UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID Official Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Chapter 5. Deep Reinforcement Learning and Generative Models

Neural Networks

2022/2023

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Departamento de Sistemas Informáticos y Computación

Index

Index

Reinforcement Learning

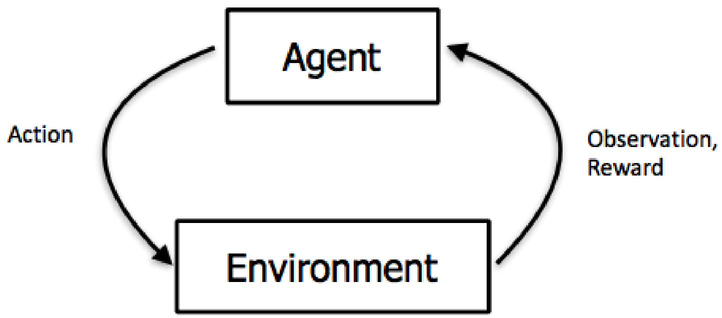
- Goal: intelligent systems that interact with the environment
- Improving over time through trial and error
- RL have had some success in the past but still limited
- Deep RL rely on:
  - Neural networks as function approximation
  - Representation Learning
- Deep learning enables RL to scale to problems that were previously intractable:
  - settings with high-dimensional states (directly from pixels)
  - extremely huge action spaces

Reinforcement Learning

- Some examples:
  - Breakout: https://www.youtube.com/watch?v=V1eYniJ0Rnk
  - Super Mario Bros: https://www.youtube.com/watch?v=qv6UVOQ0F44

Reinforcement Learning

- Learning through interaction
- Trial and error learning
- Agent is a machine learning algorithm



Reinforcement Learning

RL as a Markov Decision Process (MDP):

- $\mathcal{S}$ set of states

- $p(s_0)$ a distribution of starting states

- $\mathcal{T}$ transitions, $\mathcal{T}(s_{t+1} \mid s_t, a_t)$

- $\mathcal{R}$ reward function, $\mathcal{R}(s_t, a_t, s_{t+1})$

- γ is a discount factor, $\gamma \in [0, 1]$

Key concept: only s_t affects s_{t+1} no further past has to be considered

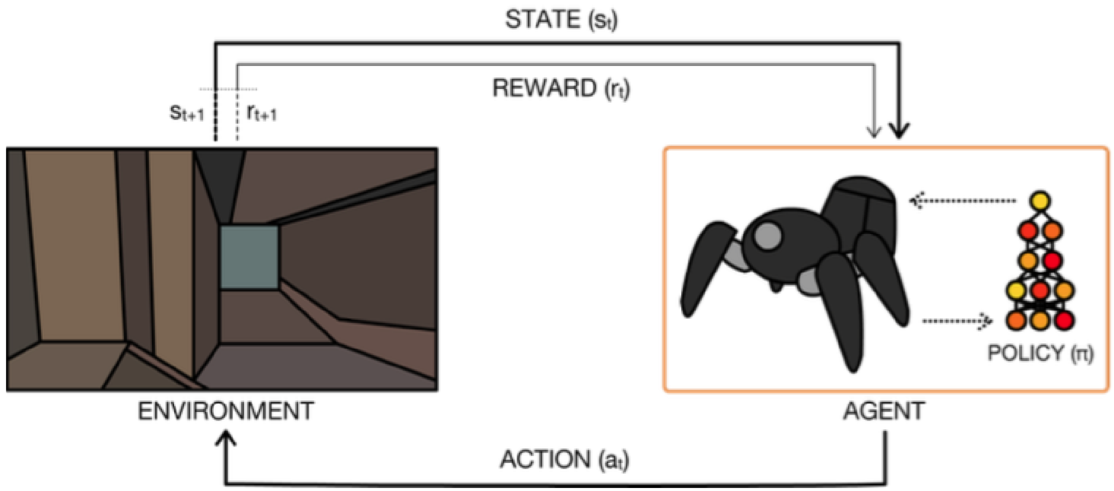
Reinforcement Learning

At time step t:

- Agent observes the environment state s_t
- Agent take an action a_t following a (stochastic) policy:

$$\pi : \mathcal{S} \rightarrow p(\mathcal{A} = a_t \mid \mathcal{S} = s_t)$$

- Agent and environment transition to new state s_{t+1}
- Environment provides a new reward r_{t+1}



- Goal: learn a *policy* that maximizes the cumulative reward

Reinforcement Learning

- Total Reward for T actions:

$$R = \sum_{t=0}^{T} \gamma^t r_{t+1}$$

- The goal of RL is to find an optimal policy π^*:

$$\pi^* = \underset{\pi}{\text{argmax}}\, \mathbb{E}[R \mid \pi]$$

given an MDP, π^* exists.

Reinforcement Learning

- The optimal policy must be inferred by trial-and-error interaction with the environment. The only learning signal the agent receives is the reward
- The observations of the agent depend on its actions and can contain strong temporal correlations
- Agents must deal with long-range time dependencies: Often the consequences of an action only materialize after many transitions of the environment

Reinforcement Learning

- RL Algorithms:
  - Value functions: Indirect estimation of policy
  - Policy Search: Direct estimation of policy

Index

Value Functions Algorithms

- Estimate the value (expected return) of being in a given state
- The state-value function:

$$V^{\pi}(s) = \mathbb{E}[R|s, \pi]$$

- the total reward starting in state s and following policy π
- the optimal state-value is the state-value with the optimal policy:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}$$

- $\boxed{\text{if we had } V^{*} \text{ we could retrieve } \pi^{*} \text{ by means of:}}$

$$\pi^{*} = \underset{\pi}{\text{argmax}}\, \mathbb{E}_{s_{t+1} \sim \mathcal{T}(s_{t+1}|s_t, a)}[V^{*}(s_{t+1})]$$

Value Functions Algorithms

- In this expression:

$$\pi^* = \underset{\pi}{\text{argmax}}\, \mathbb{E}_{s_{t+1} \sim \mathcal{T}(s_{t+1}|s_t,a)}[V^*(s_{t+1})]$$

  $\mathcal{T}$ is unknown.

- we define $Q^\pi(s,a)$, the state-action-value or quality function:

$$Q^\pi(s,a) = \mathbb{E}[R \mid s, a, \pi]$$

- $\boxed{\text{if we had } Q^\pi(s,a) \text{ the best action of the policy would be:}}$

$$a^* = \underset{a}{\text{argmax}}\, Q^\pi(s,a)$$

Value Functions Algorithms

How to learn $Q^\pi(s,a)$:

- Dynamic Programming (Bellman equation):

$$Q^\pi(s,a) = \mathbb{E}[r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi(s_{t+1})) \mid s = s_t, a = a_t]$$

- optimal values decompose into a Bellman equation:

$$Q^*(s,a) = \mathbb{E}[r_{t+1} + \gamma \max_a Q^*(s_{t+1}, a) \mid s = s_t, a = a_t]$$

Value Functions Algorithms

How to learn $Q^*(s,a)$:

- Q-learning optimal values, iteratively:

$$Q(s,a) = Q(s,a) + \alpha\delta$$

- δ is the Temporal Difference error:

$$\delta = Y - Q(s,a)$$

in step t

$$Y = r_{t+1} + \gamma \max_a Q(s_{t+1}, a)$$
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Value Functions Algorithms

- Q-learning algorithm:

```
Output: Q
  initialize Q arbitrarily, e.g., to 0
  Repeat
    select s as an initial state
    while(state s is not terminal) do
      a = action for s derived by Q (epsilon-greedy)
      take action a, observe r, sn
      Q(s,a)=Q(s,a)+alfa*[r+gamma*max_an(Q(sn, an))-Q(s,a)]
      s=sn
    end
  Until convergence
end
```

Value Functions Algorithms

v										
	Start									
	0	1	2	3	4	5	6	7	8	9

Actions:

- R: right
- JR: Jump right

Inital state: v(0)

Reward(s)

R										
	Start									
	1	1	1	-10	1	1	-10	1	1	10

Value Functions Algorithms

v	Start									
	0	1	2	3	4	5	6	7	8	9

Q function initial

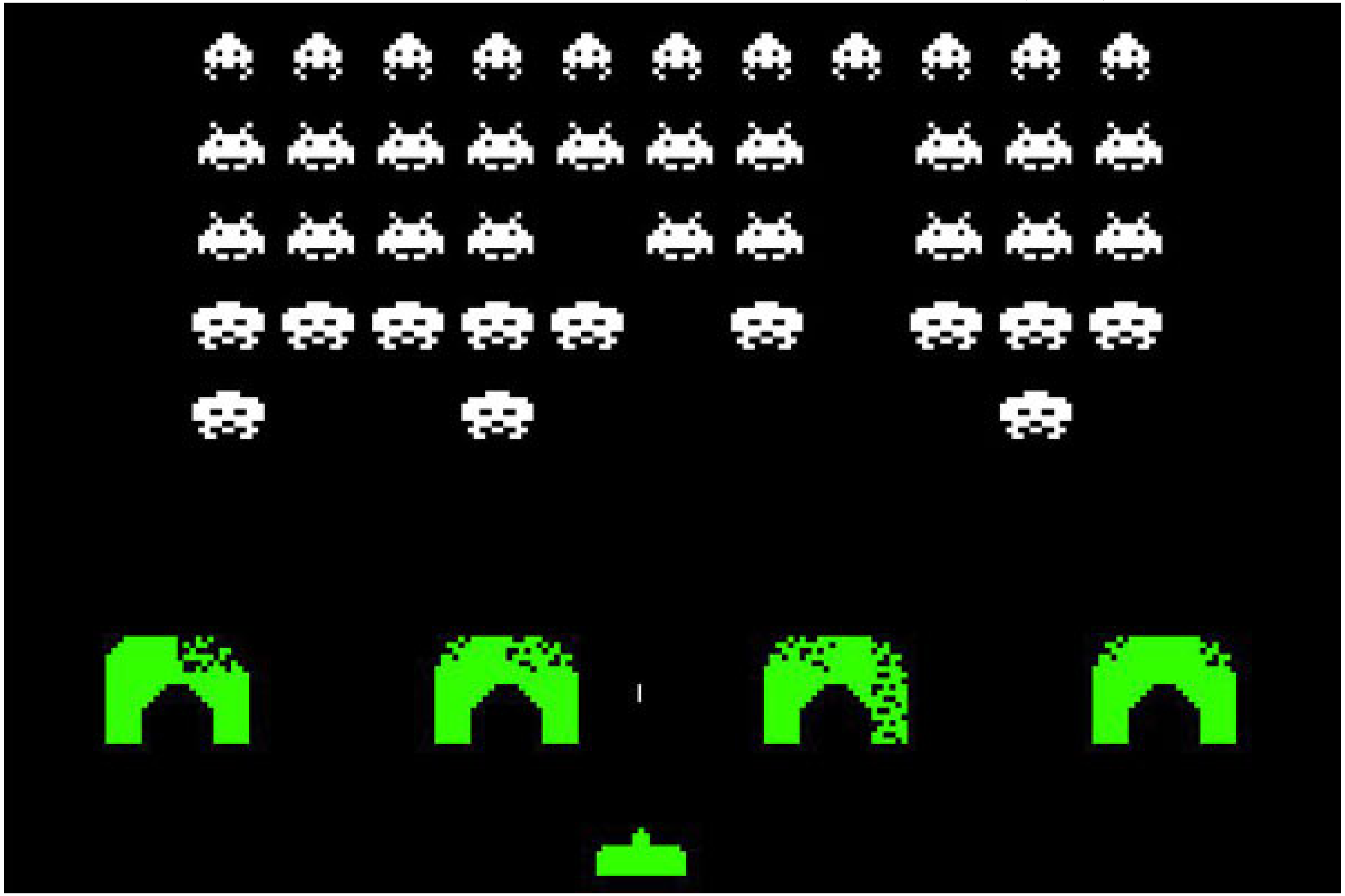
s|a	R	JR
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

Q function obtained

s|a	R	JR
0	10	10
1	10	-9.5
2	-10	10
3	0	0
4	10	-10
5	-10	10
6	0	0
7	10	10
8	10	-8.3
9	0	0

Value Functions Algorithms

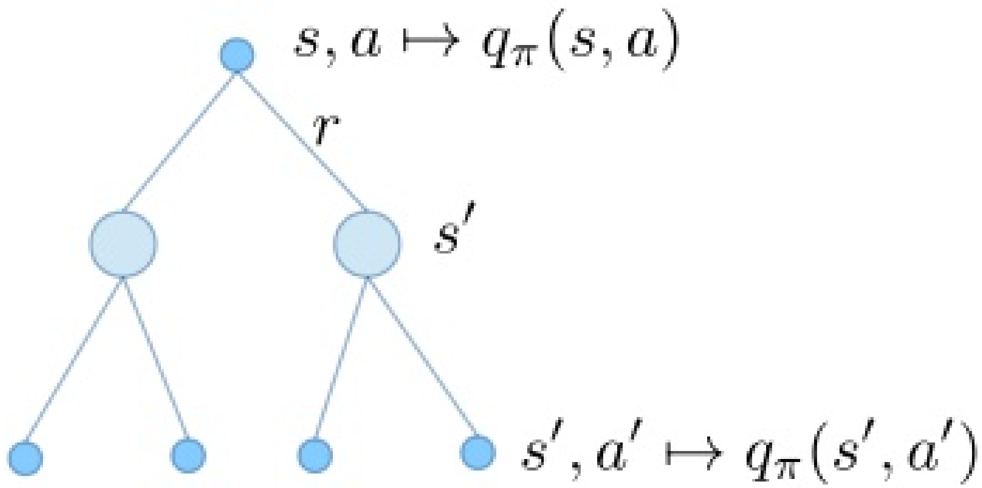
How to define the *state* of the agent? How to obtain $Q(s, a)$?



Value Functions Algorithms

Alternative way to learn $Q^{\pi}(s, a)$:

- Monte Carlo methods estimate the expected return from a state $\mathbb{E}[R \mid s, a, \pi]$ by averaging the return from multiple roll-outs of a policy



Index

Policy Search

- No need a value function model
- Directly search for an optimal policy π^*
- Typically **a parameterized policy** and parameters (Φ) are updated to maximize $\mathbb{E}[R|\Phi]$
- Neural networks can be used to encode policies

Policy Search

Neural network outputs:

- **Continuous actions**
  - parameters of a probability distribution
  - e.g., mean and standard deviations

- **Discrete actions**
  - individual probabilities
  - multinomial distribution

In any case, actions are obtaining by sampling from these distributions that essentially define the transition dynamics $\mathcal{T}$

Policy Gradients

- Compute the expected return of a given policy in order to obtain the gradients w.r.t Φ:

$$\mathbb{E}[R \mid \pi]$$

$$\nabla_\Phi \mathbb{E}[R] = \mathbb{E}[R \nabla_\Phi log \pi(a_t \mid s_t; \Phi)]$$

Update Φ by gradient ascent in the direction of:

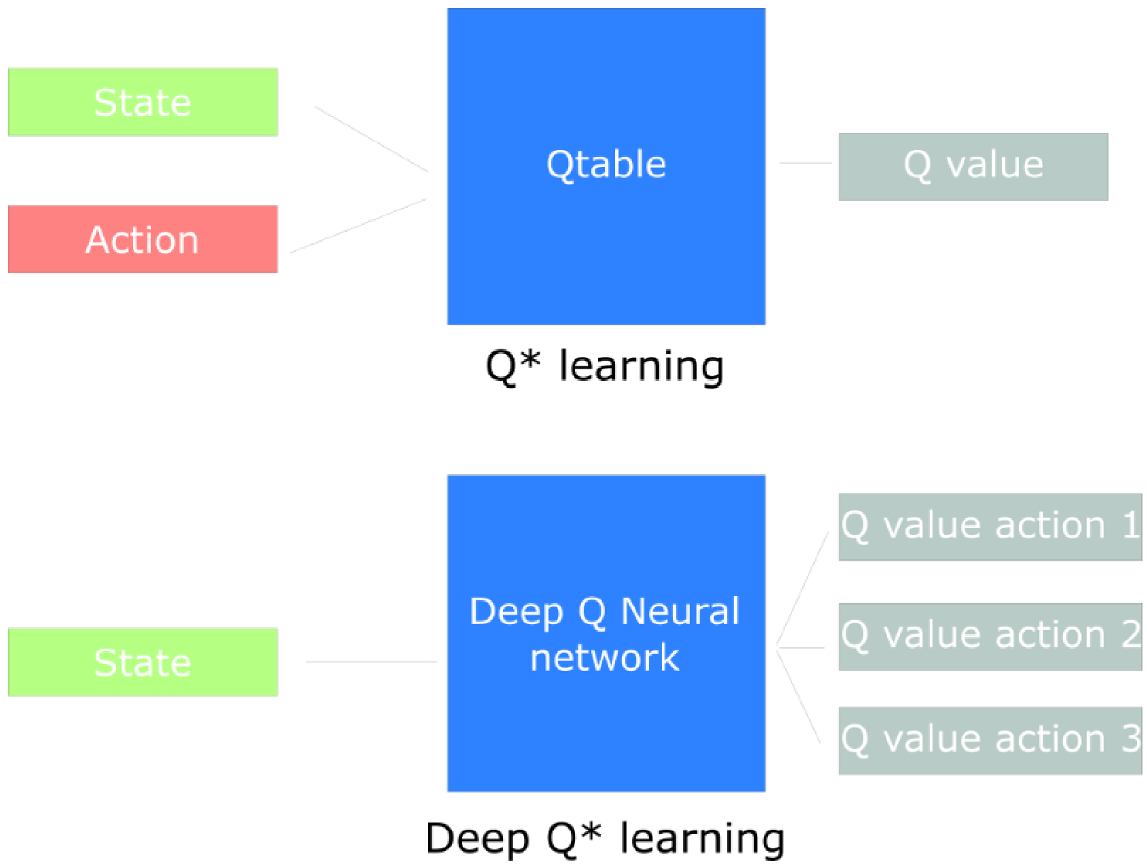
$$R \nabla_\Phi log \pi(a_t \mid s_t; \Phi)$$

Index

Deep Reinforcement Learning

- Use a Deep Neural Network as a universal approximation function for:
  - state-action-value functions $Q(s, a)$
  - policy functions $a = \pi(s)$

Deep Q-Networks (DQN)



Deep Q-Networks (DQN)

- Universal function approximation

$$Q(s,a) \approx Q(s,a,\mathbf{w})$$

  where $\mathbf{w}$ are the weight of the neural network

- Train using temporal difference:

$$Q(s_t,a_t) = \mathbb{E}[r_{t+1} + \gamma \max_a Q(s_{t+1}, \pi(s_{t+1}))]$$

- Therefore:
  - Input: (s_t, a_t) (just only s_t)
  - Target: $r_{t+1} + \gamma \max_a Q(s_{t+1}, a)$
  - Minimize MSE

$$(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))^2$$

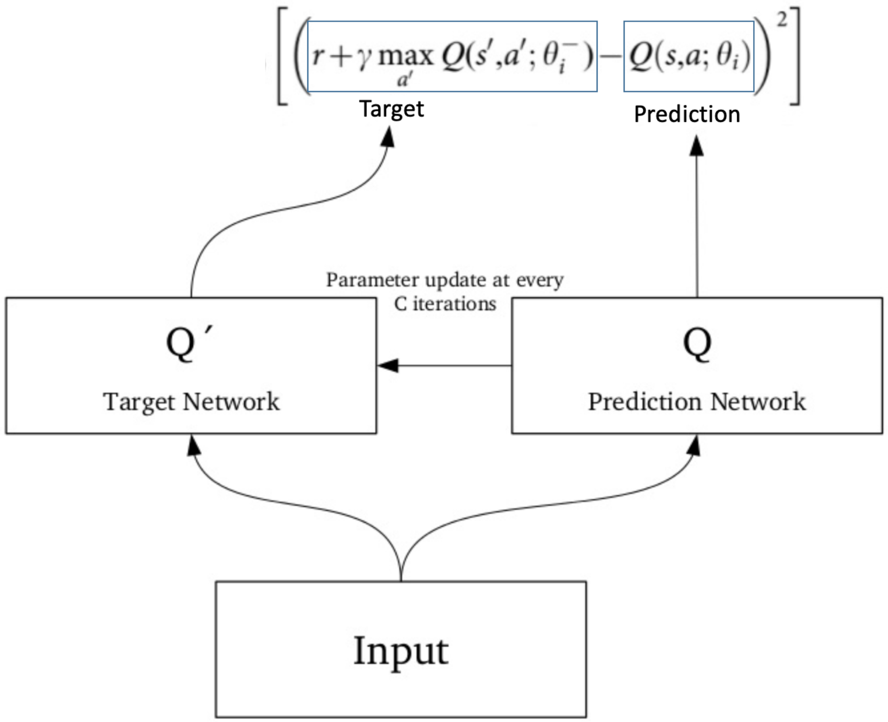
Deep Q-Networks (DQN)

- Experience replay
- re-use the tuples $s_t, a_t, r_{t+1}, s_{t+1}$ as a data set
- multiple passes with the same data is beneficial

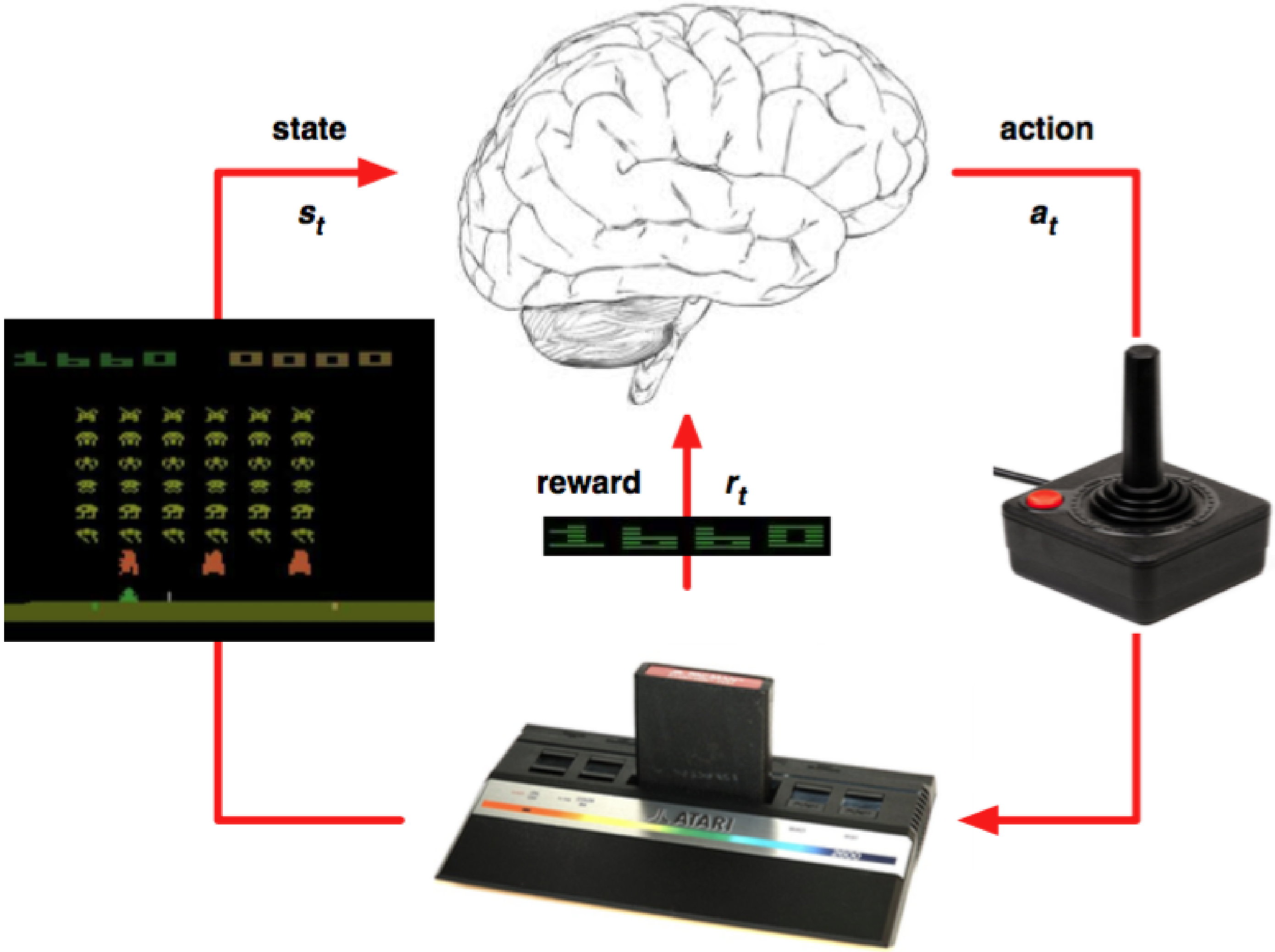
Deep Q-Networks (DQN)

- Non-stationary targets

$$\delta = r_{t+1} + \gamma \max_a Q(s_{t+1}, a, \mathbf{\Phi}^-) - Q(s_t, a_t, \Phi)$$

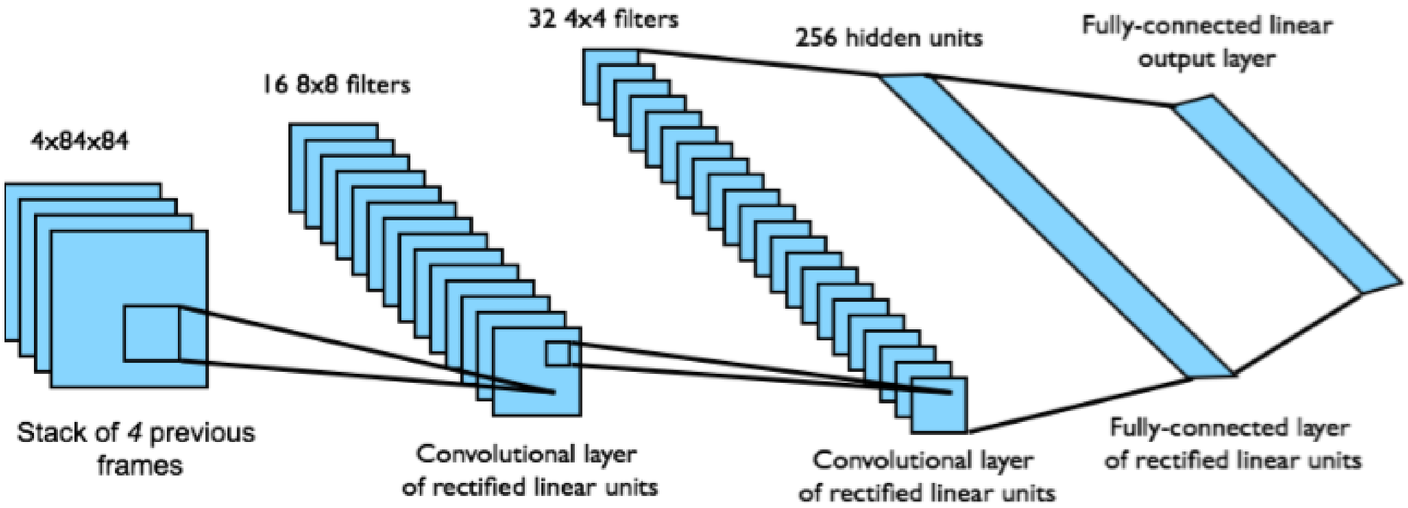


Deep Q-Networks (DQN)



Deep Q-Networks (DQN)

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

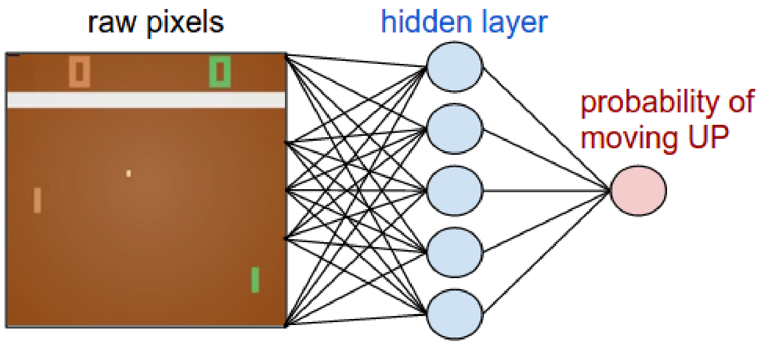
Deep Policy Networks

Policy gradients

- Update Φ by gradient ascent in the direction of:

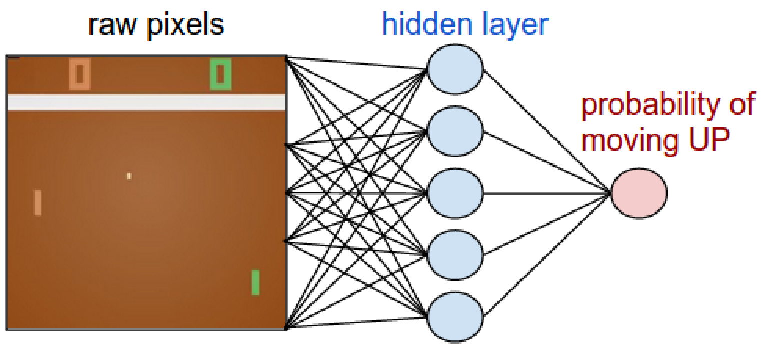
$$R\nabla_{\Phi} log\pi(a_t \mid s_t; \Phi)$$

- where s could be an image, and a a particular action, e.g moving up



Deep Policy Networks

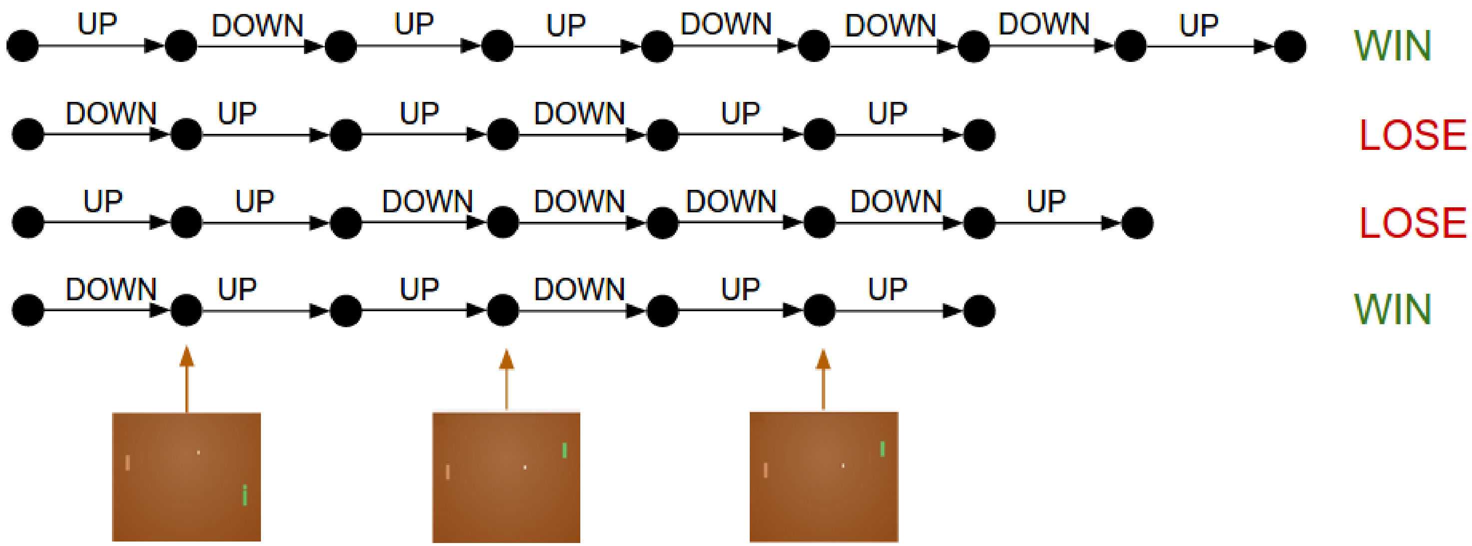
- High dimensional state representation: a frame with $210 * 160$ potential positions



- State must consider two consecutive frames

Deep Policy Networks

- Delayed reward:



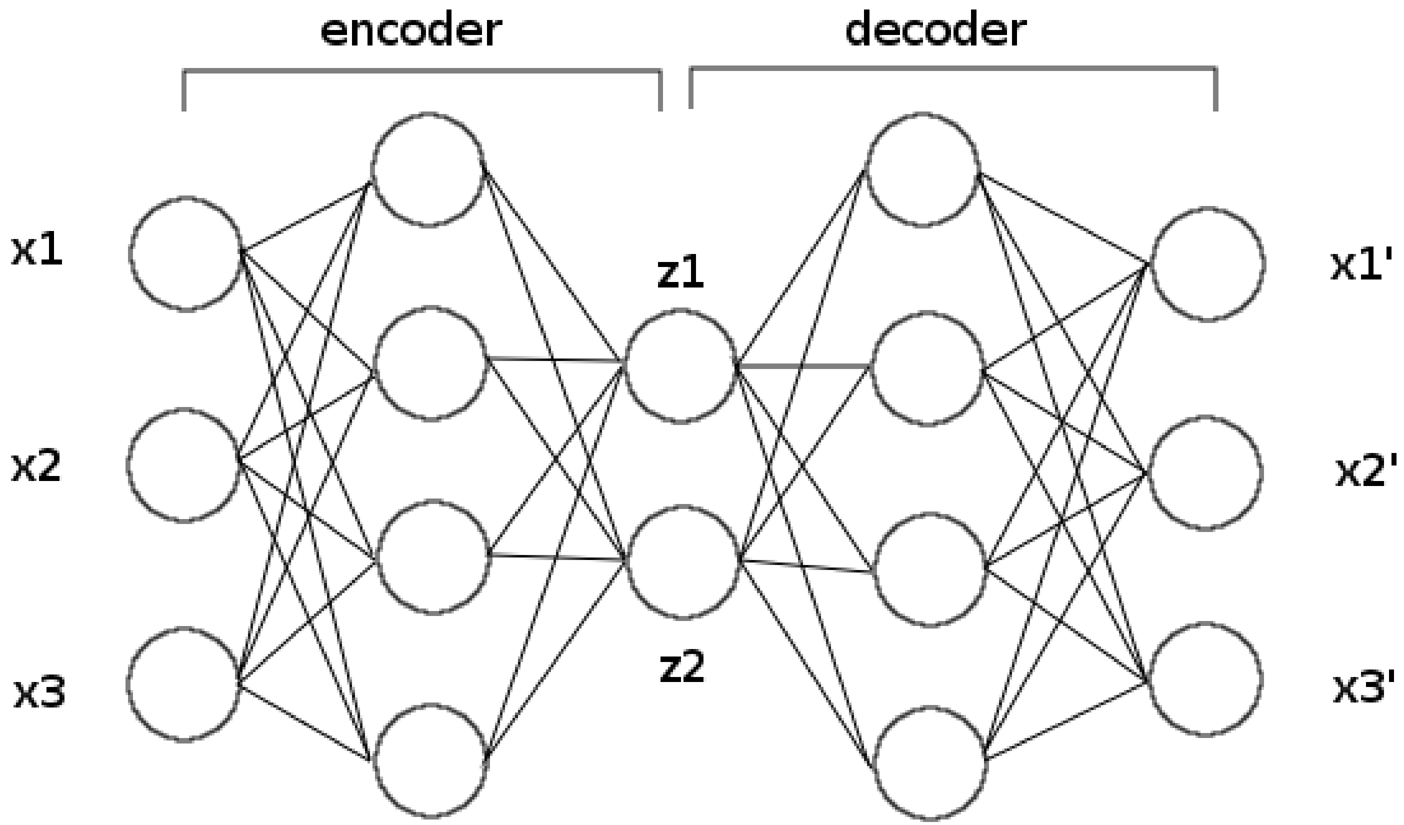
Index

Generative Models

- Autoencoders
- Denoising Autoencoders
- Variational Autoencoders
- Generative Adversarial Networks

Autoenconder

- Feed-forward neural network that reproduces the input at the output layer



- Training by minimizing the quadratic error

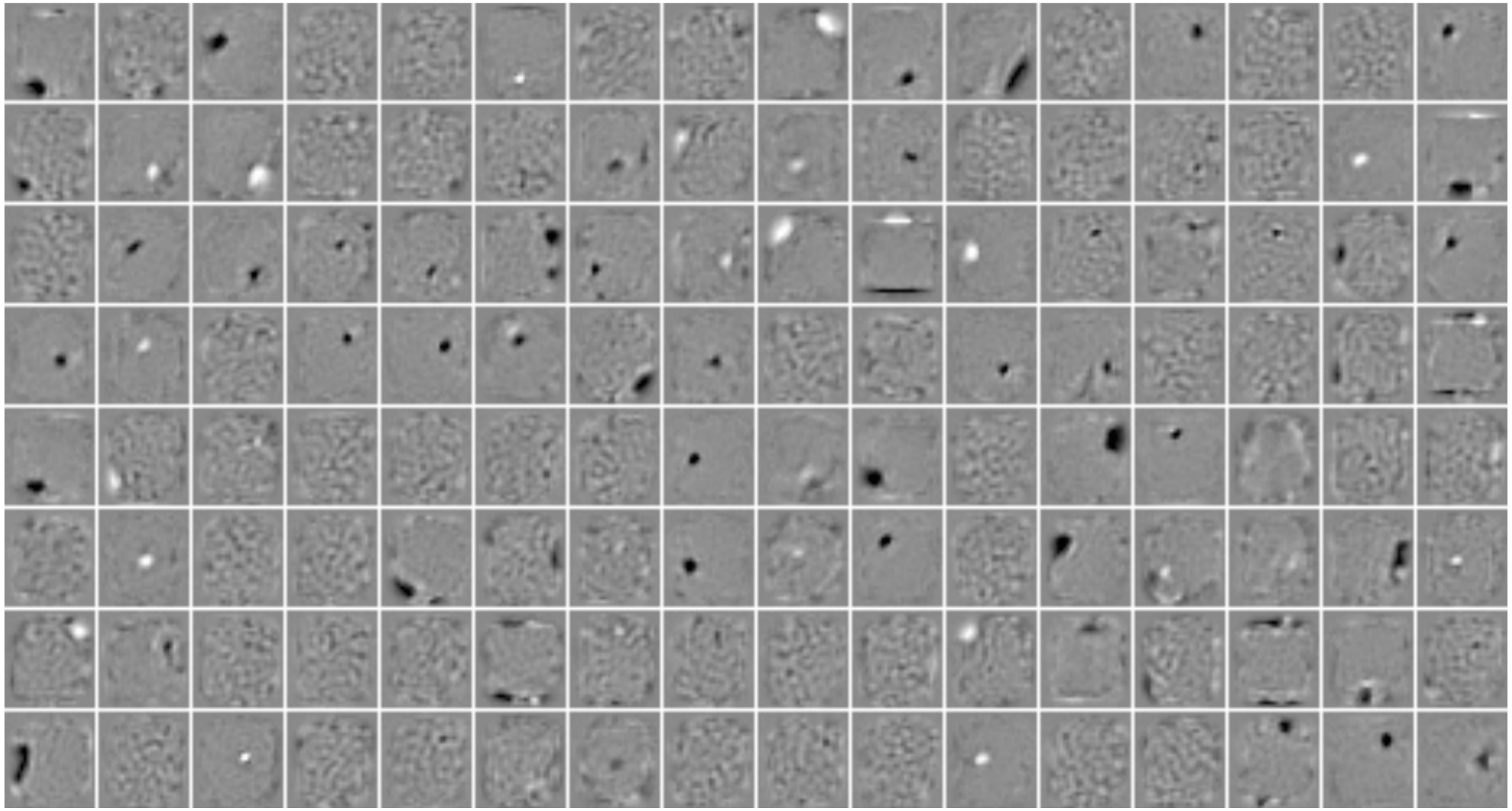
Autoencoders - Topology

- Undercomplete:
  - $dim(\mathbf{z}) < dim(\mathbf{x})$
  - compression
  - probably meaningful representation. **Latent space**

- Overcomplete:
  - $dim(\mathbf{z}) > dim(\mathbf{x})$,
  - hidden unit just copy input units
  - no meaningful representation
  - have to force sparsity

Autoencoders - weights first layer

- Training over MNIST digits



Denoising-Autoencoders

- Ideal representations should be robust to noise

- A denoising autoencoder is a feed-forward neural network that reproduce the input at the output layer:
  - One input layer $\tilde{\mathbf{x}}$
  - One hidden layer $\mathbf{h}$
  - One output layer layer $\hat{\mathbf{x}}$

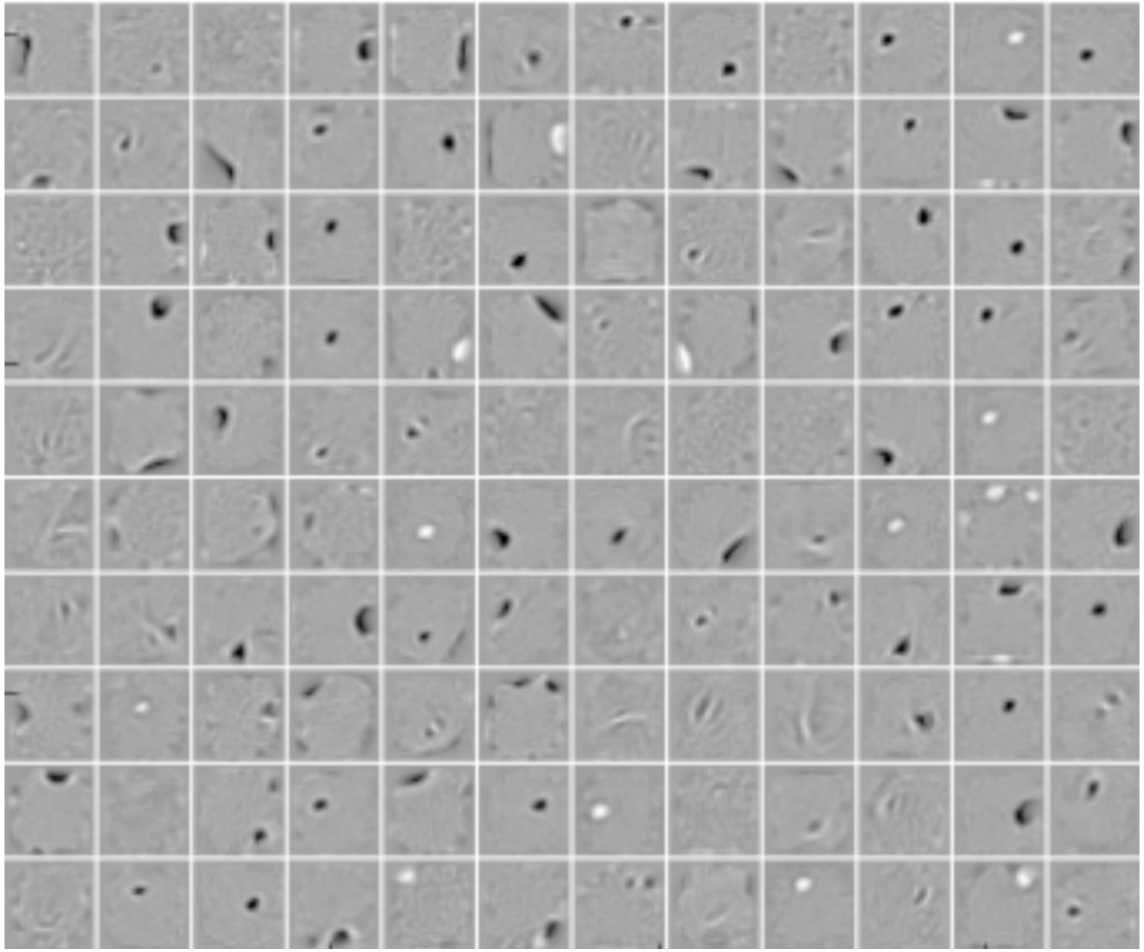
  where $\tilde{\mathbf{x}}$ is a **corrupted** version of $\mathbf{x}$

- The output $\hat{\mathbf{x}}$ is computed from a corrupted version of $\mathbf{x}$

- But the loss function compares $\hat{\mathbf{x}}$ and $\mathbf{x}$

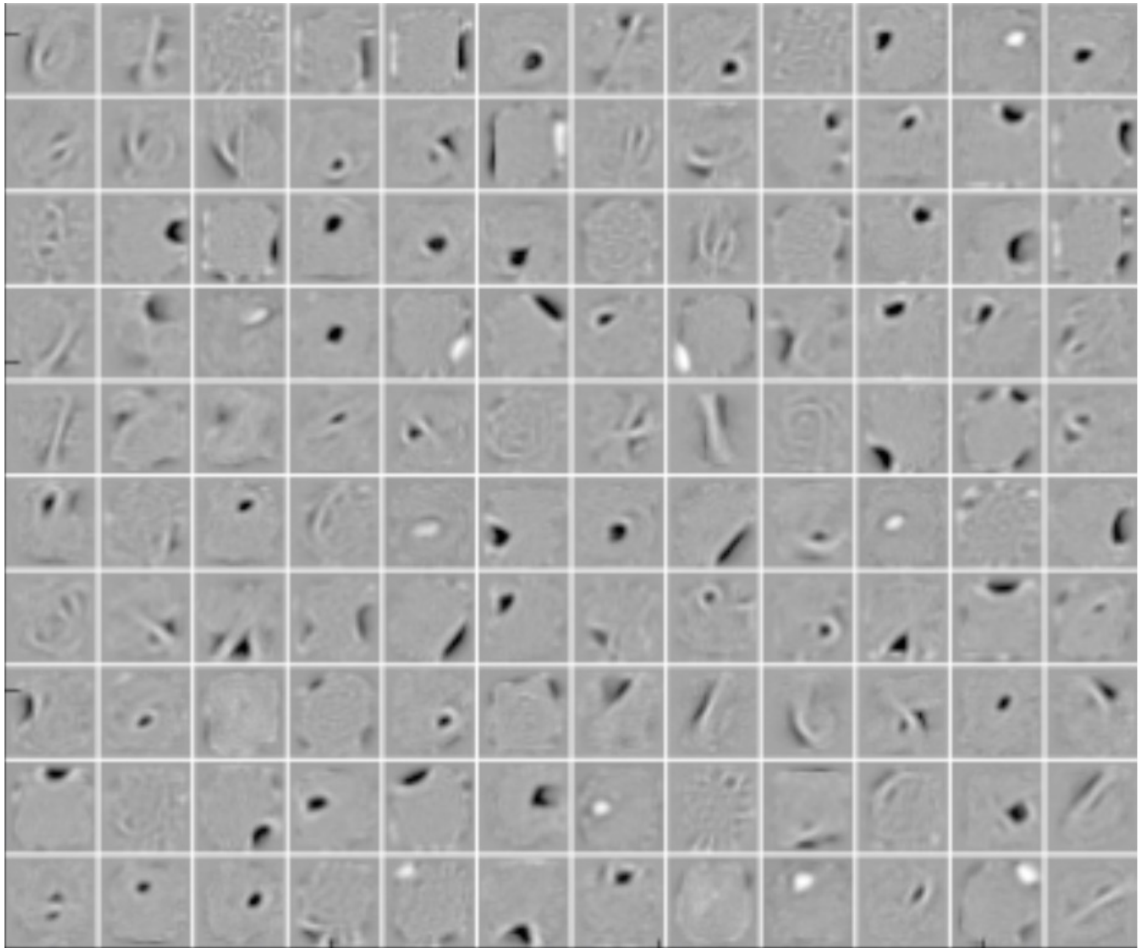
Denoising-Autoencoders

- MNITS with 25% of noise



Denoising-Autoencoders

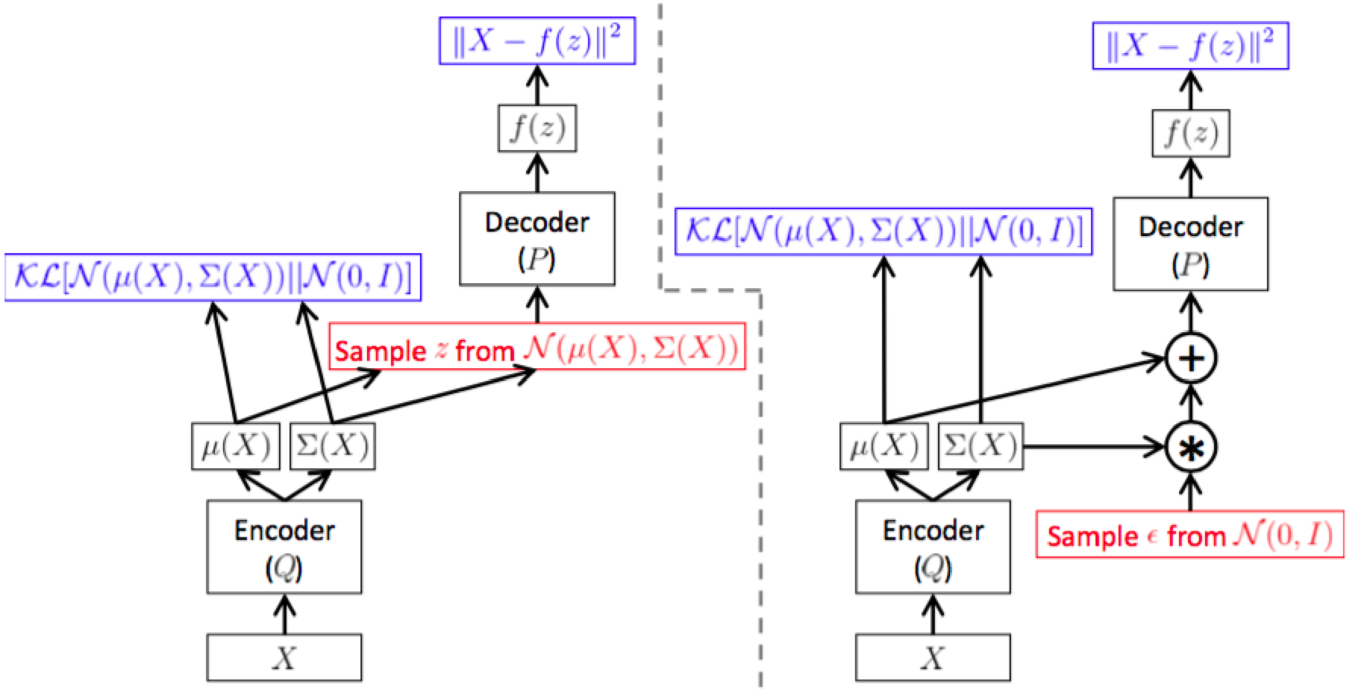
- MNITS with 50% of noise



Index

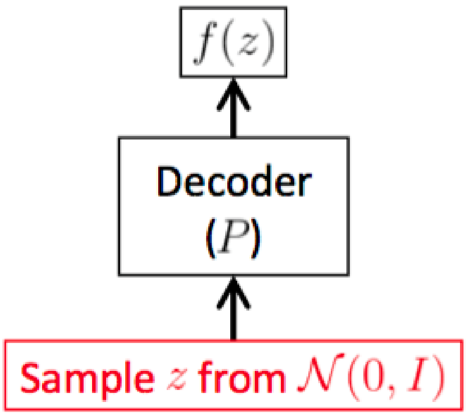
Variational Autoencoders

- The latent variables are drawn from a prior $p(\mathbf{z})$
- https://arxiv.org/pdf/1606.05908.pdf



Variational Autoencoders, Generation process

- Draw from $p(\mathbf{z})$



Variational Autoencoders, Generation process

- Encoder is a Convolutional Network
- Decoder is a De-Convolutional (convolutional transpose) Network

Index

Generative Adversarial Networks

- A Generator Network G
- A Discriminator Network D
- During training G is trained to D
- During training D is trained discriminate between real data generated data (fake)

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

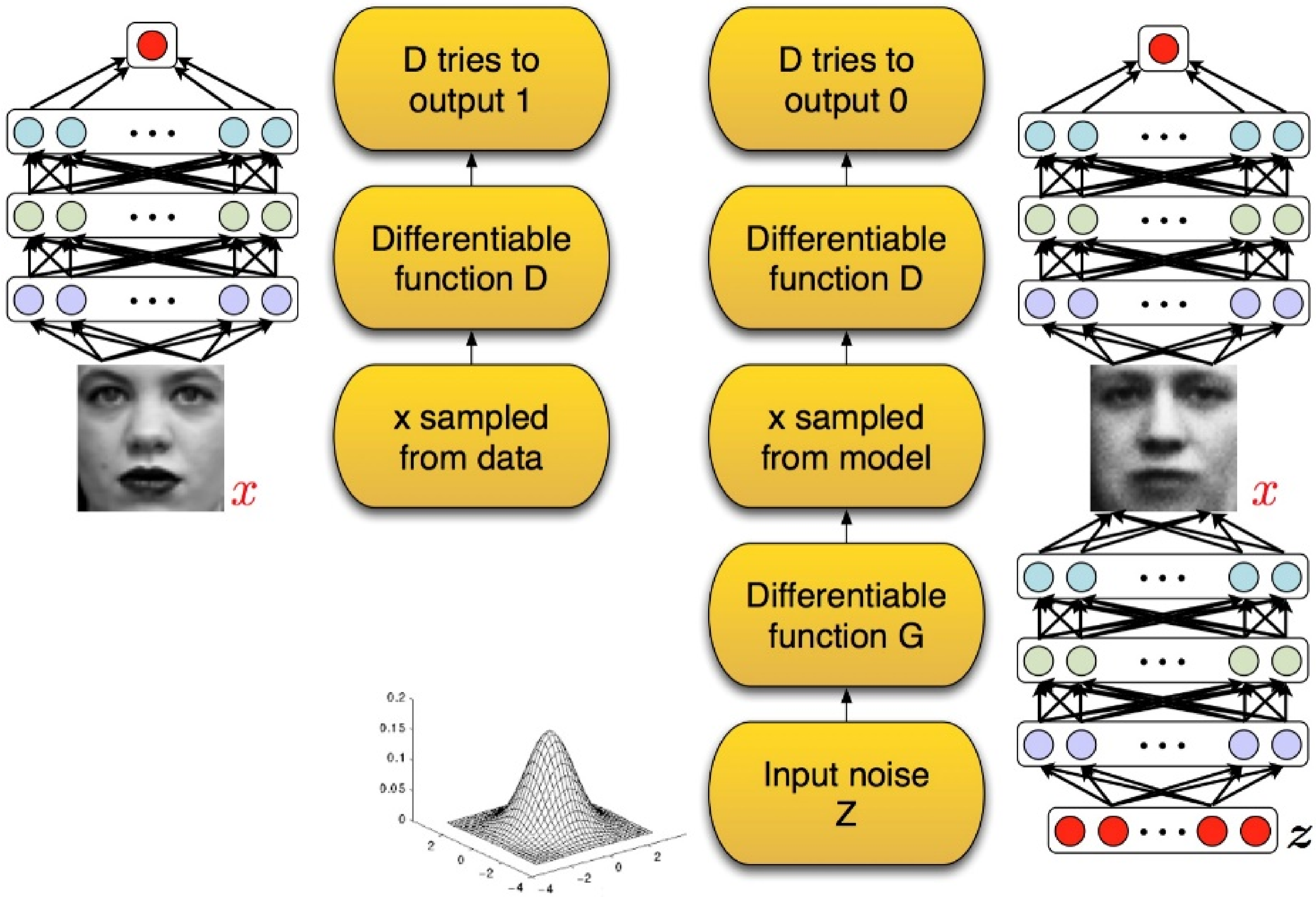
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

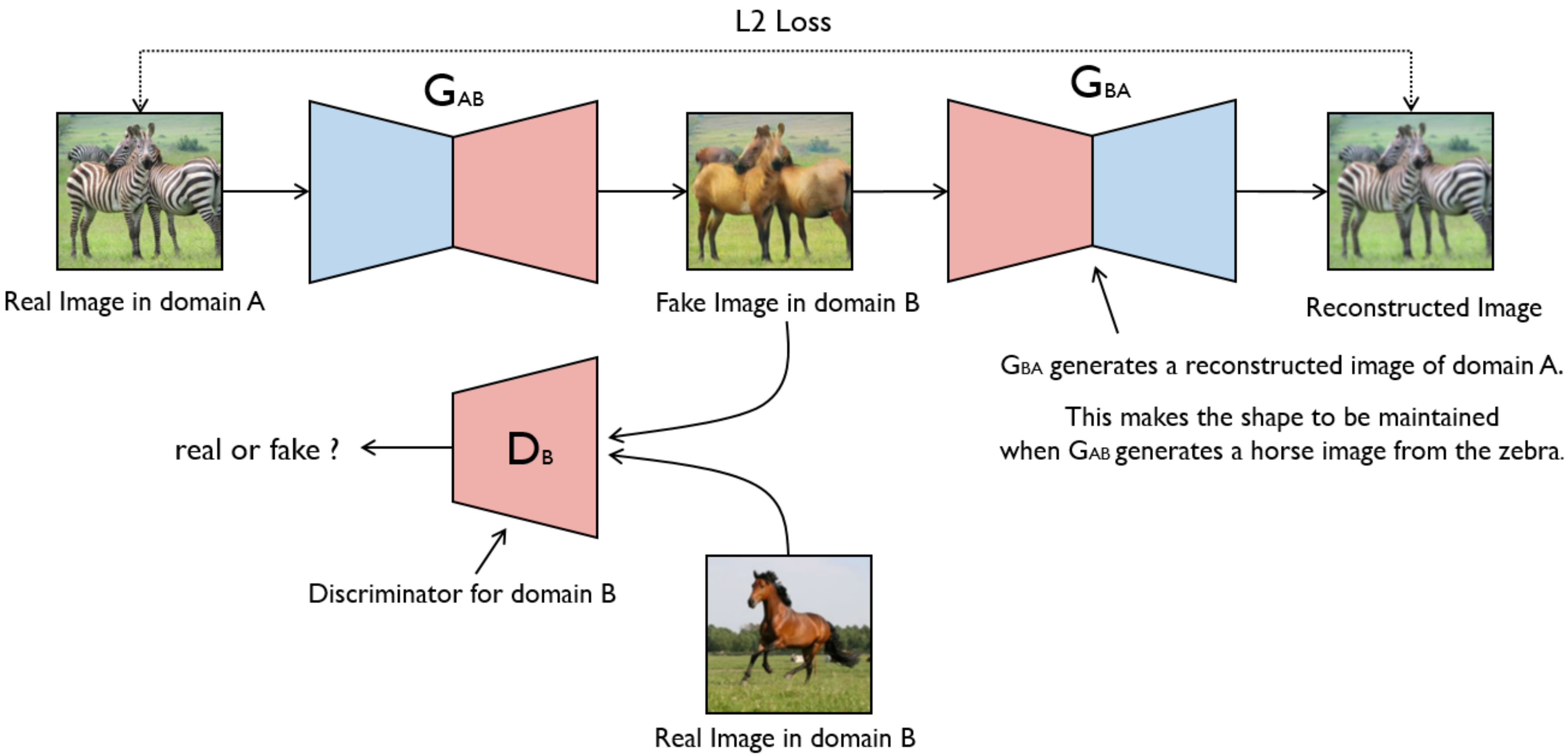
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

https://arxiv.org/abs/1406.2661

Generative Adversarial Networks



CycleGan


L2 Loss
G_{AB}
G_{BA}
Real Image in domain A
Fake Image in domain B
Reconstructed Image
G_{BA} generates a reconstructed image of domain A.
This makes the shape to be maintained
when G_{AB} generates a horse image from the zebra.
real or fake ?
D_B
Discriminator for domain B
Real Image in domain B

Text and Image

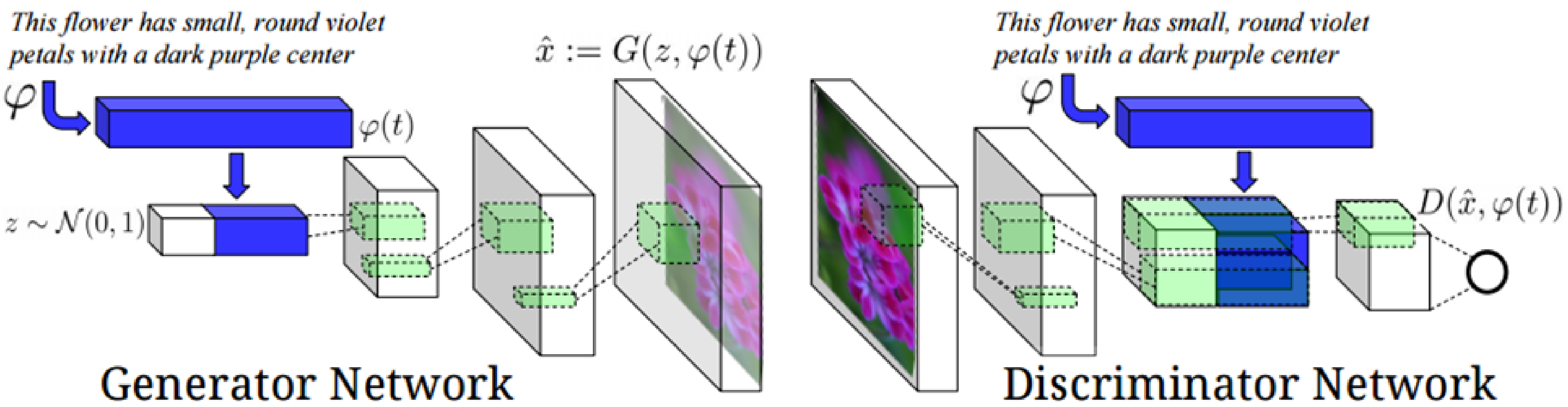


Figure 2. Our text-conditional convolutional GAN architecture. Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

Network Architecture