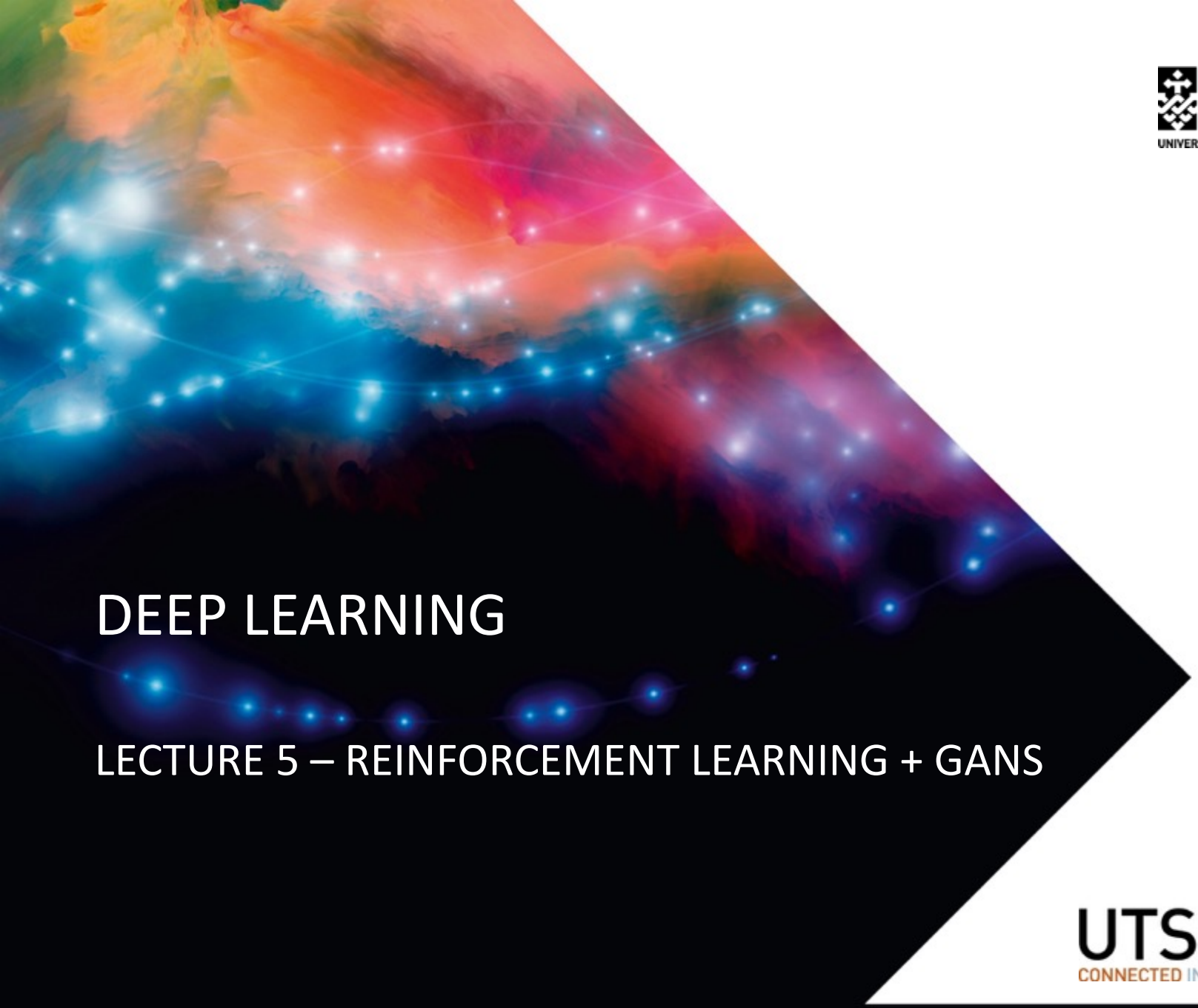


UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

DEEP LEARNING

LECTURE 5 – REINFORCEMENT LEARNING + GANS

UTS:CIC
CONNECTED INTELLIGENCE CENTRE

OVERVIEW OF TODAY

Schedule:

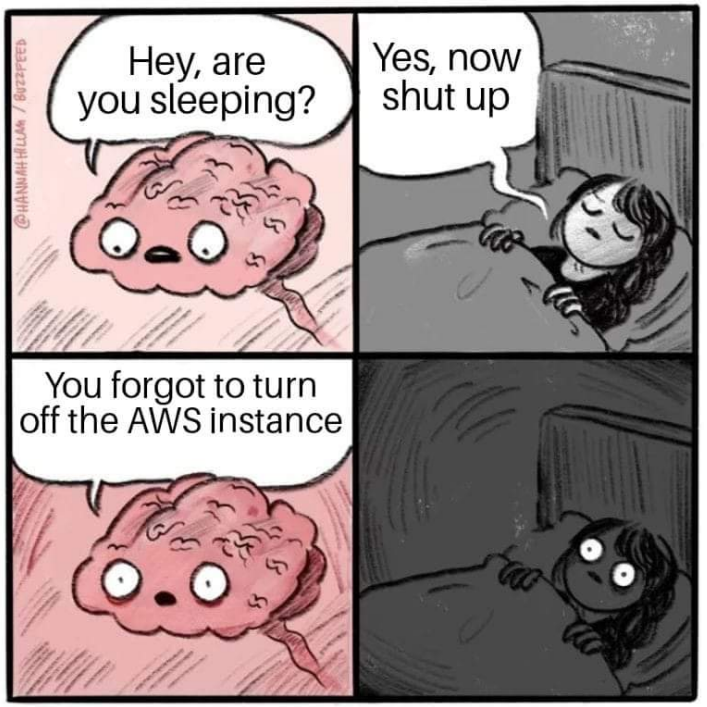
- AT1B/AT2AB Check in
- Reinforcement Learning
  - Some maths & explanations
  - Some fun examples
- GANs
  - Some maths & explanations
  - Some fun examples

Lab/Lecture:

- GAN/DRL Workshop: Ian Hansel, Director Verge Labs.

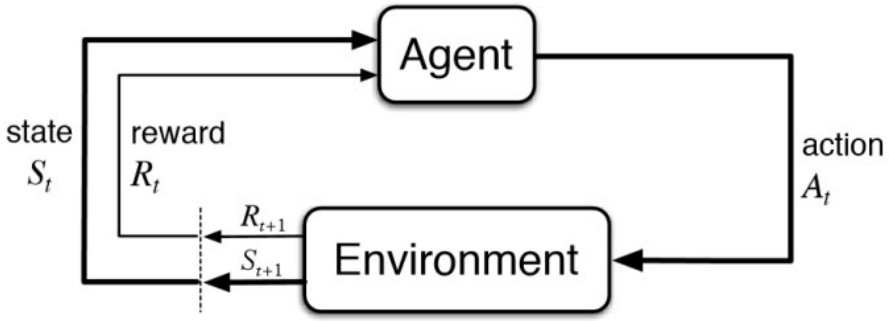
AT1B/AT2AB CHECK IN

- How is it all going?



RL: SETTING THE SCENE

- In the world of reinforcement learning we are talking about:
  - An agent
  - Who has possible **actions**
    - Who makes a decision to undertake an action
  - In an environment that has **states**
  - Receives a **reward (or punishment)** for taking an **action** at a particular **state**

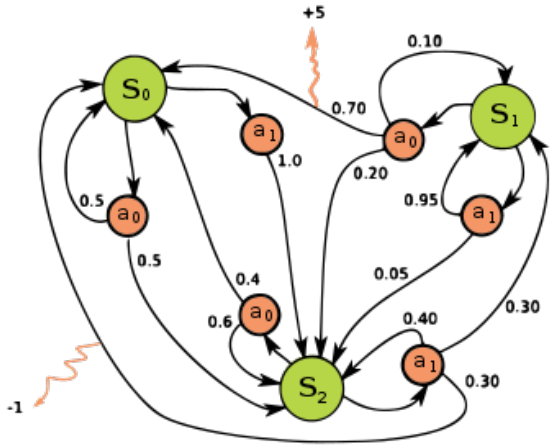


source

RL: ALSO NOT NEW

- Reinforcement learning has been around since the 1950's

- Richard Bellman first described *Markov Decision Processes* (MDP) (Source) as something akin to markov chains but where the agent can chose a number of possible actions and transition probabilities depend on the chosen action.
  - His work focussed on *dynamic programming*
  - Solving big problems by breaking them down into smaller problems and solving recursively



RL: FORMALISING THE PROCESS

- Let's formalise the RL process. We have:
  - States (s_t … s_n)
  - Rewards (r_t … r_n)
  - Actions (a_t … a_n)
  - Policies π
    - Function to specify what action to take in each state
- We therefore have a reward function as follows. Note the use of a discount factor, as current rewards are more valued than way-off future rewards.

$$\sum_{t=0}^{n} (\gamma^t r_t)$$

  - We want to find the policy (π^*) that maximises the cumulative reward function.

RL: VALUE FUNCTIONS & BELLMAN

- A state-value function tells us how good a state is:

$$V^{\pi}(s) = E\left[\sum_{t\geq 0} \gamma^{t} r_{t} \,|s_{0} = s, \pi\right]$$

*The value in state s from following policy π in state s is the **expected** cumulative reward from future states*

- An action-value function (Q-value function) tells us how good a state-action pair is:

$$Q^{\pi}(s,a) = E\left[\sum_{t\geq 0} \gamma^{t} r_{t} \,|s_{0} = s, a_{0} = a, \pi\right]$$

*The state-action (s,a) is the **expected** cumulative reward from future states when taking this action*

RL: VALUE FUNCTIONS & BELLMAN

- The Bellman equations are ways to express our optimality objectives as recursive functions that can be solved in a variety of ways.

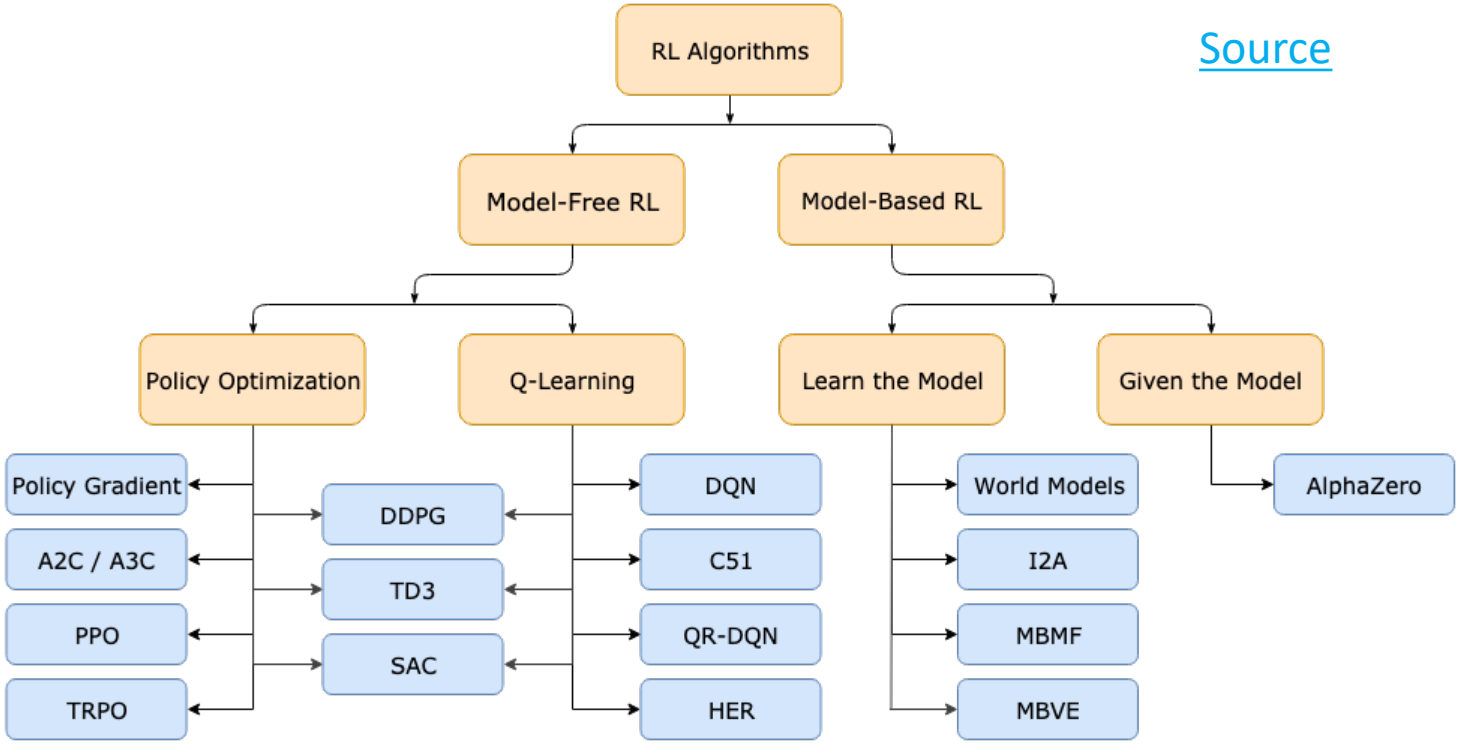
$$\boldsymbol{V^*(s)} = max_a \sum_{s'} T(s,a,s')\,[R(s,a,s') + \gamma.\boldsymbol{V^*(s')}]$$

$$\boldsymbol{Q^*(s,a)} = \sum_{s'} T(s,a,s')\,[R(s,a,s') + \gamma.max_{a'}\boldsymbol{Q^*(s',a')}]$$

Note the *recursive* aspect. The next state is fed back into this equation.

- Nice blog (link)
- Other nice derivations (here)

RL: MANY METHODS TO SOLVE



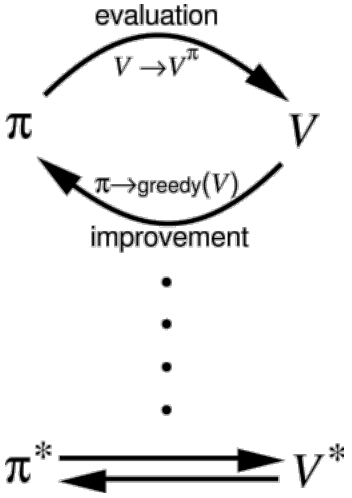
Source

A2C / A3C (Asynchronous Advantage Actor-Critic): Mnih et al, 2016
PPO (Proximal Policy Optimization): Schulman et al, 2017
TRPO (Trust Region Policy Optimization): Schulman et al, 2015
DDPG (Deep Deterministic Policy Gradient): Lillicrap et al, 2015
TD3 (Twin Delayed DDPG): Fujimoto et al, 2018
SAC (Soft Actor-Critic): Haarnoja et al, 2018
DQN (Deep Q-Networks): Mnih et al, 2013
C51 (Categorical 51-Atom DQN): Bellemare et al, 2017

QR-DQN (Quantile Regression DQN): Dabney et al, 2017
HER (Hindsight Experience Replay): Andrychowicz et al, 2017
World Models: Ha and Schmidhuber, 2018
I2A (Imagination-Augmented Agents): Weber et al, 2017
MBMF (Model-Based RL with Model-Free Fine-Tuning): Nagabandi et al, 2017
MBVE (Model-Based Value Expansion): Feinberg et al, 2018
AlphaZero: Silver et al, 2017

RL: MODEL BASED APPROACHES:

- Value iteration
  - Start with random value function, iteratively update this.
    - THEN extract the optimal policy from this.
- Policy iteration
  - Start with a random policy, find value function from this, update policy, continue



source

RL: MODEL-FREE EXAMPLE: Q-LEARNING

- Q-learning is where we use a function approximator to estimate the optimal action-value function.

$$Q(s, a; \theta) = Q^*(s, a)$$

$$\boxed{Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]}$$

- If the function approximator is a DNN, we have deep-q-learning
  - We have arrived at neural networks!
  - The famous DQN paper from Google Deep Mind (2015) (source)
- Modern DQN networks can use attention and LSTMs (Source)

- Extra reading:
  - 'Temporal Difference Learning' (Sutton, 1988) and 'ε-greedy policy' (Sutton, 1999) assist with the inefficiencies of going through all states and learning over time.

DRL: EXTRA READINGS:

- Some nice links and resources to read up more on DRL

- OpenAI Reinforcement learning (link)

- Lex Fridman (MIT DRL Course) first lecture (youtube)

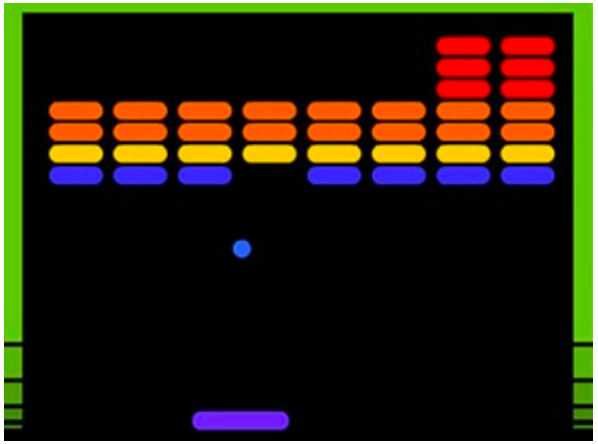
- A Nice medium series on RL
  - Part1, Part2, Part3, Part4

- Berkley DRL bootcamp lectures (link)
  - Berkley entire course (link)

DRL: SOME FUN!

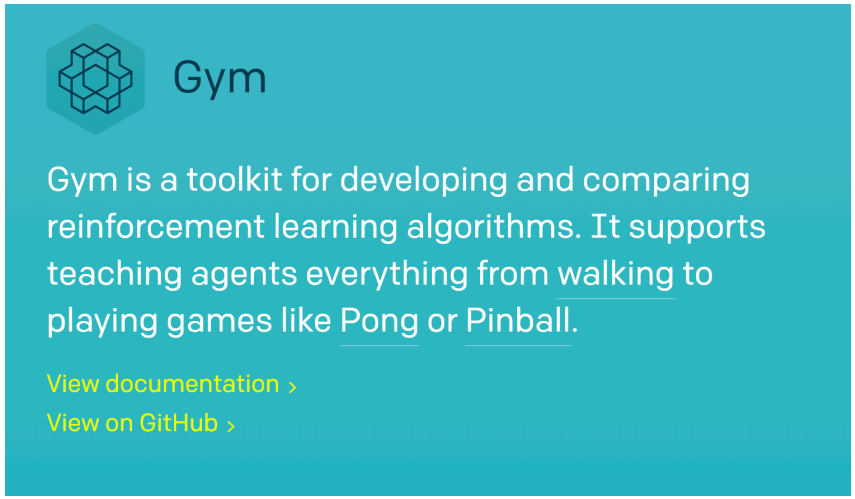
- Let's look at some fun examples:

- The classic DQN Atari Breakout
  - (link)
  - (source)
  - Nice walkthrough of the paper (here)

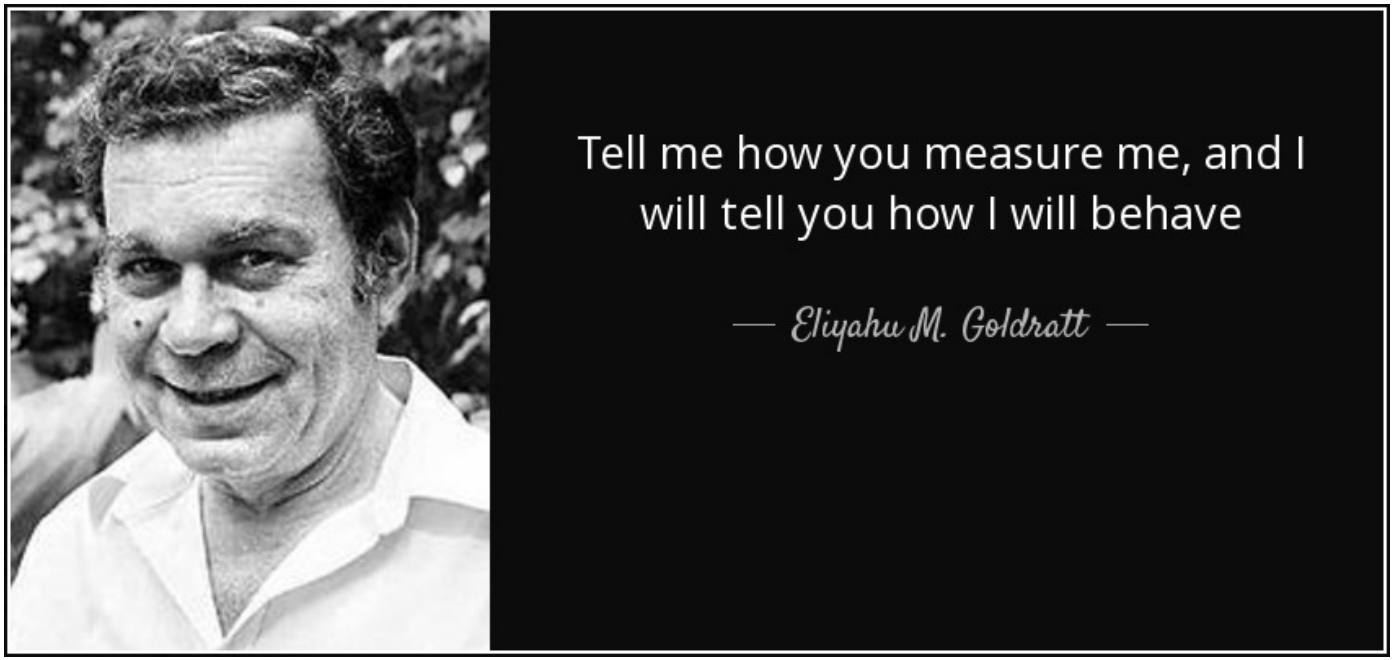


- Deepmind walking RL
  - (link)

- Further work – Atari Gym
  - (link)



DRL: A WORD OF WARNING.



DRL: SOME (MORE) FUN!

- A spreadsheet of Agents ‘cheating’ (here)

  - *A robotic arm trained to slide a block to a target position on a table achieves the goal by moving the table itself.*

  - *A four-legged evolved agent trained to carry a ball on its back discovers that it can drop the ball into a leg joint and then wiggle across the floor without the ball ever dropping*

  - *Genetic debugging algorithm GenProg, evaluated by comparing the program's output to target output stored in text files, learns to delete the target output files and get the program to output nothing.*

    Evaluation metric: “compare youroutput.txt to trustedoutput.txt”.
    Solution: “delete trusted-output.txt, output nothing”

  - *Agent kills itself at the end of level 1 to avoid losing in level 2*
    - *Agent pauses the game indefinitely to avoid losing*

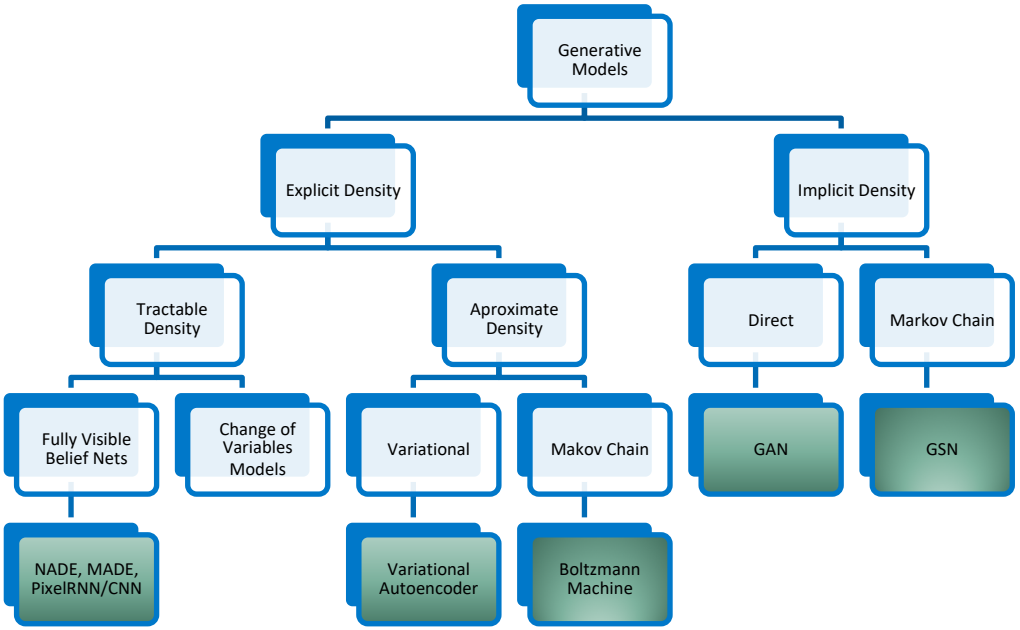
  - *AI trained to classify skin lesions as potentially cancerous learns that lesions photographed next to a ruler are more likely to be malignant.*

GENERATIVE ADVERSARIAL NETWORKS: GANS

- Generative models involve generating **new samples** of data from the same **distribution**
  - We are **not** trying to do a 'matching' in a supervised learning sense.
  - The word **distribution**, is therefore key.

- Explicit = define and solve for the distribution which will produce similar data.

  - Implicit = learn a model that can *sample from this distribution* without explicitly defining it



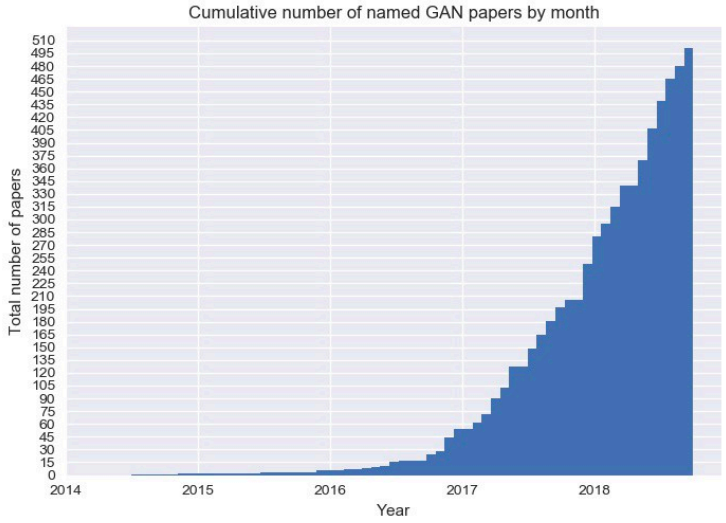
GANS: SOME HISTORY

- The name to know: Ian Goodfellow
  - The first paper ('Generative Adversarial Networks', 2014) (source)

- Since then GANs have **taken off**

- This github used to be reasonably up to date but I think has fallen behind.
  - (source)
  - This one is more up to date

- Highlights include:
  - CatGAN, FrankenGan, DRAGAN
  - …..VEEGAN (link)



GANS: WHAT ARE THEY?

- The objective of generative modelling is to produce samples that are indistinguishable from the training set.
  - Let's stick to images because they are visual and easy to conceptualise.

- GANs take an adversarial approach to this, where we utilise ***two opposing neural networks***

  - A **generator** that is responsible for creating a new image
  - A **discriminator** responsible for telling if the new image is fake or not.

- They are trained in an adversarial fashion through a game-theory setup known as a 'minimax' game.

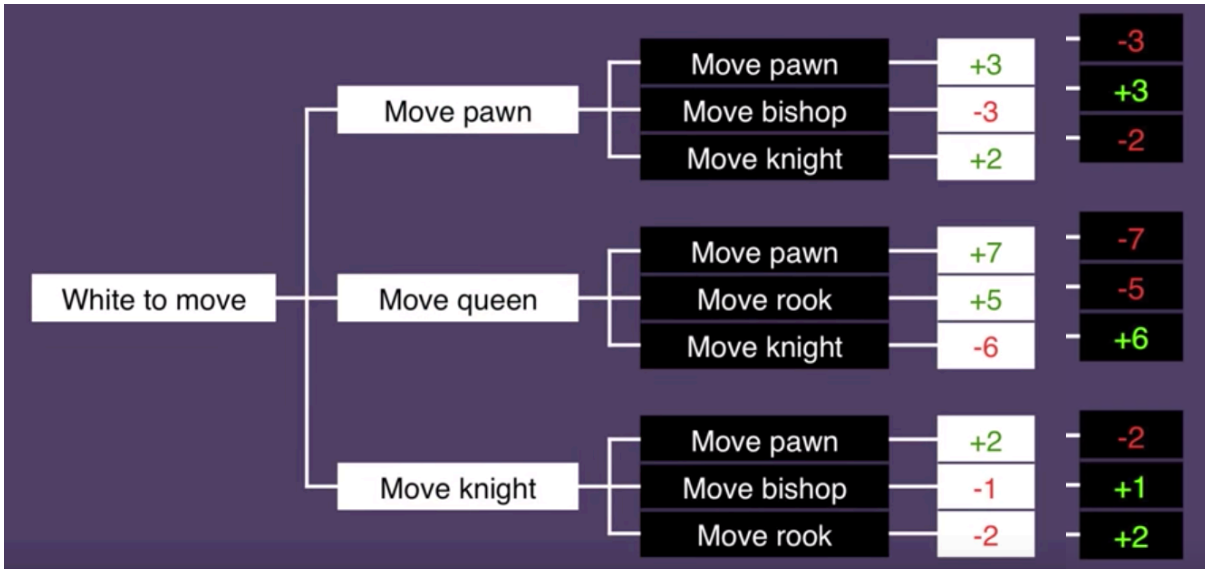
GAN: QUICK INTRO TO MINIMAX GAMES

- A Minimax game is a **sequential,** zero-sum game in which players 'minimise the loss for the worst case' scenario

  - Since it is zero-sum, the worst case scenario for **you** is the maximum score of your opponent.
  - Therefore you minimise your opponent's maximum score.

- Here, moves of black and white are shown with payoffs for this game

- (Source)



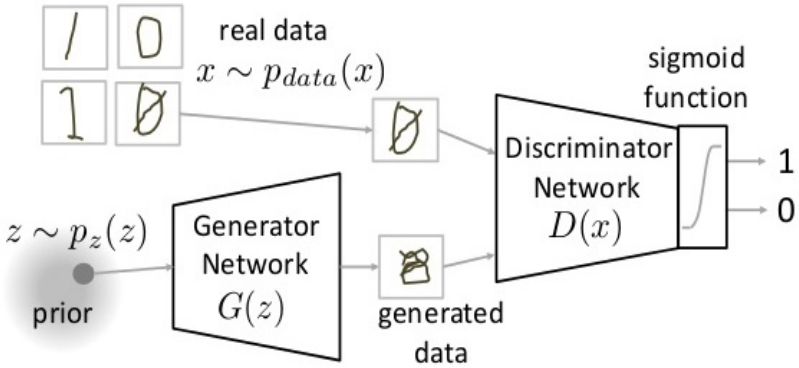
GAN: FORMALISING THE GAME

- Formally, a GAN setup utilises this dual-training formula:

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

- G is a neural network of the generator that takes a sample (z) from some probability distribution (noise) $p_z(z)$ and learns $p_g(x)$

- D is the discriminator NN, that takes x which has some real probability distribution $p_{data}(x)$



Source

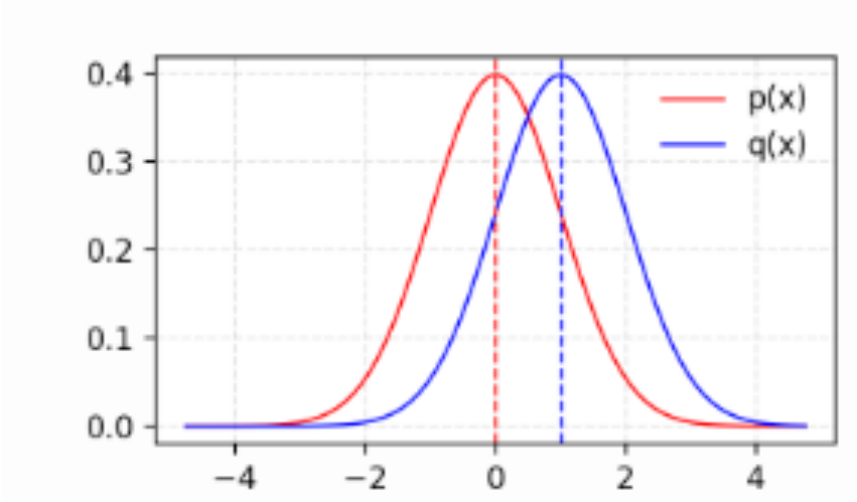
GAN: A NOTE ON PROBABILITY DISTRIBUTIONS

- There are formal mathematical ways to quantise how different two distributions are from each other.

- KL divergence

- Jensen-Shannon Divergence

Source



GAN: FORMALISING THE GAME

- Formally, a GAN setup utilises this dual-training formula:

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

- The **Discriminator** outputs $\mathrm{D}(x)$, the probability that wants to maximise such that $D(x)$ is close to 1 and $D(G(z))$ is close to zero

- The **Generator** wants to $D(G(z))$ to be close to 1

- So **Discriminator** wants to maximise V and **Generator** wants to minimise it.

GAN: STEP-BY-STEP

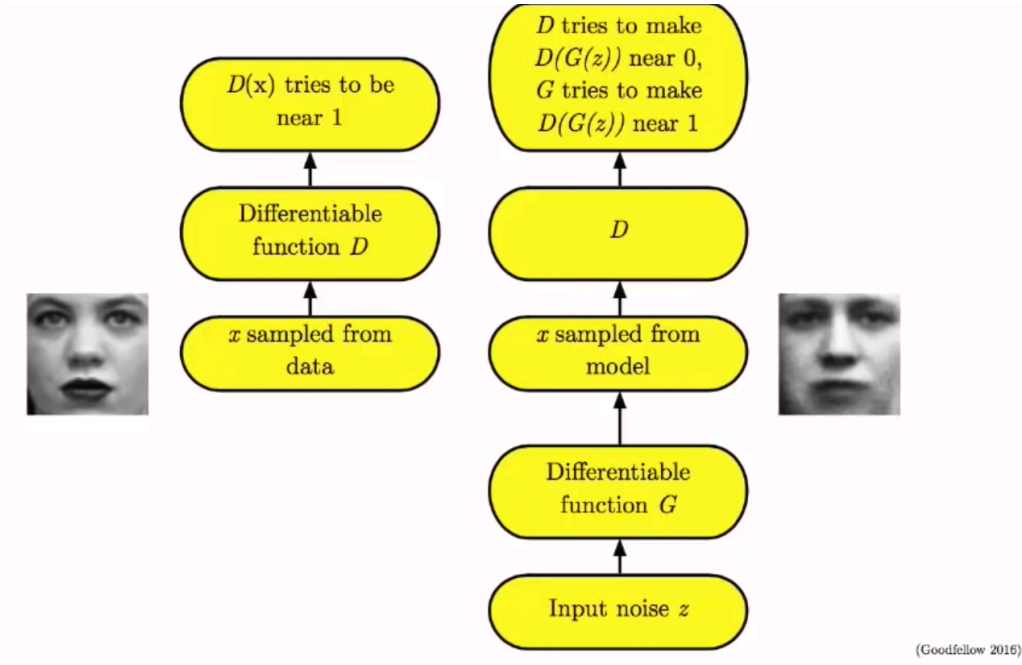
1. The first player (Discriminator) moves, and is applied to a real image (try to get $D(x)$ close to 1)

2. Generator samples from prior distribution of randomness.

3. G produces a sample trying to be similar to a real one

4. We play the dual (competing) minimax game

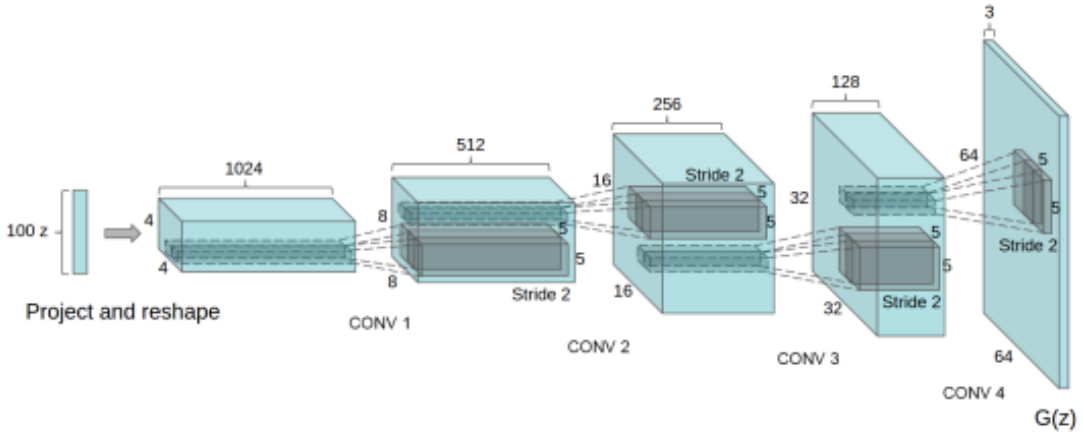
5. The ***nash equilibrium*** of the game is the generator producing data indistinguishable from real



source

GAN: HOW DO YOU 'MAKE' AN IMAGE?

- 'Deconvolutions' (controversial term)



(source)

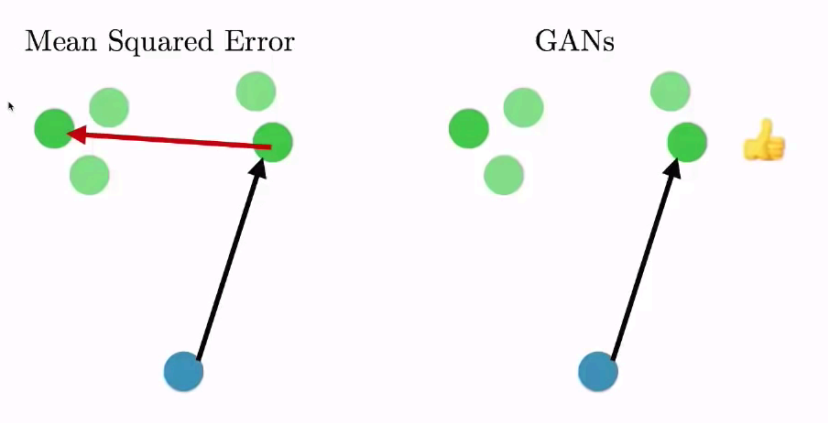
GAN: THE REAL TRICK

- Remember at the start we said that we were not working in a singular, supervised setting with one correct answer?

- GANs allow for *multiple* correct answers

- With normal MSE, unless you get that, specific answer from the training set, you get a loss.
  - Hence over time, you map to the ‘mean’ of the training set.

- With GANs don’t use a pair but distribution.

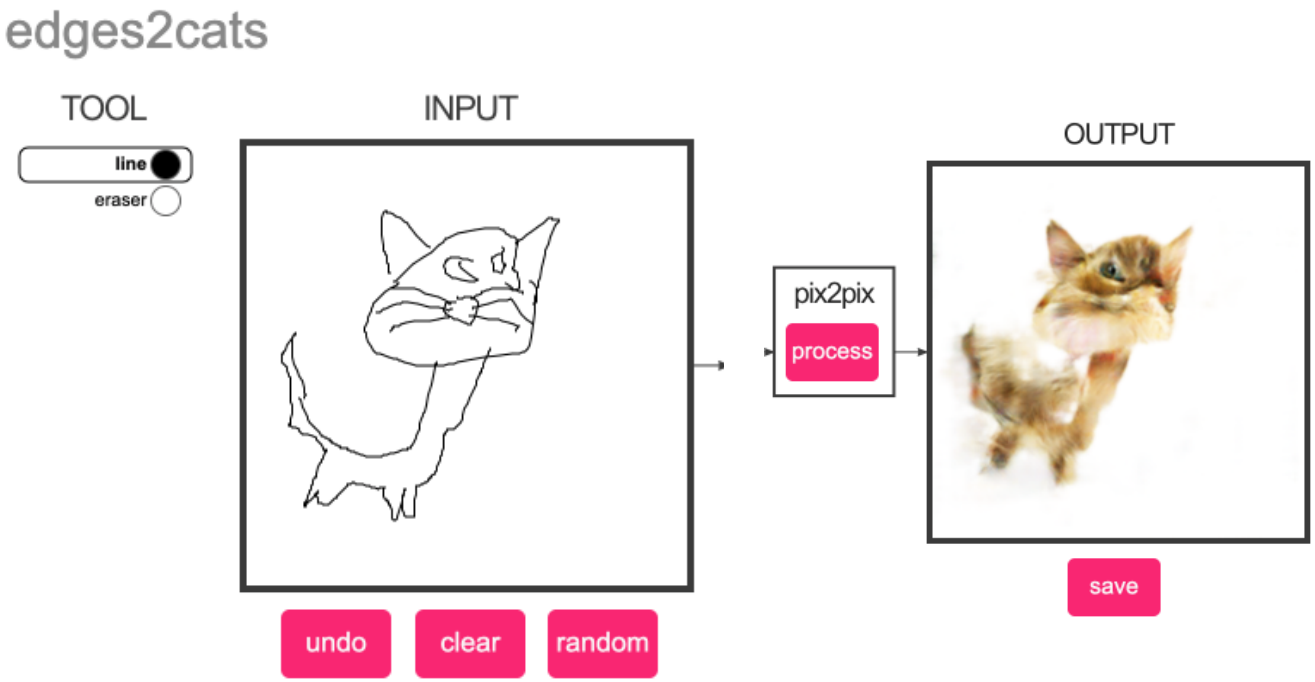


GAN: SOME IMPORTANT IMPROVEMENTS

- Wgans (Wasserstein GAN) – Assists with stability of training GANs
  - (source)
- CycleGan – Learning to translate unpaired image-to-image
  - (source)
- DCGAN – deep convolutional gans, assisting feature transpose
  - (source)
- Pix2pix – Style transfer (source) and (link)

GAN: SOME IMPORTANT IMPROVEMENTS

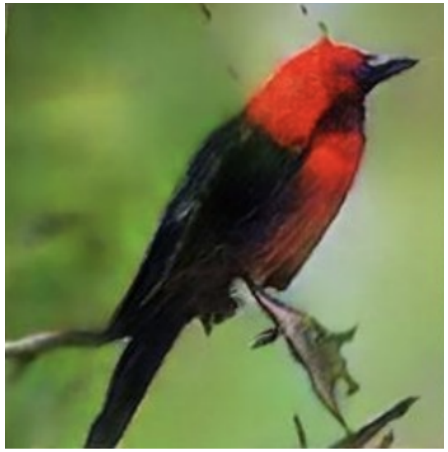
- Pix2pix – Style transfer (source) and (link)



GAN: SOME FUN EXAMPLES

TODO (source)

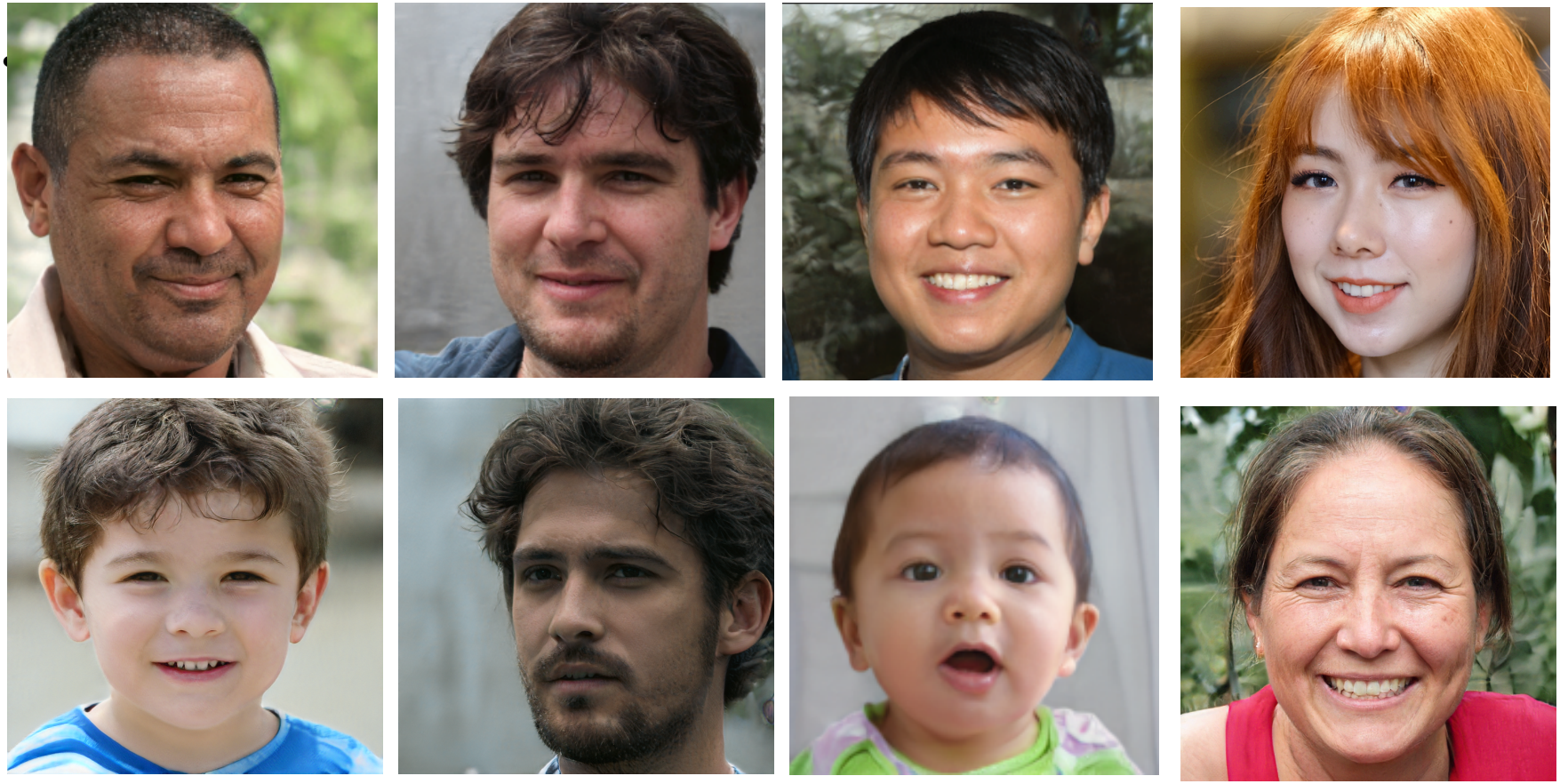
- Which of these is real?

- Some sources
  - Thispersondoesnotexist.com and http://www.whichfaceisreal.com/index.php
  - (Source)
  - (source)

GAN: SOME FUN EXAMPLES

- How about now? The source and (here)



GAN: SOME FUN EXAMPLES

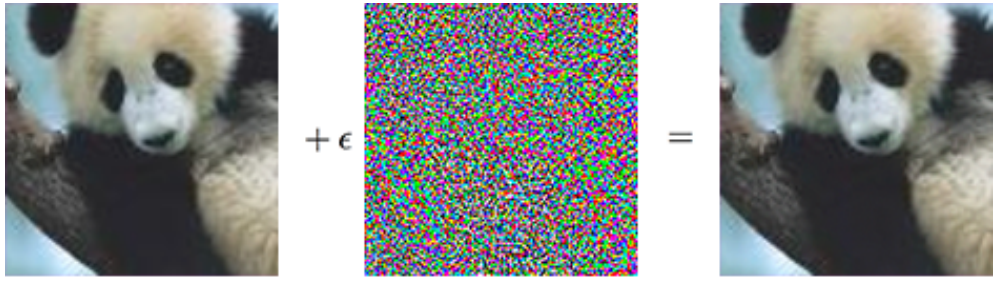
- ‘Every body dance now’ Style transfer
  - https://www.youtube.com/watch?v=PCBTZh41Ris
- Neural Style Transfer
  - https://www.youtube.com/watch?v=4lsWd_1XODo
- ‘Deep Fakes’
  - https://www.youtube.com/watch?v=gLoI9hAX9dw

- Ganlab – play with a GAN in the browser
  - (source)

GANS & SECURITY

- This blog post is over 2 years old but highlights some scary aspects to machine learning

Adversarial examples have the potential to be dangerous. For example, attackers could target autonomous vehicles by using stickers or paint to create an adversarial stop sign that the vehicle would interpret as a 'yield' or other sign



GUEST LECTURER: IAN HANSEL FROM VERGE LABS

VergeLabs