Introduction to TensorFlow

Tinghui Wang

Deep Learning in One Slide

What is it?

Extract useful patterns (features) from data

How

Neural Network
Optimization

How (Practical)

Python
Tensorflow
Friends!

Hard Part

Good questions
Good Data

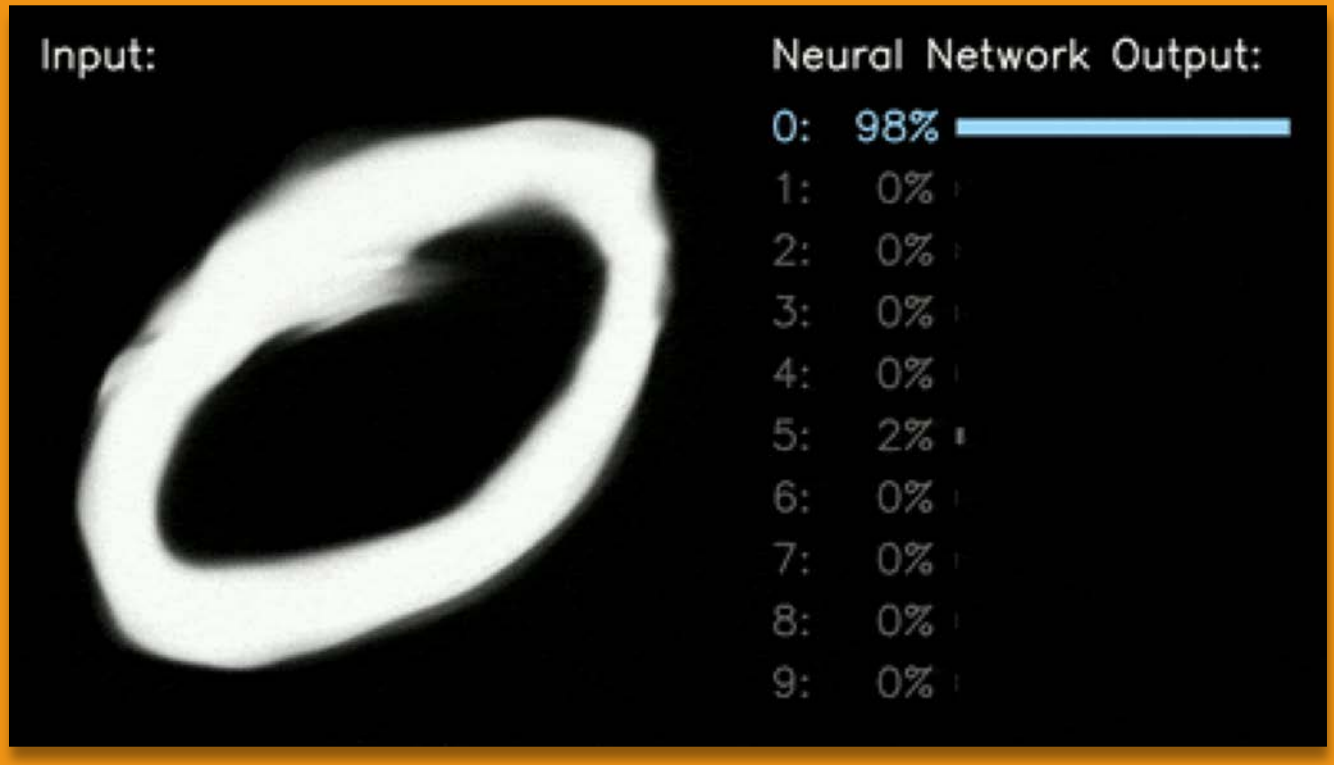
Why now?

Data
Hardware
Community
Tools
Investment

Deep Learning: Exciting Progress

- **Exciting progress:**
  - Face recognition
  - Image classification
  - Speech recognition
  - Text-to-speech generation
  - Handwriting transcription
  - Machine translation
  - Medical Diagnosis
  - Digital assistants
  - Ads, search, social recommendation
  - Game playing with deep RL
  - Auto-drive

Handwriting Digit Classification

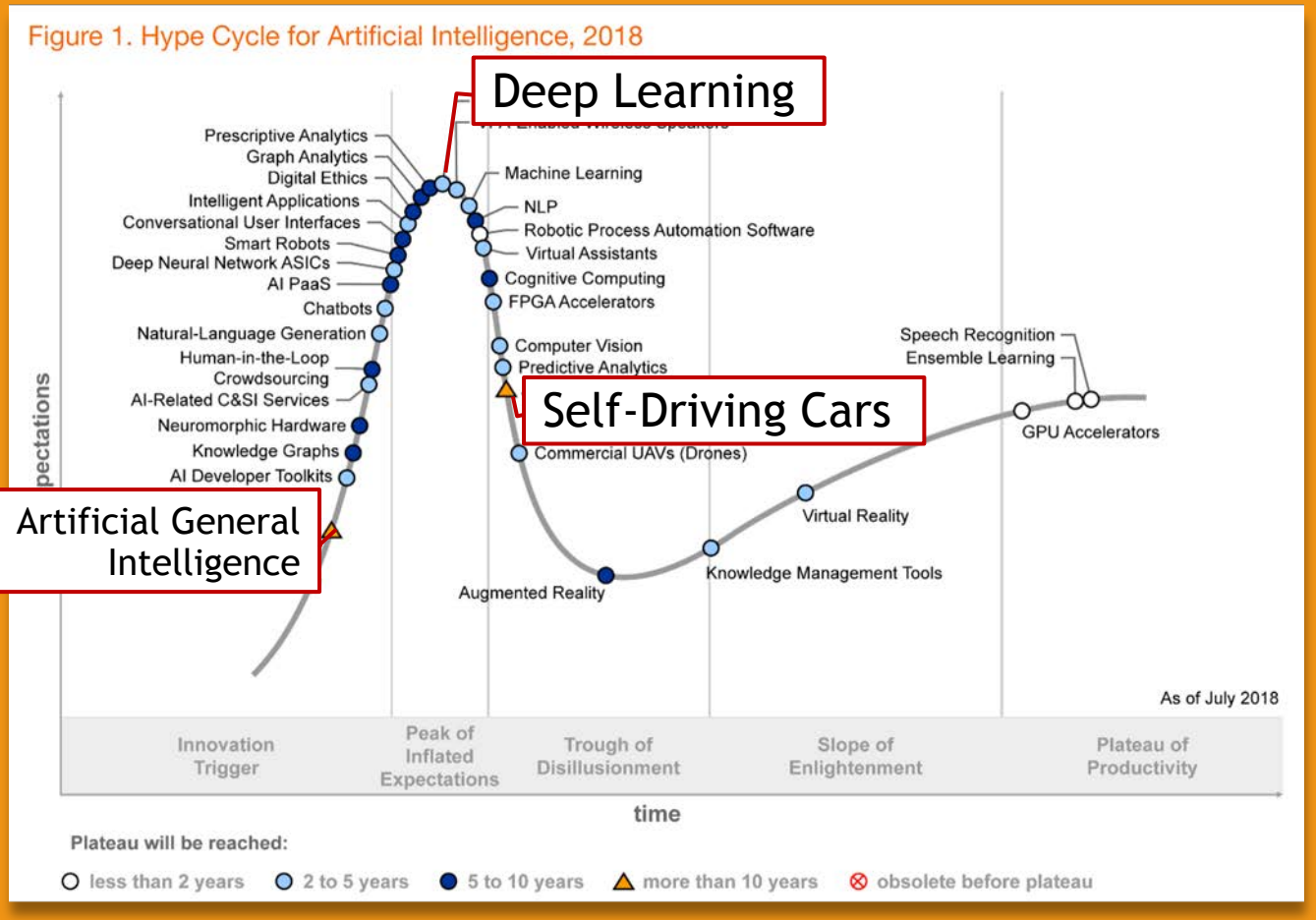


From: Deep Learning Basics, MIT (https://github.com/lexfridman/mit-deep-learning)

Hype Cycle: Artificial Intelligence

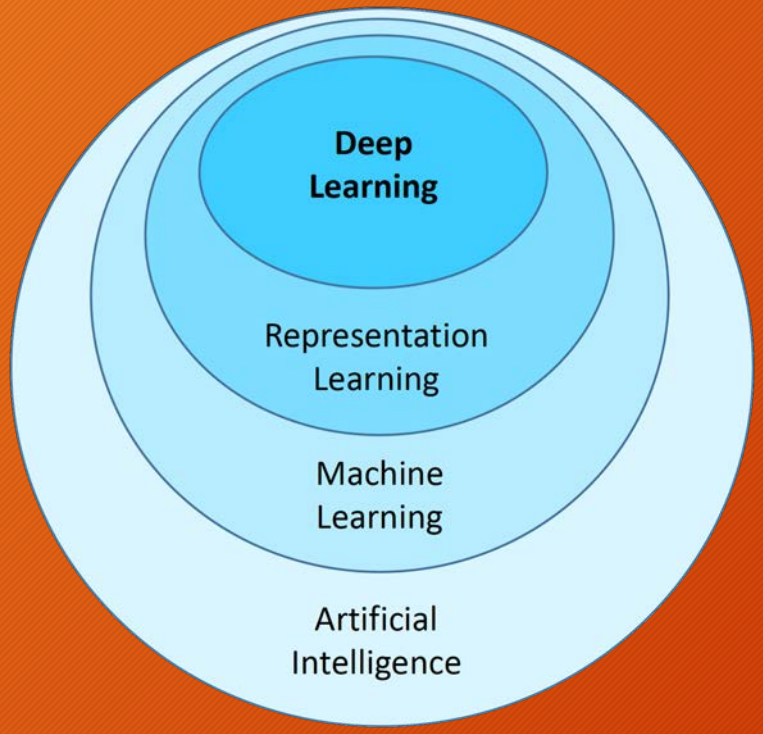
"There's not going to be an AI winter, because it drives your cellphone. In the old AI winters, AI wasn't actually part of your everyday life. Now it is."

-- Geoffery Hinton

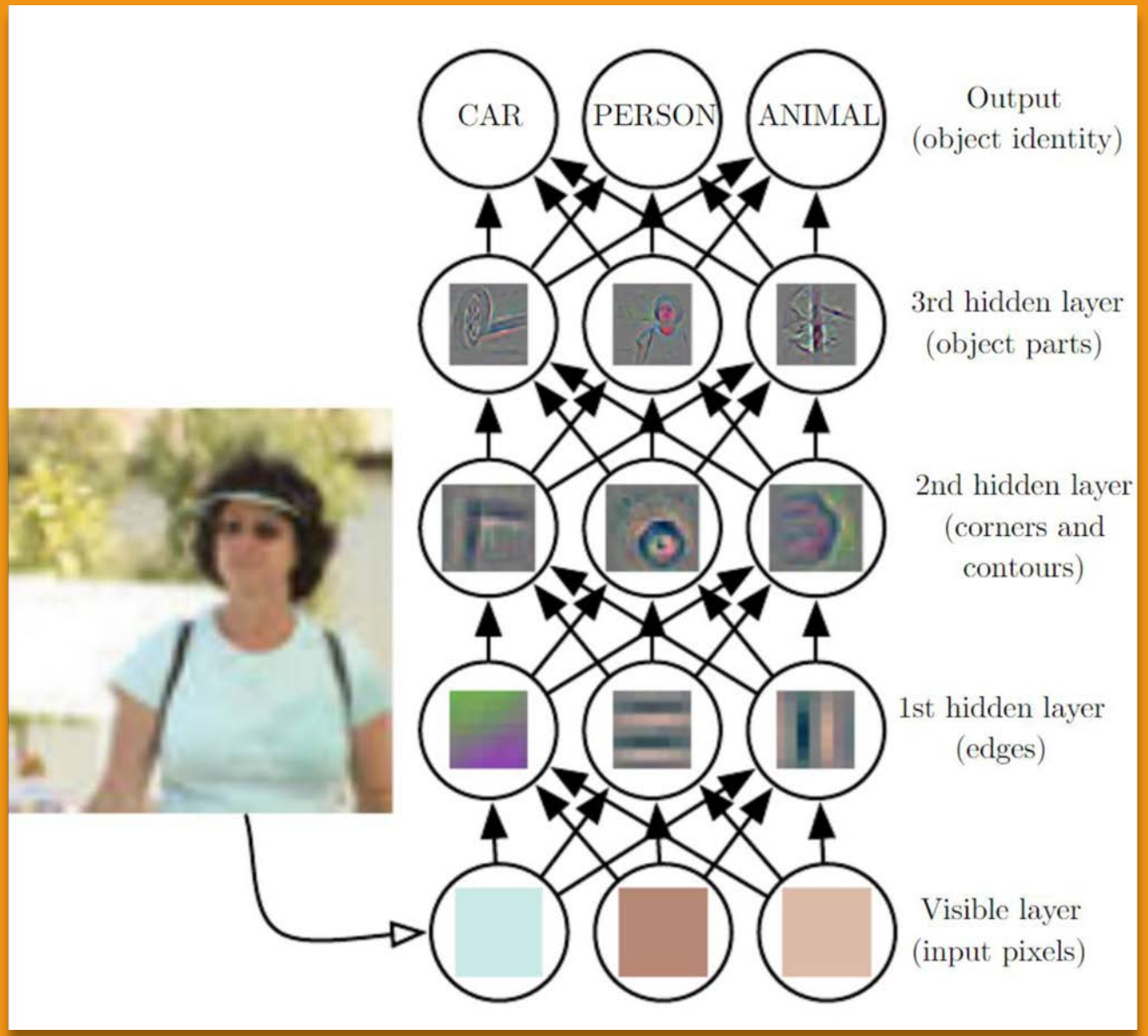


From: Gartner (August 2018)

Deep Learning is Representation Learning

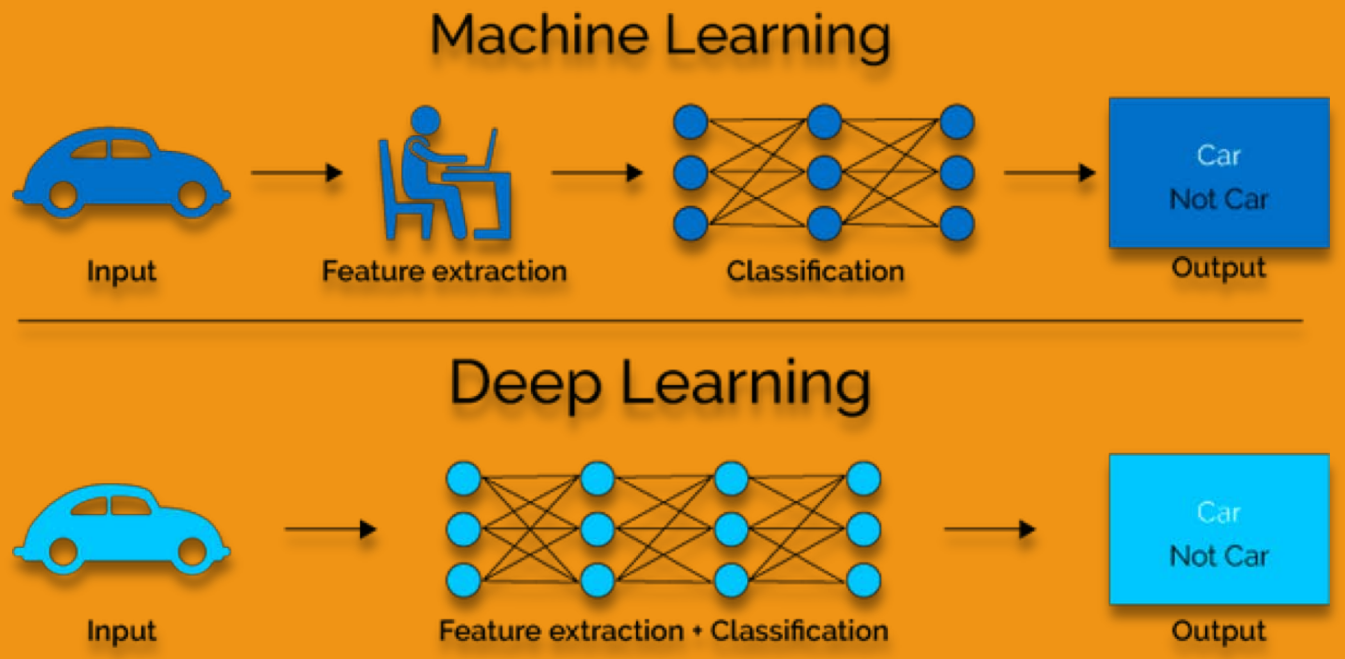


From: MIT Deep Learning (https://deeplearning.mit.edu/)



Deep Learning: Automatic Feature Extraction

- More Data = Better Performance
- End-to-end Learning

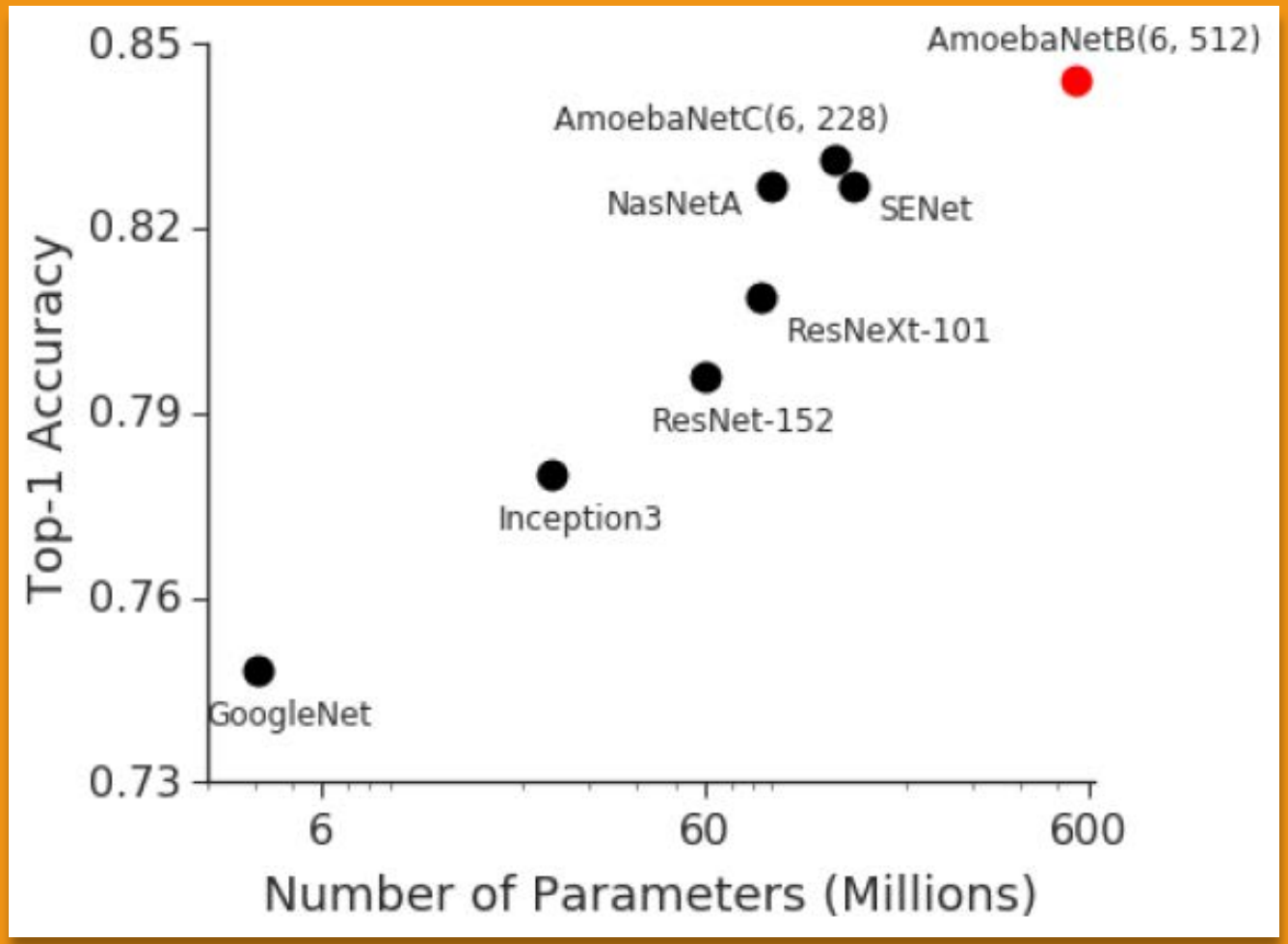


From: codeutsava.in

Model Complexity

- More Data
- More Parameters
- Training Longer
- More Computation
- Better Results

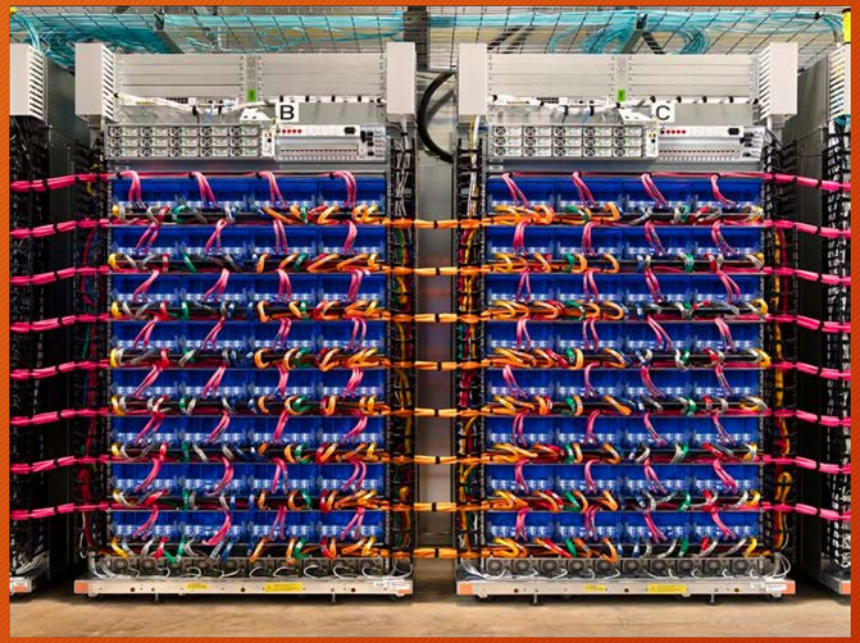
ImageNet Object Detection Challenge



From: Introducing Gpipe, an Open Source Library for Efficiently Training Large-scale Neural Network Models (Google AI Blog)

Hardware

Google Cloud TPU V2 Pod Alpha
11.5 peta FLOPs



FPGA
Programmable
Hardware

(GP)GPU
Data-Level Parallelism

TPU
Custom ASIC specialized
for machine learning

CPU
Serial, General Purpose,
Thread-level Parallelism

Apple A12

64-bit ARM 2.49GHz

8-core Neural Engine

$149

Google Coral Dev

NXP Quad Cortex-A53

Google Edge TPU coprocessor

$99

Intel Neural Compute Stick

USB 3.0 Stick

Deep Learning Accelerator

$99

Nvidia Jetson Nano

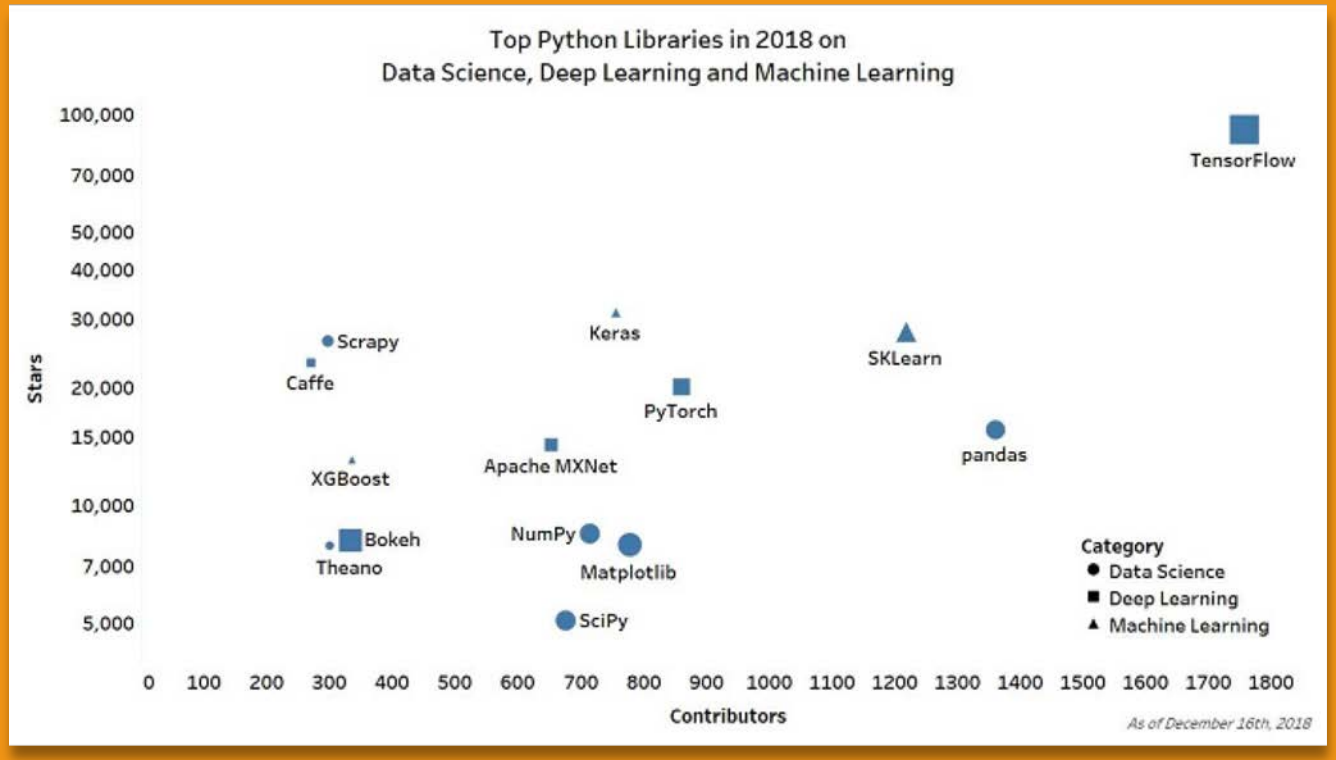
Quad ARM A57

Nvidia Maxwell, 128 CUDA cores

Edge Devices

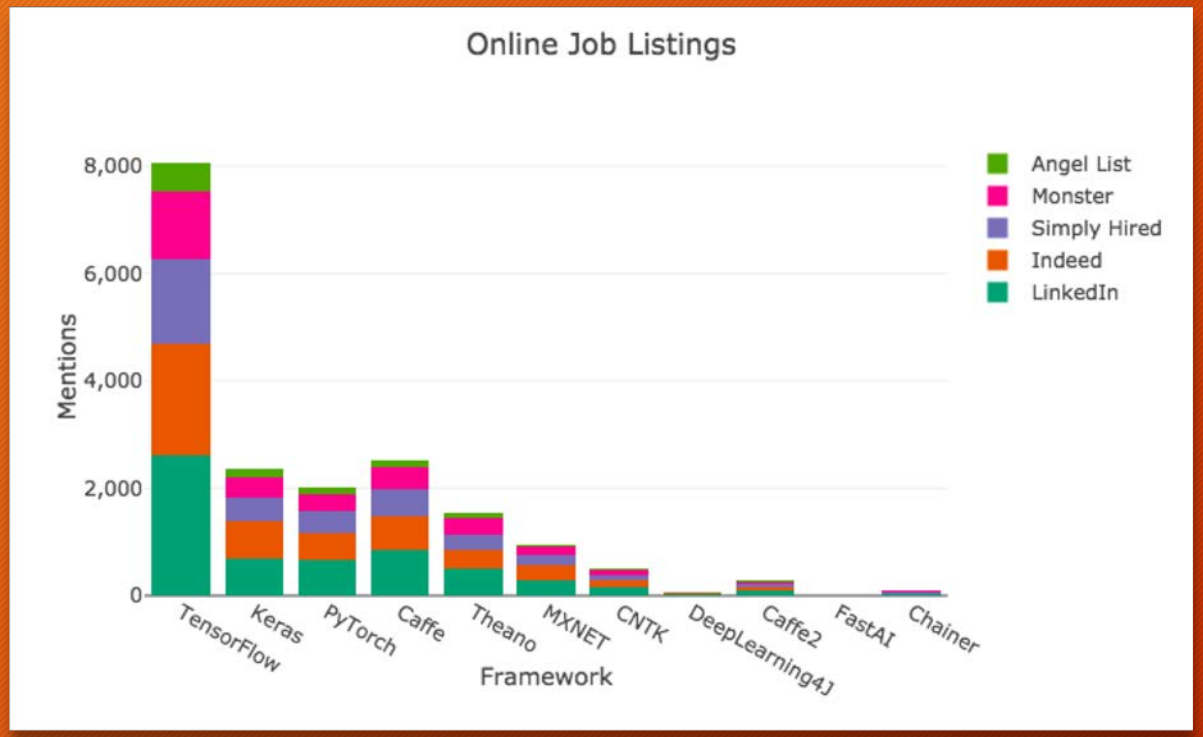
Deep Learning Frameworks

- Tensorflow (Google)
- Keras (Google)
- pyTorch (Facebook)
- Caffe (UC, Berkeley)
- Theano (Universite de Montereal)
- MXNet (Apache)
- CNTK (Microsoft)
- DeepLearning4j (Eclipse)
- Chainer (Preferred Networks)
- Fast.ai

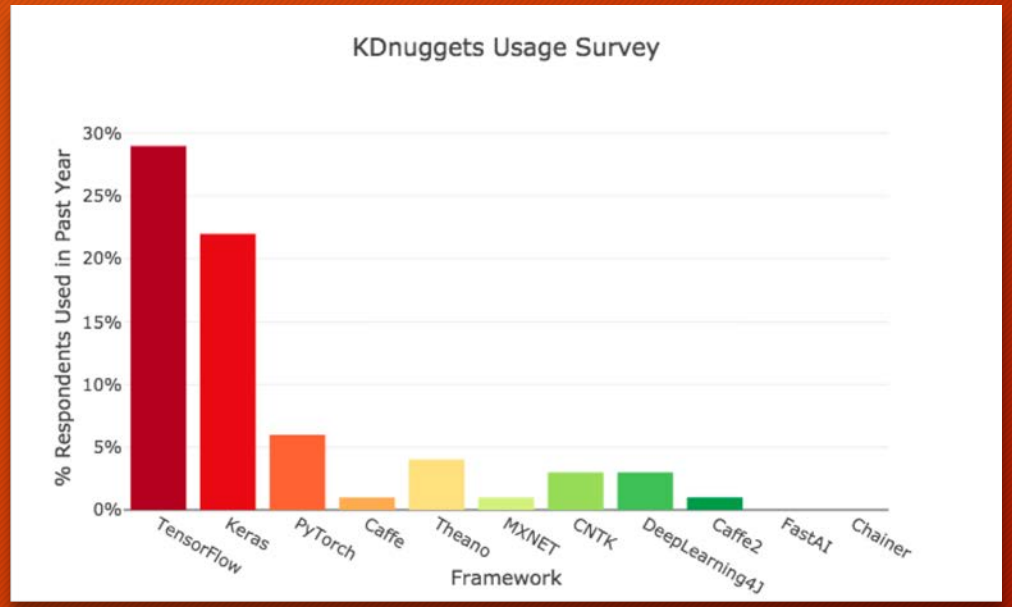


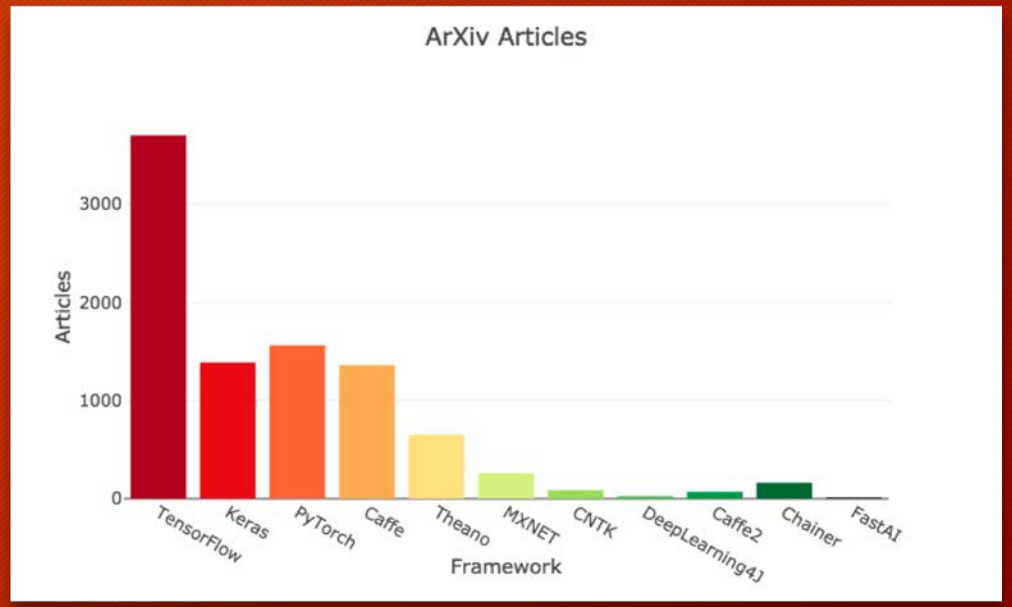
From: Top Python Libraries in 2018 in Data Science, Deep Learning, Machine Learning (KDnuggets)

Framework Popularity

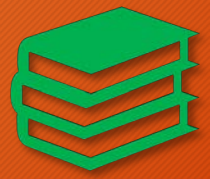


Data From: Deep Learning Framework Power Scores 2018 (Medium)





TensorFlow in One Slide



What is it:

Deep Learning Library

Application Programming Interface

(and more)

Community:

117,000+ Github Stars

46,000 StackOverflow Questions

TensorFlow.org

Blogs, Documentation, DevSummit, YouTube Talks

Ecosystem:

Keras: high-level API

TensorFlow.js: in browser

TensorFlow Lite: on the phone

TPU: Optimized hardware

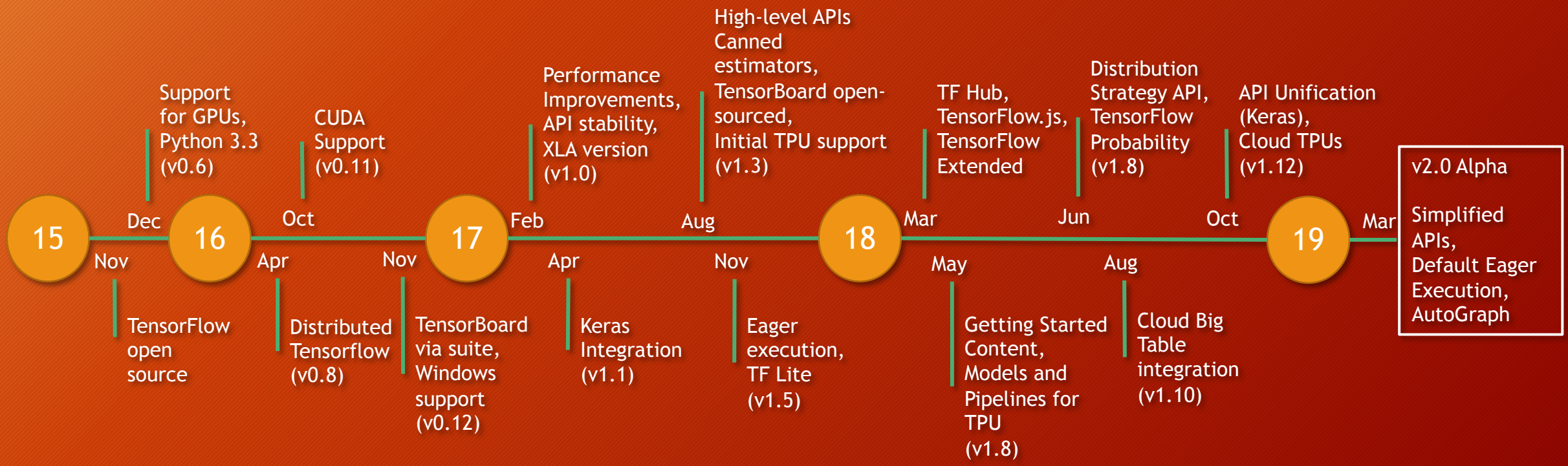
TensorBoard: visualization

TensorFlow Hub: Graph modules

Tensorflow: Architecture

From: Introduction to TensorFlow Datasets and Estimators (Google Developers Blog)

TensorFLow: Roadmap



Reference: TensorFlow Dev Summit 2019

TF: Installation (Python)

- Installation with pip
  - With GPU support
  - Without GPU Support
- For best performance:
  - Compile from source (https://github.com/tensorflow/tensorflow)
- Nvidia CUDA Support
  - CUDA 10
  - CUDNN 7.4

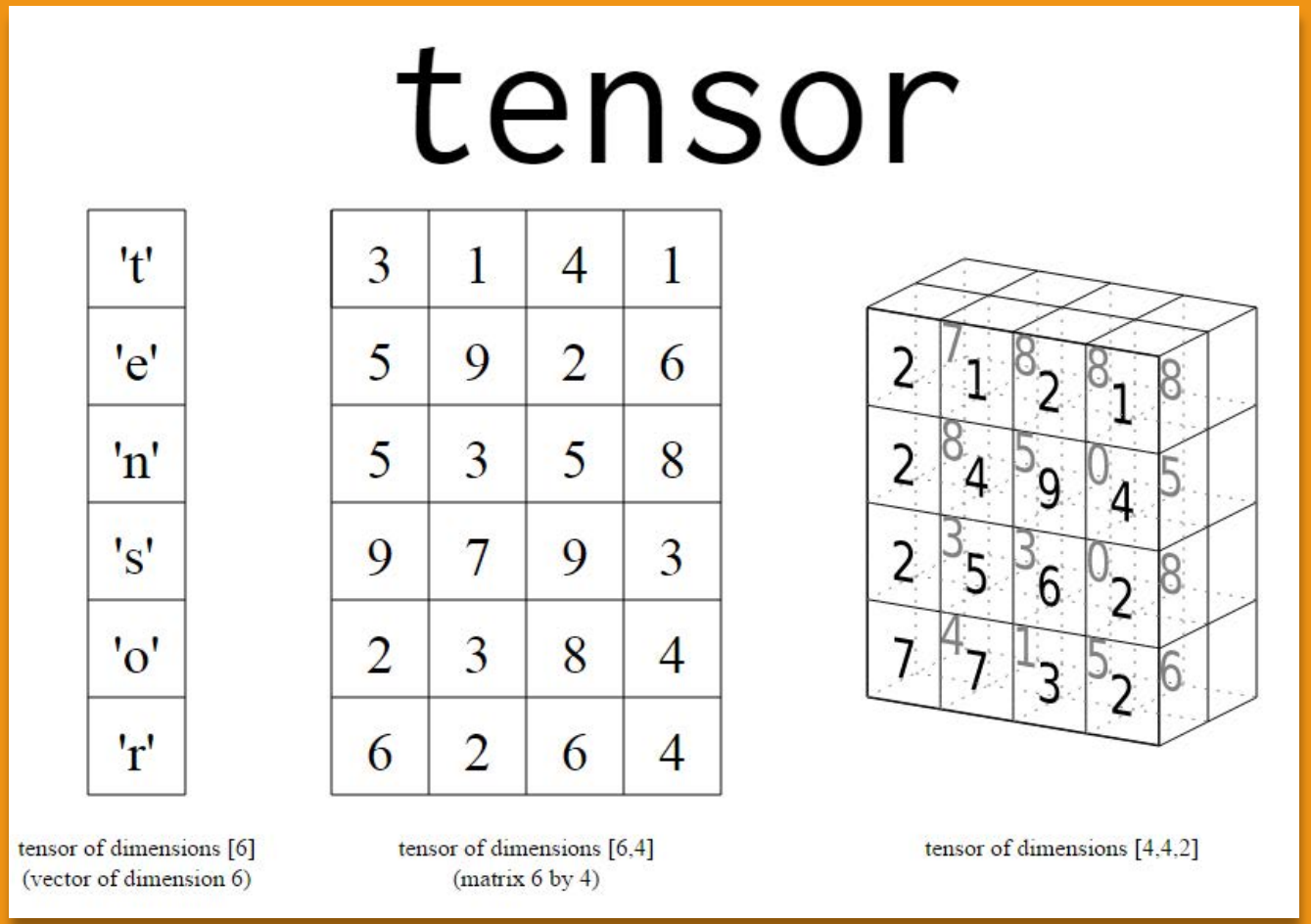
Python Environment Setup

- MiniConda 3 - Python 3.7.1
  - TensorFlow / TensorFlow-GPU
  - Scikit-Learn
  - MatPlotLib
- Nvidia GPU with CUDA 10 (optional)
- Python IDE
  - Visual Studio Code
  - PyCharm (>=2019.1)

TensorFlow: Getting Started

What is Tensor?

- Multi-dimensional Array of Numbers
- This means:
  - A scalar (number) is a tensor (0-d)
  - A vector is a tensor (1-d)
  - A matrix is a tensor (>=2-d)

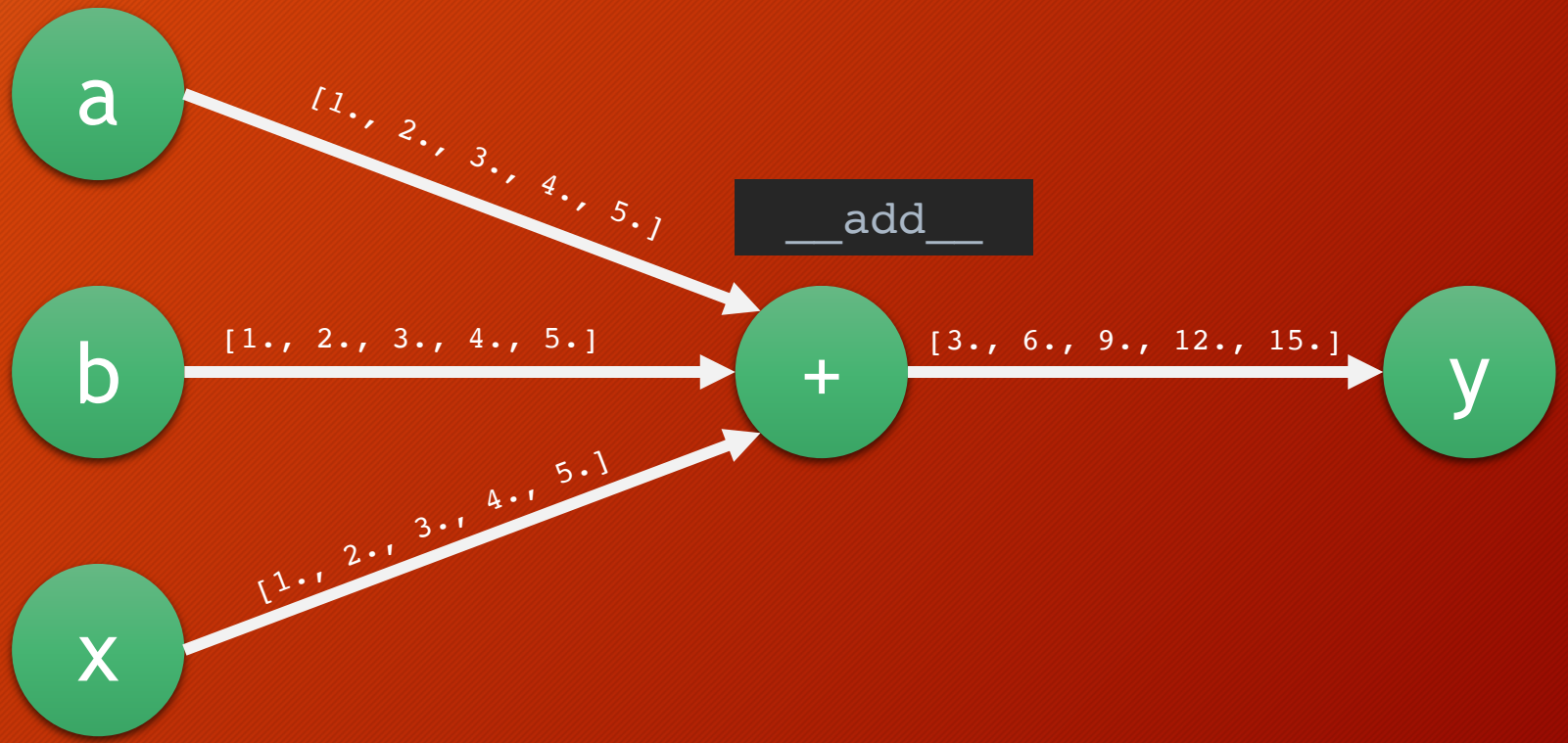


A Simple Graph

```
tf.constant
[1, 2, 3, 4, 5]
```

```
tf.Variable
[1, 2, 3, 4, 5]
```

```
array
[1, 2, 3, 4, 5]
```



TF: Symbolic Computation

Build Graph

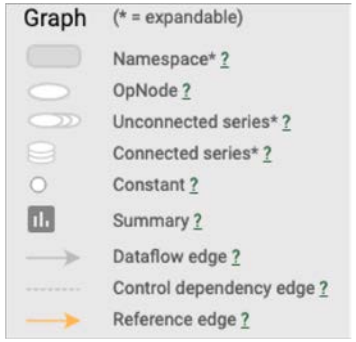
Execute Graph

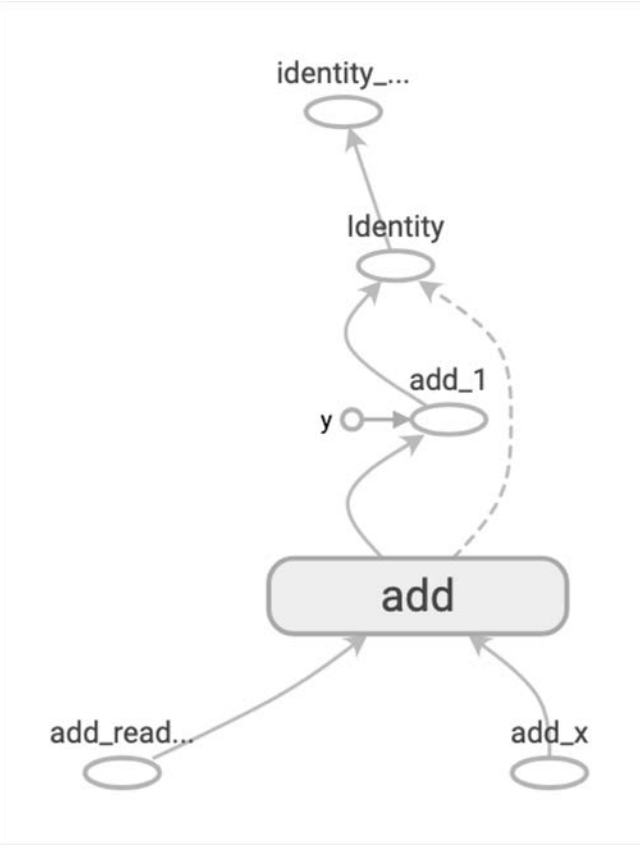
```
import tensorflow as tf

"""
Build Graph
"""
a = tf.constant(value=[1., 2., 3., 4., 5.], name="a")
b = tf.Variable(initial_value=[1., 2., 3., 4., 5.], name="b")

@tf.function
def SimpleAdder(x):
    return a + b + x

"""
Execute Graph
"""
print(SimpleAdder([1., 2., 3., 4., 5.]))
```

TF: Data Flow Graph

- Nodes:
  - Operators, constants
- Edges:
  - Tensors

```
@tf.function
def SimpleAdder(x):
    return a + b + x
```





TF 2.0: Functions, not Sessions

- Basic Idea: Python functions as Graphs

```
@tf.function
def SimpleAdder(x):
    return a + b + x
```

- The graph is executed when the function is invoked

```
SimpleAdder([1., 2., 3., 4., 5.])
```

TensorBoard: Visualization

- Create file writer

```
logdir = 'logs/func/%s' % stamp
writer = tf.summary.create_file_writer(os.path.abspath(logdir))
```

- Turn trace on before executing a graph

```
tf.summary.trace_on(graph=True)
```

- Export the trace to log directory

```
tf.summary.trace_export(name="my_func_trace", step=0, profiler_outdir=logdir)
```

- Open TensorBoard

```
(ml2019) $ tensorboard --logdir ./logs/func/

TensorBoard 1.14.0a20190301 at http://Tinghuis-iMac.local:6006 (Press CTRL+C to quit)
```

Example 1: Simple Addition

TF: Why Graph?

- Save computation
  - Only runs the subgraphs that lead to the values you want to fetch
- Auto Differentiation
- Facilitate distributed computation
  - The work can be spreaded across multiple CPUs, GPUs, TPUs or other devices.

TF: Constant

```
import tensorflow as tf

a = tf.constant([2, 2], name="a")
b = tf.constant([1., 2.], name="b")
c = tf.constant([2, 2], dtype=tf.float32, name="c")
```

```
tf.constant(
    value,
    dtype=None,
    shape=None,
    name='Const'
)
```

```
d = tf.multiply(a, tf.transpose(b))

>>> InvalidArgumentError: cannot
compute Mul as input #1(zero-based) was
expected to be a int32 tensor but is a
float tensor [Op:Mul]
```

```
d = tf.multiply(c, tf.transpose(b))
print(d)
>>> tf.Tensor([2. 4.], shape=(2,), dtype=float32)

tf.print(d)
>>> [2 4]
```

TF: Other Constant API

- Filled with a specific value

```
tf.zeros(shape, dtype=dtypes.float32, name=None)
tf.ones(shape, dtype=dtypes.float32, name=None)
tf.zeros_like(tensor, dtype=None, name=None, optimize=True)
tf.ones_like(tensor, dtype=None, name=None, optimize=True)
tf.fill(dims, value, name=None)
```

- Sequence

```
tf.linspace(start, stop, num, name=None)
tf.range(start, limit=None, delta=1, dtype=None, name="range")
```

- Random Constants

```
tf.random.normal
tf.random.uniform
tf.random.categorical
tf.random.truncated_normal
tf.random.shuffle
tf.random.poisson
```

TF: Datatypes

- Similar to numpy
- Use “dtype” whenever possible

- `tf.float16` : 16-bit half-precision floating-point.
- `tf.float32` : 32-bit single-precision floating-point.
- `tf.float64` : 64-bit double-precision floating-point.
- `tf.bfloat16` : 16-bit truncated floating-point.
- `tf.complex64` : 64-bit single-precision complex.
- `tf.complex128` : 128-bit double-precision complex.
- `tf.int8` : 8-bit signed integer.
- `tf.uint8` : 8-bit unsigned integer.
- `tf.uint16` : 16-bit unsigned integer.
- `tf.int16` : 16-bit signed integer.
- `tf.int32` : 32-bit signed integer.
- `tf.int64` : 64-bit signed integer.
- `tf.bool` : Boolean.
- `tf.string` : String.
- `tf.qint8` : Quantized 8-bit signed integer.
- `tf.quint8` : Quantized 8-bit unsigned integer.
- `tf.qint16` : Quantized 16-bit signed integer.
- `tf.quint16` : Quantized 16-bit unsigned integer.
- `tf.qint32` : Quantized 32-bit signed integer.
- `tf.resource` : Handle to a mutable resource.

What’s wrong with Constants?

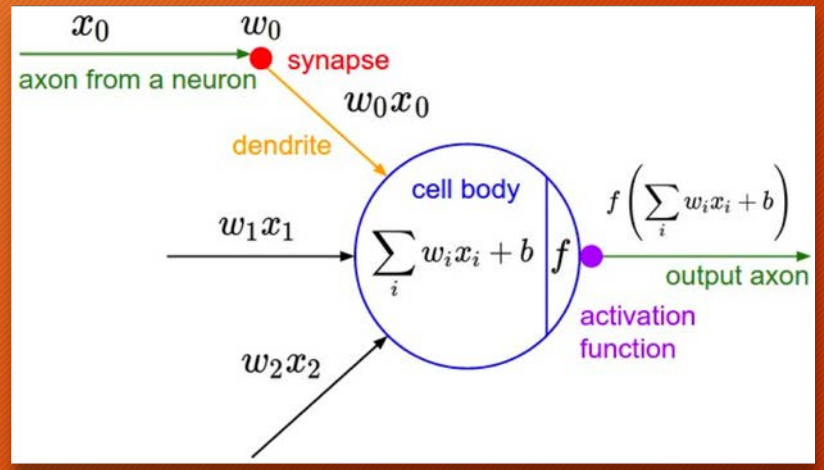
- Constants are stored in the graph definition
- This makes loading graphs expensive when constants are big

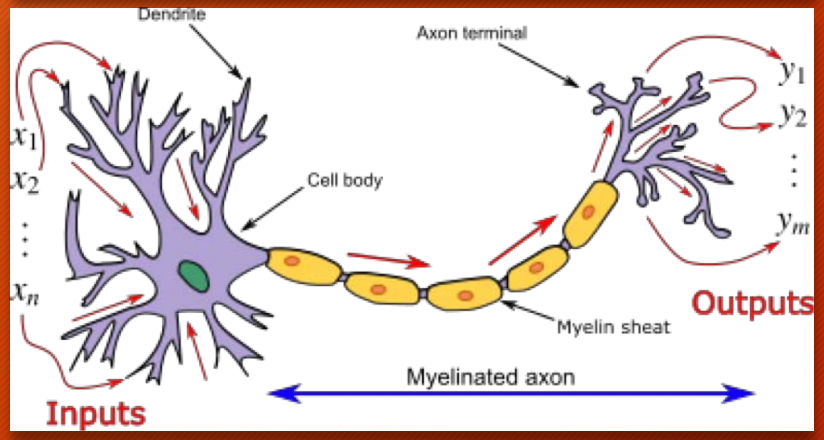
- Only use constants for primitive types
- Use variables or readers for more data that requires more memory

TF: Operations

Category	Examples
Element-wise Math Ops	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal…
Array Ops	Concat, Slice, Split, Constant, Rank, Shape, Shuffle…
Matrix Ops	MatMul, MatrixInverse, MatrixDeterminant, …
Stateful Ops	Variable, Assign, …
Neural Network Ops	SoftMax, Sigmoid, ReLU, Conv1D, Conv2D, MaxPool, …
Common Loss Ops	CrossEntropy, L1/2 Regularization, RMS, …
Optimization Ops	SGD, Adam, AdaDelta, Adagrad, RMSprop, …
Checkpointing Operations	Save, Restore
Python Language	if, for, while, continue, break, … (2.0)

Neural Network:
Biological Inspiration for Computation





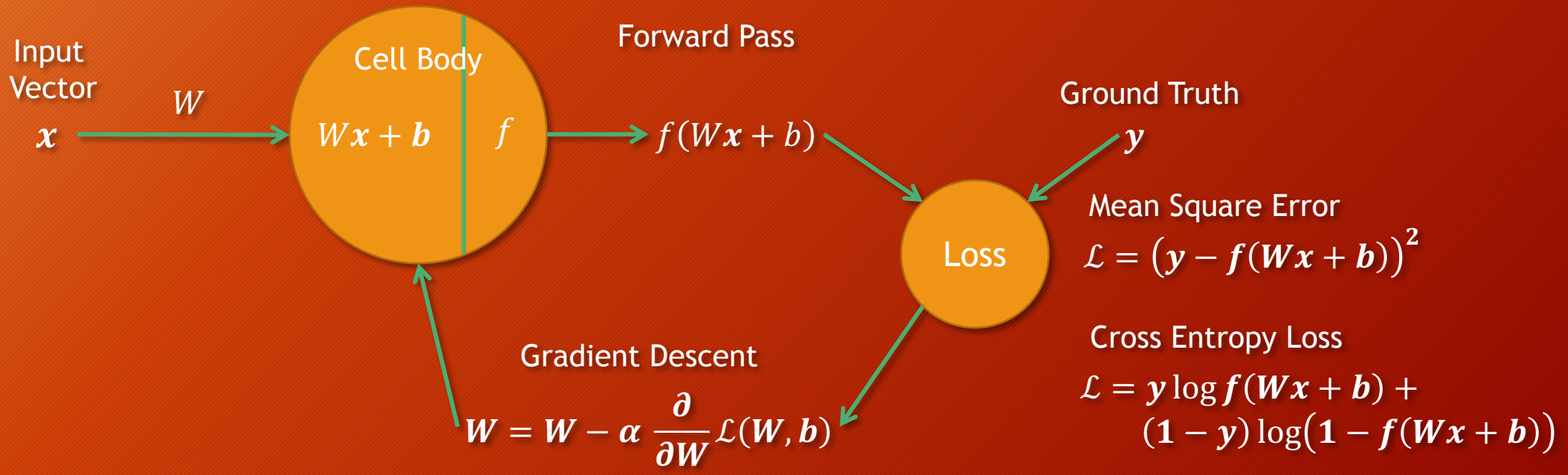
Artificial Neuron:

computational building block for the “neural network”

Neuron:

Computational building block for the brain

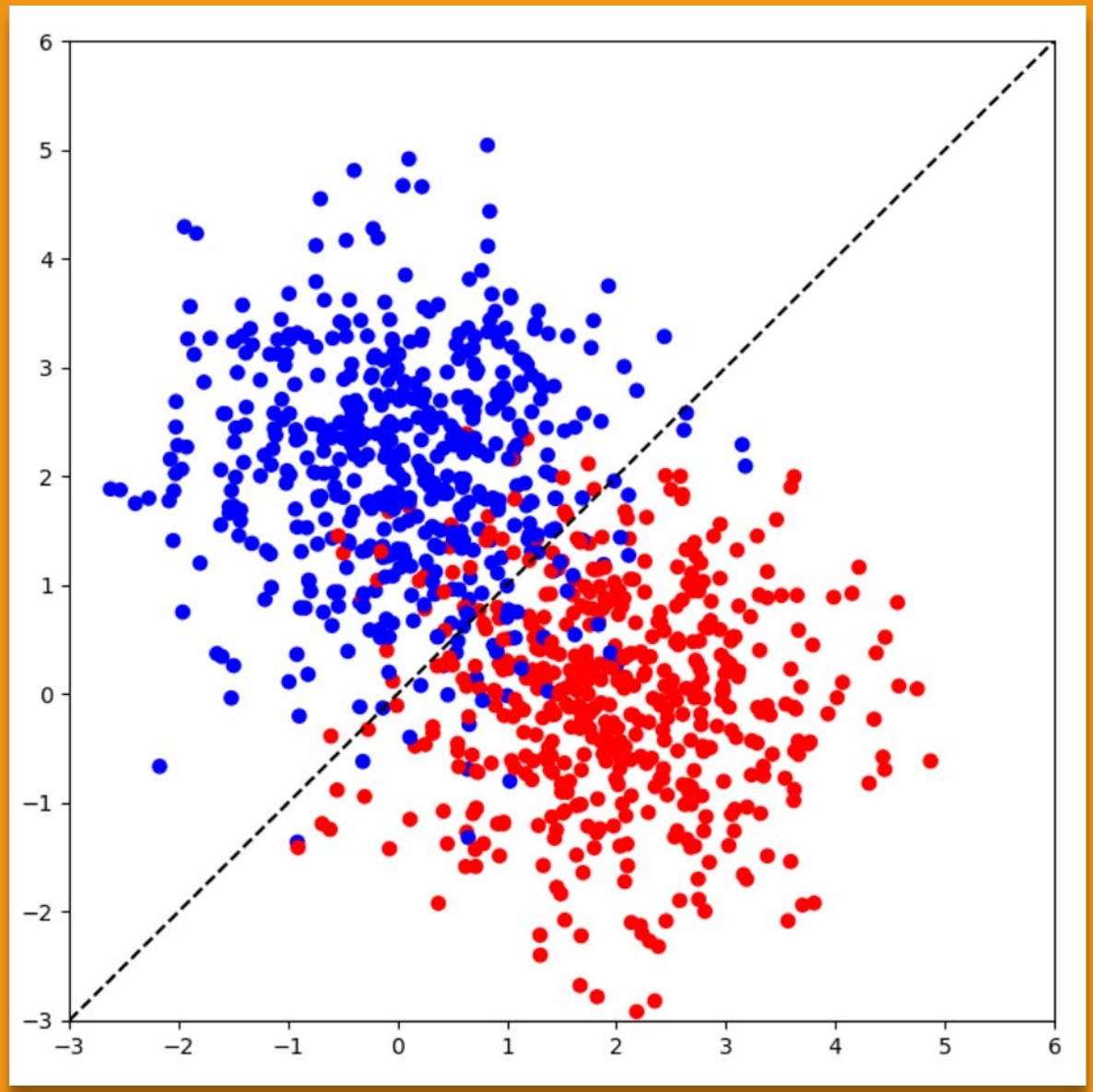
Gradient Descent



Example 2: Linear Regression (Custom Implementation)

Dataset

- Synthetic Dataset
- Training set: 1,000
- Test set: 100
- Linear Classification



Craft the model

```
w = tf.Variable(initial_value=np.random.normal(loc=0., scale=1., size=(2, 1)))
b = tf.Variable(initial_value=np.random.normal((1,)))

@tf.function
def linearModel(x):
    return tf.matmul(x, w) + b
```

X: (batch_size, 2) W: (2, 1) b: (1,)

Training 1: Create Loss

```
def loss(y_true, logits):
    return tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
            labels=y_true, logits=logits, name="cross_entropy_loss"
        ))
```

Training #2: Calculate Gradient

```
@tf.function
def train(x, y, learning_rate):
    with tf.GradientTape() as t:
        logits = linearModel(x)
        current_loss = loss(y, logits)
    dW, db = t.gradient(current_loss, [w, b])
    w.assign_sub(learning_rate * dW)
    b.assign_sub(learning_rate * db)
    return current_loss
```

Compute gradient w.r.t. W and b

Training #3: Update Variable

```
@tf.function
def train(x, y, learning_rate):
    with tf.GradientTape() as t:
        logits = linearModel(x)
        current_loss = loss(y, logits)
    dW, db = t.gradient(current_loss, [w, b])
    w.assign_sub(learning_rate * dW)
    b.assign_sub(learning_rate * db)
    return current_loss
```

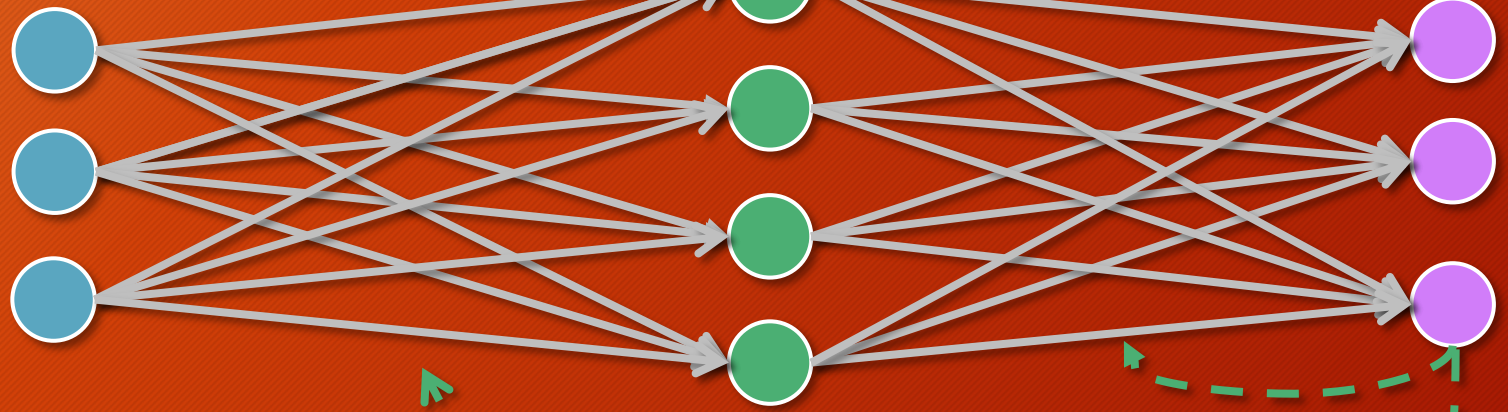
$$W \coloneqq W - lr\frac{\partial \mathcal{L}}{\partial W}$$

Training #4: Call training function

```
epochs = range(100)
for epoch in epochs:
    current_loss = train(
        x_train, y_train.reshape((1000, 1)).astype(dtype=np.float),
        learning_rate=1)
    print("Epoch %02d: loss %.8f" % (epoch, current_loss))
```

Example 3: MNIST with MLP (Keras API)

Multi-Layer Perceptron



Input Vector $\boldsymbol{x}$

Weight W_1

Hidden Layer Output $\boldsymbol{h} = f(W_1\boldsymbol{x} + b_1)$

Weight W_2

Output Logits $a = W_2\boldsymbol{h} + b_2$

$$\frac{\partial \mathcal{L}(\boldsymbol{a})}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{W_1}}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{a})}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W_2}}$$

Softmax

$$p_i = \frac{e^{a_i}}{\sum_k e^{a_k}}$$

$$p_1 = P(y = 1|x) = \frac{e^{a_1}}{e^{a_1} + e^{a_2} + e^{a_3}}$$

$$p_2 = P(y = 2|x) = \frac{e^{a_2}}{e^{a_1} + e^{a_2} + e^{a_3}}$$

$$p_3 = P(y = 3|x) = \frac{e^{a_3}}{e^{a_1} + e^{a_2} + e^{a_3}}$$

Cross Entropy Loss $\mathcal{L} = -\sum_i y_i \log(p_i)$

Multi-Layer Perceptron: Simple Implementation

1
- Load Dataset

2
- Build Forward Model

3
- Loss Function and Optimizer

4
- Training the Model

5
- Evaluation

```
# For auto-completion to work
from sklearn.metrics import classification_report
import tensorflow.python.keras as keras

if __name__ == "__main__":
    (x_train, y_train), (x_test, y_test) =
         keras.datasets.mnist.load_data()
    # Normalize image pixel intensity
    x_train = x_train / 255
    x_test = x_test / 255


    # Build models with keras
    model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(128, activation='relu'),
        keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    model.fit(
        x=x_train, y=y_train, epochs=5
    )


    y_pred = model.predict_classes(x=x_test)
    print(classification_report(y_test, y_pred, target_names=[
        '%d' % i for i in range(10)
    ]))
```

1

2

3

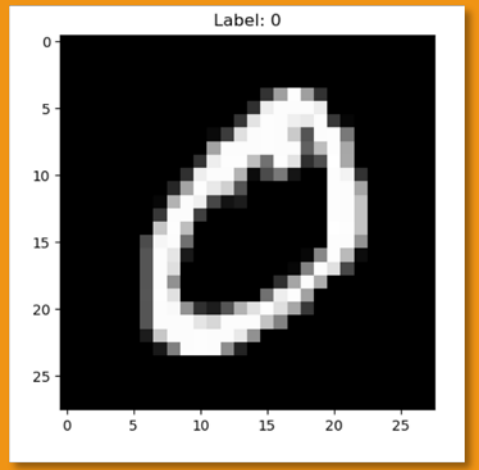
4

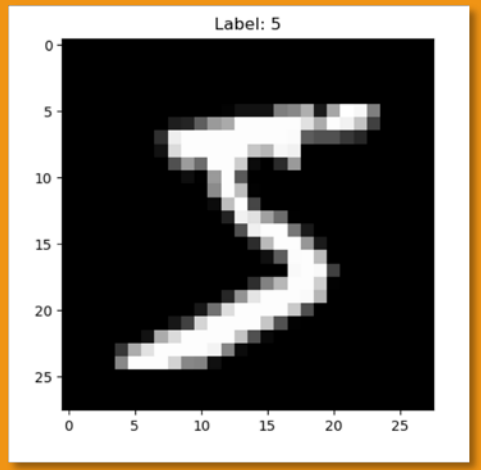
5

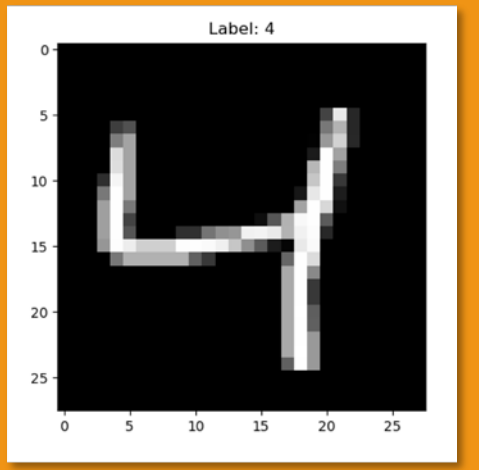
MNIST Data

- Each image is 28x28 array
- Training Set: 60,000
- Test Set: 10,000

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()


for i in range(3):
    plt.figure()
    img = x_train[i, :, :]
    plt.title("Label: %d" % y_train[i])
    plt.imshow(img, cmap="gray")
    plt.savefig("digit_%d.png" % i)
    plt.show()
```







How to learn TF?

- Take a Course
  - MIT: Deep Learning
  - Stanford: Introduction to TensorFlow
- Read Books
- Other’s Codes
- Code to learn
  - Google Colaboratory

TF: References

- TensorFlow 2.0 Alpha Tutorial/Guide
- TensorFlow API 2.0
- TensorBoard For TF 2.0
- TensorFlow Blog
- TensorFlow Dev Summit
- TensorFlow RFCs