

RÉSEAUX DE NEURONES ARTIFICIELS

David Dethan K. MINLOUBOU

12/11/2021



**SORBONNE
UNIVERSITÉ**



Contents

Présentation des Réseaux de neurones	5
Introduction	5
Structure	5
Fonctionnement	6
Apprentissage	6
Qu'est ce que l'apprentissage	6
Les règles d'apprentissage	6
Perceptron	6
Structure	6
Apprentissage supervisé	7
Application : Utilisation de R pour le perceptron multicouches(PMC) et comparaison avec d'autres méthodes statistiques	7
Chargement des images et Transformation	7
Création dataset Image Pokemon	9
Création données d'entraînement et données test	9
Création du réseau neuronal PMC	9
Prediction	10
Comparaison PMC et Random Forest	11
Comparaison PMC et Machine à vecteur supports	13
Démonstration d'un autre logiciel traitant des réseaux neuronaux(Python)	14
Chargement des librairies Python nécessaires	14
Chargement des données images	14
Encodage des labels	14
Split des données en données d'entraînement et données test	15
Perceptron Multi-couche	15
Carte de Kohonen	16
Présentation	16
Algorithme	18
Application: Carte de Kohonen sur nos données Pokemon	19
Vecteurs référents	21
Plot des distances	22
Count Plot	23
Prediction avec les cartes de Kohonen	27

Apprentissage par renforcement	28
Présentation	28
Q-Learning	28
Politique	30
Application	31
Réseaux de Convolution	73
Création données entraînement, de test et de validation	73
Mise en place du réseau de convolution	74
Graphe du modèle— Entraînement et validation	75
Historique d’entraînement et de validation	76
Création de données test	76
Evaluation sur données Test	77
Décodage de la prédiction	77
Matrice de Confusion	78
Conclusion	79
Source	80

List of Figures

1	Structure d’un neurone biologique et d’un neurone formel	5
2	Réseau neuronal à une couche caché sur Images Pokemon	10
3	Carte de Kohonen	17
4	Structure markovienne de Q-Learning	29

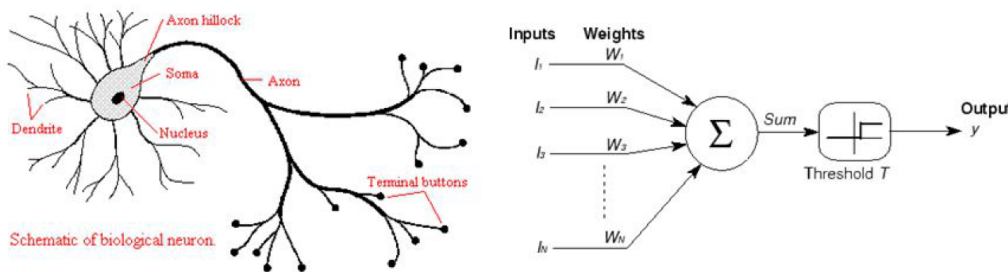
List of Tables

Présentation des Réseaux de neurones

Introduction

L'établissement des réseaux neurones artificiels se fait sur la base de l'imitation des comportements synaptiques du système cérébral humain. L'idée est donc d'en imiter la structure, et de construire des systèmes dans lesquels circuleront de l'information qui pourrait nous aider à prendre des décisions tout comme le font nos neurones.

Le neurone



Neurone biologique

Neurone artificiel

Figure 1: Structure d'un neurone biologique et d'un neurone formel

Ces systèmes que nous construisons en réseau peuvent alors nous aider à étudier des domaines complexes tels que la reconnaissance automatique de la parole, la reconnaissance des sons.

Tout comme le système nerveux, un neurone artificiel reçoit de l'information des différents neurones auxquels il est connecté. Ainsi, lors de la reconnaissance d'une image ce ne sont pas tous les neurones de notre cerveau qui sont activés. Pour ainsi imiter cette structure, il est primordial de donner des valeurs aux différents liens entre neurones pour ainsi caractériser lorsque l'information traverse entre neurones ou pas.

Structure

Pour modéliser ce qui a précédemment été dit, on définit une matrice de poids \mathbf{W} qui contiendra toutes les valeurs représentant la circulation de l'information ou pas, entre les différents neurones. On dit alors que le réseau est caractérisé par la matrice des poids \mathbf{W} , le nombre de neurones, et le type de neurones également.

Si le réseau contient des boucles, il est dit récurrent : s'il existe $i_1, i_2, \dots, i_p \in [1, N]$ tels que $i_1 = i_k$ et $w_{i_k, i_{k+1}} \neq 0$ pour $k \in [1, p-1]$.

Ce qui rentre également en vigueur dans la structure est la connectivité inter-couche. On dit que la connectivité est complète si chaque neurone d'une couche C_i est connecté à chaque neurone d'une autre couche C_j . On dit que la connectivité est bijective si et seulement si chaque neurone d'une couche C_i est connecté uniquement à un neurone d'une autre couche C_j .

La connectivité entre neurones peut également être faite de façon probabiliste. La connectivité peut également être symétrique c'est le cas par exemple dans le réseau de **Hopfield**.

Fonctionnement

Un réseau de neurones fonctionnent selon deux types d'apprentissage:

- en mode reconnaissance, le réseau est fait pour calculer.
- en mode apprentissage, le réseau est fait pour apprendre. Il apprend à partir d'exemples afin de prévoir le comportement de futures données. Cet apprentissage modifie sa matrice de poids **W**, le “**pas d'apprentissage**”, et parfois d'autres paramètres tels que les **fonctions d'activations**.

Nous allons uniquement nous consacrer au mode d'apprentissage.

Apprentissage

Qu'est ce que l'apprentissage

L'apprentissage est une phase de développement d'un réseau dans lequel la structure du réseau est modifiée dans le but d'arriver à un comportement désiré. Il existe principalement deux classes d'apprentissage :

- apprentissage supervisé : la sortie du réseau est connue, on dispose de données **labélisées**. Il s'agit de modifier la matrice des poids afin de minimiser une certaine fonction d'erreur.
- apprentissage non supervisé : la sortie du réseau n'est pas connue, il s'agit de modifier la matrice des poids afin de regrouper les données.

Les règles d'apprentissage

Il existe plusieurs règles d'apprentissage:

- La règle de **Hebb** : Il modifie les poids entre neurones de cette façon $w_{i,j} = \lambda \overline{a_i a_j}$
- La règle d'apprentissage **DELTA** : Cet algorithme est utilisé dans des réseaux appelés **perceptrons**. Cette règle modifie la matrice des poids de la façon suivante: $\Delta w_{i,j} = \lambda * a_j (d_i - s_i)$ où $s_i, d_i, w_{i,j}$ représentent respectivement la sortie du neurone i , sa sortie désirée et poids entre le neurone j (de la couche précédente) et le neurone i .

Dans les deux règles, λ est appelée **pas d'apprentissage**. Avec la règle **DELTA**, on choisit $\lambda > 0$.

Il existe évidemment d'autres règles : **Quickprop**, règle **Delta-Bar-Delta** etc...

Perceptron

Structure

Le perceptron est la structure de réseaux neuronaux la plus simple. Les couches sont disposées de façon verticale les unes après les autres. Il possède une couche d'entrée, une couche cachée et une couche de sortie.

L'activation se fait de la couche d'entrée jusqu'à la couche de sortie en passant par la couche cachée.

La fonction d'entrée est linéaire et s'écrit : $h(i) = \sum_{j=1}^n w_{i,j} a_j$ où n représente le nombre de neurones de la couche où se trouve le neurone a_i . La fonction d'activation de la couche cachée est une sigmoïde. La fonction d'activation de la couche de sortie est plutôt linéaire.

Un perceptron peut contenir plusieurs couches cachées, on parle alors de perceptron multicouches.

Apprentissage supervisé

On cherche à minimiser la fonction d'erreur $E = \frac{1}{2} \sum_{i \in C} (a_i - d_i)^2$ où C représente le corpus d'apprentissage (x_i, d_i) , a_i, d_i respectivement les sorties du réseau, et les sorties désirées.

Application : Utilisation de R pour le perceptron multicouches(PMC) et comparaison avec d'autres méthodes statistiques

Nous avons travaillé sur des images **Pokemon**. Nous avons opté pour quatre classes différentes de **Pokemon**. Chaque groupe de **Pokemon** possède **40** instances. Le but étant de classifier ces différents **Pokemon**. On est bien évidemment dans le cas d'apprentissage supervisée car nos images sont labélisées.

Chargement des images et Transformation

Comme on souhaite appliquer un perceptron multicouche à des images, il faudrait auparavant les transformer en des données numériques qui pourront être comprises par le réseau neuronal. Pour cela, on utilise la notion de pixel d'une image : une façon simple d'aborder cela est de s'imaginer une image comme étant une **matrice** que l'on divise en petits carrés homogènes (les pixels). Ces petits carrés contiennent des valeurs allant le plus souvent de 0 à 255, l'idée étant que chaque valeur de pixel fait référence à **l'intensité** de ce pixel. Cette matrice de pixel est ensuite transformée en vecteur (en juxtaposant toutes les lignes du vecteur les unes après les autres); et c'est ce vecteur qui sera envoyé dans notre réseau neuronal.

Ayant personnellement **sélectionné** chacune de ces images, j'ai noté qu'il n'avait pas tous les mêmes dimensions donc il faut donc d'abord tous les mettre à la dimension commune **28X28**, dimension choisie évidemment car elle est la plus fréquemment utilisée pour entraîner un réseau neuronal. J'ai ensuite remarqué que tous les pixels avaient des valeurs allant de 0 à 255, j'ai donc décidé de les normaliser pour qu'ils soient tous compris entre 0 et 1.

```
library(magick)
library(EBImage)
library(caret)
library(e1071)
library(neuralnet)
require(kohonen)
require(RColorBrewer)
```

```
#----- Pokemon Pikachu

setwd("/Users/david/Desktop/daviddtuto/pikachu")
images<- NULL
df=data.frame()
for (i in list.files()){
  data <- image_read(i)
  data <- image_resize(data, '28x28')
  df<-data
  data <- as.numeric(data[[1]][1, ,])
  data <- data/255
  images <- rbind(images,data)
}
```

```

label<- rep("pikachu",times=length(list.files("/Users/david/Desktop/daviddtuto1/pikachu")))
images1<- data.frame(images,label)
rownames(images1) <- 1:nrow(images1)

#----- Pokemon rondoudou

setwd("/Users/david/Desktop/daviddtuto1/rondoudou")
images2 <- NULL
for (i in list.files()){
  data <- image_read(i)
  data <- image_extent(data,"40x40")
  data <- image_resize(data,'28x28')
  data <- as.numeric(data[[1]][1, ,])
  data <- data/255
  images2 <- rbind(images2,data)
}
label1<- rep("rondoudou",times=length(list.files("/Users/david/Desktop/daviddtuto1/rondoudou")))
images3 <- data.frame(images2,label1)
rownames(images3) <- nrow(images1)+1:nrow(images3)+ nrow(images1)

#----- Pokemon Salameche

setwd("/Users/david/Desktop/daviddtuto1/Salameche")
images2 <- NULL
for (i in list.files()){
  data <- image_read(i)
  data <- image_extent(data,"40x40")
  data <- image_resize(data,'28x28')
  data <- as.numeric(data[[1]][1, ,])
  data <- data/255
  images2 <- rbind(images2,data)
}
label1<- rep("Salameche",times=length(list.files("/Users/david/Desktop/daviddtuto1/Salameche")))
images4 <- data.frame(images2,label1)
rownames(images3) <- nrow(images3)+1:nrow(images4)+ nrow(images3)

#----- Pokemon Carapuce

setwd("/Users/david/Desktop/daviddtuto1/Carapuce")
images2 <- NULL
for (i in list.files()){
  data <- image_read(i)
  data <- image_extent(data,"40x40")
  data <- image_resize(data,'28x28')
  data <- as.numeric(data[[1]][1, ,])
  data <- data/255
  images2 <- rbind(images2,data)
}
label1<- rep("Carapuce",times=length(list.files("/Users/david/Desktop/daviddtuto1/Carapuce")))
images5 <- data.frame(images2,label1)
rownames(images5) <- nrow(images4)+1:nrow(images5)+ nrow(images4)

#----- Fin chargement des images Pokemon

```


Création dataset Image Pokemon

Après chargement des images et transformations en matrice (en **pixels**), on crée un dataframe contenant toutes les coordonnées des images et leurs labels.

```
colnames(images1) <- colnames(images3)
colnames(images4) <- colnames(images3)
colnames(images5) <- colnames(images3)

final<- rbind(images1,images3,images4,images5)
dim(final)
```

```
## [1] 160 785
```

Nous avons bien 160 Pokemons ($40 * 4$) , et 785 colonnes composées de 784 pixels et d'une colonne label stipulant le type de Pokemon.

Création données d'entrainement et données test

```
split <- sample(1:nrow(final),nrow(final)*0.7)
train <- final[split,]
test <- final[-split,]
```

Création du réseau neuronal PMC

```
model<- neuralnet(label1~.,data=train, hidden=c(10,5), threshold=0.01,linear.output = FALSE)
output<- compute(model,test[, -785])
prediction <- output$net.result
```

Ayant beaucoup de variables et pas mal d'images, la fonction `plot.nn` de R peine à afficher rapidement le PMC. J'ai donc fait un autre réseau neuronal à une seule couche cachée contenant deux neurones pour avoir un aperçu du PMC sur des données Pokemon. Voici le graphe référant à ce réseau :

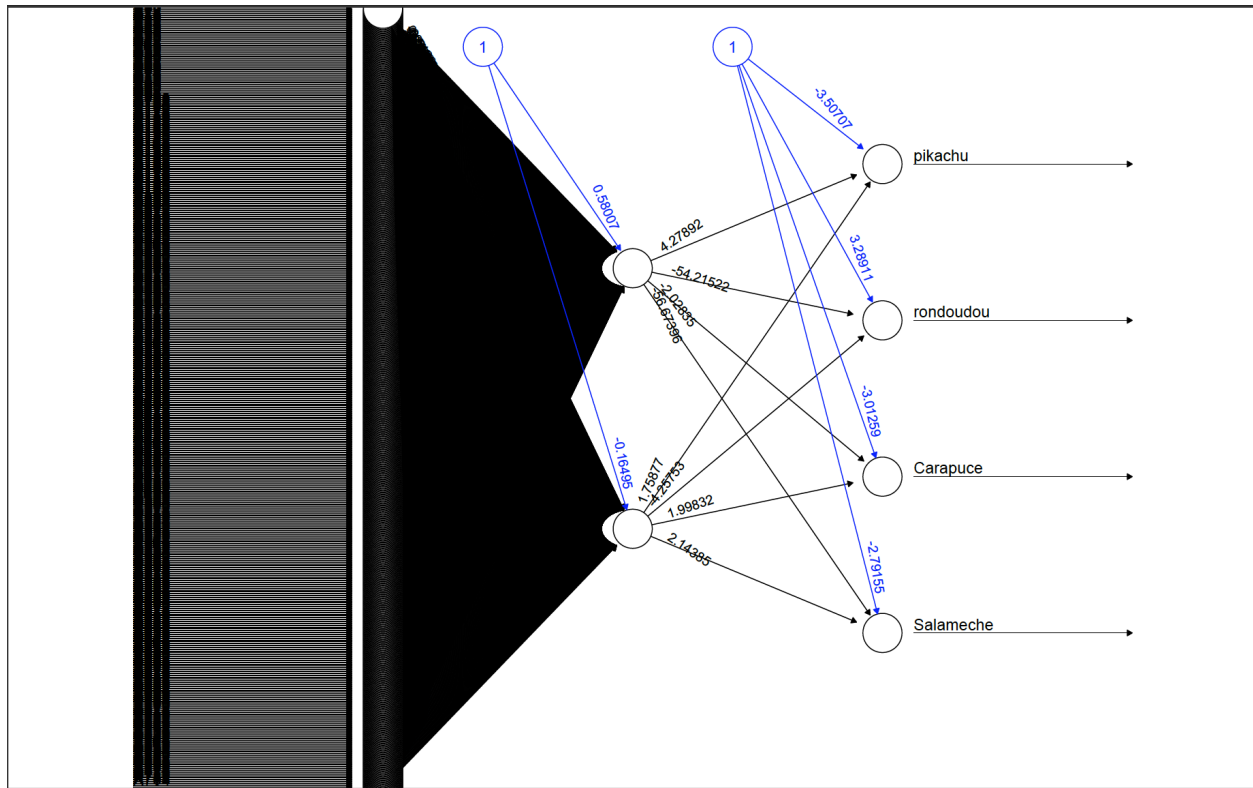


Figure 2: Réseau neuronal à une couche cachée sur Images Pokemon

Prediction

```
table(test$label1, apply(prediction, 1, which.max))
```

```
##
##      1 2 3 4
## Carapuce 6 2 4 1
## pikachu  4 4 3 3
## rondoudou 2 5 2 2
## Salameche 4 2 1 3
```

On constate que les sorties 1,2,3,4 correspondent respectivement aux Pokemons de type **Carapuce**, **Pikachu**, **rondoudou** et **Salameche** .

```
yhat=data.frame("yhat"=ifelse(max.col(prediction[,1:4])==1, "Carapuce",
                             ifelse(max.col(prediction[,1:4])==2, "pikachu",
                             ifelse(max.col(prediction[,1:4])==4, "Salameche", "rondoudou")))))

cm = confusionMatrix(as.factor(test[,785]), as.factor(yhat$yhat))
```

cm

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  Carapuce pikachu rondoudou Salameche
## Carapuce      6      2      4      1
## pikachu       4      4      3      3
## rondoudou     2      5      2      2
## Salameche     4      2      1      3
##
## Overall Statistics
##
##           Accuracy : 0.3125
##           95% CI : (0.1866, 0.4625)
##       No Information Rate : 0.3333
##       P-Value [Acc > NIR] : 0.6718
##
##           Kappa : 0.0758
##
## Mcnemar's Test P-Value : 0.6541
##
## Statistics by Class:
##
##           Class: Carapuce Class: pikachu Class: rondoudou
## Sensitivity           0.3750           0.30769           0.20000
## Specificity           0.7812           0.71429           0.76316
## Pos Pred Value        0.4615           0.28571           0.18182
## Neg Pred Value        0.7143           0.73529           0.78378
## Prevalence            0.3333           0.27083           0.20833
## Detection Rate        0.1250           0.08333           0.04167
## Detection Prevalence  0.2708           0.29167           0.22917
## Balanced Accuracy      0.5781           0.51099           0.48158
##
##           Class: Salameche
## Sensitivity           0.3333
## Specificity           0.8205
## Pos Pred Value        0.3000
## Neg Pred Value        0.8421
## Prevalence            0.1875
## Detection Rate        0.0625
## Detection Prevalence  0.2083
## Balanced Accuracy      0.5769
```

On trouve un **accuracy** assez faible en dessous de 0.5. Ce réseau neuronal n'est donc pas très efficace, essayons d'autres méthodes statistiques. On va donc comparer nos précédents résultats avec des résultats obtenues grâce aux méthodes de **Forêts aléatoires** et de **Machines à vecteurs supports**.

Comparaison PMC et Random Forest

```
library(randomForest)

train$label1 <- as.factor(train$label1)
model1 <- randomForest(label1~.,train,ntree=100,importance=TRUE,proximity=TRUE,mtry=56)
y_pred <- predict(model1,newdata=test[, -785])

confusionMatrix(as.factor(test$label),as.factor(y_pred))
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction  Carapuce pikachu rondoudou Salameche
## Carapuce      11         0          2          0
## pikachu        0        10          1          3
## rondoudou       3         0          6          2
## Salameche       2         0          3          5
##
## Overall Statistics
##
##              Accuracy : 0.6667
##              95% CI : (0.5159, 0.796)
##      No Information Rate : 0.3333
##      P-Value [Acc > NIR] : 2.425e-06
##
##              Kappa : 0.5545
##
##      McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##              Class: Carapuce Class: pikachu Class: rondoudou
## Sensitivity              0.6875              1.0000              0.5000
## Specificity              0.9375              0.8947              0.8611
## Pos Pred Value           0.8462              0.7143              0.5455
## Neg Pred Value           0.8571              1.0000              0.8378
## Prevalence               0.3333              0.2083              0.2500
## Detection Rate           0.2292              0.2083              0.1250
## Detection Prevalence     0.2708              0.2917              0.2292
## Balanced Accuracy        0.8125              0.9474              0.6806
##
##              Class: Salameche
## Sensitivity              0.5000
## Specificity              0.8684
## Pos Pred Value           0.5000
## Neg Pred Value           0.8684
## Prevalence               0.2083
## Detection Rate           0.1042
## Detection Prevalence     0.2083
## Balanced Accuracy        0.6842
```

On trouve un meilleur accuracy avec une **Forêt aléatoire** qu'avec notre précédent **perceptron multi-couche**.

Comparaison PMC et Machine à vecteur supports

```
model2 <- svm(label1~.,train,type="C-classification",fitted=TRUE,kernel="radial",cross=10)
y_pred <- predict(model2,newdata=test[,785])

confusionMatrix(as.factor(test[,785]),as.factor(y_pred))
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction  Carapuce pikachu rondoudou Salameche
##   Carapuce      10      1      0      2
##   pikachu       0      4      1      9
##   rondoudou     1      2      3      5
##   Salameche     2      0      0      8
##
## Overall Statistics
##
##              Accuracy : 0.5208
##              95% CI : (0.3719, 0.6671)
##   No Information Rate : 0.5
##   P-Value [Acc > NIR] : 0.44272
##
##              Kappa : 0.3702
##
##   McNemar's Test P-Value : 0.01207
##
## Statistics by Class:
##
##              Class: Carapuce Class: pikachu Class: rondoudou
## Sensitivity              0.7692      0.57143      0.75000
## Specificity              0.9143      0.75610      0.81818
## Pos Pred Value           0.7692      0.28571      0.27273
## Neg Pred Value           0.9143      0.91176      0.97297
## Prevalence               0.2708      0.14583      0.08333
## Detection Rate           0.2083      0.08333      0.06250
## Detection Prevalence     0.2708      0.29167      0.22917
## Balanced Accuracy        0.8418      0.66376      0.78409
##
##              Class: Salameche
## Sensitivity              0.3333
## Specificity              0.9167
## Pos Pred Value           0.8000
## Neg Pred Value           0.5789
## Prevalence               0.5000
## Detection Rate           0.1667
## Detection Prevalence     0.2083
## Balanced Accuracy        0.6250
```

Avec les **machines à vecteurs supports**, on retrouve également un meilleur modèle de classification qu'avec le **perceptron multicouche**.

Démonstration d'un autre logiciel traitant des réseaux neuronaux(Python)

J'ai opté pour le logiciel **Python** .

Chargement des librairies Python nécessaires

```
import cv2
import pandas as pd
import numpy as np
import keras
import tensorflow as tf
import os
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

Chargement des données images

```
path= "/Users/david/Desktop/daviddtuto1"
classes= ["Carapuce", "Salameche", "pikachu", "rondoudou"]

data= []
for j in classes:
    data_dir=path+str("/") +str(j)
    for img in os.listdir(data_dir):
        if img.endswith("jpeg"):
            pic = cv2.imread(os.path.join(data_dir,img))
            pic = cv2.cvtColor(pic,cv2.COLOR_BGR2RGB)
            pic = cv2.resize(pic,(28,28))
            image=np.array(pic).flatten()
            data.append([image,j])

feat=[]
lab=[]
for f, l in data:
    feat.append(f)
    lab.append(l)

data
```

```
## [[array([253, 253, 253, ..., 244, 244, 244], dtype=uint8), 'Carapuce'], [array([253, 253, 253, ..., 2
```

Encodage des labels

```
encoder_ = LabelEncoder()
y = encoder_.fit_transform(lab)
```

Split des données en données d'entraînement et données test

```
x_train,x_test,y_train,y_test= train_test_split(feats,y,test_size=0.2)
```

Perceptron Multi-couche

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(input_shape=(None,2352),units=128,activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(4,activation="relu")
])
```

```
model.summary()
```

```
## Model: "sequential"
```

```
## -----
## Layer (type)           Output Shape          Param #
## -----
## dense (Dense)          (None, None, 128)     301184
## -----
## dense_1 (Dense)         (None, None, 64)      8256
## -----
## dense_2 (Dense)         (None, None, 32)      2080
## -----
## dense_3 (Dense)         (None, None, 16)      528
## -----
## dense_4 (Dense)         (None, None, 8)       136
## -----
## dense_5 (Dense)         (None, None, 4)       36
## -----
## Total params: 312,220
## Trainable params: 312,220
## Non-trainable params: 0
## -----
```

```
model.compile(
    optimizer='adam',
    loss= tf.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'])
```

```
x_train = np.matrix(x_train)
x_test = np.matrix(x_test)
```

Entraînement du modèle et évaluation

Nous avons eu quelques problèmes avec la compilation de ce code **Python** sous **R**. En effet, il semble avoir un conflit entre la compilation de modèle **tensorflow** et **Latex** qui permet de compiler les fichiers Rmarkdown,

on effectuera la compilation sur les données **train** et l'évaluation sur les données **test** directement sous **Python** `model.fit(x_train,y_train,validation_split=0.2)`

L'évaluation se fait avec la commande : `model.fit(x_test,y_test)` .

Tout de même en regardant les résultats obtenus sous **Python**, on a également un accuracy assez faible en entraînement (inférieur à 0.5) et c'est de même sur les données d'entraînements.

Carte de Kohonen

Présentation

Les cartes de Kohonen, encore appelé cartes auto-organisatrices sont une famille de réseaux neuronaux dont l'objectif est de classer des données similaires selon une certaine notion de voisinage. Comme tout réseau neuronal, il cherche à imiter un comportement physiologique de notre cerveau. En effet, dans notre cerveau, des zones spécialisées répondent à un certain type de stimuli c'est à dire que des stimuli de même nature activent une région du cerveau particulière.

Ces cartes sont constituées d'un ensemble de neurones (μ_1, \dots, μ_N) lesquels sont reliés par une forme récurrente de voisinage. Les neurones sont initialement répartis selon ce système de voisinage. Le réseau évolue ensuite puisque chaque point de l'espace parmi l'ensemble (X_1, \dots, X_K) attire le neurone le plus proche vers lui, ce neurone attirant à son tour ses voisins.

- Chaque neurone du graphique ci-dessus représente un ensemble d'observations des données initiales
- Chaque neurone(noeud) possède un vecteur de poids représenté ici par $w_{i,j}$.
- Les positions de neurones sont importantes: ** Deux neurones voisins présentent des vecteurs de poids similaires ** Des neurones contigus représentent un profil particulier des données.

Les cartes de Kohonen appartiennent à la famille des **réseaux de compétition** c'est à dire à chaque étape on cherche à trouver le neurone performant une certaine mesure. Dans le cas des cartes de Kohonen, on cherche le **neurone gagnant** celui qui se rapproche le plus d'une certaine donnée X_i appartenant aux données de départ.

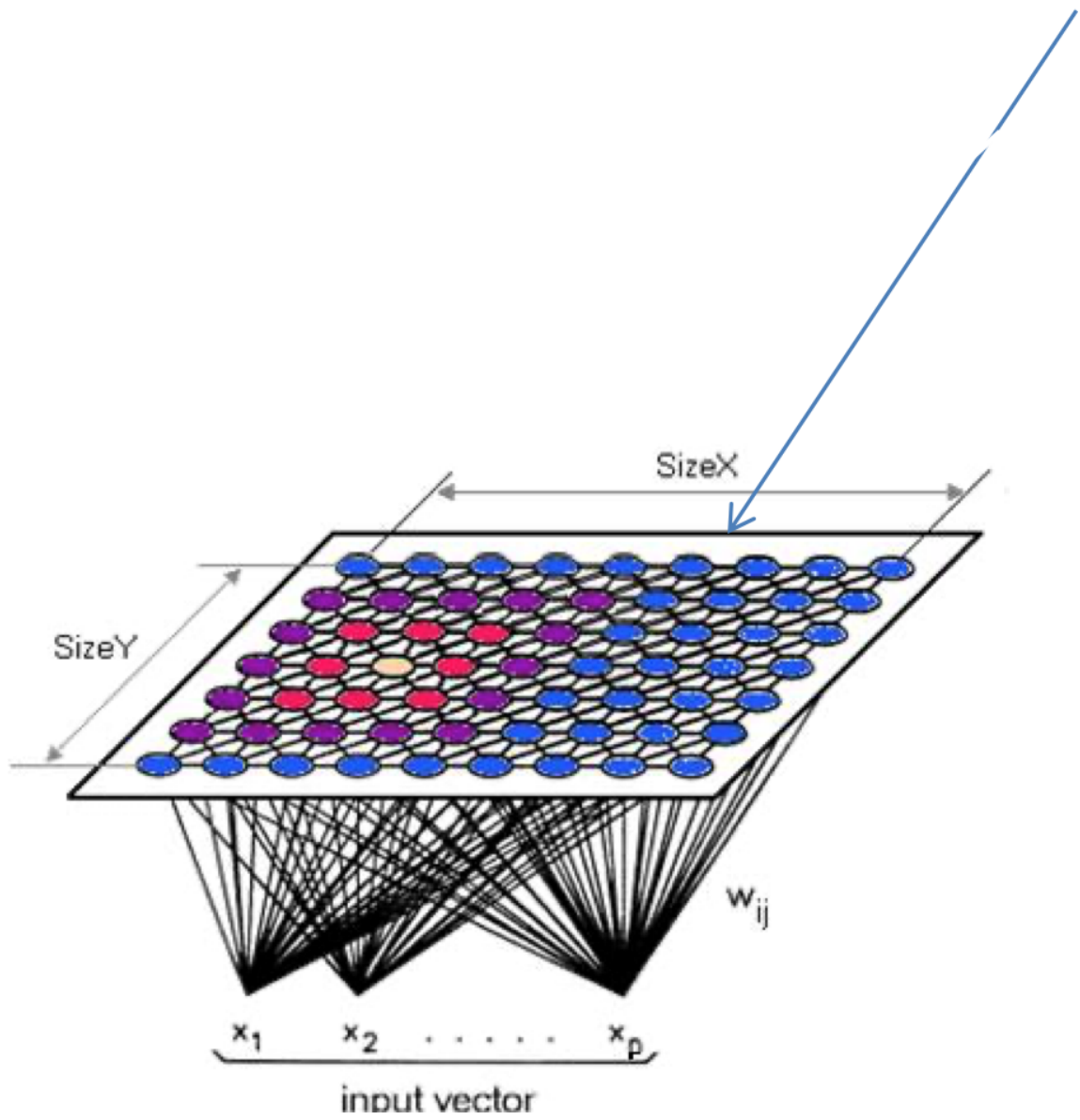


Figure 3: Carte de Kohonen

Algorithme

Entrée : Un tableau de données, taille et forme de la carte.

Sortie : carte topologique avec les poids.

- Initialisation aléatoire des poids des noeuds
- Sélectionner un individu aléatoirement
- Chercher le noeud qui est le plus proche (noeud gagnant)
- Les poids de ce noeud est mis à jour

* Les poids des noeuds voisins également, mais dans une moindre mesure

* Reduire graduellement l'intensité de la mise à jour au fur et à mesure

- Répéter de 1 à 6 pour Tmax itérations“

Règle de mise à jour d'un noeud j alors que j^* est le noeud gagnant: $w_{t+1}(j) = w_t(j) + \epsilon_t * h_t(j, j^*) * (w_t(j) - x)$

* $h()$ est une fonction de voisinage. Son amplitude diminue au fil des itérations

* $h_t(j, j^*) = \exp(-\frac{d^2(j, j^*)}{2\sigma^2(t)})$ où $d(j, j^*)$ est la distance entre les noeuds j et j^* sur la carte et

$\sigma(t) = \sigma_0 \exp(-\frac{t}{T_{max}})$

* ϵ est le pas d'apprentissage. Sa valeur diminue au fil des itérations

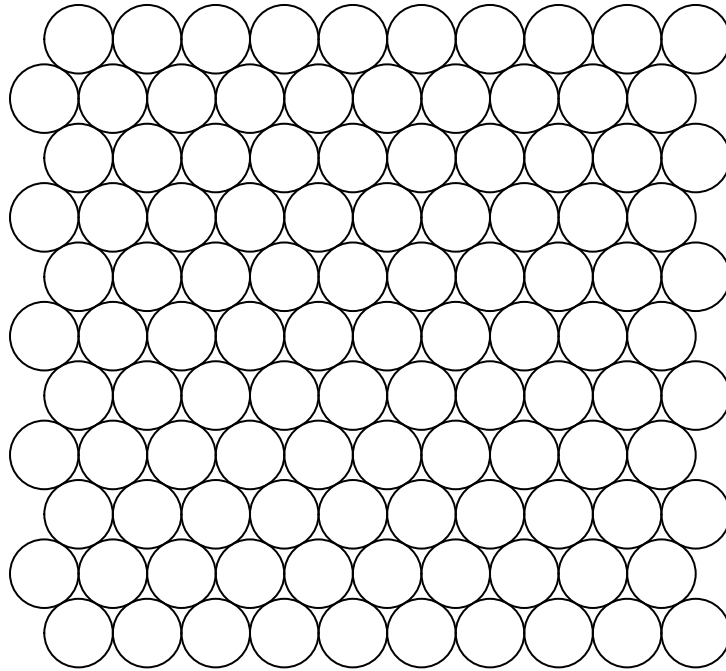
* $\epsilon(t) = \epsilon_0 \exp(-\frac{t}{T_{max}})$

Application: Carte de Kohonen sur nos données Pokemon

```
x <- scale(train[,1:784])  
som.r <- som(x, grid = somgrid(10, 11, "hexagonal"), toroidal = TRUE)
```

```
plot(som.r, type = "mapping")
```

Mapping plot



Vecteurs référents

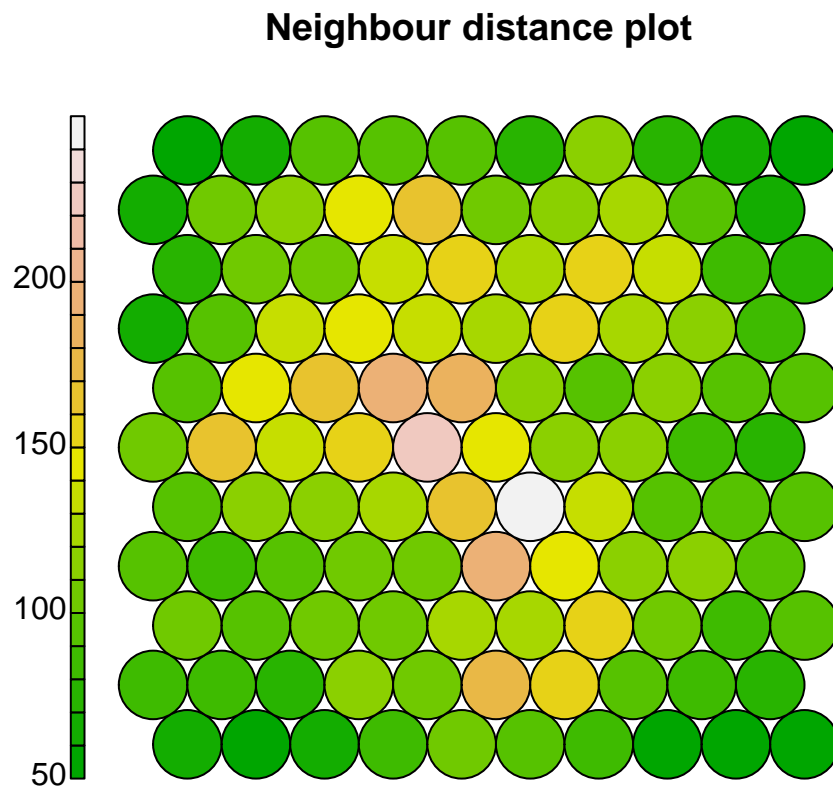
```
coolBlueHotRed <- function(n, alpha = 1) {  
  rainbow(n, end=4/6, alpha=alpha)[n:1]  
}  
plot(som.r, type="codes", codeRendering="segments", main="Profil des vecteurs référents", palette.name=  
text(som.r$grid$pts, labels = som.r$unit.classif, cex = 1.5)
```

Profil des vecteurs référents

■ X1	■ X263	■ X525
■ X2	■ X264	■ X526
■ X3	■ X265	■ X527
■ X4	■ X266	■ X528
■ X5	■ X267	■ X529
■ X6	■ X268	■ X530
■ X7	■ X269	■ X531
■ X8	■ X270	■ X532
■ X9	■ X271	■ X533
■ X10	■ X272	■ X534
■ X11	■ X273	■ X535
■ X12	■ X274	■ X536
■ X13	■ X275	■ X537
■ X14	■ X276	■ X538
■ X15	■ X277	■ X539
■ X16	■ X278	■ X540
■ X17	■ X279	■ X541
■ X18	■ X280	■ X542

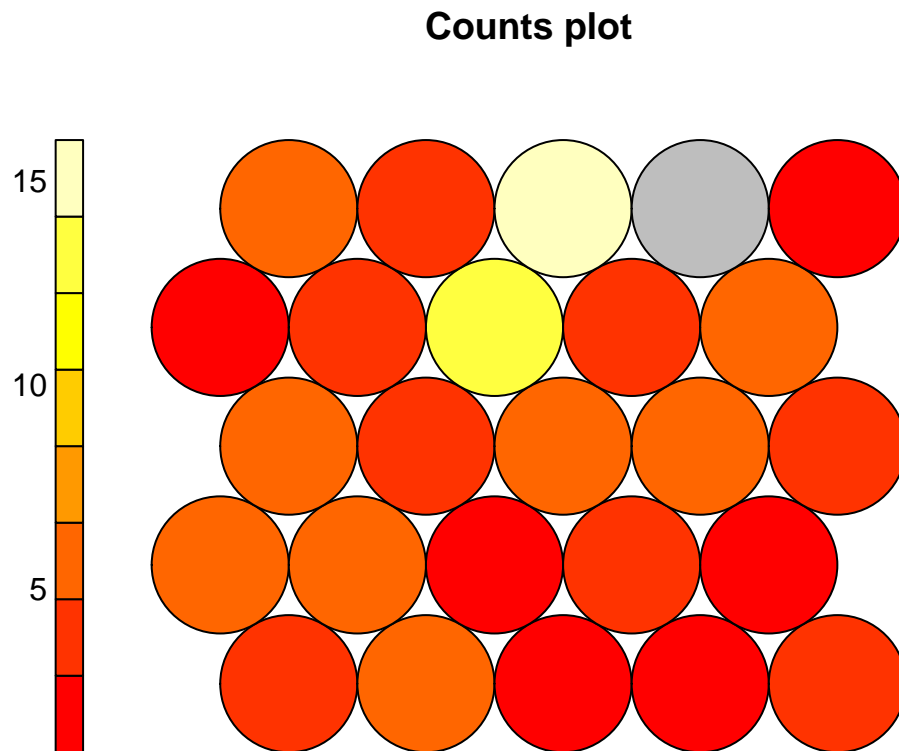
Plot des distances

```
plot(som.r, type = "dist.neighbours", palette.name = terrain.colors)
```




Count Plot

```
kohmap <- xyf(x, classvec2classmat(train[,785]),  
             grid = somgrid(5, 5, "hexagonal"), rlen=100)  
plot(kohmap, type="counts")
```



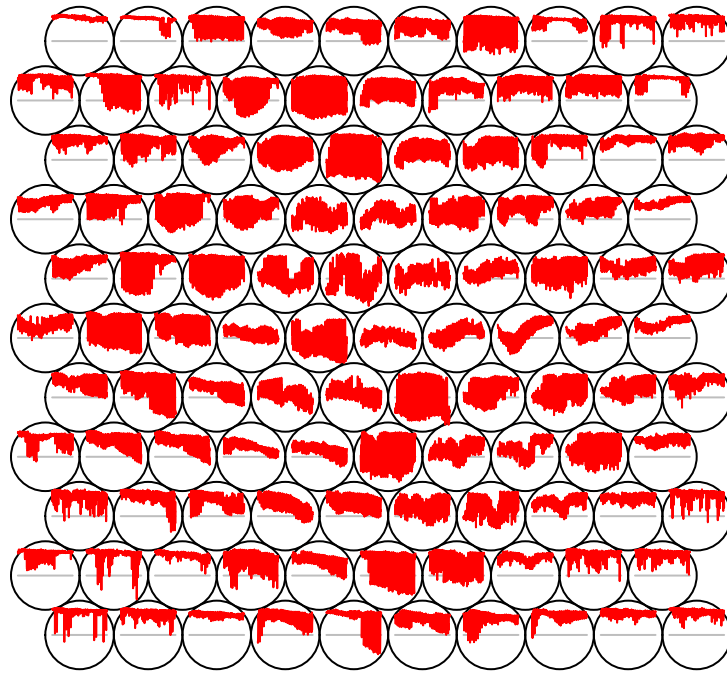
```
plot(som.r , type="codes",codeRendering='segments')
```



■ X1	■ X263	■ X525
■ X2	■ X264	■ X526
■ X3	■ X265	■ X527
■ X4	■ X266	■ X528
■ X5	■ X267	■ X529
■ X6	■ X268	■ X530
■ X7	■ X269	■ X531
■ X8	■ X270	■ X532
■ X9	■ X271	■ X533
■ X10	■ X272	■ X534
■ X11	■ X273	■ X535
■ X12	■ X274	■ X536
■ X13	■ X275	■ X537
■ X14	■ X276	■ X538
■ X15	■ X277	■ X539
■ X16	■ X278	■ X540
■ X17	■ X279	■ X541
■ X18	■ X280	■ X542
■ X19	■ X281	■ X543

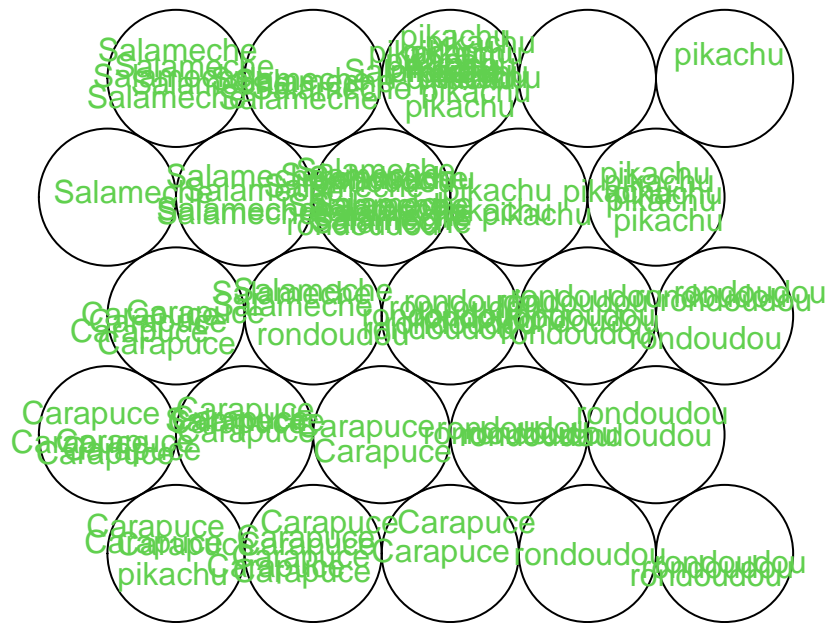

```
plot(som.r, main = "Default SOM Plot")
```

Default SOM Plot



```
plot(kohmap, type="mapping",
     labels = train[,785], col = 2+1,
     main = "mapping plot")
```

mapping plot



Prediction avec les cartes de Kohonen

```
ujimatrix <- scale(test[,1:784]) # Il faut normaliser les données

som.prediction <- predict(som.r, newdata = ujimatrix,
                          trainX = x,
                          trainY = factor(train[,785]))

table(test[,785], som.prediction$prediction)
```

```
##
##           Carapuce pikachu rondoudou Salameche
## Carapuce         10         0          3          0
## pikachu           1         2         10          1
## rondoudou         4         1          5          1
## Salameche         3         1          3          3
```

Apprentissage par renforcement

Présentation

En apprentissage automatique, l'**apprentissage par renforcement** consiste, pour un **agent autonome** à apprendre les actions à prendre, à partir d'expériences, de façon à optimiser une récompense quantitative au cours du temps.

L'agent est plongé au sein d'un environnement, et prend ses décisions en fonction de son état courant. En retour, l'environnement procure à l'agent une récompense, qui peut être positive ou négative.

L'apprentissage par renforcement imite un comportement humain, dans le sens où en tant qu'humain, nous expérimentons certaines choses. Lorsqu'elles ont un résultat positif et induisent des récompenses, on conclut que ces expériences sont positives et qu'elles doivent être répétées. Inversement, si le résultat de l'expérience n'est pas concluant, on le mémorise pour ne plus faire la même erreur.

Il existe plusieurs types d'apprentissage par renforcement, nous allons nous intéresser principalement au **Q-Learning**. Le Q-learning est à la fois relativement simple et permet en même temps de comprendre des mécanismes d'apprentissage communs à beaucoup d'autres modèles.

Q-Learning

L'algorithme d'apprentissage par renforcement qui est le plus utilisé est le Q-Learning. Son fonctionnement repose sur le calcul de l'action optimale. C'est celle qui maximise l'espérance des récompenses des prochains états, en prenant en compte un facteur d'actualisation.

Brièvement, le facteur d'actualisation est une constante comprise entre 0 et 1 qui donne une idée de l'importance des états futurs dans notre étude. S'il vaut 0, on ne considère que les états présents. S'il vaut 1 on donne plus d'importance à des états futurs.

Le facteur d'apprentissage (learning rate) est un indicateur de la vitesse d'apprentissage. Un facteur d'apprentissage élevé signifie que l'état est très modifié entre deux étapes.

Pour illustrer de manière introductive, un algorithme de Q-learning fonctionne pour résoudre un problème basique. Par exemple, dans le jeu du labyrinthe, l'objectif du jeu est d'apprendre au robot à sortir du labyrinthe le plus rapidement possible alors qu'il est placé aléatoirement sur l'une des cases blanches. Pour cela, il y a trois étapes centrales dans le processus d'apprentissage :

Connaitre : définir une fonction action-valeur **Q** ; Renforcer ses connaissances : mettre à jour la fonction **Q** ; * **Agir** : adopter une stratégie d'actions **PI** * **Ainsi**, le Q-learning est un algorithme d'apprentissage par renforcement qui cherche à trouver la meilleure action à entreprendre compte tenu de l'état actuel. Il est considéré comme hors politique parce que la fonction de Q-learning apprend des actions qui sont en dehors de la politique actuelle, comme prendre des actions aléatoires, et donc une politique n'est pas nécessaire. Plus précisément, le Q-learning cherche à apprendre une politique qui maximise la récompense totale.

Le "**Q**" de **Q-learning** est synonyme de qualité. Dans ce cas, la qualité représente l'utilité d'une action donnée pour obtenir une récompense future

Un algorithme d'apprentissage par renforcement peut être vu comme un processus de décisions markoviennes.

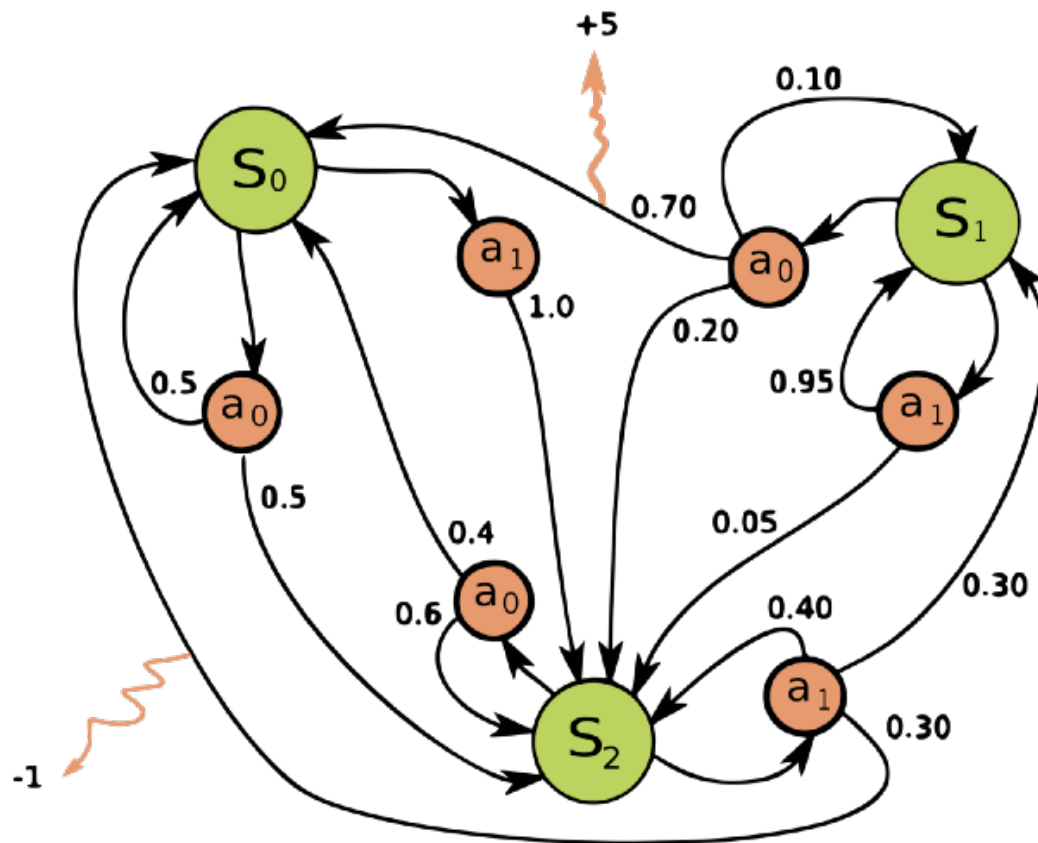


Figure 4: Structure markovienne de Q-Learning

Sur la figure précédente, les a_i représentent les différentes décisions qui ont été choisies par l'agent. Les S_i représentent les différents états dans lesquels l'environnement est mis suite aux décisions prises par l'agent.

Les prises de décisions possèdent également des probabilités, c'est à dire des chances de se passer ou non. On voit également que selon les décisions qu'on prendra, on a des récompenses:

* Si on est en S_2 après avoir pris l'action a_1 , on a 30 de chances de se retrouver en S_0 d'où on aurait un petit malus, on aurait un -1 .

- Si on est en S_1 après avoir pris la décision a_0 , on a 70 de chances de se retrouver en S_0 et là on aurait un bonus de $+5$.
- Sur tous les états, on a pas de récompenses donc aurait une récompense de 0.

Ce genre de processus de décisions markoviennes est principalement dû à **Bellman**.

Politique

Ce qu'on appelle une politique c'est l'ensemble des actions que l'on va prendre à partir d'un certain état S . Par exemple pour le graphe précédent, on peut imaginer une politique π_1 tel que:

$S_0 \longrightarrow a_0$
 $S_1 \longrightarrow a_0$
 $S_2 \longrightarrow a_0$.

Dans notre exemple, le nombre de politique sera relativement faible vu qu'on a un modèle assez simple, mais en général on en a beaucoup plus. Et donc, parmi ces dernières, il existe une politique dite **optimale** notée souvent π^* et c'est la politique qui rapportera le plus de gain.

Selon **Bellman**, la politique optimale π^* et la valeur V (valeur pour chacun des états) vérifie la formule-suivante:

$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

γ : taux de rabais : $0 < \gamma < 1$. Très souvent, on le prend proche de 1. T : constitue la position. En fait, $T(s, a, s')$ c'est la probabilité de passer de l'état s à l'état s' selon une certaine action a .

$R(s, a, s')$: constitue la récompense.

Ce qu'on peut espérer dans la formule de **Bellman**, c'est que si l'on a la politique **optimale** alors on obtient la somme des récompenses avec un rabais futur que l'on peut espérer avoir.

Le paramètre de rabais est très important comme nous allons le voir sur un code Python plutôt. En effet, c'est ce paramètre qui nous assure de la convergence.

Ce que va vouloir faire l'agent, c'est de savoir quelle décision prendre selon qu'on est dans un environnement. On va donc chercher à mettre des notes à toutes les décisions différentes dans un environnement précis, et la décision à prendre sera celle qui a une note maximale.

Bellman a également pensé à cela et propose une valeur qui est :

$Q_{k+1}(s) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma * \max_{a'} Q_k(s', a')]$ qui prend en paramètre un état et une action. Cette formule nous intéresse beaucoup plus car elle nous donne la décision qu'il fallait prendre dans chaque état.

Application

J'ai mis en place l'algorithme de la figure précédente.

```
import numpy as np
import time
import numpy as np
import time

Q=[[0, 0], [0, 0], [0, 0]]

T=[[0.50, 0.00, 0.50], [0.00, 0.00, 1.00]],
  [[0.70, 0.10, 0.20], [0.00, 0.95, 0.05]],
  [[0.40, 0.00, 0.60], [0.30, 0.30, 0.40]]

R=[[ [ 0.00, 0.00, 0.00], [ 0.00, 0.00, 0.00]],
  [[+5.00, 0.00, 0.00], [ 0.00, 0.00, 0.00]],
  [[ 0.00, 0.00, 0.00], [-1.00, 0.00, 0.00]]

gamma=0.95

for i in range(200):
    time.sleep(0.05)
    tab_somme_action=[]
    for S in range(3):
        for A in range(2):
            somme=0
            for s in range(3):
                somme+=T[S][A][s]*(R[S][A][s]+gamma*np.max(Q[s]))
            Q[S][A]=somme

    print("-----")
    print("Iteration:", i)
    for S in range(3):
        print()
        for A in range(2):
            text="Q[etat:{}, action:{}]={: +10.4f}".format(S, A, Q[S][A])
            if A==np.argmax(Q[S]):
                text=text+" <-"
            print(text)

## -----
## Iteration: 0
##
## Q[etat:0, action:0]= +0.0000 <-
## Q[etat:0, action:1]= +0.0000
##
## Q[etat:1, action:0]= +3.5000 <-
```

```

## Q[etat:1, action:1]= +3.1587
##
## Q[etat:2, action:0]= +0.0000
## Q[etat:2, action:1]= +0.6975 <-
## -----
## Iteration: 1
##
## Q[etat:0, action:0]= +0.3313
## Q[etat:0, action:1]= +0.6626 <-
##
## Q[etat:1, action:0]= +4.4057 <-
## Q[etat:1, action:1]= +4.0092
##
## Q[etat:2, action:0]= +0.6494
## Q[etat:2, action:1]= +1.4095 <-
## -----
## Iteration: 2
##
## Q[etat:0, action:0]= +0.9843
## Q[etat:0, action:1]= +1.3390 <-
##
## Q[etat:1, action:0]= +5.0768 <-
## Q[etat:1, action:1]= +4.6488
##
## Q[etat:2, action:0]= +1.3123
## Q[etat:2, action:1]= +2.0641 <-
## -----
## Iteration: 3
##
## Q[etat:0, action:0]= +1.6165
## Q[etat:0, action:1]= +1.9609 <-
##
## Q[etat:1, action:0]= +5.6785 <-
## Q[etat:1, action:1]= +5.2229
##
## Q[etat:2, action:0]= +1.9217
## Q[etat:2, action:1]= +2.6616 <-
## -----
## Iteration: 4
##
## Q[etat:0, action:0]= +2.1957
## Q[etat:0, action:1]= +2.5285 <-
##
## Q[etat:1, action:0]= +6.2266 <-
## Q[etat:1, action:1]= +5.7460
##
## Q[etat:2, action:0]= +2.4780
## Q[etat:2, action:1]= +3.2066 <-
## -----
## Iteration: 5
##
## Q[etat:0, action:0]= +2.7242
## Q[etat:0, action:1]= +3.0463 <-
##

```



```

## Q[etat:1, action:0]= +6.7266 <-
## Q[etat:1, action:1]= +6.2231
##
## Q[etat:2, action:0]= +2.9854
## Q[etat:2, action:1]= +3.7038 <-
## -----
## Iteration: 6
##
## Q[etat:0, action:0]= +3.2063
## Q[etat:0, action:1]= +3.5186 <-
##
## Q[etat:1, action:0]= +7.1826 <-
## Q[etat:1, action:1]= +6.6582
##
## Q[etat:2, action:0]= +3.4482
## Q[etat:2, action:1]= +4.1573 <-
## -----
## Iteration: 7
##
## Q[etat:0, action:0]= +3.6460
## Q[etat:0, action:1]= +3.9494 <-
##
## Q[etat:1, action:0]= +7.5986 <-
## Q[etat:1, action:1]= +7.0552
##
## Q[etat:2, action:0]= +3.8704
## Q[etat:2, action:1]= +4.5710 <-
## -----
## Iteration: 8
##
## Q[etat:0, action:0]= +4.0472
## Q[etat:0, action:1]= +4.3424 <-
##
## Q[etat:1, action:0]= +7.9780 <-
## Q[etat:1, action:1]= +7.4173
##
## Q[etat:2, action:0]= +4.2556
## Q[etat:2, action:1]= +4.9483 <-
## -----
## Iteration: 9
##
## Q[etat:0, action:0]= +4.4131
## Q[etat:0, action:1]= +4.7009 <-
##
## Q[etat:1, action:0]= +8.3242 <-
## Q[etat:1, action:1]= +7.7476
##
## Q[etat:2, action:0]= +4.6069
## Q[etat:2, action:1]= +5.2925 <-
## -----
## Iteration: 10
##
## Q[etat:0, action:0]= +4.7468
## Q[etat:0, action:1]= +5.0279 <-

```

```

##
## Q[etat:1, action:0]= +8.6399 <-
## Q[etat:1, action:1]= +8.0489
##
## Q[etat:2, action:0]= +4.9273
## Q[etat:2, action:1]= +5.6065 <-
## -----
## Iteration: 11
##
## Q[etat:0, action:0]= +5.0513
## Q[etat:0, action:1]= +5.3261 <-
##
## Q[etat:1, action:0]= +8.9279 <-
## Q[etat:1, action:1]= +8.3237
##
## Q[etat:2, action:0]= +5.2196
## Q[etat:2, action:1]= +5.8929 <-
## -----
## Iteration: 12
##
## Q[etat:0, action:0]= +5.3290
## Q[etat:0, action:1]= +5.5982 <-
##
## Q[etat:1, action:0]= +9.1906 <-
## Q[etat:1, action:1]= +8.5744
##
## Q[etat:2, action:0]= +5.4862
## Q[etat:2, action:1]= +6.1541 <-
## -----
## Iteration: 13
##
## Q[etat:0, action:0]= +5.5823
## Q[etat:0, action:1]= +5.8464 <-
##
## Q[etat:1, action:0]= +9.4302 <-
## Q[etat:1, action:1]= +8.8031
##
## Q[etat:2, action:0]= +5.7295
## Q[etat:2, action:1]= +6.3924 <-
## -----
## Iteration: 14
##
## Q[etat:0, action:0]= +5.8134
## Q[etat:0, action:1]= +6.0728 <-
##
## Q[etat:1, action:0]= +9.6488 <-
## Q[etat:1, action:1]= +9.0117
##
## Q[etat:2, action:0]= +5.9513
## Q[etat:2, action:1]= +6.6098 <-
## -----
## Iteration: 15
##
## Q[etat:0, action:0]= +6.0242

```

```

## Q[etat:0, action:1]= +6.2793 <-
##
## Q[etat:1, action:0]= +9.8482 <-
## Q[etat:1, action:1]= +9.2020
##
## Q[etat:2, action:0]= +6.1537
## Q[etat:2, action:1]= +6.8080 <-
## -----
## Iteration: 16
##
## Q[etat:0, action:0]= +6.2165
## Q[etat:0, action:1]= +6.4676 <-
##
## Q[etat:1, action:0]= +10.0301 <-
## Q[etat:1, action:1]= +9.3755
##
## Q[etat:2, action:0]= +6.3383
## Q[etat:2, action:1]= +6.9889 <-
## -----
## Iteration: 17
##
## Q[etat:0, action:0]= +6.3919
## Q[etat:0, action:1]= +6.6395 <-
##
## Q[etat:1, action:0]= +10.1960 <-
## Q[etat:1, action:1]= +9.5339
##
## Q[etat:2, action:0]= +6.5067
## Q[etat:2, action:1]= +7.1539 <-
## -----
## Iteration: 18
##
## Q[etat:0, action:0]= +6.5518
## Q[etat:0, action:1]= +6.7962 <-
##
## Q[etat:1, action:0]= +10.3473 <-
## Q[etat:1, action:1]= +9.6783
##
## Q[etat:2, action:0]= +6.6603
## Q[etat:2, action:1]= +7.3044 <-
## -----
## Iteration: 19
##
## Q[etat:0, action:0]= +6.6978
## Q[etat:0, action:1]= +6.9392 <-
##
## Q[etat:1, action:0]= +10.4854 <-
## Q[etat:1, action:1]= +9.8100
##
## Q[etat:2, action:0]= +6.8004
## Q[etat:2, action:1]= +7.4417 <-
## -----
## Iteration: 20
##

```

```

## Q[etat:0, action:0]= +6.8309
## Q[etat:0, action:1]= +7.0696 <-
##
## Q[etat:1, action:0]= +10.6113 <-
## Q[etat:1, action:1]= +9.9302
##
## Q[etat:2, action:0]= +6.9282
## Q[etat:2, action:1]= +7.5669 <-
## -----
## Iteration: 21
##
## Q[etat:0, action:0]= +6.9523
## Q[etat:0, action:1]= +7.1885 <-
##
## Q[etat:1, action:0]= +10.7262 <-
## Q[etat:1, action:1]= +10.0398
##
## Q[etat:2, action:0]= +7.0448
## Q[etat:2, action:1]= +7.6811 <-
## -----
## Iteration: 22
##
## Q[etat:0, action:0]= +7.0631
## Q[etat:0, action:1]= +7.2970 <-
##
## Q[etat:1, action:0]= +10.8309 <-
## Q[etat:1, action:1]= +10.1398
##
## Q[etat:2, action:0]= +7.1511
## Q[etat:2, action:1]= +7.7853 <-
## -----
## Iteration: 23
##
## Q[etat:0, action:0]= +7.1641
## Q[etat:0, action:1]= +7.3960 <-
##
## Q[etat:1, action:0]= +10.9265 <-
## Q[etat:1, action:1]= +10.2310
##
## Q[etat:2, action:0]= +7.2481
## Q[etat:2, action:1]= +7.8803 <-
## -----
## Iteration: 24
##
## Q[etat:0, action:0]= +7.2563
## Q[etat:0, action:1]= +7.4863 <-
##
## Q[etat:1, action:0]= +11.0137 <-
## Q[etat:1, action:1]= +10.3142
##
## Q[etat:2, action:0]= +7.3366
## Q[etat:2, action:1]= +7.9670 <-
## -----
## Iteration: 25

```

```

##
## Q[etat:0, action:0]= +7.3403
## Q[etat:0, action:1]= +7.5687 <-
##
## Q[etat:1, action:0]= +11.0932 <-
## Q[etat:1, action:1]= +10.3900
##
## Q[etat:2, action:0]= +7.4173
## Q[etat:2, action:1]= +8.0461 <-
## -----
## Iteration: 26
##
## Q[etat:0, action:0]= +7.4170
## Q[etat:0, action:1]= +7.6438 <-
##
## Q[etat:1, action:0]= +11.1657 <-
## Q[etat:1, action:1]= +10.4593
##
## Q[etat:2, action:0]= +7.4909
## Q[etat:2, action:1]= +8.1182 <-
## -----
## Iteration: 27
##
## Q[etat:0, action:0]= +7.4870
## Q[etat:0, action:1]= +7.7123 <-
##
## Q[etat:1, action:0]= +11.2319 <-
## Q[etat:1, action:1]= +10.5224
##
## Q[etat:2, action:0]= +7.5581
## Q[etat:2, action:1]= +8.1840 <-
## -----
## Iteration: 28
##
## Q[etat:0, action:0]= +7.5508
## Q[etat:0, action:1]= +7.7748 <-
##
## Q[etat:1, action:0]= +11.2923 <-
## Q[etat:1, action:1]= +10.5800
##
## Q[etat:2, action:0]= +7.6193
## Q[etat:2, action:1]= +8.2441 <-
## -----
## Iteration: 29
##
## Q[etat:0, action:0]= +7.6090
## Q[etat:0, action:1]= +7.8319 <-
##
## Q[etat:1, action:0]= +11.3473 <-
## Q[etat:1, action:1]= +10.6325
##
## Q[etat:2, action:0]= +7.6752
## Q[etat:2, action:1]= +8.2988 <-
## -----

```

```

## Iteration: 30
##
## Q[etat:0, action:0]= +7.6621
## Q[etat:0, action:1]= +7.8839 <-
##
## Q[etat:1, action:0]= +11.3975 <-
## Q[etat:1, action:1]= +10.6805
##
## Q[etat:2, action:0]= +7.7262
## Q[etat:2, action:1]= +8.3487 <-
## -----
## Iteration: 31
##
## Q[etat:0, action:0]= +7.7105
## Q[etat:0, action:1]= +7.9313 <-
##
## Q[etat:1, action:0]= +11.4433 <-
## Q[etat:1, action:1]= +10.7242
##
## Q[etat:2, action:0]= +7.7727
## Q[etat:2, action:1]= +8.3943 <-
## -----
## Iteration: 32
##
## Q[etat:0, action:0]= +7.7547
## Q[etat:0, action:1]= +7.9746 <-
##
## Q[etat:1, action:0]= +11.4851 <-
## Q[etat:1, action:1]= +10.7641
##
## Q[etat:2, action:0]= +7.8151
## Q[etat:2, action:1]= +8.4359 <-
## -----
## Iteration: 33
##
## Q[etat:0, action:0]= +7.7950
## Q[etat:0, action:1]= +8.0141 <-
##
## Q[etat:1, action:0]= +11.5233 <-
## Q[etat:1, action:1]= +10.8004
##
## Q[etat:2, action:0]= +7.8538
## Q[etat:2, action:1]= +8.4738 <-
## -----
## Iteration: 34
##
## Q[etat:0, action:0]= +7.8317
## Q[etat:0, action:1]= +8.0501 <-
##
## Q[etat:1, action:0]= +11.5580 <-
## Q[etat:1, action:1]= +10.8336
##
## Q[etat:2, action:0]= +7.8891
## Q[etat:2, action:1]= +8.5083 <-

```

```

## -----
## Iteration: 35
##
## Q[etat:0, action:0]= +7.8652
## Q[etat:0, action:1]= +8.0829 <-
##
## Q[etat:1, action:0]= +11.5897 <-
## Q[etat:1, action:1]= +10.8639
##
## Q[etat:2, action:0]= +7.9213
## Q[etat:2, action:1]= +8.5399 <-
## -----
## Iteration: 36
##
## Q[etat:0, action:0]= +7.8958
## Q[etat:0, action:1]= +8.1129 <-
##
## Q[etat:1, action:0]= +11.6187 <-
## Q[etat:1, action:1]= +10.8915
##
## Q[etat:2, action:0]= +7.9506
## Q[etat:2, action:1]= +8.5686 <-
## -----
## Iteration: 37
##
## Q[etat:0, action:0]= +7.9237
## Q[etat:0, action:1]= +8.1402 <-
##
## Q[etat:1, action:0]= +11.6451 <-
## Q[etat:1, action:1]= +10.9167
##
## Q[etat:2, action:0]= +7.9774
## Q[etat:2, action:1]= +8.5949 <-
## -----
## Iteration: 38
##
## Q[etat:0, action:0]= +7.9492
## Q[etat:0, action:1]= +8.1651 <-
##
## Q[etat:1, action:0]= +11.6691 <-
## Q[etat:1, action:1]= +10.9396
##
## Q[etat:2, action:0]= +8.0018
## Q[etat:2, action:1]= +8.6188 <-
## -----
## Iteration: 39
##
## Q[etat:0, action:0]= +7.9724
## Q[etat:0, action:1]= +8.1879 <-
##
## Q[etat:1, action:0]= +11.6911 <-
## Q[etat:1, action:1]= +10.9606
##
## Q[etat:2, action:0]= +8.0241

```

```

## Q[etat:2, action:1]= +8.6407 <-
## -----
## Iteration: 40
##
## Q[etat:0, action:0]= +7.9936
## Q[etat:0, action:1]= +8.2086 <-
##
## Q[etat:1, action:0]= +11.7111 <-
## Q[etat:1, action:1]= +10.9797
##
## Q[etat:2, action:0]= +8.0445
## Q[etat:2, action:1]= +8.6606 <-
## -----
## Iteration: 41
##
## Q[etat:0, action:0]= +8.0129
## Q[etat:0, action:1]= +8.2275 <-
##
## Q[etat:1, action:0]= +11.7294 <-
## Q[etat:1, action:1]= +10.9971
##
## Q[etat:2, action:0]= +8.0630
## Q[etat:2, action:1]= +8.6787 <-
## -----
## Iteration: 42
##
## Q[etat:0, action:0]= +8.0305
## Q[etat:0, action:1]= +8.2448 <-
##
## Q[etat:1, action:0]= +11.7460 <-
## Q[etat:1, action:1]= +11.0131
##
## Q[etat:2, action:0]= +8.0799
## Q[etat:2, action:1]= +8.6953 <-
## -----
## Iteration: 43
##
## Q[etat:0, action:0]= +8.0466
## Q[etat:0, action:1]= +8.2606 <-
##
## Q[etat:1, action:0]= +11.7613 <-
## Q[etat:1, action:1]= +11.0276
##
## Q[etat:2, action:0]= +8.0953
## Q[etat:2, action:1]= +8.7104 <-
## -----
## Iteration: 44
##
## Q[etat:0, action:0]= +8.0612
## Q[etat:0, action:1]= +8.2749 <-
##
## Q[etat:1, action:0]= +11.7751 <-
## Q[etat:1, action:1]= +11.0408
##

```



```

## Q[etat:2, action:0]= +8.1094
## Q[etat:2, action:1]= +8.7242 <-
## -----
## Iteration: 45
##
## Q[etat:0, action:0]= +8.0746
## Q[etat:0, action:1]= +8.2880 <-
##
## Q[etat:1, action:0]= +11.7878 <-
## Q[etat:1, action:1]= +11.0529
##
## Q[etat:2, action:0]= +8.1223
## Q[etat:2, action:1]= +8.7368 <-
## -----
## Iteration: 46
##
## Q[etat:0, action:0]= +8.0868
## Q[etat:0, action:1]= +8.3000 <-
##
## Q[etat:1, action:0]= +11.7993 <-
## Q[etat:1, action:1]= +11.0639
##
## Q[etat:2, action:0]= +8.1340
## Q[etat:2, action:1]= +8.7483 <-
## -----
## Iteration: 47
##
## Q[etat:0, action:0]= +8.0979
## Q[etat:0, action:1]= +8.3109 <-
##
## Q[etat:1, action:0]= +11.8098 <-
## Q[etat:1, action:1]= +11.0739
##
## Q[etat:2, action:0]= +8.1446
## Q[etat:2, action:1]= +8.7587 <-
## -----
## Iteration: 48
##
## Q[etat:0, action:0]= +8.1081
## Q[etat:0, action:1]= +8.3208 <-
##
## Q[etat:1, action:0]= +11.8194 <-
## Q[etat:1, action:1]= +11.0831
##
## Q[etat:2, action:0]= +8.1544
## Q[etat:2, action:1]= +8.7683 <-
## -----
## Iteration: 49
##
## Q[etat:0, action:0]= +8.1173
## Q[etat:0, action:1]= +8.3299 <-
##
## Q[etat:1, action:0]= +11.8282 <-
## Q[etat:1, action:1]= +11.0914

```

```

##
## Q[etat:2, action:0]= +8.1633
## Q[etat:2, action:1]= +8.7770 <-
## -----
## Iteration: 50
##
## Q[etat:0, action:0]= +8.1258
## Q[etat:0, action:1]= +8.3381 <-
##
## Q[etat:1, action:0]= +11.8362 <-
## Q[etat:1, action:1]= +11.0991
##
## Q[etat:2, action:0]= +8.1714
## Q[etat:2, action:1]= +8.7849 <-
## -----
## Iteration: 51
##
## Q[etat:0, action:0]= +8.1335
## Q[etat:0, action:1]= +8.3457 <-
##
## Q[etat:1, action:0]= +11.8435 <-
## Q[etat:1, action:1]= +11.1060
##
## Q[etat:2, action:0]= +8.1788
## Q[etat:2, action:1]= +8.7922 <-
## -----
## Iteration: 52
##
## Q[etat:0, action:0]= +8.1405
## Q[etat:0, action:1]= +8.3526 <-
##
## Q[etat:1, action:0]= +11.8501 <-
## Q[etat:1, action:1]= +11.1124
##
## Q[etat:2, action:0]= +8.1855
## Q[etat:2, action:1]= +8.7988 <-
## -----
## Iteration: 53
##
## Q[etat:0, action:0]= +8.1469
## Q[etat:0, action:1]= +8.3589 <-
##
## Q[etat:1, action:0]= +11.8562 <-
## Q[etat:1, action:1]= +11.1181
##
## Q[etat:2, action:0]= +8.1917
## Q[etat:2, action:1]= +8.8048 <-
## -----
## Iteration: 54
##
## Q[etat:0, action:0]= +8.1527
## Q[etat:0, action:1]= +8.3646 <-
##
## Q[etat:1, action:0]= +11.8617 <-

```

```

## Q[etat:1, action:1]= +11.1234
##
## Q[etat:2, action:0]= +8.1973
## Q[etat:2, action:1]= +8.8103 <-
## -----
## Iteration: 55
##
## Q[etat:0, action:0]= +8.1581
## Q[etat:0, action:1]= +8.3698 <-
##
## Q[etat:1, action:0]= +11.8667 <-
## Q[etat:1, action:1]= +11.1282
##
## Q[etat:2, action:0]= +8.2024
## Q[etat:2, action:1]= +8.8153 <-
## -----
## Iteration: 56
##
## Q[etat:0, action:0]= +8.1629
## Q[etat:0, action:1]= +8.3746 <-
##
## Q[etat:1, action:0]= +11.8713 <-
## Q[etat:1, action:1]= +11.1326
##
## Q[etat:2, action:0]= +8.2071
## Q[etat:2, action:1]= +8.8199 <-
## -----
## Iteration: 57
##
## Q[etat:0, action:0]= +8.1674
## Q[etat:0, action:1]= +8.3789 <-
##
## Q[etat:1, action:0]= +11.8755 <-
## Q[etat:1, action:1]= +11.1366
##
## Q[etat:2, action:0]= +8.2113
## Q[etat:2, action:1]= +8.8241 <-
## -----
## Iteration: 58
##
## Q[etat:0, action:0]= +8.1714
## Q[etat:0, action:1]= +8.3829 <-
##
## Q[etat:1, action:0]= +11.8794 <-
## Q[etat:1, action:1]= +11.1403
##
## Q[etat:2, action:0]= +8.2152
## Q[etat:2, action:1]= +8.8279 <-
## -----
## Iteration: 59
##
## Q[etat:0, action:0]= +8.1751
## Q[etat:0, action:1]= +8.3865 <-
##

```

```

## Q[etat:1, action:0]= +11.8829 <-
## Q[etat:1, action:1]= +11.1436
##
## Q[etat:2, action:0]= +8.2188
## Q[etat:2, action:1]= +8.8314 <-
## -----
## Iteration: 60
##
## Q[etat:0, action:0]= +8.1785
## Q[etat:0, action:1]= +8.3898 <-
##
## Q[etat:1, action:0]= +11.8861 <-
## Q[etat:1, action:1]= +11.1467
##
## Q[etat:2, action:0]= +8.2220
## Q[etat:2, action:1]= +8.8345 <-
## -----
## Iteration: 61
##
## Q[etat:0, action:0]= +8.1816
## Q[etat:0, action:1]= +8.3928 <-
##
## Q[etat:1, action:0]= +11.8890 <-
## Q[etat:1, action:1]= +11.1494
##
## Q[etat:2, action:0]= +8.2250
## Q[etat:2, action:1]= +8.8374 <-
## -----
## Iteration: 62
##
## Q[etat:0, action:0]= +8.1844
## Q[etat:0, action:1]= +8.3956 <-
##
## Q[etat:1, action:0]= +11.8916 <-
## Q[etat:1, action:1]= +11.1520
##
## Q[etat:2, action:0]= +8.2276
## Q[etat:2, action:1]= +8.8401 <-
## -----
## Iteration: 63
##
## Q[etat:0, action:0]= +8.1869
## Q[etat:0, action:1]= +8.3981 <-
##
## Q[etat:1, action:0]= +11.8940 <-
## Q[etat:1, action:1]= +11.1543
##
## Q[etat:2, action:0]= +8.2301
## Q[etat:2, action:1]= +8.8425 <-
## -----
## Iteration: 64
##
## Q[etat:0, action:0]= +8.1893
## Q[etat:0, action:1]= +8.4003 <-

```

```

##
## Q[etat:1, action:0]= +11.8962 <-
## Q[etat:1, action:1]= +11.1564
##
## Q[etat:2, action:0]= +8.2323
## Q[etat:2, action:1]= +8.8447 <-
## -----
## Iteration: 65
##
## Q[etat:0, action:0]= +8.1914
## Q[etat:0, action:1]= +8.4024 <-
##
## Q[etat:1, action:0]= +11.8982 <-
## Q[etat:1, action:1]= +11.1583
##
## Q[etat:2, action:0]= +8.2344
## Q[etat:2, action:1]= +8.8467 <-
## -----
## Iteration: 66
##
## Q[etat:0, action:0]= +8.1933
## Q[etat:0, action:1]= +8.4043 <-
##
## Q[etat:1, action:0]= +11.9001 <-
## Q[etat:1, action:1]= +11.1600
##
## Q[etat:2, action:0]= +8.2362
## Q[etat:2, action:1]= +8.8485 <-
## -----
## Iteration: 67
##
## Q[etat:0, action:0]= +8.1951
## Q[etat:0, action:1]= +8.4061 <-
##
## Q[etat:1, action:0]= +11.9018 <-
## Q[etat:1, action:1]= +11.1616
##
## Q[etat:2, action:0]= +8.2379
## Q[etat:2, action:1]= +8.8502 <-
## -----
## Iteration: 68
##
## Q[etat:0, action:0]= +8.1967
## Q[etat:0, action:1]= +8.4076 <-
##
## Q[etat:1, action:0]= +11.9033 <-
## Q[etat:1, action:1]= +11.1631
##
## Q[etat:2, action:0]= +8.2395
## Q[etat:2, action:1]= +8.8517 <-
## -----
## Iteration: 69
##
## Q[etat:0, action:0]= +8.1982

```

```

## Q[etat:0, action:1]= +8.4091 <-
##
## Q[etat:1, action:0]= +11.9047 <-
## Q[etat:1, action:1]= +11.1644
##
## Q[etat:2, action:0]= +8.2409
## Q[etat:2, action:1]= +8.8531 <-
## -----
## Iteration: 70
##
## Q[etat:0, action:0]= +8.1995
## Q[etat:0, action:1]= +8.4104 <-
##
## Q[etat:1, action:0]= +11.9059 <-
## Q[etat:1, action:1]= +11.1656
##
## Q[etat:2, action:0]= +8.2422
## Q[etat:2, action:1]= +8.8543 <-
## -----
## Iteration: 71
##
## Q[etat:0, action:0]= +8.2007
## Q[etat:0, action:1]= +8.4116 <-
##
## Q[etat:1, action:0]= +11.9071 <-
## Q[etat:1, action:1]= +11.1667
##
## Q[etat:2, action:0]= +8.2434
## Q[etat:2, action:1]= +8.8555 <-
## -----
## Iteration: 72
##
## Q[etat:0, action:0]= +8.2019
## Q[etat:0, action:1]= +8.4127 <-
##
## Q[etat:1, action:0]= +11.9082 <-
## Q[etat:1, action:1]= +11.1678
##
## Q[etat:2, action:0]= +8.2444
## Q[etat:2, action:1]= +8.8565 <-
## -----
## Iteration: 73
##
## Q[etat:0, action:0]= +8.2029
## Q[etat:0, action:1]= +8.4137 <-
##
## Q[etat:1, action:0]= +11.9091 <-
## Q[etat:1, action:1]= +11.1687
##
## Q[etat:2, action:0]= +8.2454
## Q[etat:2, action:1]= +8.8575 <-
## -----
## Iteration: 74
##

```

```

## Q[etat:0, action:0]= +8.2038
## Q[etat:0, action:1]= +8.4146 <-
##
## Q[etat:1, action:0]= +11.9100 <-
## Q[etat:1, action:1]= +11.1695
##
## Q[etat:2, action:0]= +8.2463
## Q[etat:2, action:1]= +8.8584 <-
## -----
## Iteration: 75
##
## Q[etat:0, action:0]= +8.2047
## Q[etat:0, action:1]= +8.4154 <-
##
## Q[etat:1, action:0]= +11.9108 <-
## Q[etat:1, action:1]= +11.1703
##
## Q[etat:2, action:0]= +8.2471
## Q[etat:2, action:1]= +8.8592 <-
## -----
## Iteration: 76
##
## Q[etat:0, action:0]= +8.2054
## Q[etat:0, action:1]= +8.4162 <-
##
## Q[etat:1, action:0]= +11.9115 <-
## Q[etat:1, action:1]= +11.1710
##
## Q[etat:2, action:0]= +8.2479
## Q[etat:2, action:1]= +8.8599 <-
## -----
## Iteration: 77
##
## Q[etat:0, action:0]= +8.2061
## Q[etat:0, action:1]= +8.4169 <-
##
## Q[etat:1, action:0]= +11.9122 <-
## Q[etat:1, action:1]= +11.1716
##
## Q[etat:2, action:0]= +8.2486
## Q[etat:2, action:1]= +8.8606 <-
## -----
## Iteration: 78
##
## Q[etat:0, action:0]= +8.2068
## Q[etat:0, action:1]= +8.4175 <-
##
## Q[etat:1, action:0]= +11.9128 <-
## Q[etat:1, action:1]= +11.1722
##
## Q[etat:2, action:0]= +8.2492
## Q[etat:2, action:1]= +8.8612 <-
## -----
## Iteration: 79

```

```

##
## Q[etat:0, action:0]= +8.2074
## Q[etat:0, action:1]= +8.4181 <-
##
## Q[etat:1, action:0]= +11.9134 <-
## Q[etat:1, action:1]= +11.1727
##
## Q[etat:2, action:0]= +8.2497
## Q[etat:2, action:1]= +8.8617 <-
## -----
## Iteration: 80
##
## Q[etat:0, action:0]= +8.2079
## Q[etat:0, action:1]= +8.4186 <-
##
## Q[etat:1, action:0]= +11.9139 <-
## Q[etat:1, action:1]= +11.1732
##
## Q[etat:2, action:0]= +8.2503
## Q[etat:2, action:1]= +8.8622 <-
## -----
## Iteration: 81
##
## Q[etat:0, action:0]= +8.2084
## Q[etat:0, action:1]= +8.4191 <-
##
## Q[etat:1, action:0]= +11.9143 <-
## Q[etat:1, action:1]= +11.1736
##
## Q[etat:2, action:0]= +8.2507
## Q[etat:2, action:1]= +8.8627 <-
## -----
## Iteration: 82
##
## Q[etat:0, action:0]= +8.2088
## Q[etat:0, action:1]= +8.4195 <-
##
## Q[etat:1, action:0]= +11.9148 <-
## Q[etat:1, action:1]= +11.1741
##
## Q[etat:2, action:0]= +8.2511
## Q[etat:2, action:1]= +8.8631 <-
## -----
## Iteration: 83
##
## Q[etat:0, action:0]= +8.2092
## Q[etat:0, action:1]= +8.4199 <-
##
## Q[etat:1, action:0]= +11.9151 <-
## Q[etat:1, action:1]= +11.1744
##
## Q[etat:2, action:0]= +8.2515
## Q[etat:2, action:1]= +8.8635 <-
## -----

```



```

## Iteration: 84
##
## Q[etat:0, action:0]= +8.2096
## Q[etat:0, action:1]= +8.4203 <-
##
## Q[etat:1, action:0]= +11.9155 <-
## Q[etat:1, action:1]= +11.1748
##
## Q[etat:2, action:0]= +8.2519
## Q[etat:2, action:1]= +8.8638 <-
## -----
## Iteration: 85
##
## Q[etat:0, action:0]= +8.2100
## Q[etat:0, action:1]= +8.4206 <-
##
## Q[etat:1, action:0]= +11.9158 <-
## Q[etat:1, action:1]= +11.1751
##
## Q[etat:2, action:0]= +8.2522
## Q[etat:2, action:1]= +8.8641 <-
## -----
## Iteration: 86
##
## Q[etat:0, action:0]= +8.2103
## Q[etat:0, action:1]= +8.4209 <-
##
## Q[etat:1, action:0]= +11.9161 <-
## Q[etat:1, action:1]= +11.1753
##
## Q[etat:2, action:0]= +8.2525
## Q[etat:2, action:1]= +8.8644 <-
## -----
## Iteration: 87
##
## Q[etat:0, action:0]= +8.2105
## Q[etat:0, action:1]= +8.4212 <-
##
## Q[etat:1, action:0]= +11.9164 <-
## Q[etat:1, action:1]= +11.1756
##
## Q[etat:2, action:0]= +8.2528
## Q[etat:2, action:1]= +8.8647 <-
## -----
## Iteration: 88
##
## Q[etat:0, action:0]= +8.2108
## Q[etat:0, action:1]= +8.4215 <-
##
## Q[etat:1, action:0]= +11.9166 <-
## Q[etat:1, action:1]= +11.1758
##
## Q[etat:2, action:0]= +8.2530
## Q[etat:2, action:1]= +8.8649 <-

```

```

## -----
## Iteration: 89
##
## Q[etat:0, action:0]= +8.2110
## Q[etat:0, action:1]= +8.4217 <-
##
## Q[etat:1, action:0]= +11.9168 <-
## Q[etat:1, action:1]= +11.1760
##
## Q[etat:2, action:0]= +8.2533
## Q[etat:2, action:1]= +8.8652 <-
## -----
## Iteration: 90
##
## Q[etat:0, action:0]= +8.2113
## Q[etat:0, action:1]= +8.4219 <-
##
## Q[etat:1, action:0]= +11.9170 <-
## Q[etat:1, action:1]= +11.1762
##
## Q[etat:2, action:0]= +8.2535
## Q[etat:2, action:1]= +8.8654 <-
## -----
## Iteration: 91
##
## Q[etat:0, action:0]= +8.2114
## Q[etat:0, action:1]= +8.4221 <-
##
## Q[etat:1, action:0]= +11.9172 <-
## Q[etat:1, action:1]= +11.1764
##
## Q[etat:2, action:0]= +8.2536
## Q[etat:2, action:1]= +8.8655 <-
## -----
## Iteration: 92
##
## Q[etat:0, action:0]= +8.2116
## Q[etat:0, action:1]= +8.4223 <-
##
## Q[etat:1, action:0]= +11.9174 <-
## Q[etat:1, action:1]= +11.1766
##
## Q[etat:2, action:0]= +8.2538
## Q[etat:2, action:1]= +8.8657 <-
## -----
## Iteration: 93
##
## Q[etat:0, action:0]= +8.2118
## Q[etat:0, action:1]= +8.4224 <-
##
## Q[etat:1, action:0]= +11.9175 <-
## Q[etat:1, action:1]= +11.1767
##
## Q[etat:2, action:0]= +8.2540

```

```

## Q[etat:2, action:1]= +8.8659 <-
## -----
## Iteration: 94
##
## Q[etat:0, action:0]= +8.2119
## Q[etat:0, action:1]= +8.4226 <-
##
## Q[etat:1, action:0]= +11.9177 <-
## Q[etat:1, action:1]= +11.1768
##
## Q[etat:2, action:0]= +8.2541
## Q[etat:2, action:1]= +8.8660 <-
## -----
## Iteration: 95
##
## Q[etat:0, action:0]= +8.2121
## Q[etat:0, action:1]= +8.4227 <-
##
## Q[etat:1, action:0]= +11.9178 <-
## Q[etat:1, action:1]= +11.1770
##
## Q[etat:2, action:0]= +8.2542
## Q[etat:2, action:1]= +8.8661 <-
## -----
## Iteration: 96
##
## Q[etat:0, action:0]= +8.2122
## Q[etat:0, action:1]= +8.4228 <-
##
## Q[etat:1, action:0]= +11.9179 <-
## Q[etat:1, action:1]= +11.1771
##
## Q[etat:2, action:0]= +8.2544
## Q[etat:2, action:1]= +8.8662 <-
## -----
## Iteration: 97
##
## Q[etat:0, action:0]= +8.2123
## Q[etat:0, action:1]= +8.4229 <-
##
## Q[etat:1, action:0]= +11.9180 <-
## Q[etat:1, action:1]= +11.1772
##
## Q[etat:2, action:0]= +8.2545
## Q[etat:2, action:1]= +8.8663 <-
## -----
## Iteration: 98
##
## Q[etat:0, action:0]= +8.2124
## Q[etat:0, action:1]= +8.4230 <-
##
## Q[etat:1, action:0]= +11.9181 <-
## Q[etat:1, action:1]= +11.1773
##

```

```

## Q[etat:2, action:0]= +8.2546
## Q[etat:2, action:1]= +8.8664 <-
## -----
## Iteration: 99
##
## Q[etat:0, action:0]= +8.2125
## Q[etat:0, action:1]= +8.4231 <-
##
## Q[etat:1, action:0]= +11.9182 <-
## Q[etat:1, action:1]= +11.1774
##
## Q[etat:2, action:0]= +8.2547
## Q[etat:2, action:1]= +8.8665 <-
## -----
## Iteration: 100
##
## Q[etat:0, action:0]= +8.2126
## Q[etat:0, action:1]= +8.4232 <-
##
## Q[etat:1, action:0]= +11.9183 <-
## Q[etat:1, action:1]= +11.1774
##
## Q[etat:2, action:0]= +8.2547
## Q[etat:2, action:1]= +8.8666 <-
## -----
## Iteration: 101
##
## Q[etat:0, action:0]= +8.2127
## Q[etat:0, action:1]= +8.4233 <-
##
## Q[etat:1, action:0]= +11.9184 <-
## Q[etat:1, action:1]= +11.1775
##
## Q[etat:2, action:0]= +8.2548
## Q[etat:2, action:1]= +8.8667 <-
## -----
## Iteration: 102
##
## Q[etat:0, action:0]= +8.2127
## Q[etat:0, action:1]= +8.4234 <-
##
## Q[etat:1, action:0]= +11.9184 <-
## Q[etat:1, action:1]= +11.1776
##
## Q[etat:2, action:0]= +8.2549
## Q[etat:2, action:1]= +8.8668 <-
## -----
## Iteration: 103
##
## Q[etat:0, action:0]= +8.2128
## Q[etat:0, action:1]= +8.4234 <-
##
## Q[etat:1, action:0]= +11.9185 <-
## Q[etat:1, action:1]= +11.1776

```

```

##
## Q[etat:2, action:0]= +8.2549
## Q[etat:2, action:1]= +8.8668 <-
## -----
## Iteration: 104
##
## Q[etat:0, action:0]= +8.2129
## Q[etat:0, action:1]= +8.4235 <-
##
## Q[etat:1, action:0]= +11.9186 <-
## Q[etat:1, action:1]= +11.1777
##
## Q[etat:2, action:0]= +8.2550
## Q[etat:2, action:1]= +8.8669 <-
## -----
## Iteration: 105
##
## Q[etat:0, action:0]= +8.2129
## Q[etat:0, action:1]= +8.4235 <-
##
## Q[etat:1, action:0]= +11.9186 <-
## Q[etat:1, action:1]= +11.1777
##
## Q[etat:2, action:0]= +8.2551
## Q[etat:2, action:1]= +8.8669 <-
## -----
## Iteration: 106
##
## Q[etat:0, action:0]= +8.2130
## Q[etat:0, action:1]= +8.4236 <-
##
## Q[etat:1, action:0]= +11.9187 <-
## Q[etat:1, action:1]= +11.1778
##
## Q[etat:2, action:0]= +8.2551
## Q[etat:2, action:1]= +8.8670 <-
## -----
## Iteration: 107
##
## Q[etat:0, action:0]= +8.2130
## Q[etat:0, action:1]= +8.4236 <-
##
## Q[etat:1, action:0]= +11.9187 <-
## Q[etat:1, action:1]= +11.1778
##
## Q[etat:2, action:0]= +8.2551
## Q[etat:2, action:1]= +8.8670 <-
## -----
## Iteration: 108
##
## Q[etat:0, action:0]= +8.2130
## Q[etat:0, action:1]= +8.4237 <-
##
## Q[etat:1, action:0]= +11.9187 <-

```

```

## Q[etat:1, action:1]= +11.1778
##
## Q[etat:2, action:0]= +8.2552
## Q[etat:2, action:1]= +8.8670 <-
## -----
## Iteration: 109
##
## Q[etat:0, action:0]= +8.2131
## Q[etat:0, action:1]= +8.4237 <-
##
## Q[etat:1, action:0]= +11.9188 <-
## Q[etat:1, action:1]= +11.1779
##
## Q[etat:2, action:0]= +8.2552
## Q[etat:2, action:1]= +8.8671 <-
## -----
## Iteration: 110
##
## Q[etat:0, action:0]= +8.2131
## Q[etat:0, action:1]= +8.4237 <-
##
## Q[etat:1, action:0]= +11.9188 <-
## Q[etat:1, action:1]= +11.1779
##
## Q[etat:2, action:0]= +8.2553
## Q[etat:2, action:1]= +8.8671 <-
## -----
## Iteration: 111
##
## Q[etat:0, action:0]= +8.2132
## Q[etat:0, action:1]= +8.4238 <-
##
## Q[etat:1, action:0]= +11.9188 <-
## Q[etat:1, action:1]= +11.1779
##
## Q[etat:2, action:0]= +8.2553
## Q[etat:2, action:1]= +8.8671 <-
## -----
## Iteration: 112
##
## Q[etat:0, action:0]= +8.2132
## Q[etat:0, action:1]= +8.4238 <-
##
## Q[etat:1, action:0]= +11.9189 <-
## Q[etat:1, action:1]= +11.1780
##
## Q[etat:2, action:0]= +8.2553
## Q[etat:2, action:1]= +8.8672 <-
## -----
## Iteration: 113
##
## Q[etat:0, action:0]= +8.2132
## Q[etat:0, action:1]= +8.4238 <-
##

```

```

## Q[etat:1, action:0]= +11.9189 <-
## Q[etat:1, action:1]= +11.1780
##
## Q[etat:2, action:0]= +8.2553
## Q[etat:2, action:1]= +8.8672 <-
## -----
## Iteration: 114
##
## Q[etat:0, action:0]= +8.2132
## Q[etat:0, action:1]= +8.4238 <-
##
## Q[etat:1, action:0]= +11.9189 <-
## Q[etat:1, action:1]= +11.1780
##
## Q[etat:2, action:0]= +8.2554
## Q[etat:2, action:1]= +8.8672 <-
## -----
## Iteration: 115
##
## Q[etat:0, action:0]= +8.2132
## Q[etat:0, action:1]= +8.4239 <-
##
## Q[etat:1, action:0]= +11.9189 <-
## Q[etat:1, action:1]= +11.1780
##
## Q[etat:2, action:0]= +8.2554
## Q[etat:2, action:1]= +8.8672 <-
## -----
## Iteration: 116
##
## Q[etat:0, action:0]= +8.2133
## Q[etat:0, action:1]= +8.4239 <-
##
## Q[etat:1, action:0]= +11.9190 <-
## Q[etat:1, action:1]= +11.1780
##
## Q[etat:2, action:0]= +8.2554
## Q[etat:2, action:1]= +8.8673 <-
## -----
## Iteration: 117
##
## Q[etat:0, action:0]= +8.2133
## Q[etat:0, action:1]= +8.4239 <-
##
## Q[etat:1, action:0]= +11.9190 <-
## Q[etat:1, action:1]= +11.1781
##
## Q[etat:2, action:0]= +8.2554
## Q[etat:2, action:1]= +8.8673 <-
## -----
## Iteration: 118
##
## Q[etat:0, action:0]= +8.2133
## Q[etat:0, action:1]= +8.4239 <-

```

```

##
## Q[etat:1, action:0]= +11.9190 <-
## Q[etat:1, action:1]= +11.1781
##
## Q[etat:2, action:0]= +8.2554
## Q[etat:2, action:1]= +8.8673 <-
## -----
## Iteration: 119
##
## Q[etat:0, action:0]= +8.2133
## Q[etat:0, action:1]= +8.4239 <-
##
## Q[etat:1, action:0]= +11.9190 <-
## Q[etat:1, action:1]= +11.1781
##
## Q[etat:2, action:0]= +8.2554
## Q[etat:2, action:1]= +8.8673 <-
## -----
## Iteration: 120
##
## Q[etat:0, action:0]= +8.2133
## Q[etat:0, action:1]= +8.4239 <-
##
## Q[etat:1, action:0]= +11.9190 <-
## Q[etat:1, action:1]= +11.1781
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8673 <-
## -----
## Iteration: 121
##
## Q[etat:0, action:0]= +8.2133
## Q[etat:0, action:1]= +8.4239 <-
##
## Q[etat:1, action:0]= +11.9190 <-
## Q[etat:1, action:1]= +11.1781
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8673 <-
## -----
## Iteration: 122
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9190 <-
## Q[etat:1, action:1]= +11.1781
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8673 <-
## -----
## Iteration: 123
##
## Q[etat:0, action:0]= +8.2134

```



```

## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9190 <-
## Q[etat:1, action:1]= +11.1781
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8673 <-
## -----
## Iteration: 124
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1781
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 125
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1781
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 126
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 127
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 128
##

```

```

## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 129
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 130
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 131
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 132
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2555
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 133

```

```

##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 134
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 135
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 136
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 137
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----

```

```

## Iteration: 138
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 139
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4240 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 140
##
## Q[etat:0, action:0]= +8.2134
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 141
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 142
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-

```

```

## -----
## Iteration: 143
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 144
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 145
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 146
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 147
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556

```

```

## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 148
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 149
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 150
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 151
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 152
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##

```

```

## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 153
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 154
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 155
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 156
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 157
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782

```

```

##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 158
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 159
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 160
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 161
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 162
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-

```



```

## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 163
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 164
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 165
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 166
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 167
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##

```

```

## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 168
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 169
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 170
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 171
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 172
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-

```

```

##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 173
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 174
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 175
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 176
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 177
##
## Q[etat:0, action:0]= +8.2135

```

```

## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 178
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 179
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 180
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 181
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 182
##

```

```

## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 183
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 184
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 185
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 186
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 187

```

```

##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 188
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 189
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 190
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 191
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----

```

```

## Iteration: 192
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 193
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 194
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 195
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 196
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-

```

```

## -----
## Iteration: 197
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 198
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-
## -----
## Iteration: 199
##
## Q[etat:0, action:0]= +8.2135
## Q[etat:0, action:1]= +8.4241 <-
##
## Q[etat:1, action:0]= +11.9191 <-
## Q[etat:1, action:1]= +11.1782
##
## Q[etat:2, action:0]= +8.2556
## Q[etat:2, action:1]= +8.8674 <-

print("-----")

## -----

```

On constate bien que notre algorithme converge. À partir de l'itération 150 les valeurs de bougent plus.

Notons tout de même que le plus souvent on a pas un modèle aussi simple.

C'est à dire qu'on a aucune connaissances sur les probabilités, et les récompenses. Bien évidemment **Bellman** a pensé à tout, et a donc introduit **Apprentissage par différence temporelle**.

$$V_{k+1} = (1 - \alpha)V_k(s) + \alpha(r - \gamma * V_k(s')).$$

$$Q_{k+1}(s, a) = (1 - \alpha)Q_k(s, a) + \alpha(r - \gamma * \max_{a'} Q_k(s', a')).$$

α est appelé le coefficient d'apprentissage, et est très souvent proche de 0.

Réseaux de Convolution

N'ayant pas beaucoup d'images pour effectuer un entraînement, une validation du modèle et après un test, nous avons décidé d'enrichir notre jeu de données en créant de nouvelles images. Sous R, il existe une fonction permettant de le faire c'est la fonction `image_data_generator` dans le package `imager`.

Création données entraînement, de test et de validation

```
library(tensorflow)

library(reticulate)
library(keras)
library(dplyr)
library(stringr)
library(caret)
library(imager)

use_python("~/anaconda3/envs/tf_image/bin/python")

train_gen <- image_data_generator(rescale = 1/255,
                                horizontal_flip = T,
                                vertical_flip = T,
                                rotation_range = 45,
                                zoom_range = 0.25,
                                validation_split = 0.2)

target_size= c(64,64)
batch_size= 32

train_image <- flow_images_from_directory(directory = "/Users/david/Desktop/daviddtuto1",
                                          target_size = target_size,
                                          color_mode="rgb",
                                          batch_size = batch_size,
                                          seed=123,
                                          subset="training",
                                          generator = train_gen
                                          )

val_image <- flow_images_from_directory(directory = "/Users/david/Desktop/daviddtuto1",
                                       target_size = target_size,
                                       color_mode="rgb",
                                       batch_size = batch_size,
                                       seed=123,
                                       subset="validation",
                                       generator = train_gen
                                       )
```

Mise en place du réseau de convolution

```
train_samples <- train_image$n
valid_samples <- val_image$n
output_n <- n_distinct(train_image$classes)

tensorflow::tf$random$set_seed(123)

model <- keras_model_sequential(name = "simple_model") %>%

  # Convolution Layer
  layer_conv_2d(filters = 16,
               kernel_size = c(3,3),
               padding = "same",
               activation = "relu",
               input_shape = c(target_size, 3)
  ) %>%

  # Max Pooling Layer
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  # Flattening Layer
  layer_flatten() %>%

  # Dense Layer
  layer_dense(units = 16,
              activation = "relu") %>%

  # Output Layer
  layer_dense(units = output_n,
              activation = "softmax",
              name = "Output")

model %>%
  keras::compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_adam(learning_rate = 0.01),
    metrics = "accuracy"
  )
```

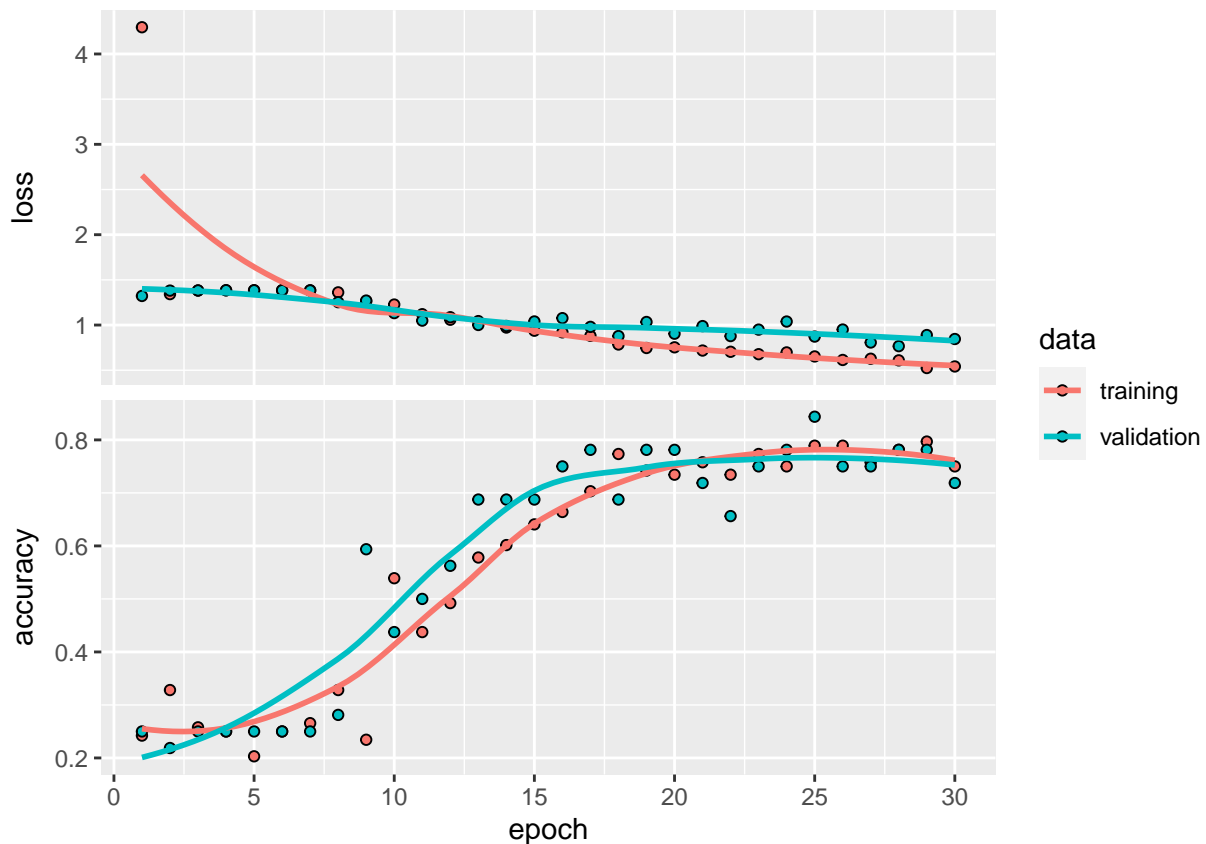
Graphe du modèle— Entraînement et validation

```
history <- model %>%  
  fit(  
    # training data  
    train_image,  
  
    # training epochs  
    steps_per_epoch = as.integer(train_samples / batch_size),  
    epochs = 30,  
  
    # validation data  
    validation_data = val_image,  
    validation_steps = as.integer(valid_samples / batch_size)  
  )
```

Historique d'entraînement et de validation

```
plot(history)
```

```
## 'geom_smooth()' using formula 'y ~ x'
```



Création de données test

```
val_data <- data.frame(file_name = paste0("/Users/david/Desktop/davidtuto1/", val_image$filenames)) %>%  
  mutate(class = str_extract(file_name, "pikachu|rondoudou|Carapuce|Salameche"))  
  
head(val_data, 10)
```

```
##           file_name      class  
## 1 /Users/david/Desktop/davidtuto1/Carapuce/Carapuce_01.jpeg Carapuce  
## 2 /Users/david/Desktop/davidtuto1/Carapuce/Carapuce_010.jpeg Carapuce  
## 3 /Users/david/Desktop/davidtuto1/Carapuce/Carapuce_011.jpeg Carapuce  
## 4 /Users/david/Desktop/davidtuto1/Carapuce/Carapuce_012.jpeg Carapuce  
## 5 /Users/david/Desktop/davidtuto1/Carapuce/Carapuce_013.jpeg Carapuce  
## 6 /Users/david/Desktop/davidtuto1/Carapuce/Carapuce_014.jpeg Carapuce  
## 7 /Users/david/Desktop/davidtuto1/Carapuce/Carapuce_015.jpeg Carapuce
```

```
## 8    /Users/david/Desktop/daviddtuto1/Carapuce/Carapuce_016.jpeg Carapuce
## 9    /Users/david/Desktop/daviddtuto1/Salameche/Salameche_01.jpeg Salameche
## 10   /Users/david/Desktop/daviddtuto1/Salameche/Salameche_010.jpeg Salameche
```

```
#### Test
image_prep <- function(x) {
  arrays <- lapply(x, function(path) {
    img <- image_load(path, target_size = target_size,
                      grayscale = F # Set FALSE if image is RGB
    )

    x <- image_to_array(img)
    x <- array_reshape(x, c(1, dim(x)))
    x <- x/255 # rescale image pixel
  })
  do.call(abind::abind, c(arrays, list(along = 1)))
}

test_x <- image_prep(val_data$file_name)
```

Evaluation sur données Test

```
pred_test <- model %>% predict(test_x) %>% k_argmax()
head(pred_test, 20)
```

```
## tf.Tensor([2 0 0 0 0 0 0 0 2 1 1 2 1 2 1 2 2 2 2], shape=(20,), dtype=int64)
```

Décodage de la prédiction

```
decode <- function(x){
  case_when(x == 2 ~ "pikachu",
            x == 1 ~ "Salameche",
            x == 0 ~ "Carapuce",
            x == 3 ~ "rondoudou"
  )
}

pred_test <- sapply(as.array(pred_test), decode)
```

Matrice de Confusion

```
confusionMatrix(as.factor(pred_test),
                 as.factor(val_data$class)
)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction  Carapuce pikachu rondoudou Salameche
##   Carapuce      7      0      0      0
##   pikachu       1      8      0      4
##   rondoudou     0      0      6      0
##   Salameche     0      0      2      4
##
## Overall Statistics
##
##              Accuracy : 0.7812
##              95% CI : (0.6003, 0.9072)
##   No Information Rate : 0.25
##   P-Value [Acc > NIR] : 4.377e-10
##
##              Kappa : 0.7083
##
##   McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##              Class: Carapuce Class: pikachu Class: rondoudou
## Sensitivity              0.8750              1.0000              0.7500
## Specificity              1.0000              0.7917              1.0000
## Pos Pred Value           1.0000              0.6154              1.0000
## Neg Pred Value           0.9600              1.0000              0.9231
## Prevalence               0.2500              0.2500              0.2500
## Detection Rate           0.2188              0.2500              0.1875
## Detection Prevalence     0.2188              0.4062              0.1875
## Balanced Accuracy        0.9375              0.8958              0.8750
##
##              Class: Salameche
## Sensitivity              0.5000
## Specificity              0.9167
## Pos Pred Value           0.6667
## Neg Pred Value           0.8462
## Prevalence               0.2500
## Detection Rate           0.1250
## Detection Prevalence     0.1875
## Balanced Accuracy        0.7083
```

Conclusion

Ayant travaillé sur des les mêmes données mais sur différents algorithmes je peux clairement dire que les **réseaux de convolution** sont plus appropriés . De plus, on peut constater qu'une méthode simple comme les **Supports Vectors Machines** a un pus gros taux de réussite qu'un perceptron multicouches. Cela, nous montre bien que très souvent, il ne sert à rien de faire des algorithmes complexes comme **les réseaux neuronaux** alors qu'une méthode simple peut tout autant avoir un bon accuracy ou encore mieux, être tout simplement meilleur.

Source

Apprentissage par renforcement: <https://www.youtube.com/watch?v=Rgxs8lfoG4I&t=762s>

Apprentissage par renforcement : <https://datascientest.com/q-learning-le-machine-learning-avec-apprentissage-par-renforcement>

Carte de Kohonen: http://www.xavierdupre.fr/app/mlstatpy/helpsphinx/c_clus/kohonen.html

Réseaux de Convolutions: https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif