Unidad 7: Herramientas y Aplicaciones

Curso: Redes Neuronales Profundas

Numpy

Image Manipulation

Theano

Theano is a **Python library**

that allows you to:

define,

optimize,

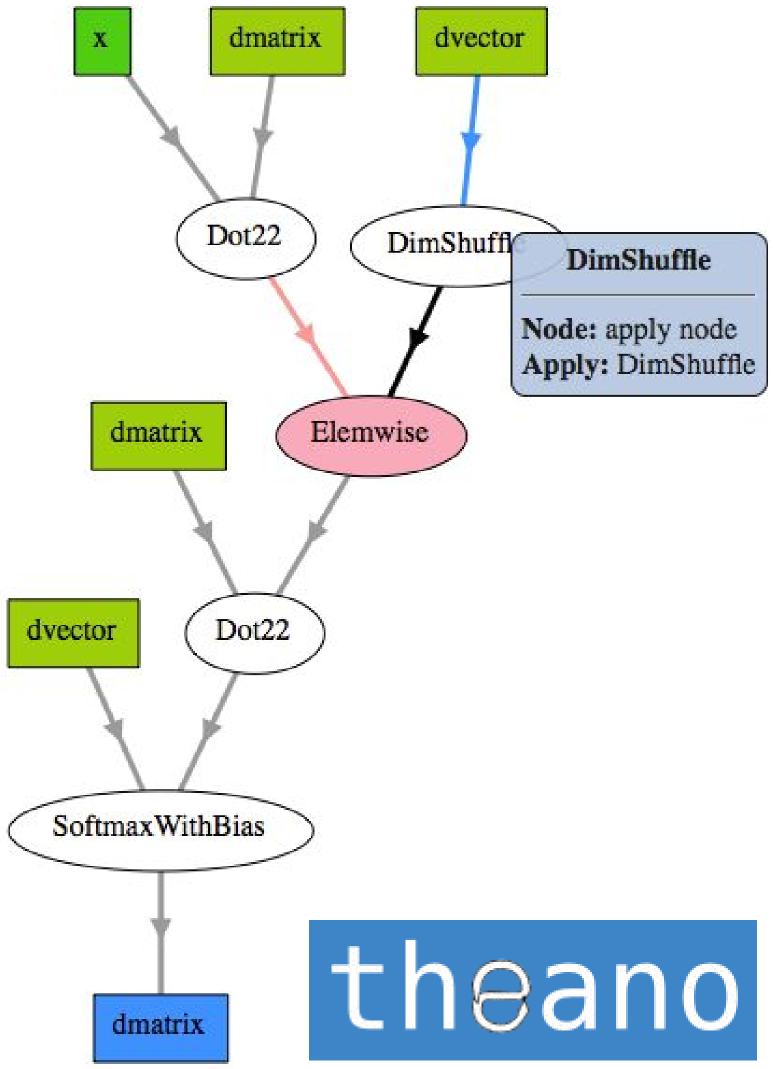
and *evaluate*

mathematical expressions

involving multi-dimensional

arrays efficiently.

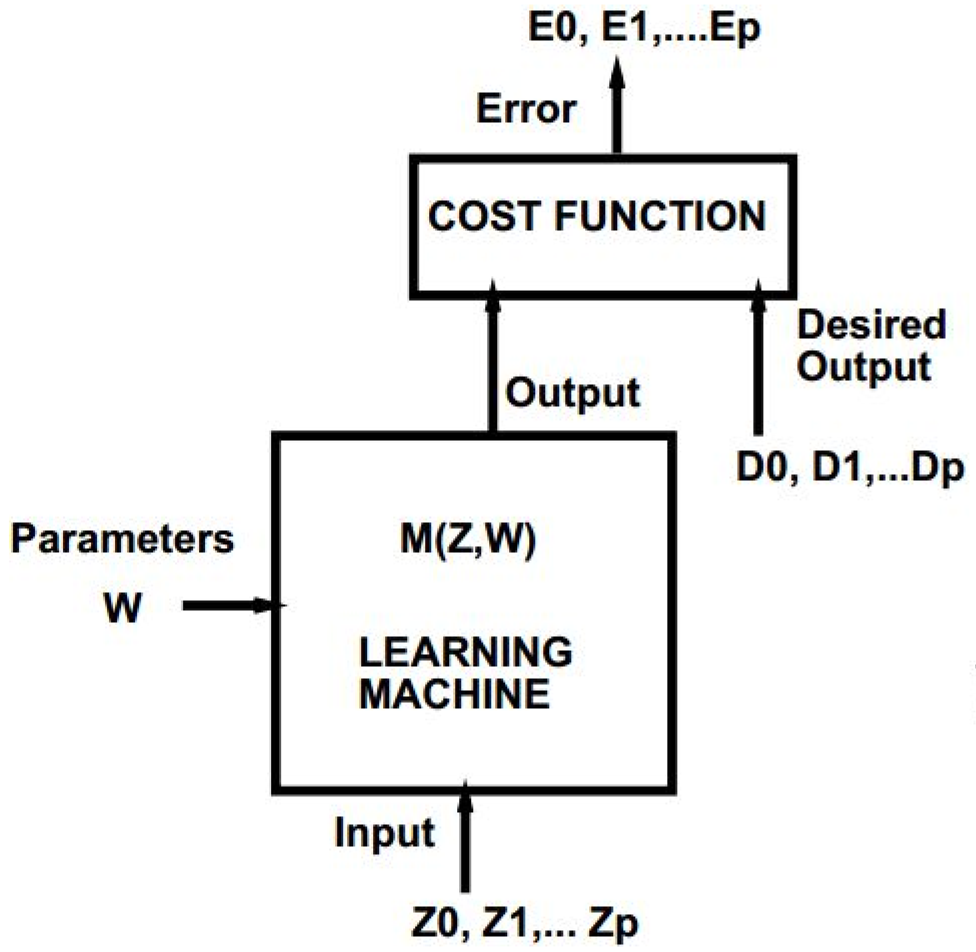
TENSORS



theano

¿Qué tiene que ver esto con deep learning?

Gradient Based Learning Machine



$$E^p = \frac{1}{2}(D^p - M(Z^p, W))^2$$

$$E_{train} = \frac{1}{P}\sum_{p=1} E^p$$

$$W_k = W_{k-1} - \epsilon \frac{\partial E(W)}{\partial W}$$

LeCun, Yann A et al. "Efficient backprop." *Neural networks: Tricks of the trade* (2012): 9-48.

Primeros pasos en Theano

```python
import numpy
import theano.tensor as T
from theano import function

x = T.matrix('x',dtype='float32')
y = T.matrix('y',dtype='float32')
z = x + y
f = function([x, y], z)

f([[1, 2], [3, 4]], [[10, 20], [30, 40]])
```

```
array([[ 11.,  22.],
       [ 33.,  44.]])
```

Primeros pasos en Theano

```
x.type
```

```
TensorType(float32, matrix)
```

```
from theano import pp
print(pp(z))
```

```
(x + y)
```

Tipos de variables de tensores

- **byte**: `bscalar, bvector, bmatrix, brow, bcol, btensor3, btensor4`
- **16-bit integers**: `wscalar, wvector, wmatrix, wrow, wcol, wtensor3, wtensor4`
- **32-bit integers**: `iscalar, ivector, imatrix, irow, icol, itensor3, itensor4`
- **64-bit integers**: `lscalar, lvector, lmatrix, lrow, lcol, ltensor3, ltensor4`
- **float**: `fscalar, fvector, fmatrix, frow, fcol, ftensor3, ftensor4`
- **double**: `dscalar, dvector, dmatrix, drow, dcol, dtensor3, dtensor4`
- **complex**: `cscalar, cvector, cmatrix, crow, ccol, ctensor3, ctensor4`

```
> THEANO_FLAGS="floatX=float32,devise=cpu" ipython
```

Más ejemplos: ReLU layer

```
import theano
import theano.tensor as T
def relu(x):
    return T.switch(x>0.0,x,0.0)

x = T.vector('x')
W = T.matrix('W')
b = T.vector('b')
z = T.dot(x,W) + b
y = relu(z)

layer = theano.function([x,W,b], y)
# layer = theano.function([x], [z,y])
layer([3,1],
      [[1,0,0],[0,-1,0]],[1,-2,3])
```

```
array([ 4.,  0.,  3.])
```

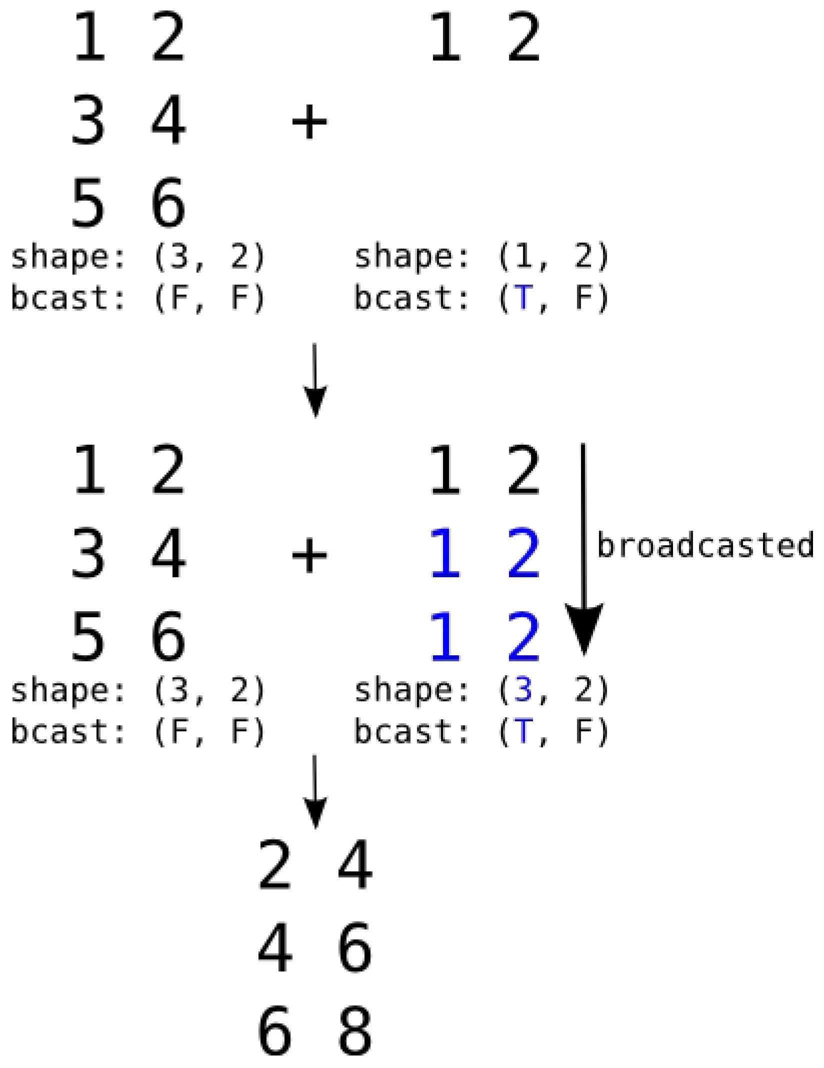
Más ejemplos: ReLU layer - Broadcast

```
import theano
import theano.tensor as T
def relu(x):
    return T.switch(x>0.0,x,0.0)

x = T.matrix('x')
W = T.matrix('W')
b = T.vector('b')
z = T.dot(x,W) + b
y = relu(z)

layer = theano.function([x,W,b], y)
# layer = theano.function([x], [z,y])
layer([[3,1],[1,3]],
      [[1,0,0],[0,-1,0]],[1,-2,3])
```

```
array([[ 4.,  0.,  3.],[ 2.,  0.,  3.]])
```

Broadcasting

- A diferencia de Numpy, en Theano es necesario indicar de antemano qué índices son *transmisibles (broadcasteables)*
- Si un índice es transmisible, entonces la dimensión en ese índice debe ser 1.
- Si tengo menor número de índices, se completa *bcast* con True's a la izquierda y *shape* con 1's a la izquierda



Reglas de broadcasting

Constructor	ndim	shape	broadcastable
scalar	0	()	()
vector	1	(?,)	(False,)
row	2	(1,?)	(True, False)
col	2	(?,1)	(False, True)
matrix	2	(?,?)	(False, False)
tensor3	3	(?,?,?)	(False, False, False)
tensor4	4	(?,?,?,?)	(False, False, False, False)

```
tensor5 = T.TensorType(dtype='float32',
                       broadcastable=(False,)*5)
x5 = tensor5('x5')
x5.type
```

```
TensorType(float32, 5D)
```

Shared Variables

```python
from theano import shared

state = shared(0)
inc = T.iscalar('inc')
accumulator = function([inc], state,
                       updates=[(state, state+inc)])

state.set_value(10)

print(state.get_value())
```

```
10
```

Random Numbers

```
from theano.tensor.shared_randomstreams import RandomStreams
from theano import function

srng = RandomStreams(seed=234)
rv_u = srng.uniform((2,))
rv_n = srng.normal((2,))

f = function([], rv_u)
g = function([], [rv_n,rv_u], no_default_updates=True)
d = function([], [2*rv_u,rv_u+rv_u])
```

Random Numbers

```
f()
```

```
array([ 0.12672381,  0.97091597], dtype=float32)
```

```
f()
```

```
array([ 0.13989098,  0.88754827], dtype=float32)
```

```
g()
```

```
[array([ 0.37328446, -0.65746671], dtype=float32),
 array([ 0.31971416,  0.47584376], dtype=float32)]
```

```
g()
```

```
[array([ 0.37328446, -0.65746671], dtype=float32),
 array([ 0.31971416,  0.47584376], dtype=float32)]
```

Random Numbers

```
g()
```

```
[array([ 0.37328446, -0.65746671], dtype=float32),
 array([ 0.31971416,  0.47584376], dtype=float32)]
```

```
f()
```

```
array([ 0.31971416,  0.47584376], dtype=float32)
```

```
f()
```

```
array([ 0.24129163,  0.42046082], dtype=float32)
```

```
d()
```

```
[array([ 0.48258325,  0.84092164], dtype=float32),
 array([ 0.48258325,  0.84092164], dtype=float32)]
```

Gradiente

```
from theano import pp
x = T.dscalar('x')
y = x ** 2
gy = T.grad(y, x)
theano.pp(gy)
```

```
'((fill((x ** TensorConstant{2}), TensorConstant{1.0})
* TensorConstant{2}) * (x ** (TensorConstant{2} -
TensorConstant{1})))'
```

```
g = theano.function([x], gy)
g(4)
```

```
array(8.0)
```

```
theano.pp(g.maker.fgraph.outputs[0])
```

```
'(TensorConstant{2.0} * x)'
```

Reducciones

```
x = T.tensor3()
total = x.sum()
marginals = x.sum(axis=(0, 2))
mx = x.max(axis=1)
```

Dimshuffle

```
y = x.dimshuffle((2, 1, 0))

a = T.matrix()
b = a.T
c = a.dimshuffle((1, 0)) # Same as b
d = a.dimshuffle((0, 1, 'x'))
e = a + d
f = function([a],[b,c,d,e])
```

Ejemplo: regresión logística

```python
from numpy import random as rng
import theano
import theano.tensor as T
feats = 784 # number of input variables

x = T.dmatrix("x")
y = T.dvector("y")
w = theano.shared(rng.randn(feats), name="w")
b = theano.shared(0., name="b")

p_1 = 1 / (1 + T.exp(-T.dot(x, w) - b)) # Prob. that target = 1
prediction = p_1 > 0.5
xent = -y * T.log(p_1) - (1-y) * T.log(1-p_1) # Cross-entropy
cost = xent.mean() + 0.01 * (w ** 2).sum()
gw, gb = T.grad(cost, [w, b])
train = theano.function(
          inputs=[x,y],
          outputs=[prediction, xent, cost],
          updates=((w, w - 0.1 * gw), (b, b - 0.1 * gb)))
predict = theano.function(inputs=[x], outputs=prediction)
```

Ejemplo: regresión logística

```
# generate a dataset: D = (input_values, target_class)
N = 400                                         # training sample size
D = (rng.randn(N, feats), rng.randint(size=N, low=0, high=2))
training_steps = 100

# Train
for i in range(training_steps):
    pred, err, costv = train(D[0], D[1])
    print costv

print("Final model:")
print(w.get_value())
print(b.get_value())
print("target values for D:")
print(D[1])
print("prediction on D:")
print(predict(D[0]))
```

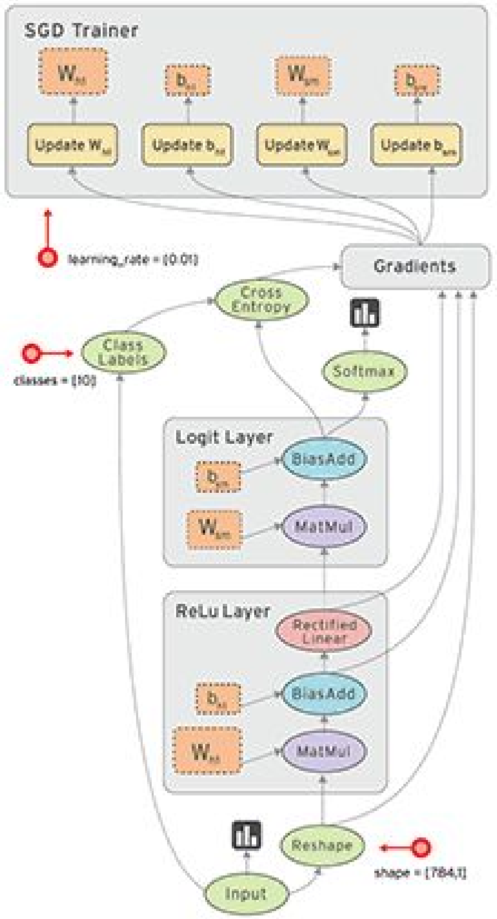
TensorFlow

"TensorFlow is an
open source software library
for numerical computation
using **data flow graphs**.

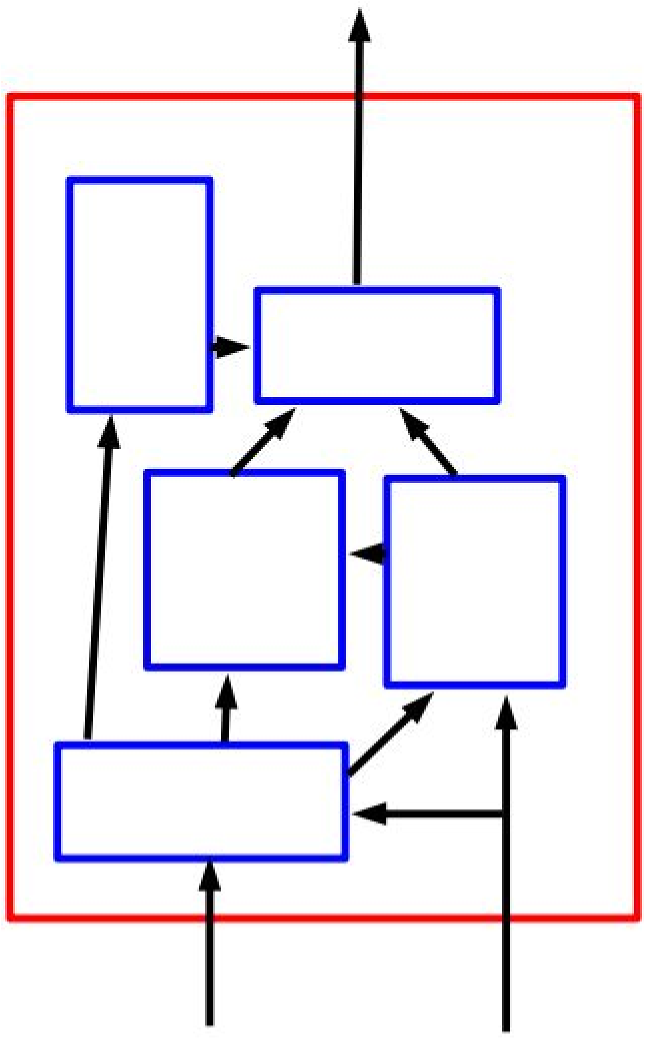
Nodes in the graph represent mathematical operations, while the graph **edges** represent the multidimensional data arrays (tensors).

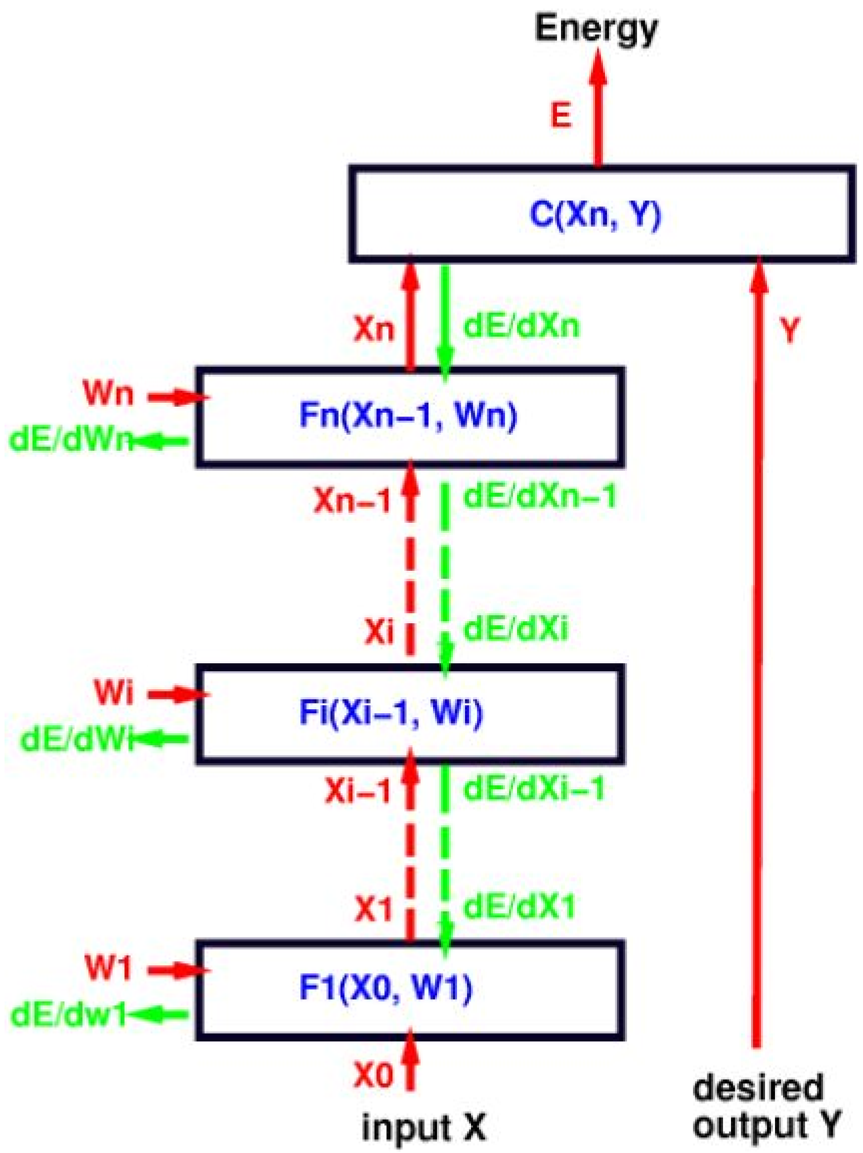
To deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API."

https://www.tensorflow.org/



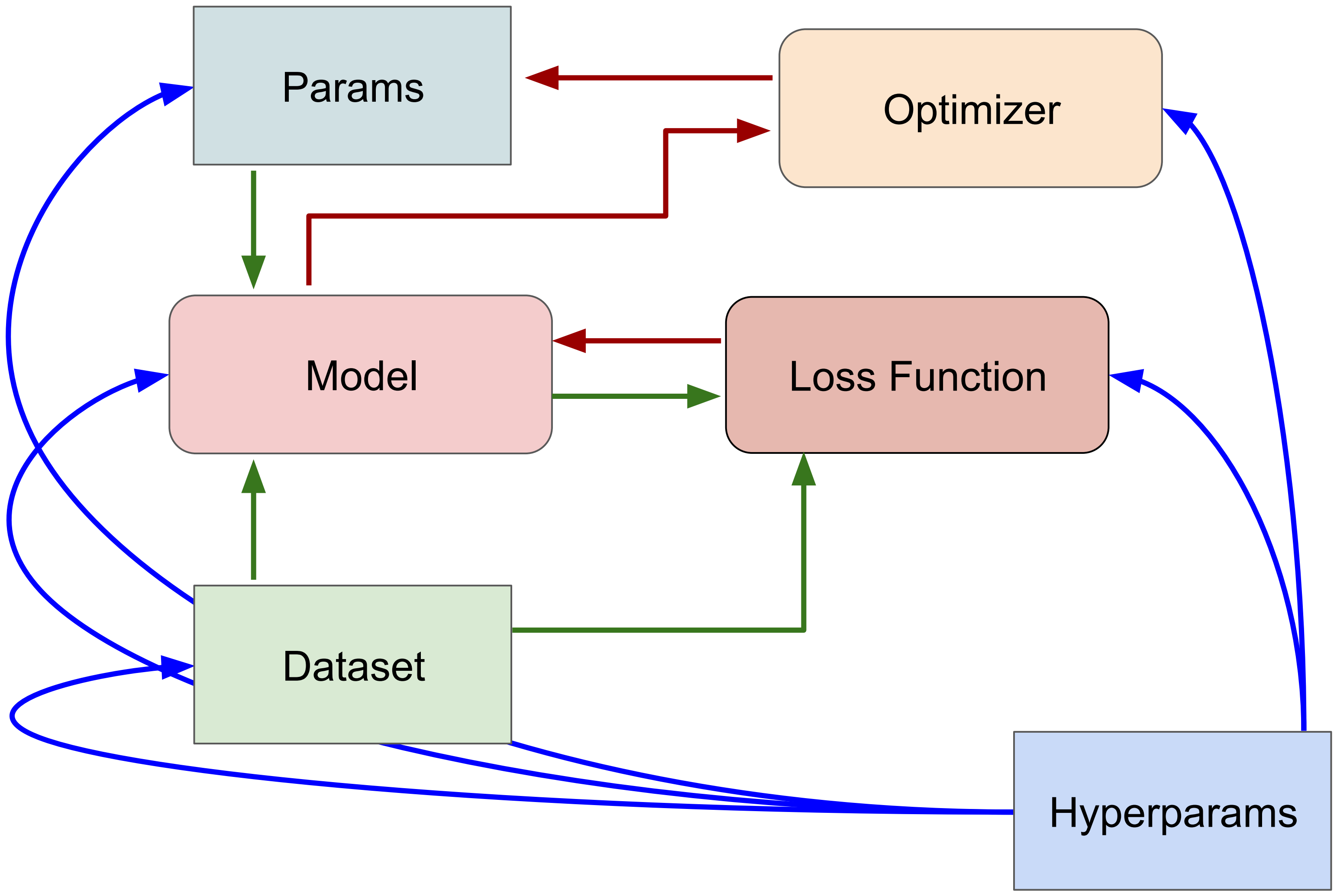
Visión Modular





http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf

Visión Modular



Deep Learning Frameworks

- Theano Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently
- TensorFlow Library for numerical computation using data flow graphs.

- Keras Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano.
- Blocks Blocks is a framework that helps you build and manage neural network models on using Theano.
- Lasagne Lasagne is a lightweight library to build and train neural networks in Theano.
- Pylearn2 This project does not have any current developer.

- Caffe Deep learning framework made with expression, speed, and modularity in mind.

- Torch7 Scientific computing framework with wide support for machine learning algorithms