# Deep Learning 01: *TensorFlow introduction*

Lecture 10

Computer Vision for Geosciences

2021-04-09

Universidad Nacional Autónoma de México

**Several frameworks exist**:

- Tensor Flow
  - developped by Google

**Several frameworks exist**:

- Tensor Flow
  - developped by Google
  - includes the high-level API Keras library (TF version $\geq$2)

**Several frameworks exist**:

- Tensor Flow
  - developped by Google
  - includes the high-level API Keras library (TF version $\geq$2)

- PyTorch
  - developed by Facebook

TensorFlow

Keras

PyTorch

**Several frameworks exist**:

- Tensor Flow
  - developped by Google
  - includes the high-level API Keras library (TF version $\geq$2)

- PyTorch
  - developed by Facebook
  - based on the Torch framework (Lua)

**Several frameworks exist**:

- Tensor Flow
  - developped by Google
  - includes the high-level API Keras library (TF version $\geq$2)
- PyTorch
  - developed by Facebook
  - based on the Torch framework (Lua)
- Caffe, Apache's MXNet, Theano, etc.

**Popularity of the main frameworks until 2018** (from Chollet 2017 [1])

NB: this graph is not up-to-date, since 2018 PyTorch has significantly gained popularity, competing with Tensor Flow

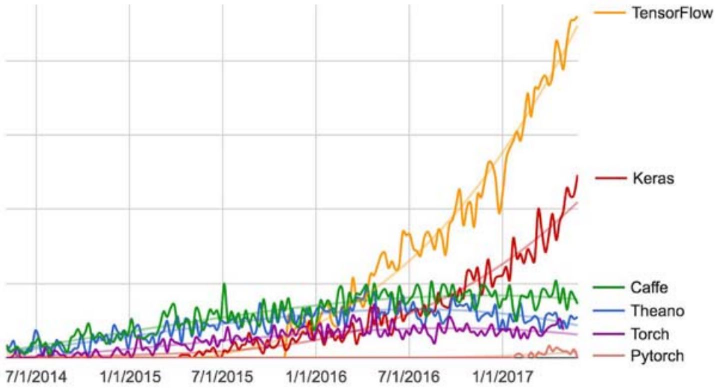



**Figure 3.2 Google web search interest for different deep-learning frameworks over time**

## Installing Tensor Flow with Anaconda (instructions):

⇒ we will install Tensor Flow in a "conda environment"

1. Create environment and install Tensor Flow package & dependencies inside

```
$ conda env list                   # optional: list existing environments
$ conda create -n tf tensorflow    # create environment called "tf" & install CPU-only TensorFlow
```

2. Activate the created environment

```
$ conda activate tf
```

3. Install additional packages in the active environment

```
$ conda install jupyter matplotlib pandas scikit-learn
```

4. Launch Jupyter from the active environment, import Tensor Flow, and you're good to go!

```
$ jupyter notebook
```

```
# Create a new notebook with Python 3 kernel
import tensorflow as tf
```

## Installing Tensor Flow with Anaconda (instructions):

⇒ we will install Tensor Flow in a "conda environment"

1. Create environment and install Tensor Flow package & dependencies inside

```
$ conda env list                    # optional: list existing environments
$ conda create -n tf tensorflow     # create environment called "tf" & install CPU-only TensorFlow
```

2. Activate the created environment

```
$ conda activate tf
```

3. Install additional packages in the active environment

```
$ conda install jupyter matplotlib pandas scikit-learn
```

4. Launch Jupyter from the active environment, import Tensor Flow, and you're good to go!

```
$ jupyter notebook
```

```
# Create a new notebook with Python 3 kernel
import tensorflow as tf
```

## Installing Tensor Flow with Anaconda (instructions):

⇒ we will install Tensor Flow in a "conda environment"

1. Create environment and install Tensor Flow package & dependencies inside

```
$ conda env list                    # optional: list existing environments
$ conda create -n tf tensorflow     # create environment called "tf" & install CPU-only TensorFlow
```

2. Activate the created environment

```
$ conda activate tf
```

3. Install additional packages in the active environment

```
$ conda install jupyter matplotlib pandas scikit-learn
```

4. Launch Jupyter from the active environment, import Tensor Flow, and you're good to go!

```
$ jupyter notebook
```

```
# Create a new notebook with Python 3 kernel
import tensorflow as tf
```

## Installing Tensor Flow with Anaconda (instructions):

⇒ we will install Tensor Flow in a "conda environment"

1. Create environment and install Tensor Flow package & dependencies inside

```
$ conda env list                     # optional: list existing environments
$ conda create -n tf tensorflow     # create environment called "tf" & install CPU-only TensorFlow
```

2. Activate the created environment

```
$ conda activate tf
```

3. Install additional packages in the active environment

```
$ conda install jupyter matplotlib pandas scikit-learn
```

4. Launch Jupyter from the active environment, import Tensor Flow, and you're good to go!

```
$ jupyter notebook
```

```
# Create a new notebook with Python 3 kernel
import tensorflow as tf
```

## Nota Bene

Two distinct versions of TF exist, depending on whether it should run on CPU (Central Processing Unit), or GPU (Graphics Processing Unit)

$\Rightarrow$ *CPU-only TensorFlow (recommended for beginners)*

```
$ conda create -n tf tensorflow
```

$\Rightarrow$ *GPU TensorFlow*

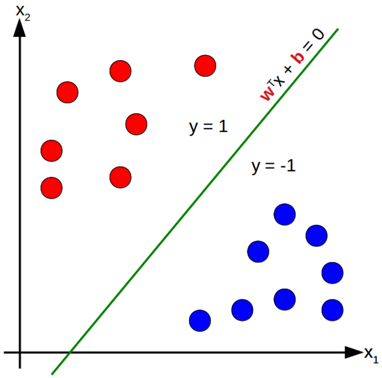
```
$ conda create -n tf-gpu tensorflow-gpu
```

$\Rightarrow$ GPU will be much faster, but more expensive, and trickier to setup (requires CUDA)

**Toy example**: linear classification task using *scikit-learn* and *tensor flow*

**perceptron**: $\boxed{y = \text{sign}(\mathbf{w}^T\mathbf{x} + b)}$

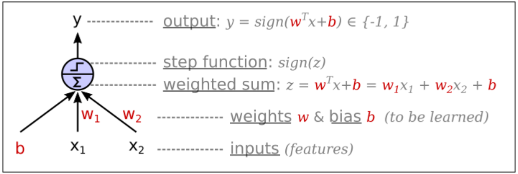
- $y \in \{-1, 1\}$: predicted class $\rightarrow$ *banana or apple*
- $\mathbf{x} \in \mathbb{R}^2$: feature vector $\rightarrow$ *[hue, elongation]*
- $\mathbf{w} \in \mathbb{R}^2$: "weight vector" $\rightarrow$ *needs to be learned*
- $b \in \mathbb{R}$: "bias" $\rightarrow$ *needs to be learned*
- *sign*: sign function returning the sign of a real number



## **Toy example**: linear classification task using *scikit-learn* and *tensor flow*

**perceptron**: $\boxed{y = \text{sign}(\mathbf{w}^T \mathbf{x} + b)}$

- $y \in \{-1, 1\}$: predicted class $\rightarrow$ *banana or apple*
- $\mathbf{x} \in \mathbb{R}^2$: feature vector $\rightarrow$ *[hue, elongation]*
- $\mathbf{w} \in \mathbb{R}^2$: "weight vector" $\rightarrow$ *needs to be learned*
- $b \in \mathbb{R}$: "bias" $\rightarrow$ *needs to be learned*
- *sign*: sign function returning the sign of a real number



## Solution with Scikit-Learn: *Perceptron classifier*

```
from sklearn import datasets
from sklearn import linear_model
from sklearn.utils import shuffle
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
```

```
# Load data
iris = datasets.load_iris()
X = iris.data[:, (2, 3)] # petal length, petal width
y = (iris.target == 0).astype('int') # Iris setosa?
```

**1.1 Load data**

```
# Preprocess data
X, y = shuffle(X, y, random_state=0)
scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train = X[:75]
y_train = y[:75]
X_test = X[75:]
y_test = y[75:]
```

**1.2 Preprocess data**
- shuffle
- scale
- split into train/test

```
# Select model
clf = linear_model.Perceptron()
```

**2. Select model**

```
# Train model
clf.fit(X_train, y_train)

print('weights:', clf.coef_)
print('bias:', clf.intercept_)
```

**3. Train model**

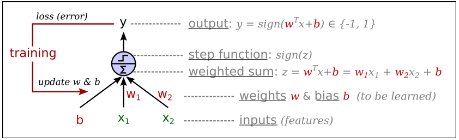
```
# Evaluate
y_pred = clf.predict(X_train)
accuracy_score(y_train, y_pred)
```

**4. Evaluate model**

```
# Predict from model
y_pred = clf.predict([[2, 0.5]])
```

**5. Predict from model**

```
# Plot data + linear classifier
#plt.scatter(X[:,0], X[:,1], c=y)
plt.scatter(X_train[:,0], X_train[:,1], c=y_train)
plt.scatter(X_test[:,0], X_test[:,1], c=y_test, alpha=.25)

weights = clf.coef_[0]
bias = clf.intercept_
slope = -weights[0] / weights[1]
yintercept = -bias / weights[1]
_x = np.linspace(-2,2)
_y = slope*_x + yintercept
plt.plot(_x, _y, '-r')
```



## Solution with Scikit-Learn: *Perceptron classifier*

```
from sklearn import datasets
from sklearn import linear_model
from sklearn.utils import shuffle
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
```

**1.1 Load data**

```
# Load data
iris = datasets.load_iris()
X = iris.data[:, (2, 3)] # petal length, petal width
y = (iris.target == 0).astype('int') # Iris setosa?
```

**1.2 Preprocess data**
- shuffle
- scale
- split into train/test

```
# Preprocess data
X, y = shuffle(X, y, random_state=0)
scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train = X[:75]
y_train = y[:75]
X_test = X[75:]
y_test = y[75:]
```

**2. Select model**

```
# Select model
clf = linear_model.Perceptron()
```

**3. Train model**

```
# Train model
clf.fit(X_train, y_train)

print('weights:', clf.coef_)
print('bias:', clf.intercept_)
```

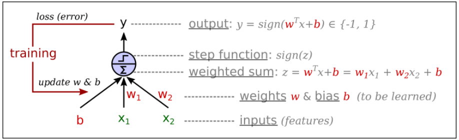
**4. Evaluate model**

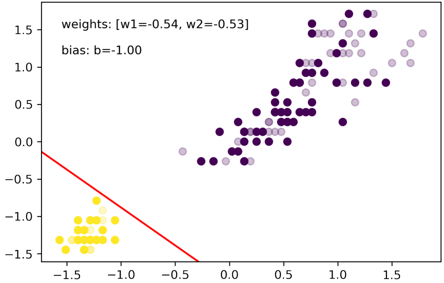
```
# Evaluate
y_pred = clf.predict(X_train)
accuracy_score(y_train, y_pred)
```

**5. Predict from model**

```
# Predict from model
y_pred = clf.predict([[2, 0.5]])
```

```
# Plot data + linear classifier
#plt.scatter(X[:,0], X[:,1], c=y)
plt.scatter(X_train[:,0], X_train[:,1], c=y_train)
plt.scatter(X_test[:,0], X_test[:,1], c=y_test, alpha=.25)

weights = clf.coef_[0]
bias = clf.intercept_
slope = -weights[0] / weights[1]
yintercept = -bias / weights[1]
_x = np.linspace(-2,2)
_y = slope*_x + yintercept
plt.plot(_x, _y, '-r')
```



## Solution with Tensor Flow - Keras: *1 neuron network*

```
import tensorflow as tf
from sklearn import datasets
from sklearn import linear_model
from sklearn.utils import shuffle
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
```

```
# Load data
iris = datasets.load_iris()
X = iris.data[:, (2, 3)] # petal length, petal width
y = (iris.target == 0).astype('int') # Iris setosa?
```

**1.1 Load data**

```
# Preprocess data
X, y = shuffle(X, y, random_state=0)
scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train = X[:75]
y_train = y[:75]
X_test = X[75:]
y_test = y[75:]
```

**1.2 Preprocess data**
- shuffle
- scale
- split into train/test

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(2,)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.summary()
```

**2.1 Build model**
- set layer type/order

```
# Compile model
model.compile(optimizer='sgd',
              loss='BinaryCrossentropy',
              metrics=['accuracy'])
```

**2.2 Compile model**
- set loss function
- set optimizer
- set metrics

```
# Train model
history = model.fit(X_train, y_train, epochs=50) #, batch_size=10)
```

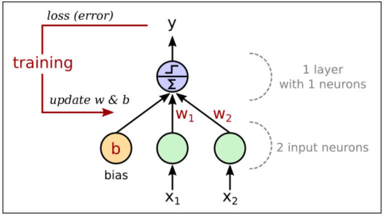
**3. Train model**
- learn layer parameters (weights/biases)
- plot training history (check for overfitting)

```
# Evaluate model
test_loss, test_acc = model.evaluate(X_test,  y_test, verbose=2)
print('Test accuracy:', test_acc)
```

**4. Evaluate model**
- evaluate accuracy on test dataset

```
# Predict (data should be preprocessed just like training data)
probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
pred = probability_model.predict([[-2, -2]])
print(pred)
```

**5. Predict from model**
- predict image class using learned model




```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 2)                 0
_________________________________________________________________
dense (Dense)                (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```

So if we can do the same thing, why switch from **sklearn** to **tensor flow** ?

**Tensor Flow** is a framework for **Deep Learning**

⇒ can design multi-layered networks, and train them in a very flexible/optimized manner

⇒ can solve much more complex problems, by optimizing several thousands/millions of weights during training!

So if we can do the same thing, why switch from **sklearn** to **tensor flow** ?

**Tensor Flow** is a framework for **Deep Learning**

$\Rightarrow$ can design multi-layered networks, and train them in a very flexible/optimized manner

$\Rightarrow$ can solve much more complex problems, by optimizing several thousands/millions of weights during training!

## **"Hello World" example in Keras TensorFlow**: MNIST fashion dataset classification task with MLP

```
import tensorflow as tf

# Load data
fashion_mnist = tf.keras.datasets.fashion_mnist

(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
X_valid, X_train = X_train_full[:5000], X_train_full[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]

# Preprocess data
X_train, X_test, X_valid = X_train/255.0, X_test/255.0, X_valid/255.0

# Build model (using the Sequential API)
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
model.summary()

# Compile model
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])

# Train model
history = model.fit(X_train, y_train, validation_data=(X_valid, y_valid),
                    epochs=30, # nb of times X_train is seen seen
                    batch_size=32) # nb of images per training instance
print('training instances per epoch = {}'.format(X_train.shape[0] / 32))

# Plot training history
import pandas as pd
pd.DataFrame(history.history).plot()

# Evaluate model
test_loss, test_acc = model.evaluate(X_test,  y_test)
print('Test accuracy:', test_acc)

# Predict
img = X_test[0,:,:]
img = (np.expand_dims(img,0)) # add image to a batch
y_proba = model.predict(img).round(2)
y_pred = np.argmax(model.predict(img), axis=-1)

plt.bar(range(10), y_proba[0])
plt.imshow(img[0,:,:], cmap='binary')
plt.title('class {} = {}'.format(y_pred, class_names[np.argmax(y_proba)]))
```

**1.1 Load data**
- training dataset
- validation dataset
- test dataset

**1.2 Preprocess data**
- scale pixel intensities to 0–1

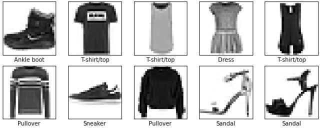
**2.1 Build model**
- set layer type/order

**2.2 Compile model**
- set loss function
- set optimizer
- set metrics

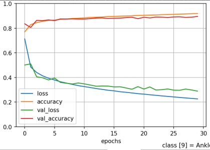
**3. Train model**
- learn layer parameters (weights/biases)
- plot training history (check for overfitting)

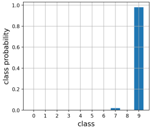
**4. Evaluate model**
- evaluate accuracy on test dataset

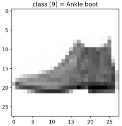
**5. Predict from model**
- predict image class using learned model




```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 300)               235500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```



## "Hello World" example in Keras TensorFlow: MNIST fashion dataset classification task with MLP

```
import tensorflow as tf

# Load data
fashion_mnist = tf.keras.datasets.fashion_mnist

(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
X_valid, X_train = X_train_full[:5000], X_train_full[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
```

**1.1 Load data**
- training dataset
- validation dataset
- test dataset

```
# Preprocess data
X_train, X_test, X_valid = X_train/255.0, X_test/255.0, X_valid/255.0
```

**1.2 Preprocess data**
- scale pixel intensities to 0–1

```
# Build model (using the Sequential API)
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
model.summary()
```

**2.1 Build model**
- set layer type/order

```
# Compile model
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

**2.2 Compile model**
- set loss function
- set optimizer
- set metrics

```
# Train model
history = model.fit(X_train, y_train, validation_data=(X_valid, y_valid),
                    epochs=30, # nb of times X_train is seen seen
                    batch_size=32) # nb of images per training instance
print('training instances per epoch = {}'.format(X_train.shape[0] / 32))

# Plot training history
import pandas as pd
pd.DataFrame(history.history).plot()
```

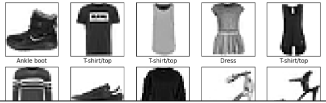
**3. Train model**
- learn layer parameters (weights/biases)
- plot training history (check for overfittin

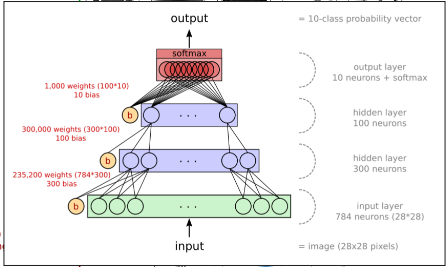
```
# Evaluate model
test_loss, test_acc = model.evaluate(X_test,  y_test)
print('Test accuracy:', test_acc)
```

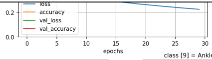
**4. Evaluate model**
- evaluate accuracy on test dataset

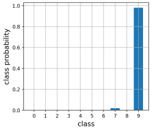
```
# Predict
img = X_test[0,:,:]
img = (np.expand_dims(img,0)) # add image to a batch
y_proba = model.predict(img).round(2)
y_pred = np.argmax(model.predict(img), axis=-1)

plt.bar(range(10), y_proba[0])
plt.imshow(img[0,:,:], cmap='binary')
plt.title('class {} = {}'.format(y_pred, class_names[np.argmax(y_proba)]))
```

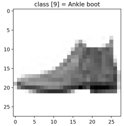
**5. Predict from model**
- predict image class using learned model



## Key parameters and definitions (from Google's ML glossary, Chollet 2017, etc.)

- **loss function** (objective function)

  The quantity that will be minimized during training. It represents a measure of success for the task at hand.

- **optimizer**

  Determines how the network will be updated based on the loss function. It implements a specific variant of stochastic gradient descent ( SGD ).

- **accuracy**

  The fraction of predictions that a classification model got right.

- **epoch**

  Each iteration over all the training data.

- **batch_size**

  Number of samples per gradient update.

- **activation function**

  A function (for example, ReLU or sigmoid) that takes in the weighted sum of all of the inputs from the previous layer and then generates and passes an output value (typically nonlinear) to the next layer.

- **softmax**

  A function that provides probabilities for each possible class in a multi-class classification model. The probabilities add up to exactly 1.0. For example, softmax might determine that the probability of a particular image being a dog at 0.9, a cat at 0.08, and a horse at 0.02.