# UVA CS 4774

# A Brief Introduction to Keras

Presented By: Zhe Wang

Professor: Dr. Yanjun Qi

March 22, 2022

# Computational Graphs

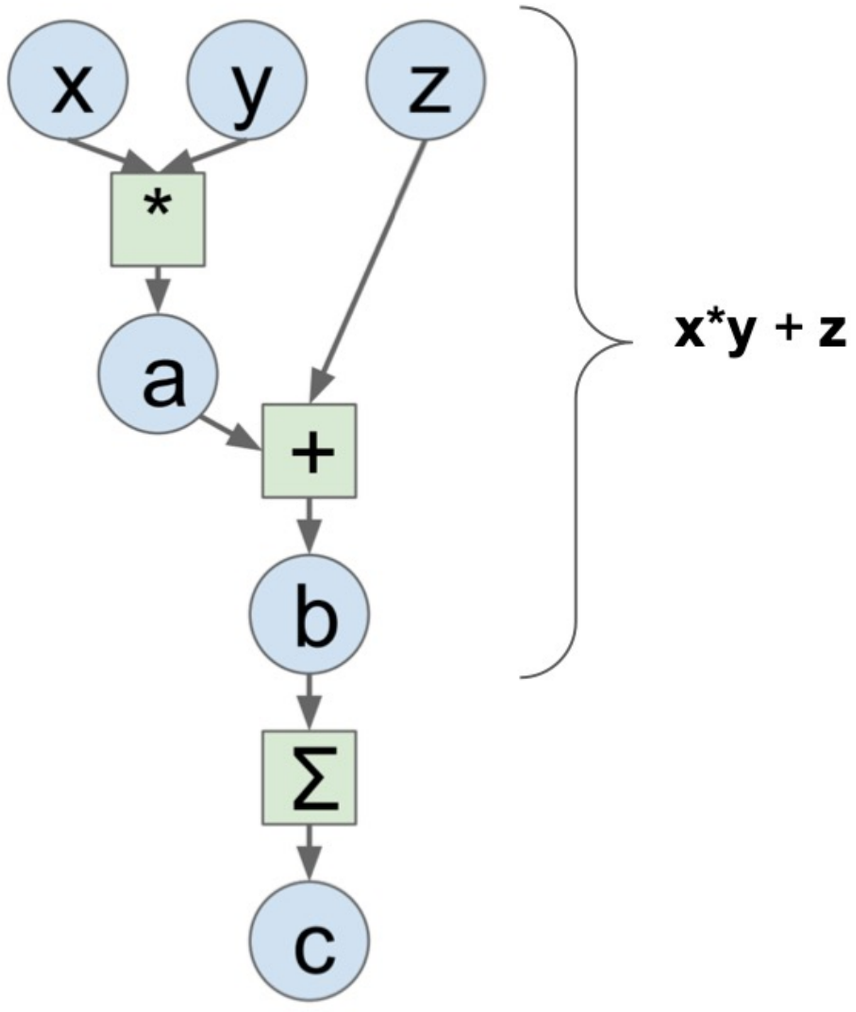
## Numpy

```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)
```



# Computational Graphs

## Numpy

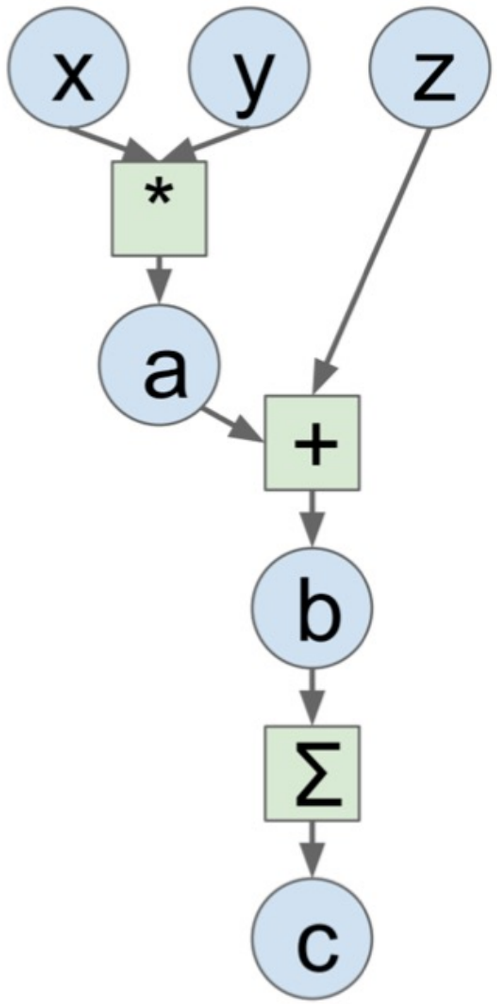
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



# Computational Graphs

## Numpy

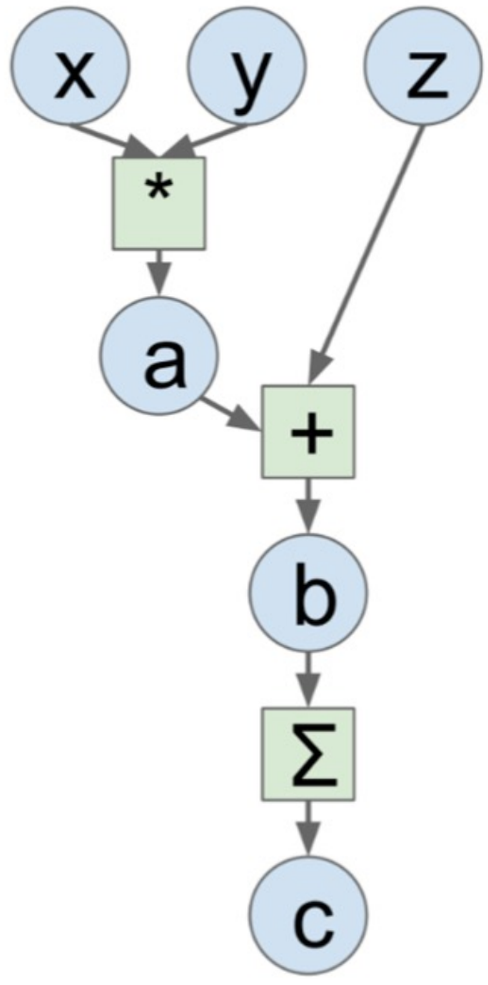
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```




**Good**:

- Clean API, easy to write numeric code

**Bad**:

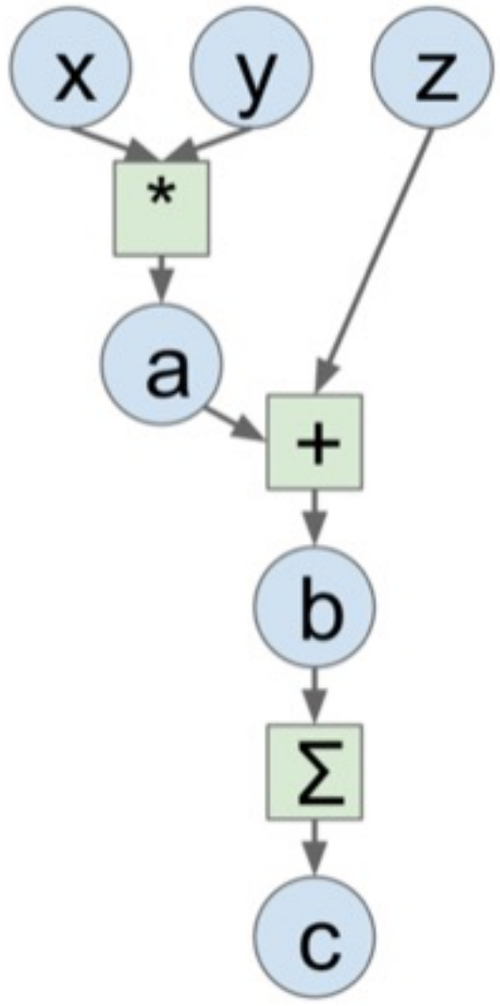
- Have to compute our own gradients
- Can’t run on GPU

# *Deep Learning Libraries: Automatic differentiation*

```python
x = tf.Variable(tf.random.normal((3, 4)), name='x')
y = tf.Variable(tf.random.normal((3, 4)), name='y')
z = tf.Variable(tf.random.normal((3, 4)), name='z')
```

```python
with tf.GradientTape(persistent=True) as tape:
  a = x*y
  b = a+z
  c = sum(b)
```

```python
dx = tape.gradient(c, [x])
```



# Deep Learning Hardware: GPUs

## CPU vs GPU

| | Cores | Clock Speed | Memory | Price | Speed |
|---|---|---|---|---|---|
| **CPU** (Intel Core i7-7700k) | 4 (8 threads with hyperthreading) | 4.2 GHz | System RAM | $385 | ~540 GFLOPs FP32 |
| **GPU** (NVIDIA RTX 2080 Ti) | 3584 | 1.6 GHz | 11 GB GDDR6 | $1199 | ~13.4 TFLOPs FP32 |

**CPU**: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

**GPU**: More cores, but each core is much slower and "dumber"; great for parallel tasks

## Create variables on GPUs

```
with tf.device('/device:GPU:2'):
  a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
  b = tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])
  c = tf.matmul(a, b)
```

*Deep Learning Packages: Fundamental Concepts*

**Tensor**: Like a numpy array, but can run on GPU

**Autograd**: Package for building computational graphs out of Tensors, and automatically computing gradients

**Module**: A neural network layer; may store state or learnable weights

# Spot the CPU!

(central processing unit)



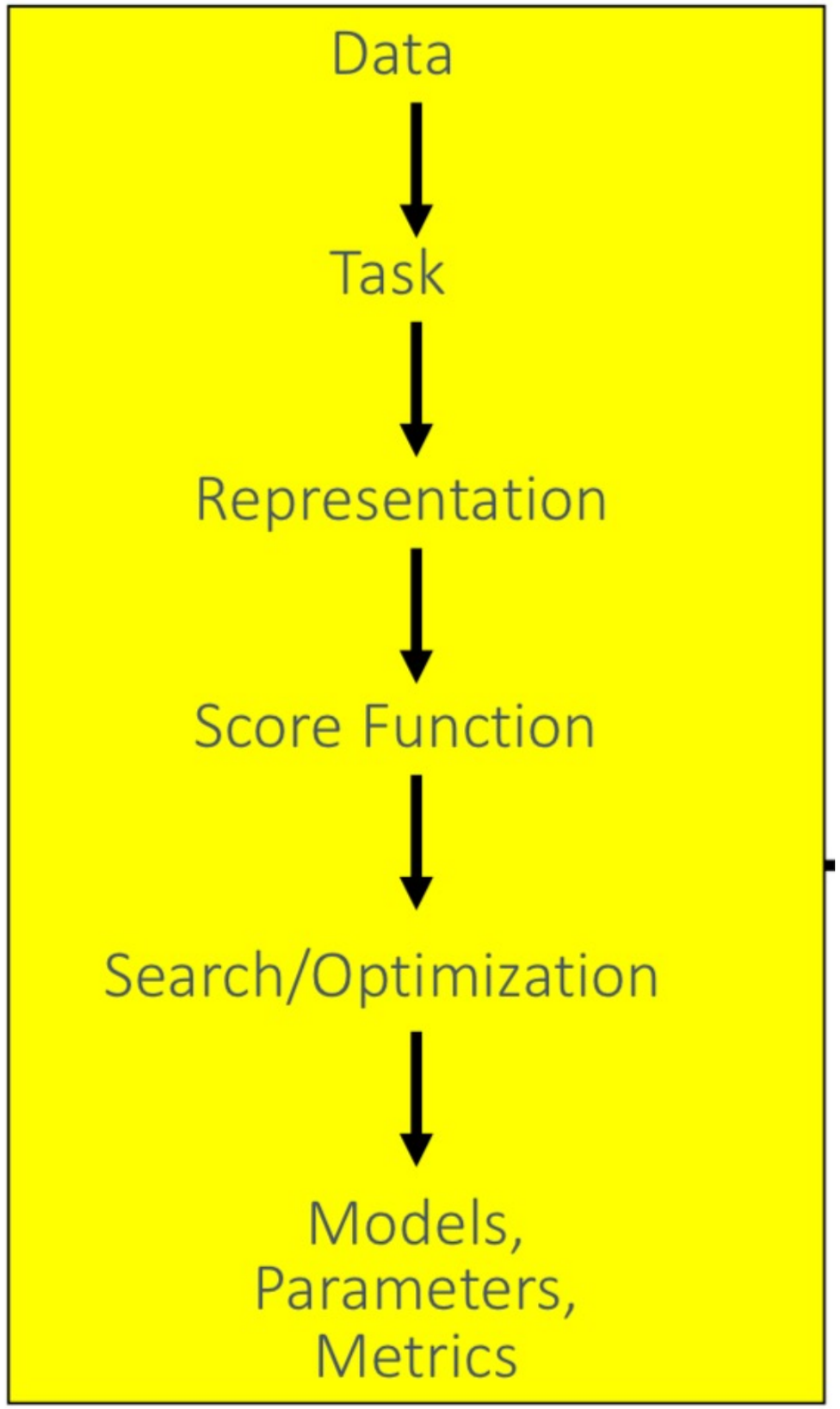

# Spot the GPUs!

(graphics processing unit)




# An introduction To Keras

# Nutshell for the Deep Learning



# I: Data

Keras.datasets module provides several toy datasets including MNIST, CIFAR10, CIFAR100,

First load the dataset:

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data(path="mnist.npz")
```

Preprocessing:

```
x_train = x_train.astype("float32") / 255  # Scale pixel values to [0, 1]
x_train = np.expand_dims(x_train, -1)   # Size of x_train = [60000, 28, 28, 1], for CNN.
x_train = x_train.reshape(-1, 784)  # Size of x_train = [60000, 784], for MLP.
```

```
y_train = keras.utils.to_categorical(y_train, num_classes) # (60000, ) to (60000, 10)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

# II: Model

Two ways to construct the model: sequential and functionals

Sequential:

```
input_shape = (28, 28, 1)
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="sigmoid"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dropout (Dropout)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 10)                16010
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

Functional:

```
class MyModel(keras.Model):

  def __init__(self):
    super(MyModel, self).__init__()
    self.dense1 = keras.layers.Dense(4, activation='relu')
  def call(self, inputs):
    x = self.dense1(inputs)
    return x

model = MyModel()
```

Print out the model:

```
model = MyModel()
model.build(input_shape=(None,784))
model.summary()
```

```
Model: "my_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              multiple                  3140
=================================================================
Total params: 3140
Trainable params: 3140
Non-trainable params: 0
_________________________________________________________________
```

## III: Define the loss function

```
my_loss = keras.losses.CategoricalCrossentropy(
    label_smoothing=0,
    name="categorical_crossentropy",
)
```

## II: Define the optimizer

```
my_opt = keras.optimizers.Adam(learning_rate=1e-3)  # Adam, SGD, Adagrad, RMSprop, etc.
```

## IV: Compile the model: configures the model for training

```
model.compile(loss=my_loss, optimizer=my_opt, metrics=["accuracy"])
```

Or, simply calling:

```
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

## VI: Start the training

```
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1, shuffle=True)
```

```
Epoch 1/5
422/422 [==============================] - 48s 114ms/step - loss: 0.3594 - accuracy: 0.8901 - val_loss: 0.0810 - val_accuracy: 0.9772
Epoch 2/5
422/422 [==============================] - 42s 100ms/step - loss: 0.1094 - accuracy: 0.9663 - val_loss: 0.0552 - val_accuracy: 0.9845
Epoch 3/5
422/422 [==============================] - 33s 78ms/step - loss: 0.0824 - accuracy: 0.9742 - val_loss: 0.0508 - val_accuracy: 0.9860
Epoch 4/5
422/422 [==============================] - 33s 78ms/step - loss: 0.0691 - accuracy: 0.9789 - val_loss: 0.0446 - val_accuracy: 0.9880
Epoch 5/5
422/422 [==============================] - 30s 72ms/step - loss: 0.0593 - accuracy: 0.9815 - val_loss: 0.0429 - val_accuracy: 0.9892
```

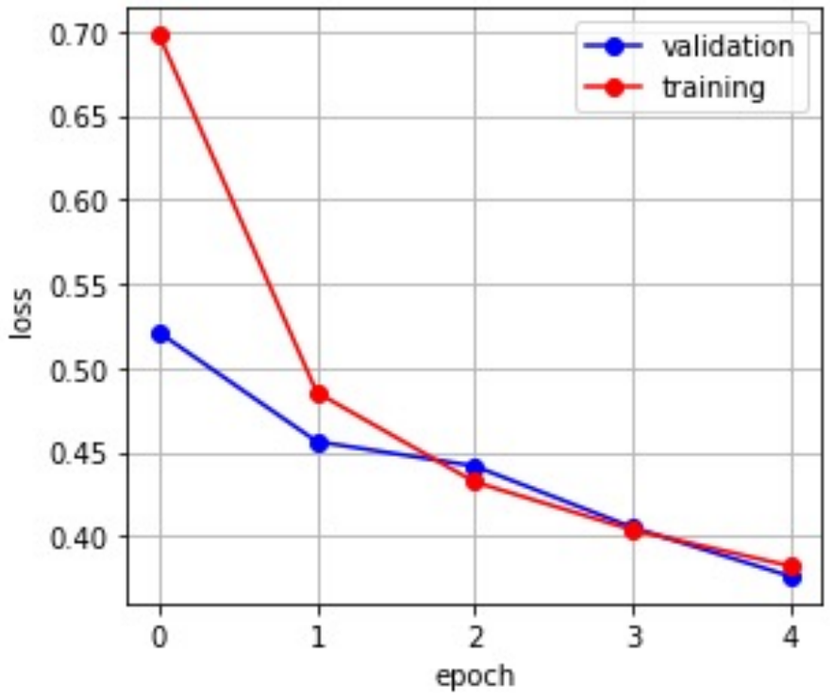
## VII: Start the evaluation

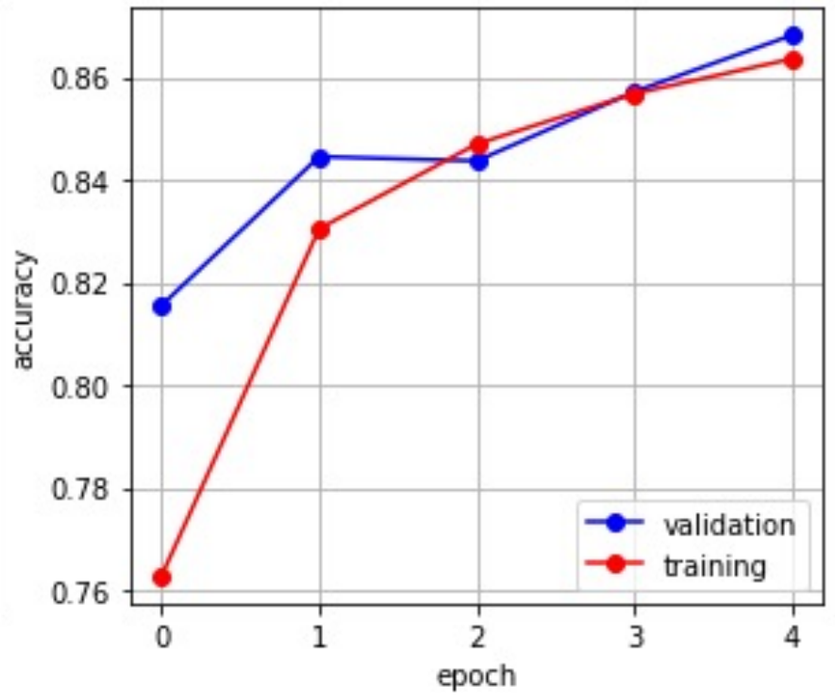
```
model.evaluate(x_test, y_test)  # Returns the loss value & metrics values for the model in test mode.
```

# VIII: Start your analysis and visualize your metric

```
train_loss_history = history.history['loss']
val_loss_history = history.history['val_loss']

train_acc_history = history.history['accuracy']
val_acc_history = history.history['val_accuracy']
```



## Extra:

1: Custom data generator: https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence

2: Data augmentation: https://keras.io/api/preprocessing/image/

3: Custom layer: https://keras.io/guides/making_new_layers_and_models_via_subclassing/

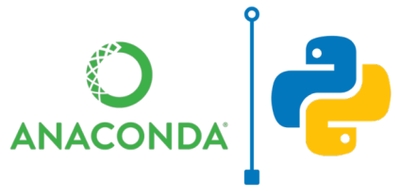
4: Custom metric: https://github.com/borundev/ml_cookbook/blob/master/Custom%20Metric%20

# Step 1: Install python

1. Go to: https://www.python.org/downloads/
2. Install the corresponding installer for your OS and launch respective installers
    a. **Linux**
        i. Most distors ships with both Python 3 and Python 2 pre-installed.
        ii. To make sure that our versions are up-to-date, let's update and upgrade the system
            1. `sudo apt update`
            2. `sudo apt -y upgrade`
    b. **Mac** | Download & Launch `"python-3.XX.X-macos11.pkg"`
    c. **Windows** | Download & Launch `"python-3.XX.X-amd64.exe"`
3. Check the installation.
    a. [*] Open your terminal/cmd and run `"python --version"`
    b. Make sure the python version is `3.7+`

[*] if python version is `2.xx` try `"python3 -version"` and use `pip3` instead of `pip` in all following steps

# Step 2: Install Anaconda




Simply, Anaconda is just a collections of python packages that you will need for data science e.g. matplotlib, numpy, pandas etc.

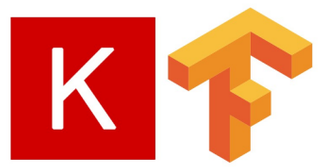
1. Anaconda documentation is very helpful and detailed
2. Installation instructions
    a. Linux | https://docs.anaconda.com/anaconda/install/linux/
    b. Mac | https://docs.anaconda.com/anaconda/install/mac-os/
    c. Windows | https://docs.anaconda.com/anaconda/install/windows/
3. Check the installation.
    a. Open your terminal/cmd and run `”conda --version”`
    b. Make sure a version is printed

# Step 3: Create a conda environment

An environment is just an isolated python installation with a set of chosen packages

1. Open your terminal; run following commands (replace env_name)
    a. Note that tensorflow requires python 3.7+
    b. `conda create -n env_name python=3.8`
    c. `conda activate env_name` or `source activate env_name`
2. This will activate an environment where you can go ahead and install packages you want with `pip` or `conda`

# Step 4: Install Tensorflow + Keras



Keras now ships as a part of the Tensorflow 2.0 package.

1. Install tensorflow using pip; open your terminal/cmd; run following:

```
# Requires the latest pip
$ pip install --upgrade pip

# Current stable release for CPU and GPU
$ pip install tensorflow
```

1. Check the installation; in your terminal/cmd; run following:
    a. `python`
    b. `import tensorflow`
    c. `from tensorflow import keras`