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

# Part II. Introduction to TensorFlow

*TensorFlow Official Website*  
<http://www.tensorflow.org>



# A Brief History of TensorFlow

TensorFlow is an end-to-end FOSS (free and open source software) library for dataflow, differentiable programming. TensorFlow is one of the most popular program frameworks for building machine learning applications.

- Google Brain built **DistBelief** in 2011 for internal usage.
- TensorFlow 1.0.0 was released on Feb 11, 2017
- TensorFlow 2.0 was released in Jan 2018.



# TensorFlow, Keras, and PyTorch

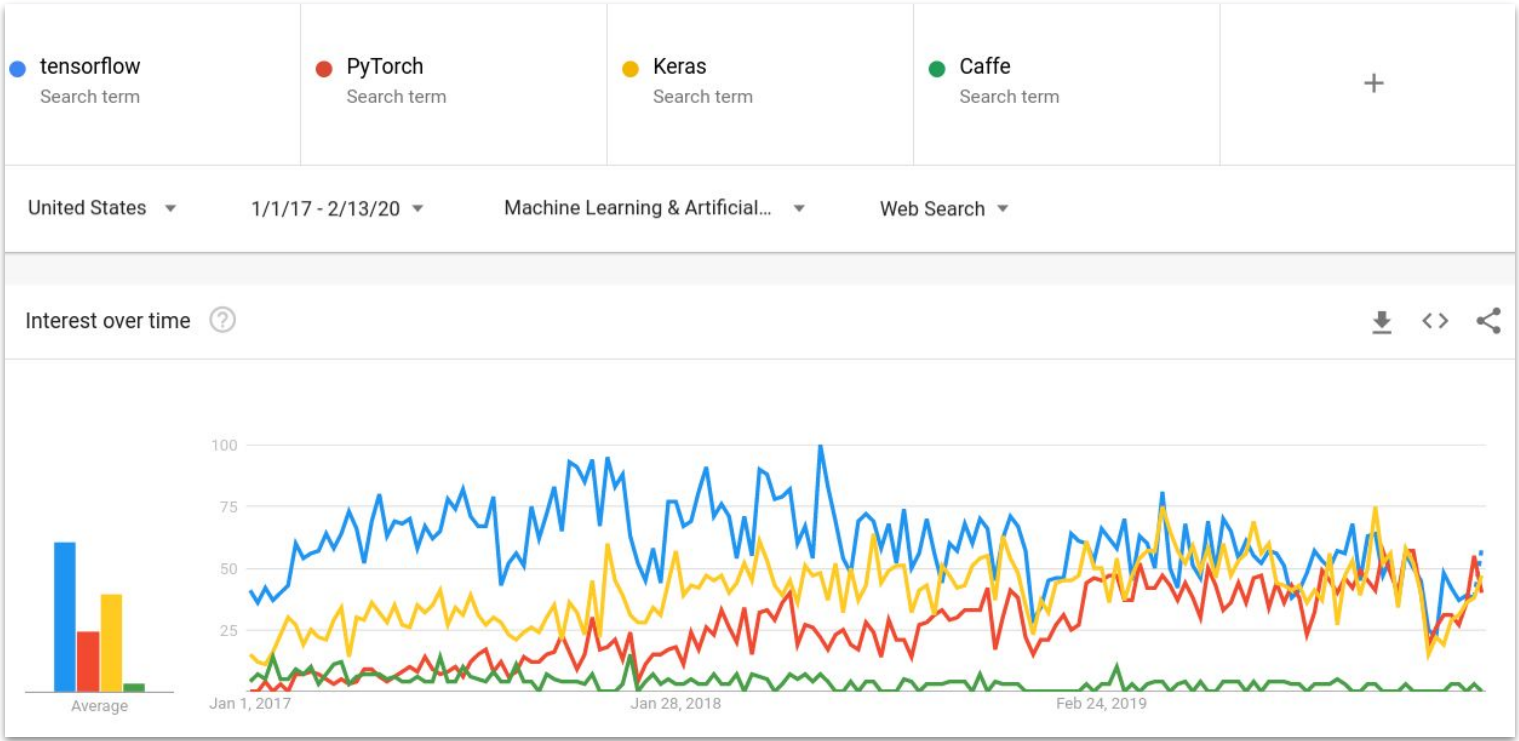


**TensorFlow** is an end-to-end open source **platform** for machine learning. It has a comprehensive, flexible ecosystem to build and deploy ML powered applications.

**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.

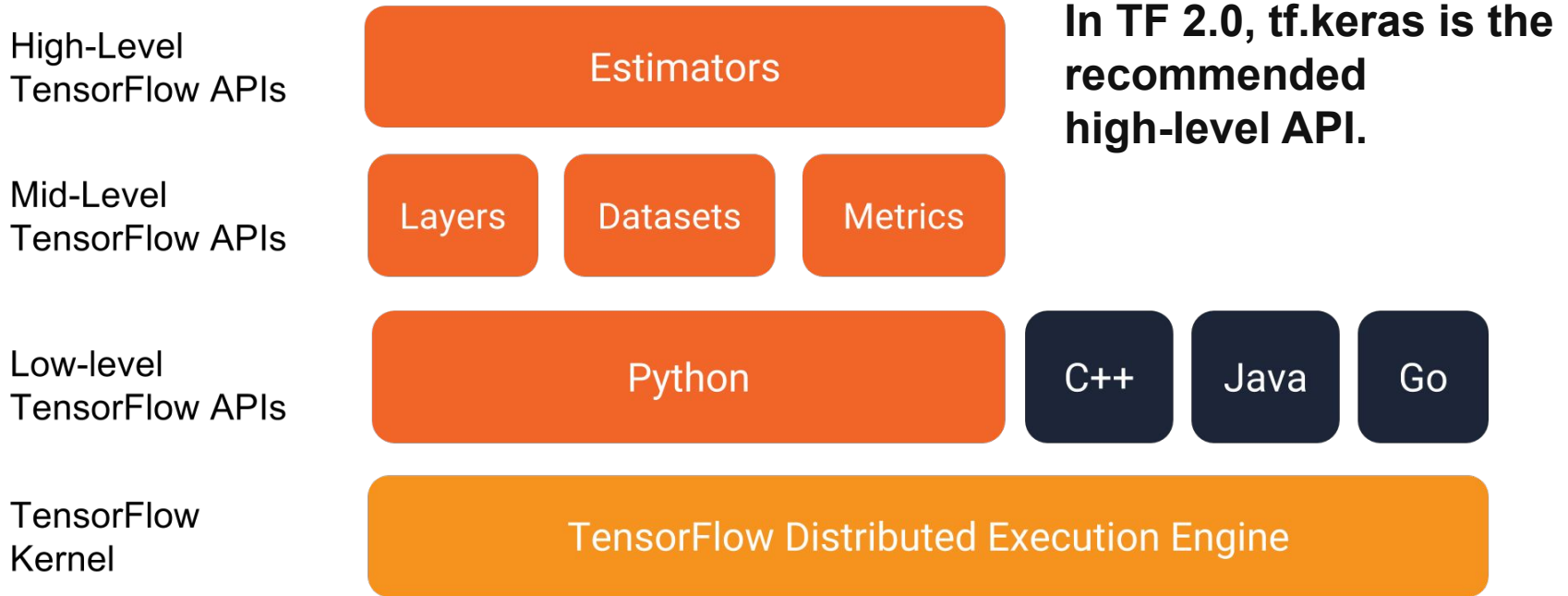
**PyTorch** is an open source machine learning **framework** that accelerates the path from research prototyping to production deployment.

# Google Trends for Popular ML Frameworks



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

# Programming Environment



(Image Credit: tensorflow.org)

# A Connected Pipeline for the Flow of Tensors



(Image Credit: Plumber Game by Mobiloids)

# What is a Tensor in TensorFlow?

- **TensorFlow** uses a tensor data structure to represent all data. A TensorFlow tensor as an n-dimensional array or list. 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	...

# TensorFlow Data Types

Basic TensorFlow data types include:

- **int[8|16|32|64], float[16|32|64], double**
- **bool**
- **string**

with **tf.cast()**, the data types of variables could be converted.

# Hello World with TensorFlow

```
import tensorflow as tf

v = tf.constant("Hello World!")

tf.print(v)
```

# TensorFlow Constants

TensorFlow provides several operations to generate constant tensor.

```
import tensorflow as tf

x = tf.constant(1, tf.int32)
zeros = tf.zeros([2, 3], tf.int32)
ones = tf.ones([2, 3], tf.int32)
y = x *(zeros + ones + ones)

tf.print(y)
```



# TensorFlow Variables

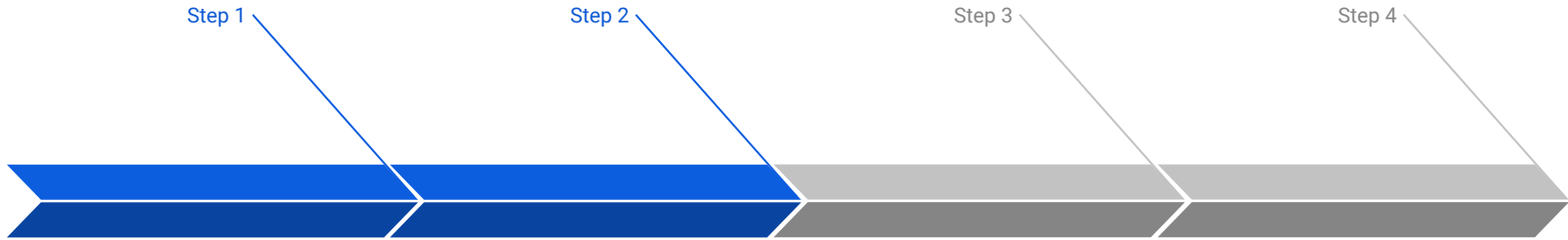
TensorFlow variables can represent shared, persistent state manipulated by your program. **Weights** and **biases** are usually stored in variables.

```
import tensorflow as tf
```

```
W = tf.Variable(tf.random.normal([2,2], stddev=0.1),  
name = "W")
```

```
b = tf.Variable(tf.zeros(shape=(2)), name="b")
```

# Machine Learning Workflow with tf.keras



## Prepare Train Data

The preprocessed data set needs to be shuffled and splitted into training and testing data.

## Define Model

A model could be defined with `tf.keras Sequential` model for a linear stack of layers or `tf.keras functional API` for complex network.

## Training Configuration

The configuration of the training process requires the specification of an optimizer, a loss function, and a list of metrics.

## Train Model

The training begins by calling the fit function. The number of epochs and batch size need to be set. The measurement metrics need to be evaluated.

# tf.keras Built-in Datasets

- tf.keras provides many popular reference datasets that could be used for demonstrating and testing deep neural network models. To name a few,
  - Boston Housing (regression)
  - CIFAR100 (classification of 100 image labels)
  - MNIST (classification of 10 digits)
  - Fashion-MNIST (classification of 10 fashion categories)
  - Reuters News (multiclass text classification)
- The built-in datasets could be easily read in for training purpose. E.g.,

```
from tensorflow.keras.datasets import boston_housing  
(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
```

# Prepare Datasets for tf.keras

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

- 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

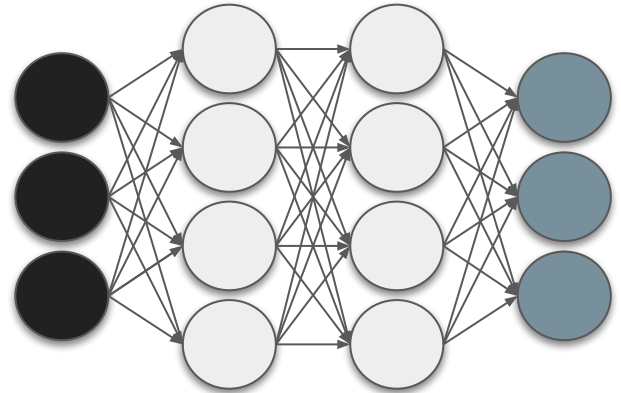
Dog	Cat	Horse
1	2	3

# Create a tf.keras Model

- Layers are the fundamental building blocks of **tf.keras** models.
- The **Sequential** model is a linear stack of layers.
- A **Sequential** model can be created with a list of layer instances to the constructor or added with the **.add()** method.
- The input shape/dimension of the first layer need to be set.

```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense,  
Activation
```

```
model = Sequential(  
    Dense(64, activation='relu', input_dim=20),  
    Dense(10, activation='softmax')  
)
```



Input

Hidden Layers

Output

# Compile a tf.keras Model

The **compile** method of a Keras model configures the learning process before the model is trained. The following 3 arguments need to be set (the optimizer and loss function are required).

- An optimizer: **Adam**, **AdaGrad**, **SGD**, **RMSprop**, etc.
- A loss function: **mean\_squared\_error**, **mean\_absolute\_error**, **mean\_squared\_logarithmic\_error**, **categorical\_crossentropy**, **kullback\_leibler\_divergence**, etc.
- A list of measurement metrics: **accuracy**, **binary\_accuracy**, **categorical\_accuracy**, etc.

# Train and Evaluate a tf.keras Model

**tf.keras** is trained on NumPy arrays of input data and labels. The training is done with the

- **fit()** function of the model class. In the fit function, the following two hyperparameters can be set:
  - **number of epochs**
  - **batch size**
- **evaluate()** function returns the loss value & metrics values for the model in test mode.
- **summary()** function prints out the network architecture.

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
dense_11 (Dense)	(None, 64)	1344
dense_12 (Dense)	(None, 10)	650

Total params: 1,994

Trainable params: 1,994

Non-trainable params: 0

None

# Make Predictions and More

After the model is trained,

- **predict()** function of the model class could be used to generate output predictions for the input samples.
- **get\_weights()** function returns a list of all weight tensors in the model, as Numpy arrays.
- **to\_json()** returns a representation of the model as a JSON string. Note that the representation does not include the weights, only the architecture.
- **save\_weights(filepath)** saves the weights of the model as a HDF5 file.



# **Hands-on Session #1**

## **Getting Started with TensorFlow**



**TensorFlow**

# Hands-on Session #2

## Classify Handwritten Digits with TensorFlow

