Generative Adversarial Nets

Richard Godden and Yogesh Garg

April 4, 2017

Introduction

What are Generative models?

The aim

- have a data set of training examples $x_i \sim p_{data}(x)$
- want to be able to generate new examples $\hat{x} \sim p_g(x)$
- want that $p_g \approx p_{data}$

Example models

- Hidden Markov models
- Graphical models (directed/undirected)
- Restricted Boltzmann Machines
- Generative autoencoders

Why Generative models

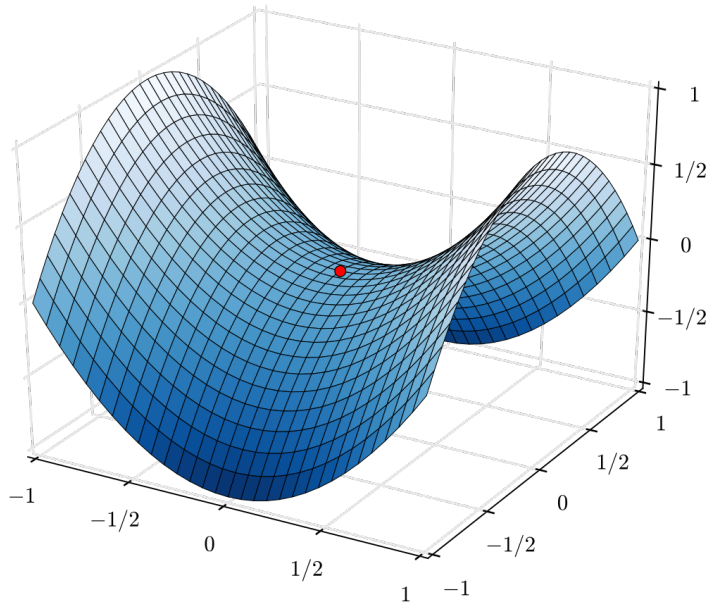
- useful or required for many tasks
  - generating realistic audio from text (text to speach)
  - machine translation
- clean up noisy or missing data

- unsupervised learning
  - can be used to learn features from raw data
  - “What I cannot create I do not understand” Richard Feynman

Game theory

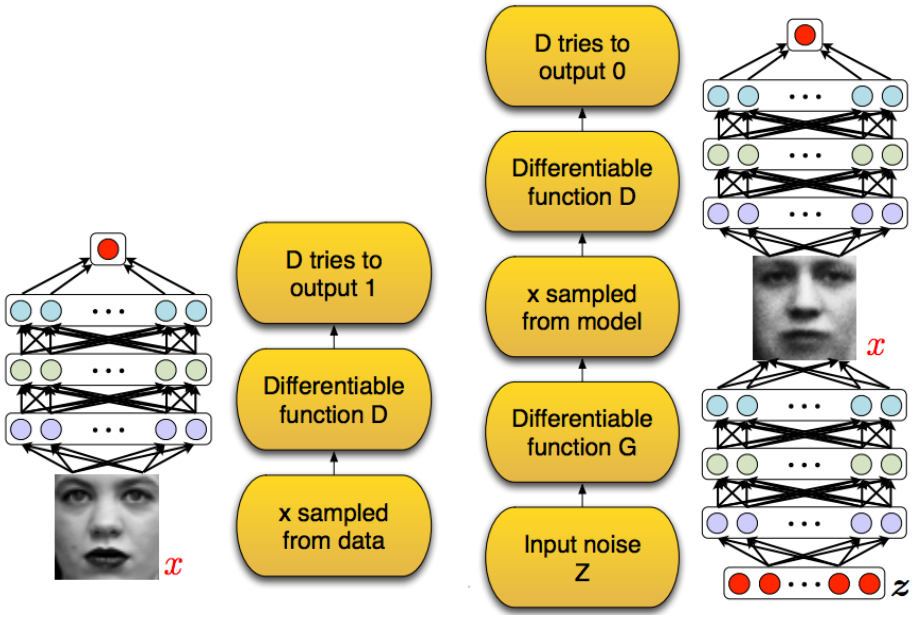
- zero sum game with mixed strategy equilibrium
- Minimax: nash equilibrium at saddle point

You \ Your opponent	Rock	Paper	Scissors
Rock	0	-1	1
Paper	1	0	-1
Scissors	-1	1	0



Adversarial networks

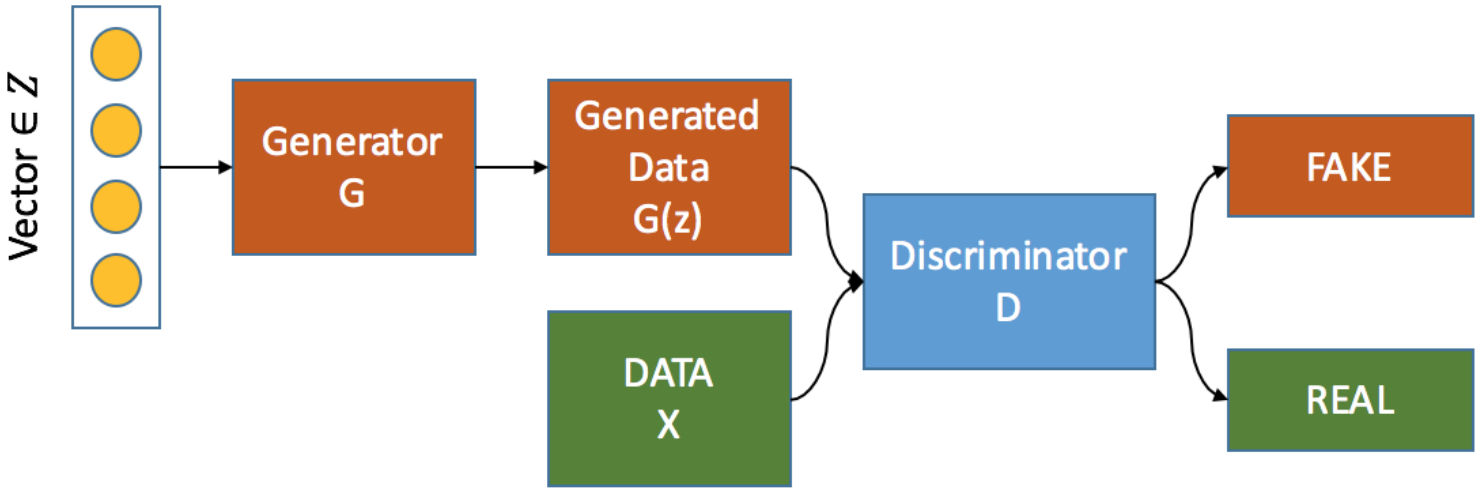
- Game between two players, mixed strategy equilibrium when one learns to generate data
- Generator tries to create fake samples that appear real
- Descriminator tries to tell which are fake and which are real



Methods

Definitions

- Z some latent space
- X a data point (generated or real)
- Generator $G(z) : Z \rightarrow X$
- Descriminator $D(x) : X \rightarrow \{0, 1\}$



Loss functions

- Minimax

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

- Discriminator

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

- Generator

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

- in practice learning for G may be bad with equation above so maximize $\log(D(G(z)))$

Algorithms

- the algorithm

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://www.youtube.com/watch?v=CILzNj2MP3s

Theorertical Results

Optimal value for the objective function

- For a fixed G aim of discriminator D is to maximize $V(G, D)$

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

$$\begin{aligned} V(G,D) &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))dx + \int_{z} p_{\boldsymbol{z}}(\boldsymbol{z}) \log(1 - D(g(\boldsymbol{z})))dz \\ &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))dx \end{aligned} \tag{3}$$

- using $argmax(alog(y) + blog(1-y)) = \frac{a}{a+b}$, we get

$$D_G^\star(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{2}$$

- Now the aim of G is to minimize $C(G)$

$$\begin{aligned} C(G) =& \max_D V(G,D) \\ =& \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^\star(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - D_G^\star(G(\boldsymbol{z})))] \\ =& \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^\star(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^\star(\boldsymbol{x}))] \\ =& \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] \end{aligned} \tag{4}$$

- ...

$$
\begin{aligned}
C(G) &= \max_D V(G, D) \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D_G^\star(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}}[\log(1 - D_G^\star(G(\boldsymbol{z})))] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D_G^\star(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D_G^\star(\boldsymbol{x}))] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right]
\end{aligned} \tag{4}
$$

- We can write this equation in terms of KL divergence between normalized distributions

$$
C(G) = -\log(4) + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right) \tag{5}
$$

- Which can also be written as the Jensen-Shannon divergence between the model's distribution and the data generating process

$$
C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right) \tag{6}
$$

- Thus, $C^* = -\log(4)$ is the optimum value attained when

$$
p_g = g_{\text{data}}
$$

Convergence of training algorithm

- Proof
  - Sub derivatives at optimal lie in sub derivatives of the overall function
  - So the training algorithm is equivalent to computing gradient descent at optimal

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

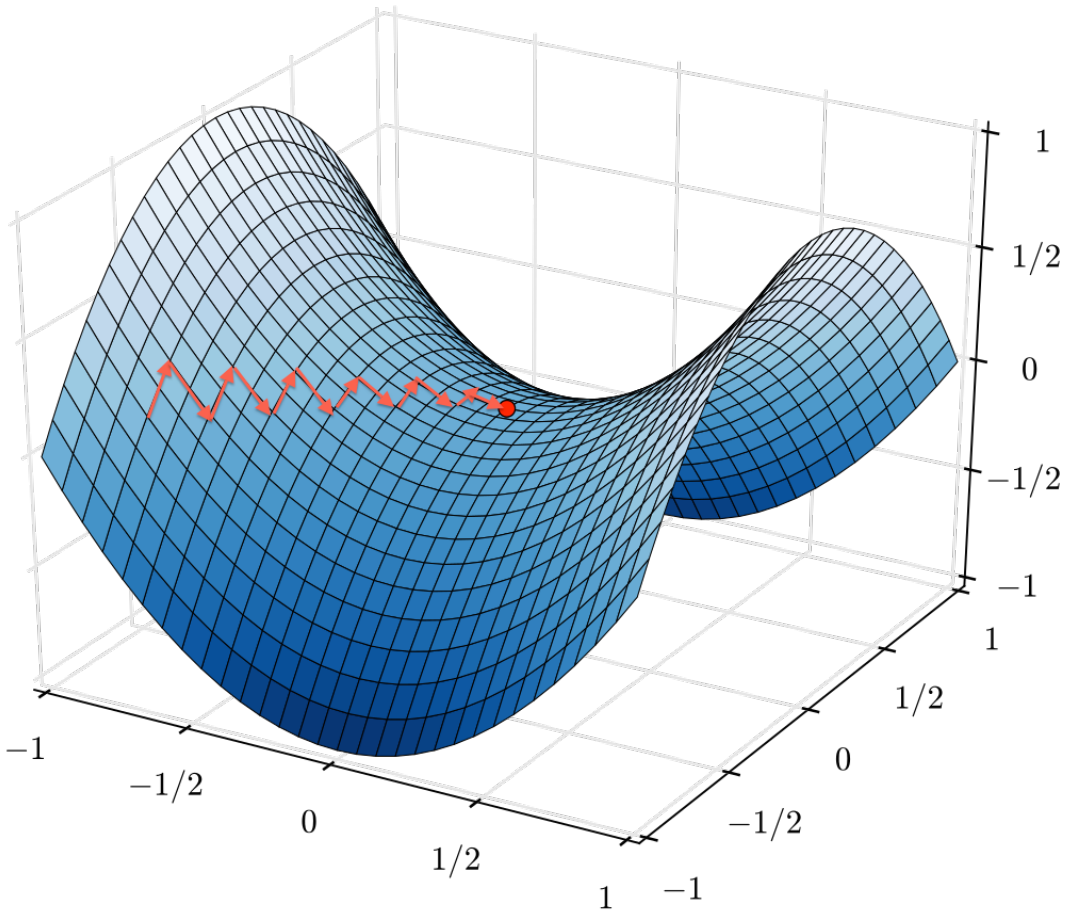
$$\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))]$$

then p_g converges to p_{data}

Proof. Consider $V(G, D) = U(p_g, D)$ as a function of p_g as done in the above criterion. Note that $U(p_g, D)$ is convex in p_g. The subderivatives of a supremum of convex functions include the derivative of the function at the point where the maximum is attained. In other words, if $f(x) = \sup_{\alpha\in\mathcal{A}} f_\alpha(x)$ and $f_\alpha(x)$ is convex in x for every α, then $\partial f_\beta(x) \in \partial f$ if $\beta = \arg\sup_{\alpha\in\mathcal{A}} f_\alpha(x)$. This is equivalent to computing a gradient descent update for p_g at the optimal D given the corresponding G. $\sup_D U(p_g, D)$ is convex in p_g with a unique global optima as proven in Thm 1, therefore with sufficiently small updates of p_g, p_g converges to p_x, concluding the proof. □

- Not sure if...

- Intuition



Experimental

- Log likelyhoods for MNIST and TFD

Model	MNIST	TFD
DBN [3]	138 ± 2	1909 ± 66
Stacked CAE [3]	121 ± 1.6	$\mathbf{2110 \pm 50}$
Deep GSN [6]	214 ± 1.1	1890 ± 29
Adversarial nets	$\mathbf{225 \pm 2}$	$\mathbf{2057 \pm 26}$

Table 1: Parzen window-based log-likelihood estimates. The reported numbers on MNIST are the mean log-likelihood of samples on test set, with the standard error of the mean computed across examples. On TFD, we computed the standard error across folds of the dataset, with a different σ chosen using the validation set of each fold. On TFD, σ was cross validated on each fold and mean log-likelihood on each fold were computed. For MNIST we compare against other models of the real-valued (rather than binary) version of dataset.

- a syntetic measure, based on Parzen window estimates of P_{data}
- need for robust measure for generative models in general

Conclusions

Advantages

This model provides many advantages on deep graphical models and their alternates:

- inference becomes simple by avoiding Markov chains
- Training becomes requires only backprop of gradients
- Any differentiable function is theoretically permissible

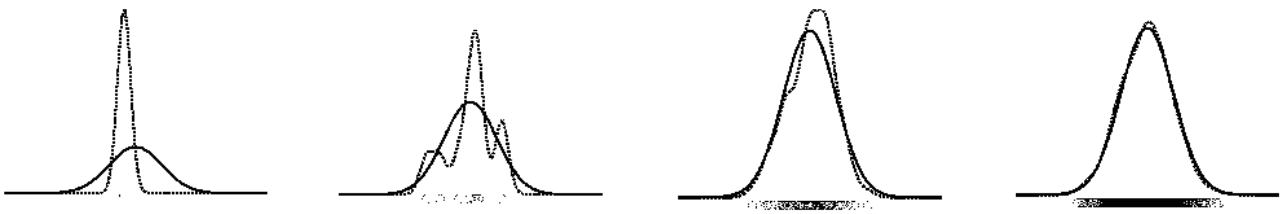
Disadvantages

Most important challenges include:

- Synchronizing the discriminator with generator
  - If G trains faster than D, it may collapse too many z to the same value of x
- there is no explicit representation of $p_g(x)$
  - approximated with Parzen density estimation

$$P(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{(h\sqrt{2\pi})^d}\exp\left(-\frac{1}{2}\left(\frac{x-x_i}{h}\right)^2\right),$$

  - Comes quite close to Gaussian for large number of samples; Plots for sample size = 1, 10, 100, 1000 [1]



[1] https://www.cs.utah.edu/~suyash/Dissertation_html/node11.html

- Generator collapses to the mean
  - https://www.youtube.com/watch?v=mObnwR-u8pc
  - https://www.youtube.com/watch?v=0r3g7-4bMYU

Interesting experiments

- Interpolation between hand written number 1 to 5

Figure 3: Digits obtained by linearly interpolating between coordinates in z space of the full model.

- Adding Glasses[2]

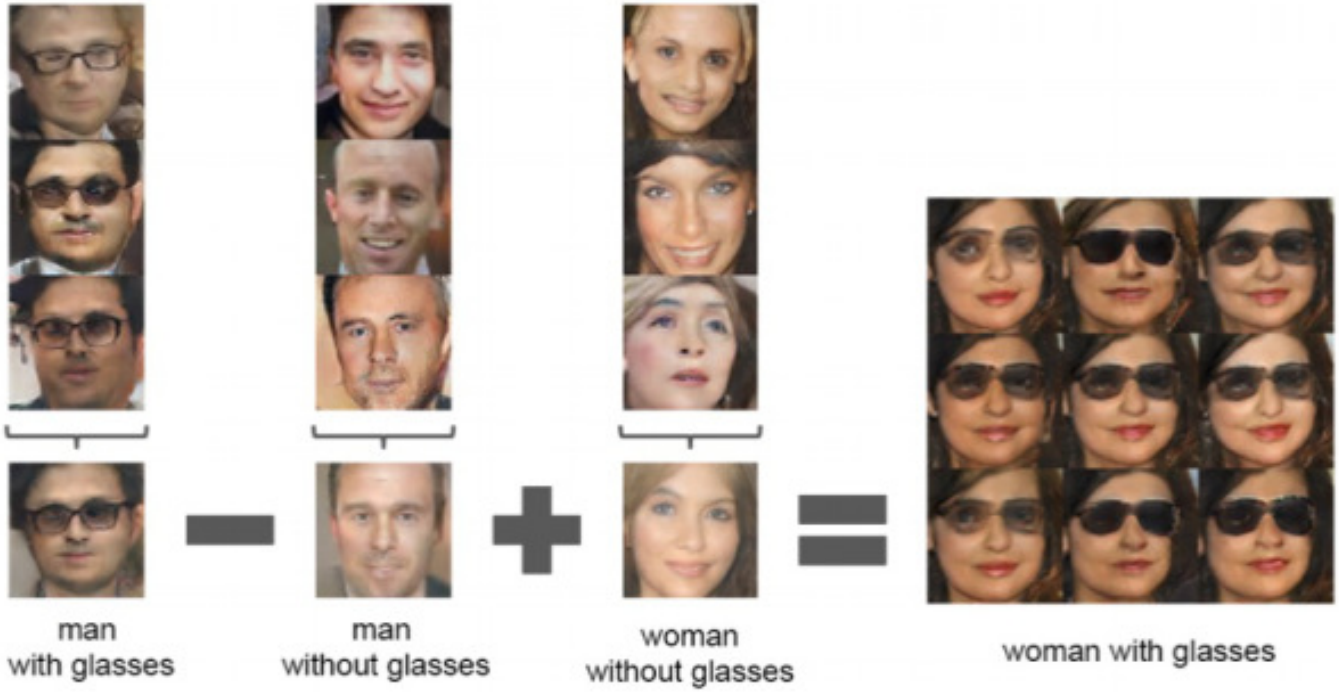


Figure 7: Vector arithmetic for visual concepts. For each column, the Z vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector Y. The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples. Applying arithmetic in the input space (bottom two examples) results in noisy overlap due to misalignment.

[2]Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. In: ICLR (2016)

- Faces turn[3]

Figure 8: A "turn" vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

[3]Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. In: ICLR (2016)