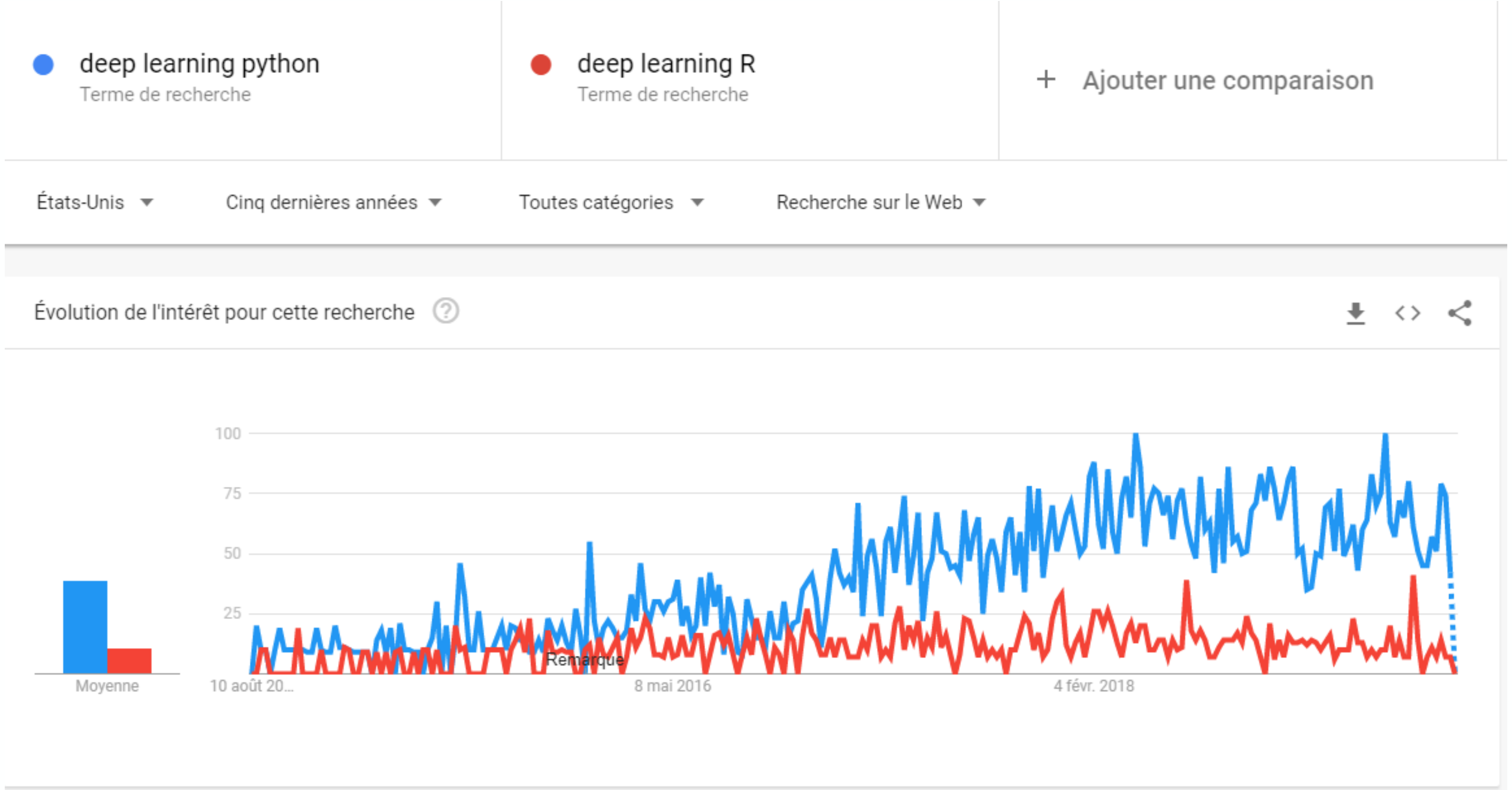
Deep LeaRning

Y a pas que le Python dans la vie

Valeuriad

Richesses humaines & expertises IT

Deep Learning: Python ou R ?



Deep Learning: Python ou R ?

- Python dispose d'un écosystème très complet (pytorch, keras, tensorflow)
- Certainement l'outil de choix
- MAIS : R permet également de faire du deep learning
  - Finalement tout termine sur tensorflow (ou autre): langage natif (compilé)
  - Les APIs de haut niveau comme Keras et Pytorch permettent de décrire les réseaux de neurones
  - Le gros du calcul se fait en langage natif

Deep Learning

Besoin d’un rappel ?

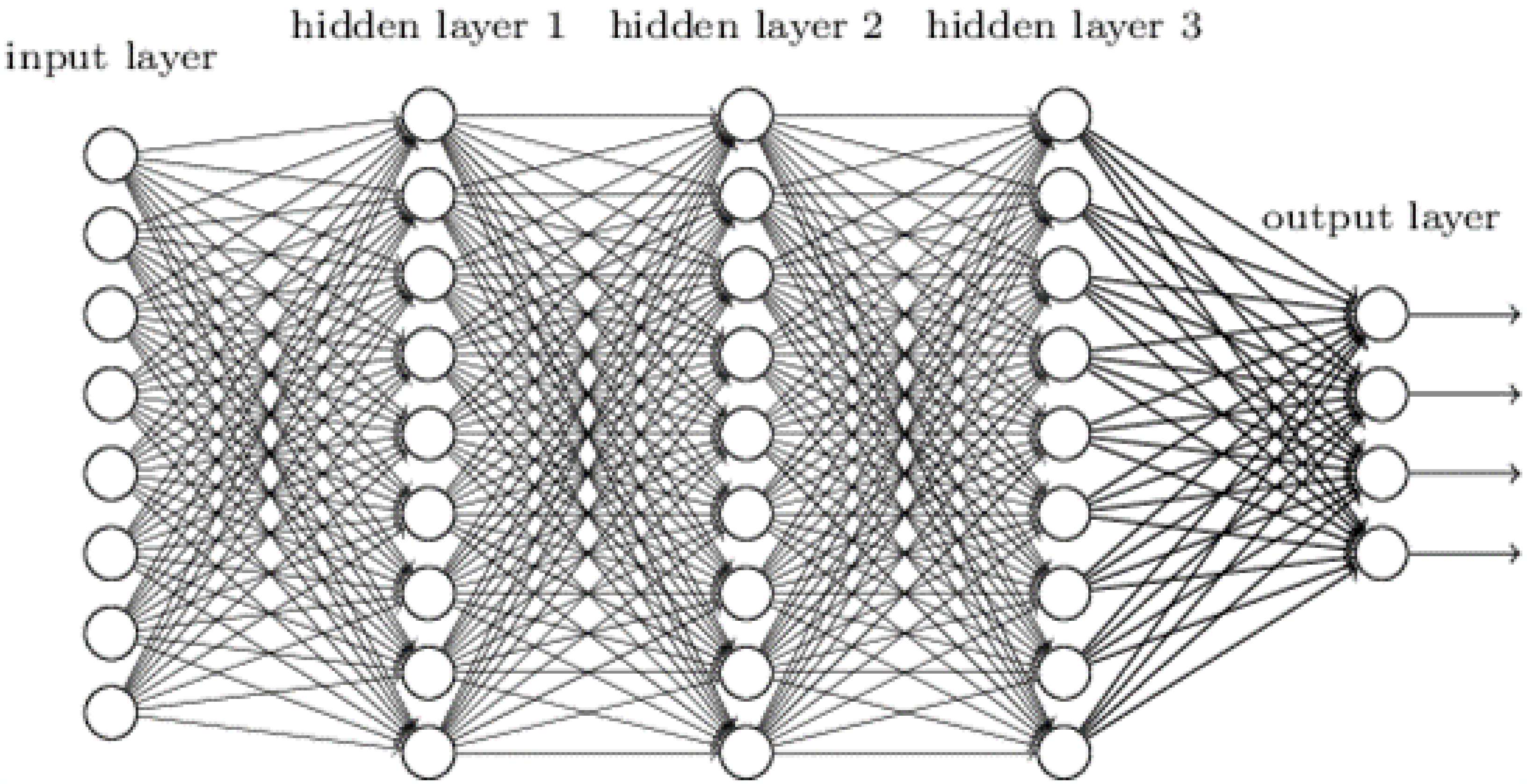
Deep Learning

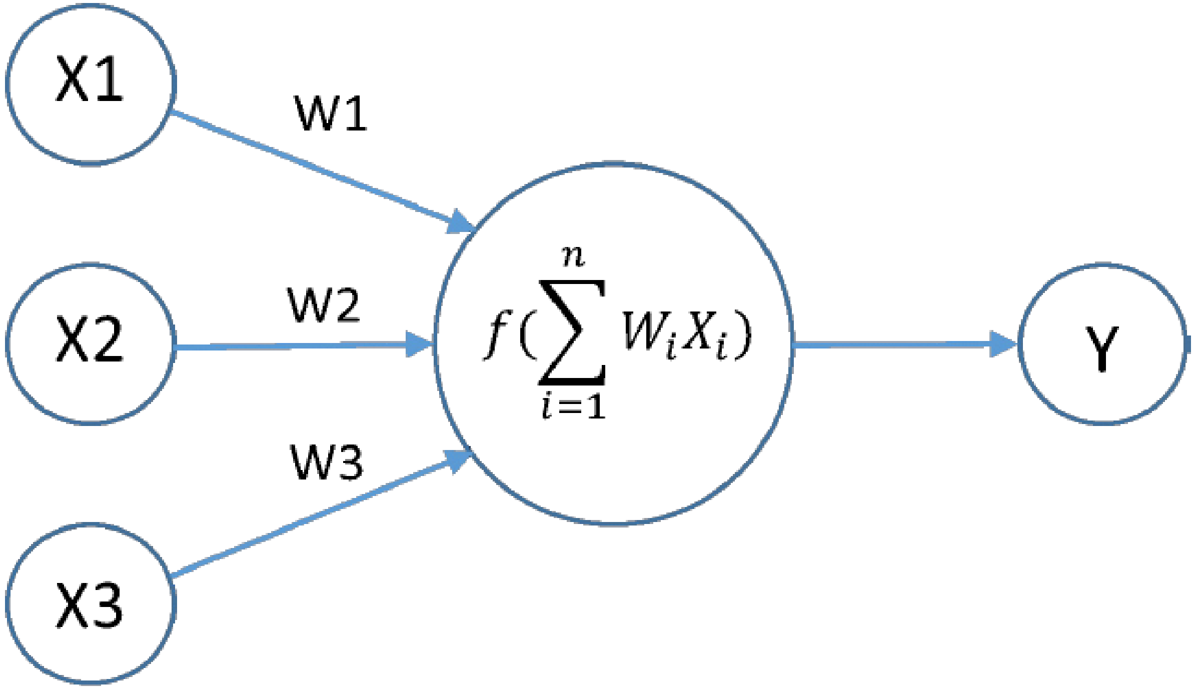
Historique: Evolution des réseaux de neurones à l'heure du PaaS

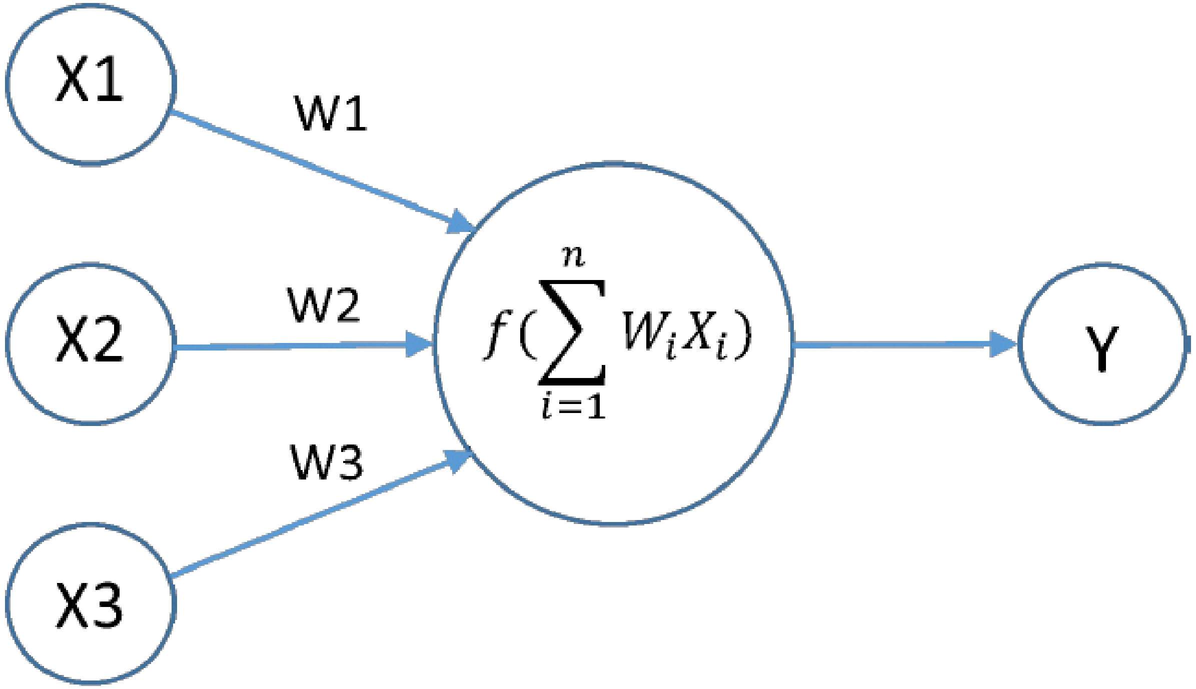
- Algorithme d'apprentissage
- Inspiré (grossièrement) du cerveau humain
- Perceptron en 1957 par F. Rosenblatt

Valeuriad

Deep neural network

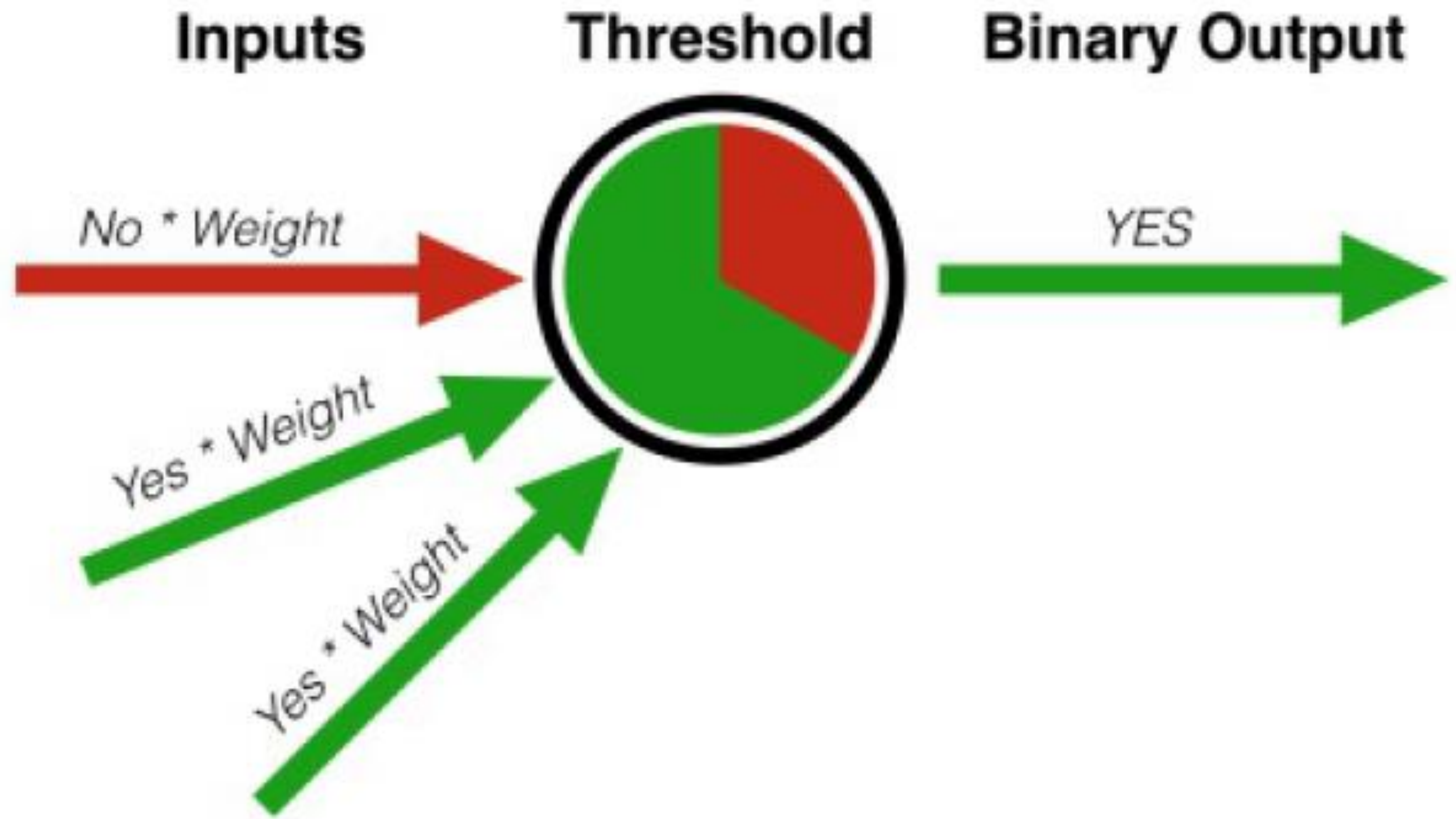



X1
W1
X2
W2
$f(\sum_{i=1}^{n} W_i X_i)$
Y
W3
X3



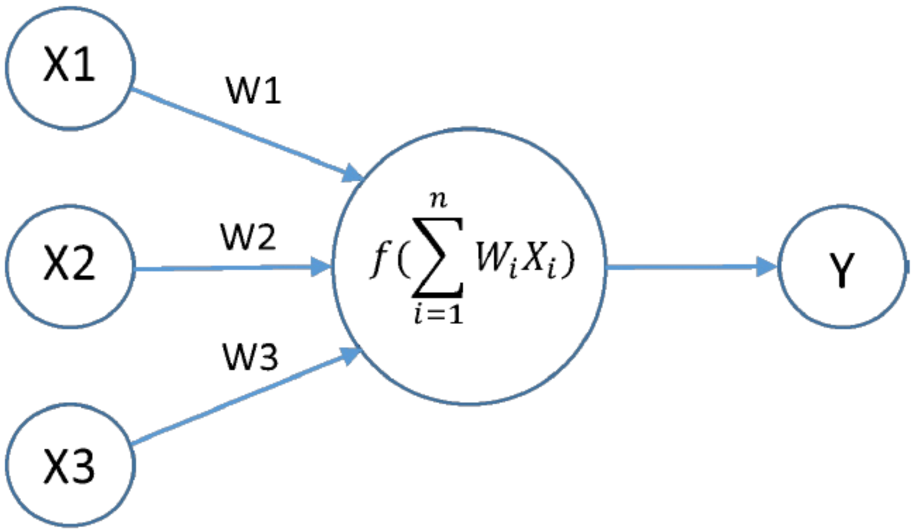
Intuition:

- Si la somme des entrées tend vers +∞ alors la sortie tend vers 1
- Si la somme des entrées tend vers -∞ alors la sortie tend vers 0


Inputs
Threshold
Binary Output
No * Weight
Yes * Weight
Yes * Weight
YES

Backpropagation

1. On calcule les sorties à partir des entrées (données en input)
2. On mesure l’erreur (par rapport à la cible que l’on connait)
3. On fait remonter l’erreur dans le réseau via « backpropagation » (dérivées partielles et on modifie les poids en conséquence

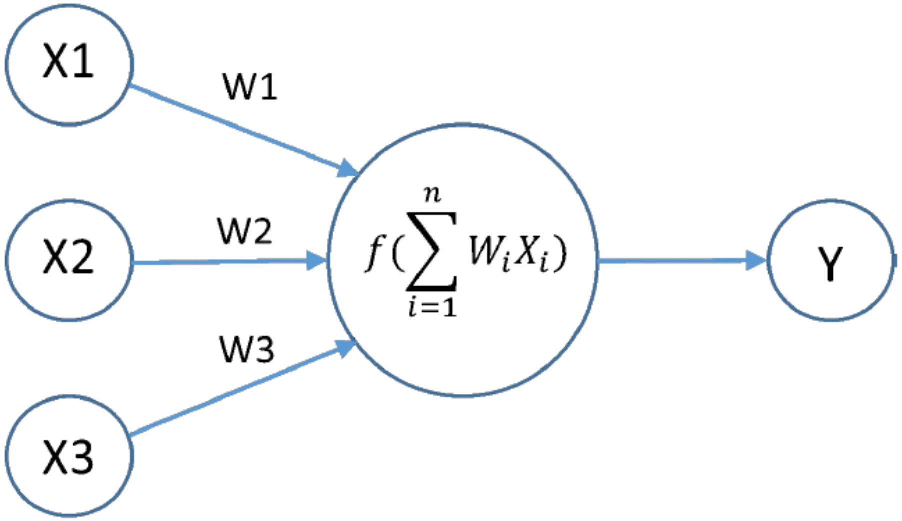


Exemple

Longueur (m)	Poilu ? (1 ou 0)	Mignon ? (1 ou 0)		Chat ? (1 ou 0)
0.4	1	1		1
0.3	1	1		1
0.3	1	0		0
0.4	1	1		1

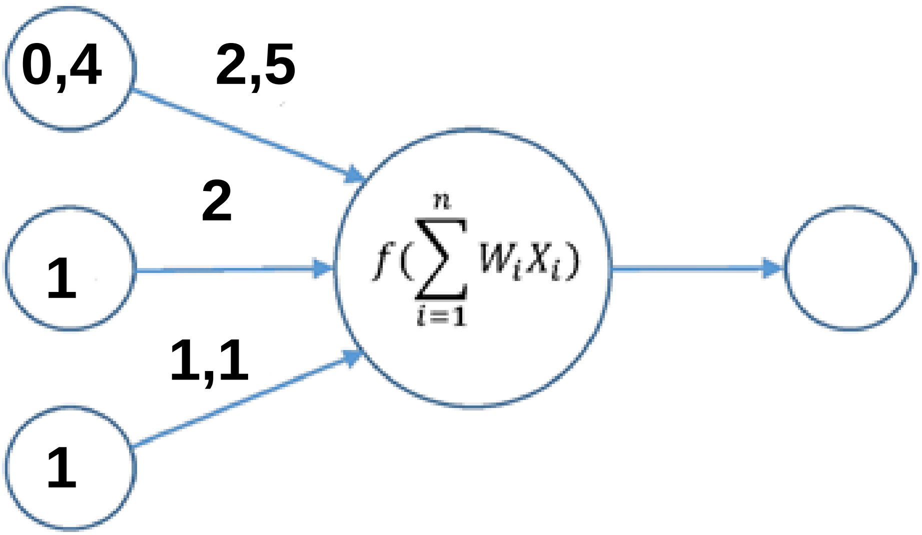
Exemple

Longueur (m)	Poilu ? (1 ou 0)	Mignon ? (1 ou 0)		Chat ? (1 ou 0)
0.4	1	1		1
0.3	1	1		1
0.3	1	0		0
0.4	1	1		1



Exemple

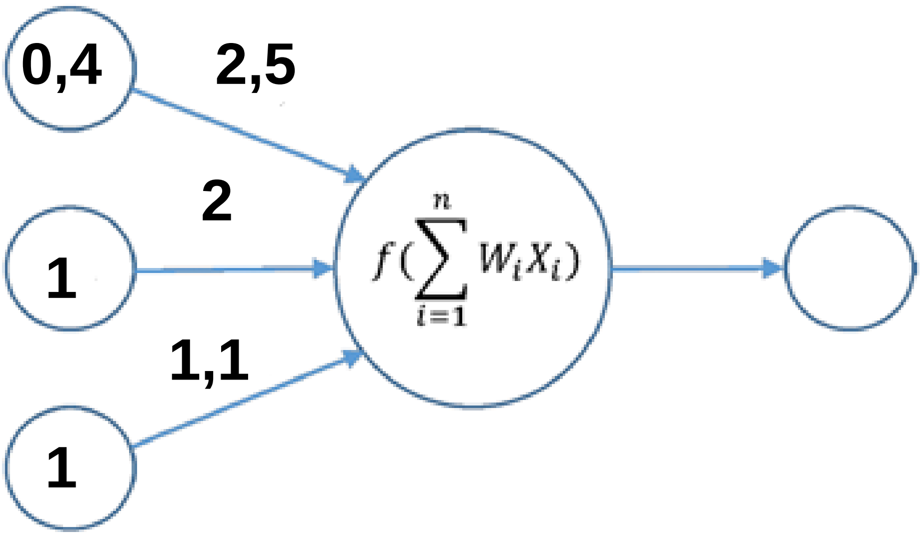
Longueur (m)	Poilu ? (1 ou 0)	Mignon ? (1 ou 0)		Chat ? (1 ou 0)
0.4	1	1		1
0.3	1	1		1
0.3	1	0		0
0.4	1	1		1



Valeuriad

Exemple

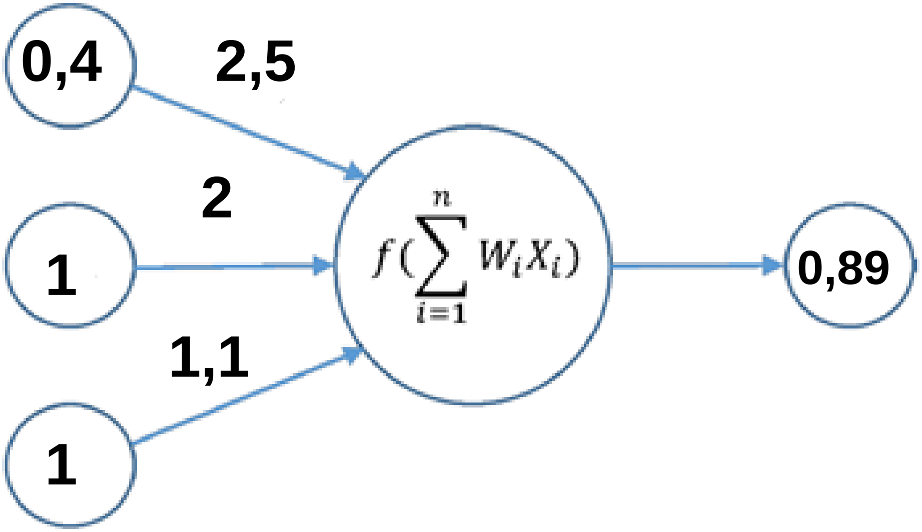
Longueur (m)	Poilu ? (1 ou 0)	Mignon ? (1 ou 0)		Chat ? (1 ou 0)
0.4	1	1		1
0.3	1	1		1
0.3	1	0		0
0.4	1	1		1



$$f(0.4 * 2.5 + 2 * 1 + 1.1 * 1)$$
$$= f(2.1)$$
$$= 0.89$$

Exemple

Longueur (m)	Poilu ? (1 ou 0)	Mignon ? (1 ou 0)		Chat ? (1 ou 0)
0.4	1	1		1
0.3	1	1		1
0.3	1	0		0
0.4	1	1		1

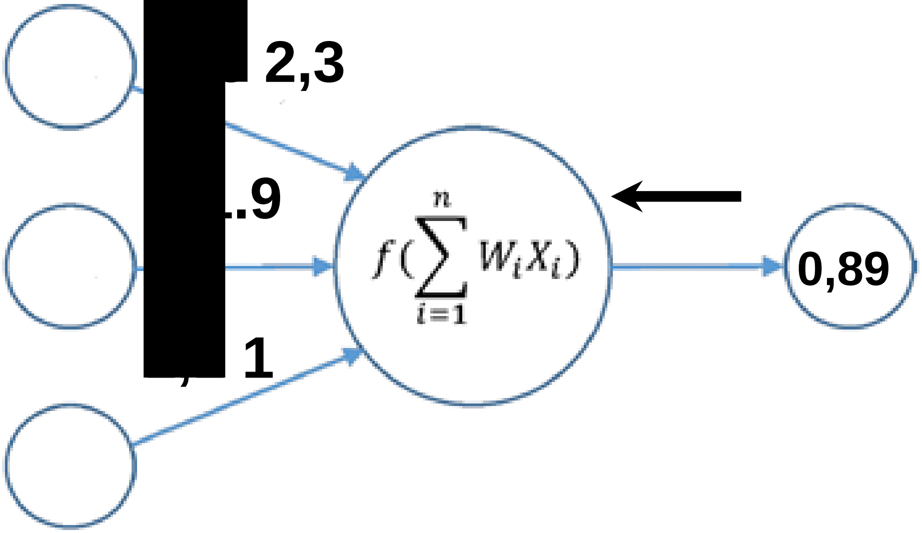


$$error = 1 - 0.89$$
$$error = 0.11$$

Valeuriad

Exemple

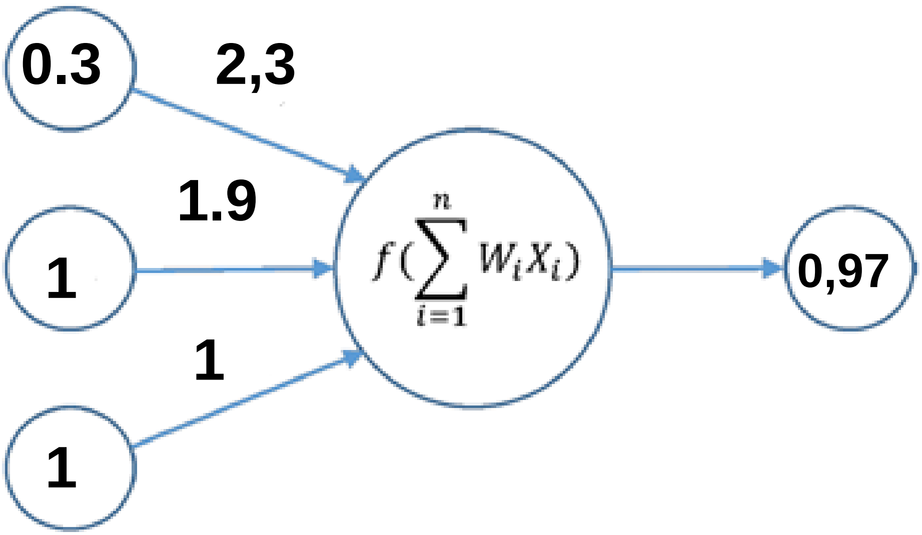
Longueur (m)	Poilu ? (1 ou 0)	Mignon ? (1 ou 0)		Chat ? (1 ou 0)
0.4	1	1		1
0.3	1	1		1
0.3	1	0		0
0.4	1	1		1



$$error = 1 - 0.89$$
$$error = 0.11$$

Exemple

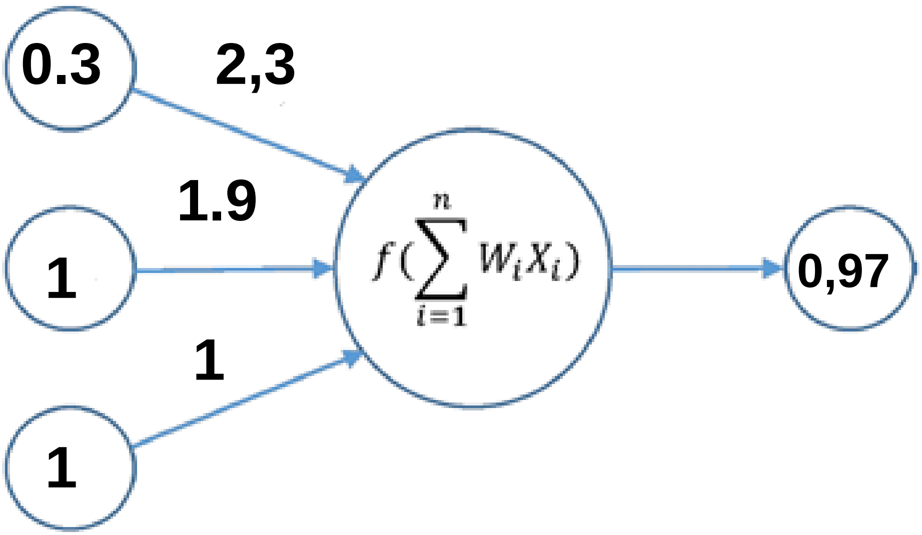
Longueur (m)	Poilu ? (1 ou 0)	Mignon ? (1 ou 0)		Chat ? (1 ou 0)
0.4	1	1		1
0.3	1	1		1
0.3	1	0		0
0.4	1	1		1



$$f(0.3 * 2.3 + 1.9 * 1 + 1 * 1)$$
$$= f(3.59)$$
$$= 0.97$$

Exemple

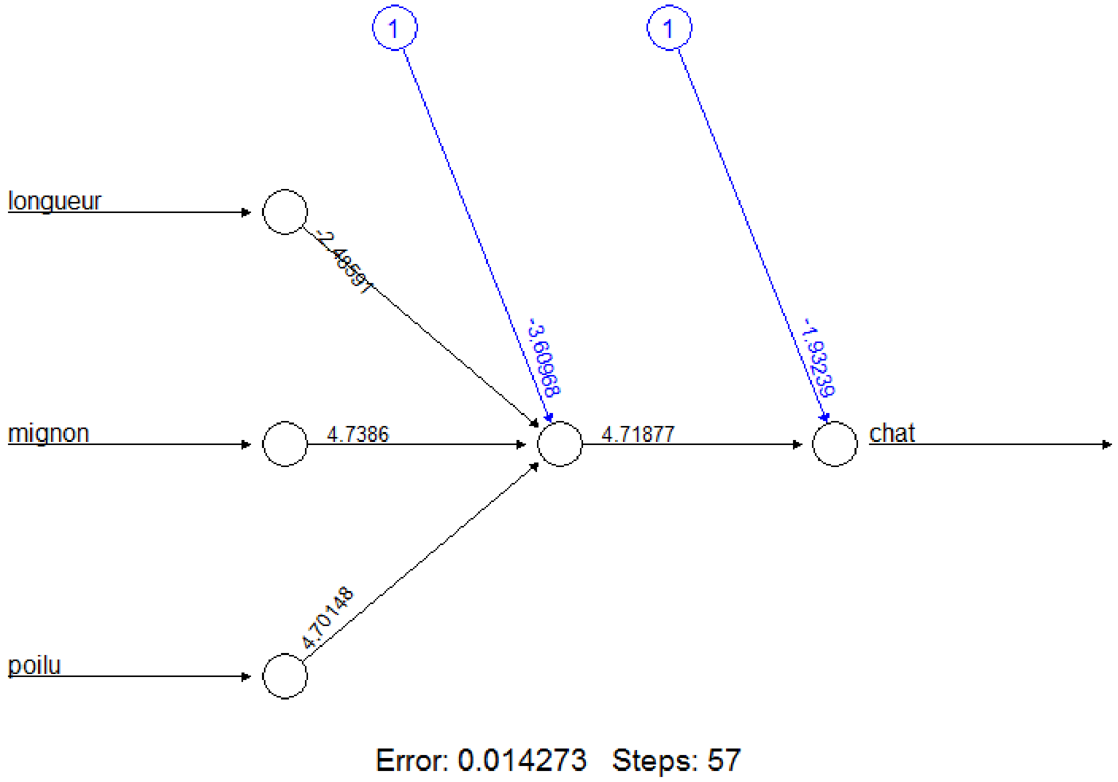
Longueur (m)	Poilu ? (1 ou 0)	Mignon ? (1 ou 0)		Chat ? (1 ou 0)
0.4	1	1		1
0.3	1	1		1
0.3	1	0		0
0.4	1	1		1



Etc …

Exemple

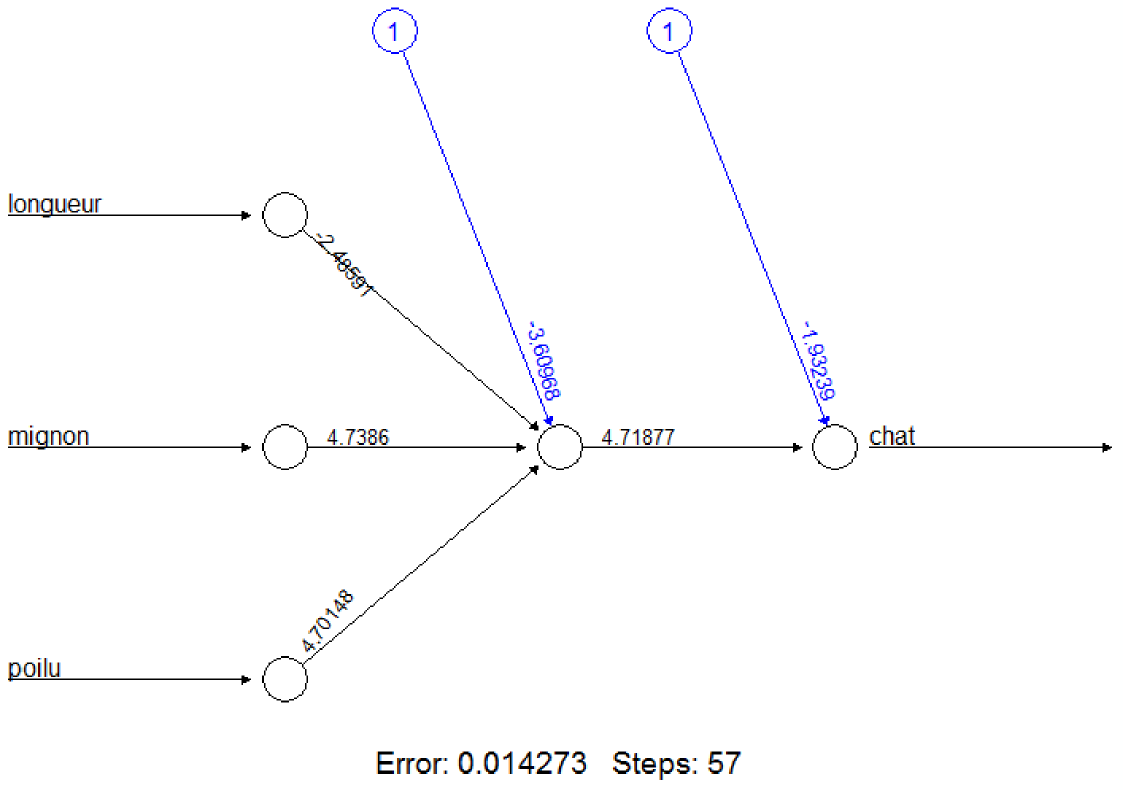
```r
install.packages("neuralnet")
library("neuralnet")

chat_d = data.frame(longueur = c(0.4, 0.3, 0.3, 0.4),
                    mignon   = c(1,   1,   0,   1),
                    poilu    = c(1,   1,   0,   1),
                    chat     = c(1,   1,   0,   1))

nn <- neuralnet(
  chat~longueur+mignon+poilu,
  data=chat_d, hidden=1, err.fct="sse",
  linear.output=FALSE)

plot(nn)
```



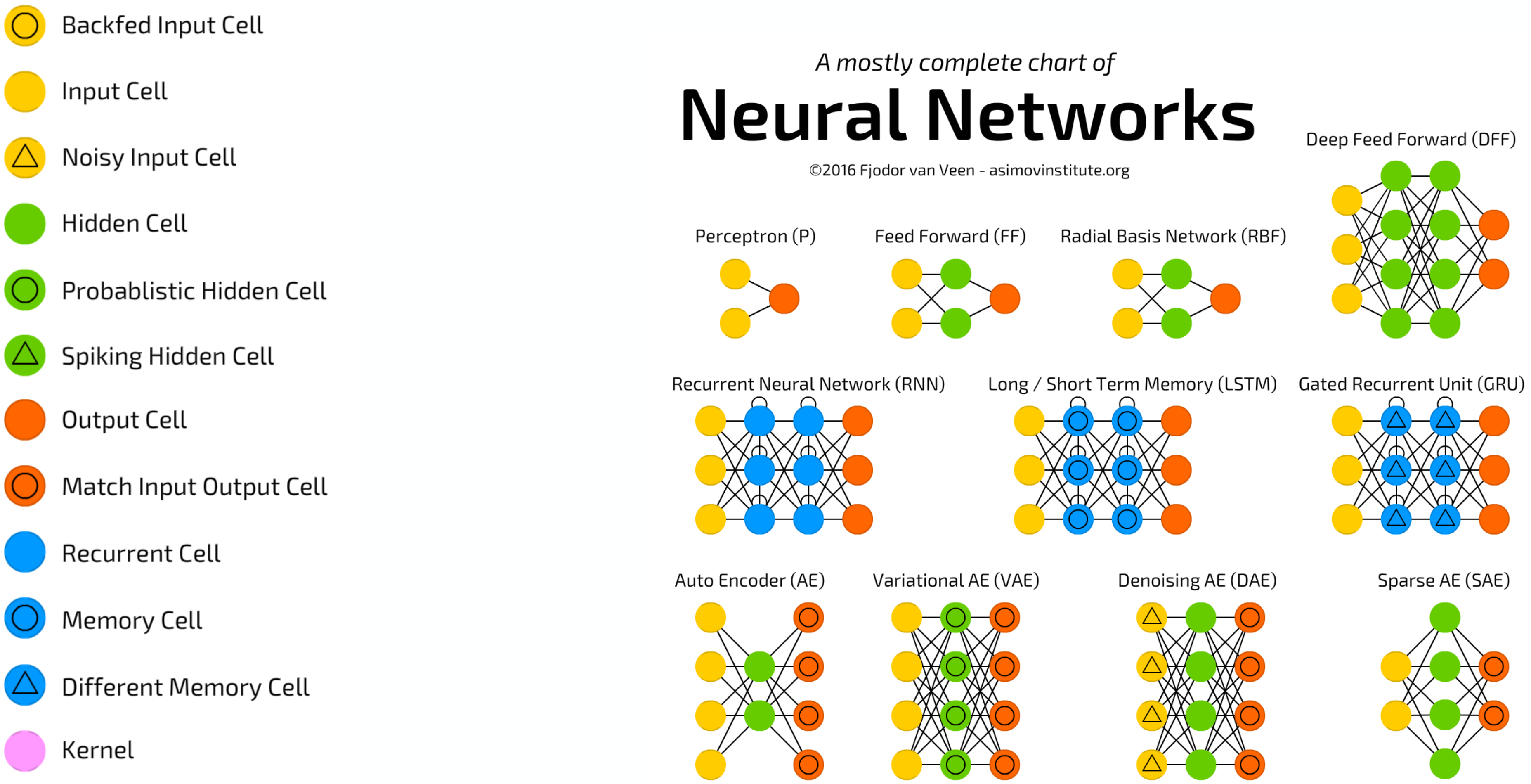
Exemple

```
install.packages("neuralnet")
library("neuralnet")

chat_d = data.frame(longueur = c(0.4, 0.3, 0.3, 0.4),
                    mignon   = c(1,     1,   0,   1),
                    poilu    = c(1,     1,   0,   1),
                    chat     = c(1,     1,   0,   1))

nn <- neuralnet(
  chat~longueur+mignon+poilu,
  data=chat_d, hidden=1, err.fct="sse",
  linear.output=FALSE)

plot(nn)
```

```
> res <- compute(nn, data.frame(longueur = 0.7, mignon = 0, poilu = 0))
> res$net.result
             [,1]
[1,] 0.1326223832
> res <- compute(nn, data.frame(longueur = 0.1, mignon = 1, poilu = 1))
> res$net.result
            [,1]
[1,] 0.940063666
```



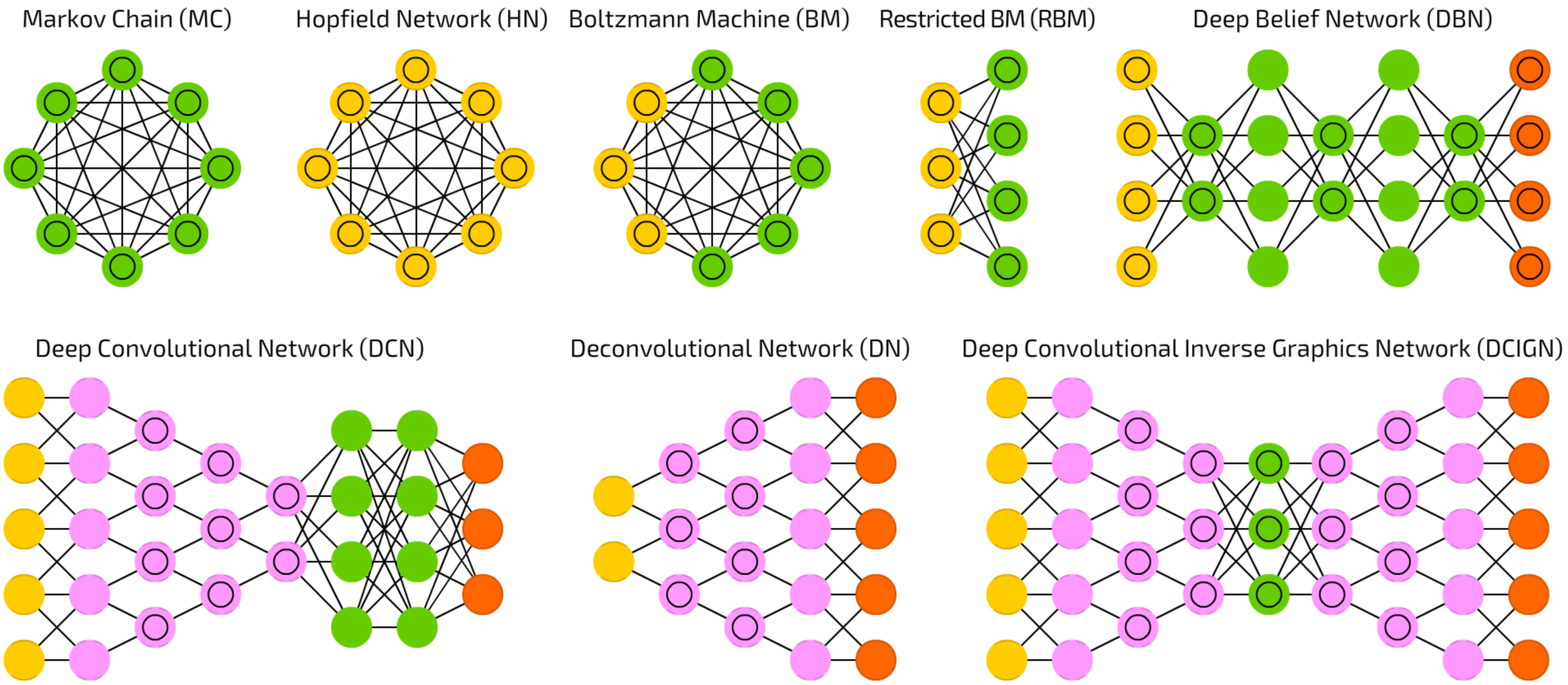
Nous ne sommes plus en 1957



https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

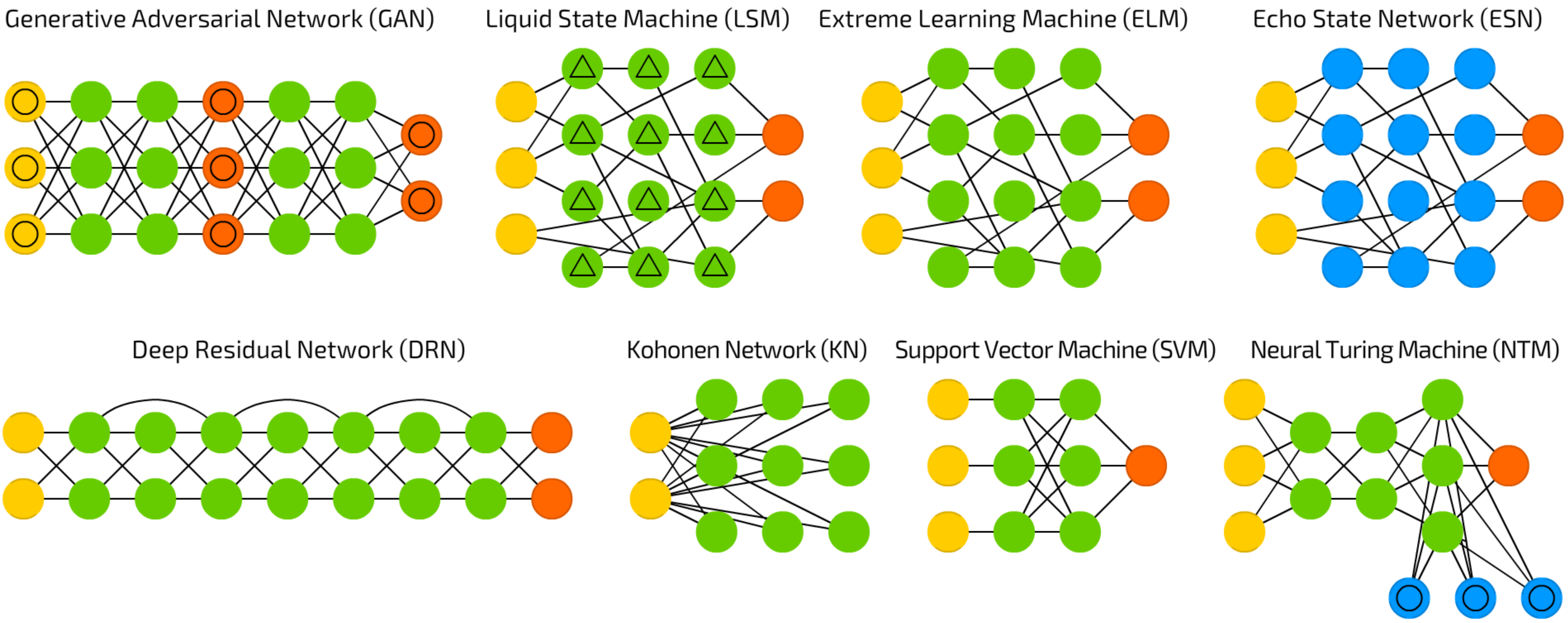
Valeuriad

Nous ne sommes plus en 1957



Valeuriad

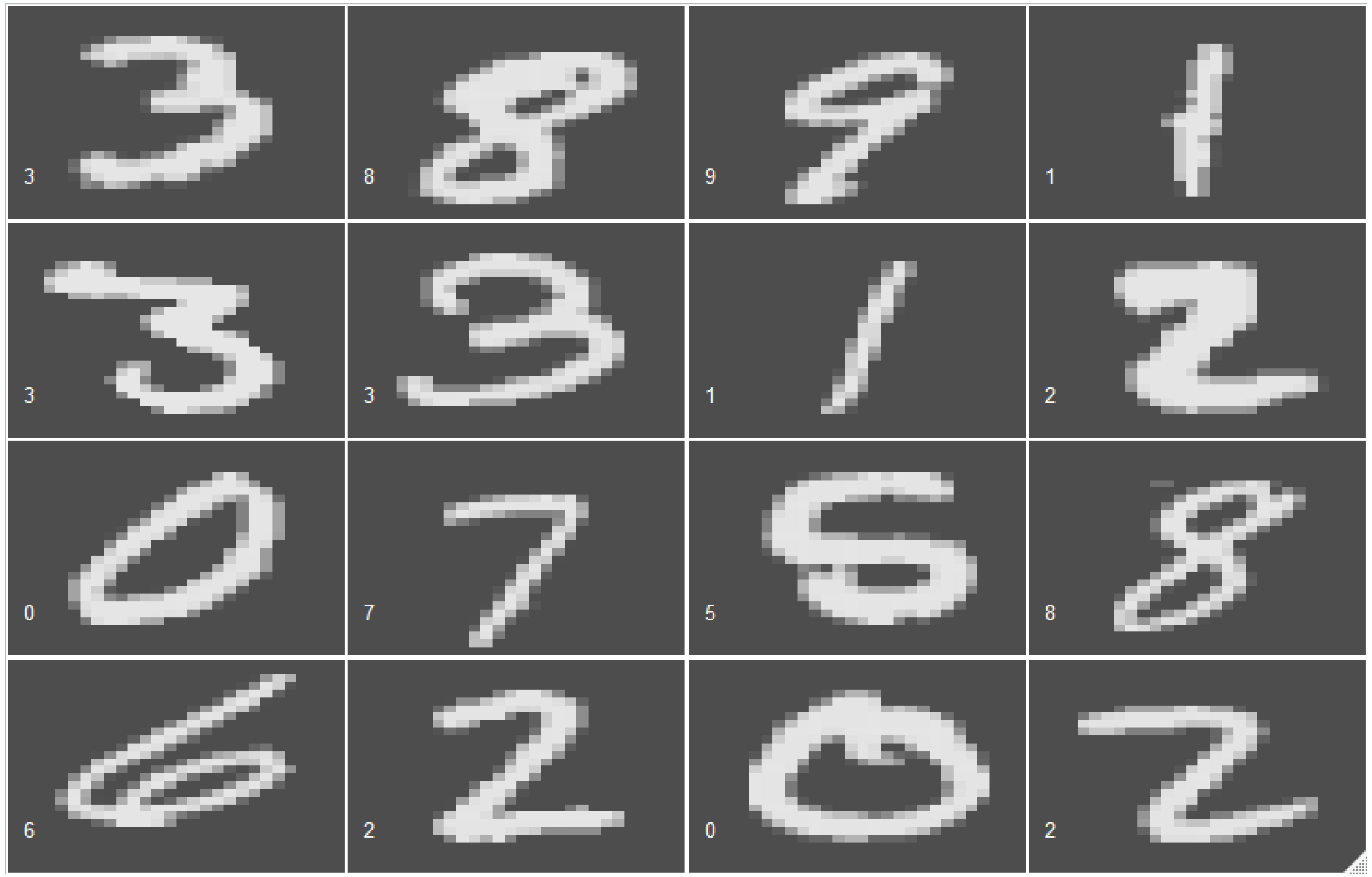
Nous ne sommes plus en 1957



Les modèles denses classiques

(MLP avec plein de layers)

MNIST Dataset

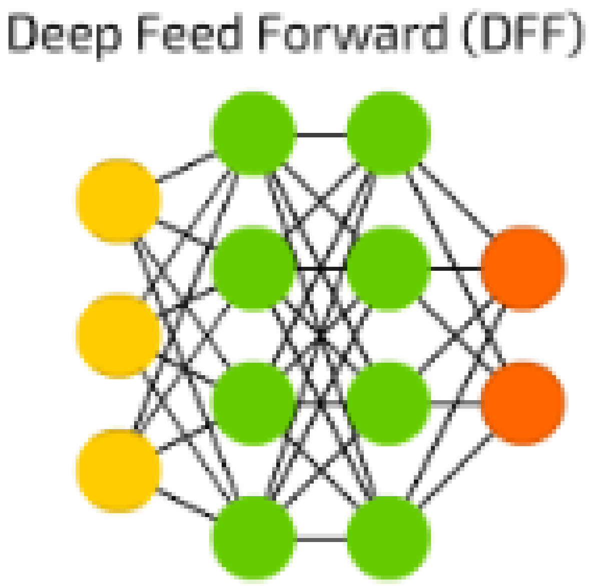


Images de 28x28
60k training set
10k test set

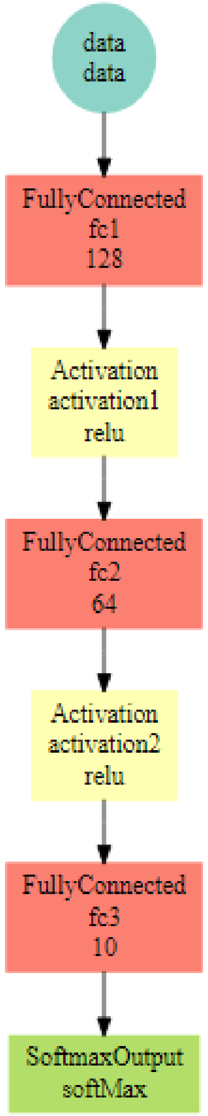
Valeuriad

- Project open-source de la fondation Apache
- Permet de déclarer / construire un réseau de neurones via une API
  - On assemble des briques
- Mode impératif ou symbolique
- Entraînement distribué
- APIs pour 8 langages (Python, Scala, Julia, Clojure, Java, C++, R, Perl)

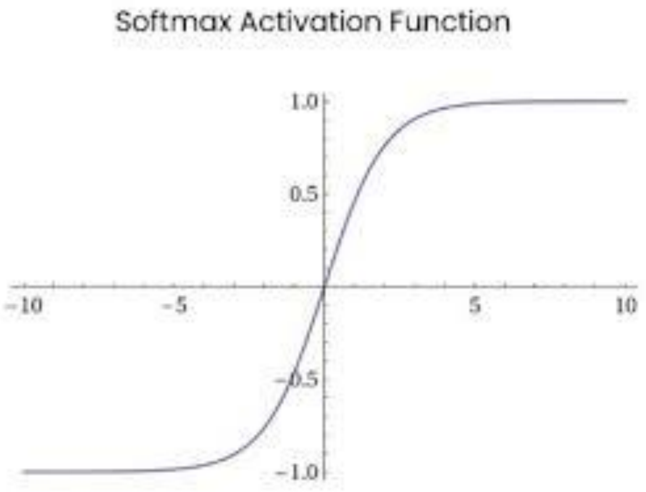
Dense



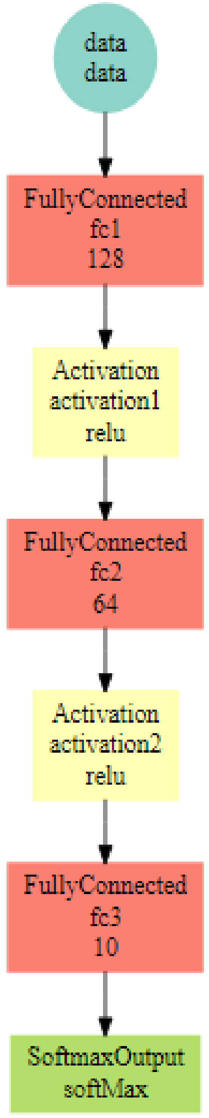
Dense


data
data
FullyConnected
fc1
128
Activation
activation1
relu
FullyConnected
fc2
64
Activation
activation2
relu
FullyConnected
fc3
10
SoftmaxOutput
softMax

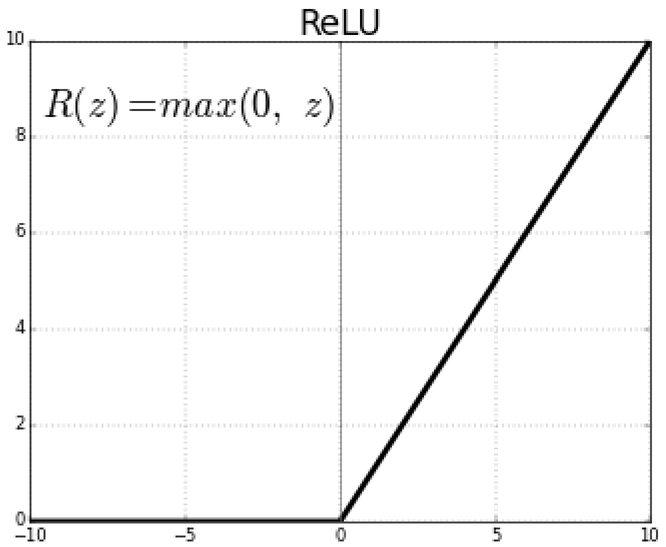
Dense



Softmax Activation Function
1.0
0.5
-0.5
-1.0
-10
-5
5
10



data
data
FullyConnected
fc1
128
Activation
activation1
relu
FullyConnected
fc2
64
Activation
activation2
relu
FullyConnected
fc3
10
SoftmaxOutput
softMax



ReLU
$R(z) = max(0, z)$
10
8
6
4
2
0
−10
−5
0
5
10

R

```r
library(mxnet)

m1.data <- mx.symbol.Variable("data") # Input Layer

m1.fc1 <- mx.symbol.FullyConnected(m1.data, name="fc1", num_hidden=128) # Dense Layer (qui prend m1.data en input)
m1.act1 <- mx.symbol.Activation(m1.fc1, name="activation1", act_type="relu") # Activation : relu

m1.fc2 <- mx.symbol.FullyConnected(m1.act1, name="fc2", num_hidden=64) # Dense Layer (qui prend m1.data en input)
m1.act2 <- mx.symbol.Activation(m1.fc2, name="activation2", act_type="relu") # Activation : relu

m1.fc3 <- mx.symbol.FullyConnected(m1.act2, name="fc3", num_hidden=10) # Dense Layer (qui prend m1.data en input)
m1.softmax <- mx.symbol.SoftmaxOutput(m1.fc3, name="softMax") # Output layer : softmax

m1 <- mx.model.FeedForward.create(m1.softmax,  #Le réseau
                                  X = train.x, #Les inputs
                                  y = train.y, #Les outputs
                                  #ctx = mx.cpu(), #CPU ou GPU
                                  num.round = 10, #Nombre d'epochs
                                  array.batch.size = 100, #Nombre de lignes par batch
                                  array.layout="colmajor", #Le sens dans lequel lire les données
                                  learning.rate = 0.001, #Learning rate
                                  eval.metric = mx.metric.accuracy, # La métrique à afficher
                                  initializer = mx.init.uniform(0.07), # Comment initialiser les poids
                                  epoch.end.callback = mx.callback.log.train.metric(1,log) # Fonction à appeler après
                                                                                           # chaque epoch
)

m1.preds <- predict(m1, test.x, array.layout = "colmajor")

```


Valeuriad

Python (API Gluon)

```python
import mxnet as mx
from mxnet import gluon
from mxnet.gluon import nn
from mxnet import autograd as ag

# define network
net = nn.Sequential()
with net.name_scope():
    net.add(nn.Dense(128, activation='relu'))
    net.add(nn.Dense(64, activation='relu'))
    net.add(nn.Dense(10))

gpus = mx.test_utils.list_gpus()
ctx = [mx.gpu()] if gpus else [mx.cpu(0), mx.cpu(1)]
net.initialize(mx.init.Xavier(magnitude=2.24), ctx=ctx)
trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate': 0.02})

epoch = 10
metric = mx.metric.Accuracy()
softmax_cross_entropy_loss = gluon.loss.SoftmaxCrossEntropyLoss()
for i in range(epoch):
    for batch in train_data:
        data = gluon.utils.split_and_load(batch.data[0], ctx_list=ctx, batch_axis=0)
        label = gluon.utils.split_and_load(batch.label[0], ctx_list=ctx, batch_axis=0)
        outputs = []
        with ag.record():
            for x, y in zip(data, label):
                z = net(x)
                loss = softmax_cross_entropy_loss(z, y)
                loss.backward()
                outputs.append(z)
        metric.update(label, outputs)
        trainer.step(batch.data[0].shape[0])
    name, acc = metric.get()
    metric.reset()
    print('training acc at epoch %d: %s=%f'%(i, name, acc))

preds = net(test_data)
```

Valeuriad

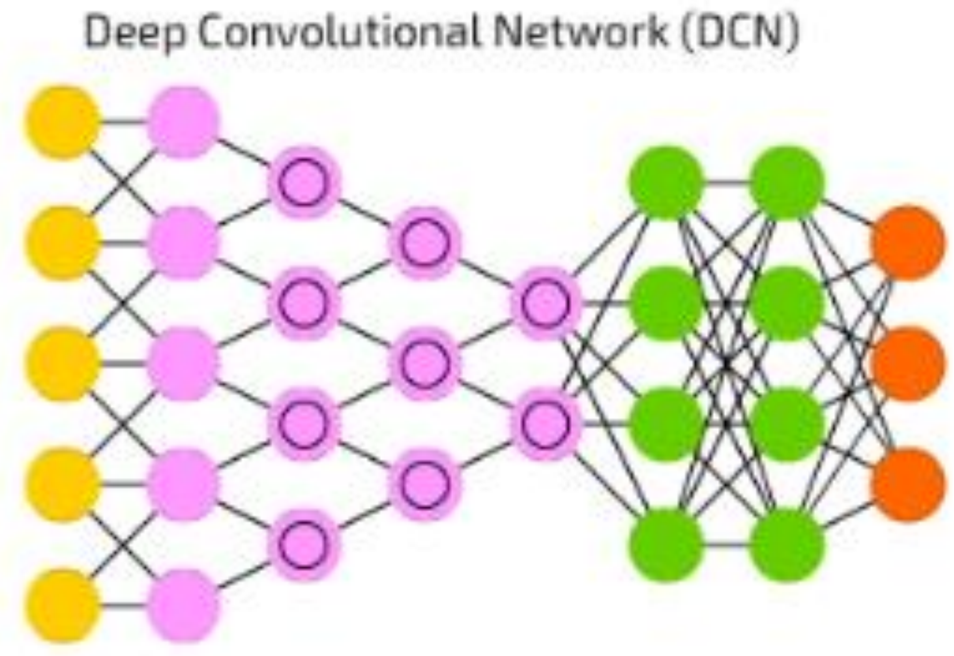
Résultats

- R:
  - MNIST Dense : 92% training accuracy; 11 sec trainng
- Python :
  - MNIST Dense : 95% training accuracy; 36 sec training

Les réseaux à convolution

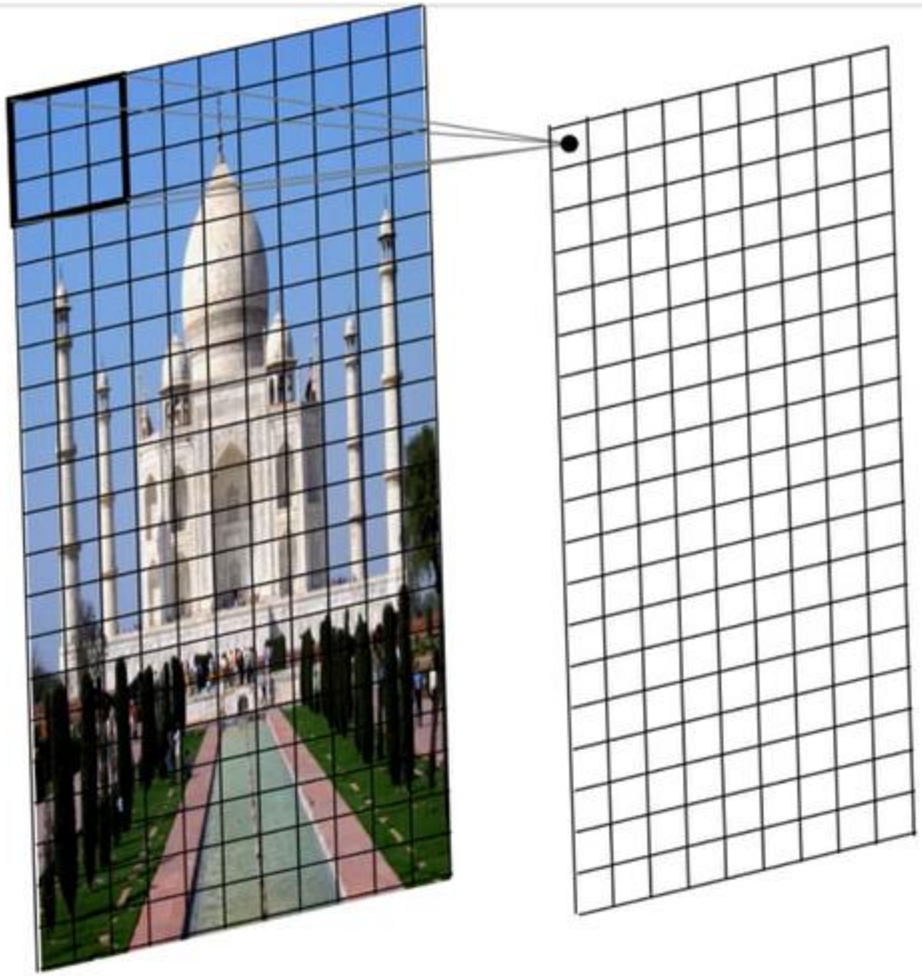
(idéal pour les images)

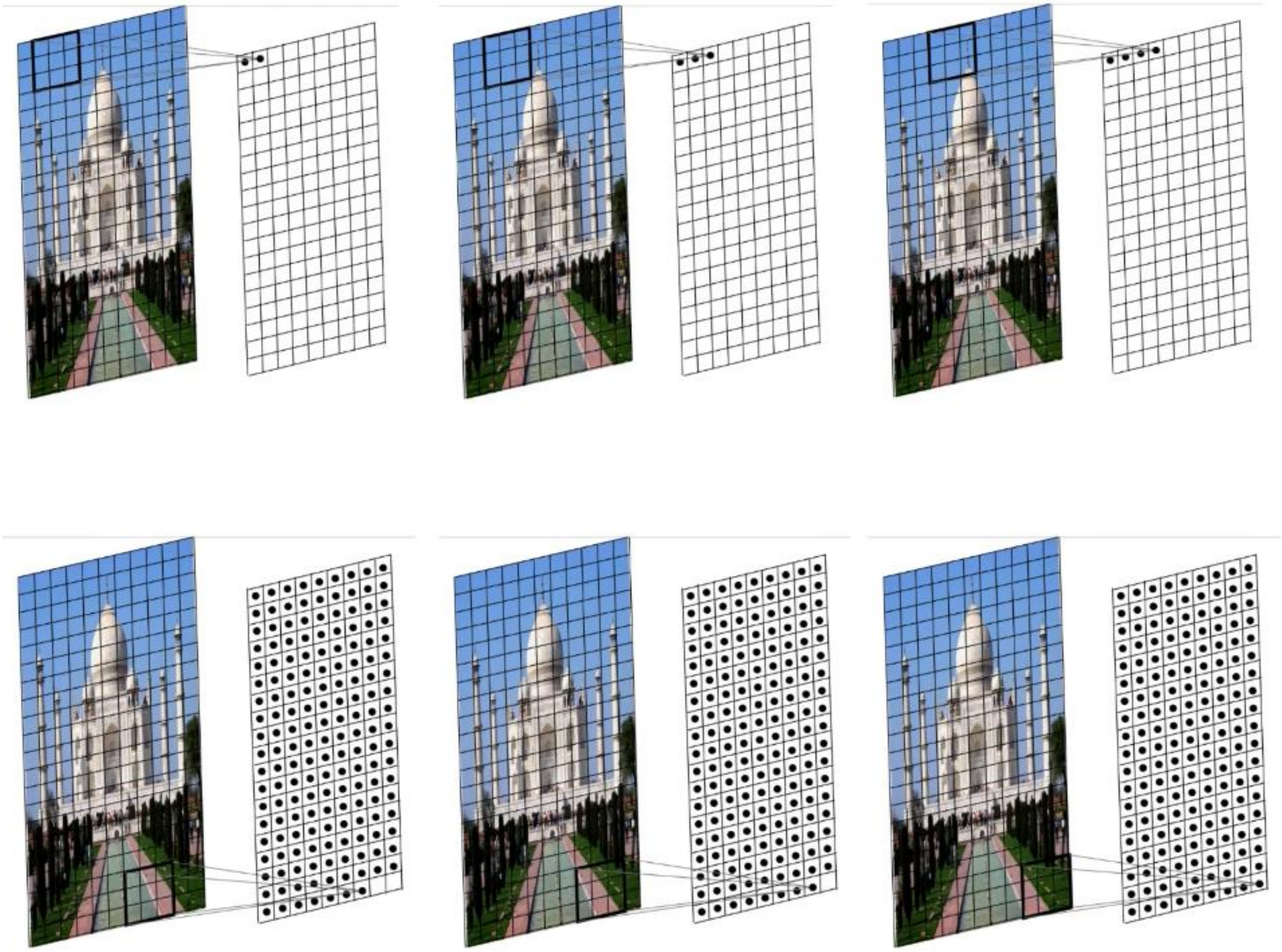
CNNs



Valeuriad

CNNs





$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

blurs the image

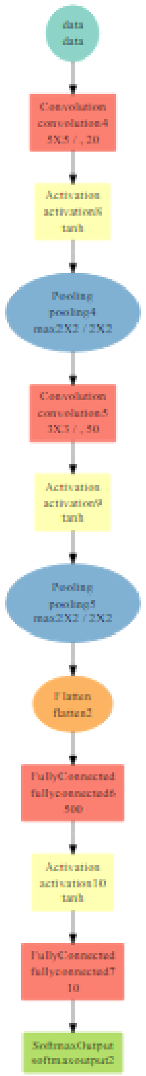
Valeuriad

Input image

Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map


data
data
Convolution
convolution4
5X5 / , 20
Activation
activation8
tanh
Pooling
pooling4
max2X2 / 2X2
Convolution
convolution5
3X3 / , 50
Activation
activation9
tanh
Pooling
pooling5
max2X2 / 2X2
Flatten
flatten2
FullyConnected
fullyconnected6
500
Activation
activation10
tanh
FullyConnected
fullyconnected7
10
SoftmaxOutput
softmaxoutput2

R

```r
m2.data <- mx.symbol.Variable("data") #Input layer
m2.conv1 <- mx.symbol.Convolution(m2.data, kernel=c(5,5), num_filter=20) # Convolution layer 1
m2.act1 <- mx.symbol.Activation(m2.conv1, act_type="tanh") # Activation Tanh
m2.pool1 <- mx.symbol.Pooling(m2.act1, pool_type="max", kernel=c(2,2), stride=c(2,2)) # Pooling Layer 1

m2.conv2 <- mx.symbol.Convolution(m2.pool1, kernel=c(3,3), num_filter=50) # Convolution layer 2
m2.act2 <- mx.symbol.Activation(m2.conv2, act_type="tanh") # Activation Tanh
m2.pool2 <- mx.symbol.Pooling(m2.act2, pool_type="max", kernel=c(2,2), stride=c(2,2)) # Pooling Layer 2
m2.flatten <- mx.symbol.Flatten(m2.pool2) # Reshape de l'output en sortie de m2.pool2

m2.fc1 <- mx.symbol.FullyConnected(m2.flatten, num_hidden=500) # Dense Layer 1
m2.act3 <- mx.symbol.Activation(m2.fc1, act_type="tanh") # Activation Tanh

m2.fc4 <- mx.symbol.FullyConnected(m2.act3, num_hidden=10) # Dense layer 2
m2.softmax <- mx.symbol.SoftmaxOutput(m2.fc4) # Output Layer : softmax

m2 <- mx.model.FeedForward.create(m2.softmax,
                                  X = train.array,
                                  y = train.y,
                                  num.round = 10,
                                  array.batch.size = 500,
                                  array.layout="colmajor",
                                  learning.rate = 0.01,
                                  eval.metric = mx.metric.accuracy,
                                  initializer = mx.init.uniform(0.07),
                                  epoch.end.callback = mx.callback.log.train.metric(1, log)
)

m2.preds <- predict(m2, test.array)
```

Python (API Gluon)

```
class Net(gluon.Block):
    def __init__(self, **kwargs):
        super(Net, self).__init__(**kwargs)
        with self.name_scope():
            self.conv1 = nn.Conv2D(20, kernel_size=(5,5))
            self.pool1 = nn.MaxPool2D(pool_size=(2,2), strides = (2,2))
            self.conv2 = nn.Conv2D(50, kernel_size=(5,5))
            self.pool2 = nn.MaxPool2D(pool_size=(2,2), strides = (2,2))
            self.fc1 = nn.Dense(500)
            self.fc2 = nn.Dense(10)

    def forward(self, x):
        x = self.pool1(F.tanh(self.conv1(x)))
        x = self.pool2(F.tanh(self.conv2(x)))
        x = x.reshape((0, -1))
        x = F.tanh(self.fc1(x))
        x = F.tanh(self.fc2(x))
        return x

net = Net()
```

MXNET

- R:
  - MNIST CNN : 97% training accuracy; 434 sec
- Python :
  - MNIST CNN : 97% training accuracy; 734 sec

Les réseaux récurrents

(Pour les données séquentielles comme le texte)

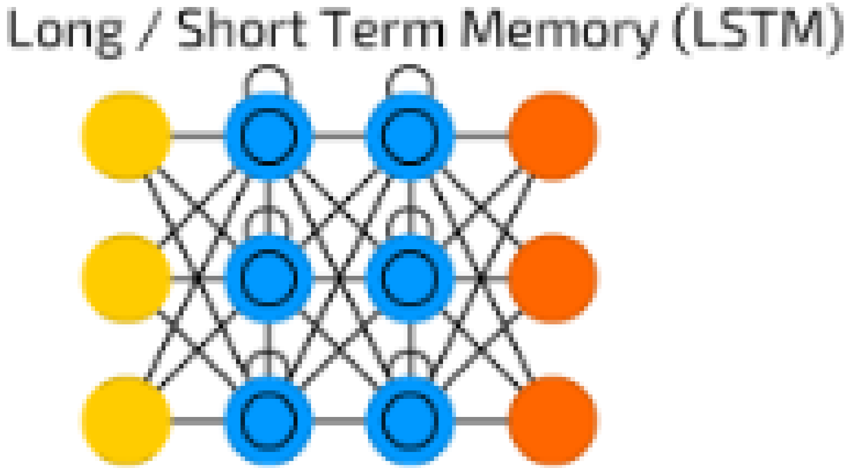
IMDB Sentiment Dataset

The best way for me to describe Europa, which is high on the list of my favourite films, is the exclamation that came from a companion after the film ended: "I didn't know films could be made like that". Entirely original in it's visual style, it is one of the best examples of what cinema can be. [.....]the elements are on an equal footing, where ONLY the BLENDING of those elements, in the order or combination in which they are presented, will communicate the idea. Reduce or eliminate the contribution of one element, and the film has no meaning. "Europa" is what cinema should strive to be.

Positive = 1

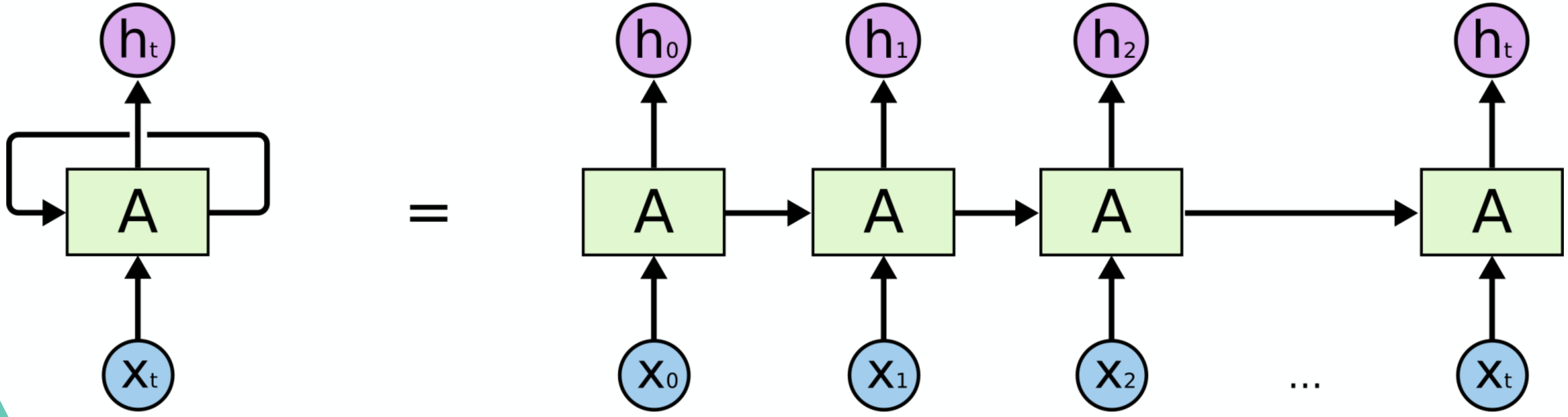
25k training set
25k test set

LSTMs



Valeuriad

LSTMs



Keras

- API haut niveau
- Compatible avec plusieurs backends (TensorFlow, CNTK, Theano)
- Développé en Python
- Faite par un français (François Chollet) : QOQORIQO
- Support R (par l'équipe de Rstudio, Hadley Wickham…)
- Désormais intégrée nativement à TensorFlow depuis TF2.0

R

```r
library(keras)

max_features <- 20000 # Nombre de mots dans le vocabulaire du réseau
batch_size <- 32 # Nombre d'entrées par batch
maxlen <- 80  # Taille max des données en input

imdb <- dataset_imdb(num_words = max_features) # Chargement des données 1 mot = 1 index
x_train <- imdb$train$x
y_train <- imdb$train$y
x_test <- imdb$test$x
y_test <- imdb$test$y

x_train <- pad_sequences(x_train, maxlen = maxlen) # ON limite toutes les séquences à maxlen
x_test <- pad_sequences(x_test, maxlen = maxlen)

model <- keras_model_sequential() # On crée un modèle séquentiel
model %>%
  layer_embedding(input_dim = max_features, output_dim = 128) %>% # Embedding Layer
  layer_lstm(units = 64, dropout = 0.2, recurrent_dropout = 0.2) %>% #LSTM Layer
  layer_dense(units = 1, activation = 'sigmoid') # Output Layer

model %>% compile(  # "Compilation du modèle", ie : transformation en graphe tensorflow
  loss = 'binary_crossentropy',
  optimizer = 'adam',
  metrics = c('accuracy')
)

model %>% fit(  # Entraînement du modèle
  x_train, y_train,
  batch_size = batch_size,
  epochs = 5,
  validation_data = list(x_test, y_test)
)

scores <- model %>% evaluate(
  x_test, y_test,
  batch_size = batch_size
)
```

Python

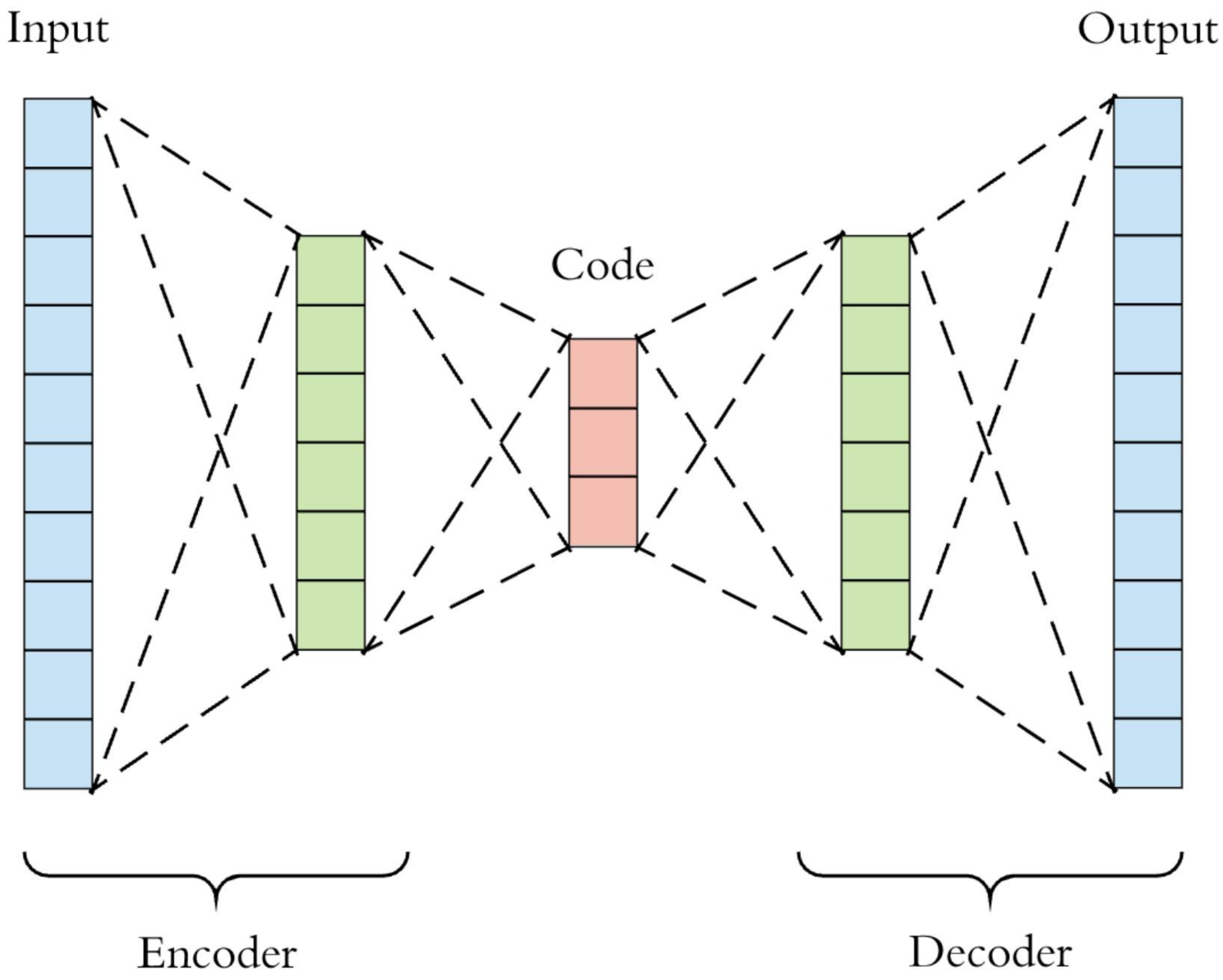
```
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM
from keras.datasets import imdb
import time
max_features = 20000
maxlen = 80
batch_size = 32
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

model = Sequential()
model.add(Embedding(max_features, 128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

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

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=5,
          validation_data=(x_test, y_test))
score, acc = model.evaluate(x_test, y_test,
                            batch_size=batch_size)
```

Résultats

- R
  - 555sec, test accuracy 81%
- Python
  - 570sec, test accuracy 81%

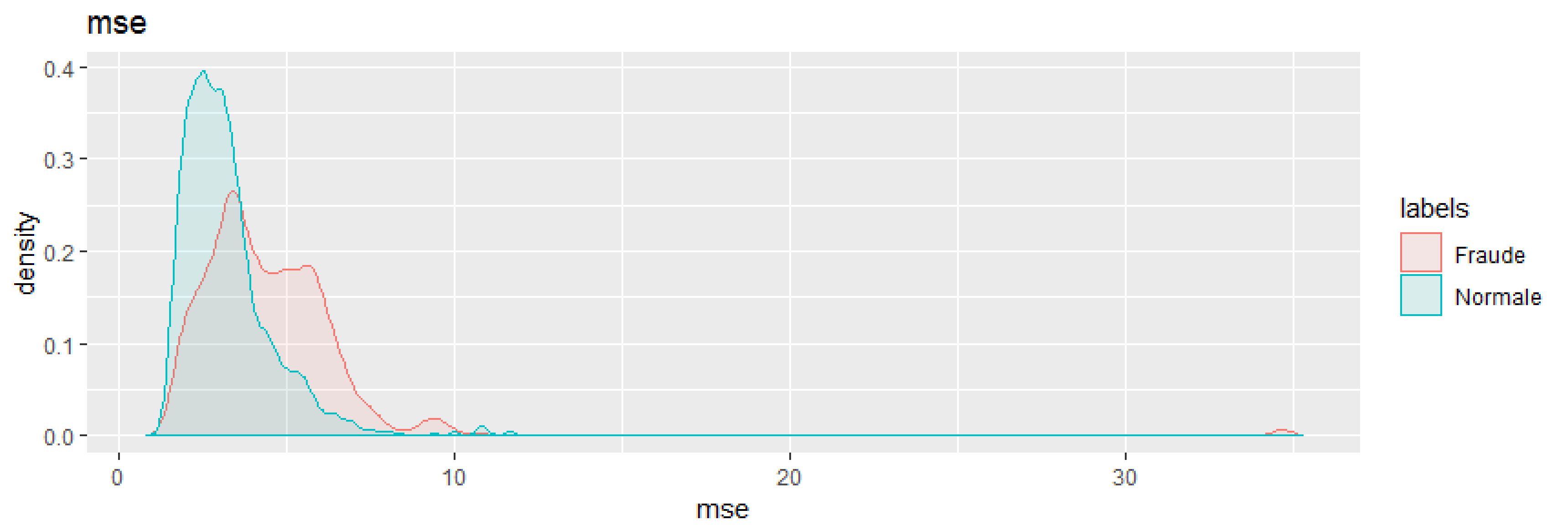
Les auto-encodeurs

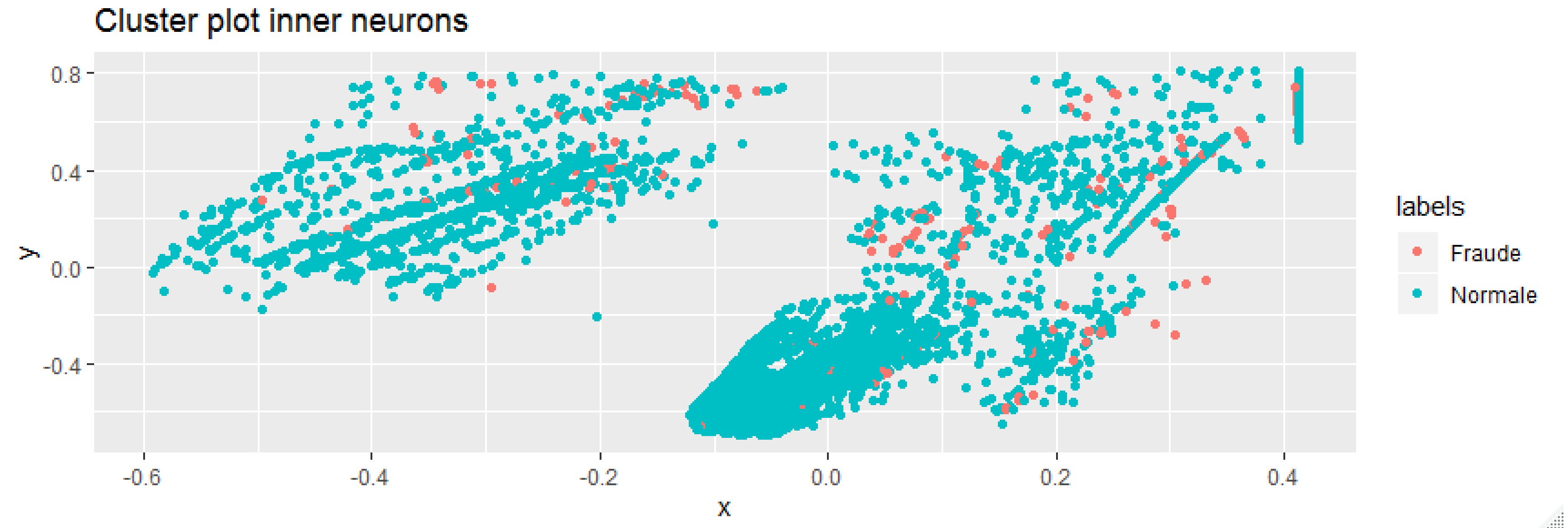
(idéal pour la détection d'anomalies)


Input
Output
Code
Encoder
Decoder

- Plateforme de machine learning in-memory
- Idéal sur les clusters Hadoop / Spark
- Facilite l’ETL autour du ML

```r

library("h2o")

h2o.init()                                      #initiation du "cluster" h2o
frame = as.h2o(d_training)                      #transformation de notre data.frame d'entraînement en frame h2o
model=h2o.deeplearning(1:21,                    #on lance l'apprentissage sur les colonnes de 1 à 21
                       training_frame=frame,    #sur notre frame
                       hidden=c(64,8,2,8,64),   #avec une typologie ayant cette structure
                       autoencoder = T,         #notre réseau essaye de reconstituer ses entrées
                       activation="Tanh")       #les neurones sont activés avec la fonction Tanh
frame = as.h2o(d_test)                          #transformation de notre data.frame de test en frame h2o
features=h2o.deepfeatures(model,                #on peut récupérer les "features" du 3e layer, ie le plus fin
                          frame,
                          layer=3)

preds = h2o.predict(model, frame)               #prédiction (ie : reconstruction)
preds = as.data.frame(preds)                    #transformation des prédictions en data.frame

mse = sqrt(rowSums((scale(preds)-scale(d_test))^2)) #calcul de la MSE: l'erreur de reconstruction

labels[order(mse,decreasing = T)][1:20]             #Affichage des labels du top 20 MSE

[1] "Normale" "Normale" "Normale" "Fraude"  "Fraude"  "Fraude"  "Fraude"  "Fraude"  "Fraude"  "Fraude"  "Fraude"
[12] "Fraude"  "Fraude"  "Fraude"  "Fraude"  "Fraude"  "Fraude"  "Fraude"  "Fraude"  "Fraude"
```

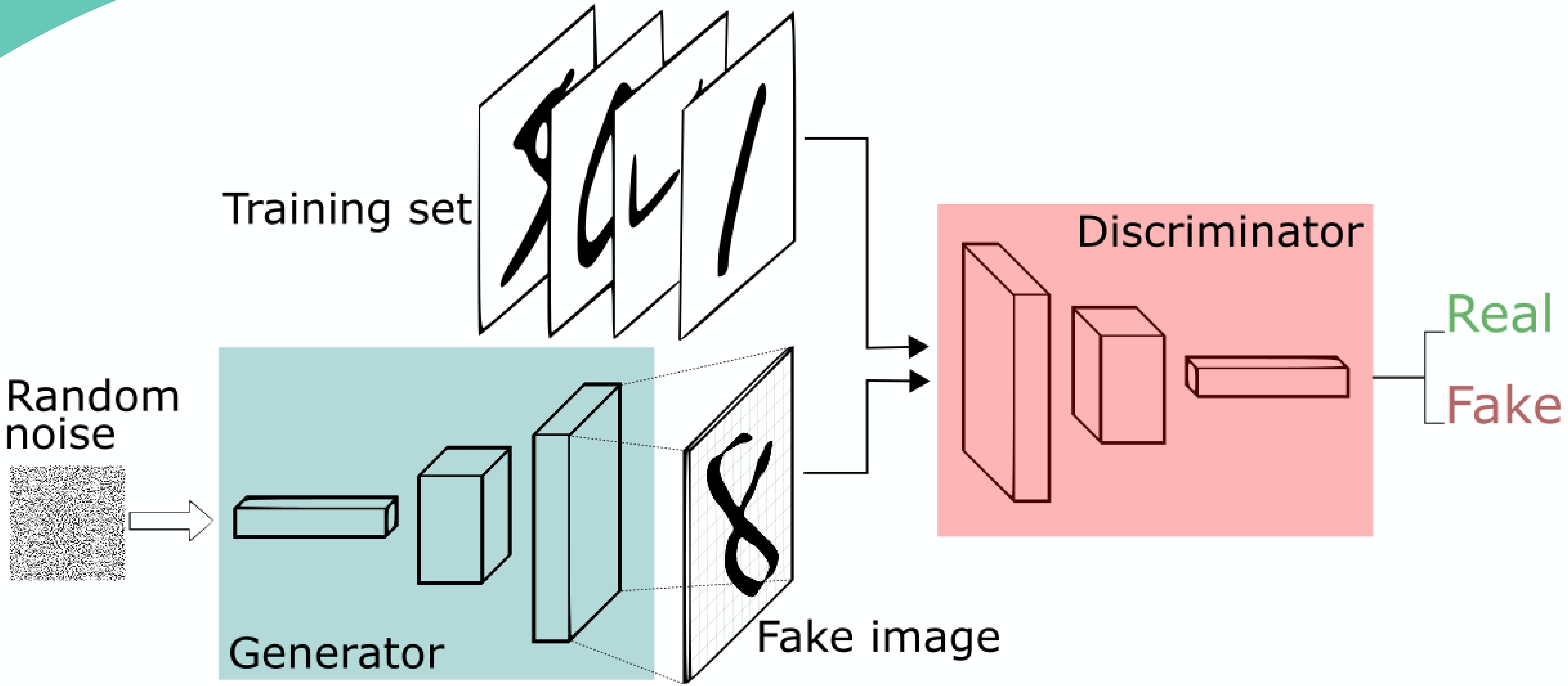

mse
0.4
0.3
0.2
0.1
0.0
density
0
10
20
30
mse
labels
Fraude
Normale


Cluster plot inner neurons
0.8
0.4
0.0
-0.4
y
-0.6
-0.4
-0.2
0.0
0.2
0.4
x
labels
Fraude
Normale

Valeuriad

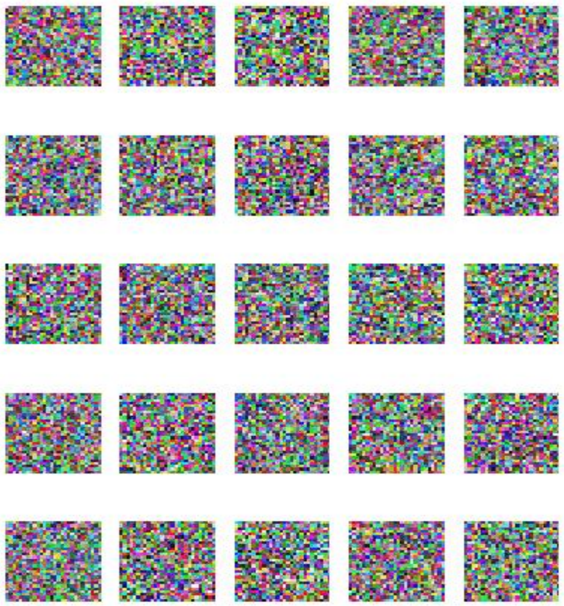
Les GANs

(Pour générer des choses)


Training set
Random noise
Generator
Fake image
Discriminator
Real
Fake

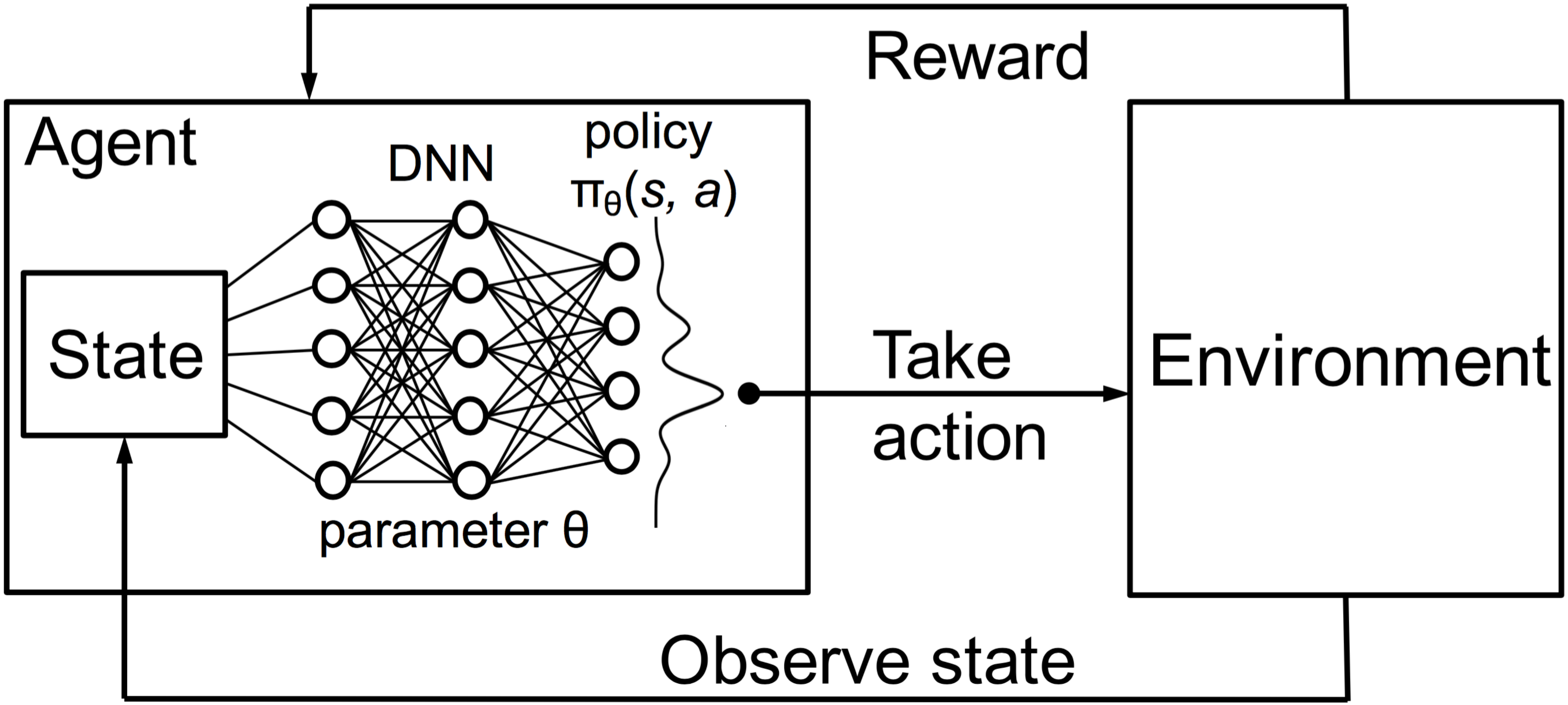
```r
build_generator <- function(latent_size, channels = 1){

  cnn <- keras_model_sequential()

  cnn %>%
    layer_dense(1024, input_shape = latent_size, activation = "tanh") %>%
    layer_dense(128*7*7, activation = "tanh") %>%
    layer_batch_normalization() %>%
    layer_reshape(c(128, 7, 7)) %>%
    layer_upsampling_2d(size = c(2, 2)) %>%
    layer_conv_2d(
      64, c(5,5), padding = "same", #activation = "tanh",
      kernel_initializer = "glorot_normal"
    ) %>%
    layer_activation_leaky_relu(alpha = 0.01) %>%
    layer_upsampling_2d(size = c(2, 2)) %>%
     layer_conv_2d(
       128, c(5,5), padding = "same", #activation = "tanh",
       kernel_initializer = "glorot_normal"
     ) %>%
    layer_activation_leaky_relu(alpha = 0.01) %>%
    layer_conv_2d(
      channels, c(5,5), padding = "same", #activation = "tanh",
      kernel_initializer = "glorot_normal"
    )
  latent <- layer_input(shape = list(latent_size))
  fake_image <- cnn(latent)

  keras_model(latent, fake_image)
}
```

```r
build_discriminator <- function(channels = 1){

  cnn <- keras_model_sequential()

  cnn %>%
    layer_conv_2d(
      64, c(5,5),
      padding = "same",
      strides = c(2,2),
      input_shape = c(channels, 28, 28)#, activation = "tanh"
    ) %>%
    layer_activation_leaky_relu(alpha = 0.01) %>%
    #layer_dropout(0.3) %>%
    layer_max_pooling_2d(pool_size = c(2,2)) %>%
    layer_conv_2d(
      128, c(5, 5),
      padding = "same",
      #activation = "tanh"
      strides = c(1,1)
      ) %>%
    layer_activation_leaky_relu(alpha = 0.01) %>%
    #layer_dropout(0.3) %>%
    layer_max_pooling_2d(pool_size = c(2,2)) %>%
    layer_flatten()

  image <- layer_input(shape = c(channels, 28, 28))
  features <- cnn(image)

  fake <- features %>%
    layer_dense(1024, activation = "tanh") %>%
    layer_dense(1, activation = "sigmoid", name = "generation")

  keras_model(image, fake)
}
```

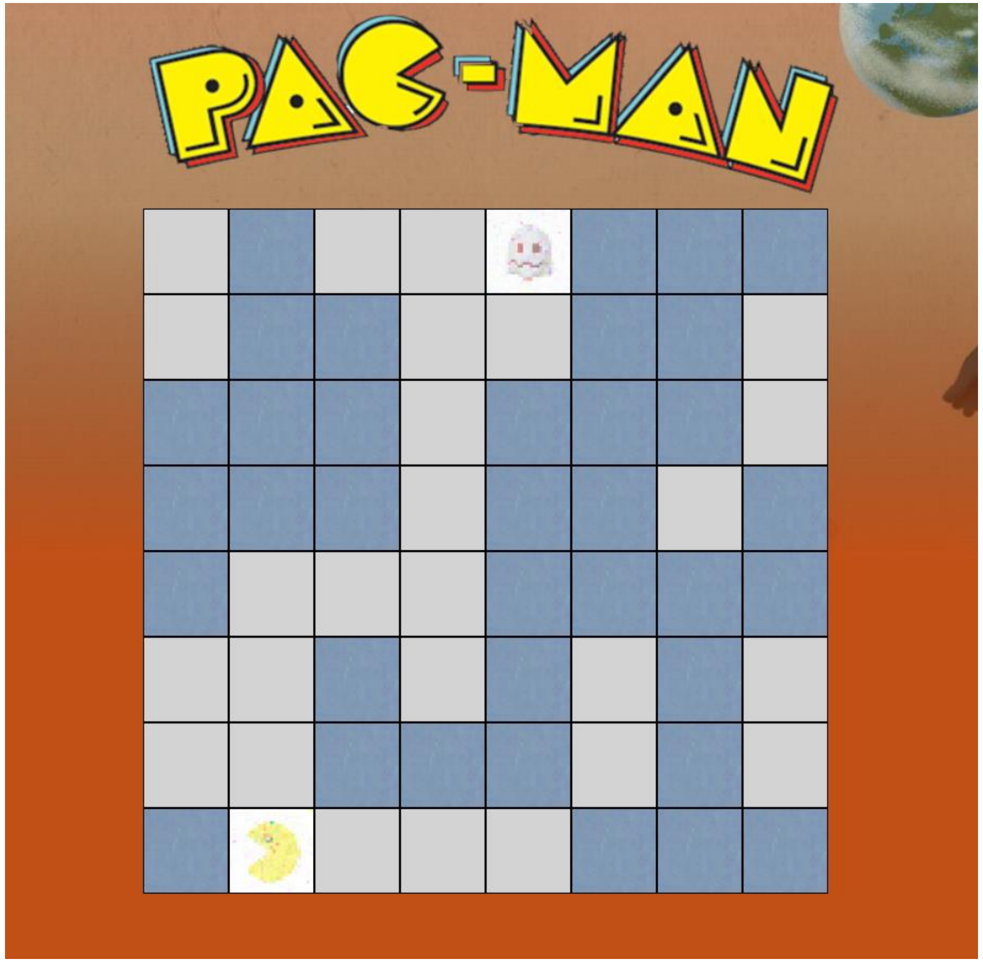


Le Renforcement

(idéal pour apprendre de ses erreurs)


Reward
Agent
DNN
policy
$\pi_\theta(s, a)$
State
parameter θ
Take
action
Environment
Observe state

- Développé directement avec l’API Tensorflow



https://github.com/Valeuriad/devfest2018/blob/master/game_logic/src/Agent.R

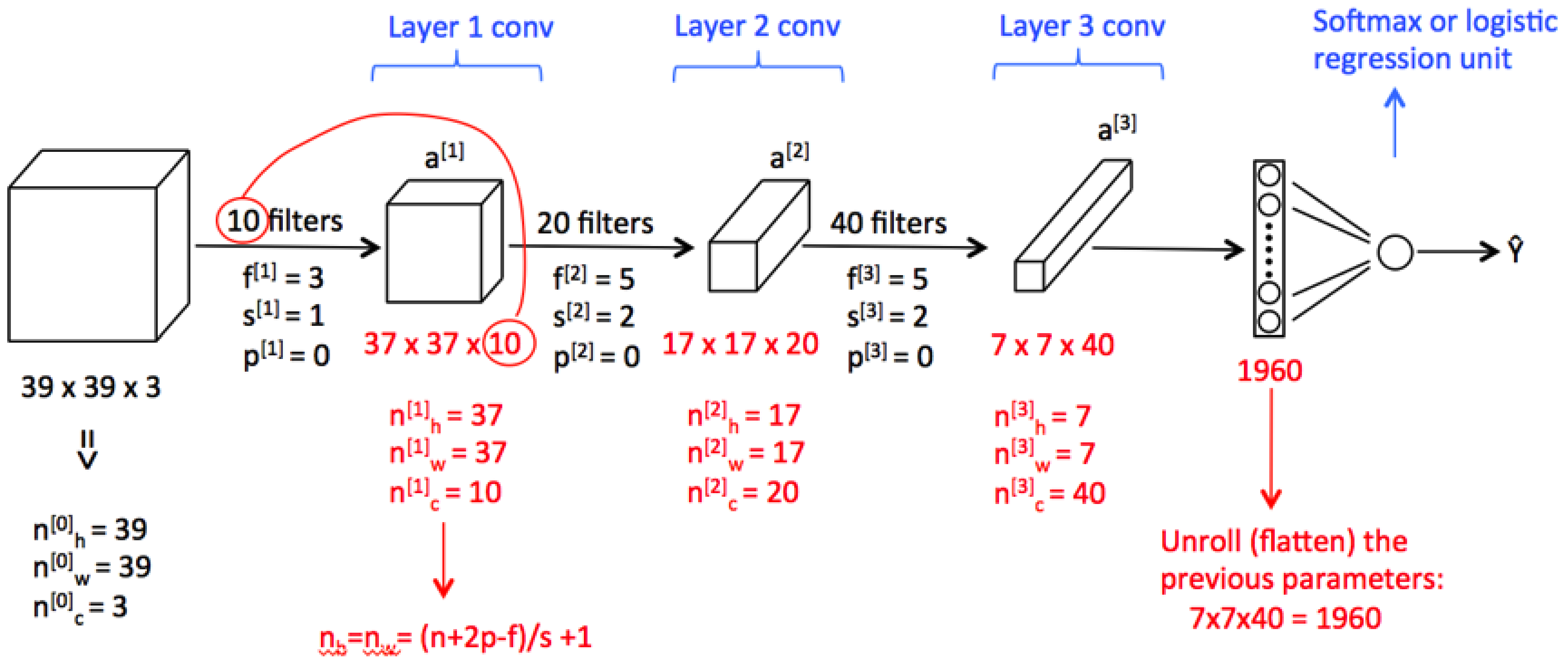
Conclusion

Conclusion

- R Permet d'utiliser la majorité des frameworks de Deep Learning actuels
  - Mis à part pytorch
  - Accès à google cloudml
- Les performances sont équivalentes
  - Peut-être même en faveur du R
- N'attendez plus, faites du Deep LeaRning


MERCI
Des questions ?
Valeuriad
www.valeuriad.fr

Annexe



https://www.coursera.org/lecture/convolutional-neural-networks/convolutions-over-volume-ctQZz