Analogie Séance de révision Train Step

Simple DQN

Acteurs :
- séance de révision → **TrainStep**
- Etudiant → **policy network**
- Professeur → **target network**
- Cahier d'exercice → **replay buffer**

Step 1 Prédiction par l'étudiant

On prend dans le cahier de problème de math passé **(replay buffer)** un exercice **(state)**
On prend ensuite la réponse de l'étudiant sur cet exercice **(action).**

On demande aujourd'hui avec les connaissances de l'étudiant de donner une note **(Q Value)** à sa réponse **(action)** de cet exercice **(state).**

```
q_values = model(states).gather(1, actions.unsqueeze(1)).squeeze(1)
```

Step 2 Evaluation par le professeur

Ensuite on demande au professeur **(target network)** d'imaginer la note **(Max_nextQ)** maximal que l'étudiant peut obtenir sur le problème suivant. (Il surestime)

```
max_next_q = target_model(next_states).max(dim=1)[0]
```

Step 3 La note cible totale

Le professeur calcule la note parfaite que l'étudiant aurait du s'attribuer à la step 1.
Il prend les points réellement obtenus et ajoute le potentiel du problème suivant en le réduisant un peu car c'est une récompense future. La note parfaite est **target Q**

```
        target_q = rewards + gamma * max_next_q * (1 - dones)
```

Step 4 la comparaison (Perte)

Le prof à maintenant 2 notes en main :
- La note que l'étudiant s'est donné **(q_values)**
- La note parfaite qu'il a calculé **(target_q)**

La différence entre les deux est la **loss**

```
loss = F.mse_loss(q_values, target_q)
```

Step 5 La Leçon (mise à jour)

En fonction de cet écart le prof donne un conseil précis à l'étudiant qui met à jour ses connaissances.

```
loss.backward()
optimizer.step()
```

Double DQN

Step 1 Prédiction par l'étudiant (identique)

On prend dans le cahier de problème de math passé **(replay buffer)** un exercice **(state)**
On prend ensuite la réponse de l'étudiant sur cet exercice **(action).**

On demande aujourd'hui avec les connaissances de l'étudiant de donner une note **(Q Value)** à sa réponse **(action)** de cet exercice **(state).**

```
q_values = model(states).gather(1, actions.unsqueeze(1)).squeeze(1)
```

Step 2 Dialogue prudent (changement radical)

Le professeur demande à l'étudiant regarde ce problème suivant (**next_state**), selon toi quelle est la meilleure réponse ? (**next_action**)

```
next_actions = model(next_state_batch).argmax(dim=1)
```

Le professeur évalue le choix le choix de cette (**next_action**) en donnant la note de ce choix (**next_q**)

```
next_q = target_model(next_state_batch).gather(1,
next_actions.unsqueeze(1)).squeeze(1)
```

Step 3 La note cible totale (prudente)

Le prof calcule la note parfaite en utilisant ce dialogue, cette nouvelle target est plus fiable et moins optimiste (car l'étudiant à donné l'action qu'il a ensuite lui noté) **(target_q)**

```
target_q = reward_batch + gamma * next_q * (1 - done_batch)
```

Step 4 Comparaison (perte) (identique)

Comme dans l'histoire 1, le prof compare la note étudiant **(q_values)** avec la note parfaite **(target_q)** qui est plus prudent et nous donne la loss

```
loss = F.mse_loss(q_values, target_q)
```

Step 5 La Leçon (mise à jour) (identique)

En fonction de cet écart le prof donne un conseil précis à l'étudiant qui met à jour ses connaissances.

```
loss.backward()
optimizer.step()
```

Bonus dans le cas d'un PER :
On calcule la loss en fonction de l'importance de la transition

```
td_errors = q_values - target_q
loss = (weights * td_errors.pow(2)).mean()
```

Le Modèle DQN : Un Cerveau en Deux Parties

Votre modèle `DQN` est composé de deux grandes sections :

1. **L'extracteur de caractéristiques (les yeux)** : Une série de couches de convolution qui analysent l'image pour identifier des éléments importants (la balle, les briques, la raquette).
2. **Le décideur (le cortex)** : Une série de couches entièrement connectées qui prennent les informations visuelles extraites et décident de la valeur de chaque action possible.

Couche 1 : `self.conv1`

- **Constitution** : Une couche de convolution 2D (`nn.Conv2d`) avec 32 filtres de taille 8x8 et un "pas" (stride) de 4.
- **Fonctionnement** : Chaque filtre de 8x8 "glisse" sur l'image d'entrée avec un pas de 4 pixels. Un filtre est spécialisé dans la détection d'un motif simple (par exemple, une ligne horizontale, un coin, un dégradé de couleur). En utilisant 32 filtres, cette couche recherche 32 types de motifs de bas niveau différents sur toute l'image. Le grand pas de 4 permet de réduire rapidement la taille de l'image.
- **Sortie** : Un tenseur de forme `(batch_size, 32, 20, 20)`. On a maintenant 32 "images" de 20x20, chacune représentant la carte d'activation d'un filtre spécifique.
- **Pourquoi elle est là ?** Pour effectuer une première passe de détection de caractéristiques très simples et pour réduire drastiquement la dimensionnalité de l'image d'entrée.

Couche 2 : `self.conv2`

- **Constitution** : Une couche de convolution 2D avec 64 filtres de taille 4x4 et un pas de 2.
- **Fonctionnement** : Cette couche prend en entrée les 32 cartes de caractéristiques de la couche précédente. Ses filtres (4x4) sont plus petits et cherchent à combiner les motifs simples détectés par `conv1` pour former des motifs plus complexes (par exemple, un groupe de lignes peut devenir une "brique", un arc de cercle peut faire partie de la "balle").
- **Sortie** : Un tenseur de forme `(batch_size, 64, 9, 9)`. La taille est encore réduite, mais le nombre de caractéristiques a doublé.
- **Pourquoi elle est là ?** Pour construire une représentation visuelle plus riche en combinant les caractéristiques de bas niveau en concepts de plus haut niveau.

Couche 3 : `self.conv3`

- **Constitution** : Une couche de convolution 2D avec 64 filtres de taille 3x3 et un pas de 1.
- **Fonctionnement** : La dernière couche de convolution. Avec un petit filtre (3x3) et un petit pas (1), elle affine la détection des caractéristiques complexes sans réduire

davantage la taille spatiale de manière agressive. Elle regarde des combinaisons encore plus élaborées des 64 cartes précédentes.
- **Sortie** : Un tenseur de forme `(batch_size, 64, 7, 7)`. C'est la représentation visuelle finale et la plus abstraite que le réseau obtient.
- **Pourquoi elle est là ?** Pour finaliser l'extraction des caractéristiques visuelles pertinentes avant de passer à la prise de décision.

Partie 2 : Le Décideur (Le Dueling DQN)

À la sortie de `conv3`, nous avons un volume de données de `64 x 7 x 7 = 3136` valeurs. C'est une représentation compressée et riche de l'état du jeu. Avant de pouvoir l'utiliser, on doit l'aplatir en un simple vecteur de 3136 nombres.

Couche 4 : `self.fc1`

- **Constitution** : Une couche entièrement connectée (`nn.Linear`) qui prend les 3136 valeurs en entrée et les transforme en 512 valeurs.
- **Fonctionnement** : Chaque neurone de cette couche est connecté à toutes les 3136 entrées. Cette couche apprend à reconnaître les relations et les combinaisons entre les différentes caractéristiques visuelles abstraites pour former une "compréhension" globale de l'état du jeu.
- **Sortie** : Un vecteur de 512 neurones. C'est le "vecteur de pensée" final de l'agent.
- **Pourquoi elle est là ?** Pour synthétiser toutes les informations visuelles en un seul vecteur de haut niveau qui représente l'état du jeu.