AI For Ardupilot Developers

Neural Networks

OlivierJB

2020 Ardupilot Developer Conference

AI Caveat emptor

- **Definition of intelligence? Many! None perfect**

- **One simple definition of intelligence, but has issues :**
  The ability to acquire and apply knowledge and skills." **(Dictionary)**

- **Ability to acquire … : "Machine learning"**

- **Lack of precise definition unfortunately contributes to misunderstandings and hype.**
  - **What's AI, what's not?**
  - **Sometimes overused and hyped buzzword**

- **Lack of precise definition   =>**
  - **Difficult to design benchmarks that cannot be gamed**
  - **Performance on specific benchmark does not necessarily indicate general performance**
  - **Can lead to inflated claims or interpretations**

- **R. J. Sternberg [1]: "*Viewed narrowly, there seem to be almost as many definitions of intelligence as there were experts asked to define it*". Legg & Hutter, "A Collection of Definitions of Intelligence", lists over 70 definitions[2]**

Roughly, two kinds of AI

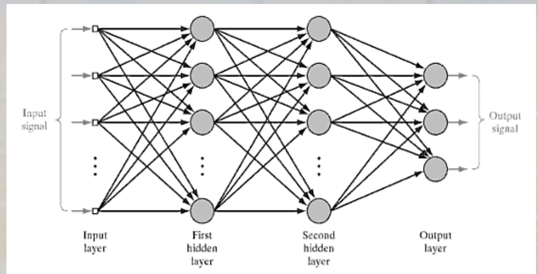
- Symbolic (Good Old Fashion AI)
  - Objects are represented with symbolic data structures, behavior is explicitly programmed
  - ```
    Objects:  Attributes and methods
    Behavior: if x in animals then …
    ```
    class animals, subclass dog, cat, …
    control flow if, while, for, …

- Subsymbolic, neuroscience inspired
  - Objects are represented over a large number of simple interconnected processing units, symbols and behavior emerge

```
dogs =      [0.2 0 0 0.6   0 0 0 0  3.2 0 0 0 0]
cats =      [0.3 0 0 1.2 0 0 0 0 2.1 0  0 0 4.3]
```

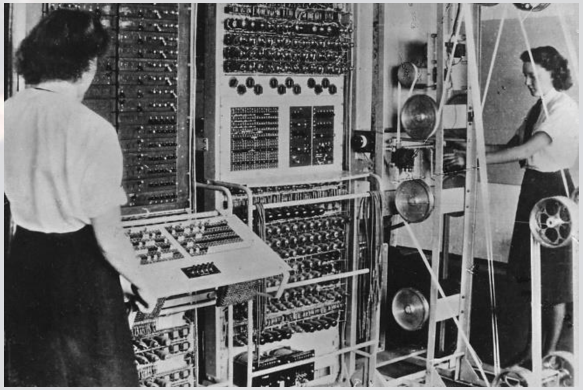
Symbols and learned behavior emerges

```
animals == [0.7 0 0 1.8 0 0 0 0  0 .3 0 0 .14 ]
```


```

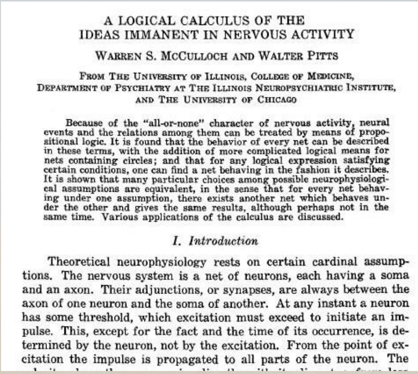
AI History

- Idea of intelligent machines almost or as old as computers
- Two approaches as old:
  - Turing machines, 1936, Turing Test, 1950 ("Symbol Processing")
  - McCullogh and Pitts, 1943   "Subsymbolic Processing, NeuroScience inspired" :



Collosus Computer, 1943

Peter Dey



A LOGICAL CALCULUS OF THE
IDEAS IMMANENT IN NERVOUS ACTIVITY

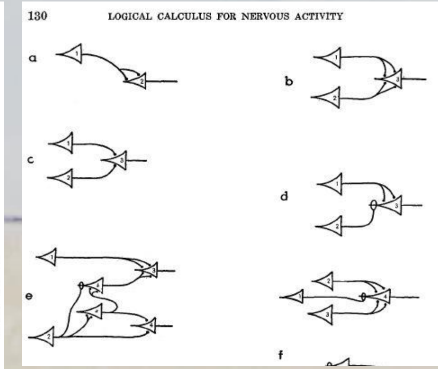
WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE, DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE, AND THE UNIVERSITY OF CHICAGO

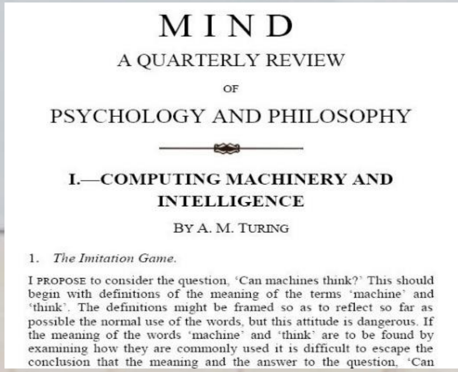
Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

I. Introduction

Theoretical neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurrence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts of the neuron. The



McCullogh and Pitts 1943



MIND

A QUARTERLY REVIEW

OF

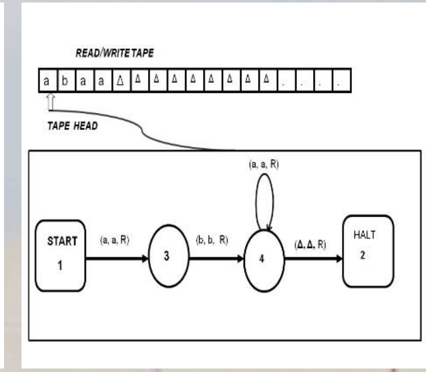
PSYCHOLOGY AND PHILOSOPHY

I.—COMPUTING MACHINERY AND INTELLIGENCE

BY A. M. TURING

1. *The Imitation Game.*

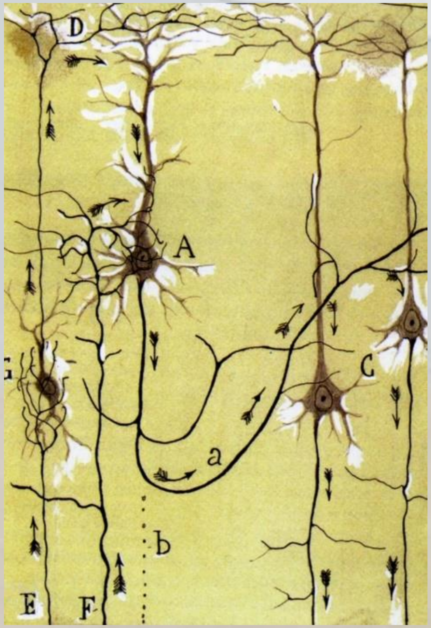
I PROPOSE to consider the question, 'Can machines think?' This should begin with definitions of the meaning of the terms 'machine' and 'think'. The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous. If the meaning of the words 'machine' and 'think' are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, 'Can



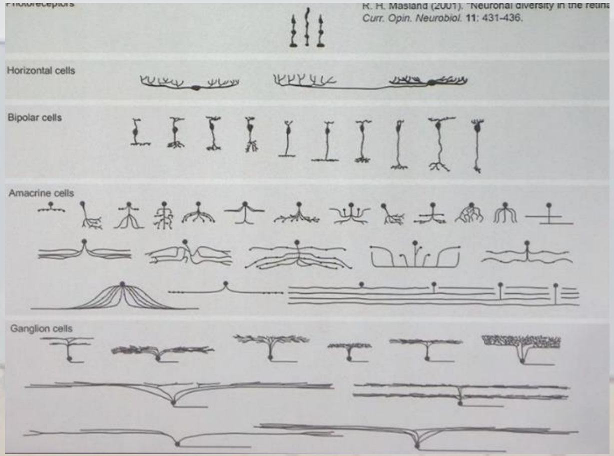
Turing, 1936

SubSymbolic: Brain Inspired

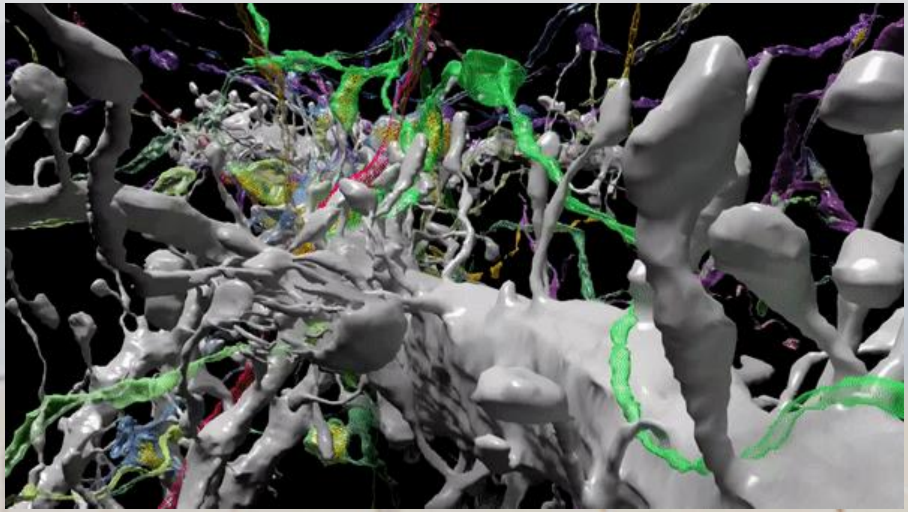
- Brain: ~ 86 * billion neurons, 100 Trillion synapses
- 100,000 miles nerve fibers. And … an absolute mess!
- Very poorly understood



Golgi, Cajal 1906 Nobel Prize



Hundreds of different nerve cells

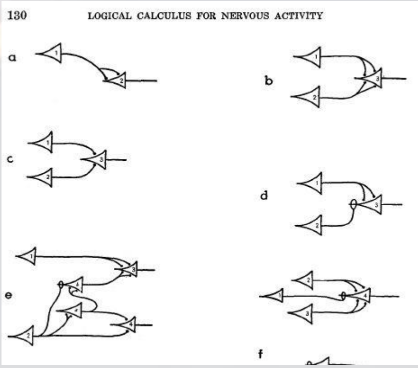


Connectomics: 1 mm3 from actual microscopy of
30nm thick slices of actual rat brain. Lichtmann

Early Models

- **McCullogh and Pitts Model, 1943**

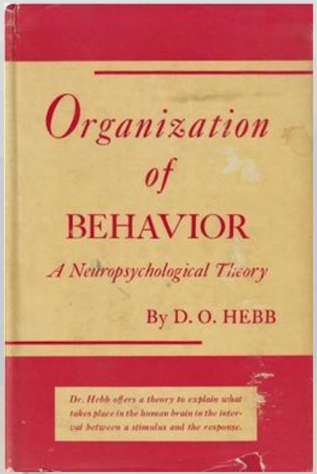
Fixed connections, neurons fire (activate) when a sufficient number of connected (via synapses) neurons fire



McCullogh and Pitts 1943

- **Hebb, 1949: Learning "Cells that fire together wire together.“**

$$\Delta w_{ij} = \eta . out_j . in_i$$



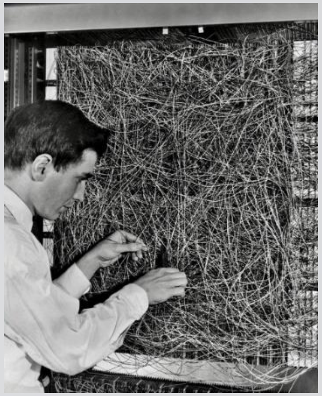
Hebb 1949

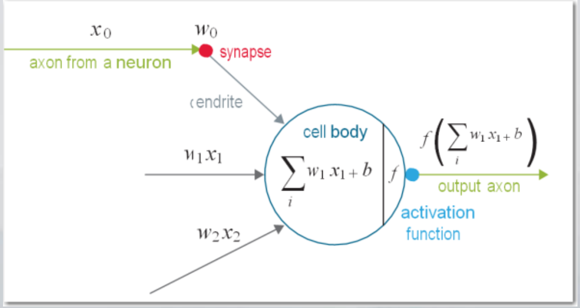
- **Modern form: Come up with an Objective function measuring output errors. (Euclidian distance of response vs desired responses for instance)**

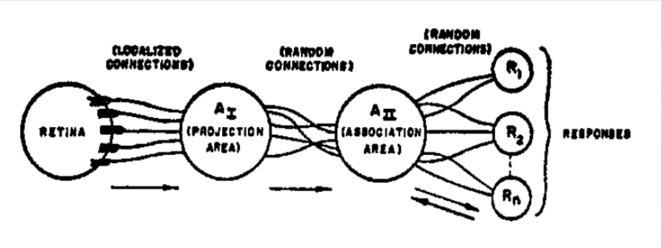
- **Change weights (learning) to minimize objective function**

Perceptrons, 1957

- Frank Rosenblatt, Cornell
- Analog computer implementation

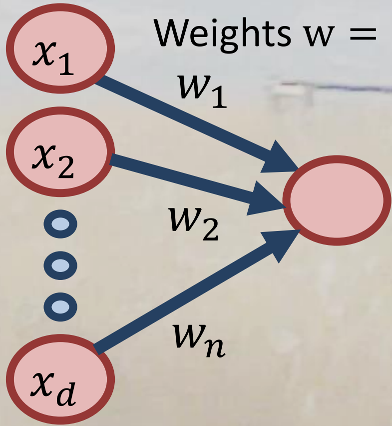






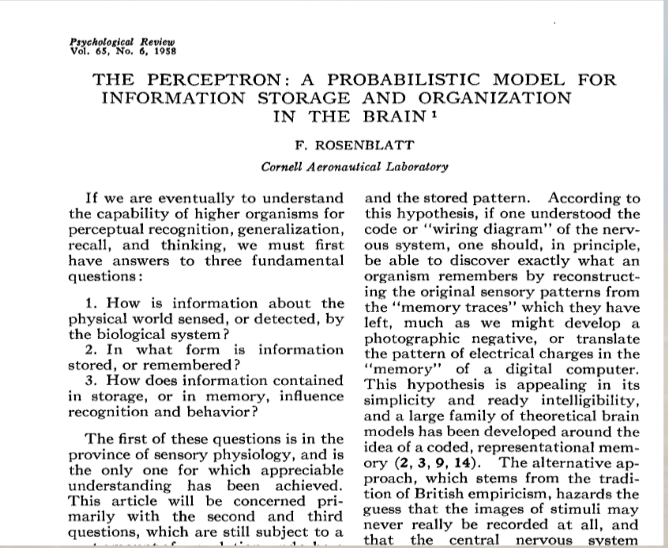
Modern formalization:

Input $x = (x_1, \ldots, x_d)$



Weights $w = (w_1, \ldots, w_d)$

$$\text{Output } f(x) = \begin{cases} 1 & (x * w^T) + b > 0 \\ 0 & \text{otherwise} \end{cases}$$



Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence recognition and behavior?

The first of these questions is in the province of sensory physiology, and is the only one for which appreciable understanding has been achieved. This article will be concerned primarily with the second and third questions, which are still subject to a

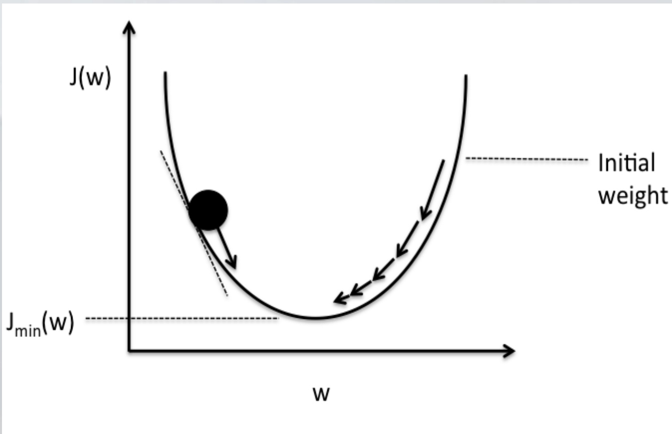
and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility, and a large family of theoretical brain models has been developed around the idea of a coded, representational memory (**2**, **3**, **9**, **14**). The alternative approach, which stems from the tradition of British empiricism, hazards the guess that the images of stimuli may never really be recorded at all, and that the central nervous system

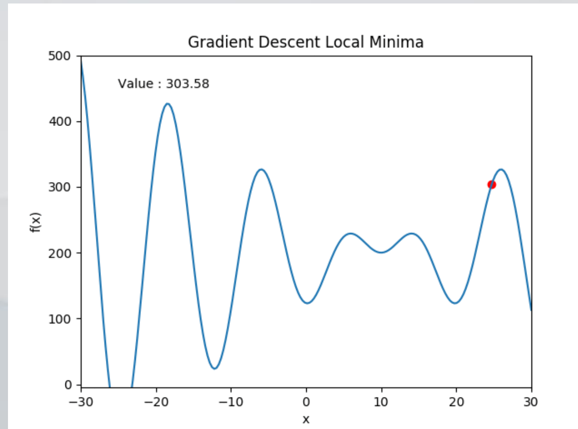
ARDUPILOT

3DVistas
Drones for Research and Industry

Learning implementation: Gradient Descent in weight Space

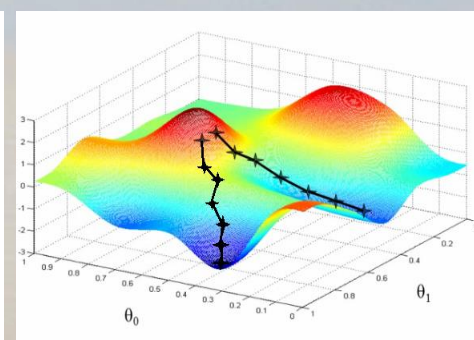
- **Gradient descent: Minimize objective function F(w) by substracting from w a fraction (learning rate) of F(w) derivative with respect to w (gradient)**





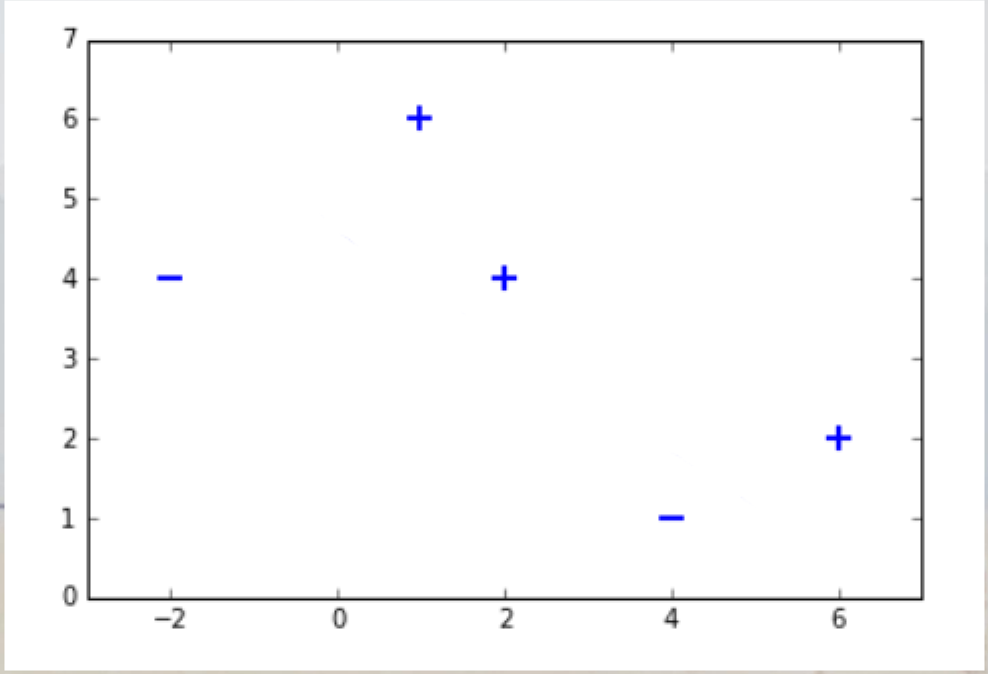
- **Generalization to multi dimensional space y = f(x), gradient of y with respect to x is the Jacobian matrix J of partial derivatives:**

$$J = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

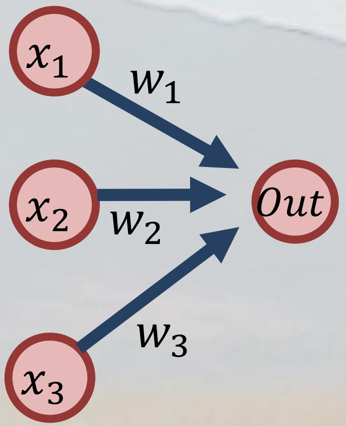


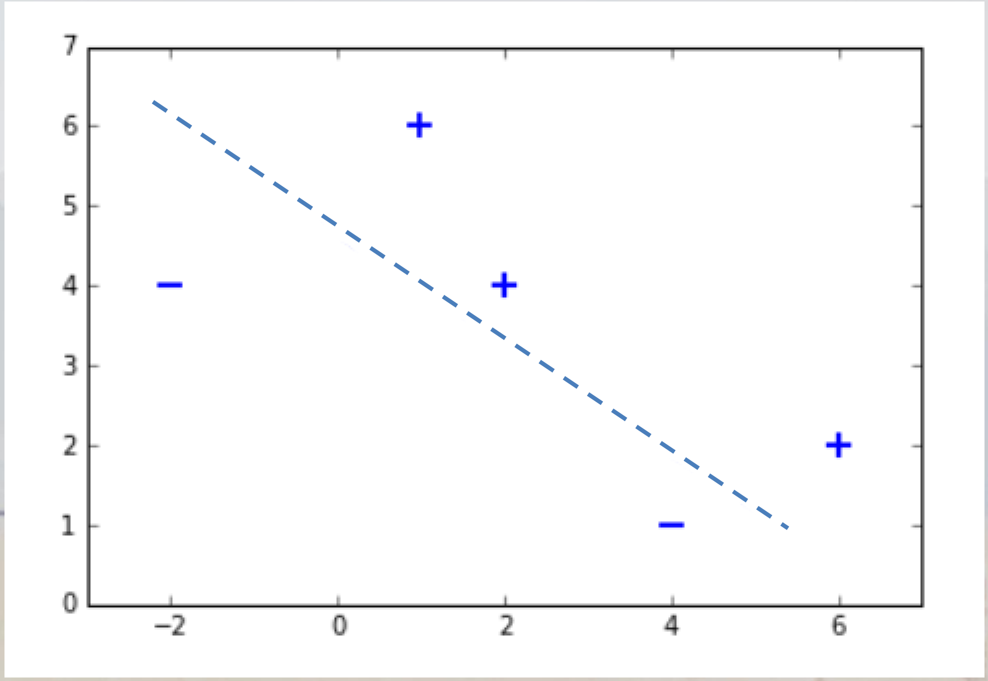
Perceptron: Python Linear Classifier Toy Example

Problem: Learn to discriminate between + and - points)

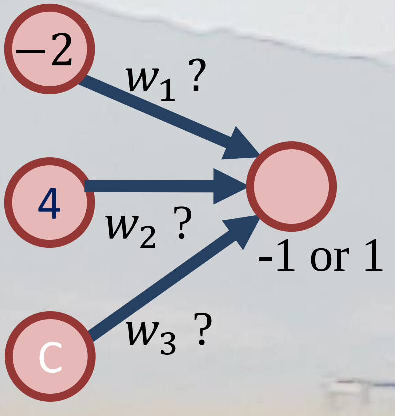


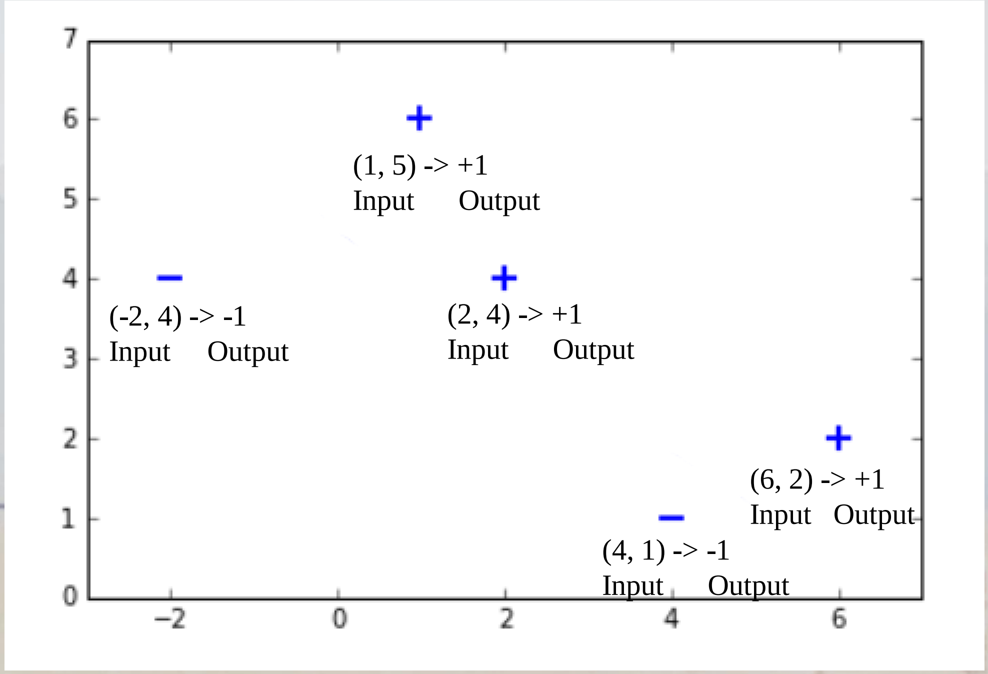
Perceptron: Python Linear Classifier Toy Example



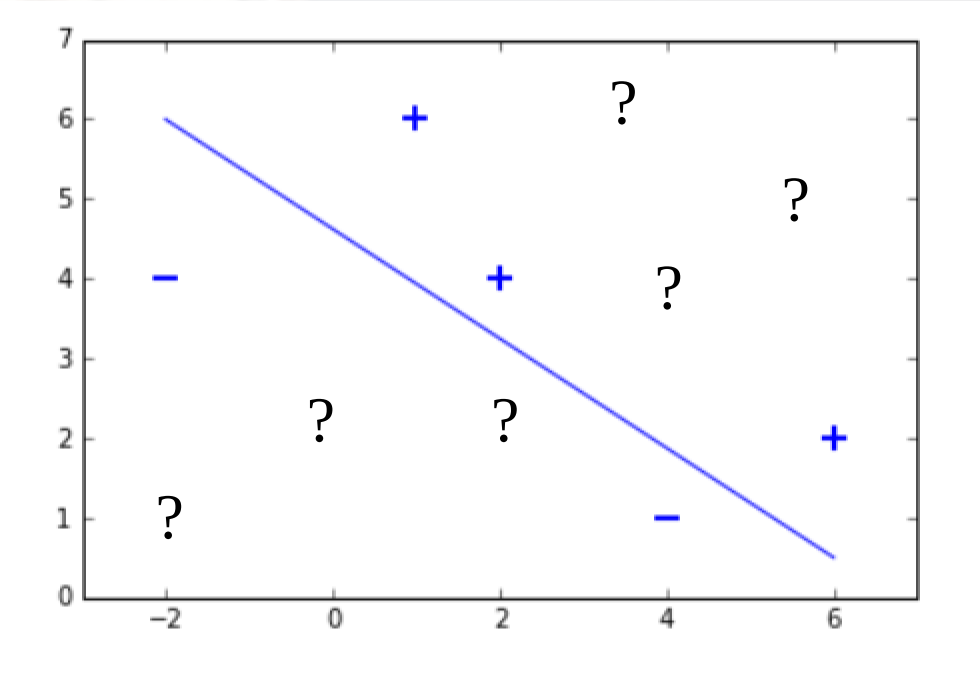


Perceptron: Python Linear Classifier Toy Example





Perceptron: Python Linear Classifier Toy Example Test:



- **Perceptron convergence Theorem (Informal description):**

If there is a solution, perceptron learning will converge to the solution.

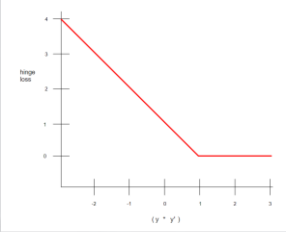
Perceptron: Python implementation with gradient descent learning

Note: We use hinge loss for objective function for historical reason
HL(Output) = max (0, (1 – DesiredOutput*Output))

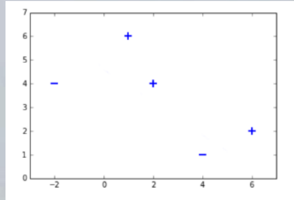
```
import numpy as np

X = np.array([ [-2,4,-1],[4,1,-1],[1,6,-1],[2,4,-1],[6,2,-1]])
#5 inputs (points in 2d + -1 bias)
y = np.array([-1,-1,1,1,1]) # 5 Targets (Plot + or - labels)

def perceptron(X, Y):
    w = np.zeros(len(X[0])) # Weights to be modified
    eta = 1
    epochs = 20
    for e in range(epochs):
        for i, x in enumerate(X):
            if (np.dot(X[i],w)*Y[i]) <= 0:
                w = w + eta*X[i]*Y[i]      #Update w with -gradient
    return w
w = perceptron(X,y)
print(w)  #Weights solution
```



Loss function



After https://github.com/MaviccPRP/perceptron

Perceptron: Python implementation with gradient descent learning

```
import numpy as np

X = np.array([ [-2,4,-1],[4,1,-1],[1,6,-1],[2,4,-1],[6,2,-1]])
#5 inputs (points in 2d + -1 bias)
y = np.array([-1,-1,1,1,1]) # 5 Targets (Plot + or - labels)

def perceptron(X, Y):
    w = np.zeros(len(X[0])) # Weights to be modified
    eta = 1
    epochs = 20
    for e in range(epochs):
        for i, x in enumerate(X):
            if (np.dot(X[i], w) * Y[i]) <= 0:
                w = w + eta*X[i]*Y[i]
    return w
w = perceptron(X,y)
print(w) #Weights solution
```

Output

Gradient (Error)

Error measured with hinge loss function:

HL(Output) = max [0, (1 – DesiredOutput*Output)]

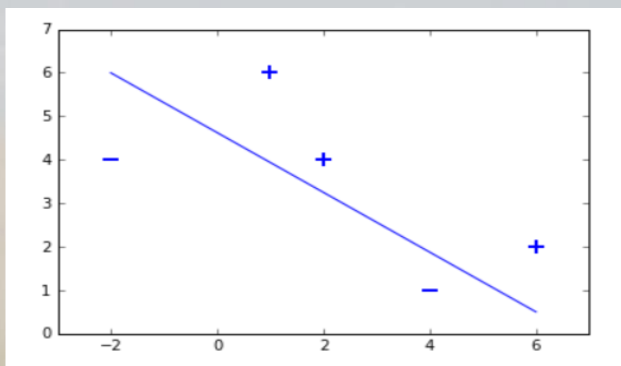
HL (wX[i]) = max [0, (1 – Y[i] * wX[i])]

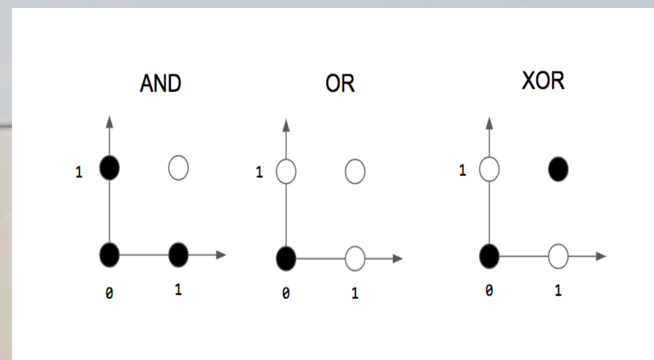
$$
\begin{aligned}
dHL(w)/dw &= d(\max[0, 1 - (Y[i] * wX[i])])/dw \\
&= 0 \qquad \text{if } 1\text{-}Y[i] * wX[i] > 0 \\
&= -\,eta * \underbrace{(-X[i]Y[i])}_{\text{Gradient}} \qquad \text{if } 1\text{-}Y[i]*wX[i] <= 0
\end{aligned}
$$

After https://github.com/MaviccPRP/perceptron

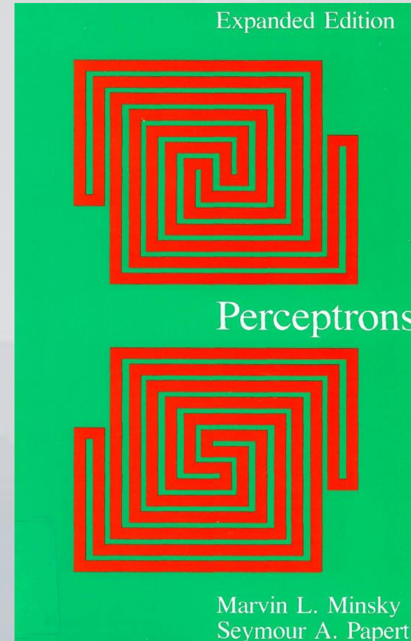
1970's Temporary end of Neural Networks!

- **What about other functions?**
- **How about Boolean operations?**
- `AND` **?** √ `OR` **?** √
- `XOR` **?** X **No can do!**
- **Need linear separability**
- **Expectations were inflated**



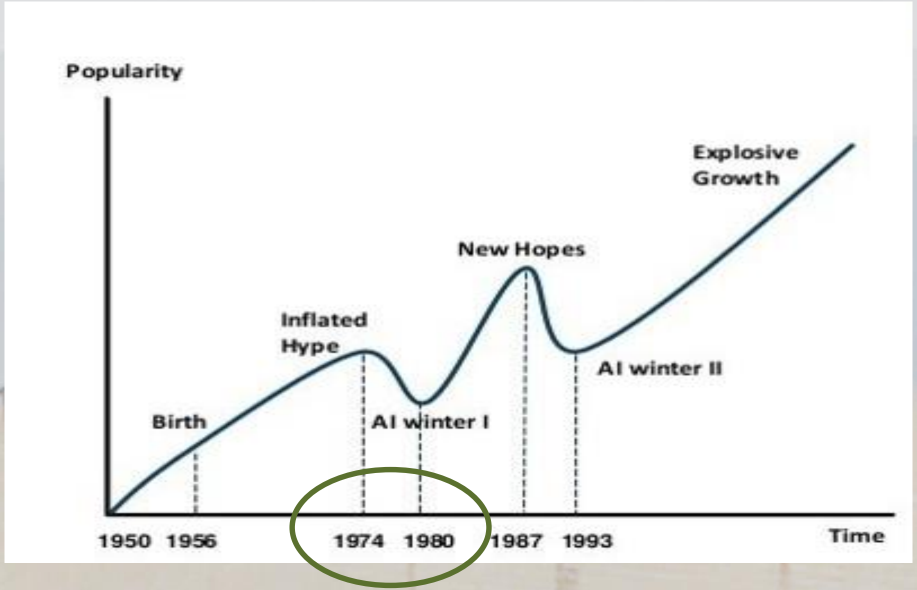


Minsky & Papert, 1969



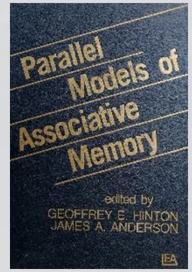
Meanwhile … 50's onward, symbolic AI

- Development of what is now called GOFAI (Good Old Fashion AI)
- Reinforcement Learning Expert systems, probabilistic models, use of Markof decision processes, models of reasoning, natural language processing, Search algorithms, etc …

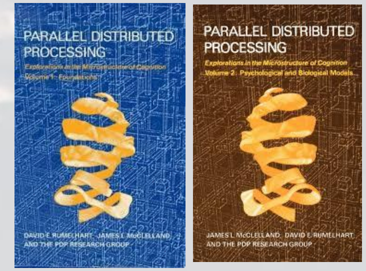


- But: Inflated expectations also!
- => First "AI Winter":

Mid 80's, rebirth: Connectionism



Hinton and Anderson, 1981



Rumelhart, McClelland and the PDP Group, 1986

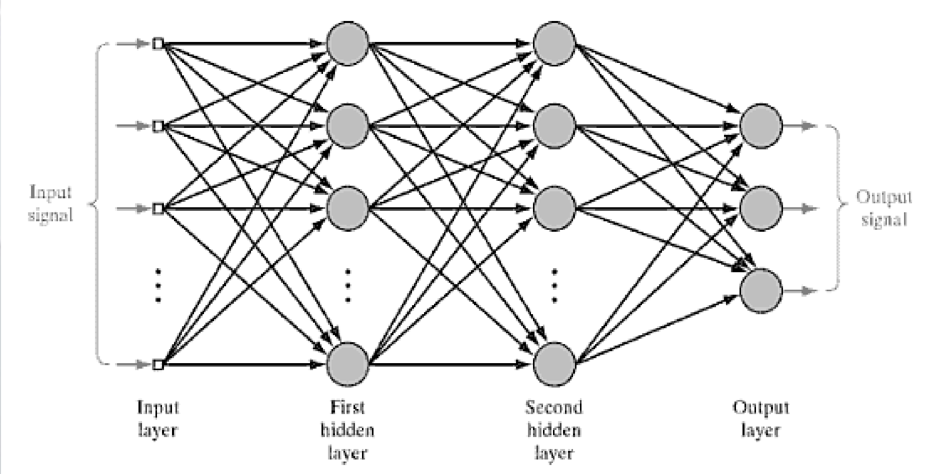
- PDP Group: Multi disciplinary group of researchers.
- Key contributions:
- Learning with Backpropagation*, for multilayer networks, with eg sigmoid squash ,
- Recurrent network architecture allowing processing in time (Backpropagation through time)
- Restricted Boltzmann machines, Boltzmann machines, …
- Many applications: Linguistic morphology, sentence processing, speech perception.
- Formulation of Subsymbolic (vs symbolic AI) theory of mind:

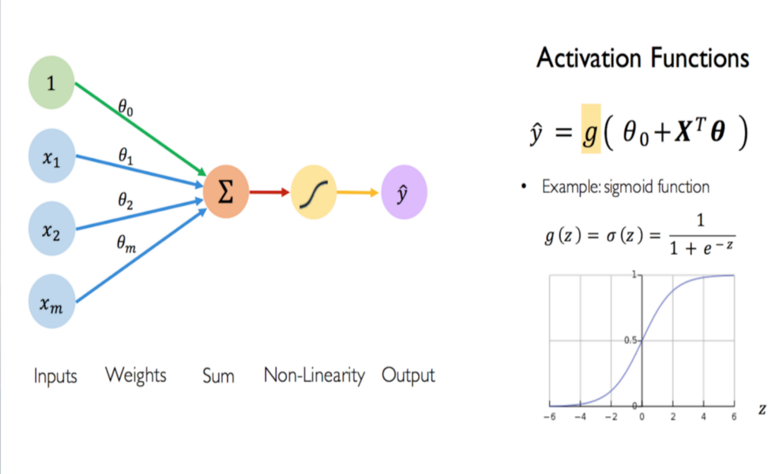
Massively distributed representations | Graceful degradation | Noise resistance

Processed in parallel | Symbols emerge from patterns of activation

*Independently discovered, prior art: Werbos, others

Multilayer Network: How can they learn? Backpropagation



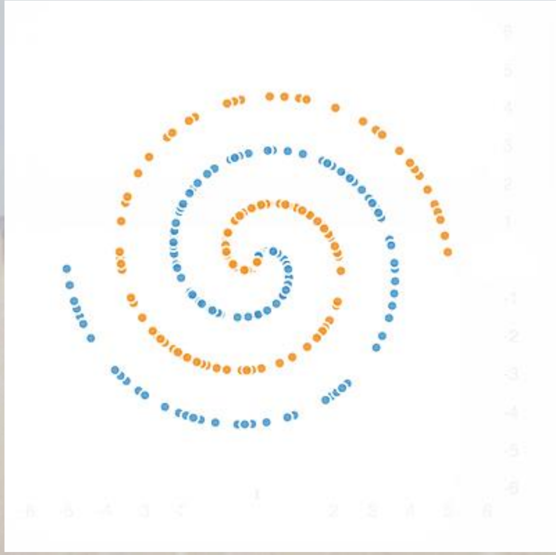


- **Use sigmoid or similar to allow for gradients**
- **General Idea: Use the chain rule for derivatives:**
- **F(x) = f (g(x)) => F'(x) = f'(g(x)) * g'(x) Use multivariable calculus: Jacobian to correct weight error.**
- **Store weight values on forward pass, then propagate error backward towards previous layers and change weights**

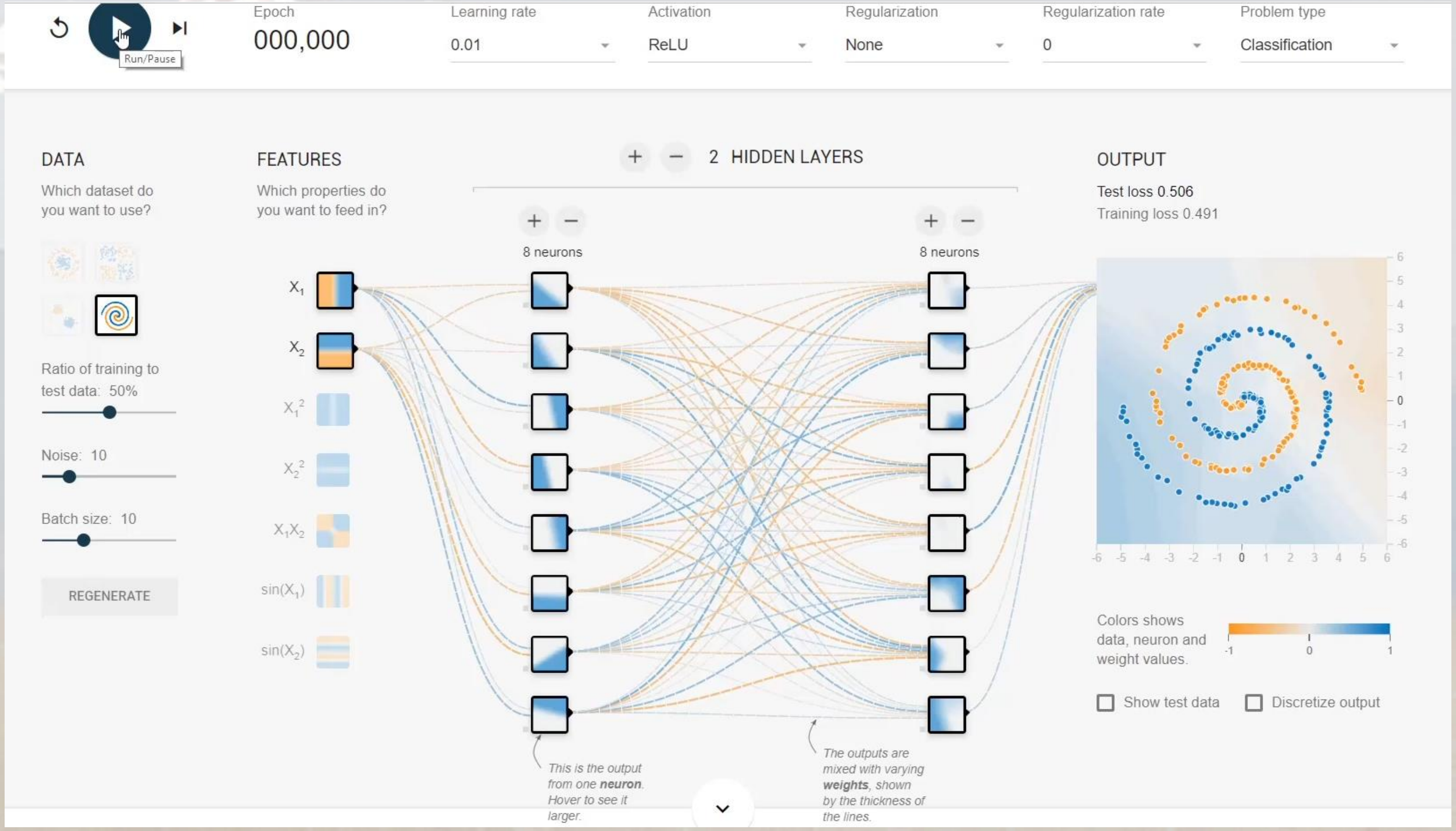
Multilayer Networks

- **Theoretical result:**
  Any continuous function can be learned with arbitrary precision with a neural network with one hidden layer (given enough hidden units)

- **[Harder classification Problem](#) that the perceptron we saw earlier cannot solve:**



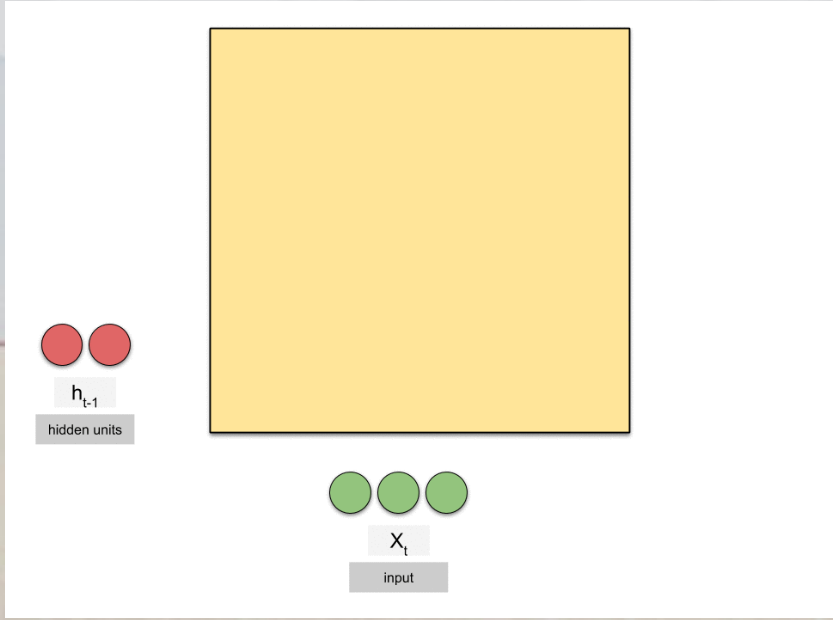
Multilayer Network: https://playground.tensorflow.org



What about time? Recurrent Neural Networks

- Temporal learning and processing of data sequences in time

General Idea: Use recurrent connections from hidden layer back to itself; Combine input layer at time t with hidden layer at time t-1



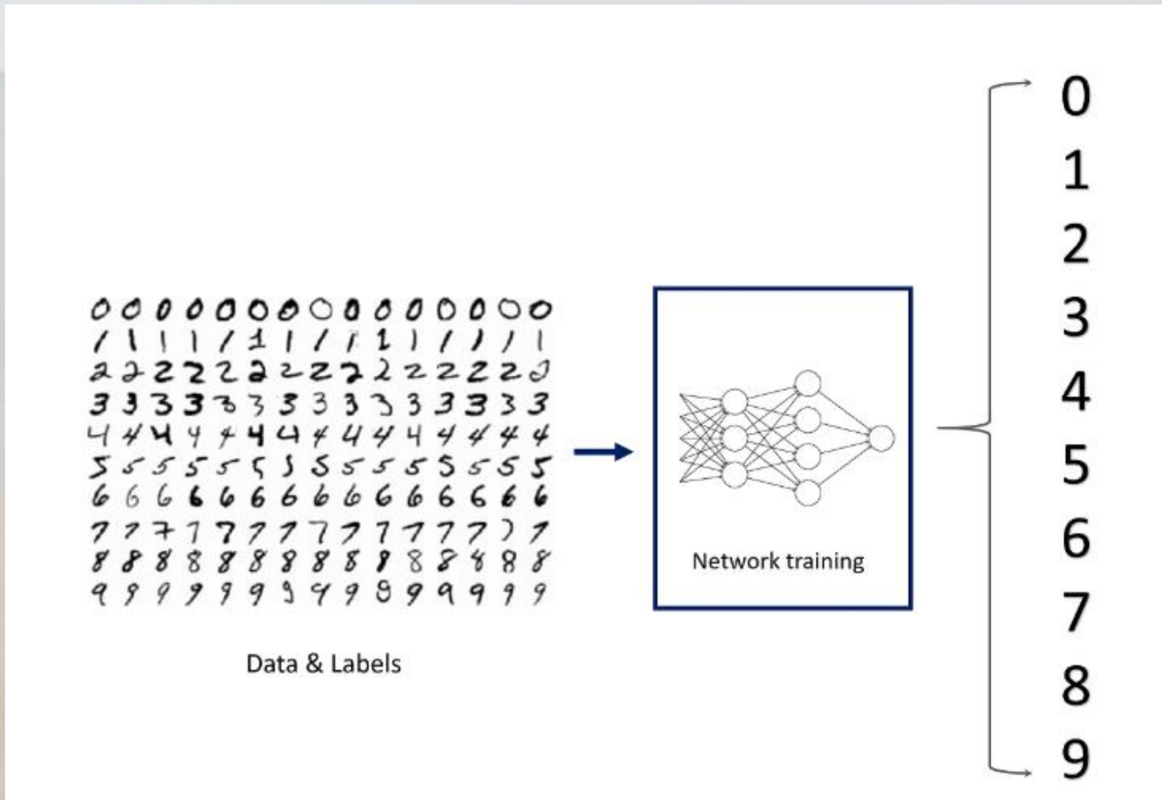
Raimi Karim

But … Mid-90s: Second AI Winter!

- NNs are data hungry, too little data
- NNs are compute hungry, too little compute
- Hype and over promises

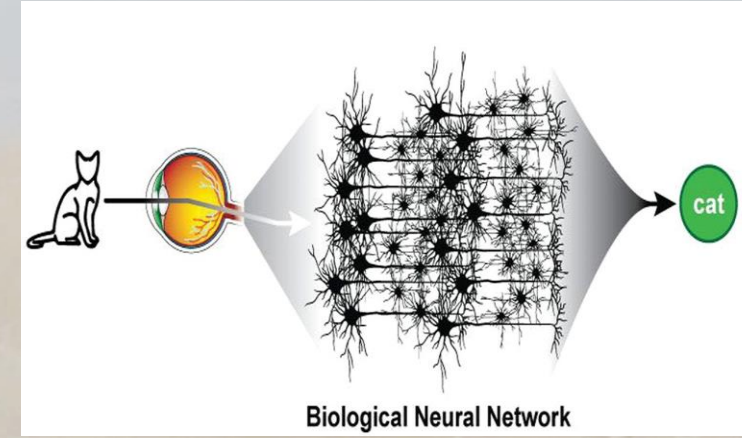
Meanwhile, however …

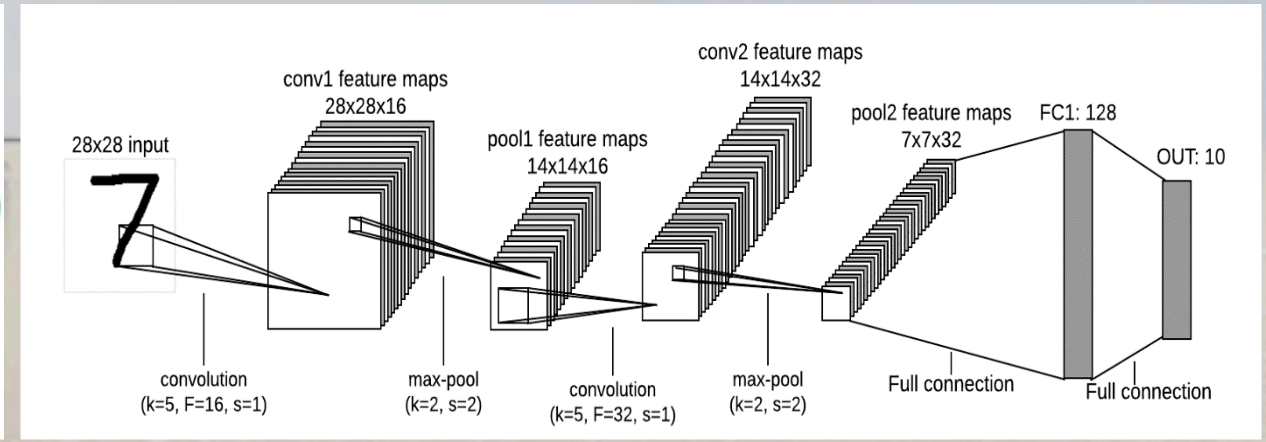
- Yann Lecun, Bell Labs
- Handwritten digit recognition



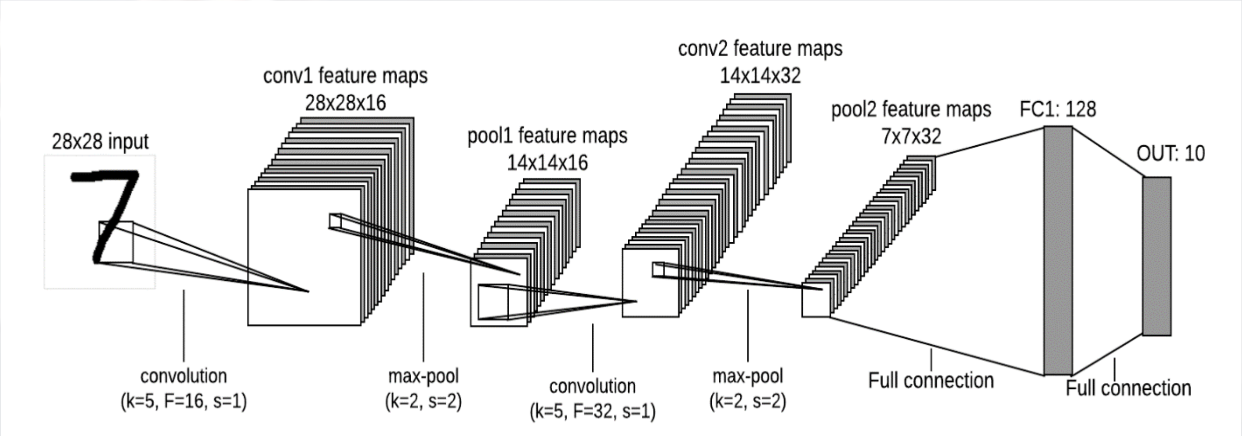
Convolution Neural Network

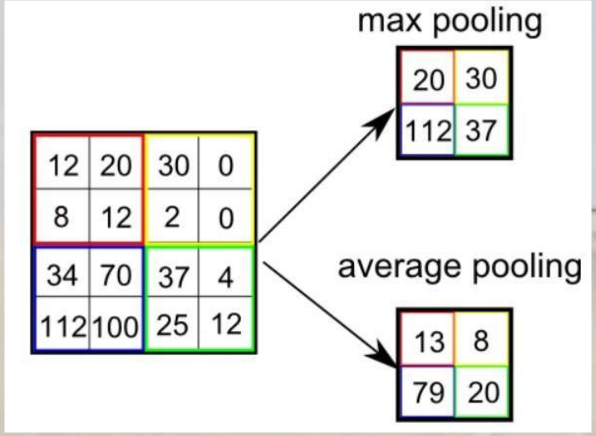
- Hierarchical processing, inspired by some cells in visual cortex: Some cells respond specifically to certain features like edges => Filters
- Idea: Intermediate layers detect features increasingly more complex
- Two new kind of layers: Convolution and pooling layers





Convolution Neural Network: Pooling



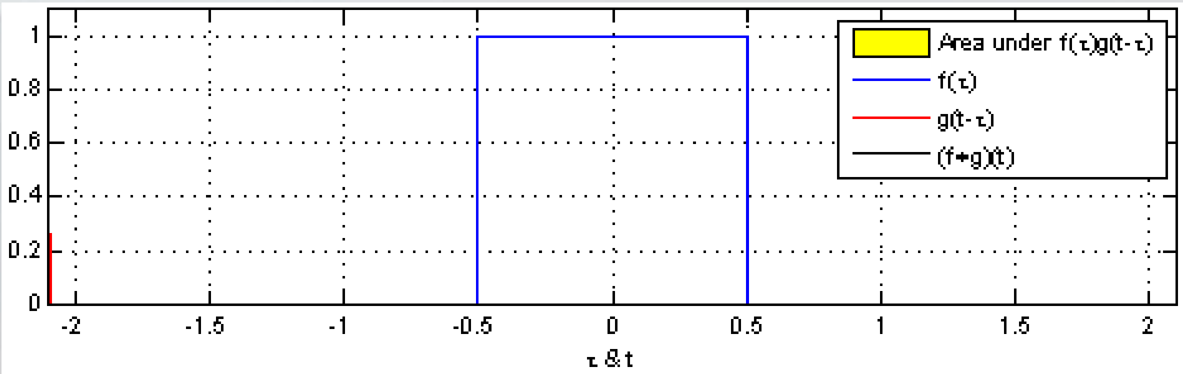


Pooling

Convolution

Central to signal processing, filters

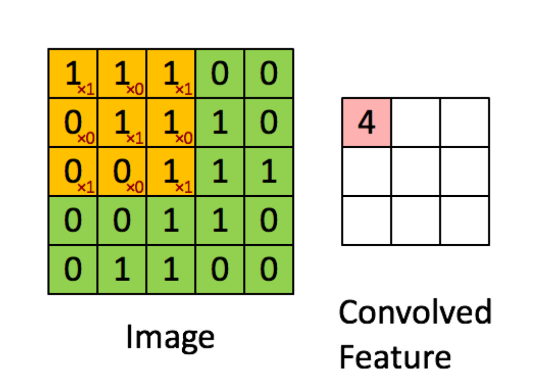
Continuous: $(f * g)(t) = \int f(\tau)g(t - \tau)\, d\tau$

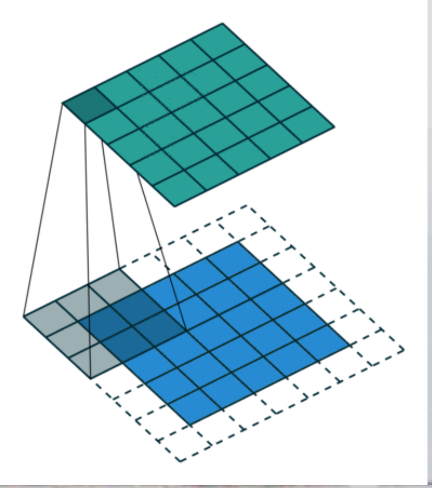


Convolutions

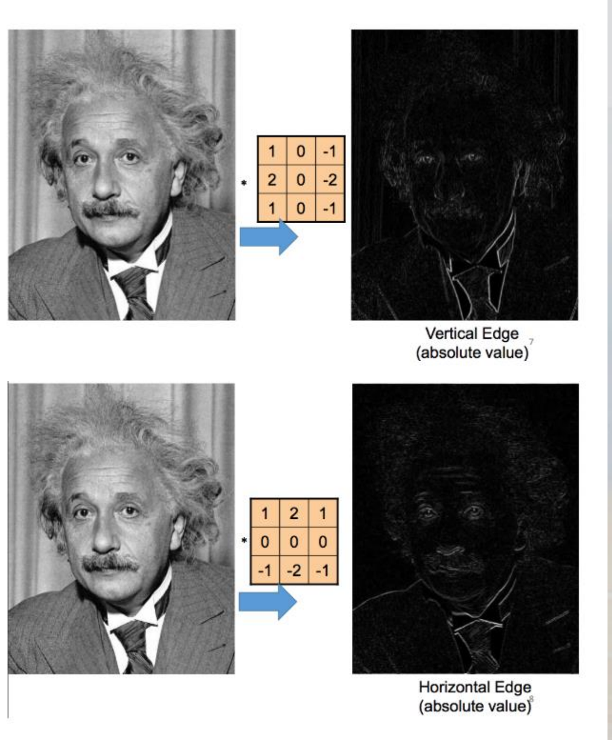
$$\text{Continuous: } (f * g)(t) = \int f(\tau)g(t-\tau)\, d\tau$$

$$\text{Discrete: } (f * g)[n] = \sum f[m]\, g[n-m]$$



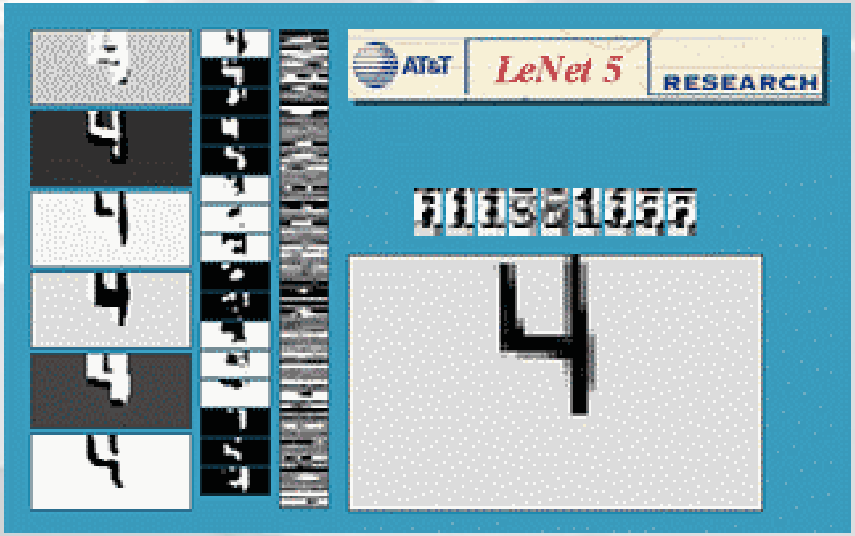


Padding; Stride (step)

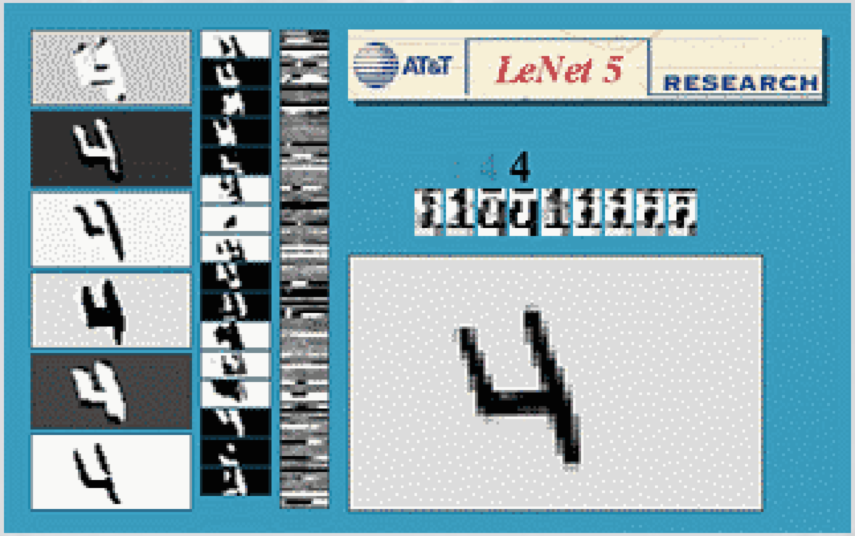


Vertical and horizontal edge filters

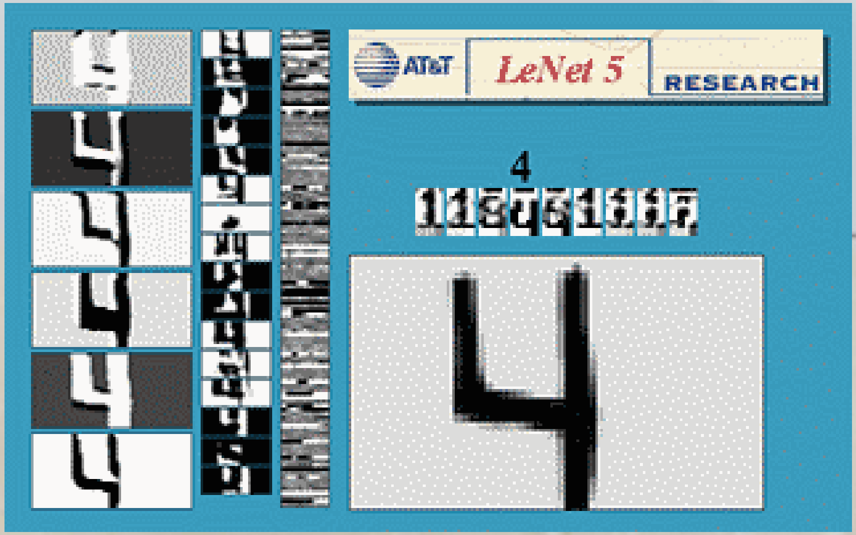
http://yann.lecun.com/exdb/lenet/index.html



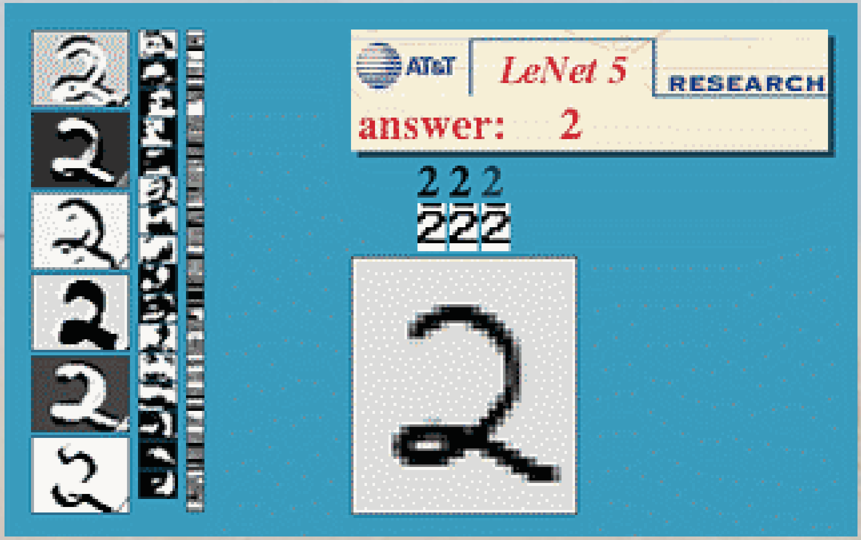
Translation Invariance



Rotation Invariance

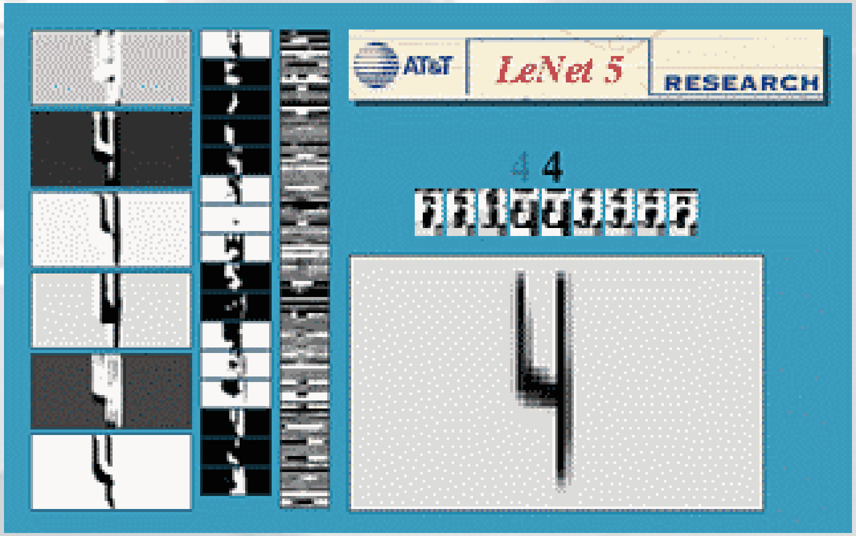


Scale Invariance

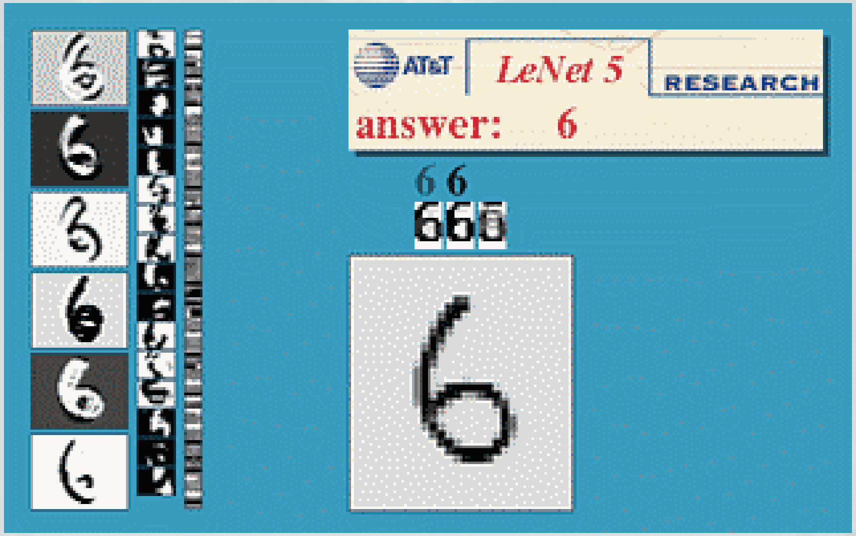


Resistance to Noise

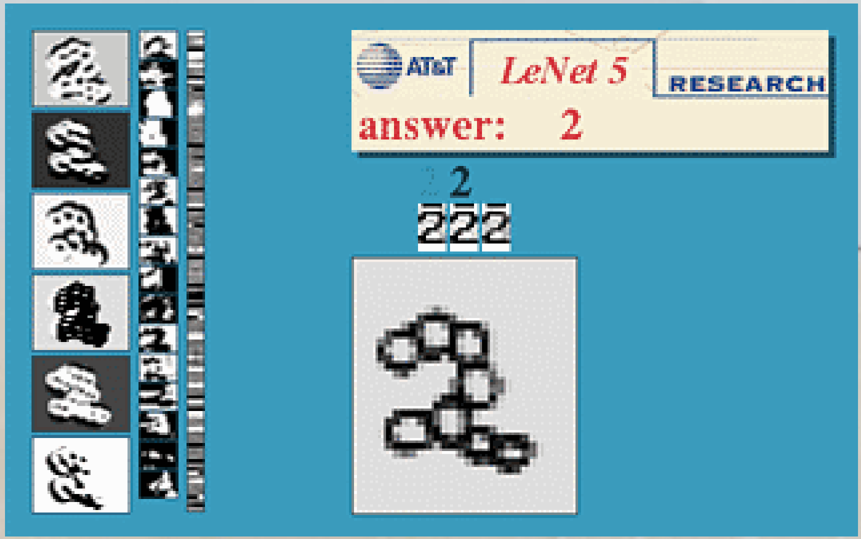
http://yann.lecun.com/exdb/lenet/index.html



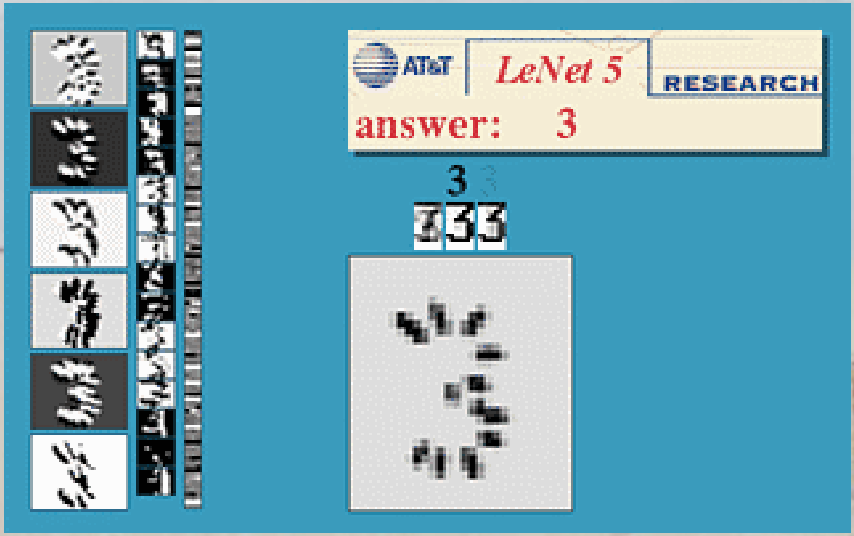
Aspect Ratio Invariance



Stroke Width Invariance



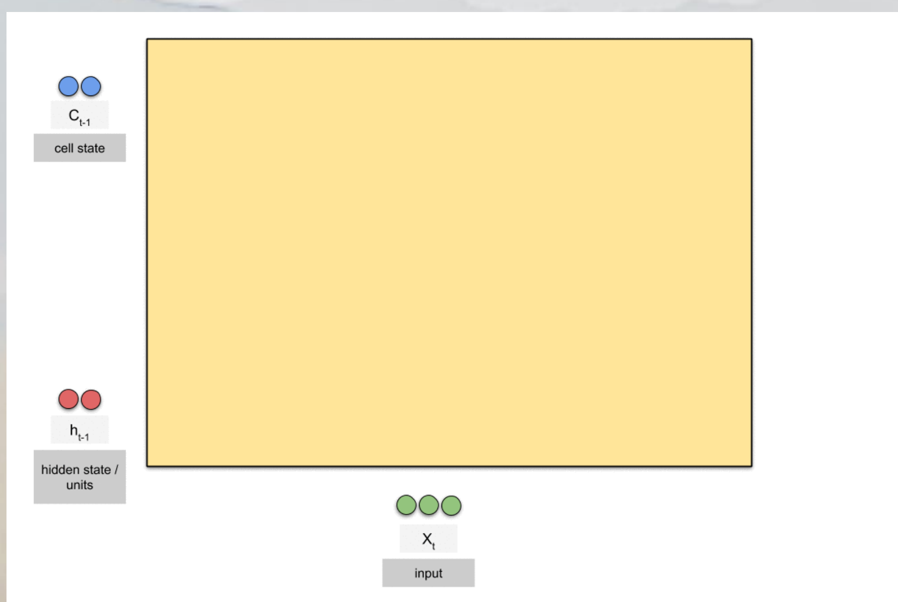
Weirdos

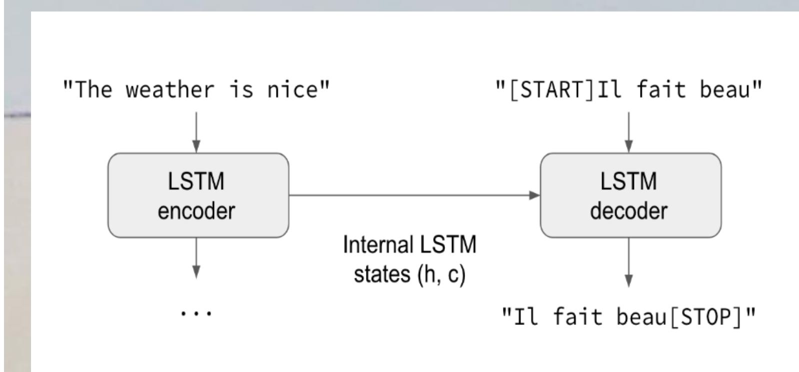


Weirdos

LSTMs: Solving the problem of vanishing gradients

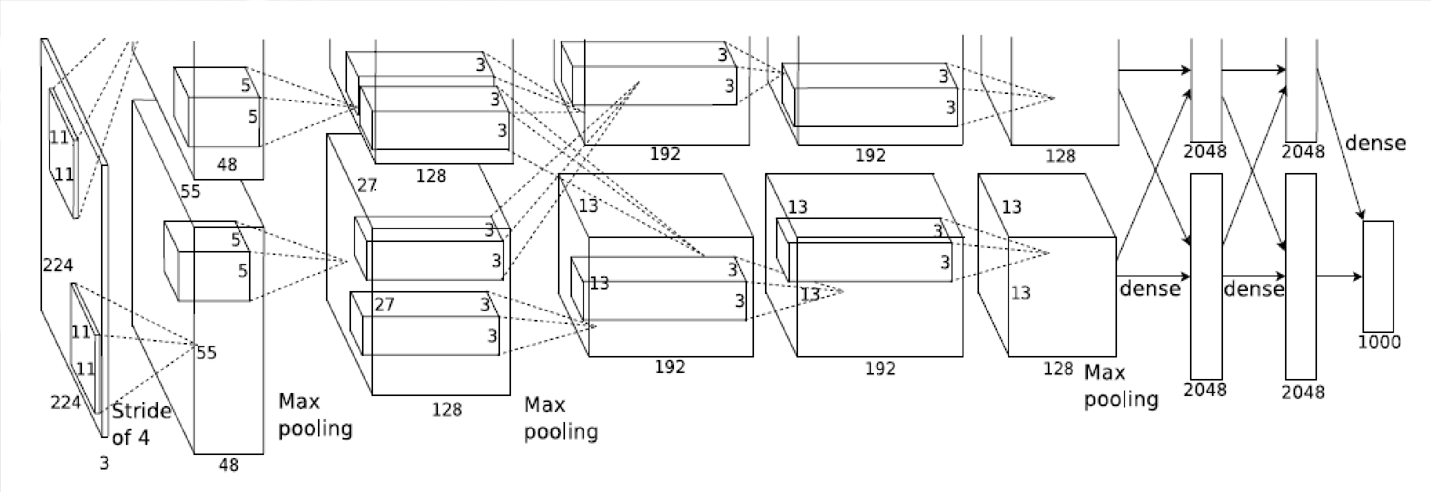
- Hochreiter & Schmidhuber, 1991, 1995
- Long Short Term Memory. “Selectively remember or forget”
- Later used with sequence to sequence networks:

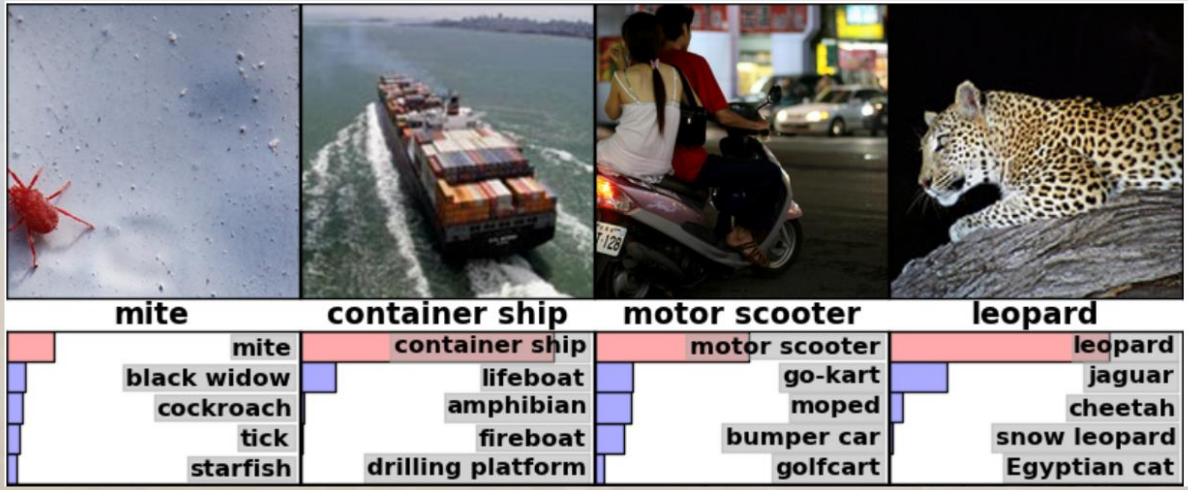




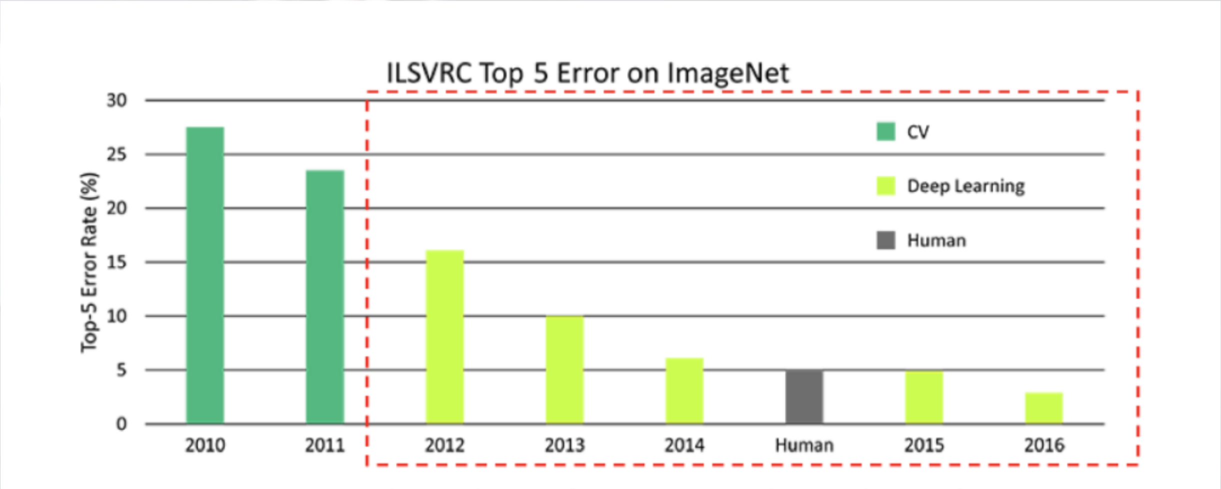
Animation credit: Raimi Karim

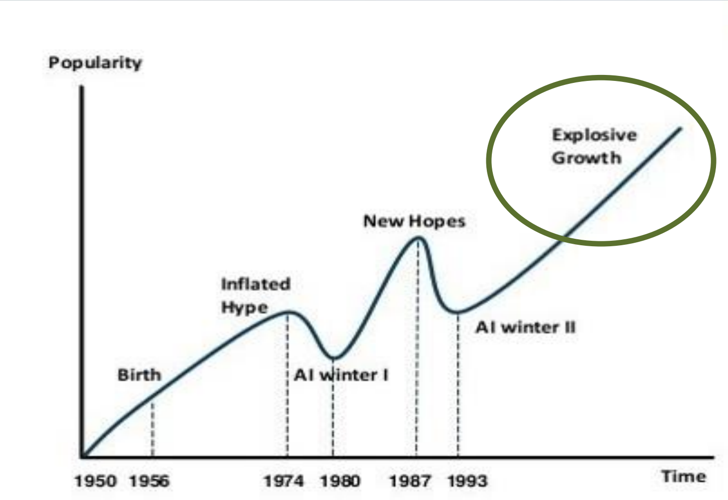
CNNs: 2012 AlexNet



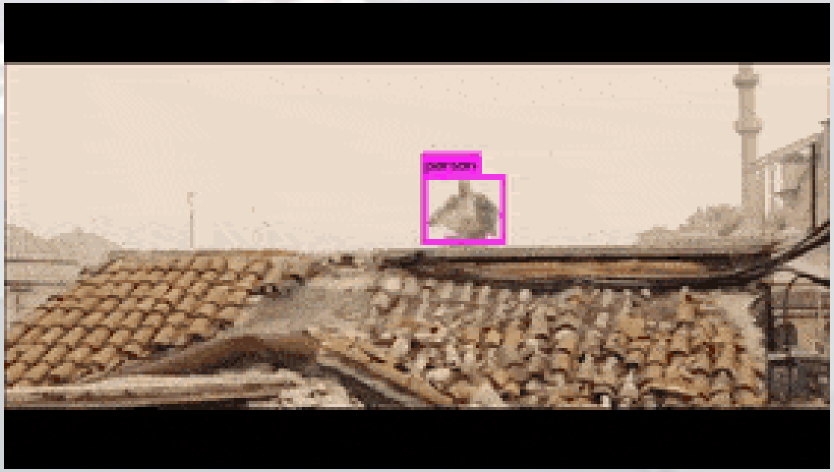


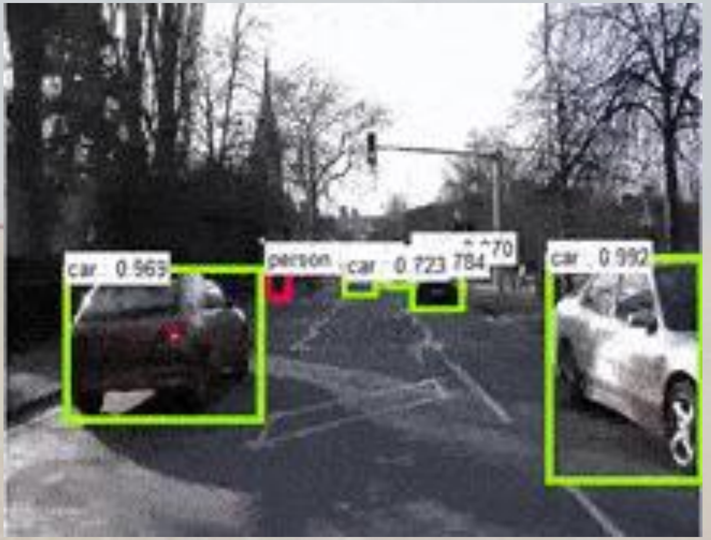
CNNs: 2012 AlexNet





From stills to videos: Sliding windows





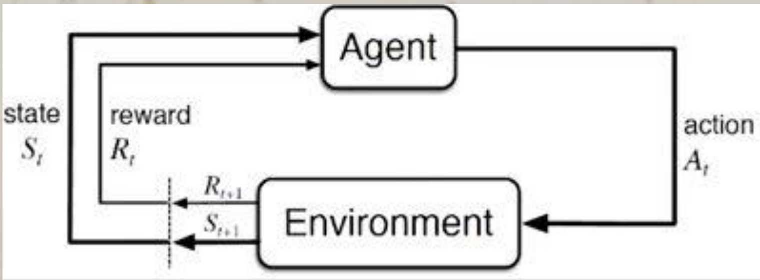
Neural network Machine Learning

- **Most is supervised Learning, but let's not forget:**

- **Unsupervised Learning eg autoencoders**
- **Reinforcement Learning**

Reinforcement Learning

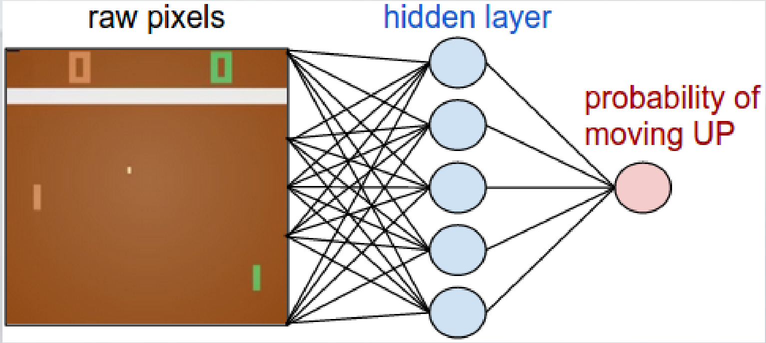
- Rich AI history from the 50's
- In its simplest form:
- Agent evolves in environment of states, performing actions leading to successive states, following a policy with rewards and penalties
- Learning consists in finding best policy maximizing rewards
- It is possible to learn policy with neural network.

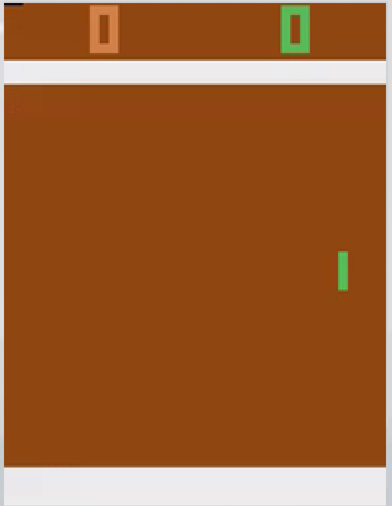


Reinforcement Learning: A simple example (2016)

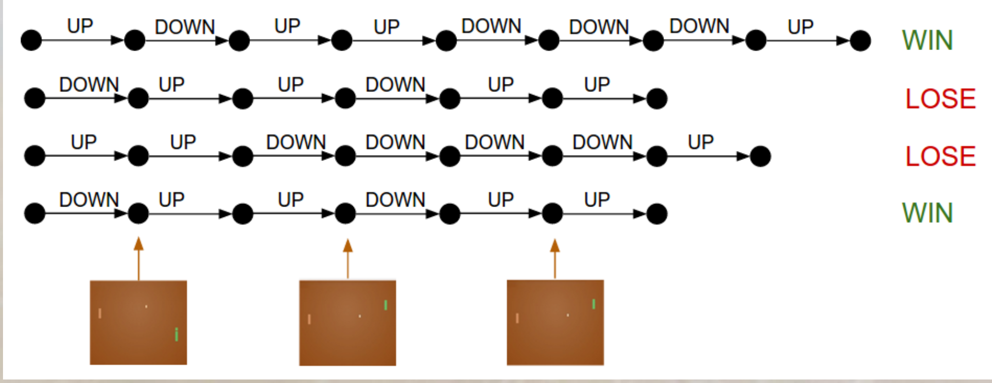
http://karpathy.github.io (Highly recommended if new to RL. Policy gradients)

- **Learn policy from raw pixels**
- **Well worth a close look:**
  130 lines Python implementation





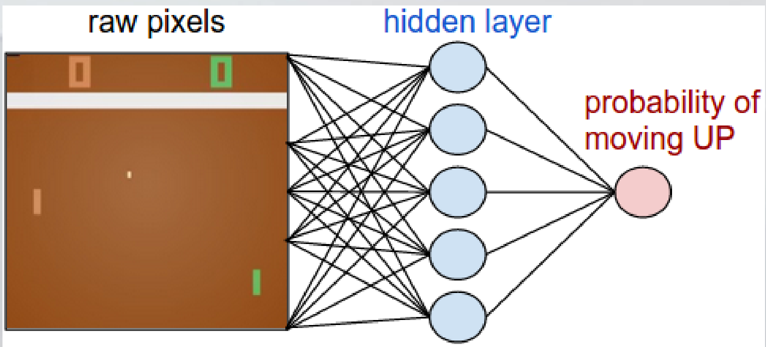
Start

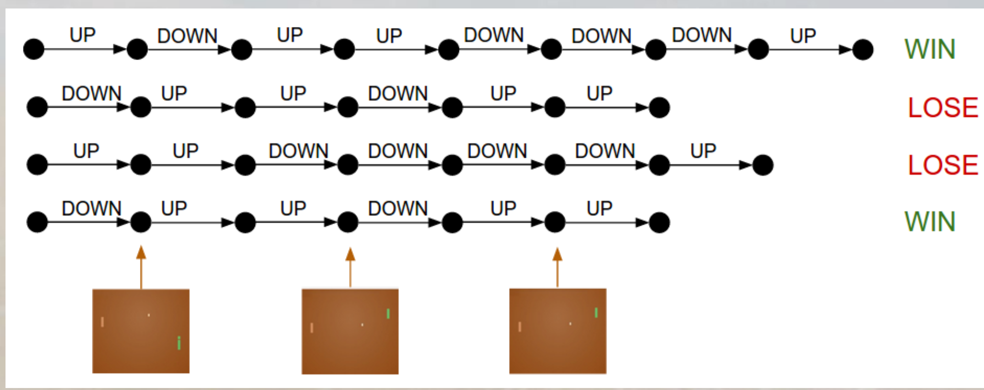


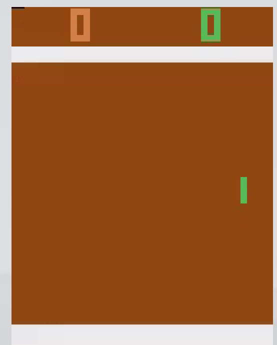
Reinforcement Learning: A simple example (2016)

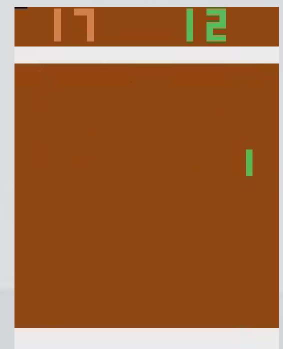
http://karpathy.github.io (Highly recommended if new to RL with policy gradients)

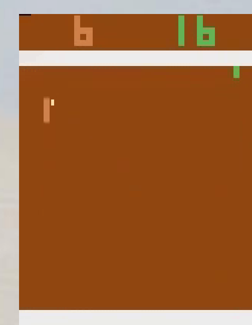
- Learn policy from raw pixels
- Well worth a close look:
  130 lines Python implementation






Start


~ ¾ training


Final

Neural Nets: Many architectures

Residuals Networks (ResNet)

GANS: Generative Adversarial Networks (two networks)

Attention networks

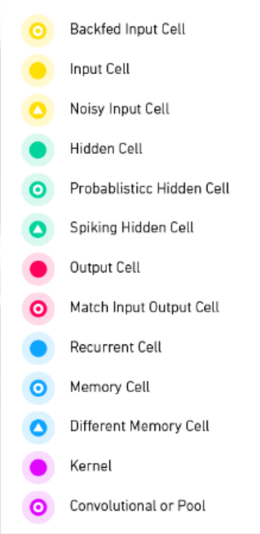
Auto encoders (Unsupervised learning)

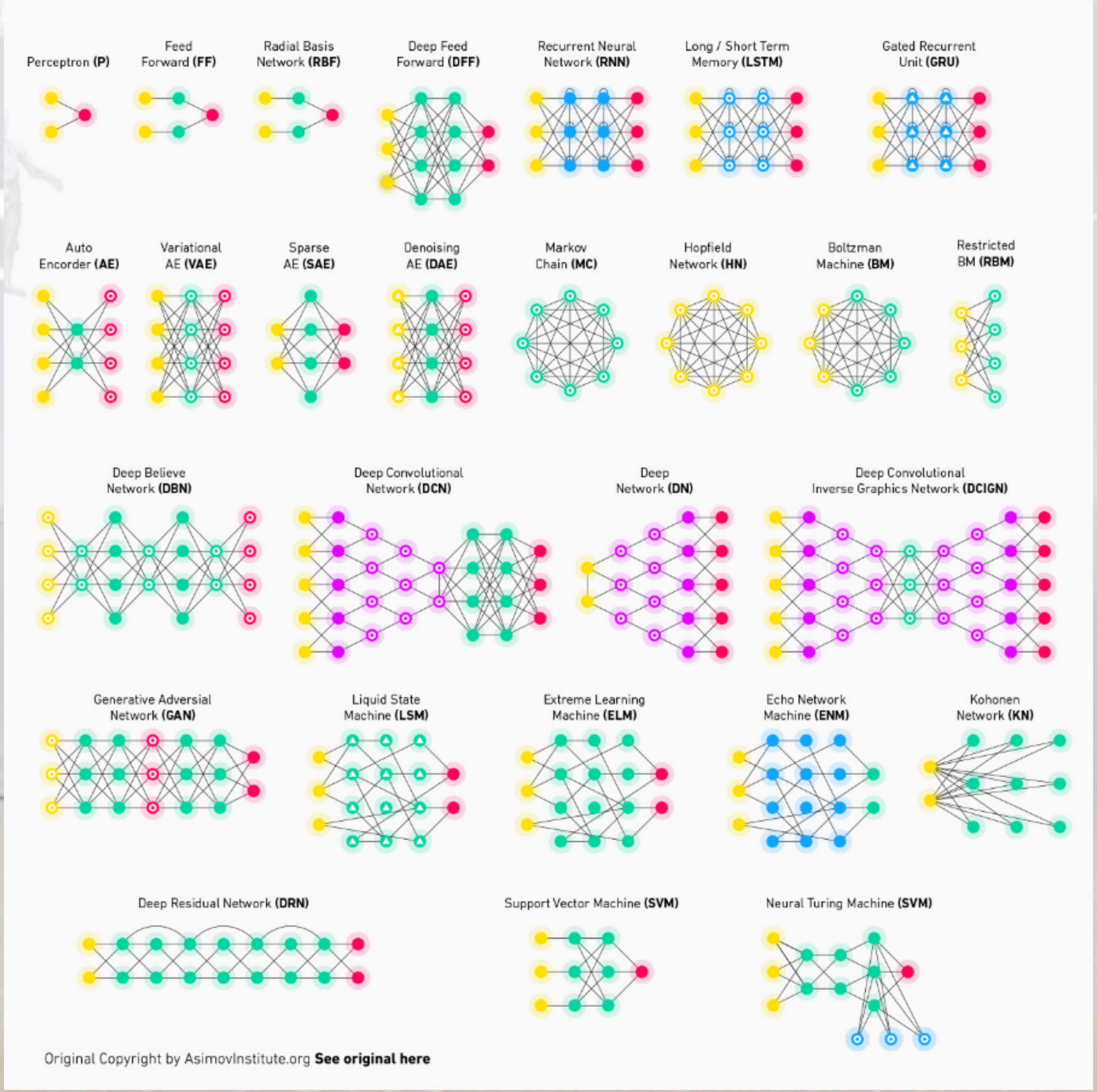
Variational Autoencoders

Restricted Boltzman machines

Boltzman Machines

…





Asimof Institute

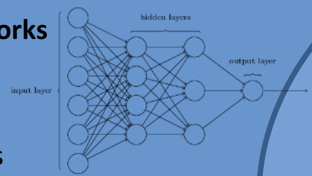
Machine Learning: we only scratched the surface

Brain inspired: Sub-Symbolic Learning

Symbolic Learning

Convolutional Neural Networks (CNNs)

Multi-Layer Neural Networks



Multi-Layer Perceptrons (MLPs)

Boltzman Machines

Generative Adversarial Networks (GANs)

Auto Encoders

Transformer Networks

Restricted Boltzman Machines

Deep Belief Networks

Support Vector Machines

Reinforcement Learning

Linear and logistic regression

Decision Trees

Ensemble Learning

Clustering

Linear Programming

Principle Component Analysis

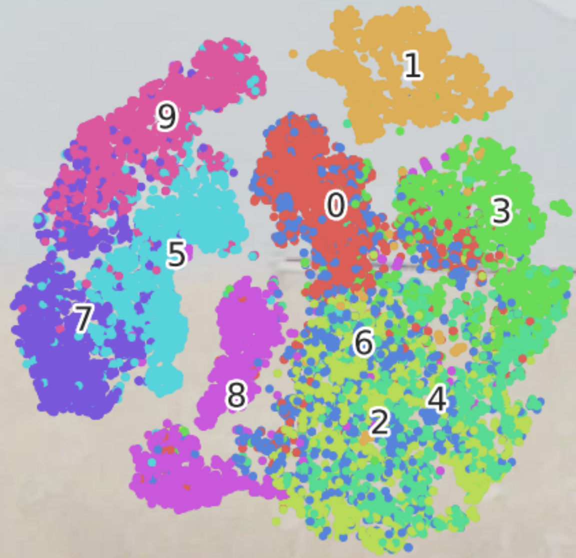
Bayesian Networks

Supervised Learning, Unsupervised Learning, Reinforcement Learning

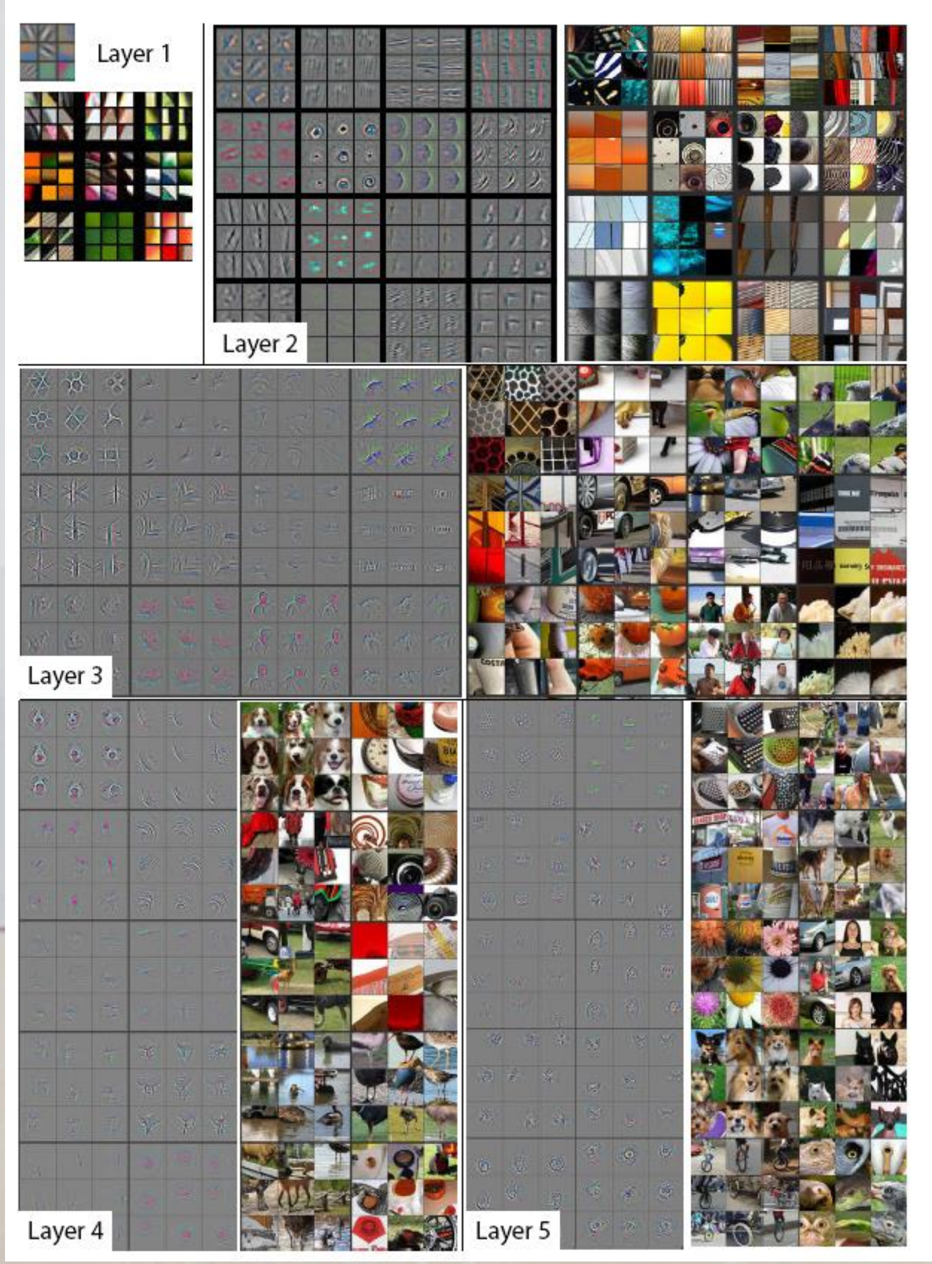
Deep Learning:
Many intermediate layers between input and output

Limitations of neural nets

- Black box, unexplainable
- Critique: It's just associations and curve fitting
- No concept of causality



Visualization with T-SNE



And thousands of applications …

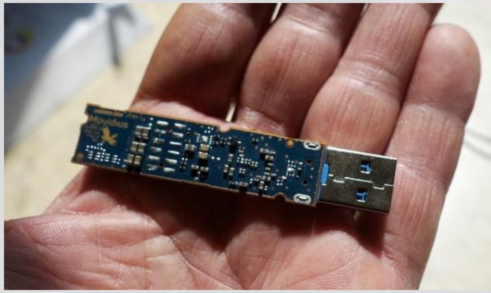
- Advertising: Consumer preferences, movie recommendations, etc …
- NLP Speech recognition, translation, document generation, chatbots (Alexa, Siri, …)
- Autonomous driving
- Just type AI in your favorite search box! AI powered this, AI powered that! ☺ ☹ ☺
- Beware the hype! Not even close to passing Turing Test (imho), forget about AGI!
- https://talktotransformer.com/ Amazing, but …

Hype: Caveat Emptor!

Hardware and programming frameworks

Hardware

- CPU
- GPU
- USB TPU dongle
- Google Colabs, Azure, AWS, … free time in some cases
- Edge AI

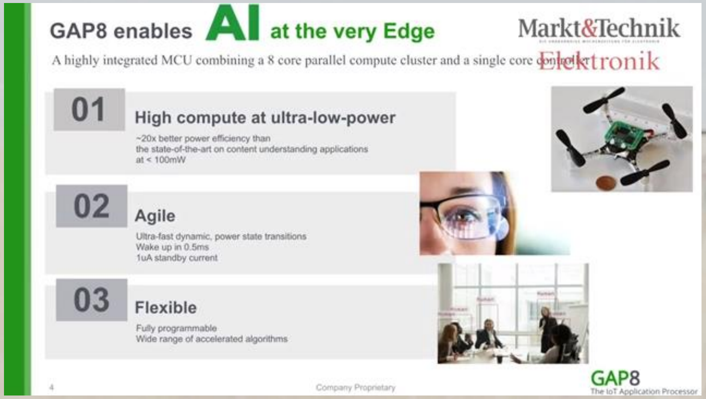


Intel Movidius

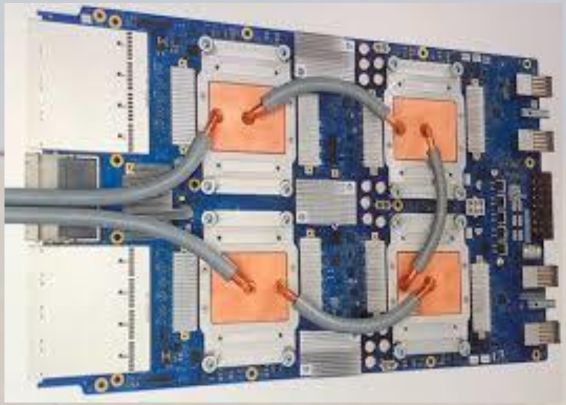
USB Google Coral

Nvidia Jetson Nano

Google Coral Board

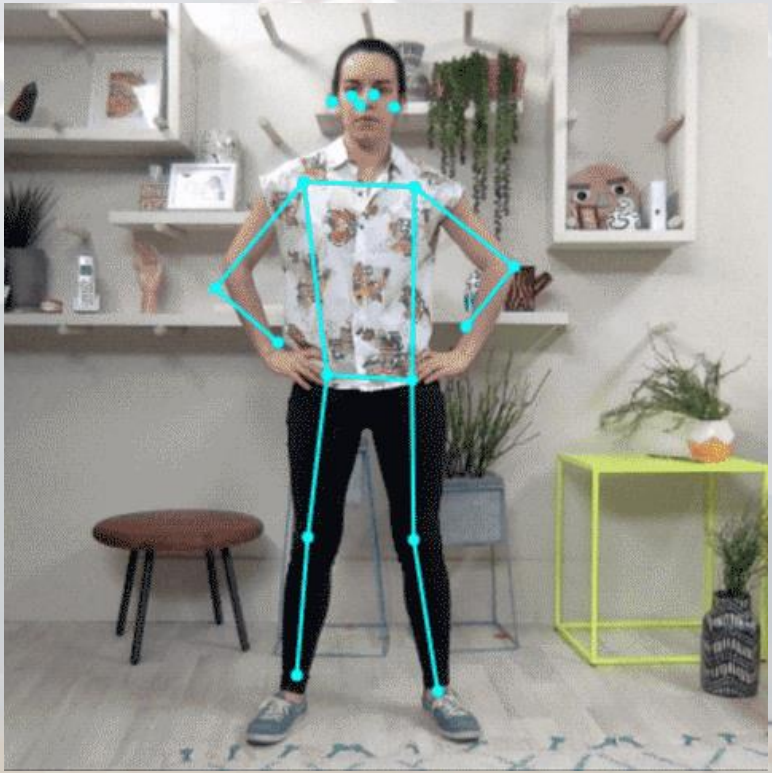


Gap 8 SoC

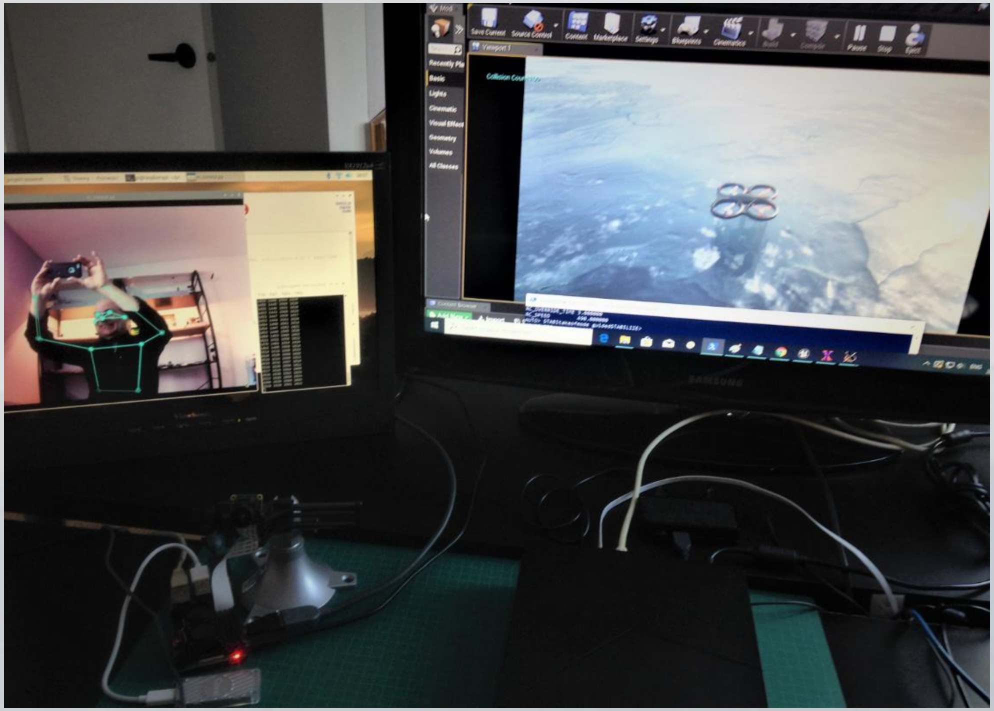


Google TPU on cloud

Hardware



PoseNet



Flying with Gestures Patrick Poirier
Pi4 + Google Coral 20FPS with posenet
https://gitter.im/ArduPilot/VisionProjects

Programming options

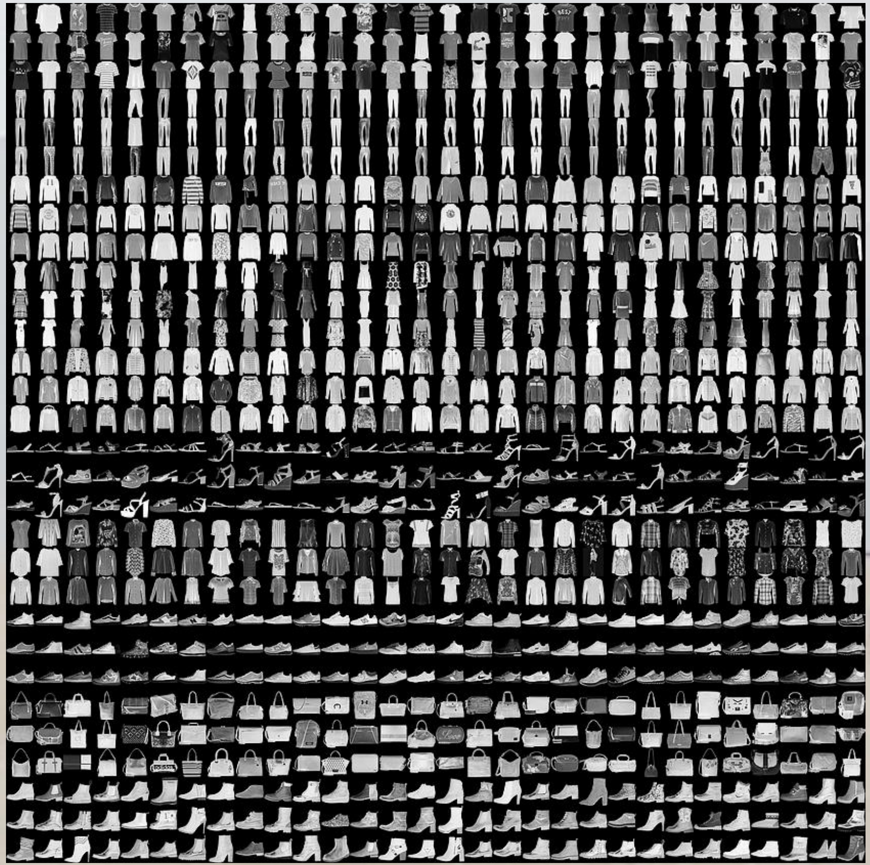
- Python, Raw C/C++/CUDA, …. ☹
- Keras: Tensorflow API, multibackend (tensorflow, CNTK, Theano) tf.keras: Tensorflow only, integrated in TensorFlow Very powerful despite simplicity
- TensorFlow (Google), PyTorch (Facebook) both Open Source)
- HAL (GPU, CPU, TPU, X on cloud) No worries about underlying hardware.
- Rich ecosystems
- Also Tensorflow Lite, TinyML (Arduino Nano and STM32!)
- ONNX (Open Neural Network Exchange) for sharing and framework interoperability, import/export to/from cloud

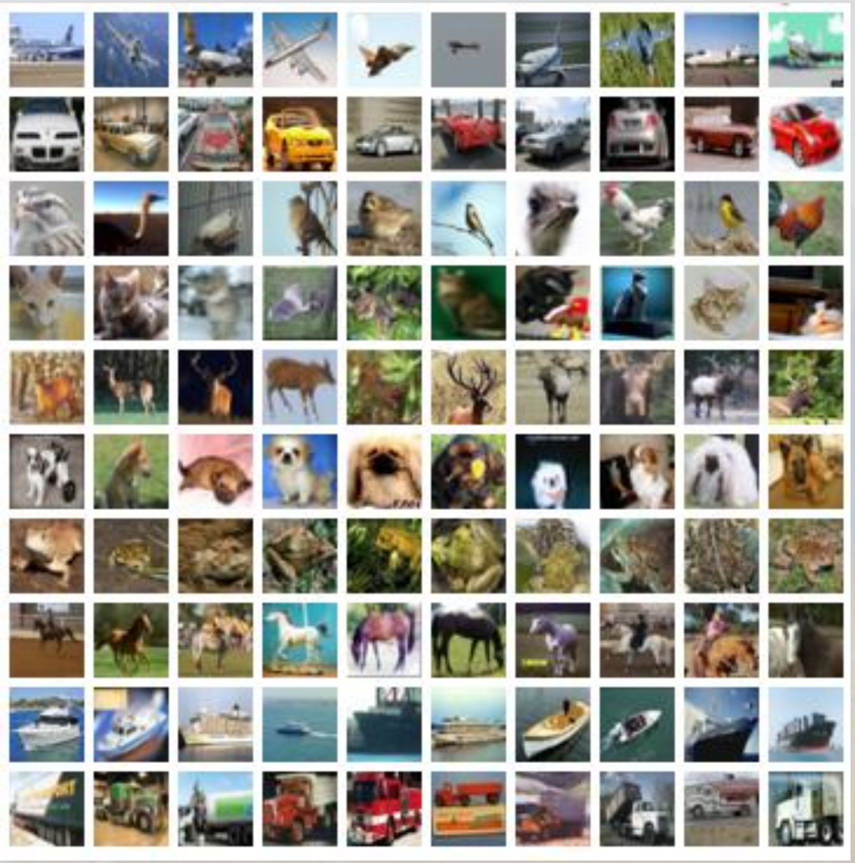
Programming options, advanced custom networks:

- Tensorflow Or PyTorch: <- 75% research papers (? …)
- PyTorch and Tensorflow revolve around tensors, generalization of matrices (n,m) dimensions to (n,m,o, …)
- Example: 1980x1080x3 channel RGB image: 3d tensor. Video: 1980x1080x(3 channel RGB) x frame#: 4D tensor
- Autodiff (TensorFlow), autograd (PyTorch): Dynamic computation graph keeping track of operations for most efficient gradient computations (Jacobians …)

Keras: Couple simple examples to show how easy it can be to get started

FashionMnist with MLP
Cifar10 with Resnet Conv. net





Demo

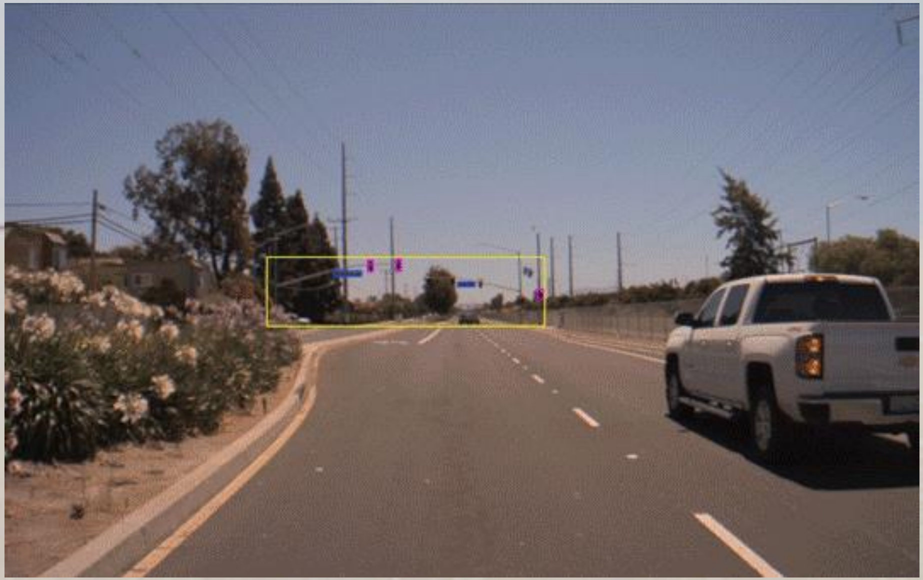
AI for UAVs and UGVs

AI for flying and ground motion

- **Supervised Learning or Reinforcement Learning**
- **Simulation or simulation+real world**

Autonomous Driving, rovers

- **NVIDIA: LaneNet, WaitNet, SignNet**
- **LightNet, ClearSightNet**
- **Ride in NVIDIA Self-driving Car:**
- https://www.youtube.com/watch?v=1W9q5SjaJTc

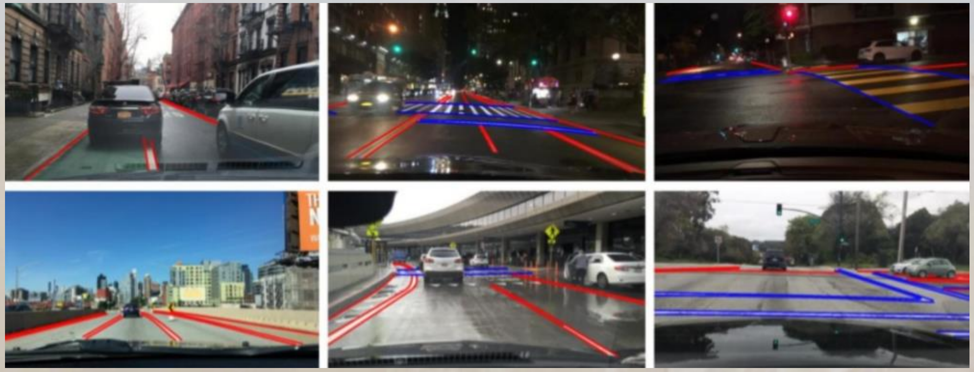


Driving Datasets

- Berkeley Deep Drive BDD 100k
- Baidu ApolloScope Dataset
- Comma.AI Dataset
- Oxford's Robotic Car Dataset
- Cityscapes Dataset
- Kitti Dataset
- Ford Campus Vision And Lidar Dataset
- Motion-based Segmentation And Recognition Dataset
- TuSimple Dataset
- CMU Visual Localization Dataset
- CCSAD Dataset
- Kul Belgium Traffic Sign Dataset

- MIT Age Lab Dataset
- Lisa: Intelligent & Safe Automobiles, UCSD Datasets
- Udacity Challenge Datasets
- NCLT Datasets
- DIPLECS Autonomous Driving Datasets
- Velodyne SLAM Dataset
- Daimler Urban Segmentation Dataset
- The Uah-driveset
- DAVIS Driving Dataset 2017 (DDD17)
- Berkeley Deepdrive (BDD) Driving Model
- MIT-AVT: Autonomous Vehicle Technology

https://sites.google.com/site/yorkyuhuang/
Kang et al. 2019


Berkeley dataset, annotated


Udacity dataset

End to end supervised learning

- **ALVINN, CMU, 1989**

What's Hidden in the Hidden Layers?

The contents can be easy to find with a geometrical problem, but the hidden layers have yet to give up all their secrets

David S. Touretzky and Dean A. Pomerleau

AUGUST 1989 • BYTE 231

tions, we fed the network road images taken under a wide variety of viewing angles and lighting conditions. It would be impractical to try to collect thousands of real road images for such a data set. Instead, we developed a synthetic road-image generator that can create as many training examples as we need.

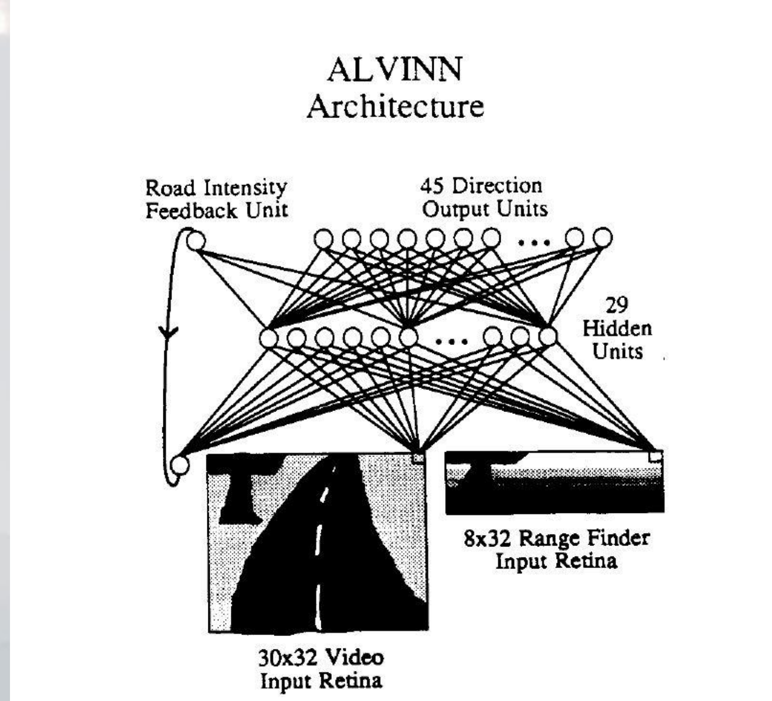
To train the network, 1200 simulated road images are presented 40 times each, while the weights are adjusted using the back-propagation learning algorithm. This takes about 30 minutes on Carnegie Mellon's Warp systolic-array supercomputer. (This machine was designed at Carnegie Mellon and is built by General Electric. It has a peak rate of 100 million floating-point operations per second and can compute weight adjustments for back-propagation networks at a rate of 20 million connections per second.)

Once it is trained, ALVINN can accurately drive the NAVLAB vehicle at about 3½ miles per hour along a path through a wooded area adjoining the Carnegie Mellon campus, under a variety of weather and lighting conditions. This speed is nearly twice as fast as that achieved by non-neural-network algorithms running on the same vehicle. Part of the reason for this is that the forward pass of a back-propagation network can be computed quickly. It takes about 200 milliseconds on the Sun-3/160 workstation installed on the NAVLAB.

The hidden-layer representations ALVINN develops are interesting. When trained on roads of a fixed width, the network chooses a representation in which hidden units act as detectors for complete roads at various positions and orientations. When trained on roads of variable

continued

Photo 1: *The NAVLAB autonomous navigation test-bed vehicle and the road used for trial runs.*



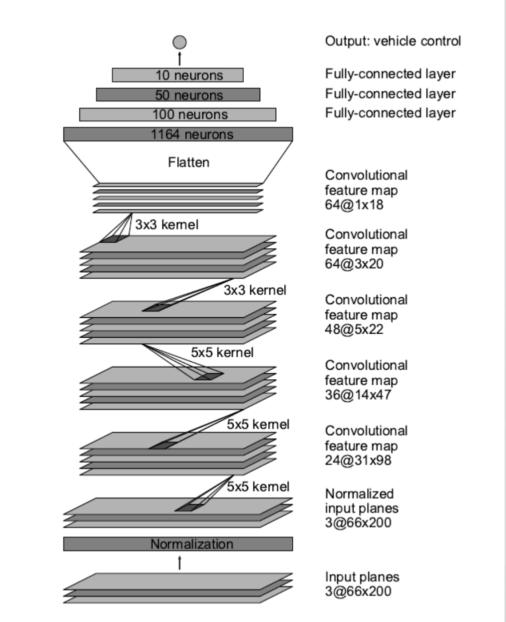
Pomerleau, 1989

ARDUPILOT

3DVistas
Drones for Research and Industry

End to end learning

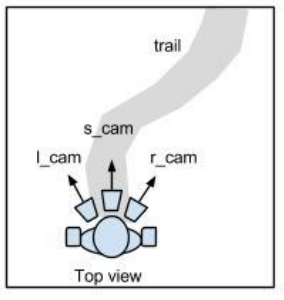
- **Nvidia, 2016. Input: road images, output: steering**

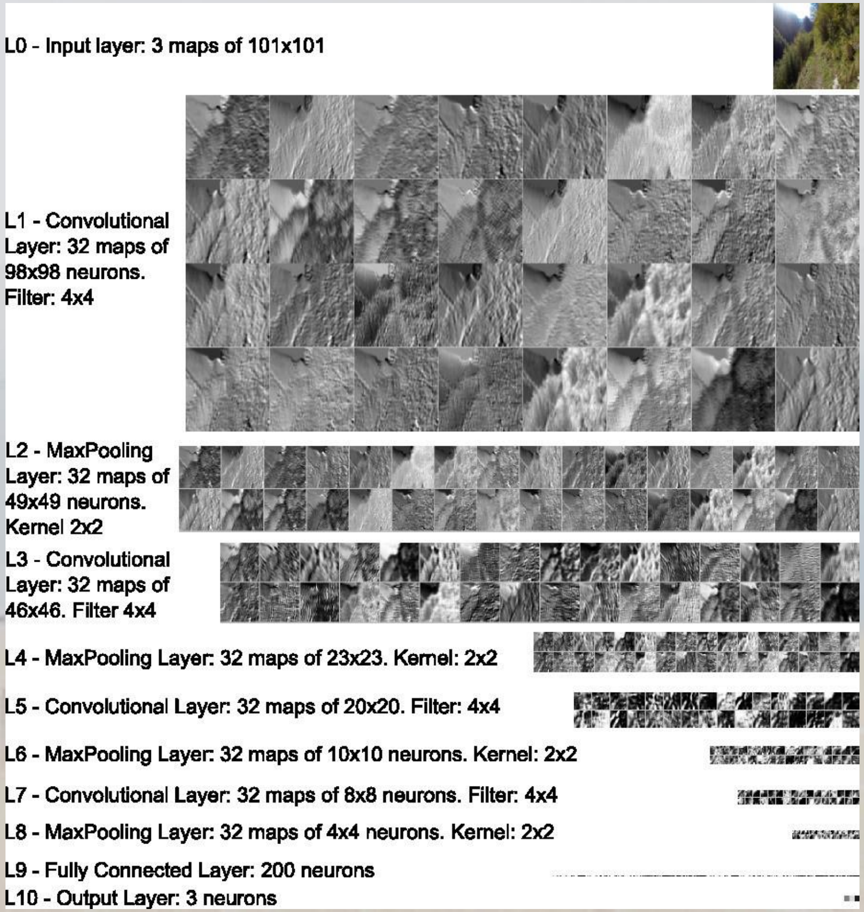


Nvidia, 2016

Drone Forest Trail Autonomous Navigation, 2016

http://people.idsia.ch/~giusti/forest/web/





- **Training examples acquired with 3 Gopros, 8 hours of video, 7km of trails, 20k+ images**
- **Convnet: 10 layers, 150k Weights, 500k Neurons, 57 Million Connections**
- **Parrot drone with net running on laptop via wifi**
- **Experiments alo made with onboard Odroid-U3, 15fps achieved**

Success (top) and failures (bottom)

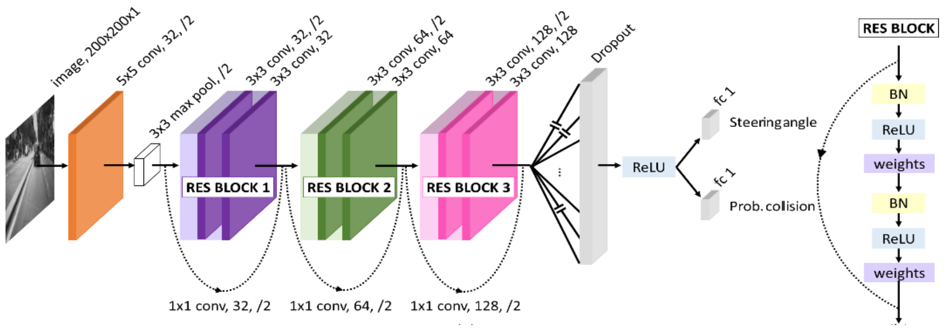
DroNet, Learning to fly by driving, 2018

http://rpg.ifi.uzh.ch/docs/RAL18_Loquercio.pdf

(a) Udacity dataset

- **ResNet architecture, trained with Keras/Tensorflow**
- **Split output: Steering Angle, Collision probability**
- **Two training datasets: Udacity car driving, 70k images for steering angle prediction; Gopro on bicycle handlebar, 32k images, manually annotated, for collision probability.**
- **Parrot Bebop; Velocity controlled from Core I7 laptop via wifi**

(b) Collected dataset



DroNet, Learning to fly by driving, 2018

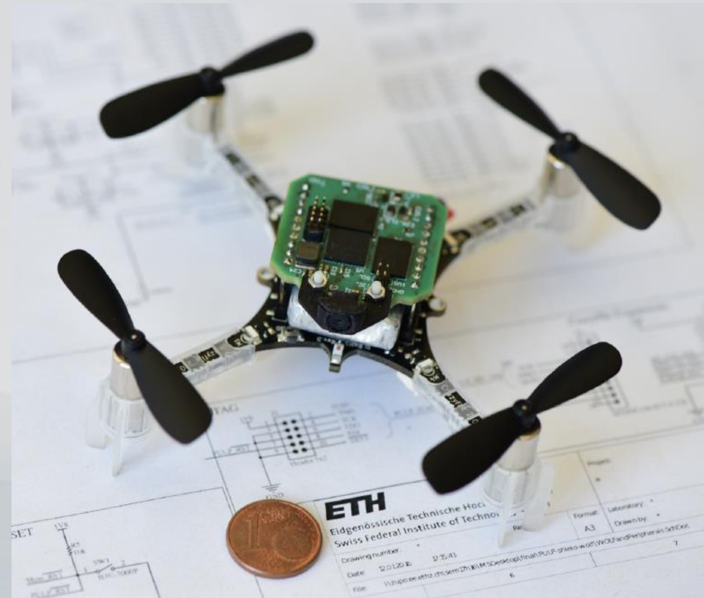
https://www.youtube.com/watch?v=ow7aw9H4BcA Code, datasets open sourced.

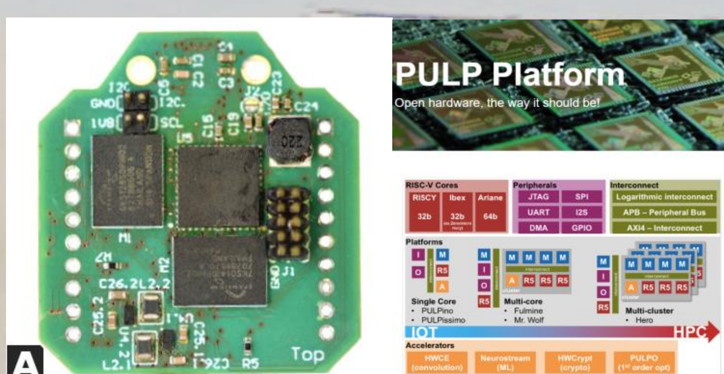
Pulp-DroNet, 2019

https://github.com/pulp-platform/pulp-dronet/

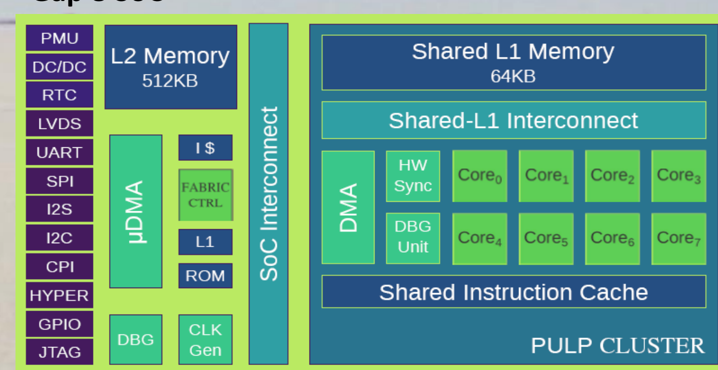
https://www.youtube.com/watch?v=57Vy5cSvnaA

- **Crazyflie 2.0 with Pulp-Shield open hardware**
- **27g, power draw < 300mw, 6 Fps visual processing**
- **Gap 8 SoC, 8 core, Greenwave Technologies**
- **Same network architecture as before, tweaked: Quantization (float32 to Fixed16), receptive field from max pooling layers size from 3x3 to 2x2**





Gap 8 SoC

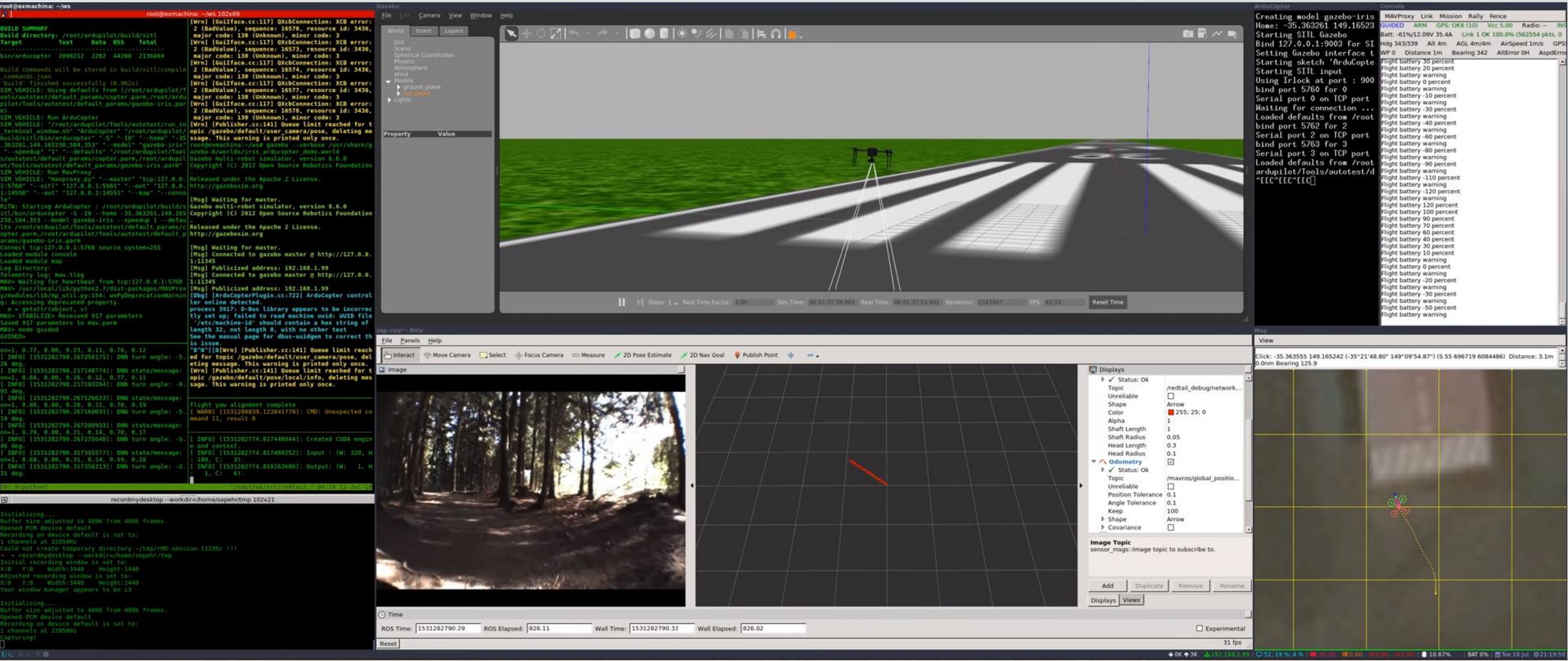


Pulp-DroNet, 2019

https://github.com/pulp-platform/pulp-dronet/

https://www.youtube.com/watch?v=57Vy5cSvnaA

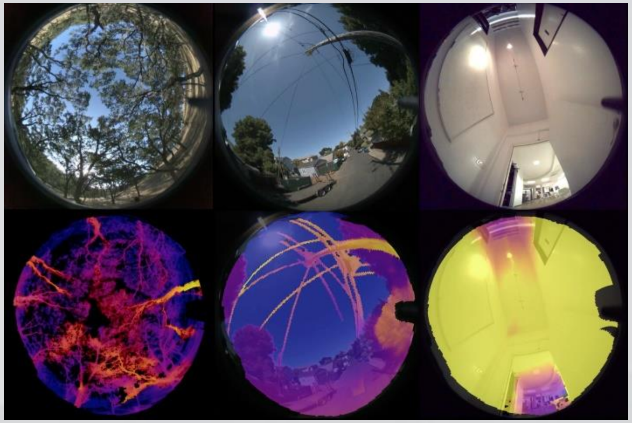
Nvidia Red Tail, GSOC 2018 Ardupilot port

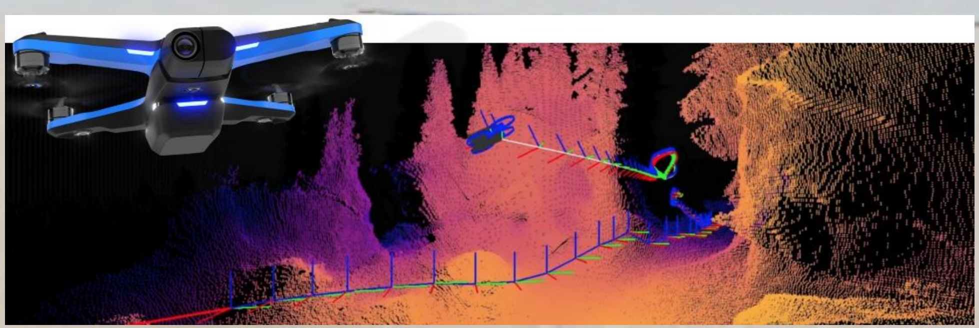


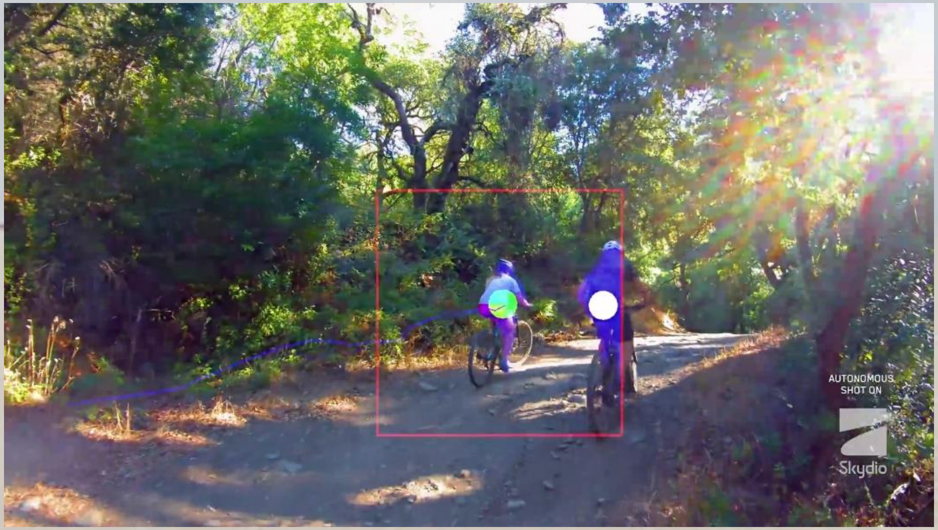
See also: SKYDIO

- Posts on Medium:
  - “Inside the mind of the Skydio 2”
  - ” Deep Neural Pilot on Skydio 2”

- CNN for 3D Map from redundant stereo pairs?
- CNN for Object tracking?
- --- ?



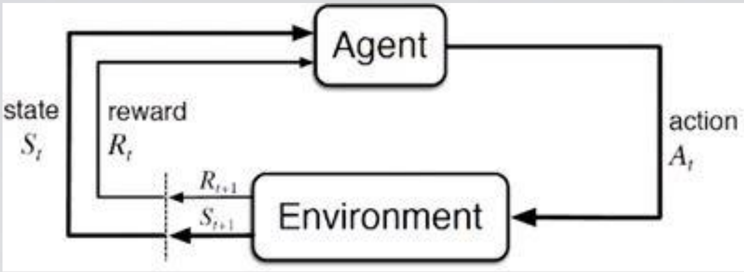




(Deep) Reinforcement Learning

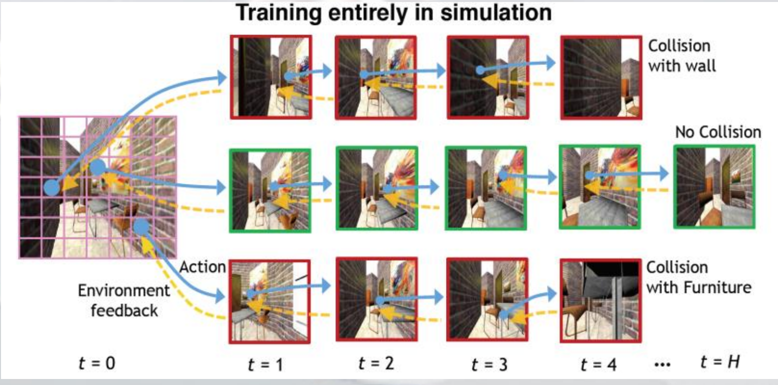
Reinforcement Learning

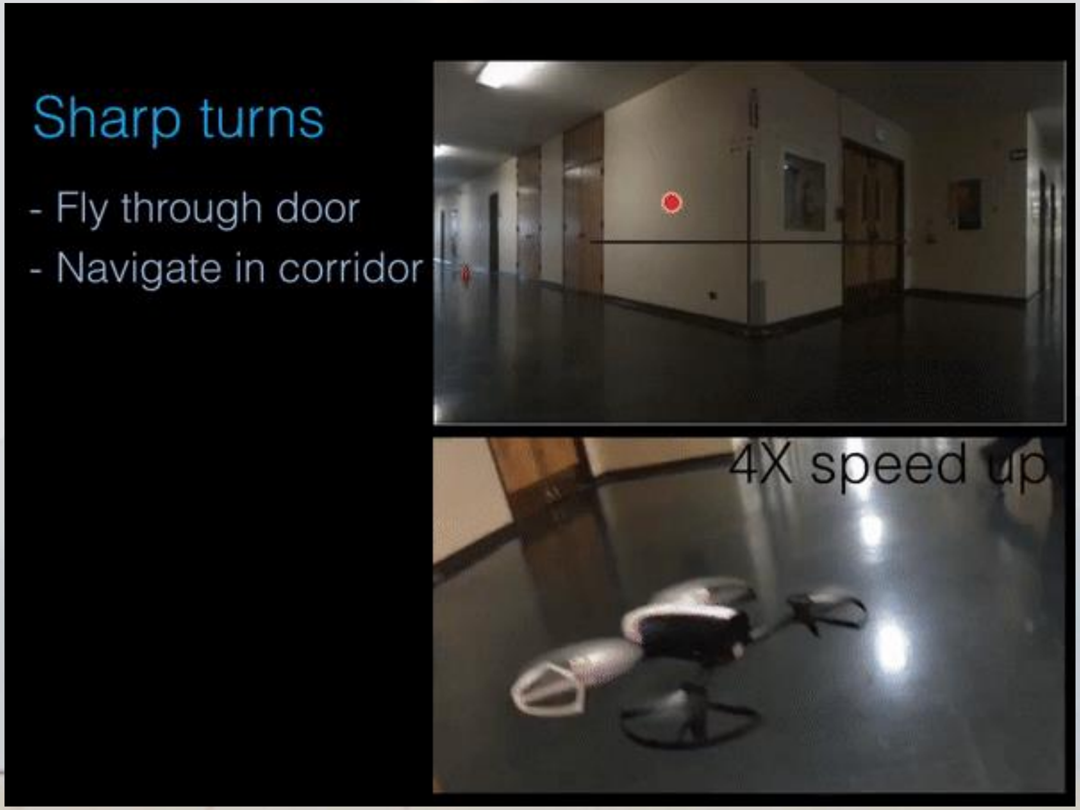
- Explore space: Fly randomly
- Crash, Hit/Avoid obstacle: Penalty/Reward
- Rinse and repeat, 1000's of times and learn optimal policy



- AirLearning
- Built on top of AirSim and OpenGym
- https://github.com/harvard-edge/airlearning

CAD²RL: "Real Single-Image Flight Without a Single Real Image", 2017





- Deep RL (VGG16)
  Training with Blender
  generated environment images

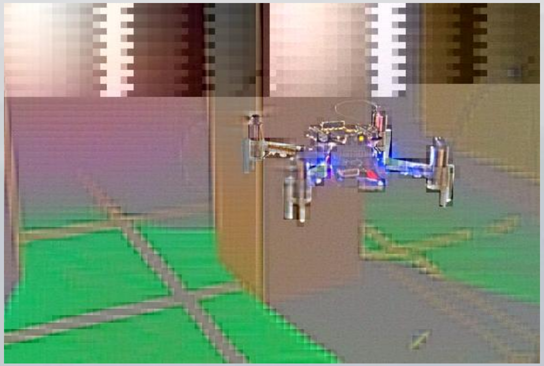
- VGG-16 Net. Randomize examples
  with different textures, lighting, objects,
  etc … to be able to generalize to
  real world

DeepRL (Cont) 2019

- https://github.com/gkahn13/GTS
- Generalization through Simulation: Integrating Simulated and Real Data into Deep Reinforcement Learning for Vision-Based Autonomous Flight, 2019
- BitCraze Crazyflie

DeepRL (cont). See also:

- "Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes Using Transfer Learning" , Oct. 2019 (DJI Tello)
  https://github.com/aqeelanwar/DRLwithTL_real



- Learning to Seek: Autonomous Source Seeking with Deep Reinforcement Learning Onboard a NanoDrone Microcontroller" Using Transfer Learning" , Sep 2019 (CrazyFlye) "
  https://github.com/harvard-edge/source-seeking

- "Learning to fly by crashing", 2017,
  https://youtu.be/u151hJaGKUo

Autonomous Drone Racing

Autonomous Drone Racing

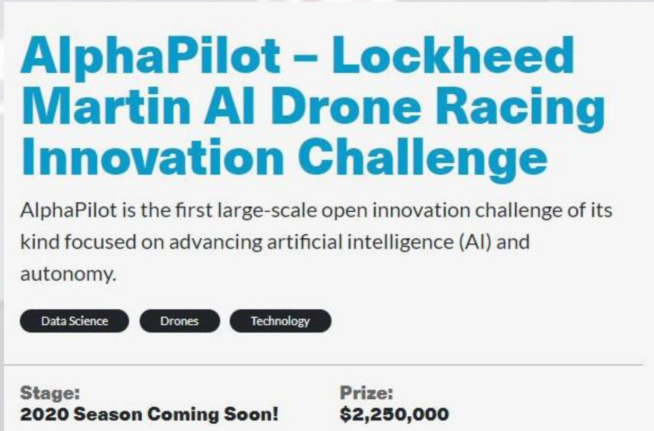
- Significant interest by the research community given input simplicity yet problem complexity

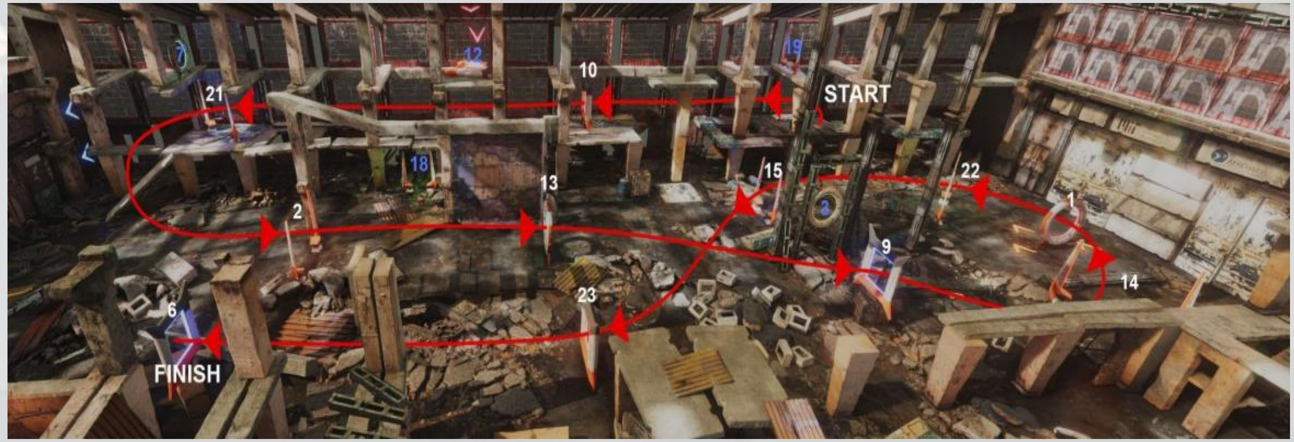
- Difficult for SLAM and visual odometry

- Challenges:

  - Dynamic environment
  - Drift with visual odometry
  - Speed
  - Limited onboard compute resources

Drone Racing: Competitions





FlightGoggles https://github.com/mit-fast/FlightGoggles

- **AlphaPilot. Pre-qualification with code sent to Lockeed Martin running on FlightGoggles simulator**
- **IROS ADR (Autonomous Drone Race) International Conference on Intelligent Robots and Systems**

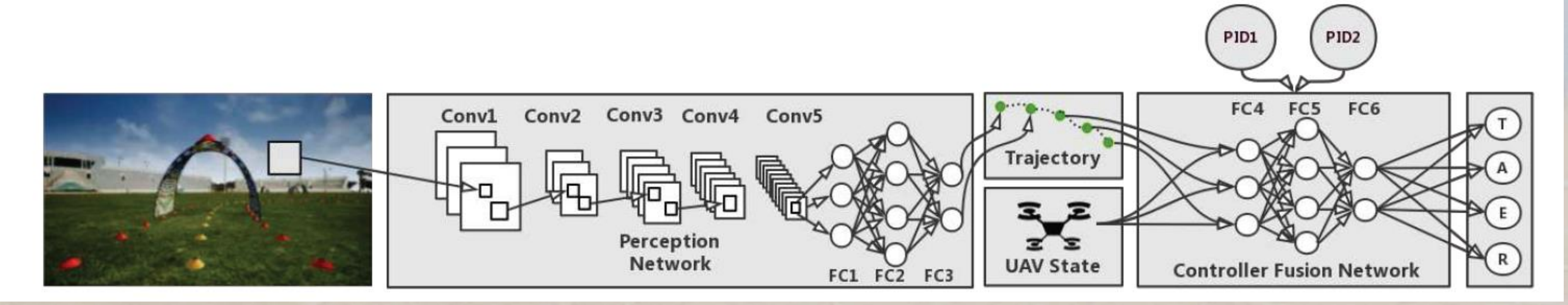
Drone Competitions

- Competion at NeuRIPS 2019: Game of Drones
- Based on AirSim
- https://github.com/microsoft/AirSim-NeurIPS2019-Drone-Racing
- https://microsoft.github.io/AirSim-NeurIPS2019-Drone-Racing/
- (Presentation, overview of participants approach, participants reports)

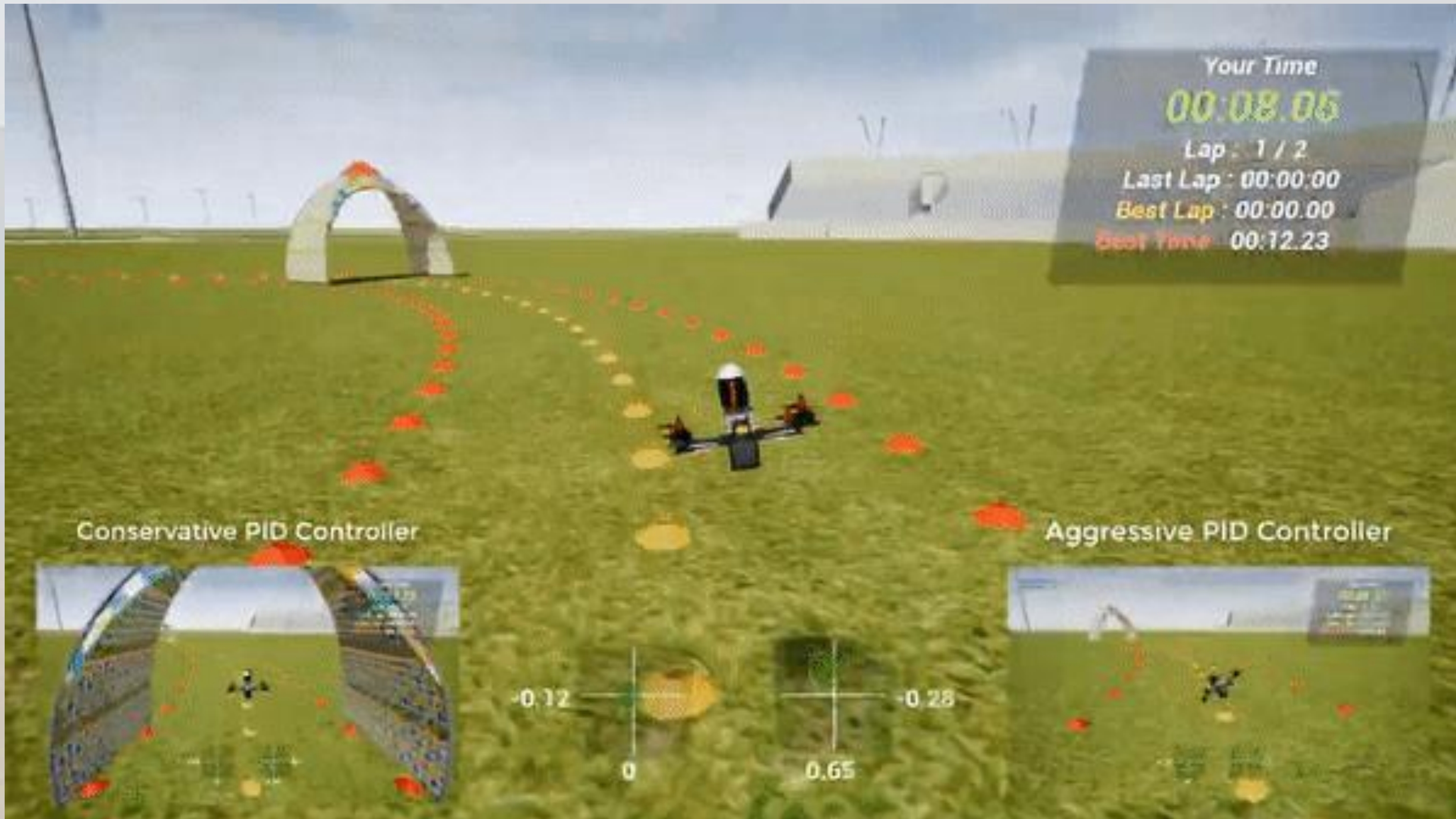
Teaching UAVs to race (KAUST)

- "Teaching UAVs to Race: End-to-End Regression of Agile Controls in Simulation, 2018"
- "Learning a Controller Fusion Network by Online Trajectory Filtering for Vision-based UAV Racing", 2019
  https://www.youtube.com/watch?v=hGKlE5X9Z5U,
  https://www.youtube.com/watch?v=9cbjOmKwbUY

- End to end via imitation learning
- Training and running on photo-realistic simulation, SIM4cv on top of UnReal, https://sim4cv.org/

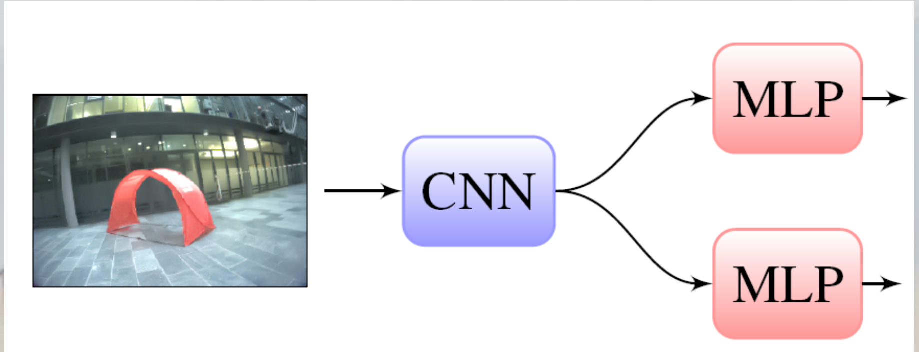


Teaching UAVs to race (KAUST)



ETH Zurich http://rpg.ifi.uzh.ch/research_drone_racing.html

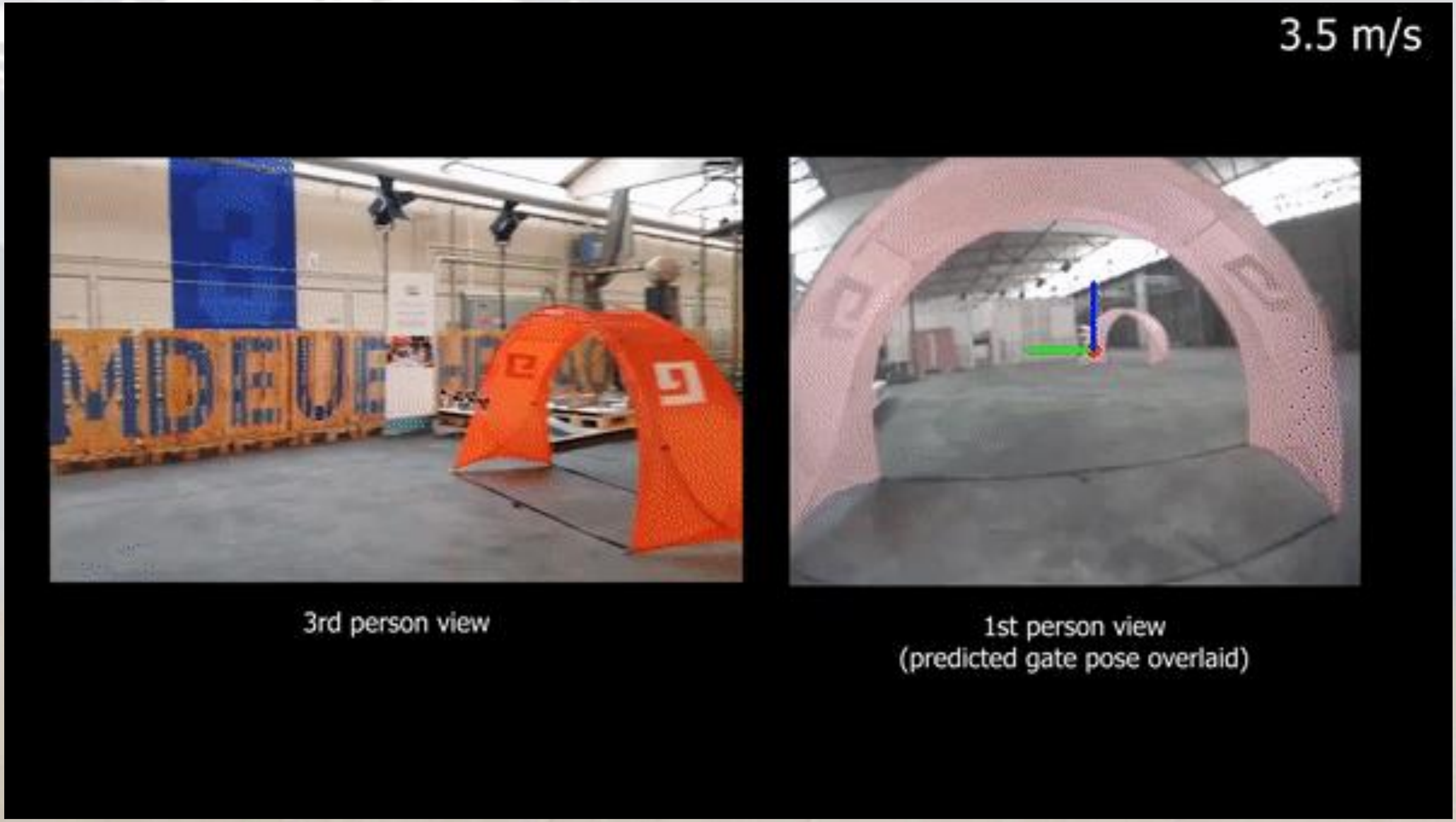
- "Beauty and the Beast: Optimal Methods Meet Learning for Drone", 3/2019
- "Deep Drone Racing: from Simulation to Reality with Domain Randomization", 11/2019
- Predicted waypoints in local body frame with CNN then fed to planner and tracker



Training samples:
Collected via measurement with manual flying

Output:
Mean and Variance of distribution describing next gate pose estimate distribution

ETH Zurich http://rpg.ifi.uzh.ch/research_drone_racing.html



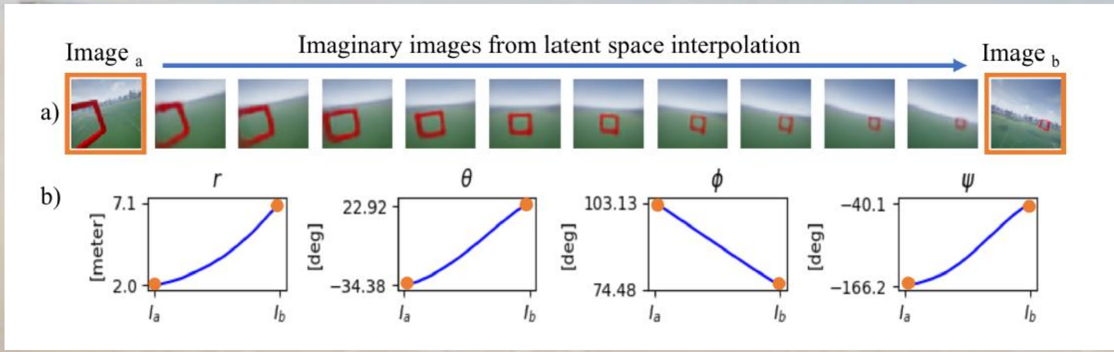
CMU, Microsoft https://arxiv.org/abs/1909.06993

- "Learning Controls Using Cross-Modal Representations: Bridging Simulation and Reality for Drone Racing", 9/2019

- Neural net used: variational autoencoder. Training of VA forces hidden representation to be as close as normally distributed as possible. => can reconstruct from arbitrary samples([Variational Autoencoders Tutorial](#)).

- Training Input/output: Cross modal: RGB image (Airsim), (body frame spherical coordinates and yaw), Dronet architecture, Resnet used.

- Training sample generation: Use planner, one gate ahead horizon, 14k data points



See also: “Autonomous drone race: A computationally efficient vision-based navigation and control strategy”, Nov 2018

https://arxiv.org/pdf/1809.05958.pdf

- TU-Delft, Delft University of Technology, The Netherlands
- Parrot Bebop with Paparazzi autopilot (Cortex A9, “dual core only”)
- Light-weight, not CNN, gate detection vision algorithm
- Beat ETH and won Alphapilot 2019 ($1 million)
- So there! ☺

Links, Survey Papers

- "A survey of deep learning techniques for autonomous driving", Oct. 2019, https://arxiv.org/pdf/1910.07738.pdf

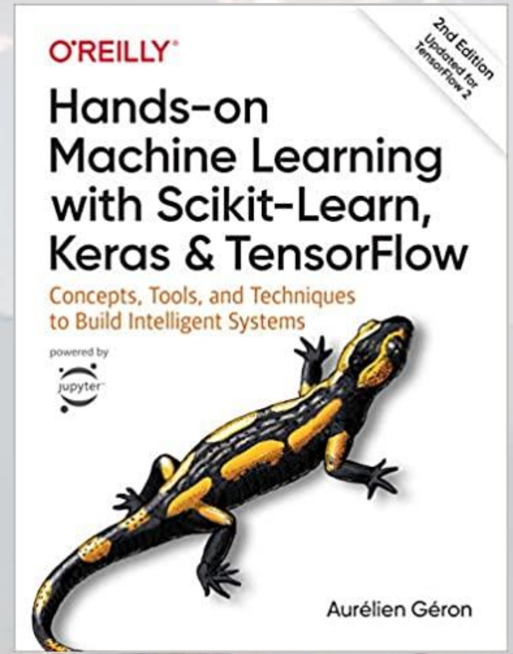
- "A Review on IoT Deep Learning UAV Systems for Autonomous Obstacle Detection and Collision Avoidance",Sep 2019, https://www.mdpi.com/2072-4292/11/18/2144

- "A Review of Deep Learning Methods and Applications for Unmanned Aerial Vehicles", 2017, https://www.hindawi.com/journals/js/2017/3296874/

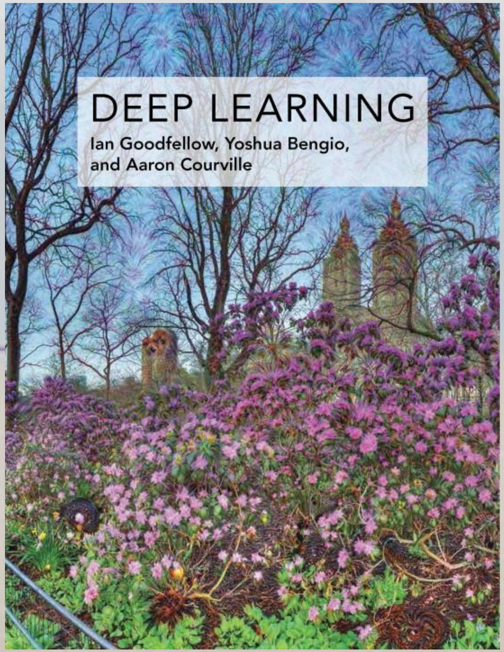
- Also of interest: Papers with code, https://paperswithcode.com/sota, if you want to check out the latest and greatest.

Recommended books

- **Applied:**



Theoretical deep dive, bible status:



Thank you!