

# Deep Learning Techniques for Music Generation Reinforcement (7)

---

Jean-Pierre Briot

`Jean-Pierre.Briot@lip6.fr`

Laboratoire d'Informatique de Paris 6 (LIP6)  
Sorbonne Université – CNRS



Programa de Pós-Graduação em Informática (PPGI)  
UNIRIO



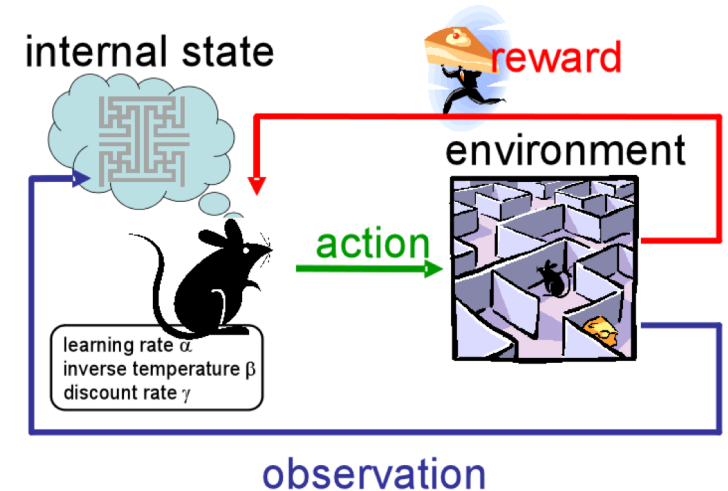
# Reinforcement Learning

---



# Reinforcement Learning [Sutton, 1984]

- Very Different Approach and Model (from Data Learning)
- Inspired from Behaviorist Psychology
- Based on Decisions/Actions
- (and States and Rewards)



[Figure from Cyber Rodent Project]

- Not Based on Dataset
- Not Supervised (No Labels/No Examples of Best Actions)
- Feedback (Delayed Rewards)
- Learning // Action (Trial and Error)
- Incremental



*Tell me and I forget,  
Teach me and I may remember,  
Involve me and I Learn.*

*- Benjamin Franklin*

The only stupid question is the one you never ask [Sutton]

# Reinforcement Learning [Sutton, 1984]

---

- Exploration vs Exploitation Dilemma
- Temporal/Delayed Credit Assignment Issue
- Formal Framework: Markov Decision Process (MDP)
- Sequential Decision Making
- Objective: Learn Optimal Policy (Best Action Decision for each State) to Maximize Expected Future Return/Gain (Accumulated Rewards)
- = Minimize Regret (Difference between expected Gain and optimal Policy's Gain)

# Melody Generation

---

## Example of Model

- State: Melody generated so far (Succession of notes)



Composition so far



- Action: Generation of next note

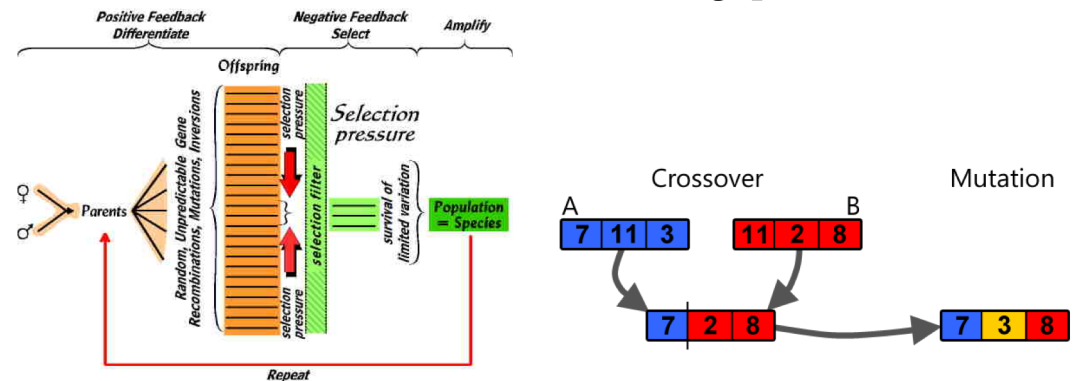
State

Action

- Feedback: Listener, or Musical Theory Rules, or/and...

# Evolutionary Algorithms, Genetic Algorithms and Programming

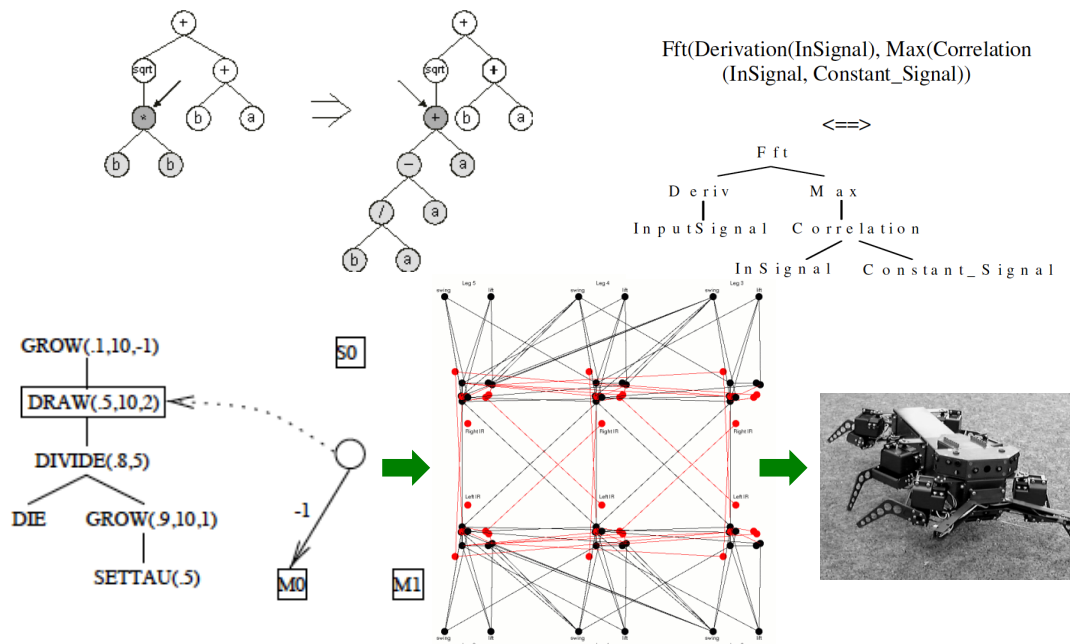
- Could be Considered as an Approach for Reinforcement Learning [Pack Kaebling et al. 1996]
- Search in the Space of Behaviors
- Selection based on Fitness
- Fitness: Global/Final Reward
- **Off-Line Learning** (Genotype -> Phenotype Generation)



- Evolutionary Algorithms
- Genetic Algorithms [Holland 1975]

- Genetic Programming [Koza 1990]
  - Phenotype (Tree structure) = Genotype

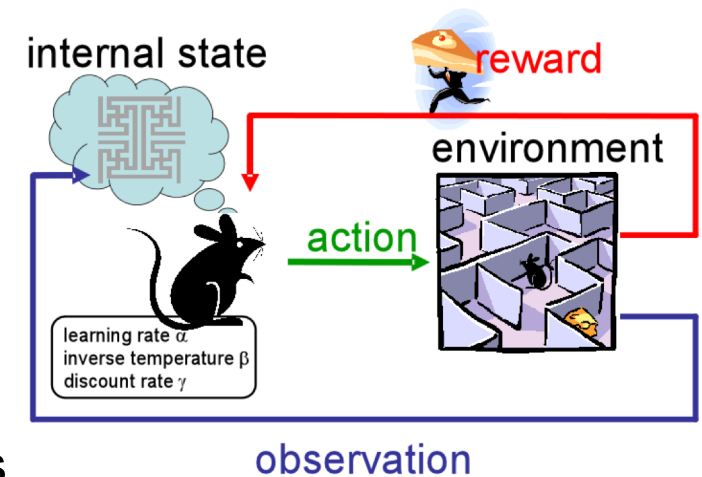
- Morphogenetic Programming [Meyer et al. 1995]



# Reinforcement Learning (RL)/MDP Basics [Silver 2015]

(at each step/time  $t$ )

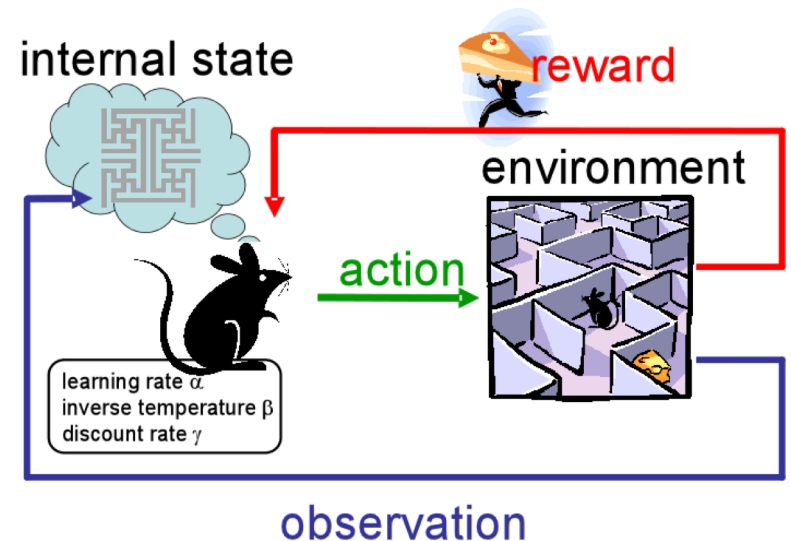
- Observation  $o_t$  of the Environment
  - Action  $a_t$  by the Agent
  - Reward  $r_t$  from the Environment  
positive or negative
- 
- History: Sequence of observations, actions, rewards
  - $H_t = o_1, a_1, r_1, o_2, a_2, r_2, \dots, o_t, a_t, r_t$
  - What happens next depends on this history
    - Decision of the agent
    - Observation of the environment
    - Reward by the environment
  - Full history is too huge
  - State: summary (what matters) of the history  $s_t = f(H_t)$



# Reinforcement Learning (RL)/MDP Basics [Silver 2015]

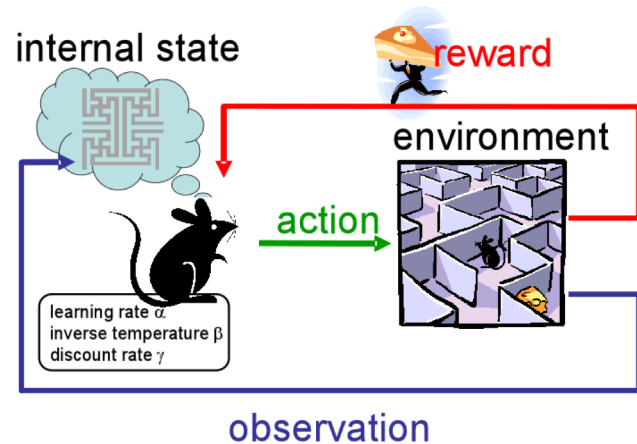
## Three Models of State [Silver 2015]:

- Environment State
  - Environment private representation
  - Not usually visible to the agent nor completely relevant
- Agent State
  - Agent internal representation
- Information State (aka Markov State)
  - Contains useful information from the history
- Markov property:  $P[s_{t+1} | s_t] = P[s_{t+1} | s_1, \dots, s_t]$ 
  - Future is independent of the past, given the present = History does not matter
  - State is sufficient statistics/distribution of the future
  - By definition, Environment State is Markov
- Fully or Partially Observable Environment
  - Full: Markov Decision Process (MDP) (*Environment State = Agent State = Markov State*)
  - Partial: Partially Observable Markov Decision Process (POMDP)
    - » Ex. of Representations: Beliefs of Environment, Recurrent Neural Networks...



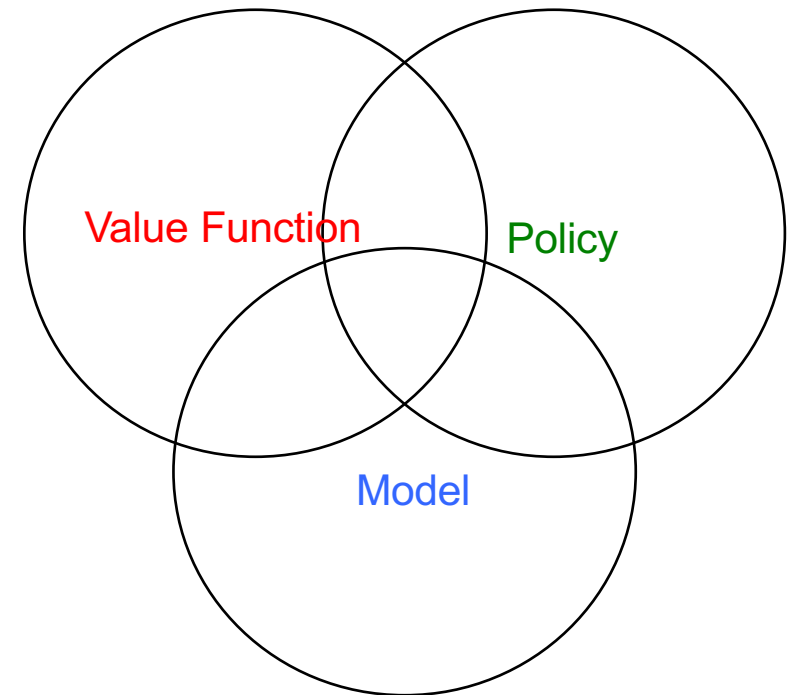
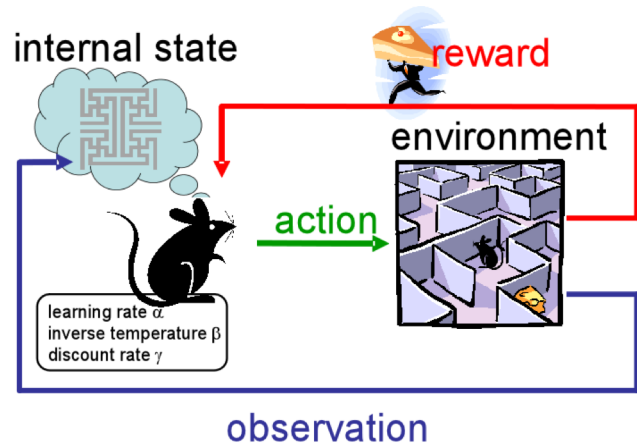
# Reinforcement Learning First Ad-Hoc/Naive Approaches

---



- Greedy strategy
  - Choose the action with the highest estimated return
  - Limit: Exploitation without Exploration
- Randomized
  - Limit: Exploration without Exploitation
- Mix:  $\epsilon$ -Greedy
  - $\epsilon$  probability to choose a random action, otherwise greedy
    - »  $\epsilon$  constant
    - » or  $\epsilon$  decreases in time from 1 (completely random) until a plateau
      - analog to simulated annealing

# Reinforcement Learning Components [Silver 2015]



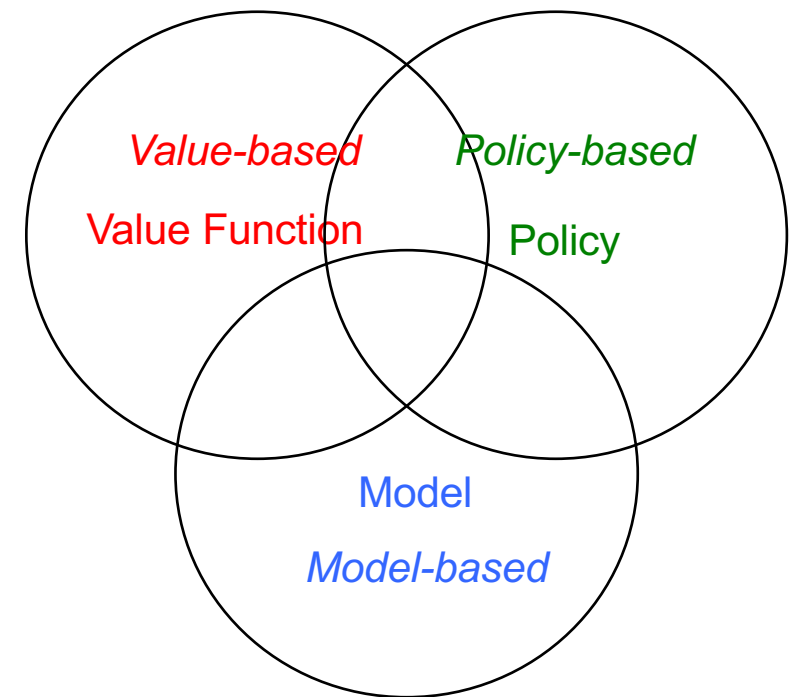
Three main components for RL [Silver 2015]:

- **Policy**
  - Agent behavior
  - $\pi(s) = a$  Function that, given a state, selects an action  $a$
- **Value Function**
  - Value of the state Expected return
- **Model**
  - Representation of the environment

# Main Approaches

Three main approaches for RL [Silver 2015]:

- **Policy**-based
- **Value**-based
- **Model**-based
- **Policy**-based
  - Search directly for Optimal Policy  $\pi^*$
- **Value**-based
  - Estimate the Optimal Value  $Q^*(s, a)$
  - Then choose Action with Highest Value function  $Q$ 
    - »  $\pi(s) = \operatorname{argmax}_a Q(s, a)$
- **Model**-based
  - Learn (estimate) a Transition Model of the Environment  $E$ 
    - »  $T(s, a) = s'$
    - »  $R(s, a) = r$
  - Plan Actions (e.g., by Lookahead) using the Model
- **Mixed**
  - Concurrent/Cooperative/Mutual Search/Approximations/Iterations



# Value Function(s)

- State Value Function

- Value of the **state**
- Expected return
- $V^\pi(s_t) = E^\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots]$
- **Discount factor**  $\gamma \in [0, 1]$  (Infinite Horizon Discounted Model)
  - » Uncertainty about the future (Life Expectancy + **Stochastic Environment**)
  - » Boundary of  $\infty$  (ex: avoids infinite returns from cycles)
  - » Biological (more appetite for immediate reward :) 
  - » Mathematically tractable
  - »  $\gamma = 0$  : short-sighted

- Action Value Function

- Value of the **state** and **action** pair
- $Q^\pi(s, a)$
- $V^\pi(s) = Q^\pi(s, \pi(s))$

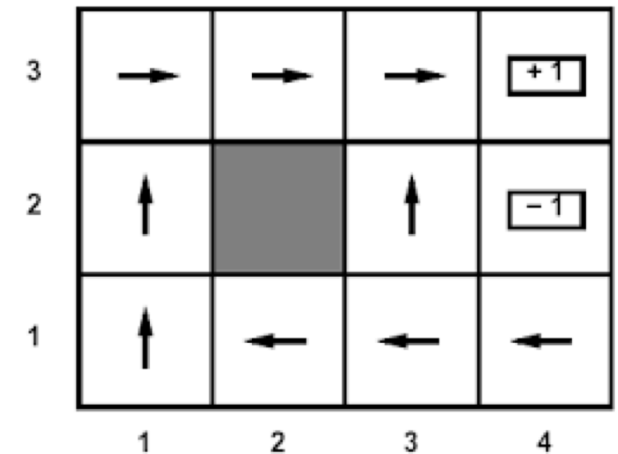
- Bellman Equation [Bellman 1957]

- value = **instant reward** + **discounted value of next state**
- $V^\pi(s_t) = E^\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots] = E^\pi[r_t] + \gamma V^\pi(s_{t+1})$
- $Q^\pi(s_t, a_t) = E^\pi[r_t] + \gamma Q^\pi(s_{t+1}, a_{t+1})$

# Policy-based and Value-based Approaches

- **Policy-based**

- Search directly for Optimal Policy  $\pi^*$
- On-Policy learning [Silver 2015]: Learn policy that is currently being followed (acted)
- Iterative Methods
  - » Monte-Carlo
    - Replace expected return with mean return (mean of samples returns)
  - » TD (Temporal Difference) [Sutton 1988]
    - Difference between estimation of the return *before* action and *after* action
    - On-line learning
    - TD(0)
    - TD( $\lambda$ ) (updates also states already visited  $\lambda$  times)

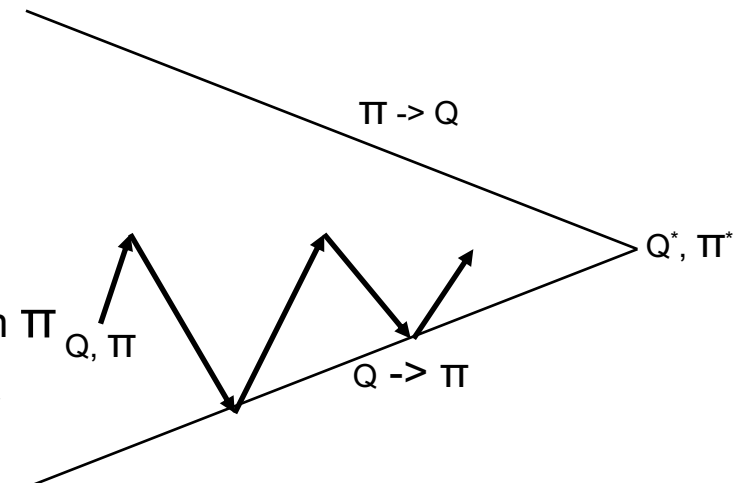


- **Value-based**

- Estimate the Optimal Value  $Q^*(s, a)$
- Then choose Action with Highest Value function  $Q$
- $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$

- **Mix //**

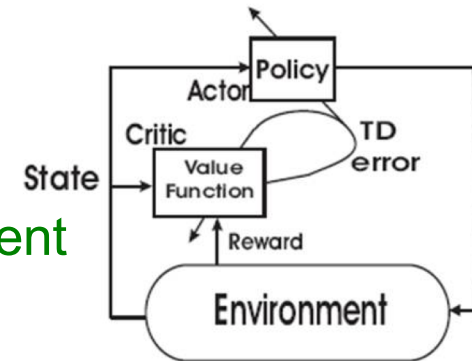
- **Iterative Policy evaluation:** TD or SARSA to Estimate  $Q$  from  $\pi$
- **Policy improvement:** Select  $\pi$  via  $\epsilon$ -greedy selection from  $Q$
- Iterate  $\pi^* \longleftrightarrow Q^*$



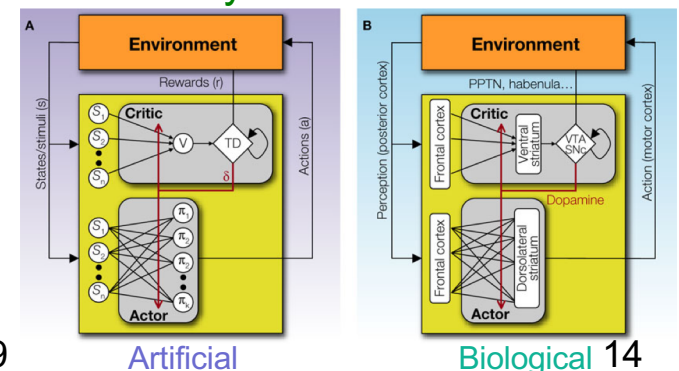
# Actor-Critic [Barto et al. 1983]

- Actor-Critic approach combines
  - Policy-based
  - Value-based
 > Actor Critic-based

- Similar to Iterative Policy evaluation // Policy improvement
- Actor acts and learns Policy
  - Uses a RL Component
  - Tries to Maximize the *Heuristic Value* of the Return (Value), computed by the Critic
- Critic learns Returns (Value) in order to Evaluate Policy
  - Uses Temporal Difference (TD(0) Algorithm [Sutton 1988])
  - TD = Difference between estimation of the Return (Value) before Action and after Action
  - Learns Mapping from States to Expected Returns (Values), given the Policy of the Actor
  - Communicates the Updated Expected Value to the Actor



- Run in //
- Co/Mutual-Improvement
- Recent (Partial) Biological Corroboration



[Tomasik 2012]

Biological 14

# Off-Policy Learning and Q-Learning

---

- Off-Policy learning [Silver 2015]
  - Idea: Learn from observing self history (off-line) or from other agents
  - Advantage: Learn about a policy (ex: optimal) while following a policy (ex: exploratory)
  - Estimate the expectation (value) of a different distribution
- Q-Learning [Watkins 1989]
- Analog to TD //  $\epsilon$ -greedy & Actor Critic but Integrates/Unifies them
  - Estimate Q and use it to define Policy
- Q-Table(s, a)
- Update Rule:
  - $Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$   
 $Q^*(s, a) = r + \gamma \max_{a'} Q(s', a')$  *Bellman equation*
- Exploration insensitive
  - The Exploration vs Exploitation Issue will not affect Convergence

# Q-Learning Algorithm

---

```
initialize Q table(#states, #actions) arbitrarily;  
observe initial state s;  
repeat  
    select and carry out action a;  
    observe reward r and new state s';  
     $Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a));$            update rule  
    s := s';  
until terminated
```

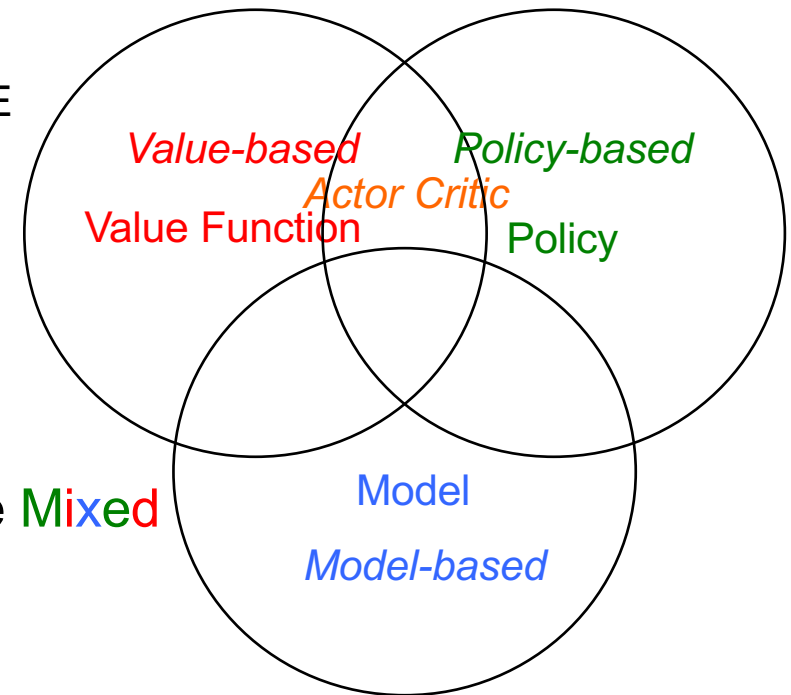
# Model-Based Reinforcement Learning

- **Model-based**

- Learn (estimate) a Transition Model of the Environment  $E$ 
  - »  $T(s, a) = s'$  or/and  $R(a, s) = r$
- (in //) Use the Model
  - » Plan Actions (e.g., by Lookahead) using the Model
  - » Or Adjust the Policy: Dyna [Sutton 1990]
    - Build Model and Adjust Policy (Mixed approach)

- Most of Current Models and Algorithms Used are **Mixed**

- They use Mutual Cooperative Solutions, Ex:
  - **Policy** // **Value**
    - » Actor-Critic [Barto et al. 1983]
    - » SARSA –  $\epsilon$ -greedy
    - » Q-Learning [Watkins 1989]
  - **Model** // **Policy**
    - » Dyna [Sutton 1990]
- They also have Variants (Optimizations, Extensions, Combinations...)
  - » Ex: Queue-Dyna, Prioritized Sweeping, RTDP, VRDP, Feudal Q-Learning...



# Model-Based Reinforcement Learning

---

- **Q-Learning** is the most known and is widely used
- Still, the user needs to adjust the **exploration vs exploitation** tradeoff
  - Ex: more exploration at the beginning
  - And more greedy at the end (once near convergence)
- **There is No Best General Approach/Algorithm**
- This Depends on Application/Scenario Characteristics
  - Information Known a priori (ex: Transition Model) vs None
  - Relative Computation vs Experience Costs (and **Risks**: ex: **Death** !)
  - Algorithm Complexity (Space and Time) vs Memory and Computing Power
  - Determinism vs Stochastic
  - Timing Constraints vs Near Optimality Convergence
  - Simplicity
  - Possibility of Incorporating Human Expertise

# Storage/Memory Issue

- Important Issue: **Storing the Transition Model**

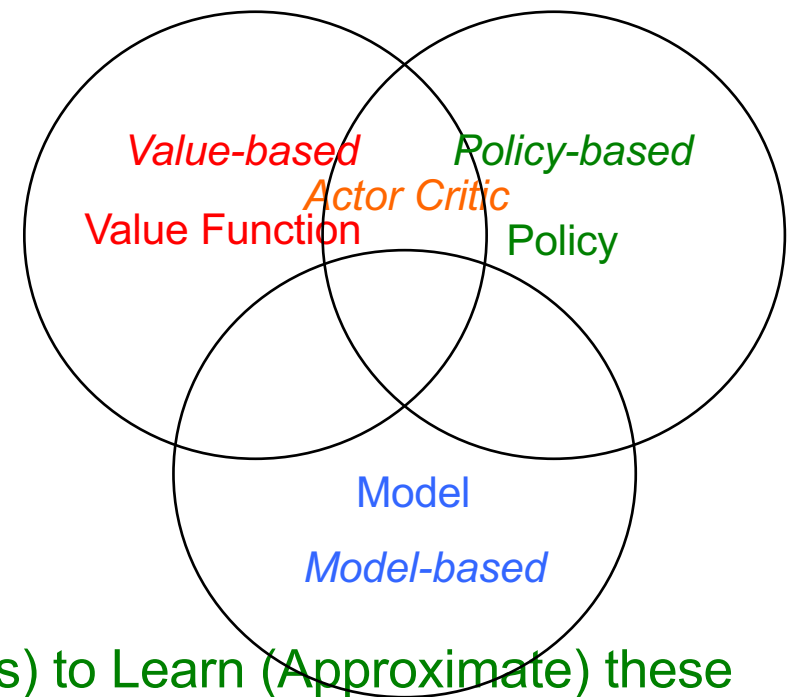
- It may be huge

- Same issue for other possible Mappings

- **Policies:**  $s \rightarrow a$
  - **State value function:**  $s \rightarrow R$
  - **Action value function:**  $\langle s, a \rangle \rightarrow R$
  - ...

- Use of Supervised Learning (ex: Neural Networks) to Learn (Approximate) these Mappings from Examples

- REINFORCE [Williams 1987]
  - Recurrent Q-learning [Schmidhuber 1991]
  - TD-Gammon [Tesauro 1992]
    - » Neural Network with TD-Learning (TD-based, in place of Backpropagation)
    - » Expert level
    - » But tentatives to apply to other games were not successful... until...
    - » Success in Backgammon : finite game (reward info) and transitions sufficiently stochastic (exploration)



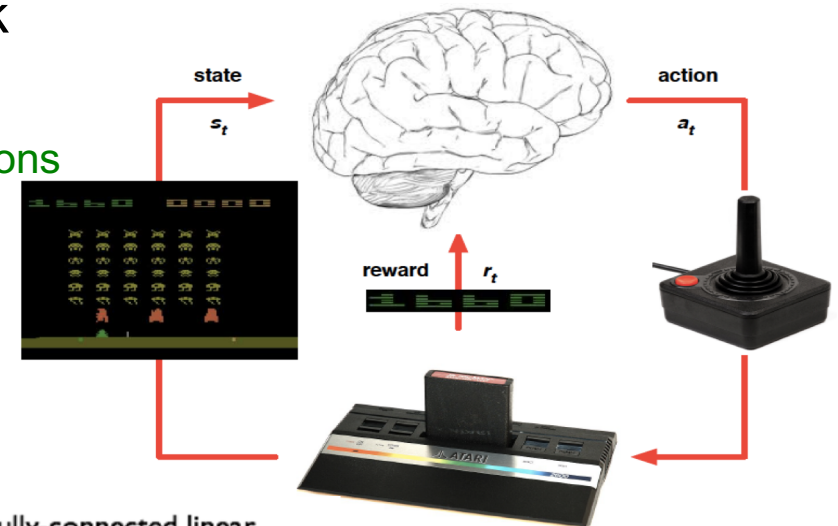
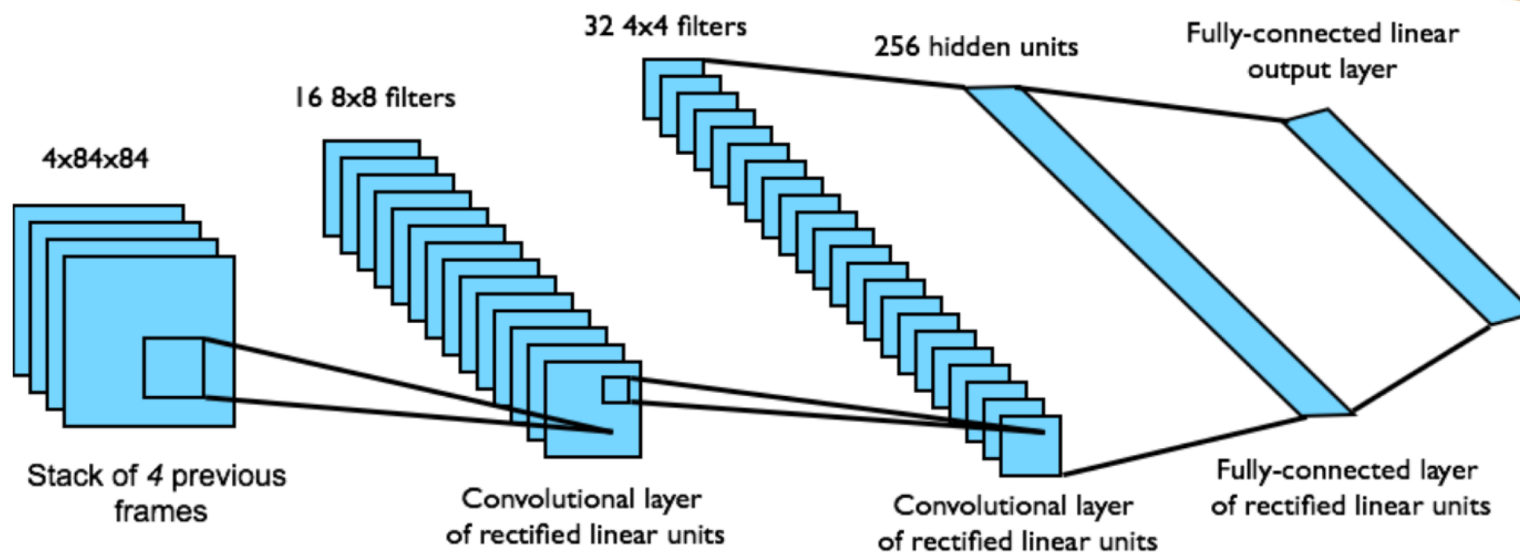
# Deep Reinforcement Learning [Silver et al. 2013]

---

- Use Deep Network to represent:
  - Policy
  - or Value
  - or Model
- Optimize it (Policy, or Value or Model) end-to-end by using stochastic gradient descent [Silver 2016]
- Ex: Deep Q-Learning [Minh et al. 2013]
- Represent Value function by a Deep Q-Network  $Q(s, a, w) \approx Q^\pi(s, a)$
- Train Value function Deep Network with inputs (<state,action> pairs) and outputs (values)

# Deep Q-Learning [Minh et al. 2013]

- Atari Games Playing (DeepMind Technologies)
- Represent Value function as a Deep Q-Network
- Training of Value function Network
  - Inputs: Game screen raw pixels & joystick/button positions
  - Outputs: Q-values (captured from game play)
    - » Reward  $\in \{-1, 0, 1\}$
- On-Line Training through Game Play
- Works with ANY (ATARI) Game !



# Q-Learning Algorithm [Watkins 1989]

---

initialize Q table(*#states*, *#actions*) arbitrarily;

observe initial state *s*;

repeat

    select and carry out action *a*;

    observe reward *r* and new state *s'*;

$Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a));$                       update rule

$s := s'$ ;

until terminated

# New Deep Q-Learning Update Rule

---

1. feedforward pass for current state  $s$  to predict Q-values for all possible actions;
2. feedforward pass for next state  $s'$  and select largest Q-value:  $\max Q := \max_a Q(s', a)$ ;  
     $\max Q$  is the best value for possible actions
3. for the action corresponding to  $\max Q$ , set Q-value target to  $r + \gamma \max Q$ ;  
     $r + \gamma \max Q$  is the target value (the best value) to be learnt  
    for each other action, set Q-value target to Q-value predicted in step 1;  
    This means error will be 0 for all of them
4. update the weights using backpropagation.

# Deep Q-Learning Algorithm [Minh et al. 2013]

---

```
initialize Q table(#states, #actions) arbitrarily;
observe initial state s;
repeat
    select and carry out action a;
    observe reward r and new state s';
    feedforward pass for current state s to predict Q-values for all possible actions;
    feedforward pass for next state s' and select largest Q-value:  $\max Q := \max_a Q(s', a)$ ;
    for the action corresponding to maxQ, set Q-value target to  $r + \gamma \max Q$ ;
    for each other action, set Q-value target to Q-value predicted in step 1;
    update the weights using backpropagation;
    s := s';
until terminated
```

# Deep Reinforcement Learning Algorithm [Minh et al. 2013]

---

(Main) Tricks/Optimizations:

## 1. Inputs/Outputs

- Input: the four last screens (images/pixels)
- Output: Q-values for each possible action

## 2. Experience Replay

- During game play, all experiences ( $\langle a, a, s', a' \rangle$ ) are stored in a replay memory
- During training, random minibatches from the replay memory are used
- Avoids similarity situation and favors exploration/generalization

## 2. $\epsilon$ -Greedy Exploration

- $\epsilon$  Probability to choose a random action, otherwise greedy (choose action with highest Q-value)
- $\epsilon$  decreases in time from 1 (completely random) until a plateau (0.1)
  - » analog to simulated annealing

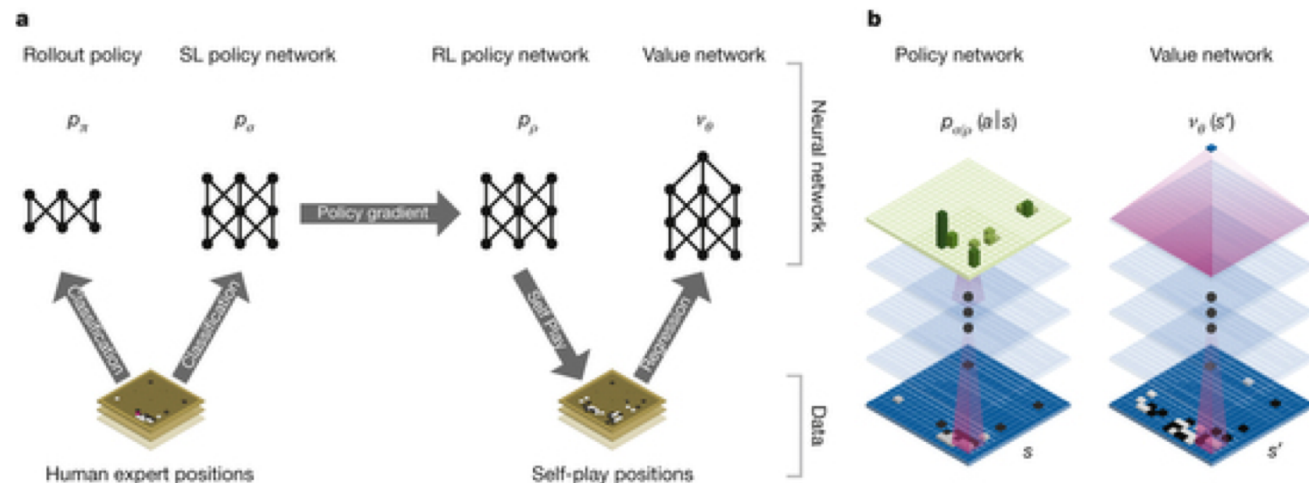
# Deep Search and Deep Reinforcement Learning

- Deep Search (and Deep Reinforcement Learning)
- AlphaGo Go Playing (Google DeepMind) [Silver et al. 2016]
- 2 Deep Networks to reduce search space
- "Policy Network" predicts next move and reduces width
- "Value Network" estimates winner in each position and reduces depth (analog to but better than alpha-beta pruning)
- Also Uses Reinforcement Learning to learn better policies,
- in order to improve the "Policy network",
- and in turn to improve the "Value Network"



AlphaGo 3 – 0 Ke Jie  
27/05/2017

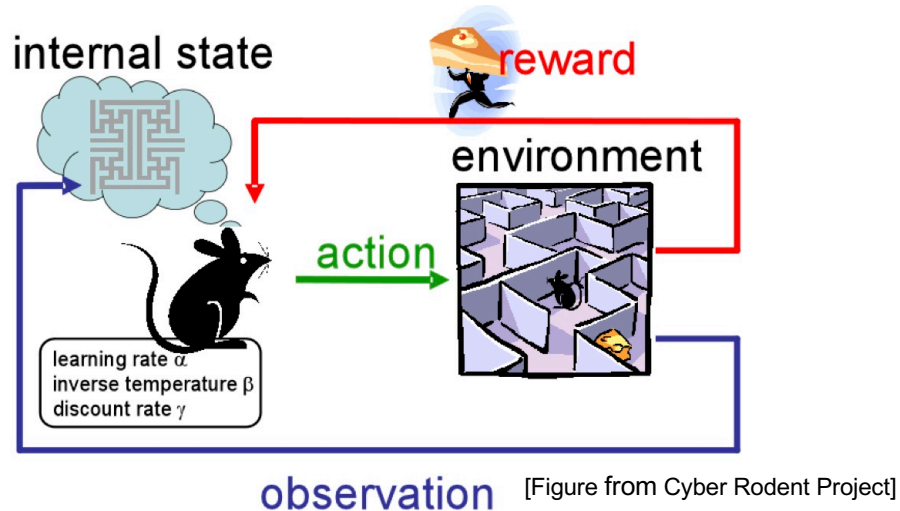
Jean-Pierre Briot



Deep Learning – Music Generation – 2019

## #4 Limitation – Control – #4 Solution: Reinforcement Learning

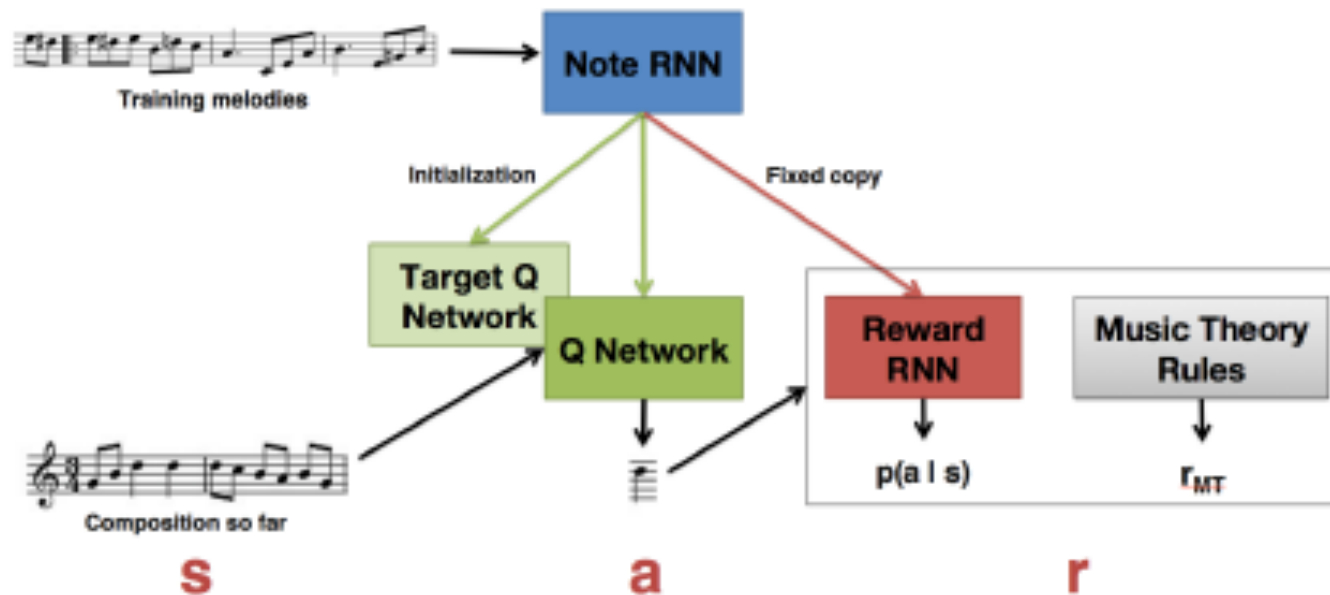
- Reformulation of Melody Generation as a Reinforcement Learning Problem



- Sequential decision/action problem
  - Objective: Learn via past decisions/actions a near optimal policy (state  $\rightarrow$  action) for maximizing the gain (sum of rewards)
- State: Melody generated so far (Succession of notes)  
  
Composition so far
  - Action: Generation of next note 

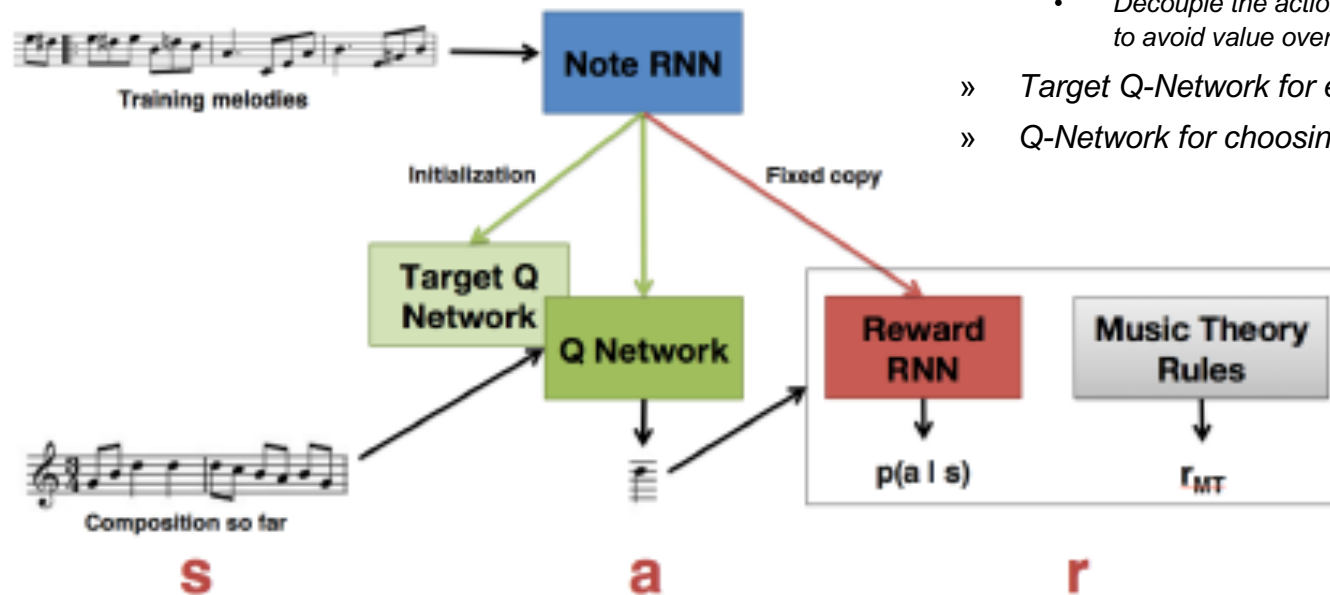
# Deep Reinforcement Learning (RL) – RL-Tuner [Jaques et al., 2016]

- *RNN (Recurrent Neural Network) Trained on the Corpus*
- **Reward** = Combination of:
  - Adherence to **What has been Learnt** by the RNN
    - » Similarity to the Prediction by the RNN
  - Adherence to **Music Theory Rules**
    - » Tonality, Avoid excessive repetitions...
    - » Open: Arbitrary **User-defined Targets**



# Deep Reinforcement Learning (RL) – RL-Tuner [Jaques et al., 2016]

- *RNN (Recurrent Neural Network) Trained on the Corpus*
- **Reward** = Combination of:
  - Adherence to **What has been Learnt** by the RNN
    - » Similarity to the Prediction by the RNN
  - Adherence to **Music Theory Rules**
    - » Tonality, Avoid excessive repetitions...
    - » Open: Arbitrary **User-defined Targets**
- Reinforcement Learning implemented via *Deep Learning ! (Deep Reinforcement learning)*
- *Double Deep Reinforcement Learning*
  - » Double Q-Learning [Hasselt et al., 2015]
    - Decouple the action selection from the evaluation, to avoid value overestimation
  - » Target Q-Network for estimating Gain (Q)
  - » Q-Network for choosing next Action (next Note)



# RL-Tuner – Results

without

with

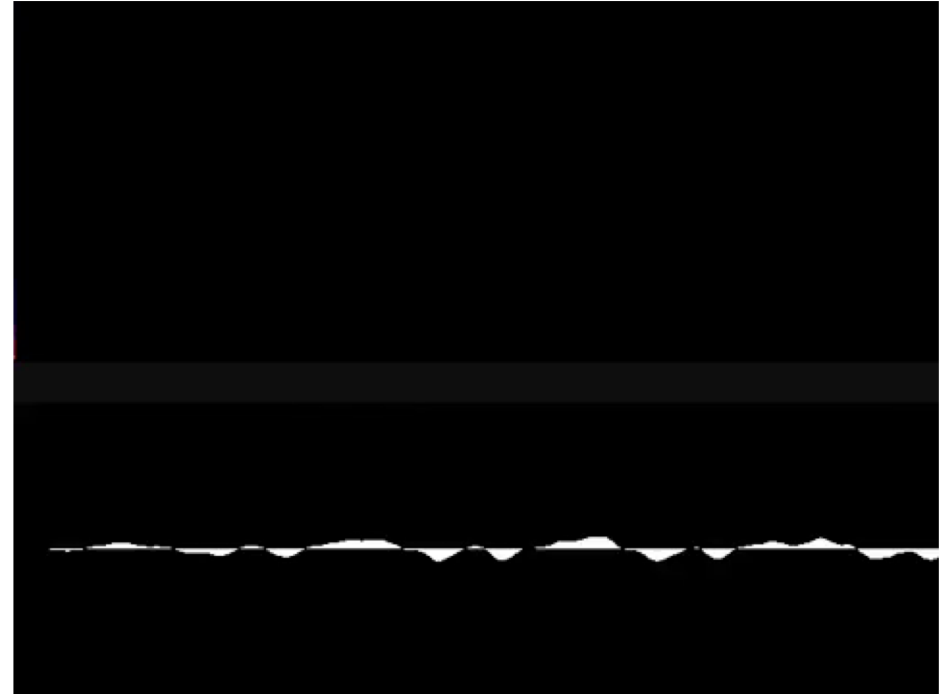
| Behavior                          | Note RNN        | Q-Learning            | Psi                    | G             |
|-----------------------------------|-----------------|-----------------------|------------------------|---------------|
| Notes excessively repeated        | 63.3%           | <b>0.0%</b>           | <b>0.02%</b>           | <b>0.03%</b>  |
| Notes not in key                  | 0.1%            | 1.0%                  | 0.6%                   | 28.7%         |
| Mean autocorrelation (lag 1,2,3)  | -.16, .14, -.13 | <b>-.11, .03, .03</b> | <b>-.10, -.01, .01</b> | .55, .31, .17 |
| Leaps resolved                    | 77.2%           | <b>91.1%</b>          | <b>90.0%</b>           | 52.2%         |
| Compositions starting with tonic  | 0.9%            | <b>28.8%</b>          | <b>28.7%</b>           | 0.0%          |
| Compositions with unique max note | 64.7%           | 56.4%                 | 59.4%                  | 37.1%         |
| Compositions with unique min note | 49.4%           | 51.9%                 | <b>58.3%</b>           | <b>56.5%</b>  |
| Notes in motif                    | 5.9%            | <b>75.7%</b>          | <b>73.8%</b>           | <b>69.3%</b>  |
| Notes in repeated motif           | 0.007%          | <b>0.11%</b>          | <b>0.09%</b>           | <b>0.01%</b>  |

# RL-Tuner – Demo

---



LSTM



RL-Tuner

<https://magenta.tensorflow.org/2016/11/09/tuning-recurrent-networks-with-reinforcement-learning>

# Autonomous Music Generated Systems

---



Deep Learning/Network

<https://www.youtube.com/watch?v=jfs1k7krXt8>

<https://www.youtube.com/watch?v=adn8KEV7Ing>

**IBM Watson Beat**

Deep Learning/Network  
and Reinforcement learning

<https://www.youtube.com/watch?v=Z5ymVzTUU6Y>

[https://www.youtube.com/watch?v=bx6\\_aEn-lj8](https://www.youtube.com/watch?v=bx6_aEn-lj8)



(Deep) Reinforcement Learning

<https://www.youtube.com/watch?v=GlpI0Fus70A>



Symbolic + Emotions Framework

<https://www.youtube.com/watch?v=GlpI0Fus70A>