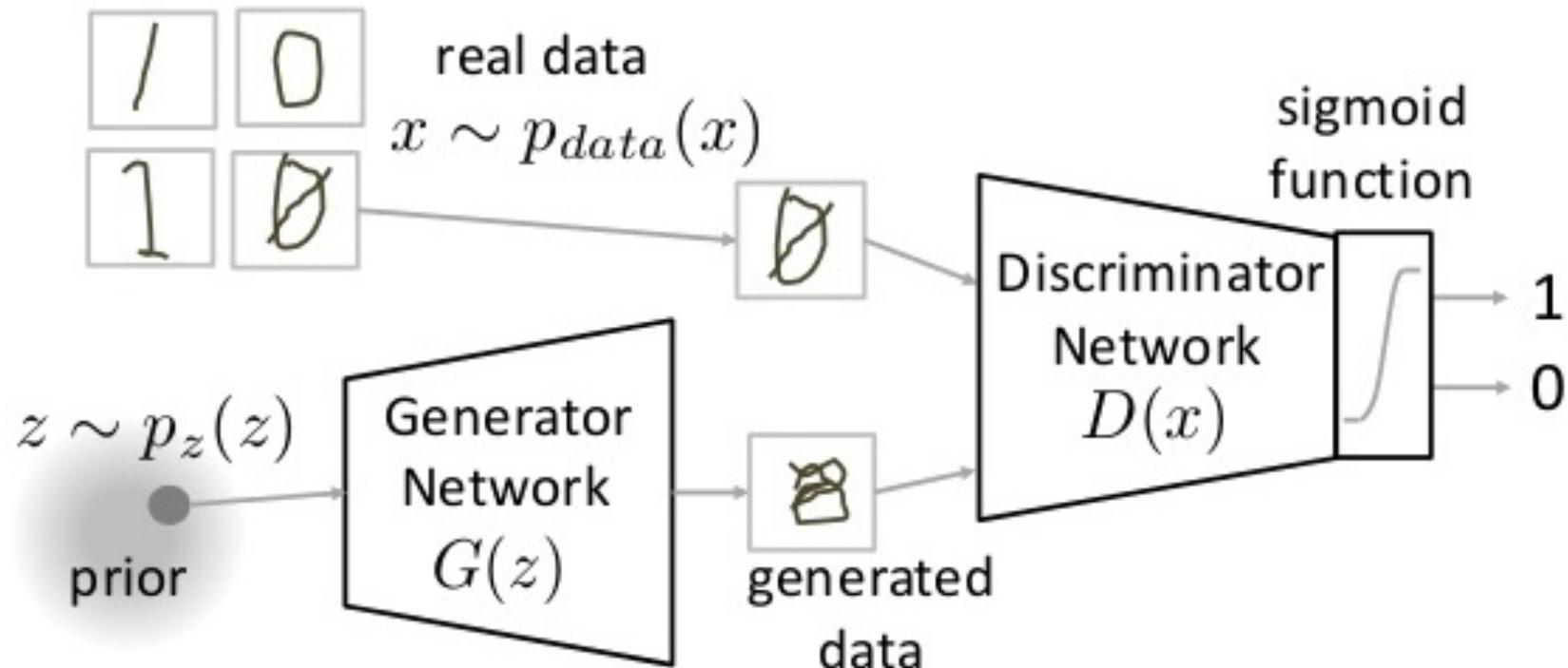


# Deep Learning: GANs, Deep Reinforcement Learning

Vincent Lepetit

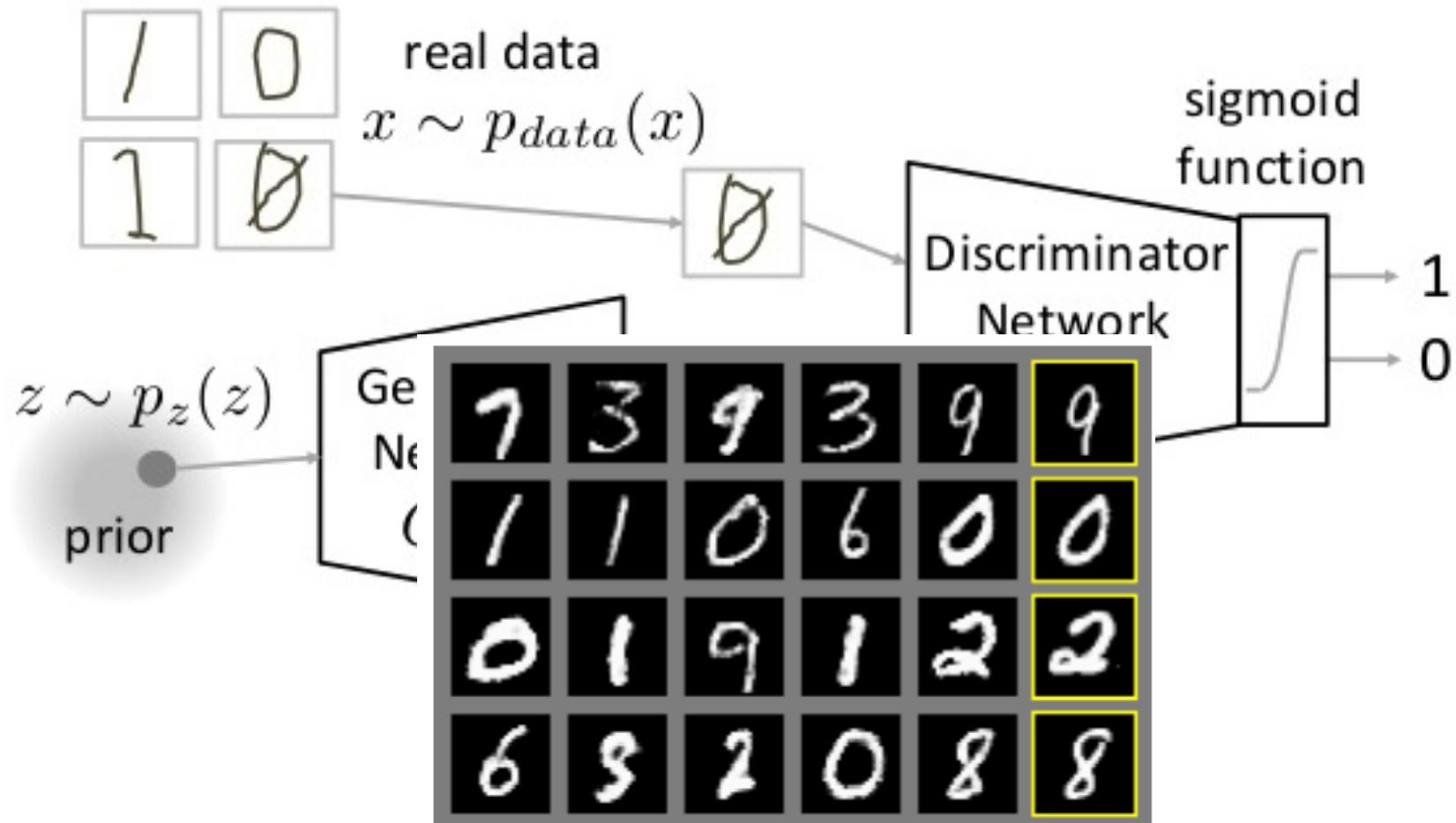
# Generative Adversarial Networks

[Goodfellow et al., NIPS14]



# Generative Adversarial Networks

[Goodfellow et al., NIPS14]



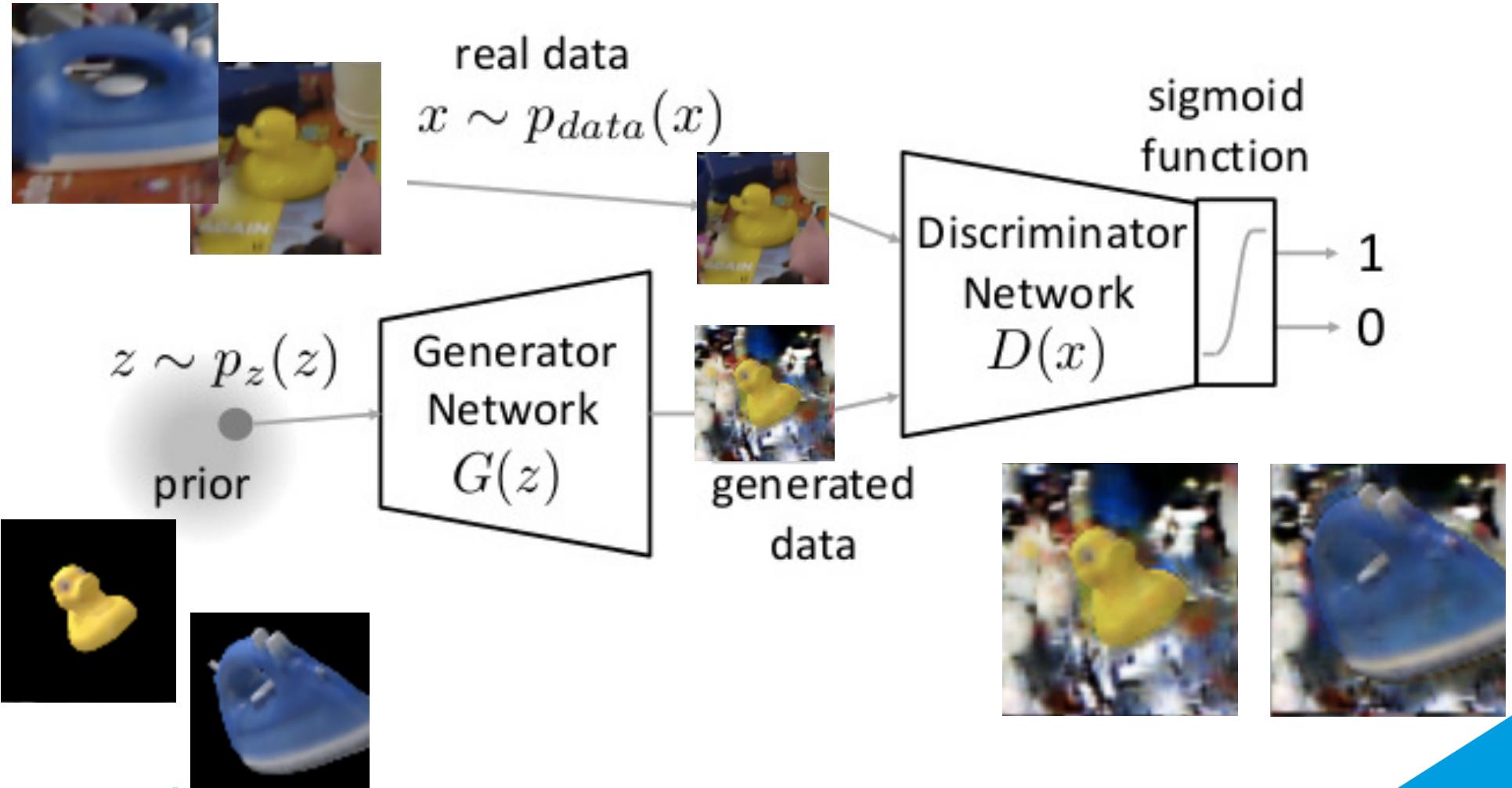


	This bird is red and brown in color, with a stubby beak	The bird is short and stubby with yellow on its body	A bird with a medium orange bill white body gray wings and webbed feet	This small black bird has a short, slightly curved bill and long legs	A small bird with varying shades of brown with white under the eyes	A small yellow bird with a black crown and a short black pointed beak	This small bird has a white breast, light grey head, and black wings and tail
Text description							
64x64 GAN-INT-CLS [22]							
128x128 GAWWN [20]							
256x256 StackGAN							

Figure 3. Example results by our proposed StackGAN, GAWWN [20], and GAN-INT-CLS [22] conditioned on text descriptions from CUB test set. GAWWN and GAN-INT-CLS generate 16 images for each text description, respectively. We select the best one for each of them to compare with our StackGAN.

# GANs for Domain Adaptation

## [Bousmalis et al, 2016]



# Next Video Frames Prediction

Mean Squared Error



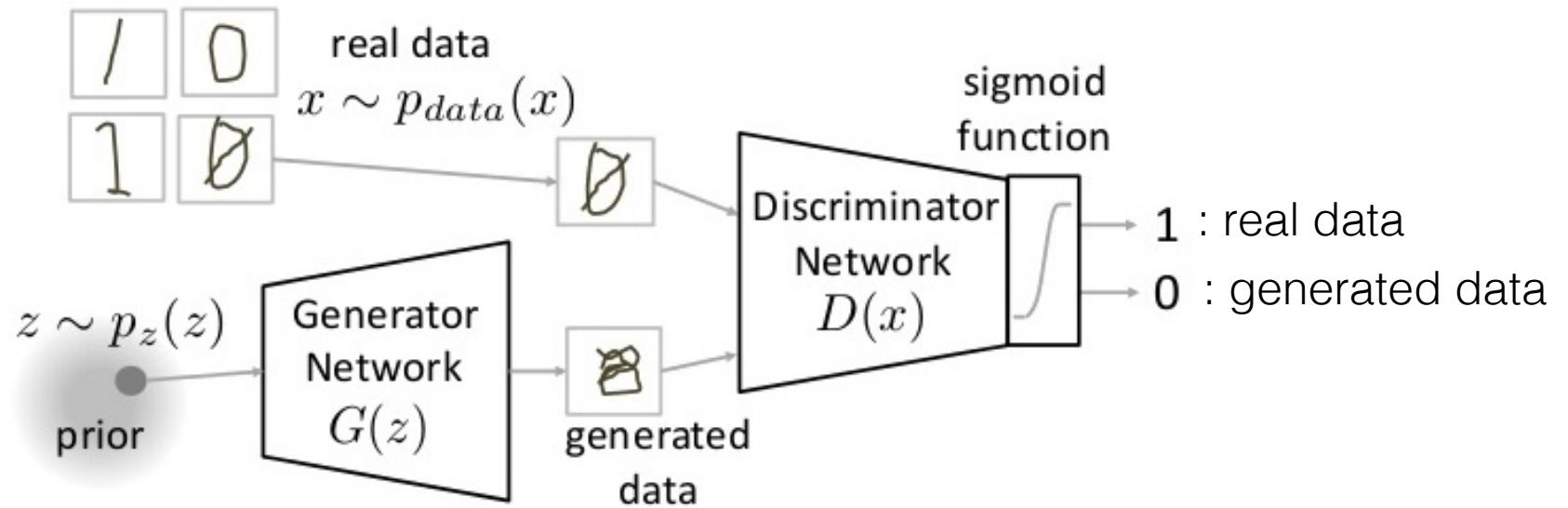
Mean Absolute Error



Adversarial



# Generative Adversarial Networks



$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

---

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

---

**for** number of training iterations **do**

**for**  $k$  steps **do**

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

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

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

---

**for** number of training iterations **do**

**for**  $k$  steps **do**

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

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

**end for**

- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_g(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

---

**for** number of training iterations **do**

**for**  $k$  steps **do**

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

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

**end for**

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

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

**end for**

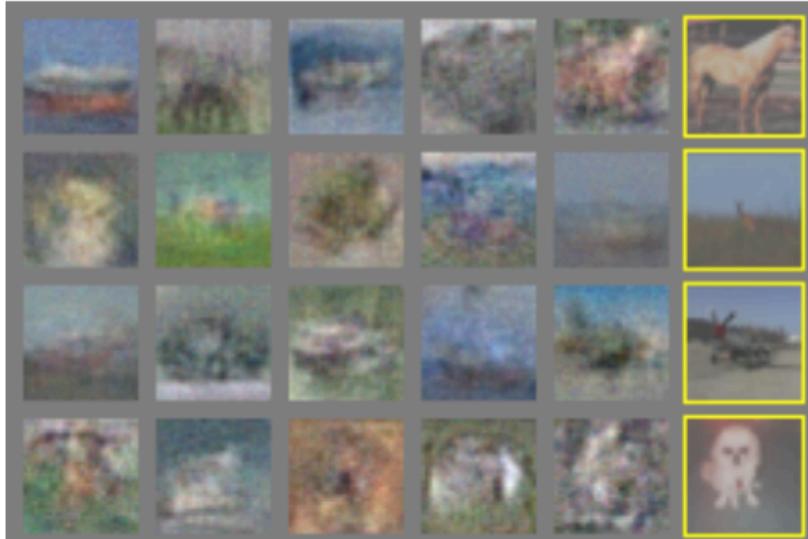
# Results



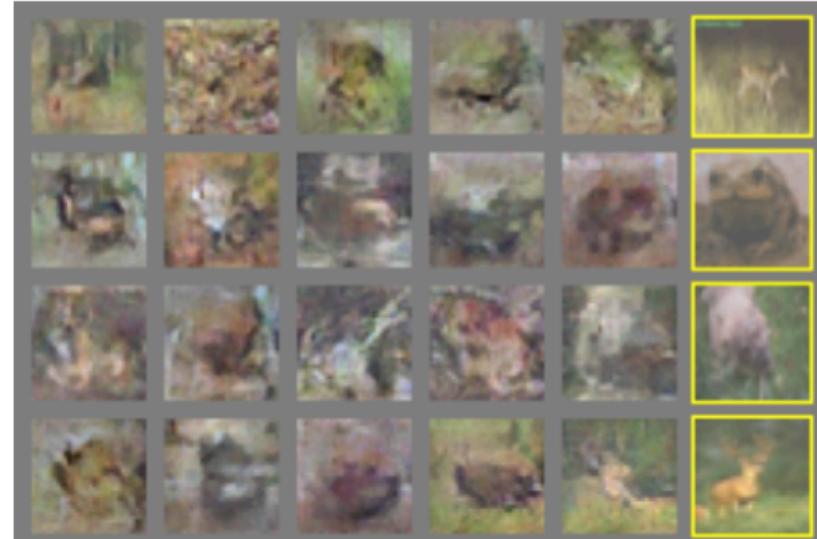
a)



b)



c)



d)

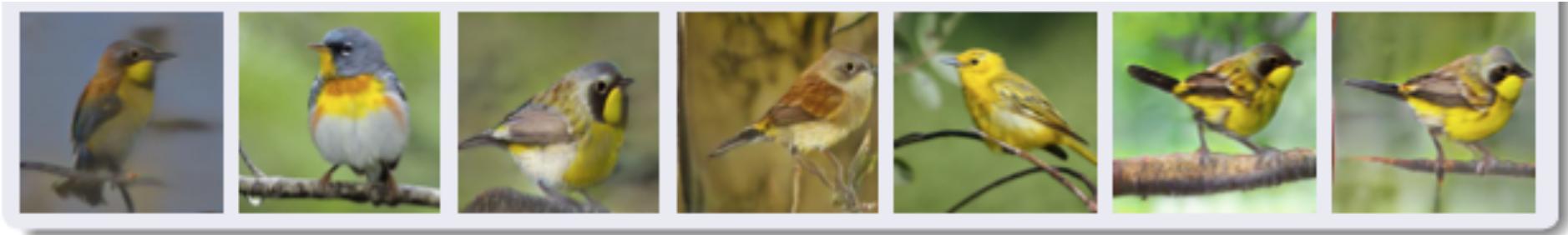
# StackGAN: Text to Photo-realistic Image Synthesis [Zhang et al, ICCV'17]



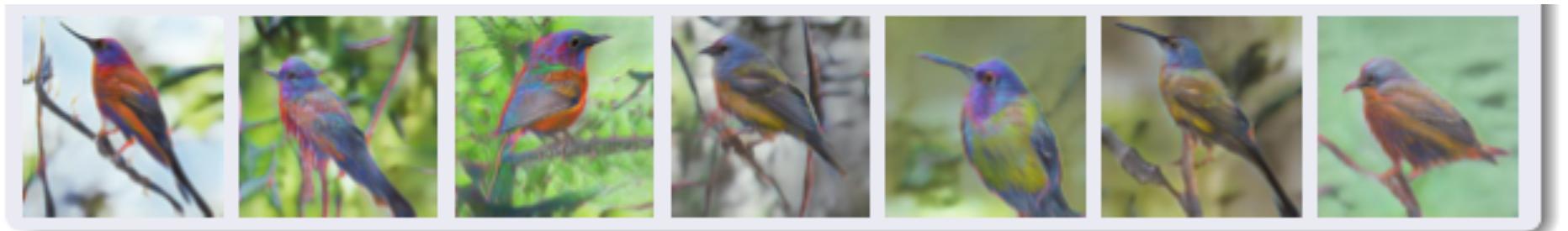
Figure 3. Example results by our proposed StackGAN, GAWWN [20], and GAN-INT-CLS [22] conditioned on text descriptions from CUB test set. GAWWN and GAN-INT-CLS generate 16 images for each text description, respectively. We select the best one for each of them to compare with our StackGAN.

# Training Set

"This bird is yellow, brown, and white in color with a sharp beak, and black eye rings."



"The colorful bird has a blue head and red neck with an orange belly and yellow back."



# Generative Adversarial Network

Generator  $G$ , Discriminator  $D$

- $D$  tries to distinguish real training data from synthetic images
- $G$  tries to fool the discriminator

Objective function:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))]$$

- $x$  is a real image from the true data distribution  $p_{\text{data}}$
- $z$  is a noise vector sampled from distribution  $p_z$  (e.g., uniform or Gaussian distribution)

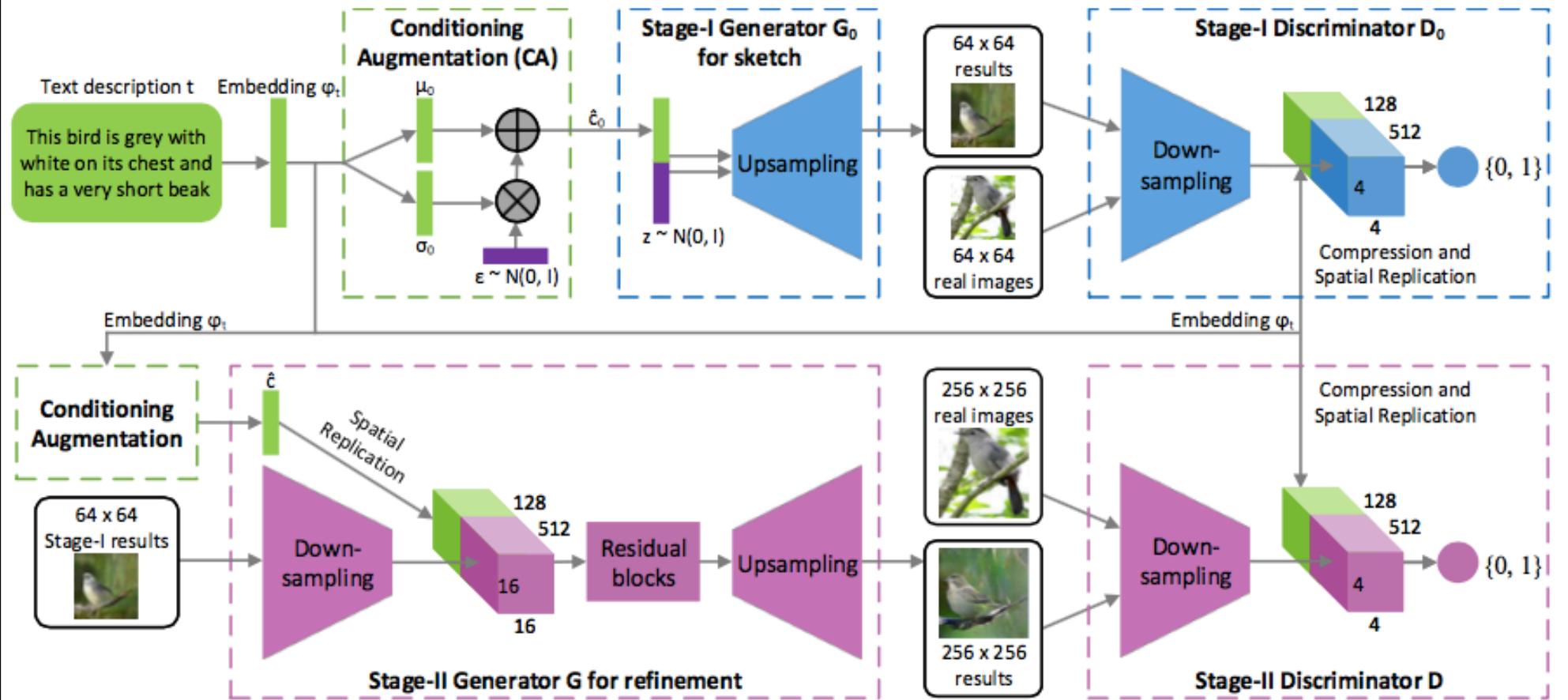
# Conditional Generative Adversarial Networks

$$\min_G \max_D V(D, G) = \mathbb{E}_{x, c \sim p_{data}(x, c)} [\log D(x, c)] + \\ \mathbb{E}_{z \sim p_z, c \sim p_{data}(c)} [\log(1 - D(G(z, c)), c)]$$

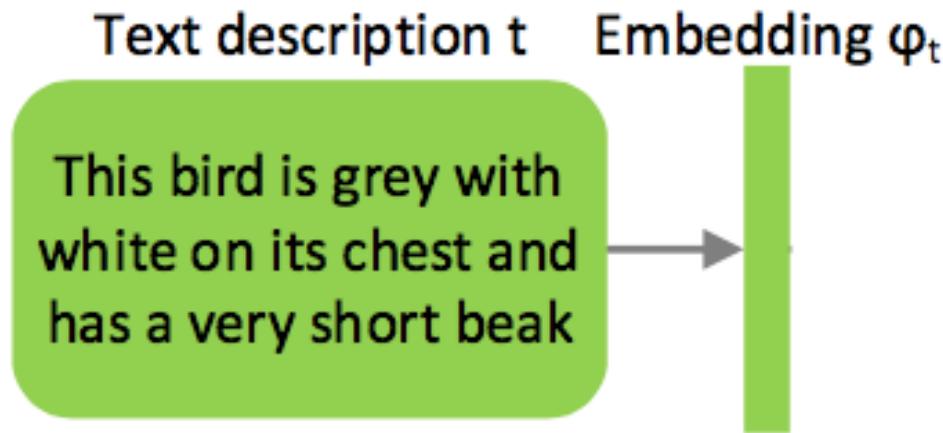
Both the generator and discriminator receive additional conditioning variables  $c$ , yielding  $G(z, c)$ ,  $D(x, c)$ .

The discriminator  $D$  tries to distinguish real pairs here (text, image) from the fake.

# StackGAN: Architecture



# StackGAN: Conditioning Augmentation



Embedding extracted from the text description by a pre-trained encoder [Reed et al., CVPR, 2016]

# Learning Deep Representations of Fine-Grained Visual Descriptions [Reed et al, CVPR 2016]

Learn two classifiers  $f_v$  and  $f_t$  by minimizing:

$$\frac{1}{N} \sum_{n=1}^N \Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n))$$

given

$$\mathcal{S} = \{(v_n, t_n, y_n), n = 1, \dots, N\}$$

$\Delta : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$  is the 0-1 loss

$v_n$  : image

$t_n$  : caption

$y_n$  : category (bird species)

$f_v$  and  $f_t$

$$\min \quad \frac{1}{N} \sum_{n=1}^N \Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n))$$

$$f_v(v) = \arg \max_{y \in \mathcal{Y}} \mathbb{E}_{t \sim \mathcal{T}(y)} [F(v, t)]$$

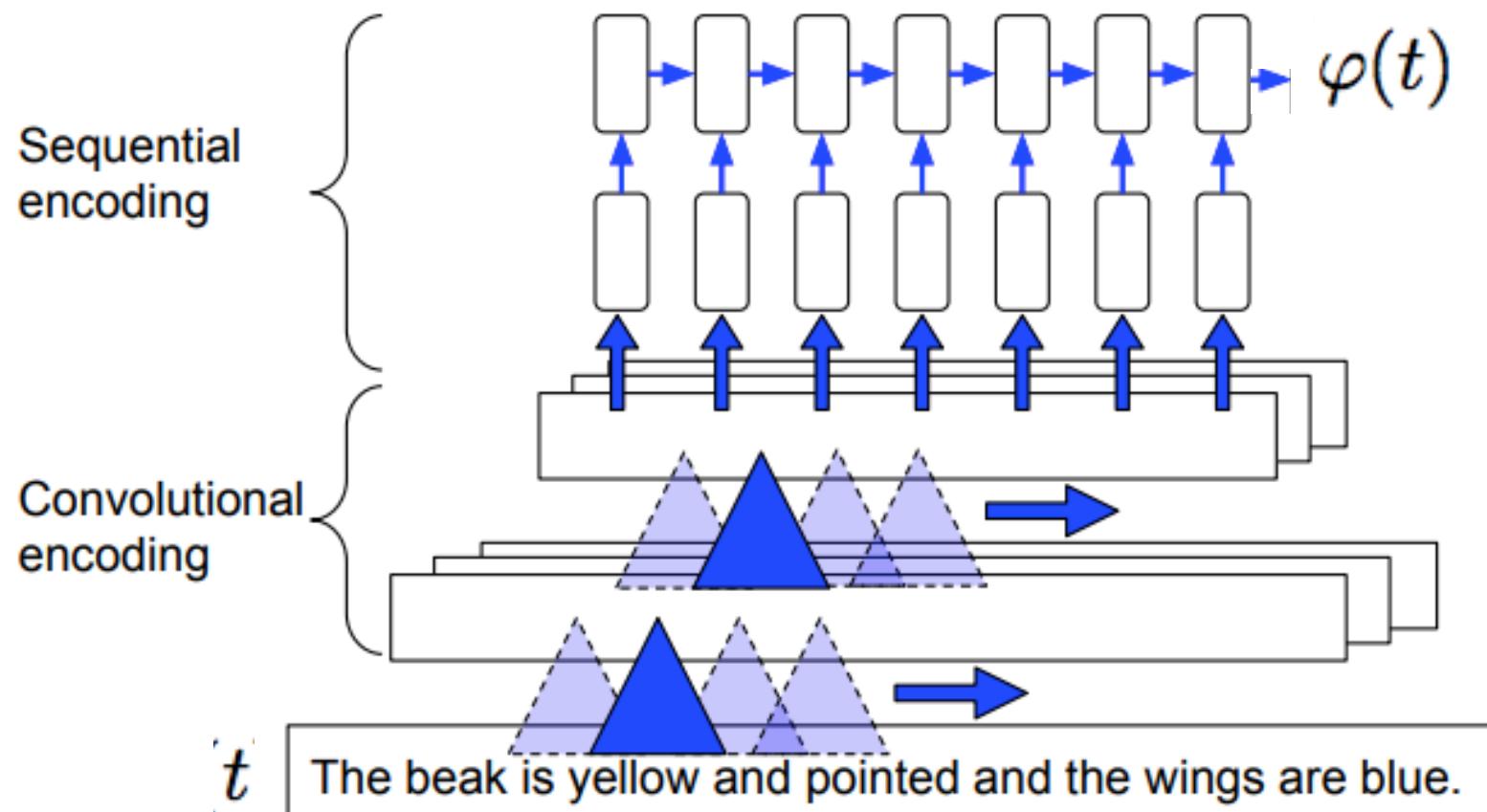
$$f_t(t) = \arg \max_{y \in \mathcal{Y}} \mathbb{E}_{v \sim \mathcal{V}(y)} [F(v, t)]$$

with:  $F(v, t) = \theta(v)^T \varphi(t)$

$\theta(v)$ : Image features computed by GoogLeNet (fixed)

$\varphi(t)$  : Textual features (learned)

# Convolutional Recurrent Network (CNN-RNN)



# Optimization

$$\min \frac{1}{N} \sum_{n=1}^N \Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n))$$

$$\min \frac{1}{N} \sum_{n=1}^N \ell_v(v_n, t_n, y_n) + \ell_t(v_n, t_n, y_n)$$

with:  $\ell_v(v_n, t_n, y_n) =$

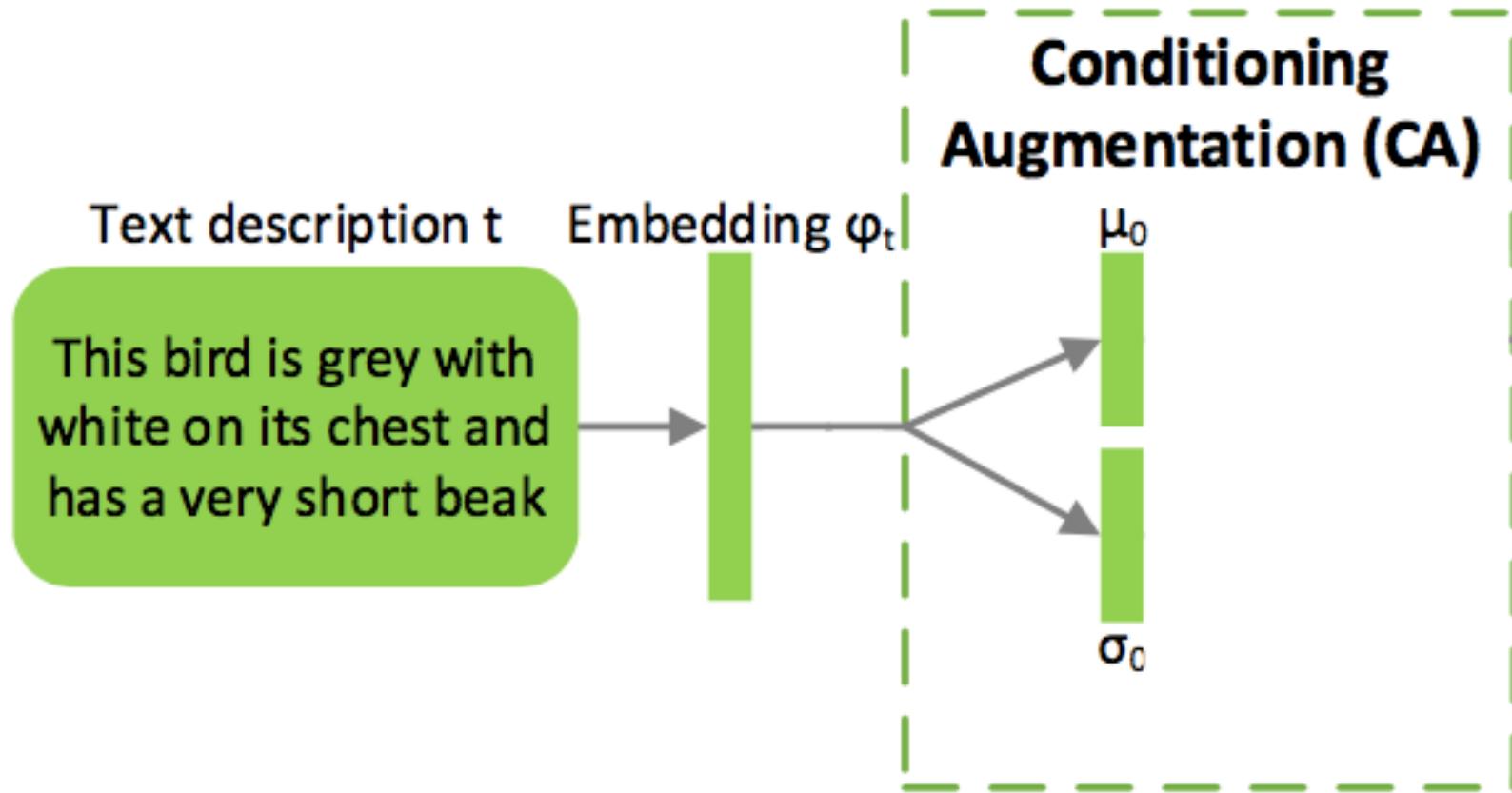
$$\max_{y \in \mathcal{Y}} (0, \Delta(y_n, y) + \mathbb{E}_{t \sim \mathcal{T}(y)} [F(v_n, t) - F(v_n, t_n)])$$

$\ell_t(v_n, t_n, y_n) =$

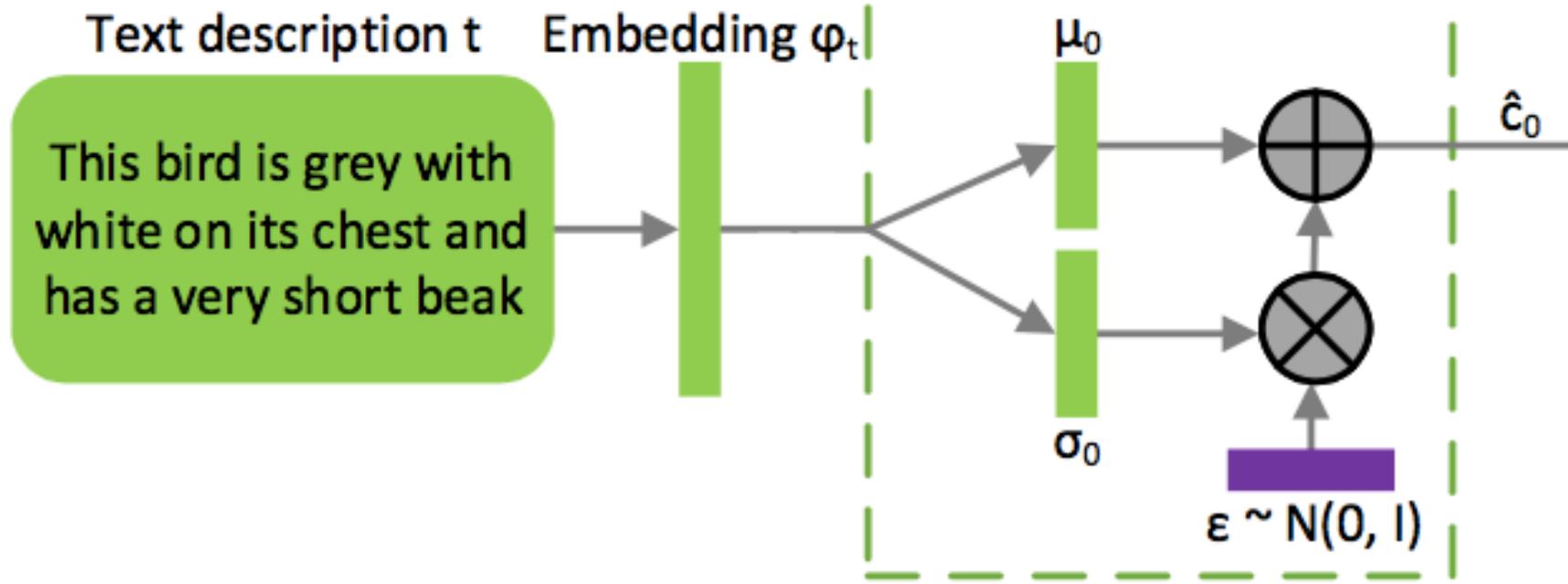
$$\max_{y \in \mathcal{Y}} (0, \Delta(y_n, y) + \mathbb{E}_{v \sim \mathcal{V}(y)} [F(v, t_n) - F(v_n, t_n)])$$

$$F(v, t) = \theta(v)^T \varphi(t)$$

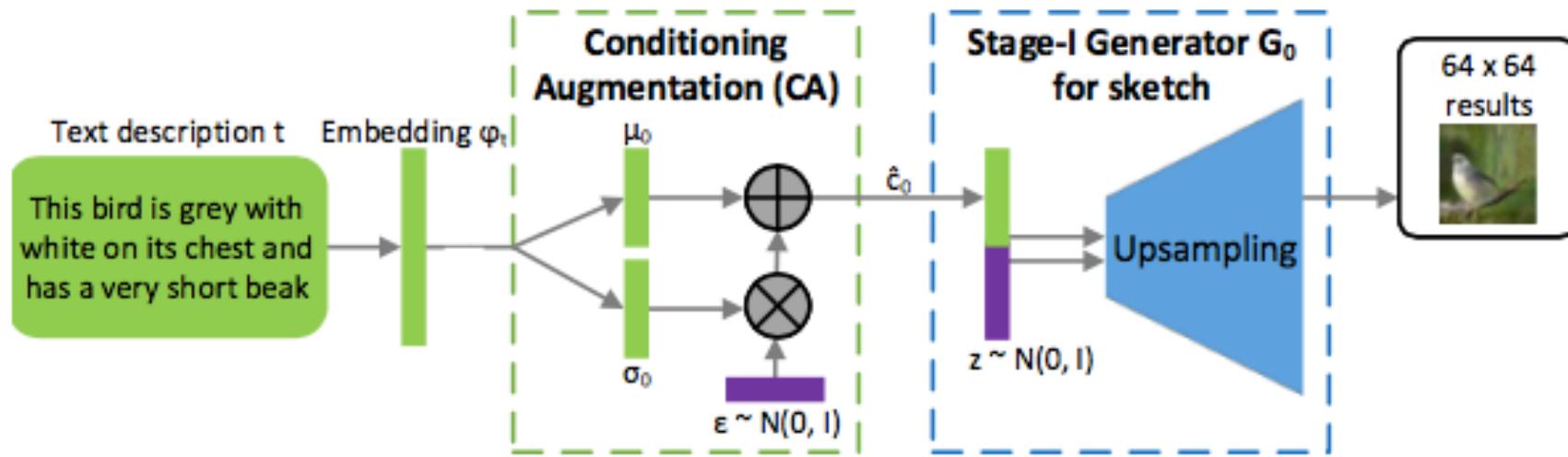
# StackGAN: Conditioning Augmentation



## Conditioning Augmentation (CA)



# StackGAN: Stage-I GAN



# StackGAN: Sentence Embedding Interpolation

The bird is completely **red** → The bird is completely **yellow**



This bird is completely **red** with **black** wings and **pointy** beak →  
this small **blue** bird has a **short** pointy beak and **brown** on its wings



# Wasserstein GANs [Arjovsky et al, 2017]

Given real examples  $\{x^{(i)}\}_{i=1..m}$ ,  
given a parametric family of densities  $P_\theta$ ,

we would like to find  $\theta$  such that

$\mathbb{P}_r$  : the real data distribution

$\mathbb{P}_\theta$  : the distribution of the parameterized density  $P_\theta$   
are as close as possible.

# The Earth Mover Distance between two distributions (or Wasserstein Metric)

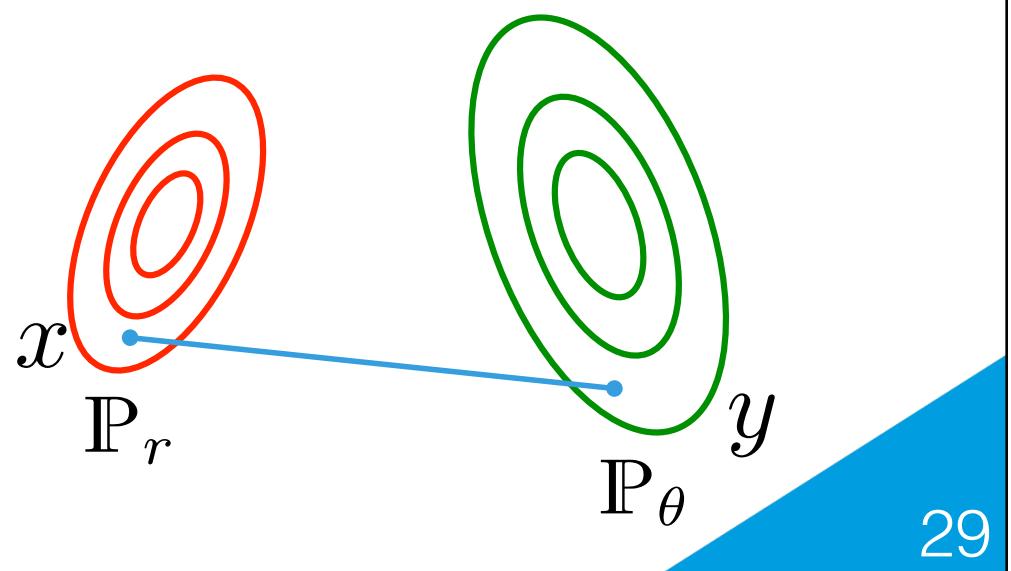
$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \inf_{\gamma \in \pi(\mathbb{P}_r, \mathbb{P}_\theta)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|]$$

$\gamma(x, y)$  : joint distribution

$$\gamma(\cdot, y) = \mathbb{P}_r$$

$$\gamma(x, \cdot) = \mathbb{P}_\theta$$

Intuitively, the cost of the optimal transport plan between the two distributions:



$g_\theta(z)$  : Neural network with parameters  $\theta$  with  $z \in Z$

$\mathbb{P}_\theta$  : the distribution of  $g_\theta(Z)$

then  $W(\mathbb{P}_r, \mathbb{P}_\theta)$  is continuous in  $\theta$ , which is not necessarily the case for other distances and divergences between distributions (Kullback-Leibler, Jensen-Shannon, Total Variation, ...)

→ this will help the convergence when training the neural networks.

[proof in the paper]

# Kantorovich-Rubinstein Duality

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L < 1} \left( \mathbb{E}_{x \sim \mathbb{P}_r} [f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta} [f(x)] \right)$$

With a parameterized family  $\{f_w\}_{w \in W}$  with  $\|f_w\|_L < 1$ :

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \max_{w \in W} \left( \mathbb{E}_{x \sim \mathbb{P}_r} [f_w(x)] - \mathbb{E}_{z \sim p(z)} [f_w(g_\theta(z))] \right)$$

# Kantorovich-Rubinstein Duality

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L < 1} \left( \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)] \right)$$

With a parameterized family  $\{f_w\}_{w \in W}$  with  $\|f_w\|_L < 1$ :

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \max_{w \in W} \left( \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z \sim p(z)}[f_w(g_\theta(z))] \right)$$

Idea: Implement  $f_w$  with a neural network.

$$\min_{\theta} \max_{w \in W} \left( \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z \sim p(z)}[f_w(g_\theta(z))] \right)$$

# Algorithm

---

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

# Algorithm

---

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

1: **while**  $\theta$  has not converged **do**

# Algorithm

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

```
1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  do
```

# Algorithm

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

- 1: **while**  $\theta$  has not converged **do**
- 2:   **for**  $t = 0, \dots, n_{\text{critic}}$  **do**
- 3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
- 4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.

$$\min_{\theta} \max_{w \in W} \left( \mathbb{E}_{x \sim \mathbb{P}_r} [f_w(x)] - \mathbb{E}_{z \sim p(z)} [f_w(g_{\theta}(z))] \right)$$

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

- 1: **while**  $\theta$  has not converged **do**
- 2:   **for**  $t = 0, \dots, n_{\text{critic}}$  **do**
- 3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
- 4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
- 5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_{\theta}(z^{(i)})) \right]$

# Algorithm

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

- 1: **while**  $\theta$  has not converged **do**
- 2:   **for**  $t = 0, \dots, n_{\text{critic}}$  **do**
- 3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
- 4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
- 5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$
- 6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$

# Algorithm

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

- 1: **while**  $\theta$  has not converged **do**
- 2:   **for**  $t = 0, \dots, n_{\text{critic}}$  **do**
- 3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
- 4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
- 5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$
- 6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
- 7:      $w \leftarrow \text{clip}(w, -c, c)$

forces  $f_w$  to be a Lipschitz function

# Algorithm

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

```
1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  do
3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$ 
6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$ 
7:      $w \leftarrow \text{clip}(w, -c, c)$ 
8:   end for
```

# Algorithm

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

- 1: **while**  $\theta$  has not converged **do**
- 2:   **for**  $t = 0, \dots, n_{\text{critic}}$  **do**
- 3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
- 4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
- 5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$
- 6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
- 7:      $w \leftarrow \text{clip}(w, -c, c)$
- 8:   **end for**
- 9:   Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
- 10:    $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \max_{w \in W} \left( \mathbb{E}_{x \sim \mathbb{P}_r} [f_w(x)] - \mathbb{E}_{z \sim p(z)} [f_w(g_\theta(z))] \right)$$

$$f_w = \max_{w' \in W} \left( \mathbb{E}_{x \sim \mathbb{P}_r} [f_{w'}(x)] - \mathbb{E}_{z \sim p(z)} [f_{w'}(g_\theta(z))] \right)$$

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z \sim p(z)} [f_w(g_\theta(z))]$$

# Algorithm

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

```
1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  do
3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$ 
6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$ 
7:      $w \leftarrow \text{clip}(w, -c, c)$ 
8:   end for
9:   Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
10:   $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$ 
11:   $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$ 
```

# Algorithm

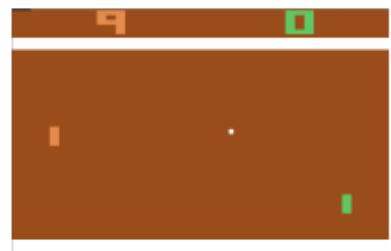
**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

```
1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  do
3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$ 
6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$ 
7:      $w \leftarrow \text{clip}(w, -c, c)$ 
8:   end for
9:   Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
10:   $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$ 
11:   $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$ 
12: end while
```

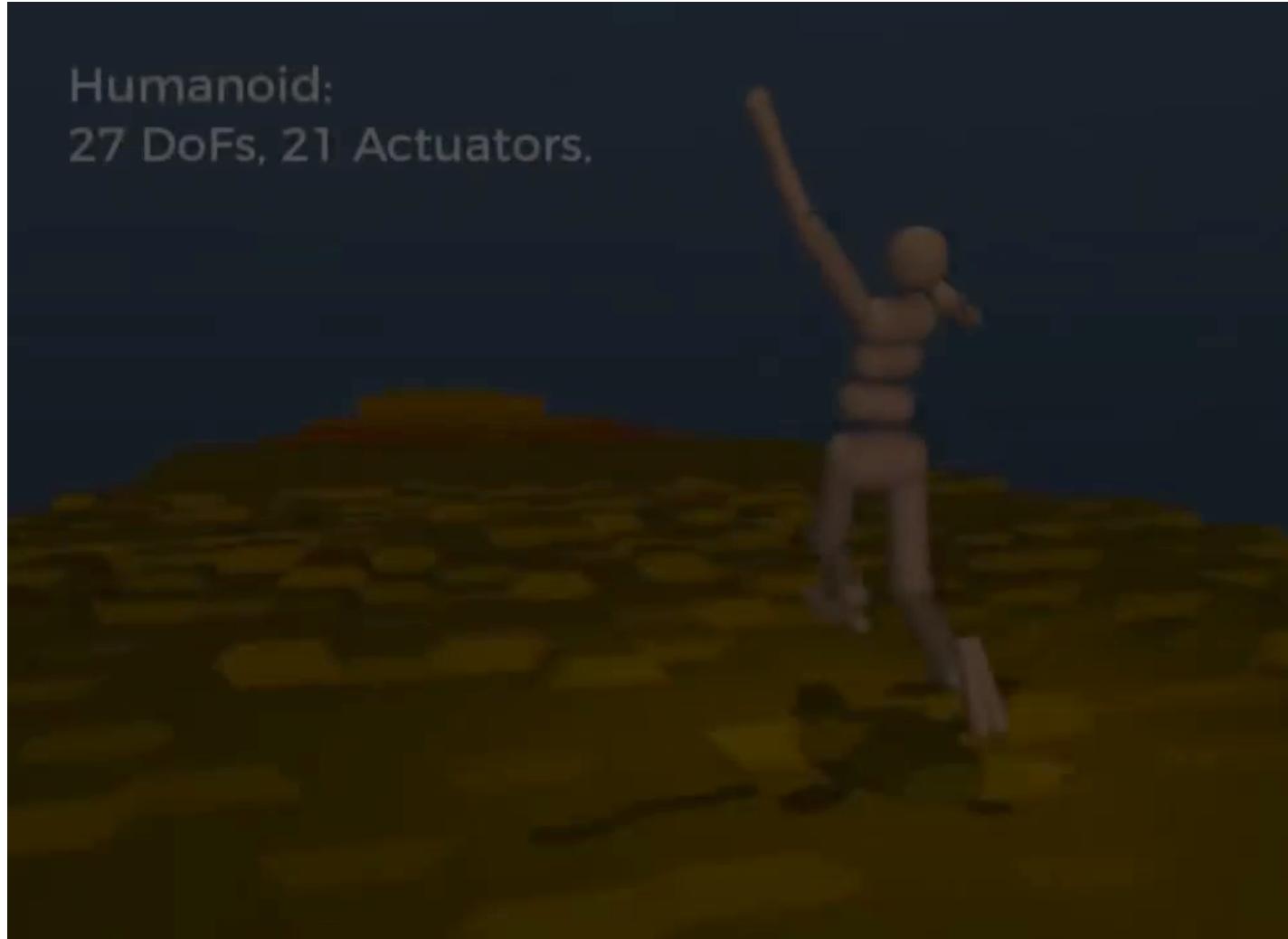
# Deep Reinforcement Learning

# Learning to Play Atari Games





# Learning to Walk



Emergence of Locomotion Behaviours in Rich Environments - DeepMind

# Reinforcement Learning

An agent learns to solve a task by trial-and-error on the environment.

The agent observes the rewards for the actions tried.

Not supervised learning, not unsupervised learning [we need to provide a reward function]

This slide and the next ones are directly inspired by a presentation from Shimon Whiteson

# Formalization [contextual K-armed bandit problem]

- $K$  possible actions at each time step;
- After the  $t$ -th action  $a_t$ :
  - the agent receives a reward  $r_t$ ;
  - the agent makes an observation  $\phi_{t+1}$  of the environment.
- Agent's Goal: Maximize the expected return  $R_t$ :

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad \text{with } \gamma \in [0; 1)$$

# Trade-off Exploration/Exploitation

The agent must *explore* the actions in order to learn about them and improve its chances of getting future reward;

But the agent must also use what it already knows in order to maximise its total reward

→ it must *exploit* by taking the actions it expects to give the largest rewards.

# Action-Value Function

Action-value of a policy  $\pi$  is:

$$Q^\pi(\phi, a) = E_\pi [R_t \mid \phi_t = \phi, a_t = a] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid \phi_t = \phi, a_t = a \right]$$

*Exploiting* means taking the greedy action:

$$a^* = \arg \max_a Q^\pi(\phi, a)$$

*Exploring* means taking any other action.

# Bellman Equation

Gives a recursive definition for  $Q$ :

$$Q^\pi(\phi, a) = \sum_{\phi'} P_{\phi, \phi'}^a \left[ R_{\phi, \phi'}^a + \gamma \sum_{a'} \pi(\phi', a') Q^\pi(\phi', a') \right]$$

$P_{\phi, \phi'}^a$  : probability of moving from  $\phi$  to  $\phi'$  after action  $a$

$R_{\phi, \phi'}^a$  : reward when moving from  $\phi$  to  $\phi'$  with action  $a$

# Turning Bellman Optimality Equation into an Update

$$Q_{t+1}^{\pi}(\phi, a) \leftarrow \sum_{\phi'} P_{\phi, \phi'}^a \left[ R_{\phi, \phi'}^a + \gamma \max_{a'} Q_t^{\pi}(\phi', a') \right]$$

Learning function  $Q$  with a Neural Network of parameters  $\theta$ :

- Perform action  $a_t$  (by exploitation or exploration);
- Get reward  $r_t$ , observe  $\phi_{t+1}$
- Let  $y_t = r_t + \gamma \max_a Q(\phi_{t+1}, a; \theta)$
- Perform a gradient descent step on  $(y_t - Q(\phi_t, a_t; \theta))^2$

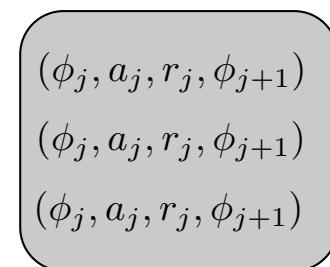
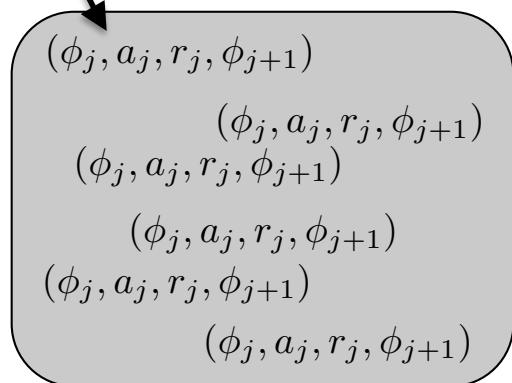
# Experience Replay [Mnih et al. 2015]

Instead of using only the last action to train  $Q$ ,  
store all previous data  $\{(\phi_t, a_t, r_t, \phi_{t+1})\}$

At each step:

- store  $(\phi_t, a_t, r_t, \phi_{t+1})$

- run one random minibatch optimization:



$$\Delta\theta$$

Random minibatch

Replay memory

# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3

**end for**

**end for**

---

# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

Initialize replay memory  $\mathcal{D}$  to capacity  $N$



Replay memory

# Algorithm

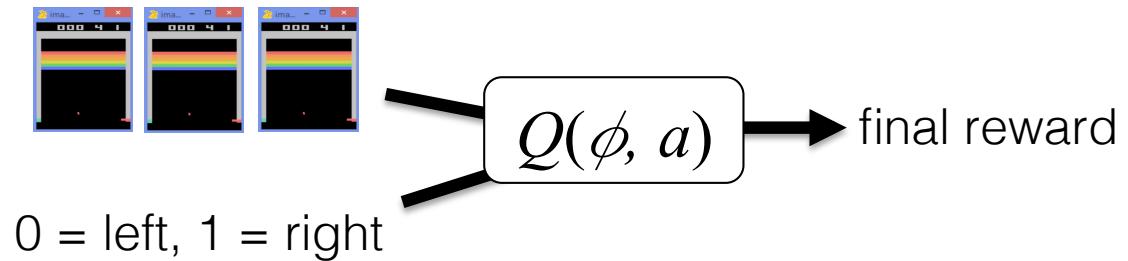
---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights



# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

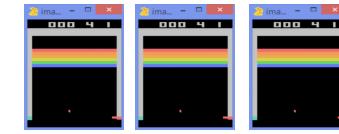
# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$



Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$



# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$



# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$



# Algorithm

## Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

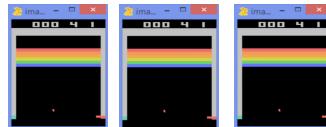
        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

Replay memory



# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

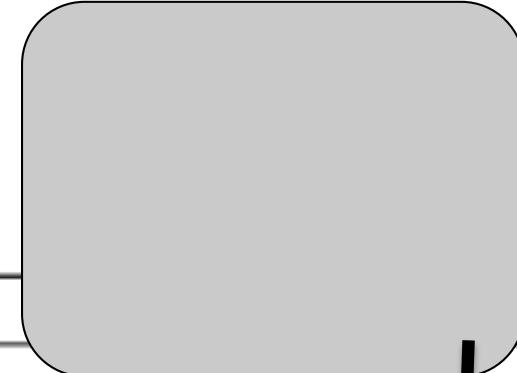
        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

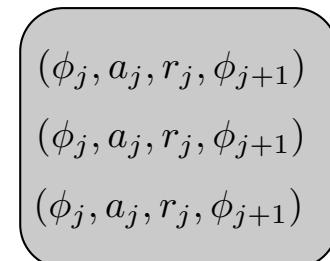
        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

**Sample** random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$



Replay memory



Random minibatch

# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

    Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

    Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

$(\phi_j, a_j, r_j, \phi_{j+1})$

$(\phi_j, a_j, r_j, \phi_{j+1})$

$(\phi_j, a_j, r_j, \phi_{j+1})$

# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

    Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

    Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

    Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3

$(\phi_j, a_j, r_j, \phi_{j+1})$   
 $(\phi_j, a_j, r_j, \phi_{j+1})$   
 $(\phi_j, a_j, r_j, \phi_{j+1})$



$\Delta\theta$

# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3

**end for**

# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3

**end for**

**end for**

# Algorithm

---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3

**end for**

**end for**

---