

# IMITATION LEARNING WITH DAGGER OR PRIVILEGED INFORMATION

Ευφυής Έλεγχος (ECE\_ΔΚ807)



Μαρία - Νίκη Ζωγράφου / Α.Μ.: 1096060

Γέροντας Νικόλαος / Α.Μ.: 1092813

# Περιεχόμενα

Στόχος Εργασίας.....	2
Expert.....	3
Εκπαίδευση Expert Πράκτορα με Deep Q-Learning.....	3
Περιγραφή Προβλήματος: .....	3
Προεπεξεργασία Περιβάλλοντος: .....	3
Αρχιτεκτονική Νευρωνικού Δικτύου:.....	3
Deep Q-Learning .....	5
Reward Shaping.....	5
Εμπειρική Μάθηση – Experience Replay.....	5
Διαδικασία Εκπαίδευσης .....	6
Κριτήρια Εξειδίκευσης Expert.....	6
Visual utils .....	6
Γραφικές.....	7
Συμπεράσματα:.....	9
Imitation Learning.....	10
Behaviour Cloning .....	10
Συλλογή Δεδομένων (Expert Demonstrations).....	10
Εκπαίδευση του Behavior Cloning Agent .....	10
Εκτέλεση Πειράματος .....	11
Αξιολόγηση .....	11
DAGGER.....	15
Παρουσίαση αποτελεσμάτων: .....	16
DAgger με Behaviour Cloning Warm up .....	17
Σύγκριση μεθόδων: Behavior Cloning vs DAgger: .....	18
Οδηγίες Εγκατάστασης .....	19
Προβλήματα που αντιμετωπίσαμε .....	21
Βιβλιογραφία .....	22

## Τα βίντεο μας στο YouTube:

[https://youtu.be/h9JXUzJS6rU?si=cJCze\\_dh0Qy8pmU7](https://youtu.be/h9JXUzJS6rU?si=cJCze_dh0Qy8pmU7)  
<https://youtu.be/idxq4bWbvA0?si=uFaUeUJ6ANvZHxFK>

## GitHub link:

[https://github.com/ManyaZ1/DAGGER\\_Imitation\\_Learning](https://github.com/ManyaZ1/DAGGER_Imitation_Learning)

## Στόχος Εργασίας

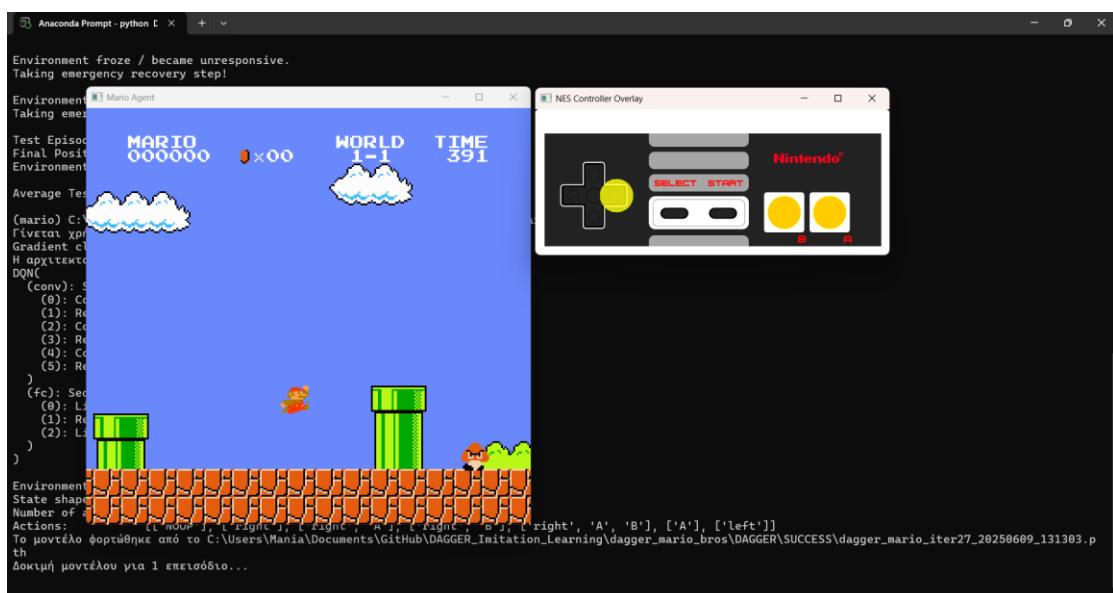
Στόχος της εργασίας είναι η εκπαίδευση ενός agent μέσω παραδειγμάτων από έναν expert, ο οποίος έχει πρόσβαση στο πλήρες state του περιβάλλοντος (privileged information). Αντίθετα, ο agent που εκπαιδεύεται έχει πρόσβαση μόνο σε μερικώς παρατηρήσιμα ή θορυβώδη δεδομένα, όπως εικόνες ή χαμηλής διάστασης χαρακτηριστικά.

Επιλέξαμε να εφαρμόσουμε αυτές τις **Reinforcement Learning** μεθόδους στον σχεδιασμό και την εκπαίδευση ενός πράκτορα (agent) που παίζει το κλασικό παιχνίδι **Super Mario Bros πίστα 1-1**. Το συγκεκριμένο περιβάλλον αποτελεί μια πρόκληση για μάθηση μέσω μίμησης, καθώς συνδυάζει δυναμική φυσική, συνεχείς μεταβάσεις κατάστασης και υψηλής διάστασης δεδομένα (εικόνες). Στόχος μας είναι η μελέτη της αποτελεσματικότητας του **Imitation Learning**, με έμφαση στη σύγκριση των μεθόδων **Behavior Cloning** και **DAGGER**, υπό συνθήκες μερικής παρατηρησιμότητα (partial observability).



Αρχικά υλοποιήθηκε η εκπαίδευση του expert, χρησιμοποιώντας Deep Q-Learning. Στη συνέχεια εφαρμόστηκε το **Behavior Cloning** για την απευθείας εκπαίδευση του agent από τον expert. Τέλος εφαρμόστηκε η μέθοδος **DAGGER (Dataset Aggregation)**, όπου ο agent εκπαιδεύεται επαναληπτικά στο περιβάλλον ενώ κάνει ερωτήσεις στον expert.

Σκοπός είναι να συγκριθεί η απόδοση των δύο μεθόδων (Behavior Cloning και DAGGER) υπό καθεστώς περιορισμένης πληροφορίας εισόδου και να αναλυθεί η ικανότητα γενίκευσης της policy σε καταστάσεις με θόρυβο ή νέα δεδομένα.



# Expert

## Εκπαίδευση Expert Πράκτορα με Deep Q-Learning

### Περιγραφή Προβλήματος:

Ο στόχος μας είναι η εκπαίδευση ενός ενισχυτικού πράκτορα (reinforcement learning agent) ικανού να ολοκληρώσει πίστες του περιβάλλοντος Super Mario Bros. Ο expert πράκτορας χρησιμοποιείται στη συνέχεια για Imitation Learning σε κατάσταση με μερική παρατηρησιμότητα. Ο agent βασίζεται στον αλγόριθμο Deep Q-Network (DQN) και επιτυγχάνει επιτυχή ολοκλήρωση επιπέδων. Για την εκπαίδευσή του, πέραν από Deep Q-Learning χρησιμοποιούμε βελτιώσεις όπως frame skipping, reward shaping, experience replay, εξερεύνηση με  $\epsilon$ -greedy, και target network.

### Προεπεξεργασία Περιβάλλοντος:

Πριν τη μάθηση, το περιβάλλον περνά από wrapper MarioPreprocessor, ο οποίος:

- Μετατρέπει τα RGB frames σε grayscale.
- Κάνει resize σε 84x84 pixels.
- Διατηρεί τα τελευταία 4 frames (frame stacking) για να εισάγει χρονική εξάρτηση.
- Εφαρμόζει frame skipping (4 frames ανά action).

Αυτό μειώνει τη διάσταση της παρατήρησης και βοηθάει στην απόδοση του DQN.

**Σημείωση:** Το frame skipping (παράλειψη καρέ) είναι μια τεχνική που χρησιμοποιείται συχνά σε περιβάλλοντα Reinforcement Learning. Κανονικά, ένα παιχνίδι (όπως το Mario) τρέχει σε 60 FPS. Αν ο πράκτορας κάνει επιλογή κάθε frame, οι αποφάσεις του θα είναι αχρείαστα λεπτομερείς. Με frame skipping, επιλέγει μία ενέργεια κάθε 4 frames και το ίδιο action **επαναλαμβάνεται 4 φορές**. Έτσι η εκπαίδευση γίνεται ταχύτερη και υπάρχει λιγότερος θόρυβος στο state.

### Αρχιτεκτονική Νευρωνικού Δικτύου:

Το μοντέλο είναι ένα **Deep Q-Network (DQN)** που παίρνει ως είσοδο **4 καρέ διαστάσεων 84x84**, και προβλέπει **Q-τιμές** για κάθε διαθέσιμη ενέργεια. Περιλαμβάνει:

- 3 συνελικτικά επίπεδα (Conv layers):
  - 32 φίλτρα ( $8 \times 8$ , stride 4)
  - 64 φίλτρα ( $4 \times 4$ , stride 2)
  - 64 φίλτρα ( $3 \times 3$ , stride 1)
- 1 fully-connected επίπεδο με 512 μονάδες.
- Τελική έξοδο για  $n_{actions}$  Q-values.

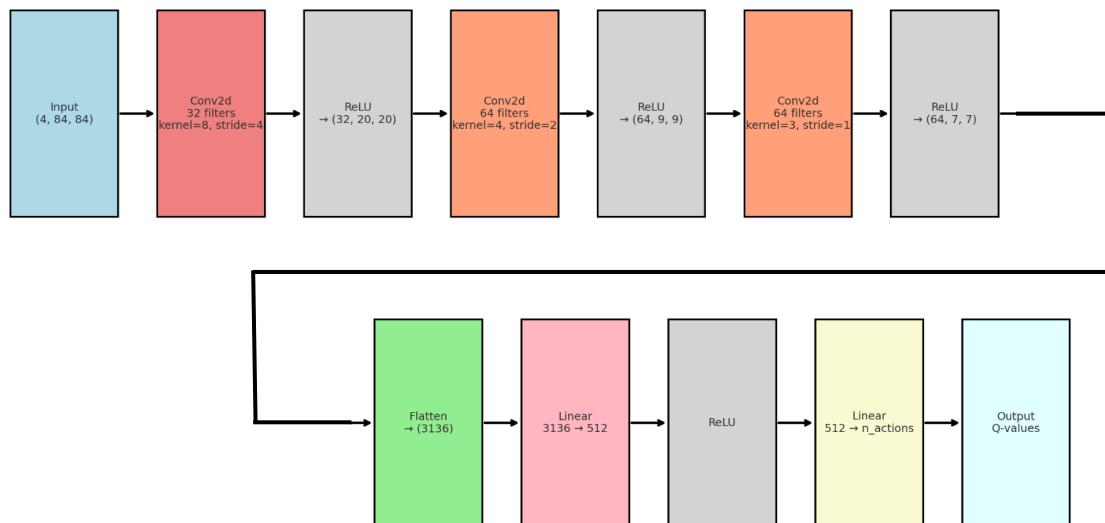
Χρησιμοποιούνται **ReLU** ενεργοποιήσεις και **Adam Optimizer** με learning rate 5e-5.

### Δομή Νευρωνικού στον κώδικα:

```
# 1o συνελικτικό επίπεδο
# 32 separate convolution filters, stride 4 to downscale image,
input_shape[0] = 4
nn.Conv2d(input_shape[0], 32, kernel_size = 8, stride = 4),
# input (4, 84, 84) → 4 channels => 32 outputs : each output is a
(20×20) map
nn.ReLU(), # element-wise relu
# Shape: (32, 20, 20)

# 2o συνελικτικό επίπεδο
nn.Conv2d(32, 64, kernel_size = 4, stride = 2),
nn.ReLU(),
# Shape: (64, 9, 9)

# 3o συνελικτικό επίπεδο
nn.Conv2d(64, 64, kernel_size = 3, stride = 1),
nn.ReLU()
# Shape: (64, 7, 7)
```



## Deep Q-Learning

Ο πράκτορας χρησιμοποιεί  **$\epsilon$ -greedy policy** με αρχικό  $\epsilon = 1.0$ , το οποίο φθίνει εκθετικά ( $\epsilon^* = 0.99995$ ) ως  $\epsilon_{\min} = 0.01$ .

Κατά την εκπαίδευση, εφαρμόζεται Deep Q-learning:

- Οι ενημερώσεις Q βασίζονται στο target network για σταθερότητα.
- Η ενημέρωση γίνεται ανά 1000 βήματα.
- Εφαρμόζεται **gradient clipping** ( $||\nabla|| \leq 1$ ) για αποφυγή exploding gradients.

Η εξίσωση ενημέρωσης Q είναι:

$$Q(s, a) \leftarrow r + \gamma * \max_{a'} Q_{target}(s', a')$$

όπου:

- r shaped reward
- $\gamma=0.99$  discount factor

## Reward Shaping

Για να ενισχυθεί η εκπαίδευση, εφαρμόστηκε **custom shaped reward**:

- **Πρόοδος προς τα δεξιά** (βάσει x\_pos)  $\rightarrow$  bonus +0.1 ανά pixel.
- **Χρονικό penalty**  $\rightarrow$  -0.1 ανά βήμα.
- **Ποινή για θάνατο**  $\rightarrow$  -10.
- **Bonus για τερματισμό**  $\rightarrow$  +100.

Η παραπάνω διαμόρφωση βοηθάει στην κατανόηση του στόχου από τον πράκτορα, οδηγώντας τον ταχύτερα στο να μάθει ότι για να ολοκληρώσει επιτυχώς την πίστα, πρέπει να κινηθεί προς τα δεξιά πατώντας σωστά και σε συνδυασμό τα κουμπιά A και B.

## Εμπειρική Μάθηση – Experience Replay

Η εκπαίδευση γίνεται με χρήση buffer **100,000 memories/ experiences** (states, actions, rewards, next\_states, dones).

- Σε κάθε βήμα επιλέγεται ένα batch των 32 (self.batch\_size).
- Υπολογίζονται οι Q-τιμές και γίνεται backpropagation με **MSE loss**.

## Διαδικασία Εκπαίδευσης

Η εκπαίδευση πραγματοποιείται μέσω της κλάσης MarioTrainer:

- Κάθε επεισόδιο ξεκινάει με reset του περιβάλλοντος.
- Ο agent εκτελεί actions μέχρι να τερματίσει ή φτάσει max\_steps=800 (το World 1-1 απαιτεί το πολύ ~300 steps για να φτάσει ο Mario το flagpole).
- Μετά από κάθε action:
  - Αποθηκεύεται η εμπειρία/memory.
  - Γίνεται ενημέρωση μέσω replay().
- Η αποθήκευση των models γίνεται είτε περιοδικά, είτε μέσω cheat/hack που κάνει save το agent model που τερμάτισε την πίστα (τεστάρετε και με  $\epsilon = 0$ ).

## Κριτήρια Εξειδίκευσης Expert

Ο πράκτορας χαρακτηρίζεται ως «expert» εάν:

- Τερματίσει (info['flag\_get'] == True).
- Έχει reward > 3400 (βάση του shaped reward).
- Μπορεί να αναπαράξει το αποτέλεσμα χωρίς exploration ( $\epsilon = 0$ ).

Σε αυτή την περίπτωση, το μοντέλο αποθηκεύεται ως expert checkpoint για μεταγενέστερο imitation learning.

## Visual utils

Ο κώδικας του αρχείου visual\_utils.py έχει 2 κλάσεις:

- MarioRenderer  
Προβάλλει σε πραγματικό χρόνο το περιβάλλον του Gym σε παράθυρο αυξημένης ανάλυσης (βάσει scale factor). Μετατρέπει τα χρωματικά επίπεδα για σωστή εμφάνιση μέσω OpenCV και διατηρεί σταθερό ρυθμό προβολής στα 60 FPS.

- NESControllerOverlay

Χρησιμοποιώντας την εικόνα του NES controller ως interface, προβάλλονται σε πραγματικό χρόνο οι ενέργειες του agent με κίτρινη οπτική επισήμανση στα αντίστοιχα κουμπιά, κατά την εκτέλεση των actions του.

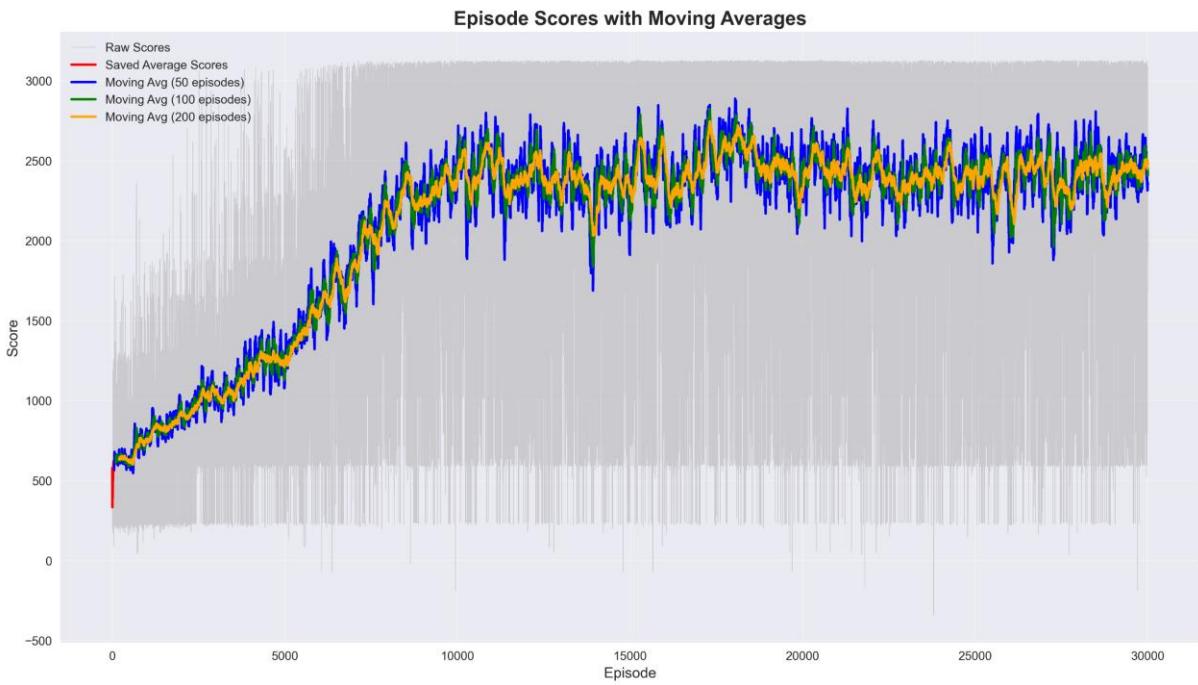


## Γραφικές

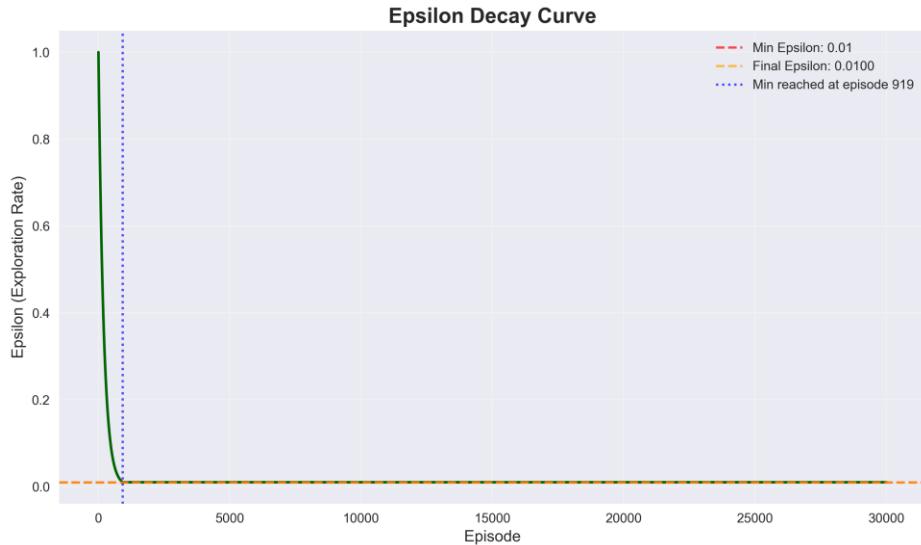
Σκορ επεισοδίων κατά τη διάρκεια της εκπαίδευσης:



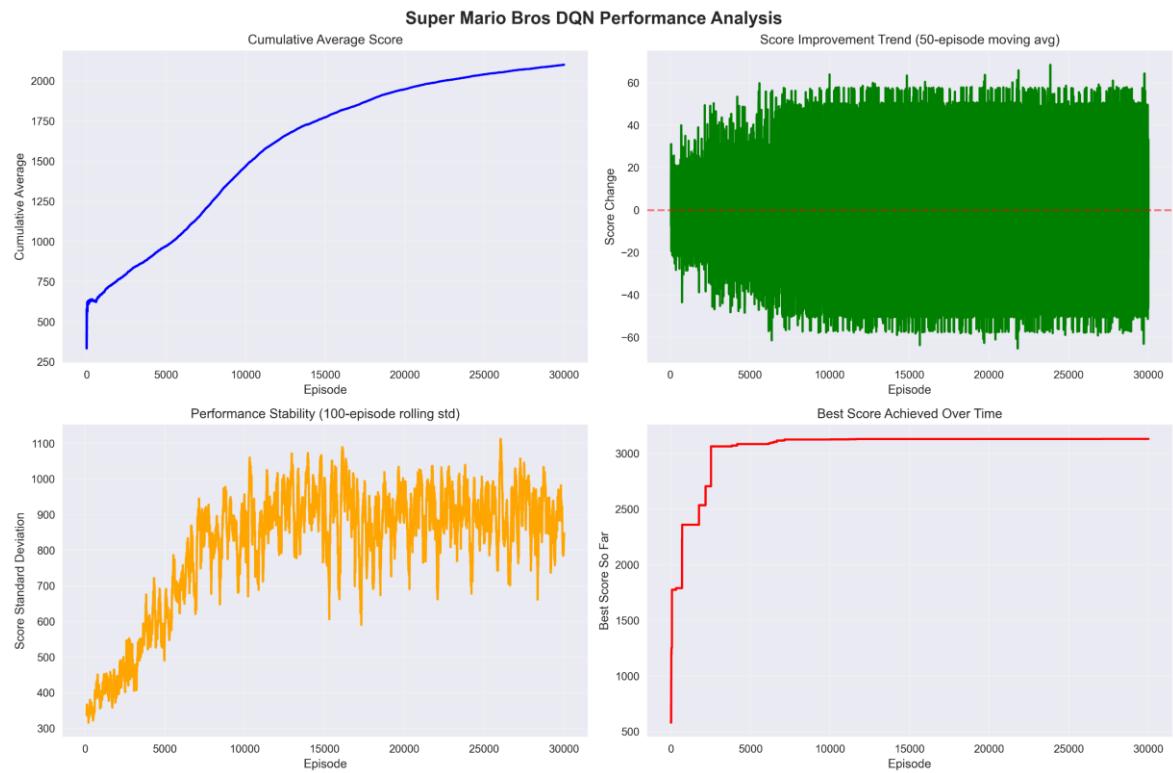
Κινούμενος μέσος όρος επεισοδίων κατά τη διάρκεια της εκπαίδευσης:



## Εκθετική μείωση του $\epsilon$ :



## Δείκτες Απόδοσης:



## Συμπεράσματα:

Παρατηρείται ότι, παρά την εκτεταμένη εκπαίδευση του agent σε πάνω από 30.000 επεισόδια, το σκορ παρουσιάζει **έντονες διακυμάνσεις**, ιδιαίτερα στα raw scores. Ωστόσο, **καταγράφονται επαναλαμβανόμενα επεισόδια όπου ο agent ολοκληρώνει επιτυχώς την πίστα**, με αποτέλεσμα την αποθήκευση μοντέλων που χαρακτηρίζονται ως *experts*.

Αυτό καταδεικνύει ότι ο agent έχει μάθει πώς να τερματίζει την πίστα όμως δεν έχει σταθεροποιηθεί πλήρως σε συμπεριφορά general-purpose που να οδηγεί αξιόπιστα σε υψηλό σκορ.

### Αιτιολόγηση των διακυμάνσεων

Οι βασικοί παράγοντες που ενδέχεται να ευθύνονται για την αστάθεια είναι η στοχαστικότητα της πολιτικής ( $\epsilon$ -greedy) και το time penalty ανά βήμα. Το ελάχιστο  $\epsilon=0.01$  και εφόσον η πολιτική είναι ακόμα stochastically greedy, μπορεί να αποτυγχάνει τυχαία. Επιπλέον το σχήμα επιβράβευσης περιλαμβάνει αρνητική ενίσχυση ανά βήμα, κάτι που βοηθάει σημαντικά στην αρχική εκπαίδευση, όμως στα τελικά στάδια μπορεί να τιμωρεί υπερβολικά έναν agent που καθυστερεί ελαφρώς ή "κολλάει" σε συγκεκριμένα σημεία, χωρίς να αποτυγχάνει πλήρως. Υπάρχει η πιθανότητα επίσης το buffer να κρατάει "bad episodes" και τα δείγματα να μην είναι καλά ισορροπημένα, η μάθηση γίνεται noisy.

### Απόφαση

Πιθανώς μια μεγαλύτερη εκπαίδευση (40.000+ επεισόδια ήταν προτεινόμενα από το Pytorch tutorial), με περισσότερο  $\epsilon$ -decay και παραπάνω τεχνάσματα, όπως αυξημένο batch size ή χρήση Prioritized Replay Buffer να έδινε πιο σταθερά αποτελέσματα. Αποφασίστηκε όμως να μην συνεχιστεί η εκπαίδευση του expert για τους εξής λόγους. Αρχικά, η **εκπαίδευση ήταν χρονοβόρα** και **υπολογιστικά απαιτητική**. Κατά δεύτερον, υπήρχαν **σαφώς επαναλήψιμα expert μοντέλα** που πετύχαιναν τον στόχο (τερματισμός πίστας). Τέλος η πίστα του παιχνιδιού είναι **ντετερμινιστική**, επομένως ένα μοντέλο που καταφέρνει να τερματίσει θα έχει αξιόπιστη συμπεριφορά και **θα τερματίζει πάντα**.

### Επιτυχημένα μοντέλα:

Στον παρακάτω φάκελο του αποθετηρίου μας στο GitHub έχουν κρατηθεί 4 από τα επιτυχημένα μοντέλα:

DAGGER\_Imitation\_Learning\daggerer\_mario\_bros\expert-SMB\_DQN\models

# Imitation Learning

## Behaviour Cloning

Το **Behavior Cloning (BC)** είναι μια supervised learning μέθοδος που χρησιμοποιείται για να εκπαιδεύσει ένα policy απευθείας από παρατηρήσεις expert. Είναι μια απλή μέθοδος, η οποία συλλέγει ζεύγη (**state, action**) από έναν **expert agent** ( $\pi^*$ ), και μετά εκπαιδεύει ένα νευρωνικό δίκτυο να προβλέπει τις ίδιες ενέργειες όταν βλέπει τις ίδιες καταστάσεις.

Δηλαδή: **(state) → model → (predicted action)**

Η εκπαίδευση γίνεται με τη χρήση Cross Entropy Loss μεταξύ των predicted actions και των expert actions.

Σύνηθες πρόβλημα του Behaviour Cloning agent, είναι ότι αφού δεν εκπαιδεύεται ο ίδιος στο περιβάλλον και απλά μαθαίνει να αντιγράφει τον expert κάνει κακή γενίκευση και όταν φτάνει σε καταστάσεις που δεν έχει δει ο expert αποτυγχάνει [1].

## Συλλογή Δεδομένων (Expert Demonstrations)

**Αρχείο:** behavior\_cloning.py, κλάση BehaviorCloningAgent

```
states, actions = self._prepare_data(demonstrations, observation_wrapper)
```

Χρησιμοποιείται ο ήδη εκπαιδευμένος expert agent DQN και εκτελείται στο περιβάλλον για N episodes. Για κάθε βήμα καταγράφεται το τρέχον state και η action του expert. Αποτέλεσμα: μια λίστα από λεξικά τύπου { 'state': ..., 'action': ... }.

## Εκπαίδευση του Behavior Cloning Agent

Ο BehaviorCloningAgent περιέχει ένα fully connected ή convolutional νευρωνικό δίκτυο (ανάλογα με το shape του observation). Κατά την εκπαίδευση:

1. το dataset χωρίζεται σε training/validation
2. κάθε epoch υπολογίζει training loss, validation loss, και validation accuracy
3. χρησιμοποιείται Cross Entropy Loss
4. αποθηκεύονται training history για plotting

Η εκπαίδευση γίνεται με χρήση της μεθόδου:

```
bc_agent.train(demonstrations=demonstrations, observation_wrapper=obs_wrapper, epochs=epochs, batch_size=32, validation_split=0.2)
```

Η εκπαίδευση διαρκεί 100 εποχές. Σε κάθε εποχή, το δίκτυο περνάει ολόκληρο το dataset των 2910 state-action ζευγών μέσω mini-batch gradient descent (batch\_size = 32), υπολογίζοντας training και validation metrics.

## Εκτέλεση Πειράματος

Το πλήρες πειράμα υλοποιείται στην κλάση MarioBehaviorCloningExperiment (cloning.py). Περιλαμβάνει:

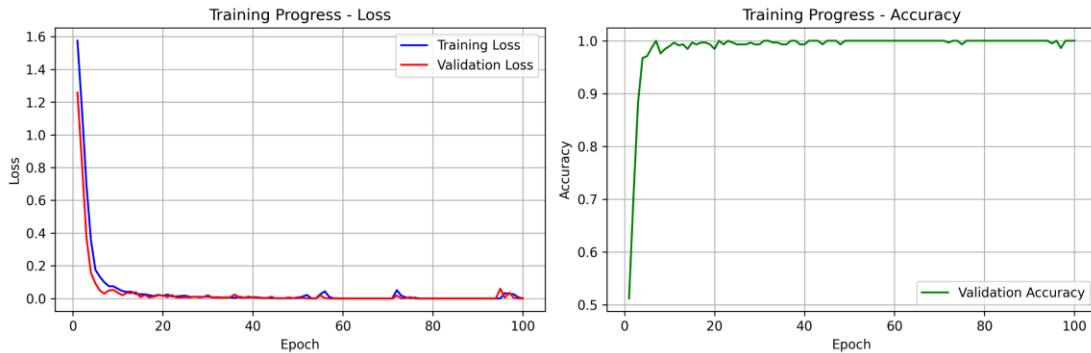
1. **Συλλογή expert demonstrations** από εκπαίδευμένο DQN agent
2. **Εκπαίδευση πολλών BC agents** κάτω από διαφορετικά observation settings:
  - o full\_state
  - o partial\_obs (2/4 channels)
  - o noisy\_obs ( $\sigma=0.1$ )
  - o noisy\_obs ( $\sigma=0.2$ )
  - o downsampled
3. **Αξιολόγηση agents** σε unseen episodes και σύγκριση performance με τον expert

## Αξιολόγηση

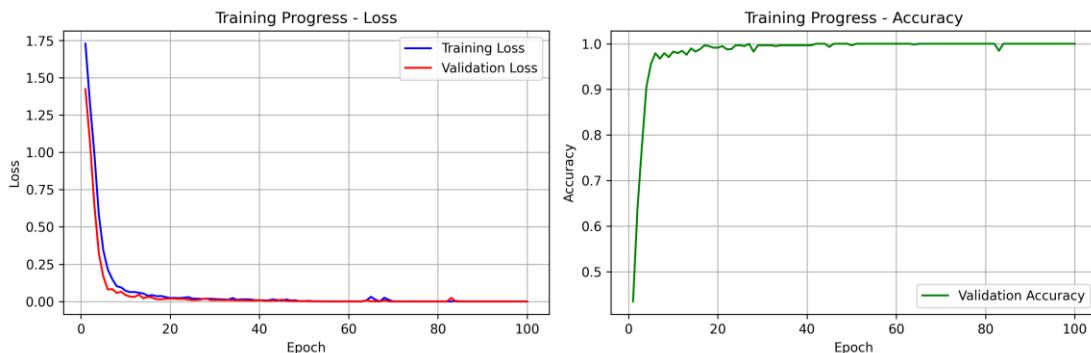
Παράγουμε training progress και loss plots και για τους 4 agents.

### Training progress/loss graphs

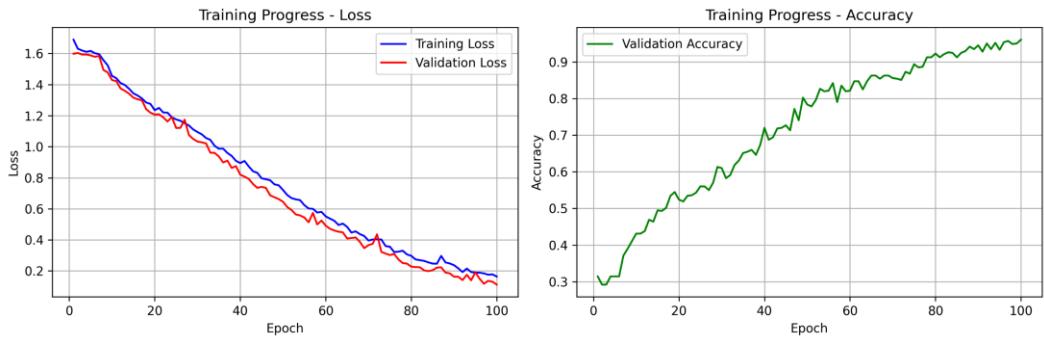
#### Full state observation agent:



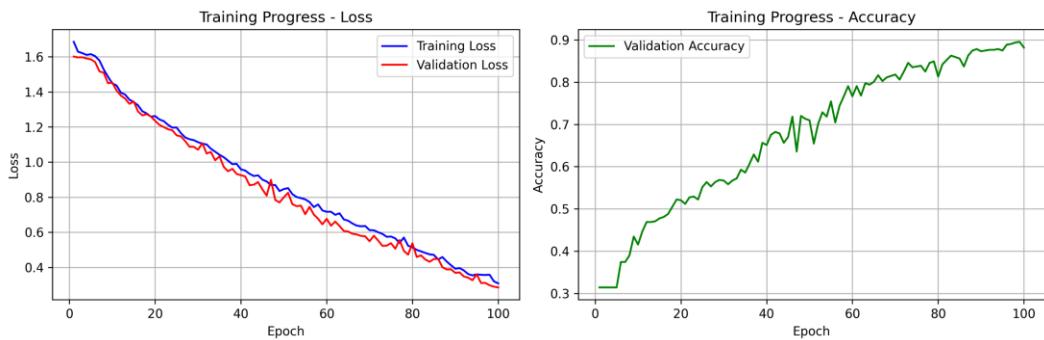
#### Partial observation 2/4 channels agent:



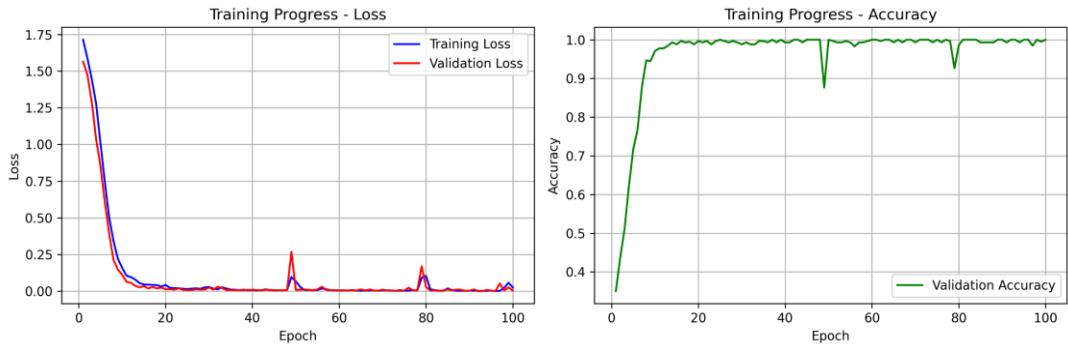
### Noisy observations ( $\sigma=0.1$ ) agent:



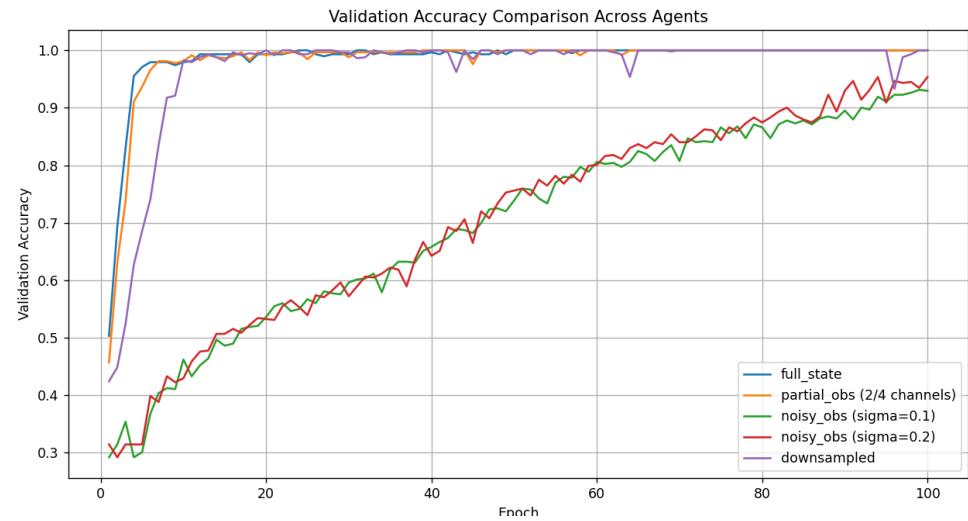
### Noisy observations ( $\sigma=0.2$ ) agent:



### Down sampled observations agent:



### Σύγκριση όλων των agent:



Παρατηρούμε ότι όπως αναμενόμενο όταν προσθέτουμε θόρυβο στις παρατηρήσεις το behavior cloning δυσκολεύεται να μάθει με ακρίβεια την συμπεριφορά του expert. Αντίθετα σε full state, partial state η εκμάθηση είναι γρήγορη και ομαλή. Στο down sampled επίσης η εκμάθηση είναι ομαλή με λίγες παραπάνω διαταραχές.

### Αποτελέσματα του testing

Το αρχείο **CLONING/tester.py** επιτρέπει στον χρήστη να επιλέξει agent και να δει την πίστα που παίζει το μοντέλο καθώς και να σώσει το σκορ του σε αρχείο txt.

Κατά την εκτέλεση πολλαπλών runs παρατηρήθηκαν τα εξής:

- **Full-state, partial observation και downsampled agents** παρουσιάζουν **σταθερή συμπεριφορά** και **τερματίζουν αξιόπιστα** το επίπεδο.
- **Noisy observation agents ( $\sigma=0.1$ )** εμφανίζουν **αστάθεια** — μπορούν να τερματίσουν, αλλά όχι πάντα, ανάλογα την εκπαίδευση.
- **Noisy agents με  $\sigma=0.2$**  αποτυγχάνουν **συστηματικά** να ολοκληρώσουν την πίστα, επιβεβαιώνοντας ότι υψηλότερα επίπεδα θορύβου διαταράσσουν σημαντικά την απόδοση.

```
-----evaluation_log_partial_obs.txt-----  
✓ Ολοκληρώθηκε! Score: 3063.0, X: 3161, Flag: True  
✓ Ολοκληρώθηκε! Score: 3063.0, X: 3161, Flag: True  
✓ Ολοκληρώθηκε! Score: 3063.0, X: 3161, Flag: True
```

```
-----evaluation_log_noisy_obs_sigma=0.1.txt-----  
(για διαφορετικά μοντέλα)  
✓ Ολοκληρώθηκε! Score: 3064.0, X: 3161, Flag: True  
✓ Ολοκληρώθηκε! Score: 1566.0, X: 1666, Flag: False  
✓ Ολοκληρώθηκε! Score: 1434.0, X: 1529, Flag: False  
✓ Ολοκληρώθηκε! Score: 3064.0, X: 3161, Flag: True
```

```
-----evaluation_log_noisy_obs_sigma=0.2.txt-----  
✓ Ολοκληρώθηκε! Score: 1886.0, X: 1977, Flag: False
```

```
-----evaluation_log_downsampled.txt-----  
✓ Ολοκληρώθηκε! Score: 3063.0, X: 3161, Flag: True
```

**Επόμενα Βήματα:**

1. Δημιουργία DAGGER για πιο robust συμπεριφορά
2. Εκπαίδευση του  $\sigma=0.1$  για παραπάνω εποχές.

Δοκιμή με  $\sigma=0.1$  για 200 εποχές:

Δοκιμή 1: Score: 2699.0, X: 3154, Flag: False

Δοκιμή 2:  Ολοκληρώθηκε! Score: 3063.0, X: 3161, Flag: True

Παρατηρούμε και πάλι ότι δεν είναι σταθερή η συμπεριφορά με θόρυβο, γεγονός που υποδεικνύει ότι για χρήση σε καταστάσεις με θόρυβο χρειάζεται πιο robust εκπαίδευση με DAGGER.

## DAGGER

Ο αλγόριθμος DAGGER (Dataset Aggregation) εκπαιδεύει πολιτικές με εποπτευόμενο τρόπο, αξιοποιώντας έναν expert agent. Η διαδικασία είναι επαναληπτική: σε κάθε iteration, η learner policy αλληλεπιδρά με το περιβάλλον (World 1-1) και παράγει trajectories με βάση τις τρέχουσες παρατηρήσεις της. Για κάθε παρατήρηση, ερωτάται ο expert ποια ενέργεια θα εκτελούσε. Το ζεύγος (state, expert\_action) αποθηκεύεται σε ένα dataset, το οποίο σταδιακά εμπλουτίζεται με δεδομένα που καλύπτουν τις αποτυχημένες ή edge-case συμπεριφορές του learner - δηλαδή καταστάσεις που πιθανόν να μην καλύπτονται από το αρχικό distribution του expert.

Αντί για Q-learning, η πολιτική του learner εκπαιδεύεται μέσω supervised learning, κάνοντας minimize την cross-entropy απώλεια μεταξύ των ενεργειών της και των ενεργειών του expert. Η υλοποίηση υποστηρίζει partial ή noisy observations με κατάλληλο wrapper [observation\_wrapper.py], έτσι ώστε ο learner να μαθαίνει να δρα με περιορισμένη ή αλλοιωμένη πληροφορία, ενώ ο expert έχει πλήρη αντίληψη του περιβάλλοντος. Αυτό ενισχύει την ικανότητα του learner να ανακάμπτει από δύσκολες ή άγνωστες καταστάσεις.

Ο DAGGER υπερβαίνει τους περιορισμούς του Behavior Cloning επειδή διορθώνει το distributional shift μεταξύ των states που βλέπει ο expert και εκείνων στα οποία οδηγείται ο learner όταν κάνει λάθη. Με το να μαθαίνει από τα δικά του λάθη (μέσω των expert labels), ο agent μαθαίνει ενεργά να αποφεύγει ή να ανακάμπτει από τις λανθασμένες αποφάσεις του.

Αναλυτικότερα:

3. Για κάθε iteration  $i$ , ο learner  $\pi_i$  παράγει trajectory  $\tau_i = \{(s_t)\}_{t=1}^T$ . Για κάθε  $s_t$  ο expert  $\pi^*$  δίνει label  $a_t^* = \pi^*(s_t)$ . To dataset D ενημερώνεται:

$$D \leftarrow D \cup \{(s_t, a_t^*)\}_{t=1}^T$$

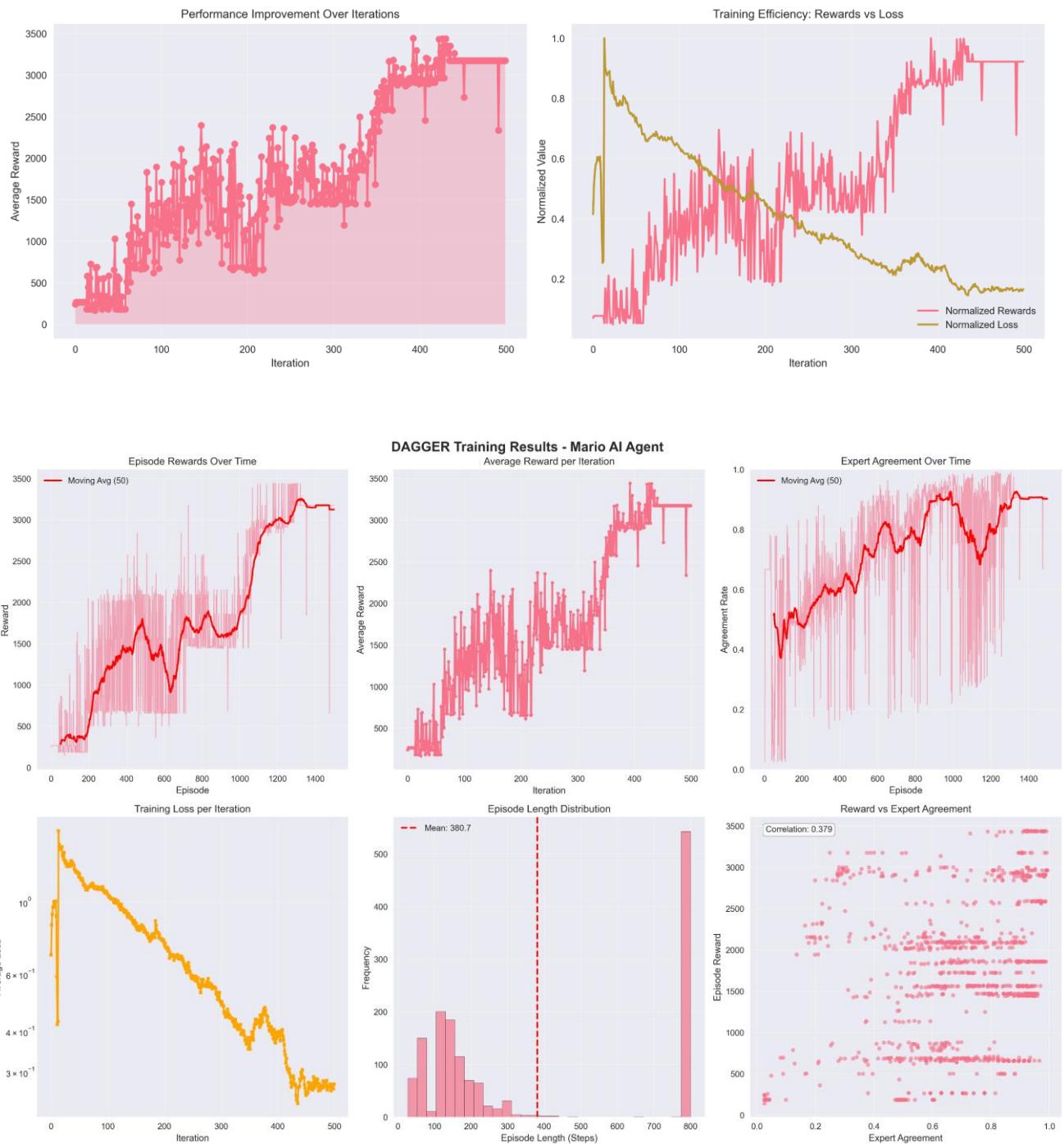
*υλοποιείται στη remember() μέθοδο*

4. Η πολιτική  $\pi_{i+1}$  προκύπτει από supervised learning:

$$\pi_{i+1} = \arg \min_{\pi} \mathbb{E}_{(s, a^*) \sim \mathcal{D}} [\mathcal{L}_{CE}(\pi(s), a^*)]$$

*υλοποιείται στη μέθοδο replay()*

## Παρουσίαση αποτελεσμάτων:



### DAGGER Training Summary

---

Total Episodes: 1500  
 Total Iterations: 500  
 Best Episode Reward: 3443.30  
 Average Episode Reward: 1825.55  
 Final Iteration Reward: 3175.40  
 Average Expert Agreement: 0.737  
 Final Expert Agreement: 0.907  
 Average Training Loss: 0.671067  
 Final Training Loss: 0.275337  
 Average Episode Length: 380.7

## DAgger με Behaviour Cloning Warm up

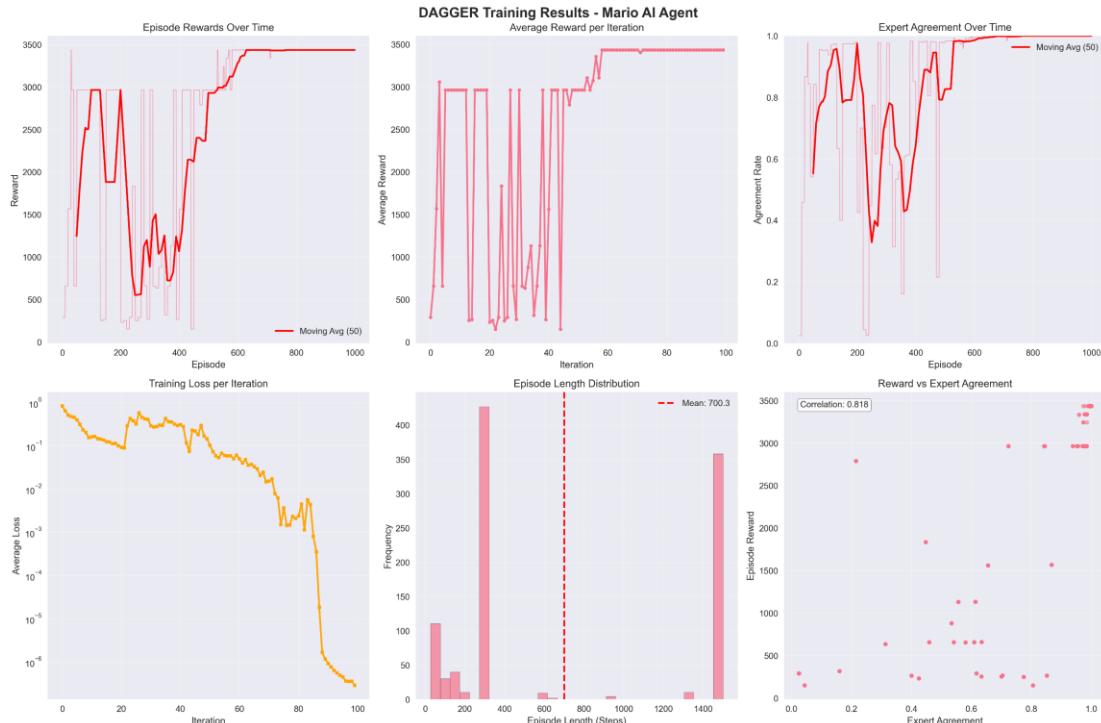
Δοκιμάσαμε επίσης την εκτέλεση του αλγορίθμου **DAgger** μετά από ένα σύντομο στάδιο **behavior cloning (BC)**. Η διαδικασία αυτή διασφαλίζει ένα αρχικό ποσοστό συμφωνίας (expert agreement) του agent με τον expert, το οποίο επιταχύνει αισθητά την σύγκλιση του DAGGER. Η υλοποίηση βρίσκεται στο αρχείο [dagger\\_trainer\\_BC.py](#), όπου μετά τη φάση BC — η οποία τερματίζεται είτε όταν επιτευχθεί προκαθορισμένο ποσοστό συμφωνίας είτε όταν φτάσει το μέγιστο όριο max\_frames — ξεκινά αυτόματα η φάση Dataset Aggregation.

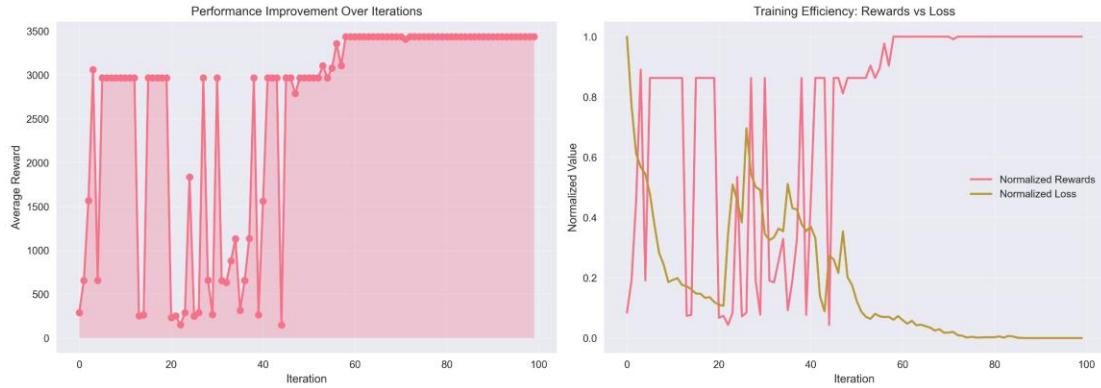
Έχουμε θέσει το behavior cloning να έχει ως στόχο ο agent να αποκτήσει συμφωνία με τον έξπερτ 70% είτε να συνεχίσει να εκπαιδεύεται μέχρι 10 χιλιάδες frames. (Στο προτεινόμενο paper[5] η εκπαίδευση συνεχίζοταν έως  $10 * 10^4$  data points.) Ο agent κατά την εκπαίδευση που εκτελέσαμε έφτασε γρήγορα στο agreement  $\geq 70\%$ .

```
self._behaviour_cloning_warmup(n_frames= 10_000  
                                pretrain_steps= 500, target_agreement=0.7 )
```

Τεστάραμε αυτή την προσέγγιση για την τυπική περίπτωση του **partial observation**. Παρατηρήθηκε ότι η εκπαίδευση του **DAgger** επιταχύνεται σημαντικά για παράδειγμα, αν ο agent φτάσει σε **80% Agreement** κατά το BC, ήδη από την 4η επανάληψη ο **DAgger agent** ενδέχεται να έχει μάθει πολιτική που τερματίζει την πίστα, ενώ στις περίπου **60 επαναλήψεις** η μάθηση έχει σταθεροποιηθεί, ενώ προηγουμένως χρειάστηκαν **~1400**. Υπήρξε δηλαδή σημαντική επιτάχυνση.

### Αποτελέσματα:





## Σύγκριση μεθόδων: Behavior Cloning vs DAgger:

Όπως περιεγράφηκε εκτενώς παραπάνω, και με τους δυο αλγορίθμους δημιουργούνται μοντέλα που τερματίζουν επιτυχώς την πίστα, έχοντας μερική παρατηρησιμότητα.

Βέβαια, στην περίπτωση εκπαίδευσης σε μια ντετερμινιστική πίστα με προϋπάρχοντα expert, η μέθοδος **Behavior Cloning (BC)** αποδείχθηκε η πιο **αποδοτική** προσέγγιση για τη δημιουργία agent. Χάρη στην απουσία αλληλεπίδρασης με το περιβάλλον κατά τη φάση της εκπαίδευσης, ο agent μαθαίνει **ταχύτερα** και δεν χρειάζεται τόσο έντονη εκπαίδευση. Τα μοντέλα που παράχθηκαν με BC, πετυχαίνουν αποτελέσματα συγκρίσιμα με τον expert, σε συνθήκες **partial observation**, όπου η είσοδος είναι τα μισά frames από όσα δέχεται ο expert ή η είσοδος είναι μία down sampled εικόνα της αρχικής. Επίσης παρατηρήθηκαν επιτυχημένα μοντέλα και για την περίπτωση χαμηλού θορύβου ( $\sigma=0.1$ ), όχι τόσο αξιόπιστα όμως. Η μέθοδος αυτή απέτυχε μόνο στην περίπτωση σημαντικού θορύβου ( $\sigma=0.2$ ). Επομένως αν ο στόχος είναι απλά ένα μοντέλο που εκτελεί μια ντετερμινιστική πίστα με μερική παρατηρησιμότητα, ενδείκνυται η εκπαίδευση learner με behaviour cloning.

Αντιθέτως, όταν ο στόχος είναι η **ανθεκτικότητα σε θόρυβο** ή **υπάρχει μειωμένη αξιοπιστία παρατήρησης**, η μέθοδος **DAgger** υπερέχει. Αν και απαιτεί περισσότερες επαναλήψεις και αλληλεπίδραση με το περιβάλλον, οδηγεί σε **σημαντικά βελτιωμένη γενίκευση και σταθερότητα**. Ενδεικτικά, με προσθήκη **θορύβου ( $\sigma=0.2$ )** στο παρατηρούμενο input, παρατηρήθηκε ότι ο agent που εκπαιδεύτηκε αποκλειστικά με BC **απέτυχε να μάθει αποτελεσματική πολιτική** και δεν τερμάτισε την πίστα – λογικό, αφού ο expert δεν θα έχει συναντήσει ποτέ τέτοιες θορυβώδεις συνθήκες. Αντιθέτως, ο agent που εκπαιδεύτηκε με **DAgger πέτυχε πλήρη τερματισμό**, δείχνοντας ότι η συνεχής ανατροφοδότηση από τον expert σε “πραγματικά” states βοηθά στην προσαρμογή σε μη ιδανικές συνθήκες.

Για την επιτάχυνση της εκπαίδευσης του **DAgger**, που είναι το σημαντικότερο μειονέκτημα του έναντι του **Behavior Cloning**, καταλήξαμε πώς καταλυτικό ρόλο μπορεί να παίξει ένα **αρχικό training με behavior cloning**, ώστε ο agent να ξεκινήσει έχοντα ήδη ένα **baseline agreement** με τον expert και ύστερα να αρχίσει δοκιμές στο περιβάλλον για να αποκτήσει πιο σθεναρή συμπεριφορά γρηγορότερα.

# Οδηγίες Εγκατάστασης

Δημιουργία environment:

```
# --- Create a new conda environment ---
conda create -n mario python=3.8 -y

# --- Activate the environment ---
conda activate mario

# --- Install required Python packages via pip ---
# Microsoft Visual C++ 14.0 or greater is required.
https://visualstudio.microsoft.com/visual-cpp-build-tools/

pip install -r requirements_mario.txt

or

pip install gym==0.23.1
pip install nes-py==8.1.8
pip install gym-super-mario-bros==7.4.0
pip install opencv-python
pip install scikit-learn
pip install matplotlib
pip install seaborn
pip install torch

# --- For CUDA-compatible version of PyTorch ---
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu118
```

```
# Requirements
gym==0.23.1
nes-py==8.1.8
gym-super-mario-bros==7.4.0
opencv-python
scikit-learn
matplotlib
seaborn
torch
```

Προτείνετε να χρησιμοποιήστε anaconda prompt ή κάποιο άλλο terminal για να τρέξετε τα προγράμματα καθώς υπάρχει θέμα με την εύρεση των path στα windows.

Λειτουργίες:

εκπαίδευση behaviour cloning: CLONING/cloning.py  
εκπαίδευση expert: expert-SMB\_DQN/main.py → επιλογή train ή test με visualize  
εκπαίδευση DAGGER: DAGGER/dagger\_trainer.py

Αναλυτικότερα:

**Expert:**

Τα models που τερματίζουνε επιτυχώς την πίστα ή έχουνε οριστεί να αποθηκεύονται βάση checkpoint βρίσκονται στο directory: `expert-SMB_DQN\models`. Υπάρχουν ήδη 4 μοντέλα στον φάκελο με ένα .txt να εξηγεί τον λόγο που τα κρατήσαμε.

Γραφήματα στο:

`DAGGER_Imitation_Learning\daggerer_mario_bros\expert-SMB_DQN\mario_plots`

**Behaviour Cloning:**

Για εκπαίδευση εκ νέου: `python cloning_main.py`

Για τεστ μοντέλων: `python tester.py`

Γραφήματα στο:

`DAGGER_Imitation_Learning\daggerer_mario_bros\CLONING\test_results`

Μοντέλα στο:

`DAGGER_Imitation_Learning\daggerer_mario_bros\CLONING\models`

**DAGGER:**

Έχουμε κρατήσει μοντέλα που τερματίζουνε επιτυχώς την πίστα στο directory: `DAGGER\SUCCESS`. Νέα μοντέλα από την εκπαίδευση βρίσκονται στο: `models_dagger`. Παράλληλα, μέσω του αρχείου `DAGGER_main.py` μπορούμε να δούμε τον τρόπο με τον οποίο παίζει το μοντέλο που επιλέγουμε.

# Προβλήματα που αντιμετωπίσαμε

Κατά την ανάπτυξη και εκπαίδευση των μοντέλων παρατηρήθηκαν σημαντικές τεχνικές δυσκολίες, τόσο σε επίπεδο λογισμικού όσο και υπολογιστικής υποδομής.

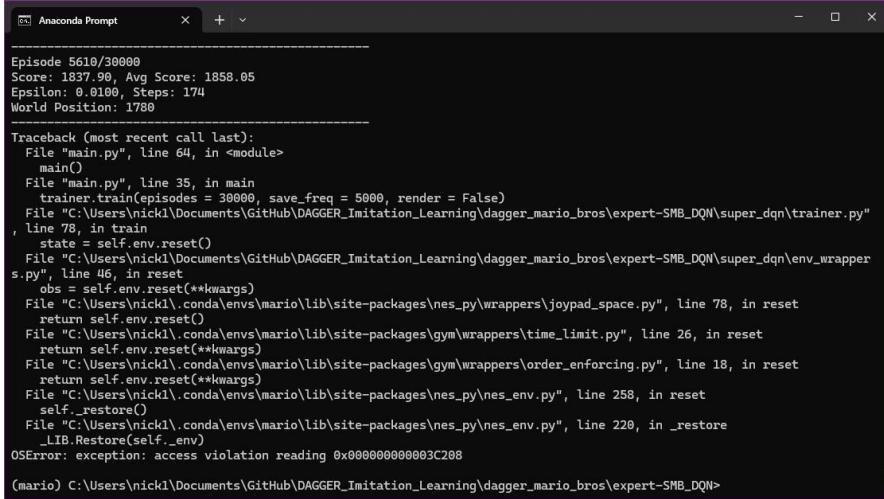
Πιο συγκεκριμένα:

Η χρήση του Python Gym environment, και ιδιαίτερα του nes\_py emulator, παρουσίασε συχνά σφάλματα μνήμης και αστάθεια, με χαρακτηριστικό παράδειγμα το access violation error της παρακάτω εικόνας. Το συγκεκριμένο σφάλμα εμφανιζόταν τυχαία κατά τη διάρκεια του training, ειδικά μετά από μεγάλα χρονικά διαστήματα εκτέλεσης.

Παράλληλα, η εκπαίδευση του expert μοντέλου διήρκησε πάνω από 72 ώρες, χωρίς να ολοκληρωθεί επιτυχώς, εξαιτίας memory leaks και αναιτιολόγητων termination κατά την εκτέλεση.

Επιπλέον το DAGGER training εμφάνισε αντίστοιχα προβλήματα, καθώς απαιτούσε συνεχή συνεργασία μεταξύ 2 agents, επαναλαμβανόμενα training loops και επεξεργασία χιλιάδων frames, καθιστώντας το σύστημα ευάλωτο σε σφάλματα μνήμης και επιβράδυνση.

Η παρακάτω εικόνα αποτυπώνει ενδεικτικά ένα τέτοιο σφάλμα, το οποίο επηρέασε την ομαλή ροή της εκπαίδευσης:



```
Anaconda Prompt
-----
Episode 5610/30000
Score: 1837.98, Avg Score: 1858.05
Epsilon: 0.0100, Steps: 174
World Position: 1780

Traceback (most recent call last):
  File "main.py", line 64, in <module>
    main()
  File "main.py", line 35, in main
    trainer.train(episodes = 30000, save_freq = 5000, render = False)
  File "C:/Users/nickl/Documents/GitHub/DAGGER_Imitation_Learning/dagger_mario_bros/expert-SMB_DQN/super_dqn/trainer.py"
, line 78, in train
    state = self.env.reset()
  File "C:/Users/nickl/Documents/GitHub/DAGGER_Imitation_Learning/dagger_mario_bros/expert-SMB_DQN/super_dqn/env_wrapper
s.py", line 46, in reset
    obs = self.env.reset(**kwargs)
  File "C:/Users/nickl/.conda/envs/mario/lib/site-packages/nes_py/wrappers/joypad_space.py", line 78, in reset
    return self._env.reset()
  File "C:/Users/nickl/.conda/envs/mario/lib/site-packages/gym/wrappers/time_limit.py", line 26, in reset
    return self._env.reset(**kwargs)
  File "C:/Users/nickl/.conda/envs/mario/lib/site-packages/gym/wrappers/order_enforcing.py", line 18, in reset
    return self._env.reset(**kwargs)
  File "C:/Users/nickl/.conda/envs/mario/lib/site-packages/nes_py/nes_env.py", line 258, in reset
    self._restore()
  File "C:/Users/nickl/.conda/envs/mario/lib/site-packages/nes_py/nes_env.py", line 220, in _restore
    _LTB.Restore(self._env)
OSError: exception: access violation reading 0x00000000003C208
(mario) C:/Users/nickl/Documents/GitHub/DAGGER_Imitation_Learning/dagger_mario_bros/expert-SMB_DQN>
```

Τέλος, λόγω του υψηλού υπολογιστικού κόστους της εκπαίδευσης και του χρόνου που απαιτούσε για να ολοκληρωθεί, προσπαθήσαμε να κάνουμε χρήση google Colab, το περιβάλλον του gym όμως υποστηριζόταν μόνο από python 3.8 και δεν υπάρχει δυνατότητα down grade της python στο google colab.

## Βιβλιογραφία

- [1] <https://imitation.readthedocs.io/en/latest/algorithms/bc.html>
- [2] <https://arxiv.org/pdf/1509.06461>
- [3] [https://docs.pytorch.org/tutorials/intermediate/mario\\_rl\\_tutorial.html](https://docs.pytorch.org/tutorials/intermediate/mario_rl_tutorial.html)
- [4] <https://imitation.readthedocs.io/en/latest/algorithms/dagger.html>
- [5] [https://www.ri.cmu.edu/pub\\_files/2011/4/Ross-AISTATS11-NoRegret.pdf](https://www.ri.cmu.edu/pub_files/2011/4/Ross-AISTATS11-NoRegret.pdf)
- [6] [2010.11251](https://arxiv.org/pdf/1010.11251.pdf)