גלעד מרקמן

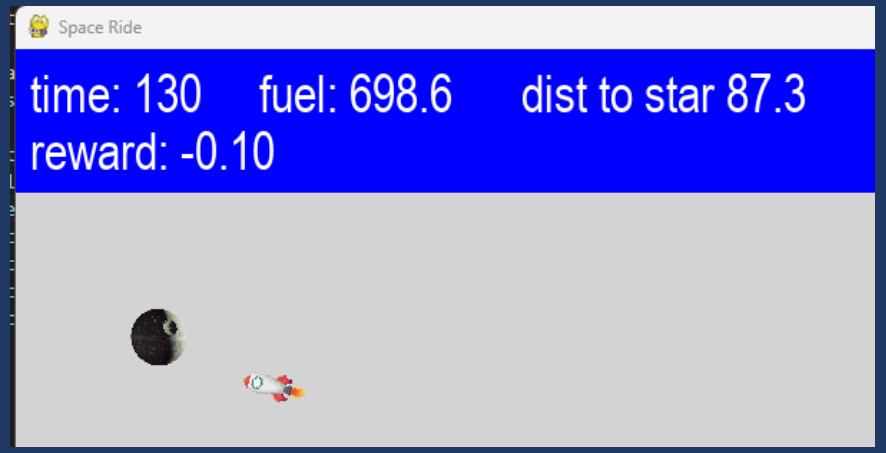
קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

# REINFORCE MC Continuous


קישור ל GitHub

# מרחב פעולות רציף

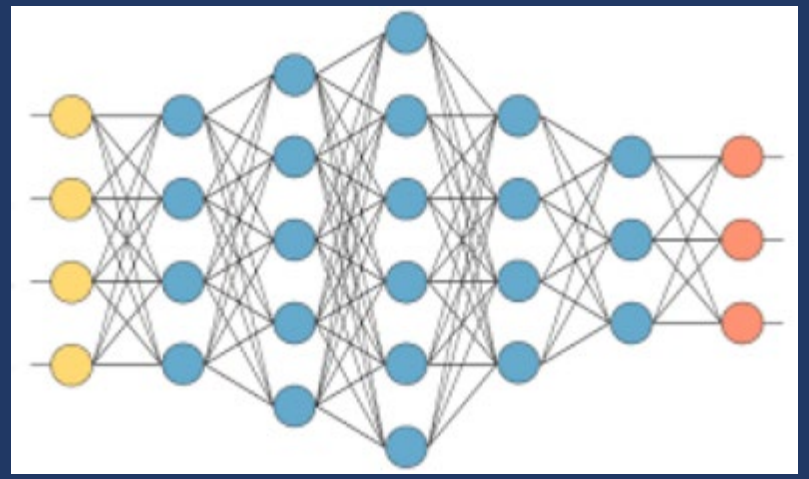
- אחד היתרונות המשמעותיים של המודלים המבוססים על Policy Gradient הוא היכולת לטפל בסביבות בהן מרחב הפעולות הוא רציף ולא בדיד.
- נדגים באמצעות משחק פשוט שבו החללית צריכה להגיע למטרה.
- לחללית שתי פעולות:
  - מצערת של המנוע קדימה : [0,1] בהתאם לכיוון האף של החללית.
  - סיבוב ימינה שמאלה [-1,1].
- תנועת החללית כוללת חיכוך.



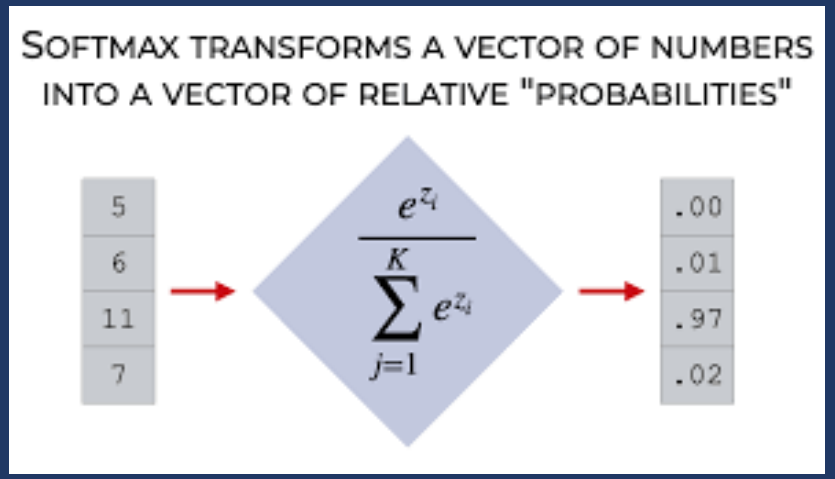
# פונקציית ה Policy במרחב בדיד

- תזכורת – במרחב פעולות בדיד רשת הנוירונים שלנו מקבלת state ומוציאה התפלגות לכל אחד מהפעולות (actions).
- ההתפלגות נעשית באמצעות פונקציית SoftMax.
- אנחנו בוחרים את הפעולה באמצעות דגימה הסתברותית, ככל שערך של הפעולה גבוה יותר יש לה סיכוי גבוה יותר להבחר.

State

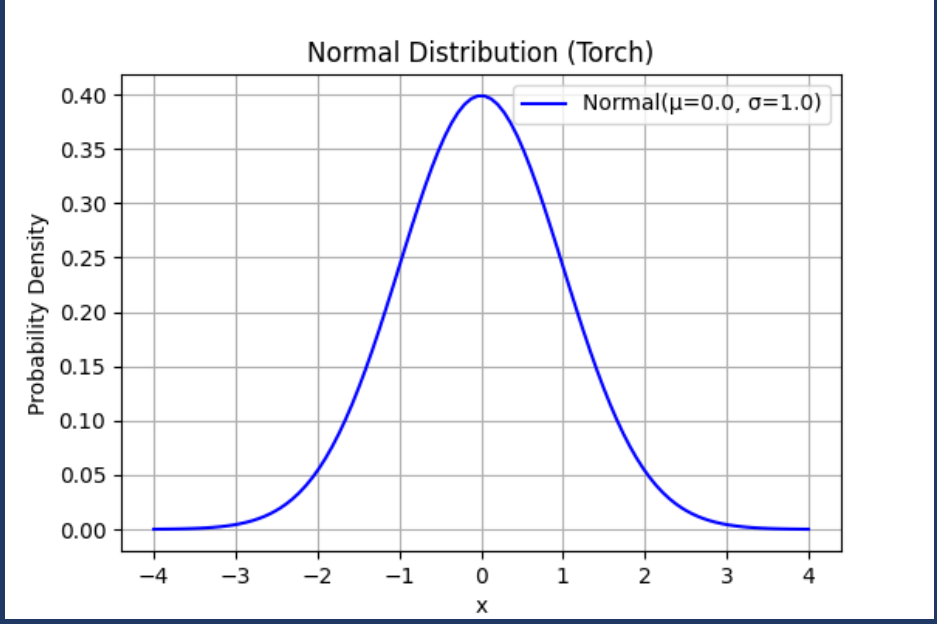


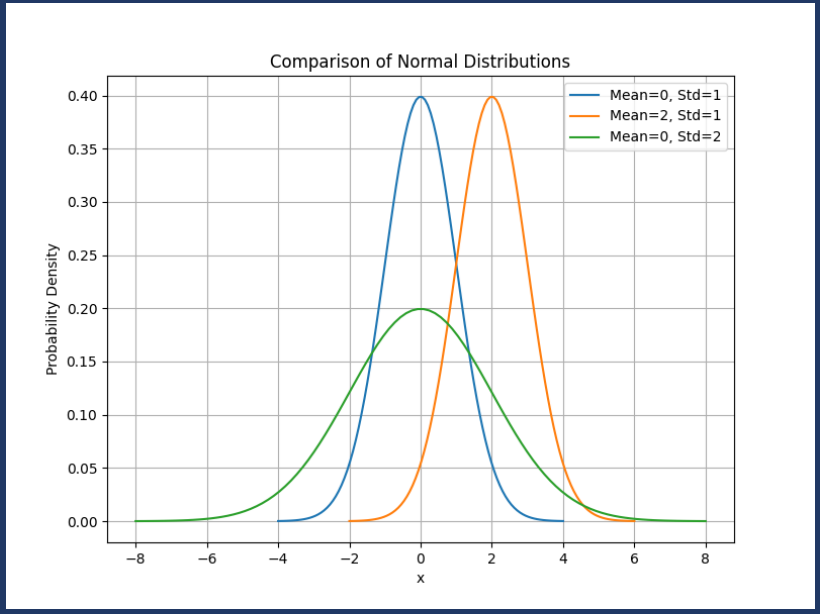
Actions
Probs



# מימוש פונקציית המדיניות במרחב רציף

- במרחב רציף נייצג את פונקציית המדיניות (Policy) באמצעות התפלגות נורמלית.
- התפלגות נורמלית מיוצגת על ידי שני משתנים: ממוצע (mean) וסטיית תקן (std).

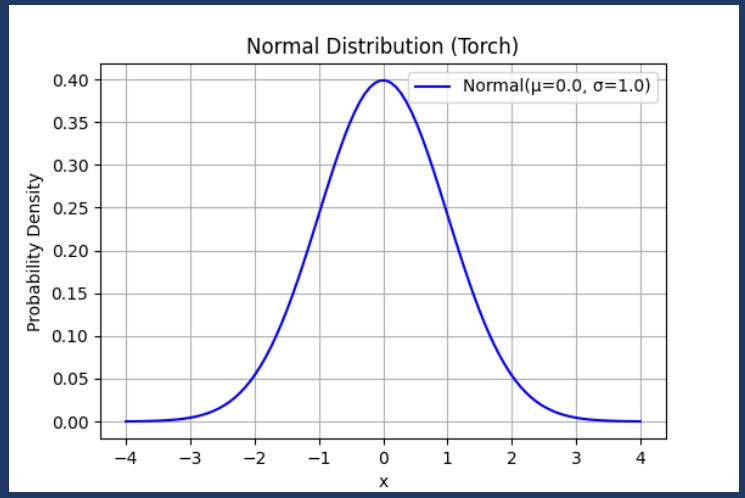


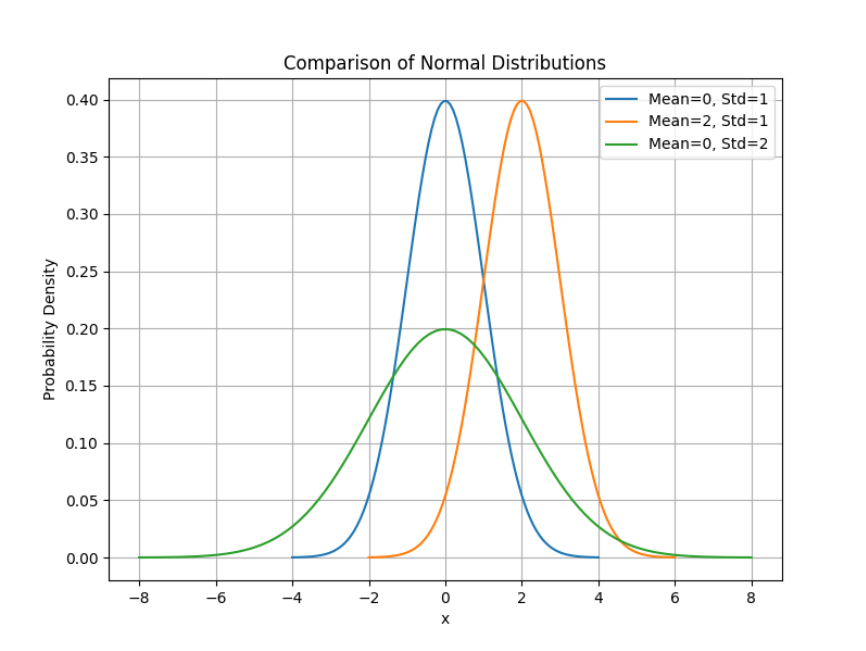
- נוסחה ההתפלגות הנורמלית היא:

$$\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

  - Mean $\mu$ - תוחלת
  - Std $\sigma$ – שונות
- היתרון בהתפלגות זו שהיא נקבעת על פי שני משתנים בלבד. אם ידועים לנו משתנים אלה נוכל לשרטט את הגרף של כל ההתפלגות.



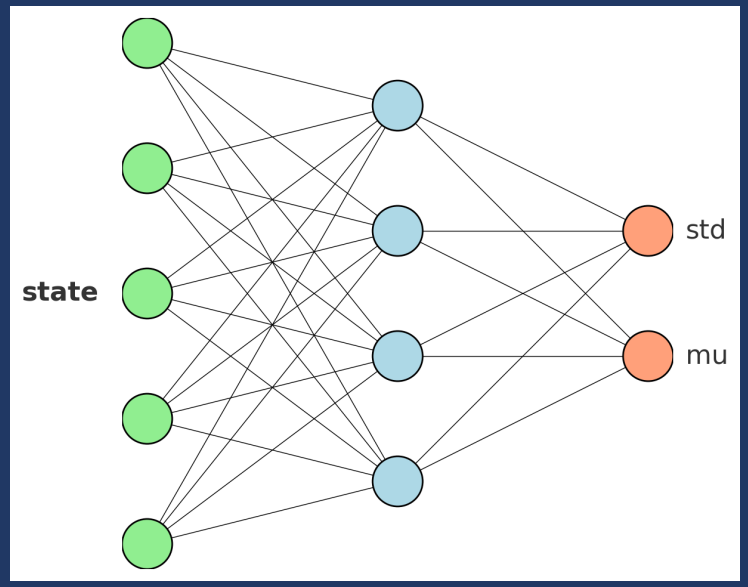
# דגימה מהתפלגות נורמלית



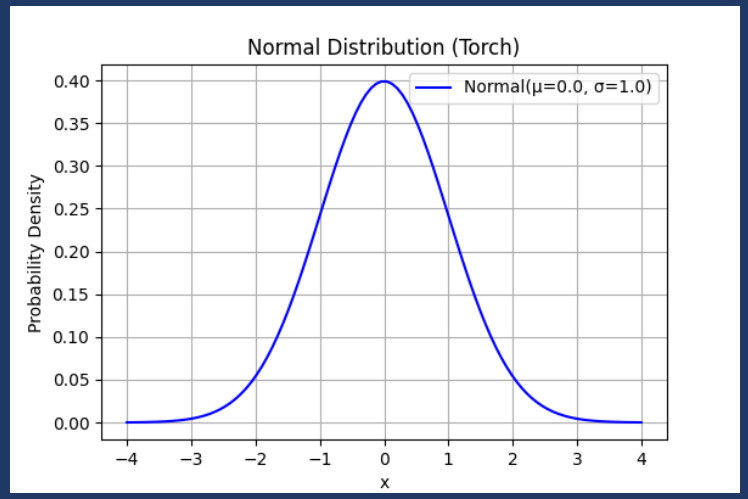

- כאשר אנחנו מבצעים דגימה מהתפלגות נורמלית הסיכוי הגבוה ביותר הוא שנקבל את ערך הממוצע (mean). ערכים הרחוקים מהממוצע נדגום אותם בסיכוי נמוך יותר.
- סטיית התקן (std) קובעת את "רוחב ההתפלגות". אם סטיית התקן קטנה הסיכוי לקבל ערך הקרוב לממוצע הוא גדול יותר.

- למשל, הסיכוי לקבל ערך בין [mean-std, mean+std] הוא 68.27%.

# פונקציית המדיניות במרחב רציף




- השימוש בהתפלגות הנורמלית בפונקציית המדיניות יעשה בצורה הבאה:
  - הקלט יהיה ה state.
  - הפלט יכלול שני ערכים: סטיית תקן, ממוצע.




- הדגימה מההתפלגות הנורמלית תהווה את הפעולה action הנבחרת.
- הפעולה הנבחרת תהיה קרובה לממוצע בהסתברות גבוהה, ורחוקה מהממוצע בהסתברות נמוכה, הכל בהתאם להתפלגות.

# torch distribution Normal



- ספריית torch מספקת לנו אובייקט לטיפול בהסתברות נורמלית.
- בדוגמה הבאה אנחנו יוצרים שתי התפלגויות נורמליות (כל אחת עם ממוצע וסטיית תקן אחרים).
- האובייקט מאפשר לנו לדגום את ההתפלגות, ולקבל log_probs של דגימה מסויימת.

```python
import torch
from torch.distributions import Normal

mean, std = torch.tensor([0.2, -0.1]), torch.tensor([1.5, 1.2])
dist = Normal(mean, std)

sample = dist.sample()        # Normal sampling
sample = dist.rsample()       # Normal sampling with gradients

prob = dist.log_prob(x)       # Log probability of value x
entropy = dist.entropy()      # Entropy of the distribution
```

# מימוש רשת הנוריונים

```python
class REINFORCE_Network (nn.Module):
    def __init__(self, state_dim, action_dim, lr=0.00001, fc_dims=256, chkpt=1):
        super().__init__()
        self.linear1 = nn.Linear(state_dim, fc_dims)
        self.relu = nn.ReLU()
        self.linear2 = nn.Linear(fc_dims, fc_dims)
        self.mean_layer = nn.Linear(fc_dims, action_dim)
        self.std_layer = nn.Linear(fc_dims, action_dim)
        self.lr = lr
        self.optimizer = optim.Adam(self.parameters(), lr=lr)
        self.device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
        self.to(self.device)
        self.checkpoint_file = f'Data/REINFORCE{chkpt}.pth'

    def forward(self, state):
        x = self.relu(self.linear1(state))
        x = self.relu(self.linear2(x))

        mean_raw = self.mean_layer(x)
        thrust_mean = (torch.tanh(mean_raw[:, 0]) + 1) / 2  # Thrust in [0,1]
        spin_mean = torch.tanh(mean_raw[:, 1])              # Spin in [-1,1]
        mean = torch.stack([thrust_mean, spin_mean], dim=1)
        std = F.softplus(self.std_layer(x)) + 1e-6          # Ensure std is positive
        return mean, std
```

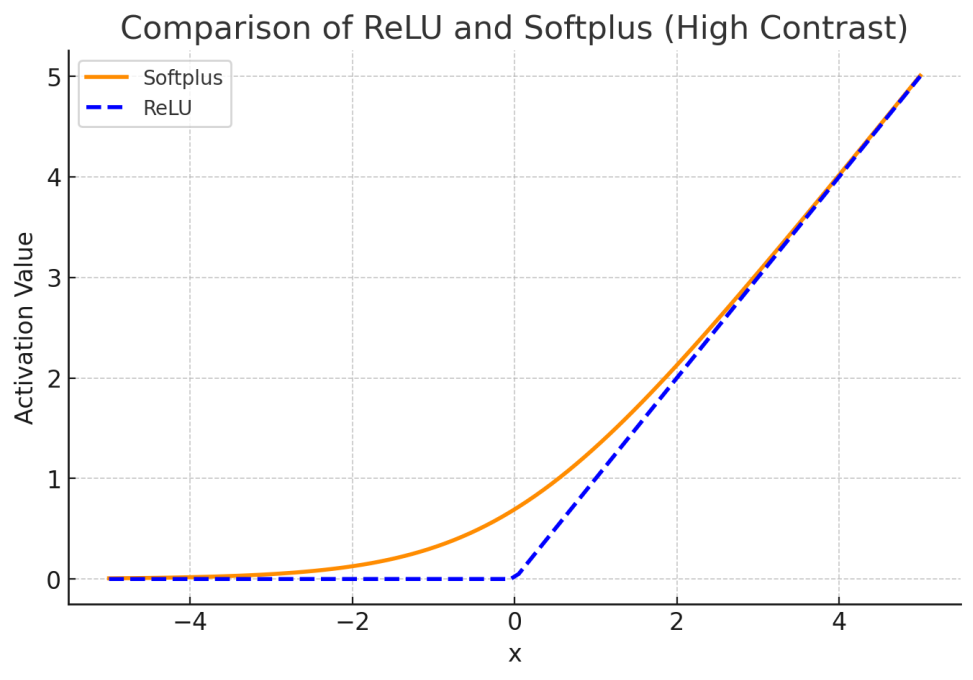
- הפלט שלנו הוא כפול thrust, spin:
  - 2 ממוצעים.
  - 2 סטיות תקן.
- אנחנו מגבילים את הממוצע לטווח של הפעולות שלנו באמצעות הפונקציה tanh:
  - Thrust = (0, 1)
  - Spin = (-1, 1)
- לבסוף, אנו דואגים שסטיית תקן תהיה תמיד חיובית באמצעות softplus.

# הפונקציה softplus

- הפונקציה softplus מוגדרת כך:

$$softplus(x) = \ln(1 + e^x)$$

- היתרון שלה על פני ReLu שהיא גזירה גם ב 0 ויוצרת לנו גרף חיובי עם עקומה חלקה יותר.



# הפעולה getAction

- הפעולה getAction מקבלת state ומחזירה לנו את הפעולות : thrust, spin, באמצעות דגימה מההתפלגות הנורמלית שרשת הנוירונים יוצרת לנו.
- אם אנחנו באימון נחזיר גם את ערכי המצערת והסיבוב וגם את אובייקט ה dist.

```python
def get_action(self, state, events = None, train = False):
    state_tensor = state.to(self.policy.device).view(1,-1)
    with torch.no_grad():
        mean, std = self.policy(state_tensor)
    mean = mean.squeeze(0)
    std = std.squeeze(0)
    dist = Normal(mean, std)
    action = dist.rsample()
    action = action.clamp(min=torch.tensor([0, -1], device=self.policy.device),
                          max=torch.tensor([1, 1], device=self.policy.device))

    thrust, spin = action[0].item(), action[1].item()

    if train:
        return (thrust, spin), action
    return thrust, spin
```

- נשים לב, שהרשת מספקת לנו שני ערכים לכל פעולה: mean, std.
- כיוון שדגימה מההתפלגות הנורמלית עלולה להביא לנו ערכים מחוץ לטווח של הפעולות, אנחנו תוחמים אותם באמצעות הפונקציה clamp.

# הפעולה getDist

- הפעולה get_dist מקבלת את המצב state ומחזירה אובייקט של התפלגות נורמלית.
- פעולה זו היא חלק מגרף החישוב ונועדה לעדכון רשת הנוירונים.

```
def get_dist (self, states_tensor):
    states_tensor = states_tensor.to(self.policy.device)
    mean, std = self.policy(states_tensor)
    dist = Normal(mean, std)
    return dist
```

# אימון הסוכן trainer

```python
class REINFORCE_Trainer:
    def __init__(self,  chkpt):
        self.agent:REINFORCE_Agent = REINFORCE_Agent(chkpt=chkpt)
        self.env:SpaceShipRide = SpaceShipRide()
        self.chkpt = chkpt

    def train(self, epochs=100000):
        self.init_wandb(project_name="REINFORCE-continuous",chkpt=self.chkpt,resume=False)
        wins = 0
        for epoch in range(epochs):
            self.env.restart()
            self.env.step = 0
            score = 0
            self.env.done = False
            ########### sample environment ###########
            while not self.env.done:
                self.env.step += 1
                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        return
                pygame.event.pump()
                state = self.env.get_state()
                action, action_tensor = self.agent.get_action(state, train=True)
                self.env.render(action)
                reward = self.env.get_reward()

                #reward2 += reward1
                self.agent.remember(state, action_tensor, reward)
                score += reward


            ########### train ###########
            self.agent.learn()
```

- אין כל שינוי באימון רשת הנוירונים במקרה של מרחב פעולות בדיד או רציף.
- נחזור על אלגוריתם האימון:
  - אנחנו דוגמים את הסביבה עד לסיום המשחק.
  - שומרים את הדגימות בזכרון memory.
  - בסיום של משחק מבצעים אימון של רשת הנוירונים באמצעות הפונקציה learn.

# הפונקציה learn

```python
def learn(self):

    states, actions, rewards = self.memory.get_tensors()

    G = self.compute_G(rewards)

    dist = self.get_dist(states)
    log_probs = dist.log_prob(actions).sum(dim=-1)

    # Compute the loss using vectorized operations
    policy_loss = -(G * log_probs).mean()

    # Add Entropy regularization
    entropy = dist.entropy().mean()
    loss = policy_loss - self.entropy_coe * entropy


    # backward
    self.policy.optimizer.zero_grad()
    loss.backward()

    self.policy.optimizer.step()

    #### for logging
    self.sum_loss += loss.detach()
    self.sum_entropy += dist.entropy().detach().mean()
```

- הפונקציה learn בתוך ה Agent מבצעת את אימון רשת הנוירונים בהתאם לאלגוריתם REINFORCE.
- כפי שניתן לראות, אין הבדל בין אימון במרחב רציף או בדיד.
- שימו לב, שאובייקט ה dist במקרה זה הוא מסוג התפלגות נורמלית ולא התפלגות בדידה כמו בדוגמה הקודמת.

# תוצאות האימון