Pattern Recognition
Introduction to TFLearn

Francesco Tortorella

University of Cassino and Southern Latium

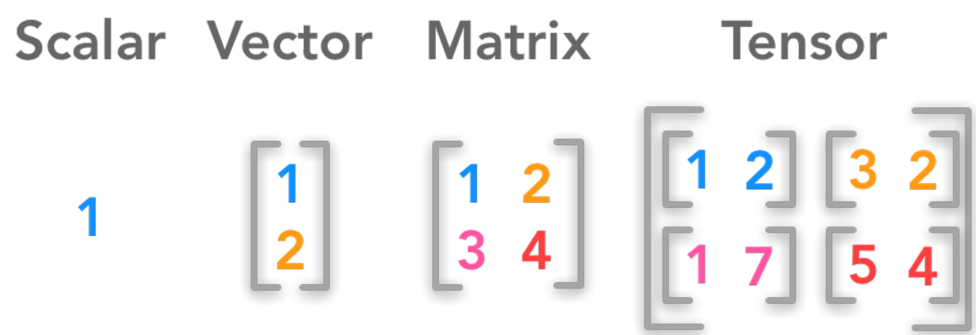
Cassino, Italy

TensorFlow

- Open source software library for numerical computation using data flow graphs
- Developed by Google Brain Team to conduct machine learning and deep neural networks research
- General enough to be applicable in a wide variety of other domains as well
- Apache 2.0 license
- Built on C++ with a Python interface

TensorFlow

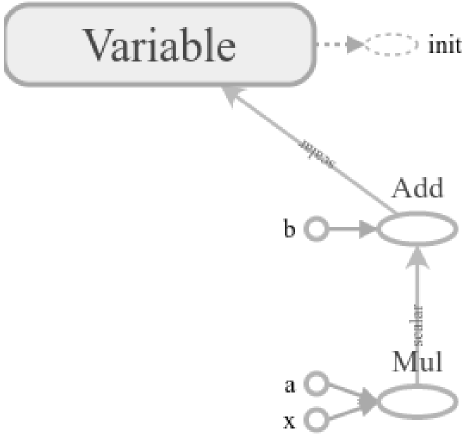
- Tensor



- Flow
  - TF is a library for *dataflow programming*, a programming paradigm that models a program as a directed graph of the data flowing between operations.

Dataflow Programming

- Each TF operation is performed through a data flow graph (computational graph)
- A directed graph made up of:
  - **Nodes**, representing operations (ex: sum of two integers)
  - **Directed Arcs**, representing data on which operations are performed




```
import tensorflow as tf

x = tf.constant(-2.0, name="x", dtype=tf.float32)
a = tf.constant(5.0, name="a", dtype=tf.float32)
b = tf.constant(13.0, name="b", dtype=tf.float32)

y = tf.Variable(tf.add(tf.multiply(a, x), b))
```

Dataflow Programming

- In TF computation is described using data flow graphs.
- Each node of the graph represents an instance of a mathematical operation (like addition, division, or multiplication) and each edge is a multi-dimensional data set (tensor) on which the operations are performed.

In summary

- TensorFlow is a powerful framework particularly suited for working with mathematical expressions
- Something fundamentally necessary in machine learning and specially in deep learning

TFlearn

- TFlearn is a modular and transparent deep learning library built on top of Tensorflow.
- Designed to provide a higher-level API to TensorFlow in order to facilitate and speed-up experimentations, while remaining fully transparent and compatible with it.

Some Tflearn features

- Easy-to-use and understand high-level API for implementing deep neural networks
- Fast prototyping through highly modular built-in neural network layers, regularizers, optimizers, metrics...
- Full transparency over Tensorflow. All functions are built over tensors and can be used independently of TFLearn.

Tflearn API

- Layers
- Built-in Operations
- Training, Evaluating & Predicting

Tflearn API → Layers

- Layers are a core feature of TFLearn.
- They represent an abstract set of operations to make building neural networks more convenient.
- For example, a convolutional layer will:
  - Create and initialize weights and biases variables
  - Apply convolution over incoming tensor
  - Add an activation function after the convolution
  - etc...

Tflearn API → Layers

File	Layers
core	input_data, fully_connected, dropout, custom_layer, reshape, flatten, activation, single_unit, highway, one_hot_encoding, time_distributed
conv	conv_2d, conv_2d_transpose, max_pool_2d, avg_pool_2d, upsample_2d, conv_1d, max_pool_1d, avg_pool_1d, residual_block, residual_bottleneck, conv_3d, max_pool_3d, avg_pool_3d, highway_conv_1d, highway_conv_2d, global_avg_pool, global_max_pool
recurrent	simple_rnn, lstm, gru, bidirectionnal_rnn, dynamic_rnn
embedding	embedding
normalization	batch_normalization, local_response_normalization, l2_normalize
merge	merge, merge_outputs
estimator	regression

Tflearn API → Layers → Core → Input Data

Input Data

tflearn.layers.core.input_data (shape=None, placeholder=None, dtype=tf.float32, data_preprocessing=None, data_augmentation=None, name='InputData')

This layer is used for inputting (aka. feeding) data to a network. A TensorFlow placeholder will be used if it is supplied, otherwise a new placeholder will be created with the given shape.

Tflearn API
→ Layers → Core → Fully connected

Fully Connected

tflearn.layers.core.fully_connected (incoming, n_units, activation='linear', bias=True, weights_init='truncated_normal', bias_init='zeros', regularizer=None, weight_decay=0.001, trainable=True, restore=True, reuse=False, scope=None, name='FullyConnected')

A fully connected layer.

Tflearn API
→ Layers → Core → Dropout

Dropout

tflearn.layers.core.dropout (incoming, keep_prob, noise_shape=None, name='Dropout')

Outputs the input element scaled up by `1 / keep_prob`. The scaling is so that the expected sum is unchanged.

By default, each element is kept or dropped independently. If noise_shape is specified, it must be broadcastable to the shape of x, and only dimensions with noise_shape[i] == shape(x)[i] will make independent decisions. For example, if shape(x) = [k, l, m, n] and noise_shape = [k, 1, 1, n], each batch and channel component will be kept independently and each row and column will be kept or not kept together.

Tflearn API
→ Layers → Estimators → Regression

Regression

tflearn.layers.estimator.regression (incoming, placeholder='default', optimizer='adam', loss='categorical_crossentropy', metric='default', learning_rate=0.001, dtype=tf.float32, batch_size=64, shuffle_batches=True, to_one_hot=False, n_classes=None, trainable_vars=None, restore=True, op_name=None, validation_monitors=None, validation_batch_size=None, name=None)

The regression layer is used in TFLearn to apply a regression (linear or logistic) to the provided input. It requires to specify a TensorFlow gradient descent optimizer 'optimizer' that will minimize the provided loss function 'loss' (which calculate the errors). A metric can also be provided, to evaluate the model performance.

Tflearn API
→ Built-in operations

- Besides layers concept, TFLearn also provides many different ops to be used when building a neural network.
- These ops are firstly mean to be part of the above 'layers' arguments, but they can also be used independently in any other Tensorflow graph for convenience.
- In practice, just providing the op name as argument is enough (such as activation='relu' or regularizer='L2' for conv_2d), but a function can also be provided for further customization.

Tflearn API
→ Built-in Operations

File	Ops
activations	linear, tanh, sigmoid, softmax, softplus, softsign, relu, relu6, leaky_relu, prelu, elu
objectives	softmax_categorical_crossentropy, categorical_crossentropy, binary_crossentropy, mean_square, hinge_loss, roc_auc_score, weak_cross_entropy_2d
optimizers	SGD, RMSProp, Adam, Momentum, AdaGrad, Ftrl, AdaDelta
metrics	Accuracy, Top_k, R2
initializations	zeros, uniform, uniform_scaling, normal, truncated_normal, xavier, variance_scaling
losses	l1, l2

Tflearn API
Built-in Operations typical uses

- Activations: used in the fully connected layers
- Objectives: the possible loss functions
- Optimizers: the algorithms for the GD
- Metrics: the metrics to be shown during training
- Inizializations: inizialization methods of the weights
- Losses: the norms for the regularization of the weights

Tflearn API
→Training, Evaluating & Predicting

- Training functions are another core feature of TFLearn.
- In TensorFlow, there are no pre-built API to train a network, so TFLearn integrates a set of functions that can easily handle any neural network training, whatever the number of inputs, outputs and optimizers.

Tflearn API
→Training, Evaluating & Predicting

Deep Neural Network Model

The model of the DNN is built

tflearn.models.dnn.DNN (network, clip_gradients=5.0, tensorboard_verbose=0, tensorboard_dir='/tmp/tflearn_logs/', checkpoint_path=None, best_checkpoint_path=None, max_checkpoints=None, session=None, best_val_accuracy=0.0)

tensorboard_verbose → network parameter visualization

```
0: Loss, Accuracy (Best Speed).
1: Loss, Accuracy, Gradients.
2: Loss, Accuracy, Gradients, Weights.
3: Loss, Accuracy, Gradients, Weights, Activations, Sparsity.(Best visualization)
```

Tflearn API
→Training, Evaluating & Predicting

Deep Neural Network Model

The model of the DNN is built

tflearn.models.dnn.DNN (network, clip_gradients=5.0, tensorboard_verbose=0, tensorboard_dir='/tmp/tflearn_logs/', checkpoint_path=None, best_checkpoint_path=None, max_checkpoints=None, session=None, best_val_accuracy=0.0)

tensorboard_verbose → network parameter visualization

```
0: Loss, Accuracy (Best Speed).
1: Loss, Accuracy, Gradients.
2: Loss, Accuracy, Gradients, Weights.
3: Loss, Accuracy, Gradients, Weights, Activations, Sparsity.(Best visualization)
```

Tflearn API
→Training, Evaluating & Predicting

Train the model, feeding X_inputs and Y_targets to the network.

```
fit (X_inputs, Y_targets, n_epoch=10, validation_set=None, show_metric=False,
batch_size=None, shuffle=None, snapshot_epoch=True, snapshot_step=None,
excl_trainops=None, validation_batch_size=None, run_id=None, callbacks=[])
```

Tflearn API
→Training, Evaluating & Predicting

Fit arguments

- **X_inputs**: array, `list` of array (if multiple inputs) or `dict` (with inputs layer name as keys). Data to feed to train model.
- **Y_targets**: array, `list` of array (if multiple inputs) or `dict` (with estimators layer name as keys). Targets (Labels) to feed to train model.
- **n_epoch**: `int`. Number of epoch to run. Default: None.
- **validation_set**: `tuple`. Represents data used for validation. `tuple` holds data and targets (provided as same type as X_inputs and Y_targets). Additionally, it also accepts `float` (<1) to performs a data split over training data.
- **show_metric**: `bool`. Display or not accuracy at every step.
- **batch_size**: `int` or None. If `int`, overrides all network estimators 'batch_size' by this value. Also overrides `validation_batch_size` if `int`, and if `validation_batch_size` is None.
- **validation_batch_size**: `int` or None. If `int`, overrides all network estimators 'validation_batch_size' by this value.

Tflearn API
→Training, Evaluating & Predicting

Fit arguments

- **shuffle**: `bool` or None. If `bool`, overrides all network estimators 'shuffle' by this value.
- **snapshot_epoch**: `bool`. If True, it will snapshot model at the end of every epoch. (Snapshot a model will evaluate this model on validation set, as well as create a checkpoint if 'checkpoint_path' specified).
- **snapshot_step**: `int` or None. If `int`, it will snapshot model every 'snapshot_step' steps.
- **excl_trainops**: `list` of `TrainOp`. A list of train ops to exclude from training process (TrainOps can be retrieve through `tf.get_collection_ref(tf.GraphKeys.TRAIN_OPS)`).
- **run_id**: `str`. Give a name for this run. (Useful for Tensorboard).
- **callbacks**: `Callback` or `list`. Custom callbacks to use in the training life cycle

Tflearn API
→Training, Evaluating & Predicting

Other methods

- `predict(X)`
- `predict_label(X)`
- `load(model_file)`
- `save(model_file)`

Building and training a DNN

- Insert the input data layer
- Insert the fully connected layers
  - For each connected layer a dropout layer could be inserted
- Insert the output layer
  - A fully connected layer with proper number of nodes and activation function
- Insert the regression layer
  - Previously define the optimizer, the metric, ...

Building and training a DNN

```
net = tflearn.input_data([None, 2])
net = tflearn.fully_connected(net, 3, activation='softmax')
gd = tflearn.SGD(learning_rate=1.0)
net = tflearn.regression(net, optimizer=gd, loss='categorical_crossentropy')

Y = to_categorical(y,3)
lm = tflearn.DNN(net,tensorboard_verbose=0)
lm.fit(X, Y, show_metric=True, batch_size=len(X), n_epoch=1000, snapshot_epoch=False)
```