

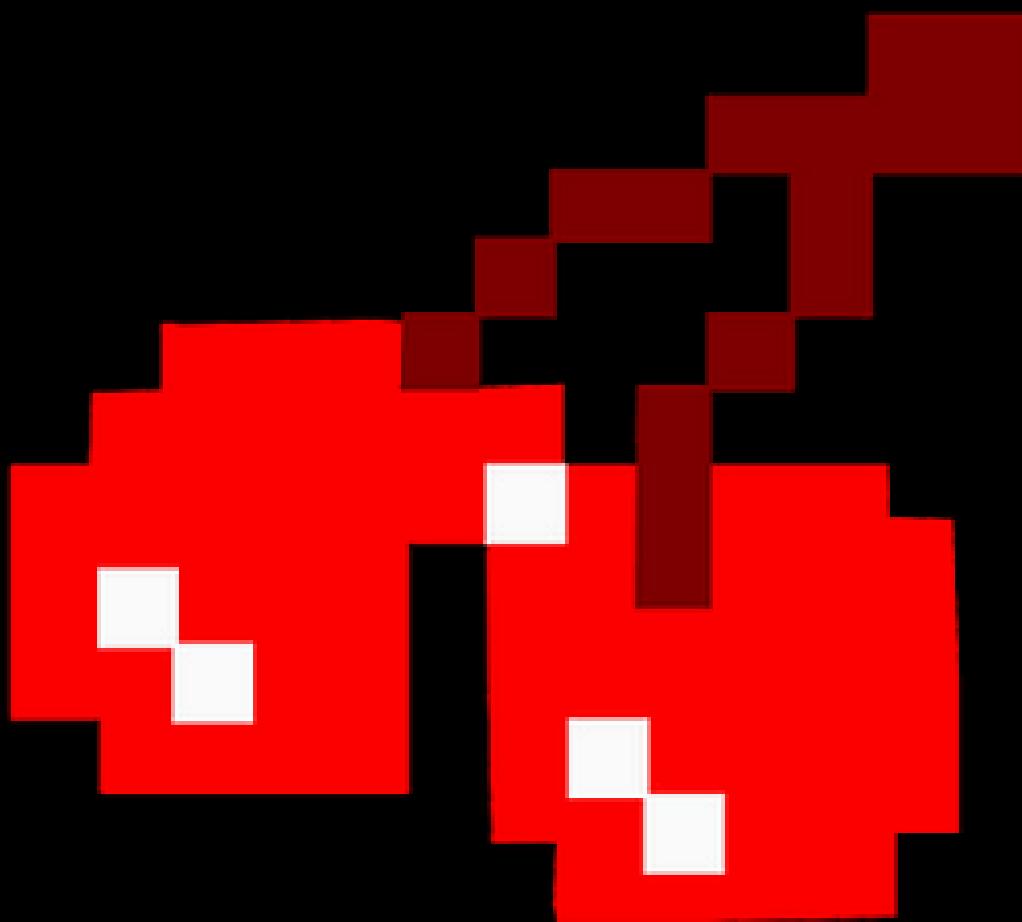
# Modelos Supervisados y por Refuerzo Aplicados a Pac-Man

Por: Marhia José Granada Restrepo

# PROBLEMA

¿Es mejor enseñarle a una máquina qué hacer o dejar que aprenda?

Caso de estudio: Pac-Man, un entorno donde el agente debe tomar decisiones óptimas en tiempo real.



# A. SUPERVISADO

- Concepto
- Algoritmo
- Entrenamiento



# A. SUPERVISADO



```
# Train model
X = clean_df.drop(columns=["Action"])
y = clean_df["Action"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

clf = RandomForestClassifier(max_depth=9, random_state=0)
clf.fit(X_train, y_train)

# Save model
with open('ml/trained_model.pkl', 'wb') as f:
    pickle.dump(clf, f)

print("Model trained!!")
```

```
def getAction(self, state):
    # Load model on first call
    if not self._model_loaded:
        self._load_model()

    legal = state.getLegalPacmanActions()

    snapshot = self.extract_snapshot(state)
    row = self.get_row(snapshot)

    action = self.clf.predict(row)[0]
    if action not in legal:
        return Directions.STOP
    return action
```

# A. POR REFORZO

- Concepto
- Funcionamiento

$$Q(s, a; w) = \sum_{i=1}^n f_i(s, a)w_i$$

```
def getQValue(self, state, action):
    #Q(s,a) = weights • features
    features = self.get_features(state, action)
    return np.dot(self.weights, features)
```



# A. POR REFORZO



```
# :D
# Survival
reward += 2

# 2. Food
score_diff = next_state.getScore() - state.getScore()
if score_diff == 10:
    reward += 15 #Food Reward

# 3. Ghost eating
elif score_diff == 200:
    reward += 250

# Win
if next_state.isWin():
    return 500
```

-Recompensas:  
+15 por comer un pellet, +2 por sobrevivir, +250 por comer un fantasma, +500 por ganar, +3 por estar cerca de comida, -10 por estar frente a un fantasma, -100 por ser atrapado

# A. POR REFORZO

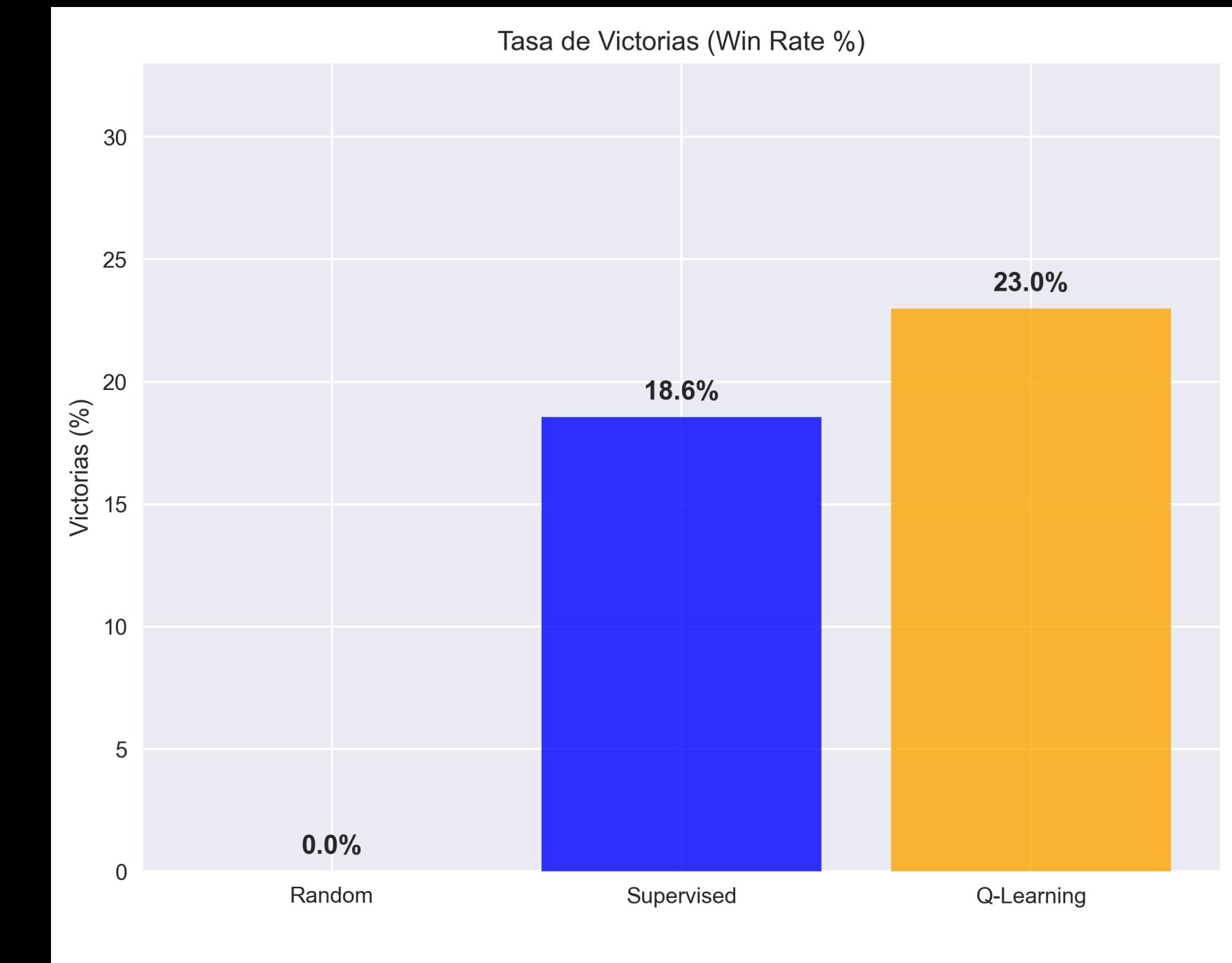
```
def getAction(self, state):
    legal = state.getLegalPacmanActions()
    if not legal:
        return Directions.STOP

    if np.random.rand() < self.epsilon:
        return np.random.choice(legal)

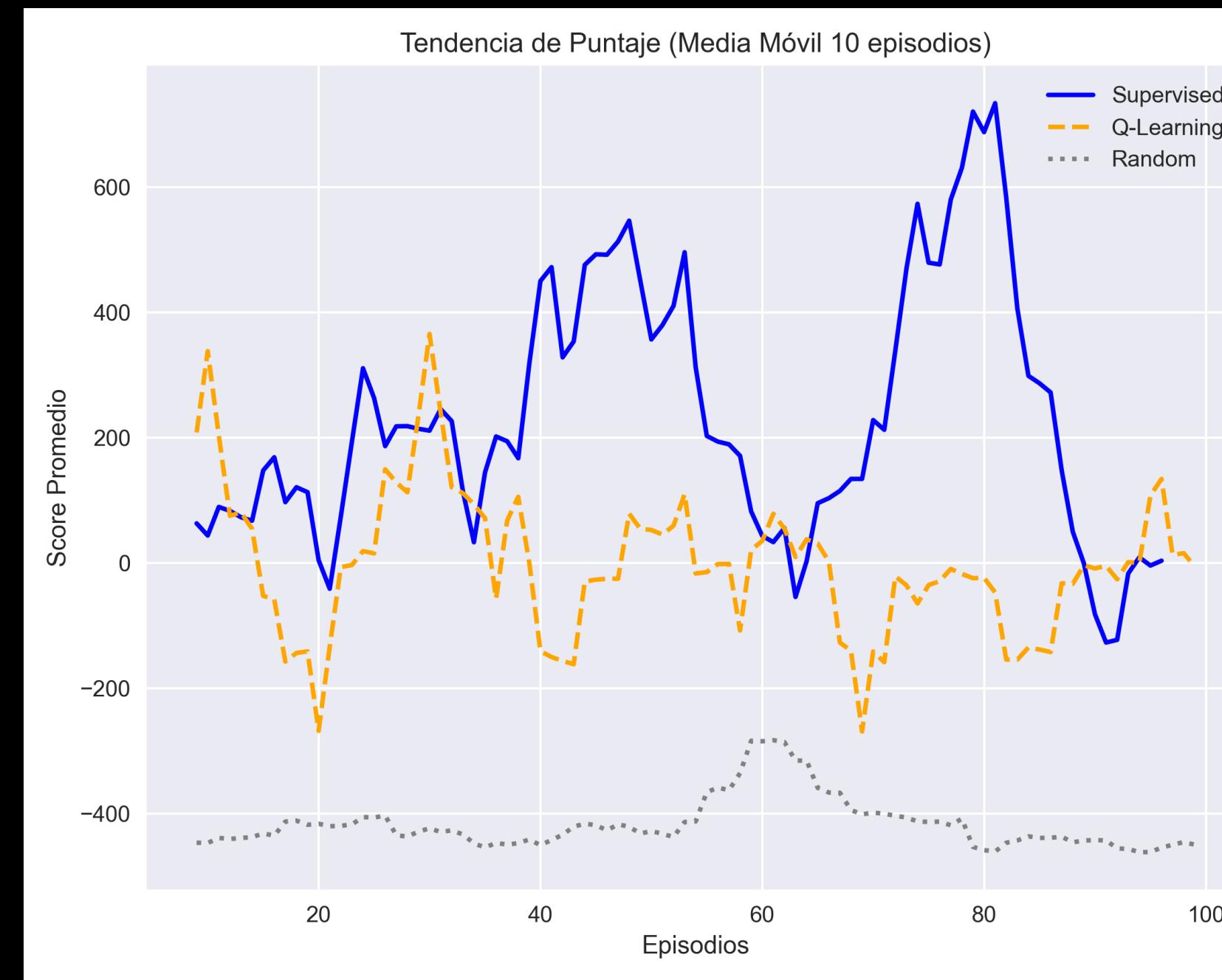
    # Otherwise, choose best action
    q_values = [self.getQValue(state, a) for a in legal]
    max_q = max(q_values)
    best_actions = [a for a, q in zip(legal, q_values) if q == max_q]
    return np.random.choice(best_actions)
```



# RESULTADOS



# RESULTADOS

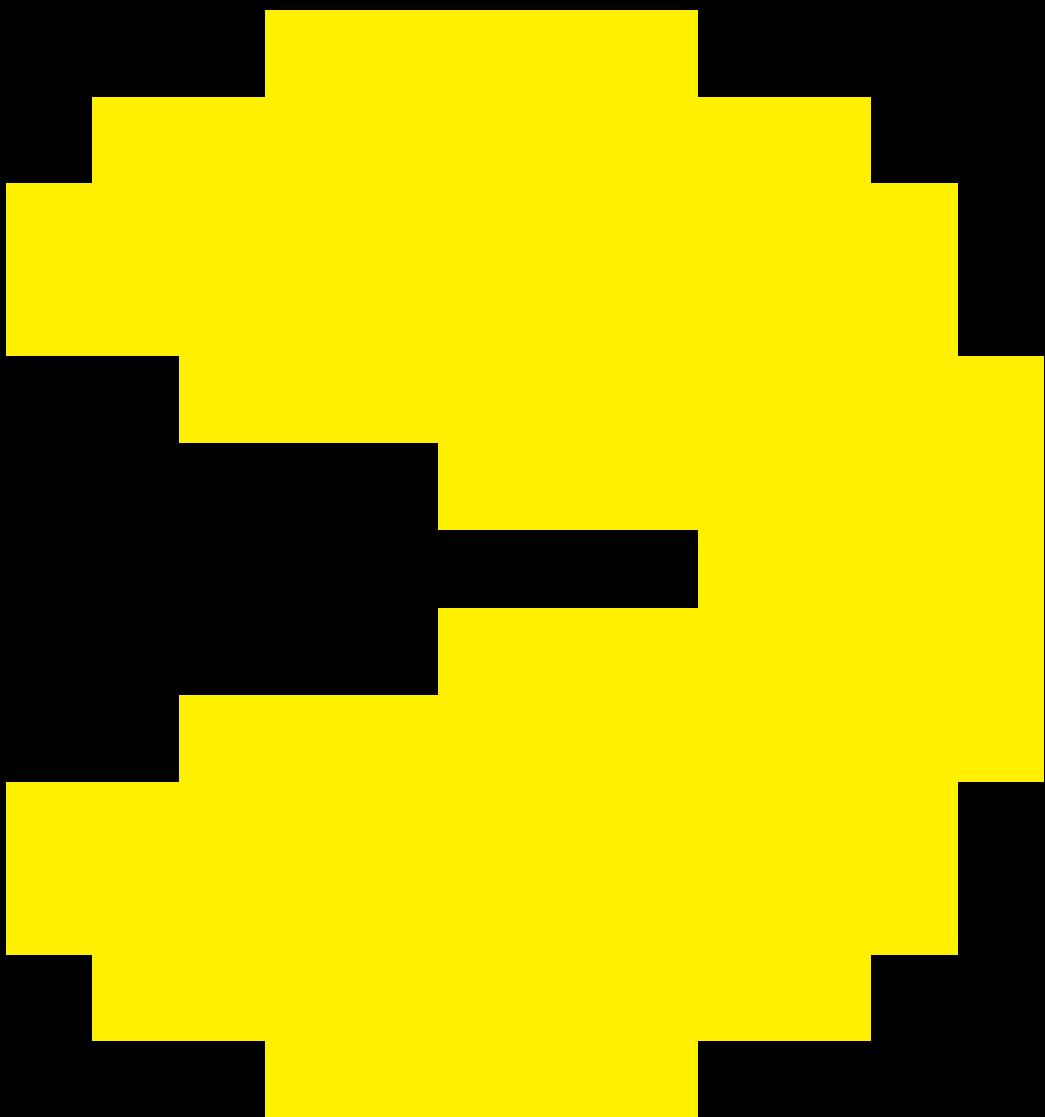


# ANÁLISIS

- El modelo supervisado sigue un camino fijo hasta cierto punto.
- En el aprendizaje por refuerzo se puede ver el modelo mejorar durante un juego más no durante las distintas sesiones.



# CONCLUSIONES



- Aprendizaje supervisado empieza major
- Q Learning mejora a largo plazo
- Se puede mejorar el algoritmo de Q learning añadiendo más y mejores features.
- Se necesita un mejor experto

# REFERENCIAS

Watkins & Dayan (1992) – Q-Learning.

Stanford CS229 (2017)  
– Pac-Man RL Agent.

UC Berkeley CS188 (2024)  
– Environment setup.

**Reinforcement Learning in Pacman**

Abeynaya Gnanasekaran, Jordi Feliu Faba, Jing An  
SUNet IDs: abeynaya, jfeliu, jingan

**I. ABSTRACT**

We apply various reinforcement learning methods on the classical game Pacman; we study and compare Q-learning, approximate Q-learning and Deep Q-learning based on the total rewards and win-rate. While Q-learning has been proved to be quite effective on smallGrid, it becomes inefficient to find the optimal policy in large grid-layouts. In approximate

**• Deep Recursive Q-network [4] (DQRN) which is a combination of a Long Short Term Memory (LSTM) and a Deep Q-Network, better handles the loss of information than does DQN.**

**• Dual Q-network [5], where different Q-values are used to select and to evaluate an action by using two different DQN with different weights. This helps to avoid overestimation of Q-values.**

Machine Learning, 8, 279–292 (1992)  
© 1992 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

**Technical Note**  
**Q-Learning**

CHRISTOPHER J.C.H. WATKINS  
25b Framfield Road, Highbury, London N5 1UU, England

PETER DAYAN  
Centre for Cognitive Science, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9EH, Scotland

**Abstract.** Q-learning (Watkins, 1989) is a simple way for agents to learn how to act optimally in controlled Markovian domains. It amounts to an incremental method for dynamic programming which imposes limited computational demands. It works by successively improving its evaluations of the quality of particular actions at particular states. This paper presents and proves in detail a convergence theorem for Q-learning based on that outlined in Watkins (1989). We show that Q-learning converges to the optimum action-values with probability 1 so long as all actions are repeatedly sampled in all states and the action-values are represented discretely. We also sketch extensions to the cases of non-discounted, but absorbing, Markov environments, and where many Q values can be changed each iteration, rather than just one.

**Keywords.** Q-learning, reinforcement learning, temporal differences, asynchronous dynamic programming

**GRACIAS**

**PT 2**