Modern Game AI Algorithms
Artificial Neural Networks & Learning from Pixels

Mike Preuss | LIACS

Universiteit Leiden
The Netherlands

roadmap

lecture plan (Thursdays 14:15), https://smart.newrow.com/room/nr2/?room_id=wft-364

Feb	10	introduction	Mar	31	player experience modeling
	17	learning and optimization	Apr	7	believable behavior
	24	procedural content generation I		14	team AI and mass behavior
Mar	3	Monte Carlo tree search		21	realtime strategy AI
	10	experimentation		28	(game) AI outlook with Zhao Yang
	17	ANN & learning from pixels	May	12	free project presentations 1
	24	procedural content generation II		19	free project presentations 2

labs plan (Thursdays 16:15)

Feb	10	no lab	Mar	31	assignment 2
	17	single assignment: Minecraft house	Apr	7	deadline ass. 2 / start free proj
	24	single assignment (week 2)		14	free projects (assignment 3)
Mar	3	single assignment (week 3)		21	free projects (assignment 3)
	10	single assignment (extension)		28	free projects (assignment 3)
	17	single ass. deadline / start ass 2	May	12	
	24	assignment 2		26	free project submission deadline

assessment

what is graded:

- single assignment 25%, smaller group assignment 30%,
  free project assignment 45% (including presentation), no written exam

the single assignment (deadline today):

- simple assignment uses Python based GUI tool for Minecraft add-ons
- task is to procedurally create a house

smaller group project (since last week):

- group work, new environment for strategy games, will have internal competition
- assumed working groups approx. 3 people, exceptions possible

free group project (free means you choose the topic):

- working in small groups (4-5 people, exceptions possible)
- during the second half of the labs, and in your preparation time

deep stuff we look at today

- ANN primer
- deep learning motivation
- generative adversarial networks (GAN)
- deep reinforcement learning
- world models

picture from Andy Bay on Pixabay

artificial neural networks primer

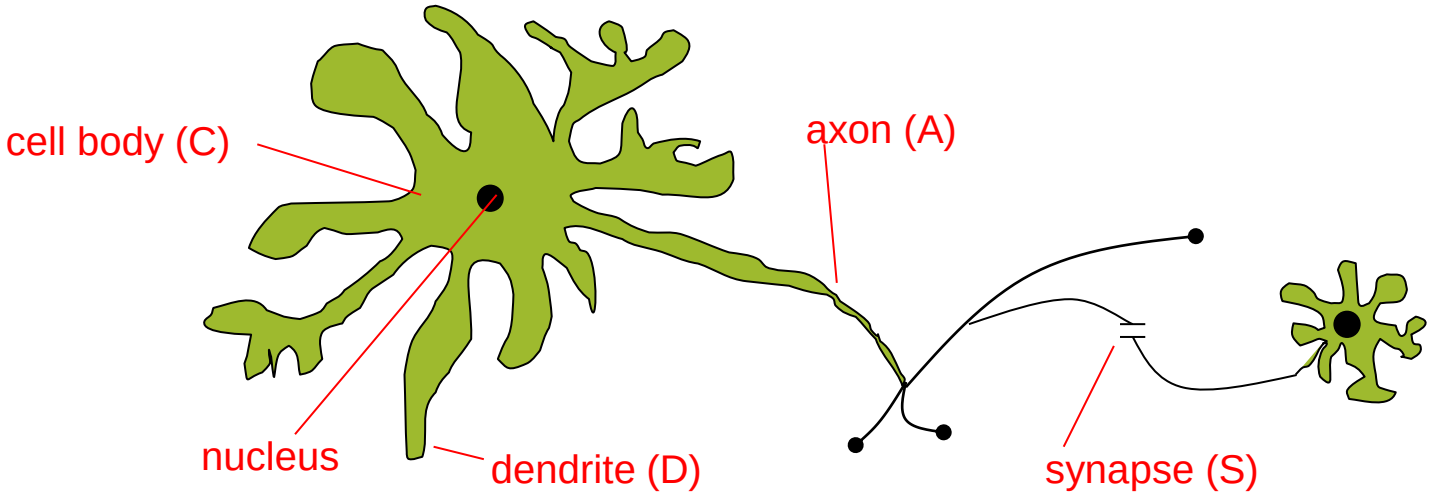
imitating biology (neuron):

- collect information (D)
- process information (C)
- distribute information (A / S)

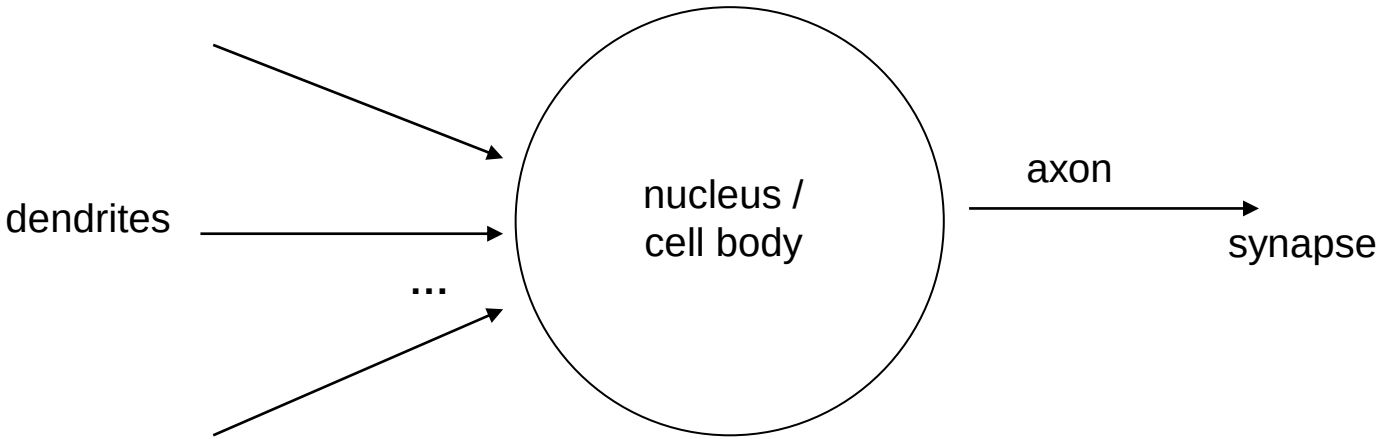
human: 10^{12} neurons

electricity in mV range

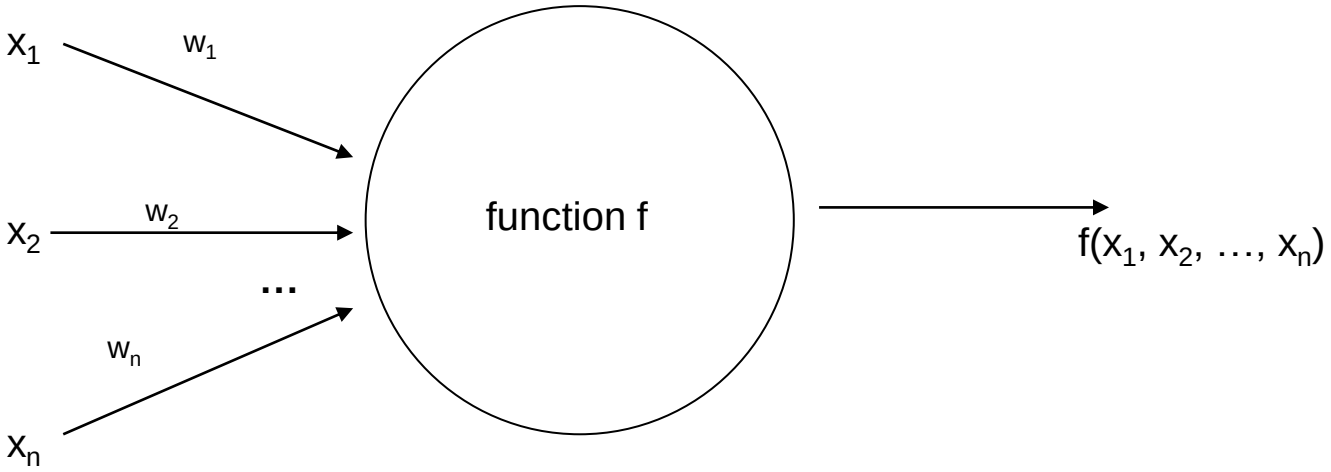
speed: 120 m/s



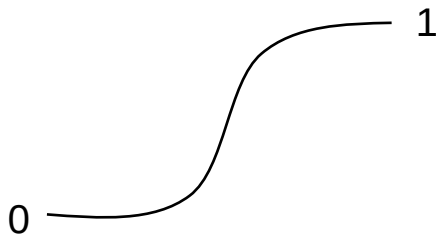
abstraction



single-layer perceptron (SLP)

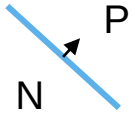


- f = activation function,
  used to be 0/1 decision with threshold
- better: sigmoid function (differentiable)

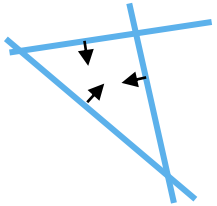


why an additional layer?

- Single-Layer Perceptron (SLP)
  hyper plane separates space in 2 subspaces



- Two-Layer Perceptron
  can represent any convex set



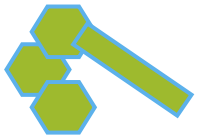
connected by AND gate in 2nd layer

- Three-Layer Perceptron
  represents arbitrary sets

several convex sets are represented in the 2nd layer and get combined in the 3rd layer

we do not need more than 3 layers

however, deep learning does exactly that!



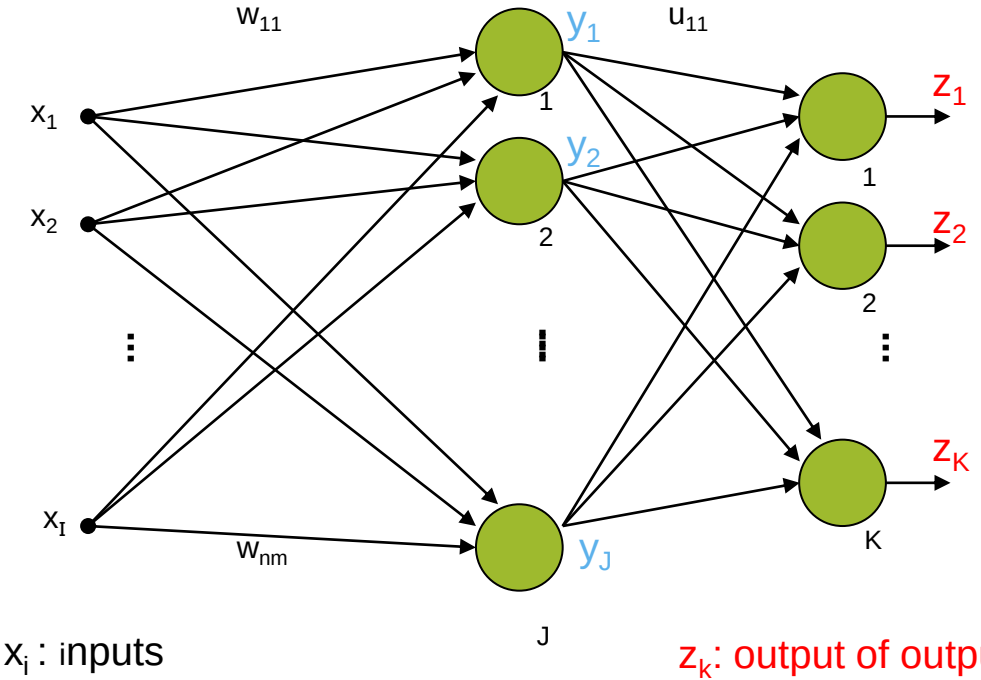
convex sets in 2nd layer, connected via OR gate in 3rd layer

multi-layer perceptron (MLP)

- inputs are passed on through multiple layers to the output nodes
- node output =

$$o_j = f(\sum_i w_{ij} \cdot x_i)$$

- output of one layer is input for next layer
- network is trained with examples (supervised)
- training means to adapt the weights (w and u)



x_i : inputs

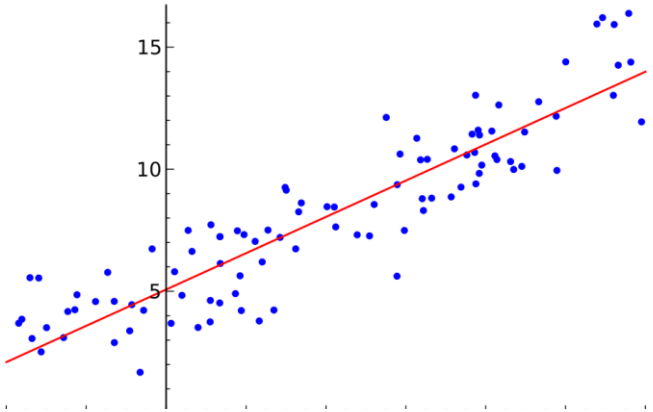
z_k: output of output nodes

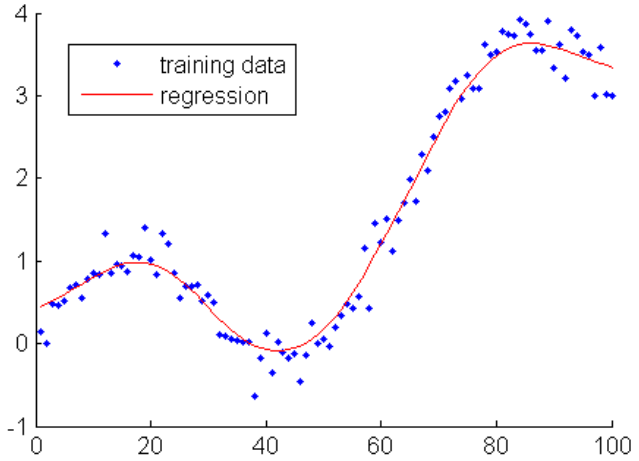
y_j : output values of hidden layer

modeling with ann: function approximation

what kind of model?

- we can do function approximation (sort of regression) of analytically not tractable functions
- this only makes sense (learning is expensive) for functions much more complicated than linear
- could be results from a simulation
- typically learned from training data (input-output pairs that resemble the “ground truth”)
- input values are provided to input neurons, output error is measured



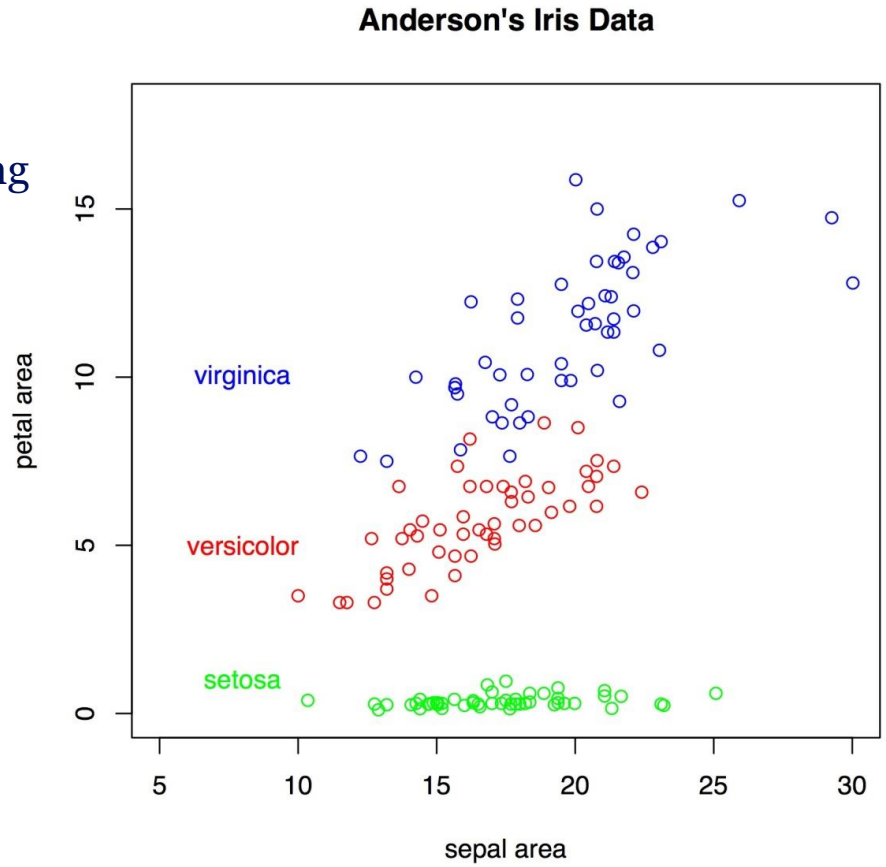


modeling with ann: classification

related use (discrete output): classification

- if we have training data -> supervised learning
- often one output node per class
- example use cases:
  - handwriting recognition
  - spam filtering
  - decision making

try out examples at:
http://cs.stanford.edu/people/karpathy/convnetjs/

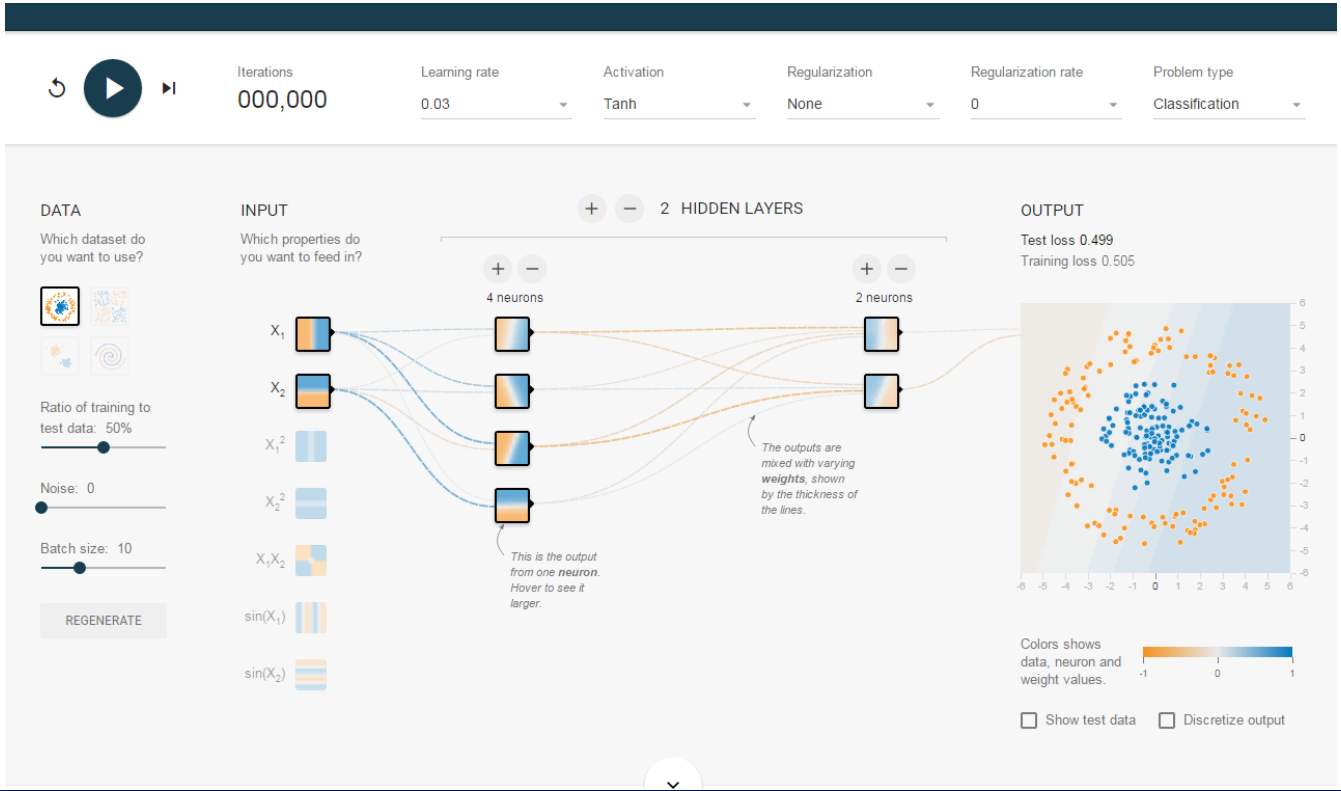


practical example

go to page: http://playground.tensorflow.org/

task: setup a minimal network with test loss 0.000

record number of neurons (without output) and number of iterations



adjusting weights

- how can we adjust all weights in an MLP?
- Rumelhart, Hinton and Williams (1986): backpropagation algorithm
- we assume supervised training with some training patterns B
- idea: measure error between desired outputs (from training patterns) and actual output and propagate the error through the network

Quantification of classification error of MLP:

- Total Sum Squared Error (TSSE)

$$f(w) = \sum_{x \in B} \| \underbrace{g(w; x)}_{\text{output of net for weights w and input x}} - \underbrace{g^*(x)}_{\text{target output of net for input x}} \|^2$$

- Total Mean Squared Error (TMSE)

$$f(w) = \frac{1}{|B| \cdot \ell} \sum_{x \in B} \| g(w; x) - g^*(x) \|^2 = \underbrace{\frac{1}{|B| \cdot \ell}}_{\text{const.}} \cdot \text{TSSE}$$

training patters

output neurons

=>leads to same solution as TSSE

backpropagation algorithm (batch mode)

parameters:

sets of nodes, connections, weights (=network),
training examples, γ (learning rate), stopping criterion

algorithm:

1. initialize network weights randomly
2. **do**
   3. prediction: compute network output for all examples
   4. compute error (TSSE) for examples at the output units
   5. **for** layer from last hidden layer to input layer
      6. compute Δweight for weights from current to next layer
   7. **end for**
   8. update network weights
9. **until** stopping criterion satisfied
10. **return** network

note: works similar online, then update weights for every example

activation function

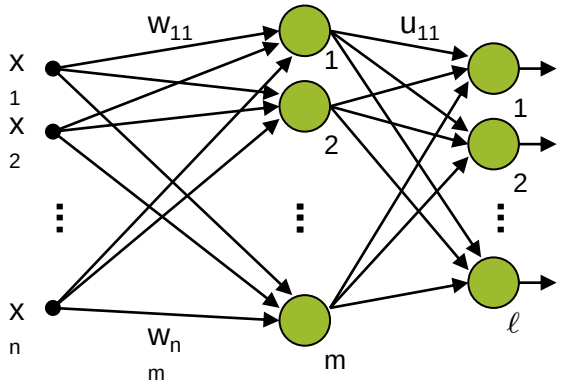
example: weight adaptation for 2-layer MLP

idea: minimize error!

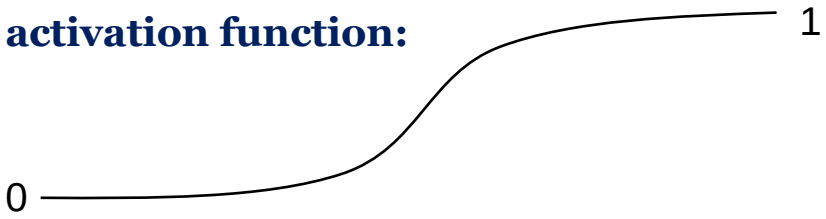
$f(w_t, u_t) = \text{TSSE} \rightarrow \text{min!}$

gradient method

$$u_{t+1} = u_t - \gamma \nabla_u f(w_t, u_t)$$
$$w_{t+1} = w_t - \gamma \nabla_w f(w_t, u_t)$$



activation function:



- monotonously increasing
- differentiable
- non-linear

e.g.:

- $a(x) = \dfrac{1}{1+e^{-x}}$ $\quad a'(x) = a(x)(1-a(x))$
- $a(x) = \tanh(x)$ $\quad a'(x) = (1-a^2(x))$

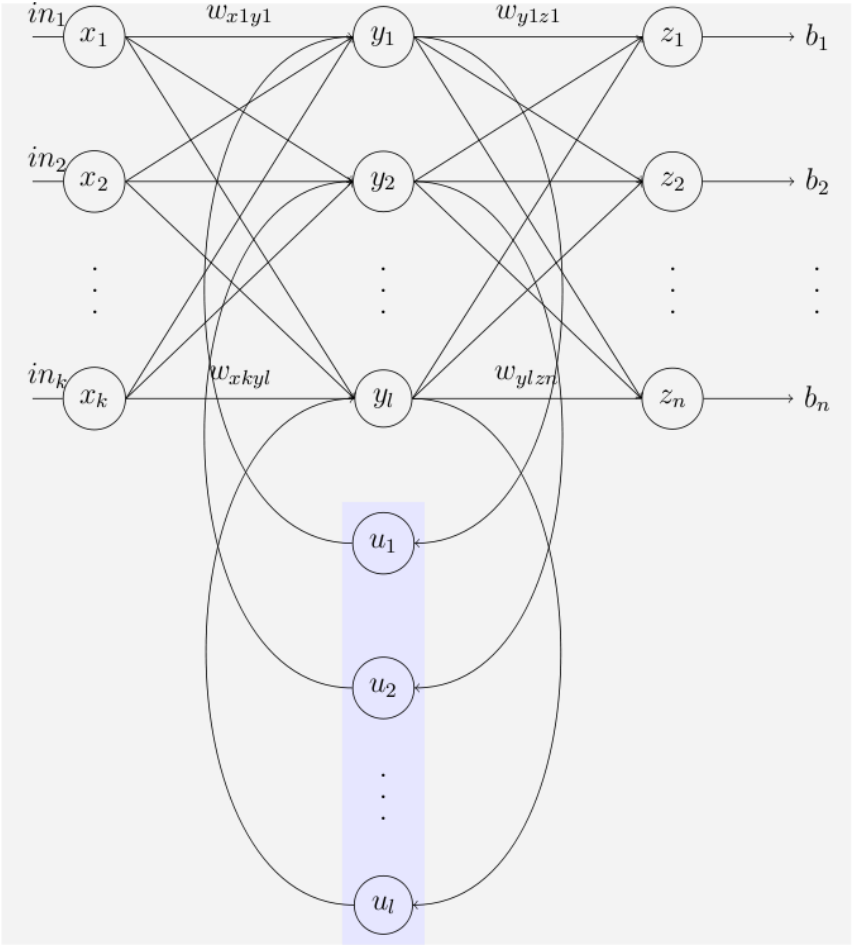
values of derivatives directly determinable from function values

recurrent neural networks

- Elman networks of 1990: addition of “context” neurons
- recurrent networks make the state history available
- many different possibilities

problems:

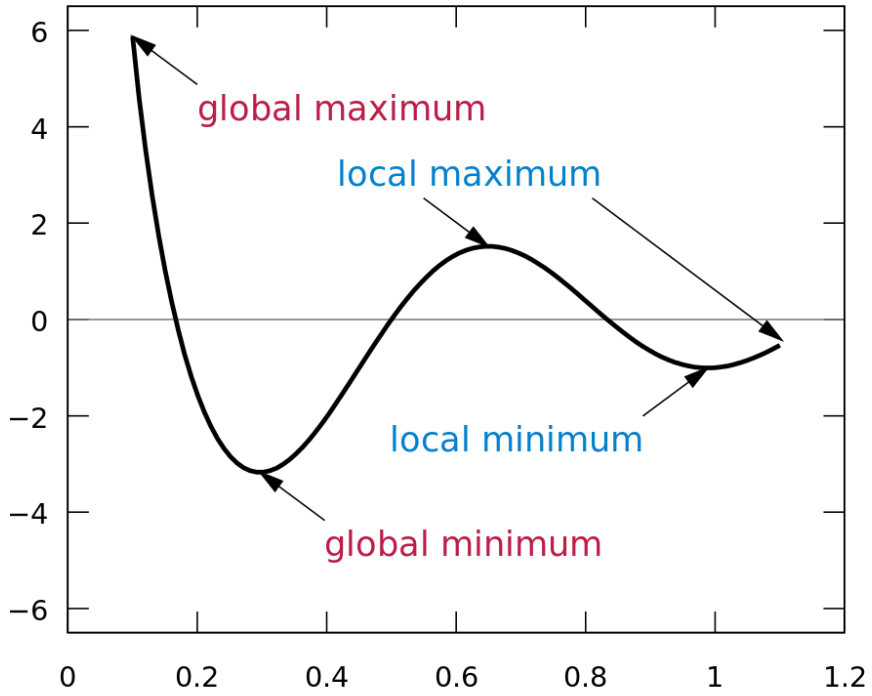
- vanishing gradients – bad for learning with backpropagation
- forgetting is necessary



learning methods and limitations

- backpropagation uses gradient descent
- this may lead to locally optimal solution
- global optimization methods as evolutionary algorithms can prevent that (but take time: large sets of weights!)

- especially for small sets of training examples: danger of overfitting
- we want to learn relationship behind trainig data, not training data itself!
- separate data into training and test set (usually 2/3 and 1/3)
- do not measure on learning (training) data!



using artificial neural networks in practice

- structure: how many layers, how many neurons -> NAS (neural architecture search)
- parameters: learning rate, stopping criterion?
- partly black magic: rules of thumb, but you need to experiment!

what to use as input for the ANN? (some hints)

- normalize continuous data in [0,1] or apply z-transform
  (subtract the mean, divide by the standard deviation)
- transform discrete class values into binary inputs (one per class)
- e.g., selected weapon {knife, shotgun,bow} turns into knife{0,1}, shotgun{0,1}, bow{0,1}
- USE DOMAIN KNOWLEDGE!!!

ANN: the application side

- learned networks are getting very complex
- see impressive comparison of different, mostly ANN-based methods on handwriting recognition (classification) at: http://yann.lecun.com/exdb/mnist/index.html
- ANN models are black-box: no explanatory use!
  (as, e.g., random forests)

- the BotPrize was an industry-sponsored competition based on Unreal Tournament 2004 for the most human-like bot
- one of 2 teams who finally won the BotPrize in 2012 heavily relied on ANN for imitating human behavior
  http://nn.cs.utexas.edu/?ut2

Left-top: Mihai Polceanu
Middle: Philip Hingston
Bottom-right: UT^2 team

deep learning -> visualized

- deep neural networks get popular since around 2009
- partly because GPU power makes backprop for huge networks feasible
- basic idea: we use more hidden layers and huge networks
- by this we build layers of features with increasing levels of abstraction
- also: use ensembles of deep neural networks
- top results for difficult tasks as handwriting/speech recognition etc.

deep dreaming (Google):

- already there: image recognition ANN
- task of the process: what does the ANN (layers) see?

1. feed an image to an ANN
2. pick a layer and enhance what this layer detected
3. data is fed into the ANN again to produce an image based on what that layer saw

deep dreams

create your own: https://dreamscopeapp.com/deep-dream-generator

picture by Lorenzo Tlacaelel

why do we need more layers ?

- usually, we have only one hidden layer
- in principle, we can characterize any set with 3 layers, so why more?

because of modularization:

- it is much easier to achieve something by composing simple parts than by doing everything from scratch
- recent works show that there are many functions that can be replicated with 3 layers, but at a **much** higher cost (nodes) than with more layers

because our computers can do (now):

- modern computers can deal with huge data sets (needed for DL!)
- we can use GPUs and extreme parallelization

recommended: lecture by Yoshua Bengio

http://videolectures.net/deeplearning2015_bengio_theoretical_motivations/

the curse of dimensions

for many interesting tasks, we have high-dimensional input:

- image recognition (every pixel is an input!)
- natural language processing
- many related complex problems are high-dimensional

in high dimensions, we encounter new problems:

- every data set gets sparse (combinatorics)
- distance measures hardly work any more
- nearest neighbors get almost meaningless
- we have no good intuition of what happens

picture from 024-657-834 on Pixabay

weight learning complexity

- deep learning networks have lots of weights
- backpropagation does not work in the original way
- gradients can explode or vanish (Diploma thesis of Hochreiter, 1991)
- methods to cope with that:
  - brute force
  - various extensions, as LSTM (long short term memory)

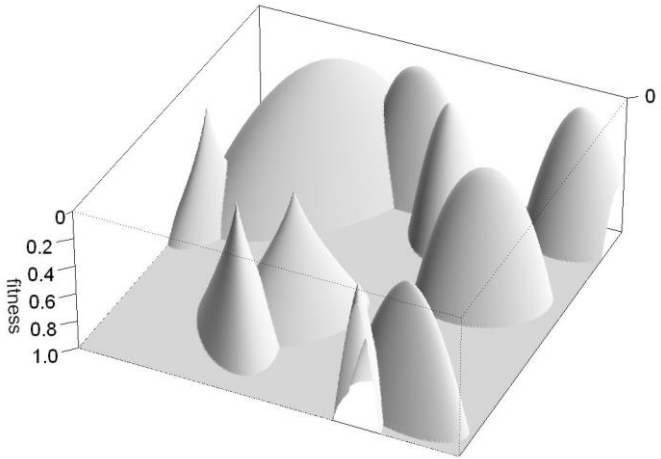
- generally, the weight optimization problem is non-convex
- that means, classic optimization methods do not obtain global optima
- recent attempts to do weight learning with evolutionary algorithms only partly successful

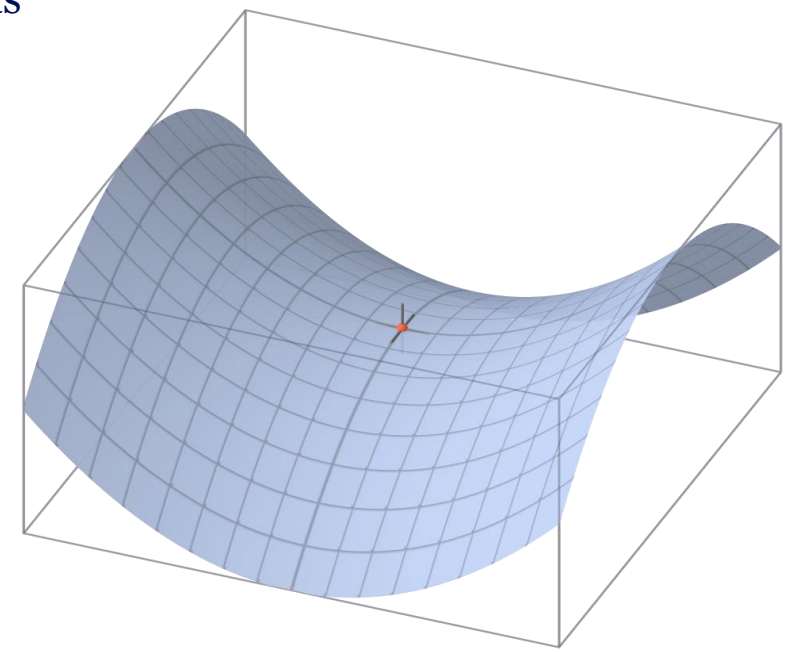
picture from Pexels on Pixabay

why non-convexity may not be a problem

recent results (experimental and theoretical) lead to the conjecture that:

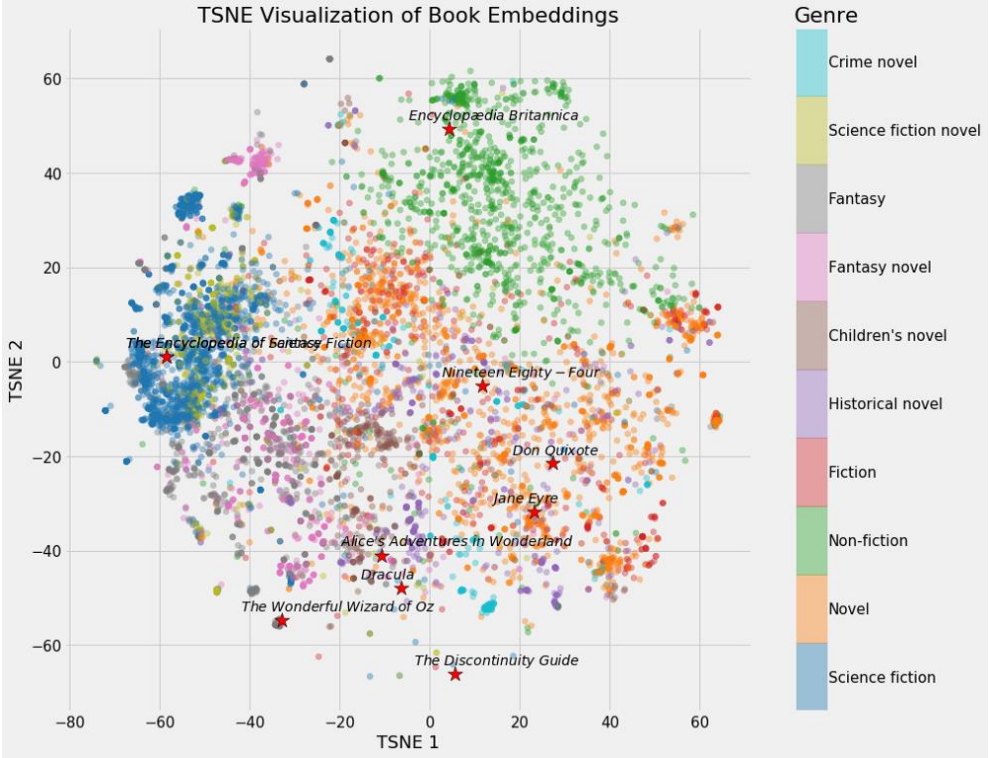
- it may be ok if we find only a local optimum
- because local optima are already quite good
- they can be found only where gradients in all dimensions lead into the same direction
- most zero gradient points are saddle points





embedding

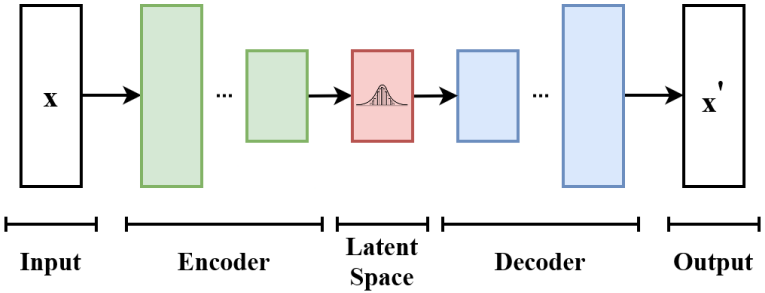
- generally often applied method in deep learning
- basically a dimension reduction, needed because ANN have fixed dimension input
- can deal with categorical data in arbitrary dimensions/sizes
- target space still too large to show, e.g. 32 dimensions
- TSNE for visualization
- we can find “neighbors” with approximate-NN algorithms fast



https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

(variational) autoencoder

- we force the network to map the input into a latent space (encode)
- and then to map it back to the original input space (decode)
- this gives us another representation of the input space (usually much smaller)
- together with a way to go back from latent space to input/output space
- the latent space is presumably “ordered” somehow, we can use it for interpolation
- variational autoencoders make an attempt to better “order” the latent space



picture from EugenioTL, creative commons 4

overview: deep learning in video games

Deep Learning for Video Game Playing. Niels Justesen, Philip Bontrager, Julian Togelius, Sebastian Risi, arXiv, 2017

getting a bit old meanwhile, but seemingly no new survey available...

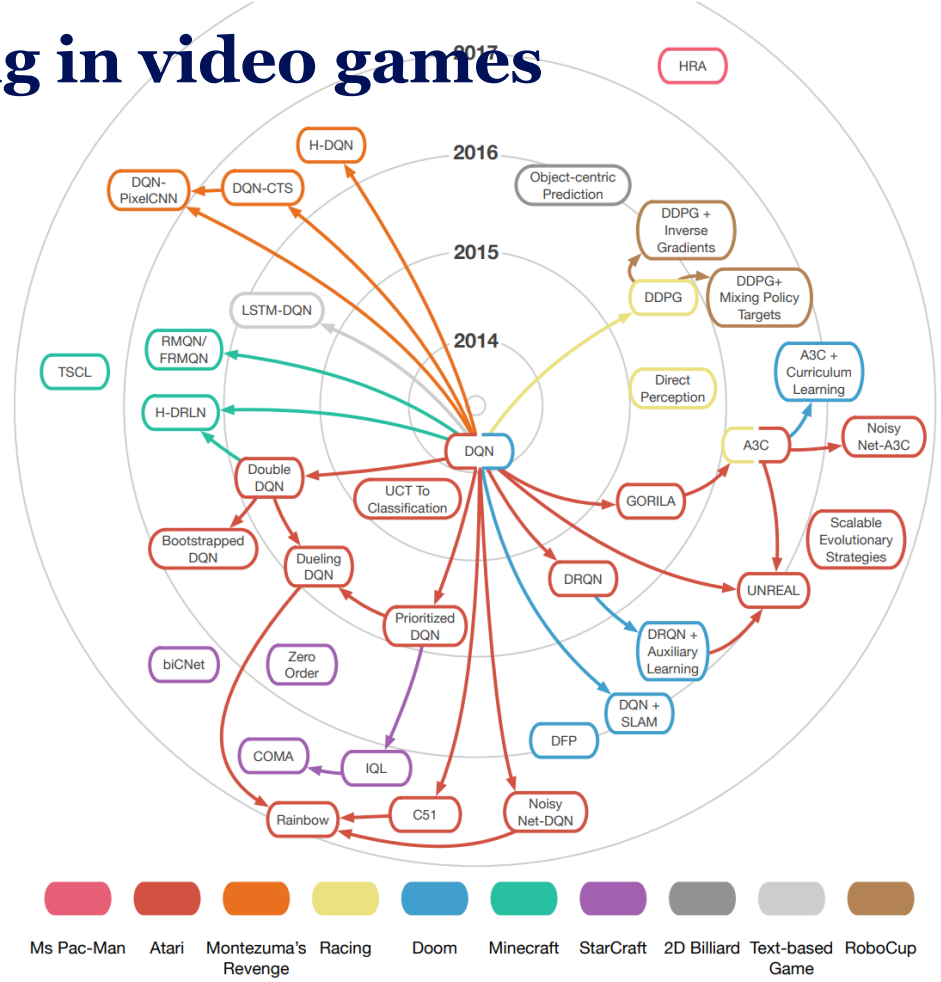


Fig. 3. Influence diagram of the deep learning techniques discussed in this paper. Each node is an algorithm while the color represents the game benchmark. The distance from the center represents the date that the original paper was published on arXiv. The arrows represent how techniques are related. Each node points to all other nodes that used or modified that technique. Arrows pointing to a particular algorithm show which algorithms influenced its design. Influences are not transitive: if algorithm a influenced b and b influenced c, a did not necessarily influence c.

generative adversarial networks (GAN)

- a straightforward "multi-agentification" of ANN
- instead of one we have two networks that work against each other
- in principle also doable with other generative systems/recognizers, usually done with DL
- given a starting sample, it can produce more examples of "the same"

- basic idea:
  - one network generates samples that are as near as possible to a given seed set of samples
  - the other network tries to improve its ability to recognize the generated "fake" samples

picture from Peggy und Marco Lachmann-Anke on Pixabay

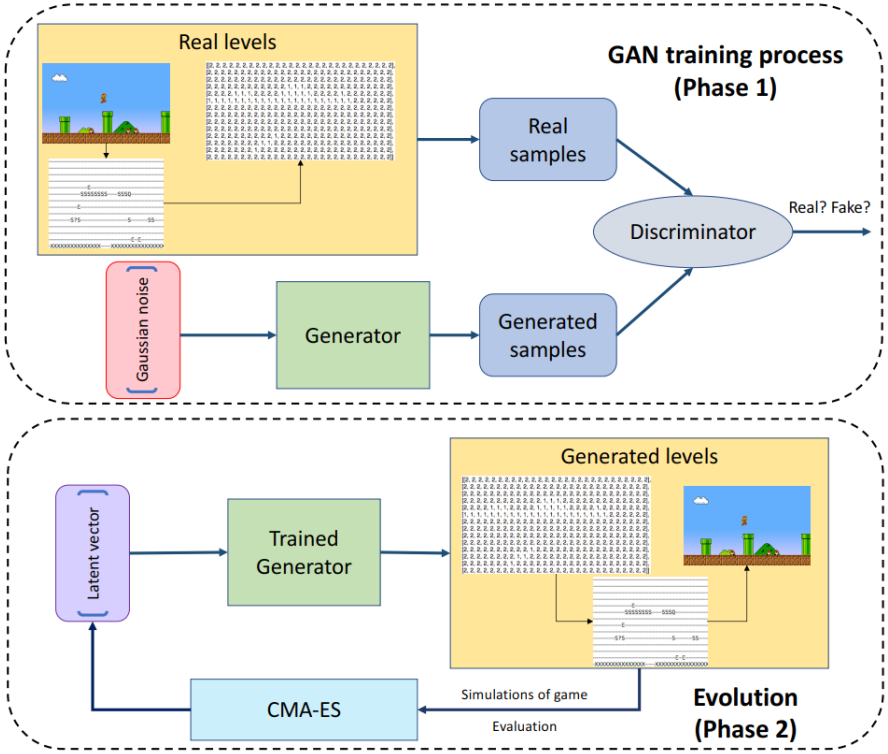
another type of content: using DL for making faces

- based on styleGan, we can make realistically looking faces
- this is a quite recent DL method with 2 neural networks playing "against" each other
- can you detect which is real and which is fake?
- this is a "face based" Turing test :-)

http://www.whichfaceisreal.com/index.php

marioGAN

- if we can make "fake pictures", we can also generate levels
- we train the generator on images from a single mario level
- the generator learns a mapping from the "latent space" to a level design
- we can use this to directly optimize for design criteria in the latent space



https://www.youtube.com/watch?v=NObqDuPuk7Q

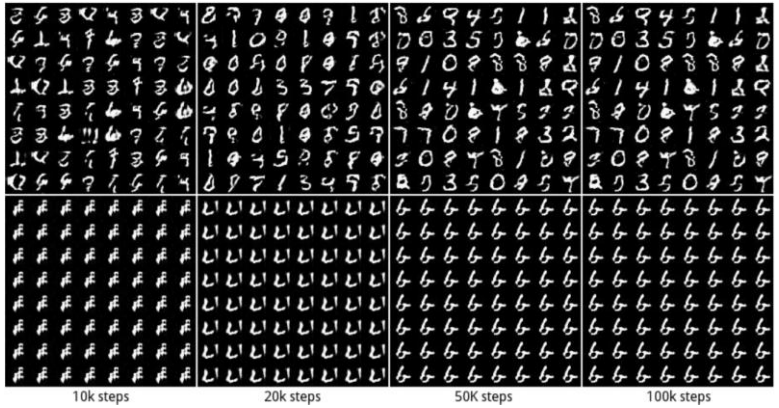
Volz, Schrum, Liu, Lucas, Smith, Risi: Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network, GECCO 2018

GAN control problems

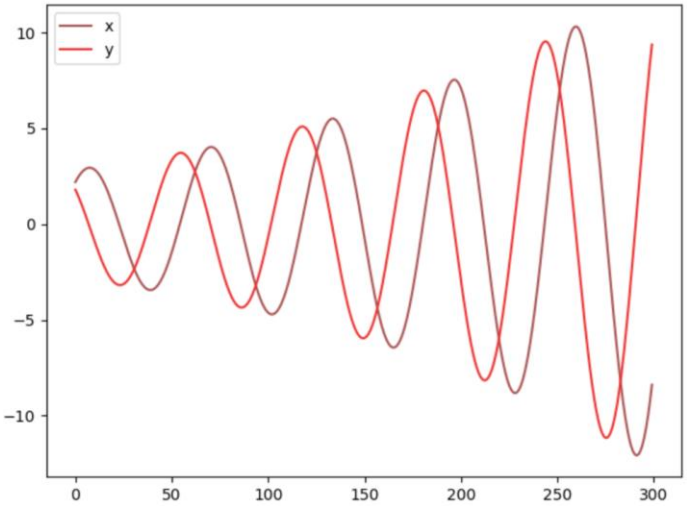
general problem - training resembles competitive co-evolution which can diverge:

- non-convergence: the model parameters oscillate, destabilize and never converge,
- mode collapse: the generator collapses which produces limited varieties of samples,
- diminished gradient: the discriminator gets so successful that the generator gradient vanishes and learns nothing,
- unbalance between the generator and discriminator causing overfitting
- strong hyperparameter sensitivity

https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

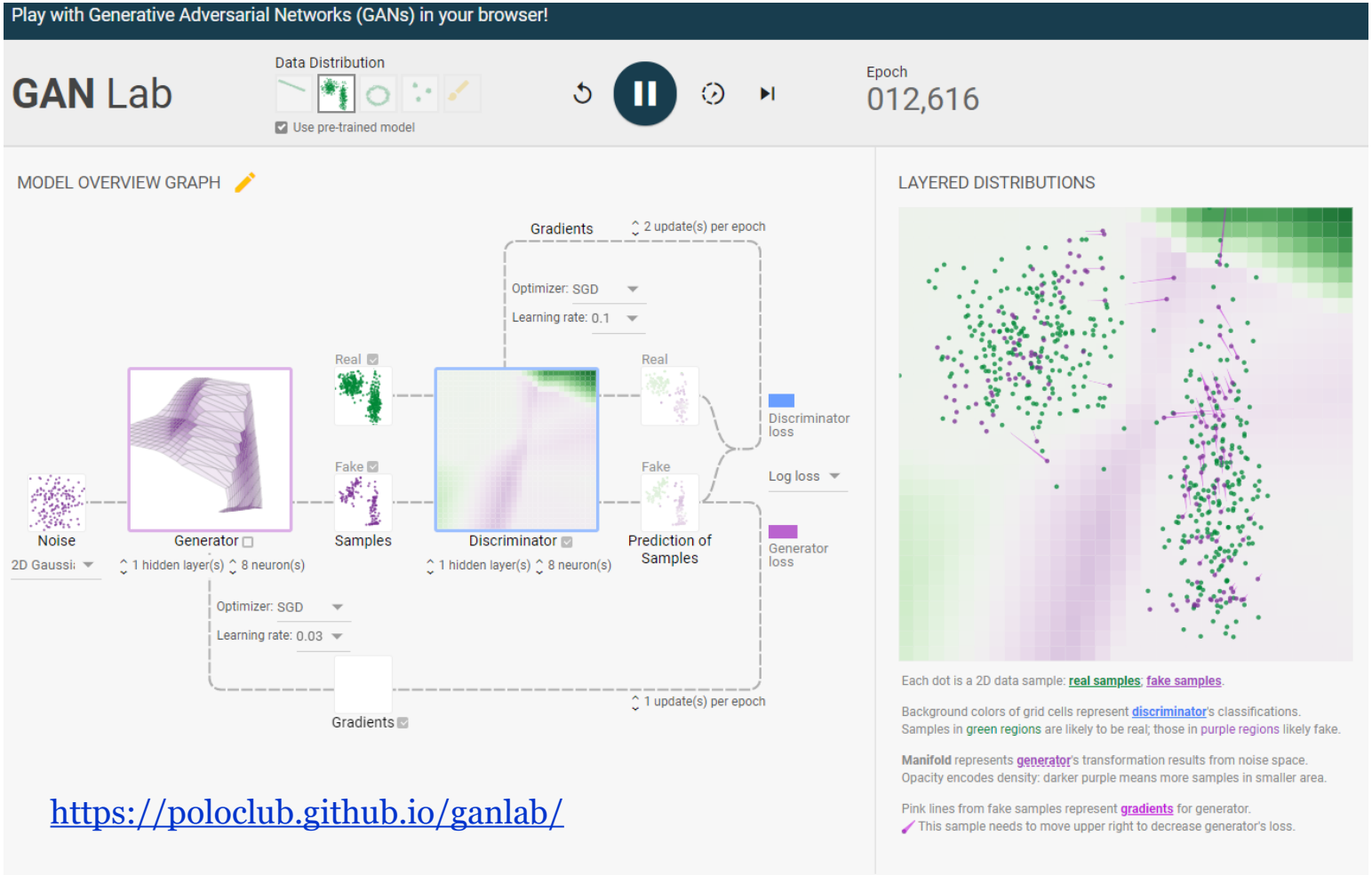


mode collapse (from the medium article)



learning oszillation (from the medium article)

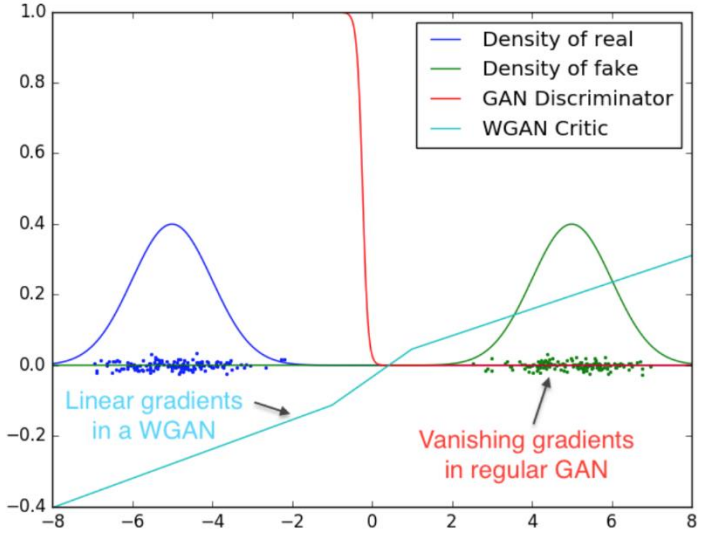
generative adversarial networks (GAN)



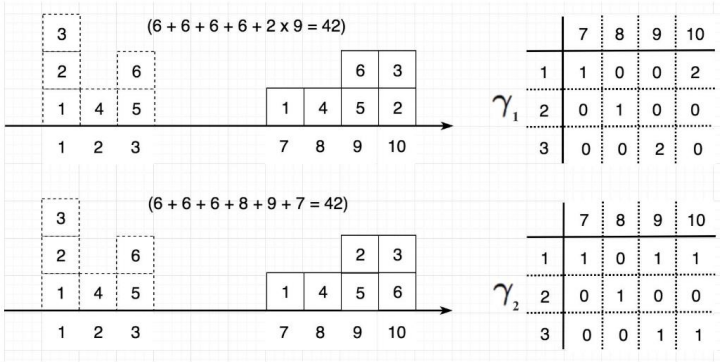
https://poloclub.github.io/ganlab/

Wasserstein GANs

- some of these problems are cured with an alternative distance metric
- the distance between original and fake distribution is much more meaningful
- the basic idea here is to use the earth movers distance for measuring it
- better gradient means that the generator can learn even if it does not yet produce good results
- the correlation between loss function and image quality improves



improved gradient (from Wasserstein article)



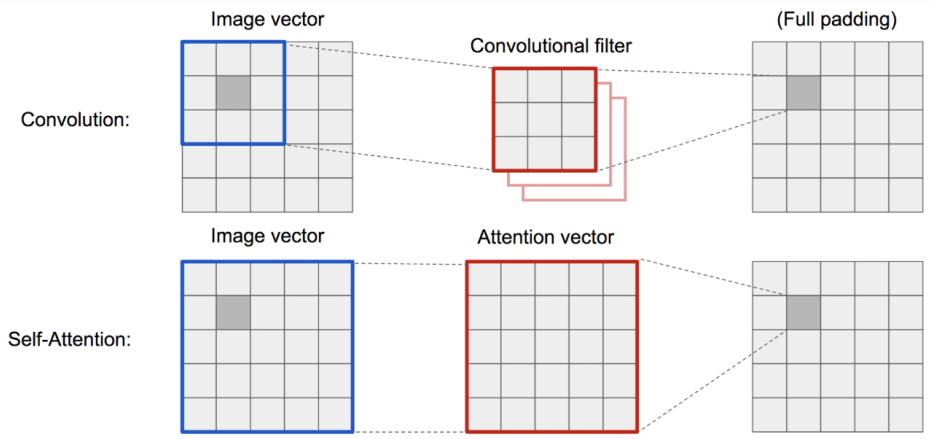
Earth movers distance (from medium article)

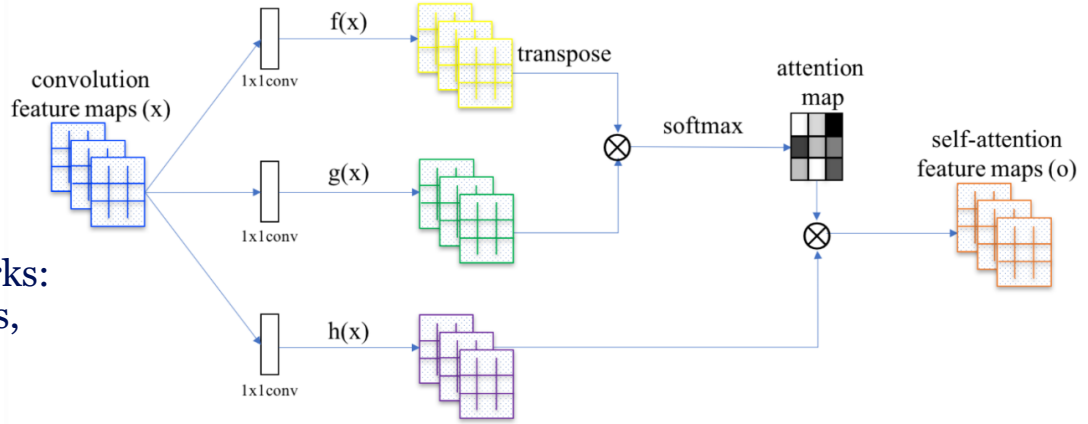
Wasserstein GAN: Martin Arjovsky, Soumith Chintala, Leon Bottou. ArXiv 2017

https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

self-attention GAN

- self-attention is a mechanism to allow interaction between distant pixels
- convolution filters only allow interaction between nearby pixels
- weights of attention values are scaled up during the learning process
- networks shall learn locally at first, then switch to more global
- the idea is that the network can deal better with details





Self-Attention Generative Adversarial Networks: Han Zhang, Ian Goodfellow, Dimitris Metaxas, Augustus Odena, arXiv 2019

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

conditional GANs for game level generation

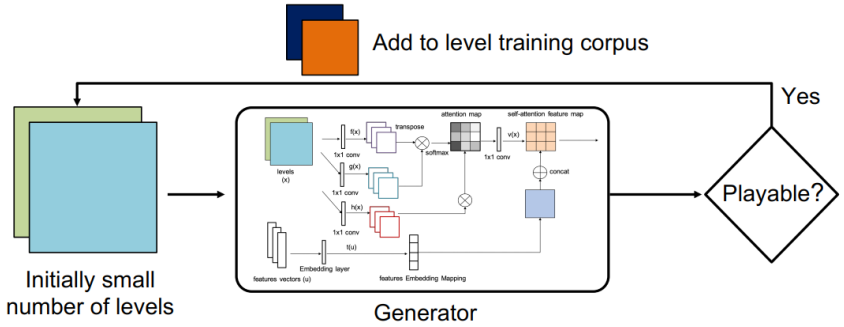
2 problems in GAN based level generation:

- sparse data to start from (small number of available levels)
- very little control over the generation process

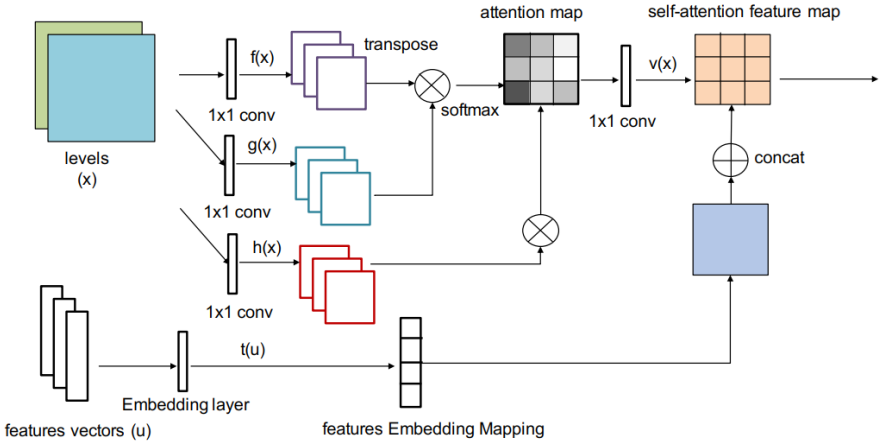
solutions:

- we generate examples via the generator and add them to "real" examples if playable
- we add a feature embedding (features are measurements on levels) to the level based attention map

Bootstrapping Conditional GANs for Video Game Level Generation: Ruben Rodriguez Torrado, Ahmed Khalifa, Michael Cerny Green, Niels Justesen, Sebastian Risi, Julian Togelius, arXiv 2019



bootstrapping (from paper)

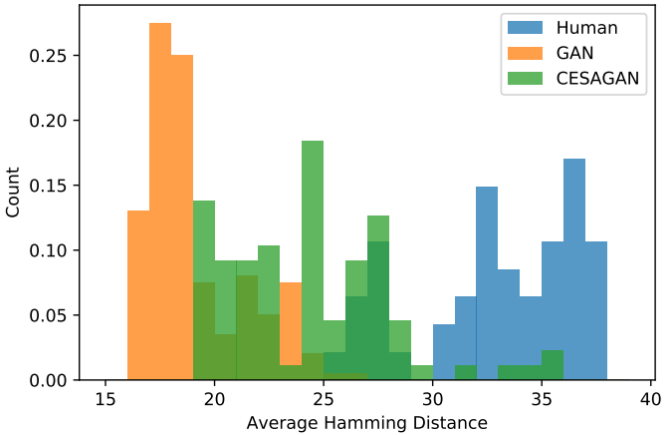


conditional embedding self-attention GAN (from paper)

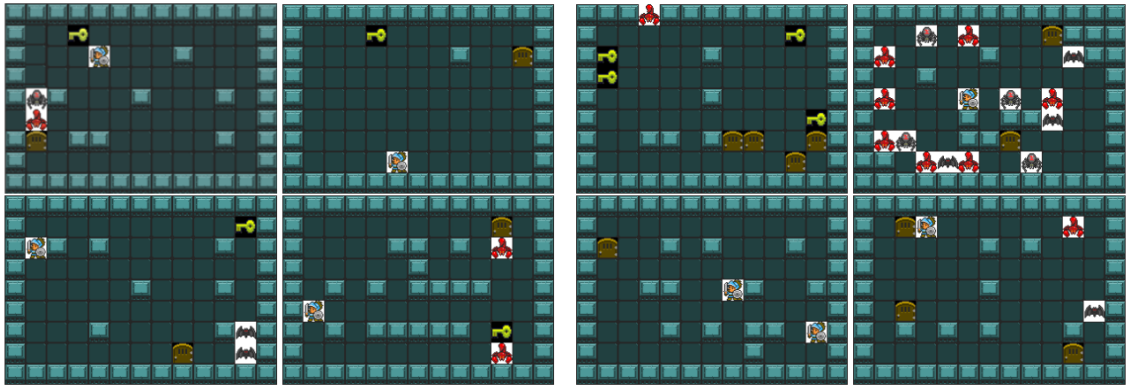
CESAGAN on GVGAI

- test case: Zelda levels
- additional input features can be number of enemies, avatars…

human designed levels (from paper)



distances between different distributions (from paper)



(a) Playable Levels (b) Unplayable Levels

generated levels (from paper)

one small step for [a] man....

Deep Reinforcement Learning: ALE
(pixel input, self-play)

AlphaGo: Go
(MCTS, human samples, self-play)

Go-Explore: Montezuma
(population based learning)

AlphaGoZero: Go
(MCTS-, self-play)

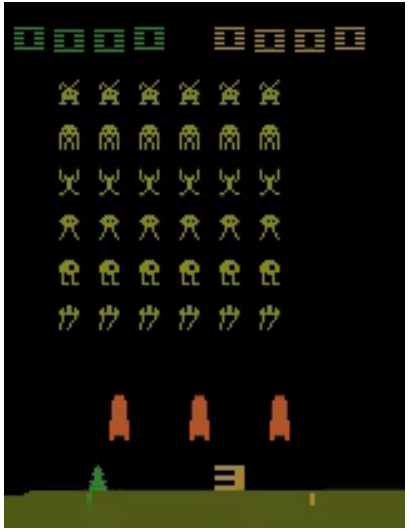
ForTheWin: Capture the Flag
(2 speed rnns, population based learning, generated scenarios, team AI)

AlphaZero: Go, Chess, Shogi
(MCTS-, self-play)

HideAndSeek
(pixel input, generated scenarios, population based learning, team AI)

AlphaStar: StarCraft
(pixel+data input, self-play, human samples, population based learning)

Obstacle Tower Challenge
(pixel input, generated scenarios, human samples)



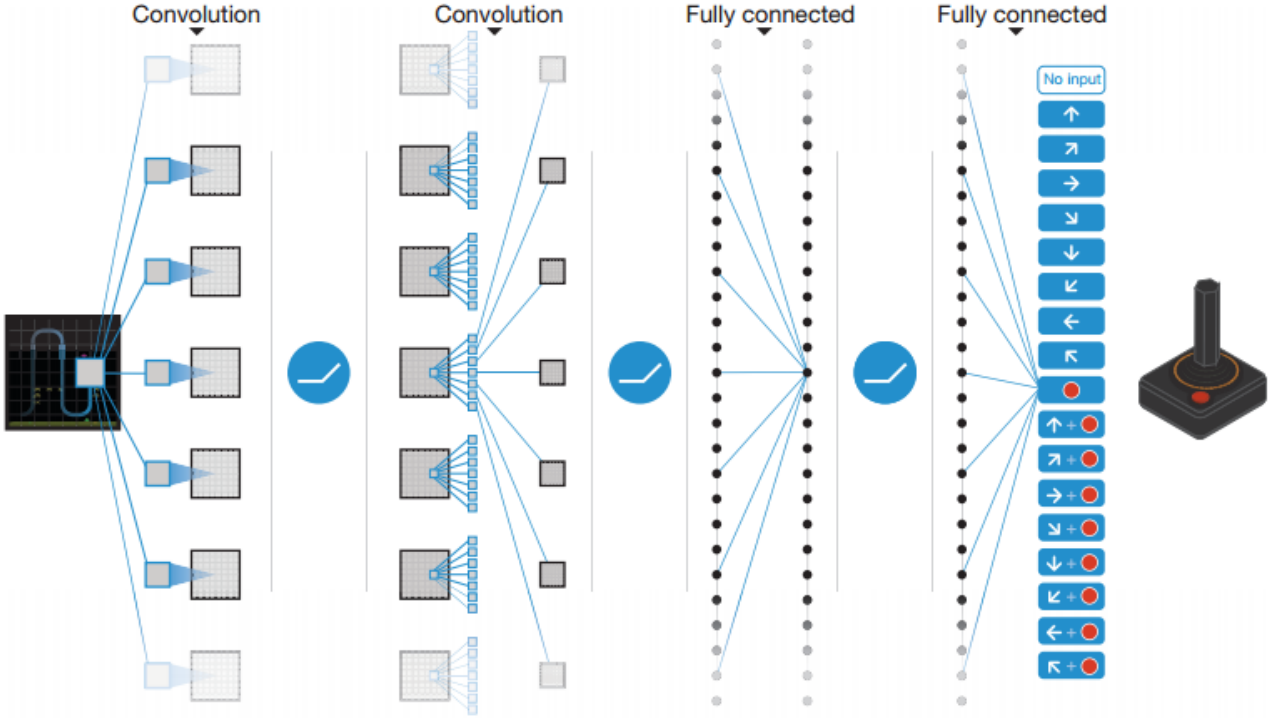
concepts

- input:
  state input / pixel input,
  human sample data
- learning tools:
  MCTS, self-play, recurrent NN,
  convolutional NN
- population based learning:
  several agents learn
  (competitive/cooperative or both)
- generated scenarios:
  high number of different scenarios,
  usually PCG generated

picture from Pexels on Pixabay

deep Q-learning Atari games

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, Demis Hassabis. "Human-level control through deep reinforcement learning", in Nature 518, 529–533, 2015.



deep convolutional NN approximates the optimal action-value function (end-to-end)

example: ViZDoom

- active (end-to-end) deep learning-based competition environment
- advantage: game is old, rendering is cheap
- up to 7000 fps is possible with modern machines
- Python interface available (and C++)
- paper won the best paper award at CIG 2016 conference

http://vizdoom.cs.put.edu.pl/

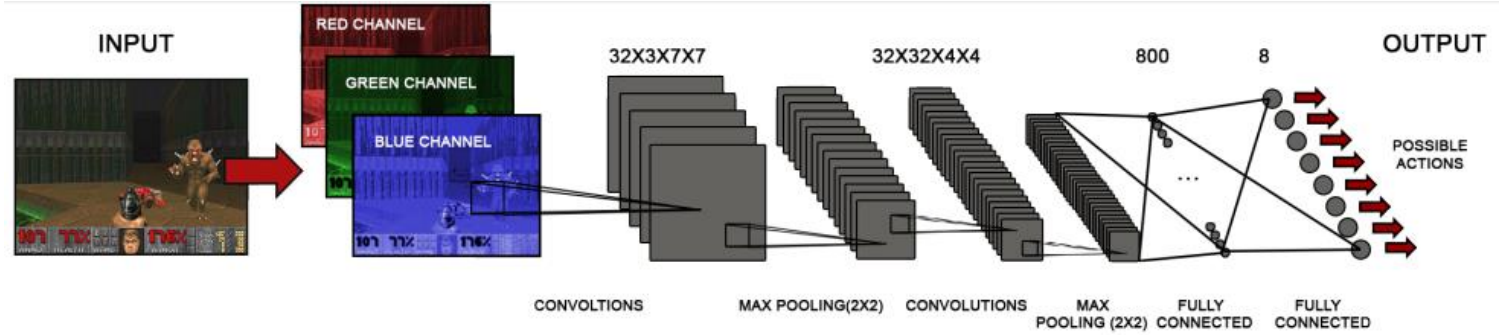
Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek & Wojciech Jaśkowski: ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning, CIG 2016

ViZDoom

- deep learning does not use hand-crafted features but raw data
- this makes it computationally expensive (but more general)

example (ViZDoom paper, experiment 1):

- agents learn for 600 000 steps
  (performing an action, observing a transition, updating the network)
- final controllers evaluated on 10 000 episodes
- run on i7-4790k 4GHz with GeForce GTX 970 (GPU for the neural net)
- looks pretty much like the Atari network, right? ;-)

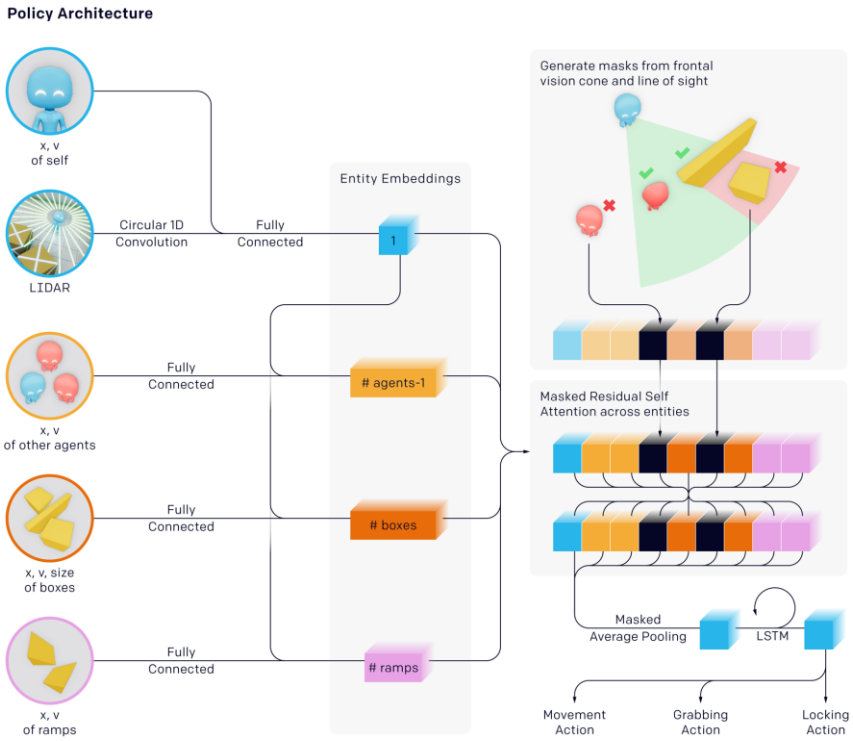


hide and seek (OpenAI)

- combines DRL from pixels and from data and (from time)
- with an evolutionary cycle of several interactive agents
- however, no human data, only self-play
- behaviour is complexified over time
- counter-strategies evolve
- arms-raise effect known from competitive co-evolution
- heavily relies on generated worlds (procedural content generation, PCG)!

blog: https://openai.com/blog/emergent-tool-use/

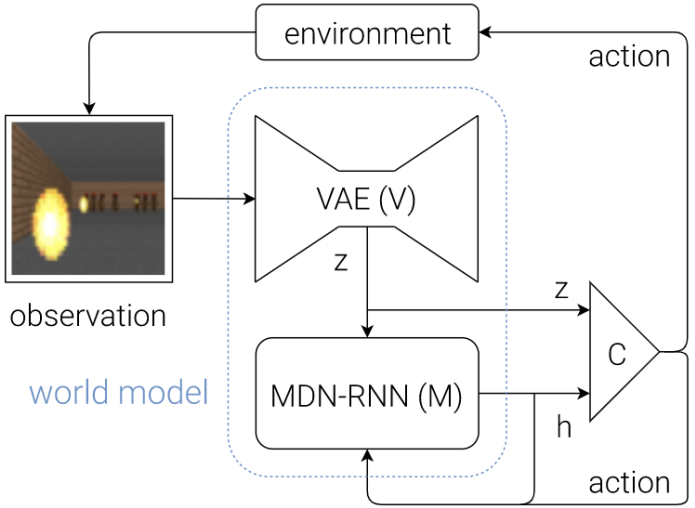
https://www.youtube.com/watch?v=kopoLzvh5jY



Emergent Tool Use From Multi-Agent Autocurricula
Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, Igor Mordatch (arXiv 2019)

world models

- humans learn to represent the outer world in their minds
- we can use this for "intuitive predictions"
- can agents learn only in their own world model?
- what if we represent the world via variational auto-encoder (VAE)
- and then only learn actions for control problems in this environment
- later on apply them in the real environment



Flow diagram of our Agent model. The raw observation is first processed by V at each time step t to produce z_t. The input into C is this latent vector z_t concatenated with M's hidden state h_t at each time step. C will then output an action vector a_t for motor control. M will then take the current z_t and action a_t as an input to update its own hidden state to produce h_{t+1} to be used at time $t+1$.

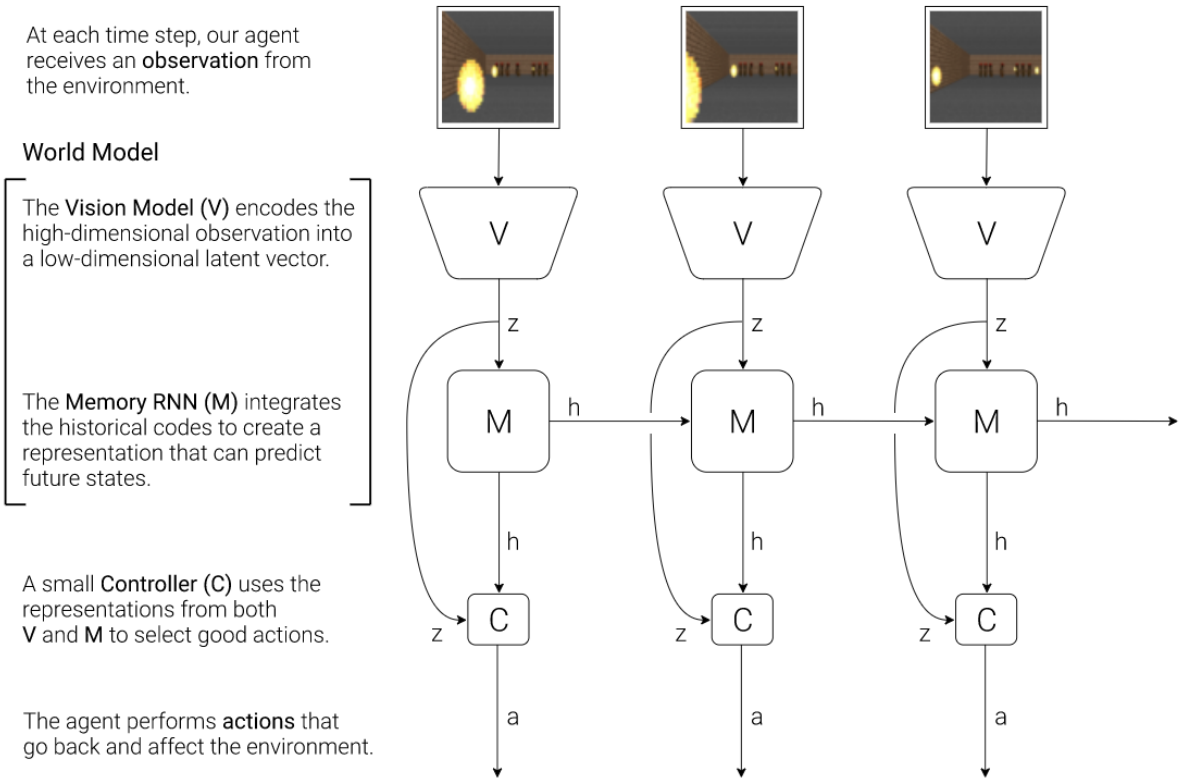
https://worldmodels.github.io/

Recurrent World Models Facilitate Policy Evolution:
David Ha, Jürgen Schmidhuber, NIPS 2018.

world model architecture

- the VAE based vision model encodes the world into a latent vector
- the memory component realizes time flows and establishes a forward model
- the controller selects the action that is then applied in the real world
- recent works show that it is also possible to directly train in the latent vector space

https://worldmodels.github.io/



take home

- deep learning is possible now due to hardware resources and data
- GAN are difficult to control, Wasserstein GANs are much more robust
- deep reinforcement learning directly learns actions from pixel data
- it is more and more connected to other techniques (state information, human samples, generated scenarios, population based learning)

picture from OpenClipart-Vectors on Pixabay