Yandex

SCHOOL OF DATA ANALYSIS

Introduction into Deep Learning

Clermont University, Nov 2017

Andrey Ustyuzhanin
Yandex School of Data Analysis, Higher School of Economics

Artificial Neural Networks

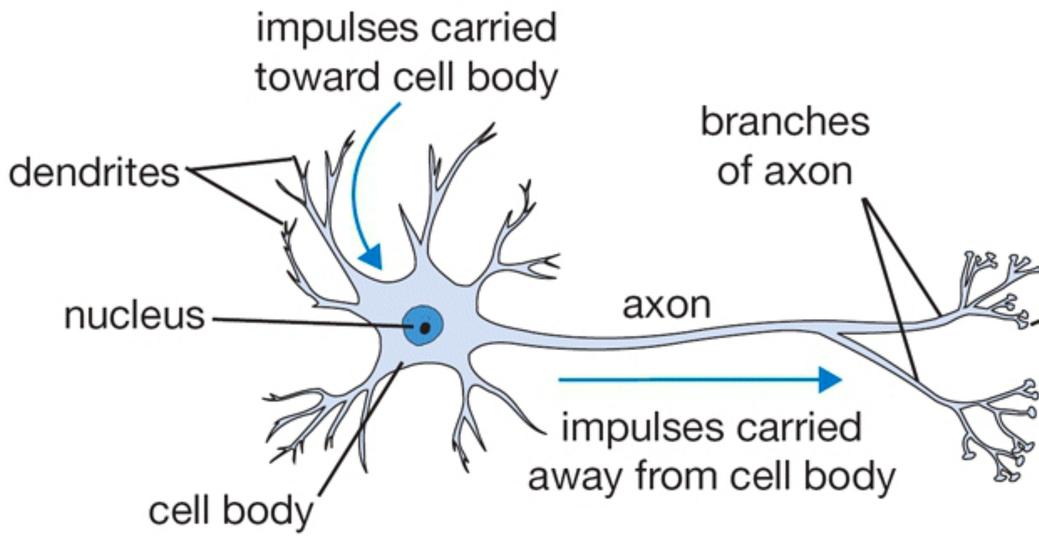
Artificial Neural Network history

Origin:

1943, W. McCulloch & W. Pitts — first computational model
1949, D. Hebb — method for learning 1954, W. Clark — first simulations

Further development:

1970, Linnainmaa, *Backpropagation* in the modern form
1979, Fukushima The earliest convolutional networks
1985, Rumelhart, Hinton, and Williams, backpropagation in neural networks could yield interesting representations
1989, LeCun,
convolutional networks + *backpropagation* = LeNet

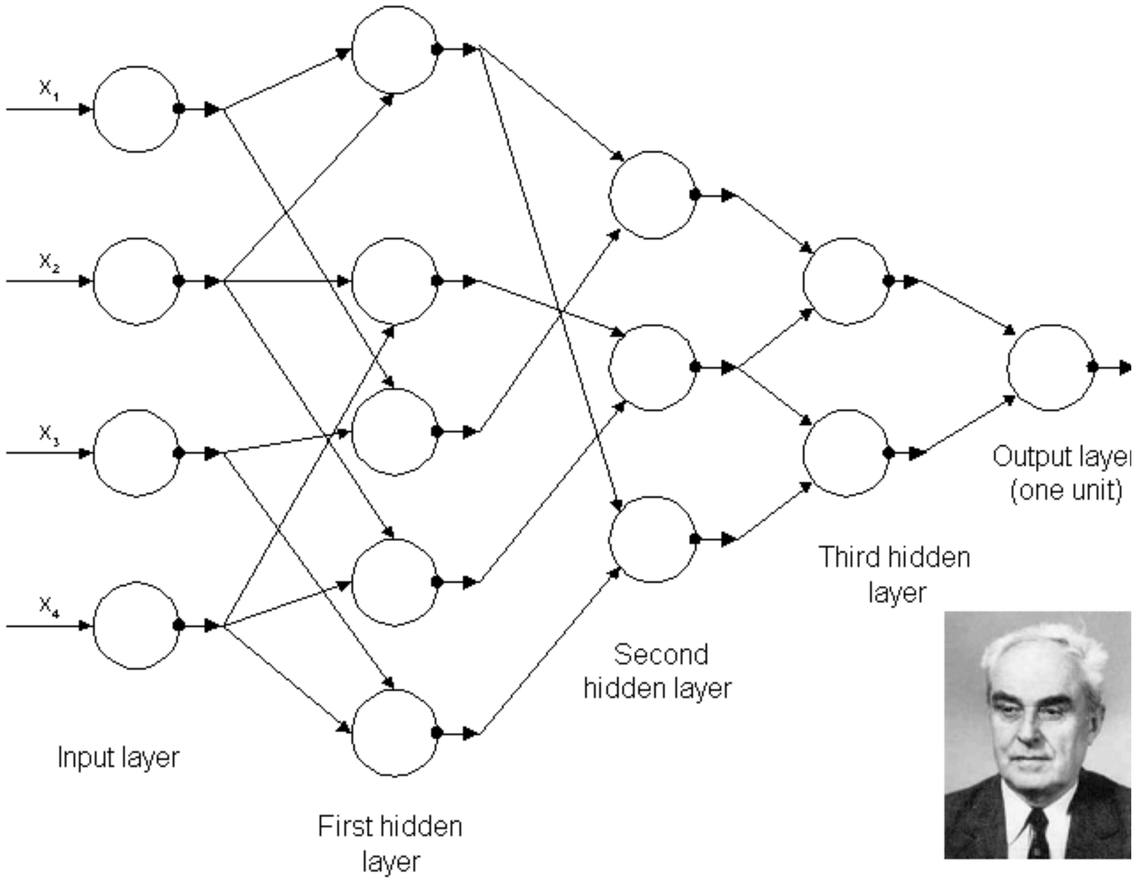


AI stages

1965 The first deep network architecture, A. Ivakhnenko (USSR)

1980s, AI winter - in applications simple (linear) models models (in particular, decision trees, SVM) clearly demonstrate better performance than Neural Nets

2006 — till now Neural networks Renaissance



Single Neuron

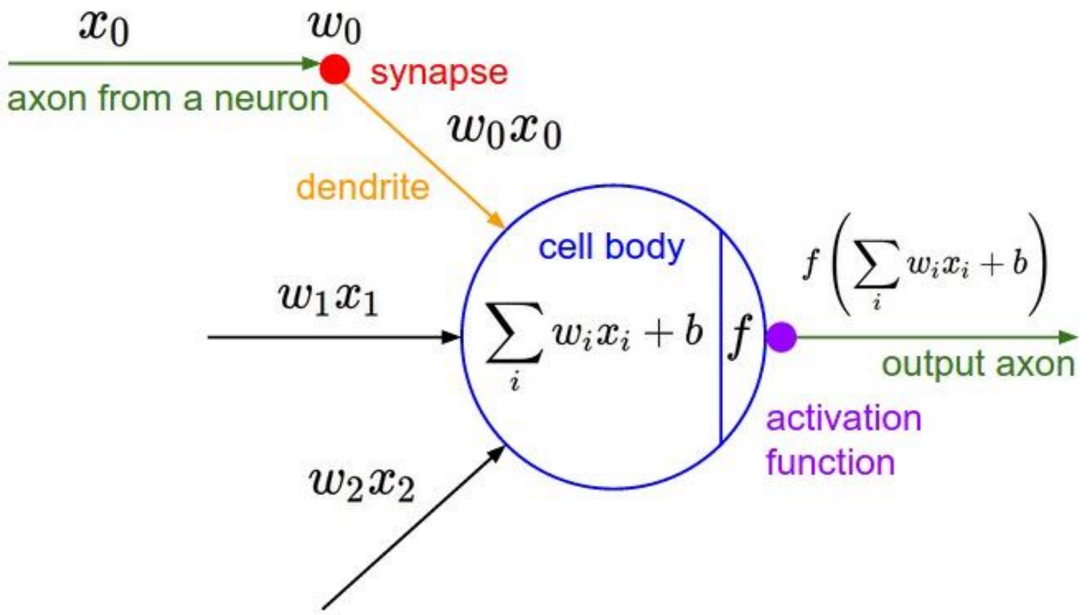
Neuron can be activated or not (h = 1, h = 0) by signals from receptors or other neurons.

in the simplest case:

$$h=\Theta\left(\sum w_i x_i + b\right)$$

- Θ(x) — Heaviside function
  (0, if x ≤ 0, 1 otherwise)
- non-smooth ⇒ hard to optimize

can we use other functions?



Activation functions

- Sigmoid:

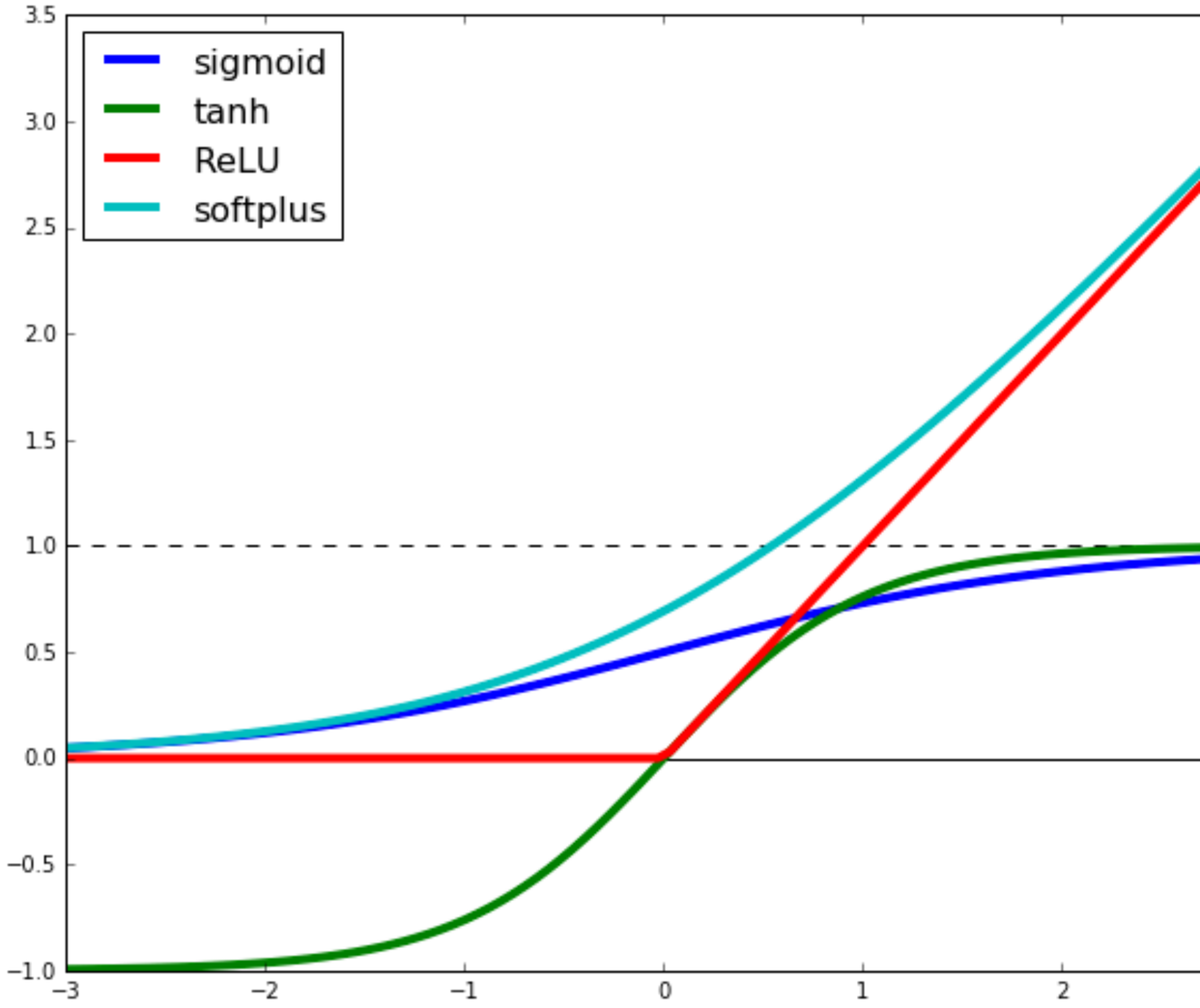
$$f(x) = \frac{1}{1+e^{-x}}$$

- ReLU - rectifier linear unit (biologically inspired)

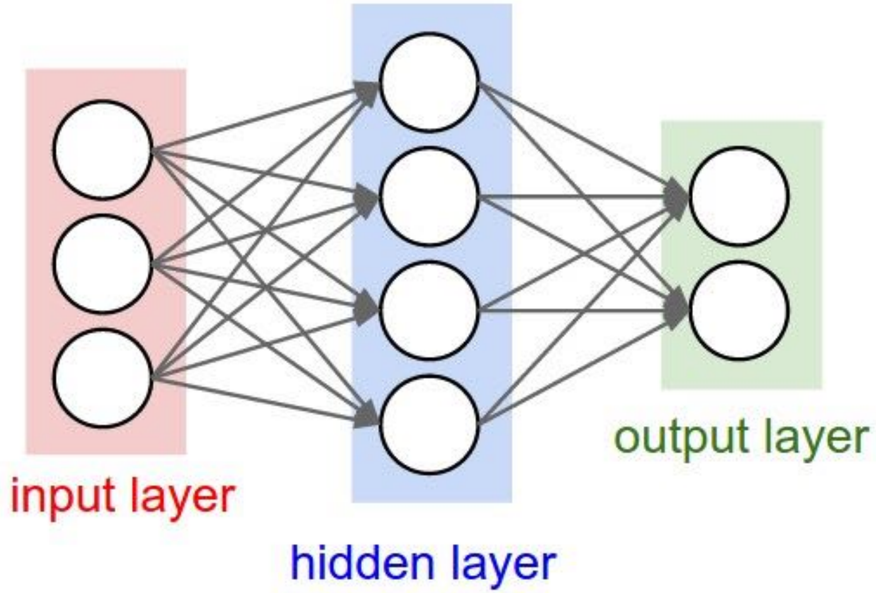
$$f(x) = \max(0, x)$$

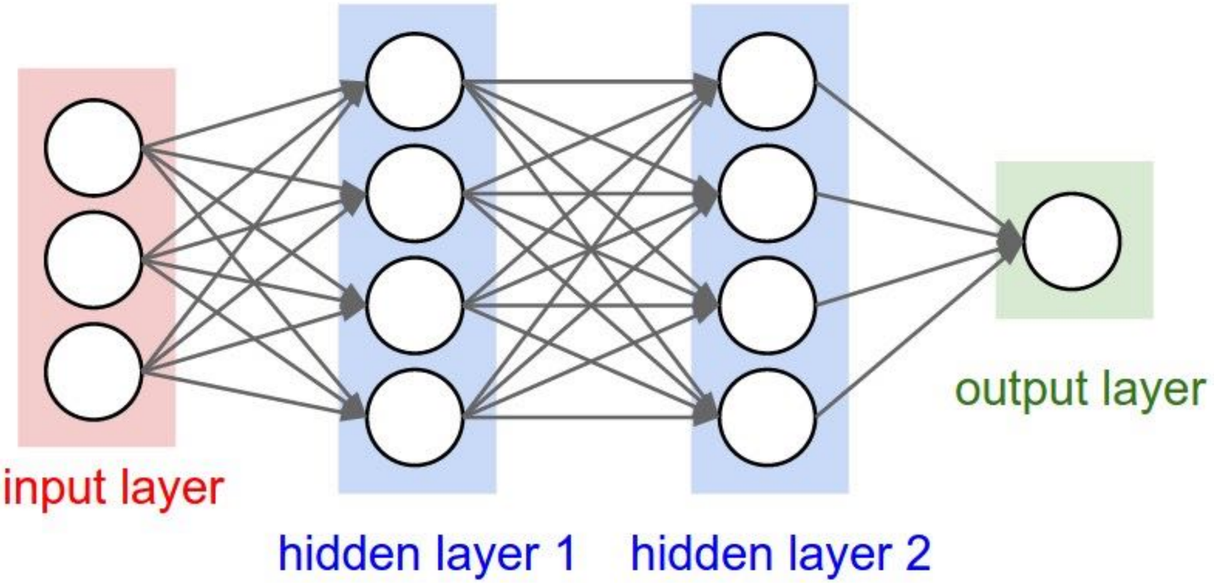
- Softplus (smoothed version of ReLU)

$$f(x) = \ln(1+e^{x})$$



Artificial Neural Network





How many parameters are there between hidden layer 1 and hidden layer 2? (right side)

Computations in neural network

input:

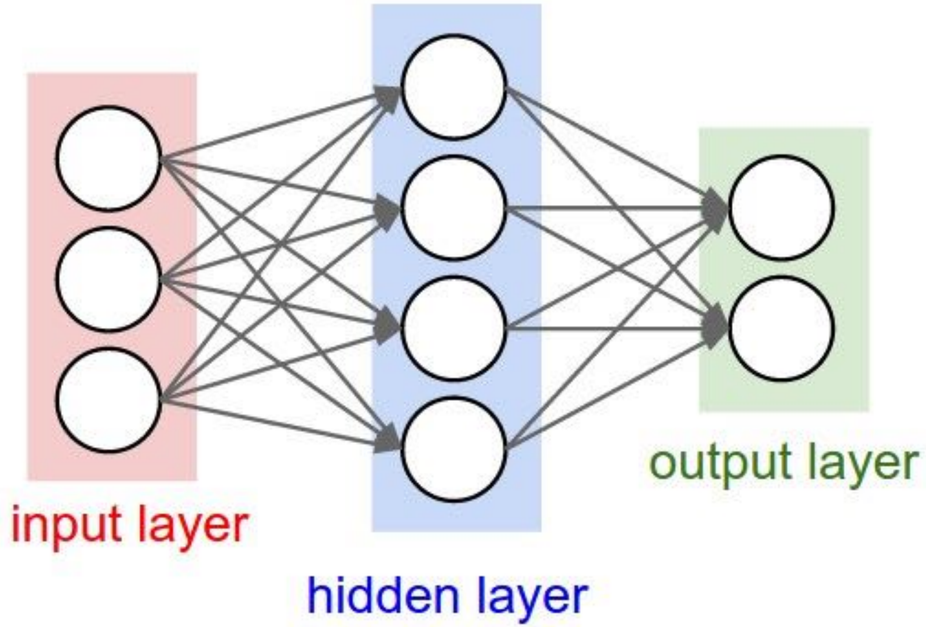
$$(x_1, x_2, \ldots, x_d)$$

Activations of hidden layer:

$$h_i = f\left(\sum_{j=1}^{n_{inputs}} W_{ij} x_j + b_i\right)$$

Activations of output layer:

$$o_i = f\left(\sum_{j=1}^{n_{hidden}} \tilde{W}_{ij} h_j + \tilde{b}_i\right)$$

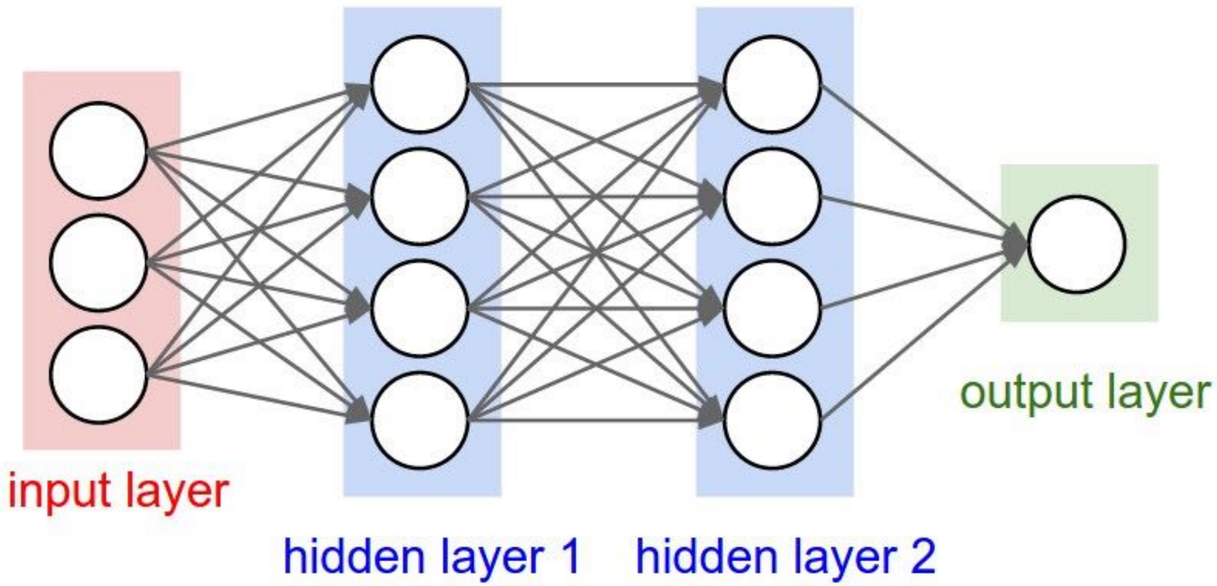


$$\begin{cases} \boldsymbol{h}_{linear} = W\boldsymbol{x} + \boldsymbol{b} \\ \boldsymbol{h} = f(\boldsymbol{h}_{linear}) \\ \\ \boldsymbol{o}_{linear} = \tilde{W}\boldsymbol{h} + \tilde{\boldsymbol{b}} \\ \boldsymbol{o} = f(\boldsymbol{o}_{linear}) \end{cases}$$

Several layers (Multi Layer Perceptron)

in matrix form, ***n*** hidden layers:

$$x_1 = f(W_0 x_0 + b_0)$$
$$x_2 = f(W_1 x_1 + b_1)$$
$$x_3 = f(W_2 x_2 + b_2)$$
$$\dots$$
$$\hat{y} = f(W_n x_n + b_n) - output$$

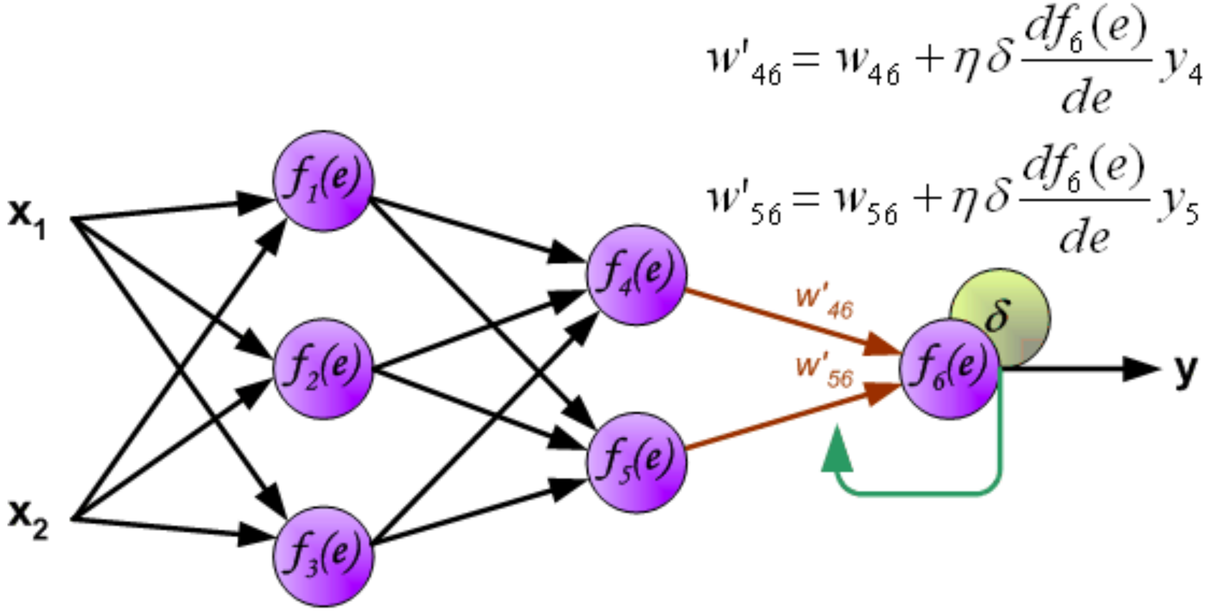


should compare with ***y*** - true labels corresponding to ***X***.

Training Neural Net

Mathematically, neural network is a function with many parameters, so we use smooth optimization. Output of network can be a single value (in case of one neuron) or several.

The only thing needed is to have some smooth loss function for optimization



$$l(y, \hat{y}) = \sum_i (\hat{y}_i - y_i)^2$$

$$l(y, \hat{y}) = -\sum_{i=0}^{N} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Gradient Descent

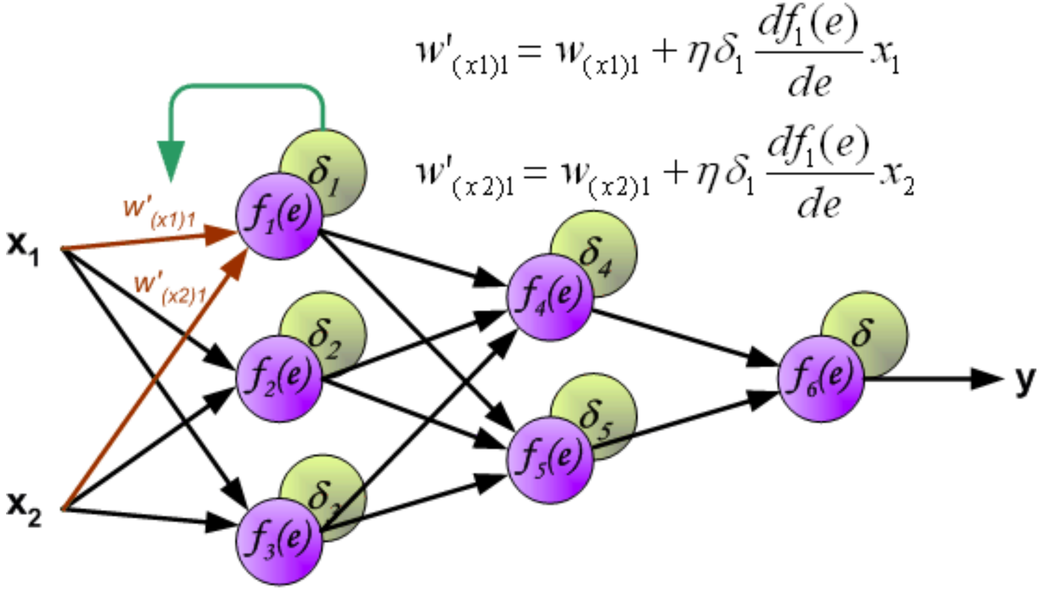
Problem: find w to minimize l.

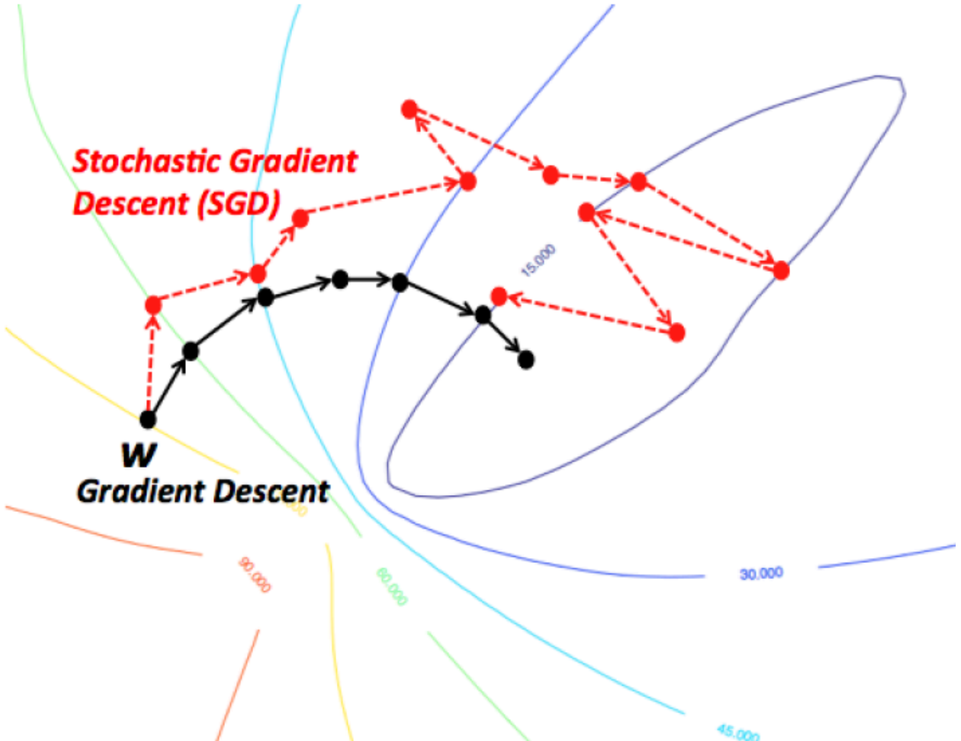
Solution: use gradient descent

(repeat until convergence.)

$$w \leftarrow w - \eta \frac{\partial l}{\partial w}$$

η is a step size
(also called *shrinkage*, *learning rate*)





Other kinds of optimizers

Stochastic gradient descent:

› calculate error using subsample of training set

Stochastic gradient descent with momentum:

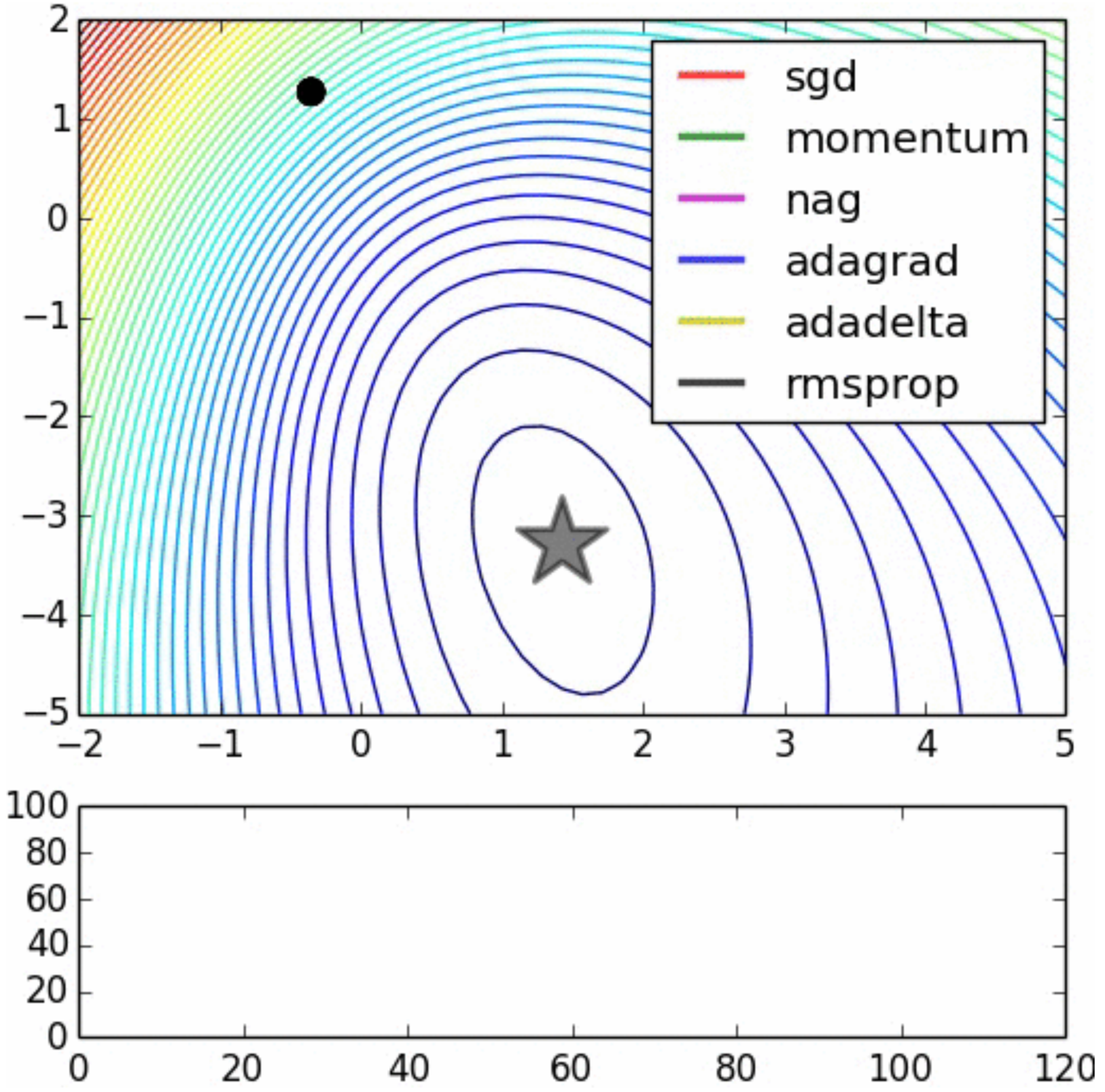
› moves towards "overall gradient direction", not just current gradient

AdaGrad:

› decreases learning rate individually for each parameter in proportion to sum of it's gradients so far

RMSProp:

› makes sure all gradient steps have ~ same magnitude (by keeping moving average of step magnitude)


sgd
momentum
nag
adagrad
adadelta
rmsprop
2
1
0
−1
−2
−3
−4
−5
−2
−1
0
1
2
3
4
5
100
80
60
40
20
0
0
20
40
60
80
100
120

Regularizations

Many parameters → overfitting

To avoid too complicated models regularisation terms are added to plain loss function:

$$L_1 = ||W||_1, \qquad L_2 = ||W||_2^2,$$

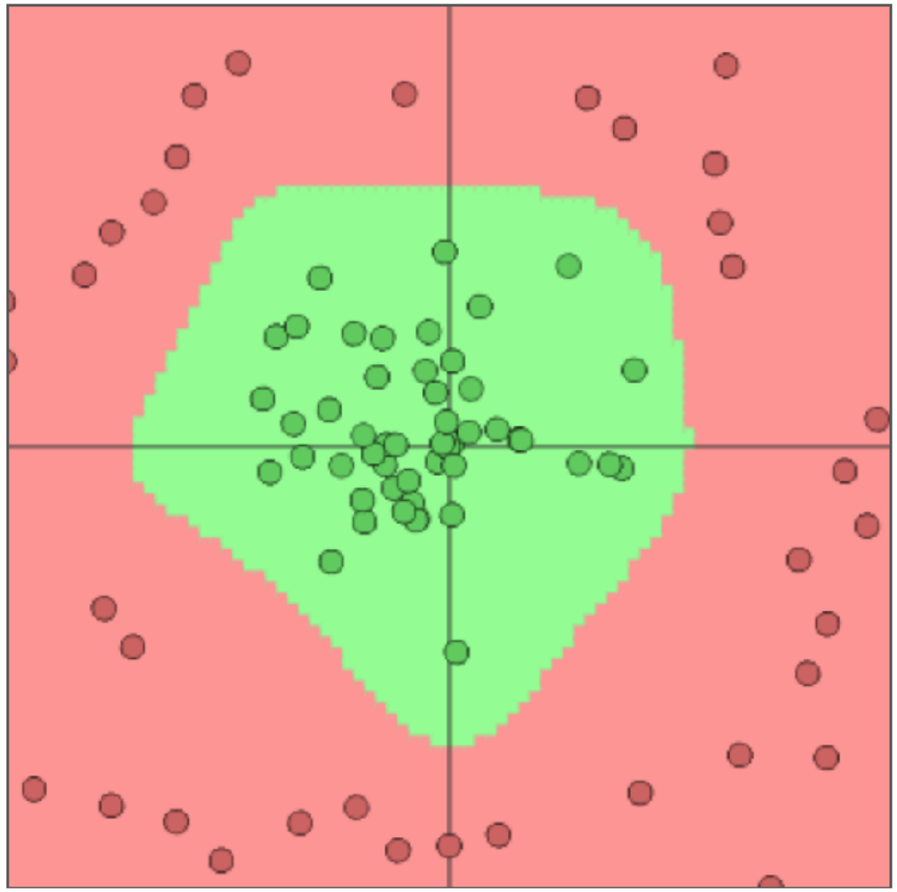
L1 - leads to sparse weight matrix, but is not differentiable at 0

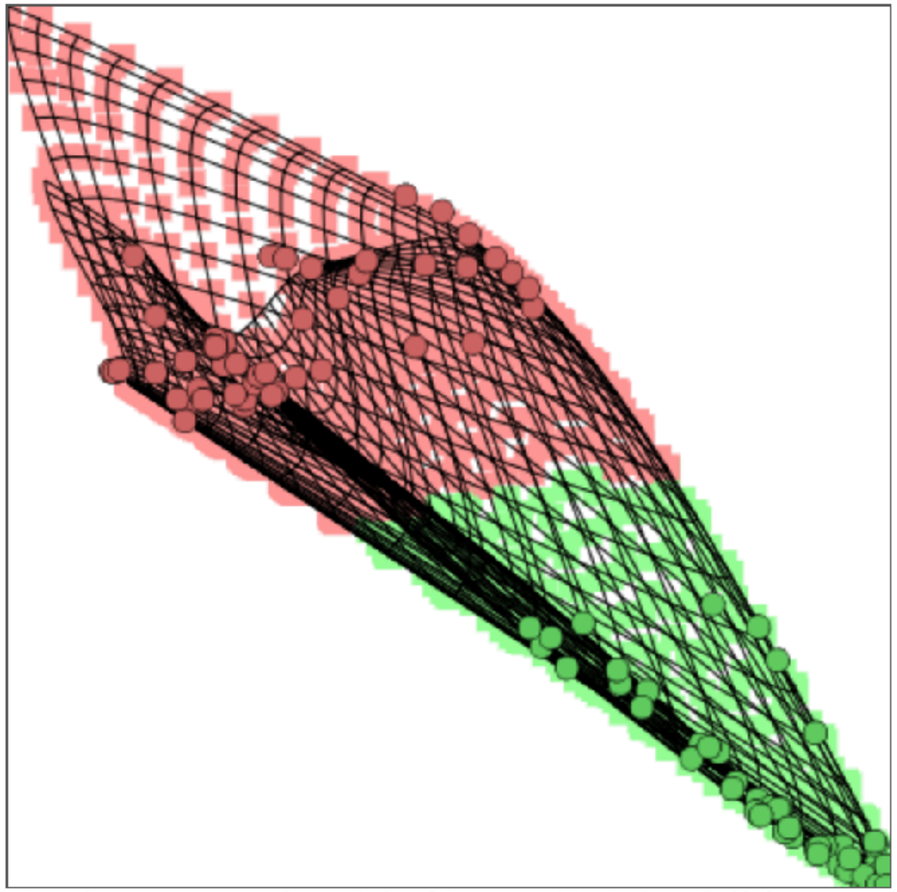
L2 - gives lower errors

Playing with Neural Nets in a browser

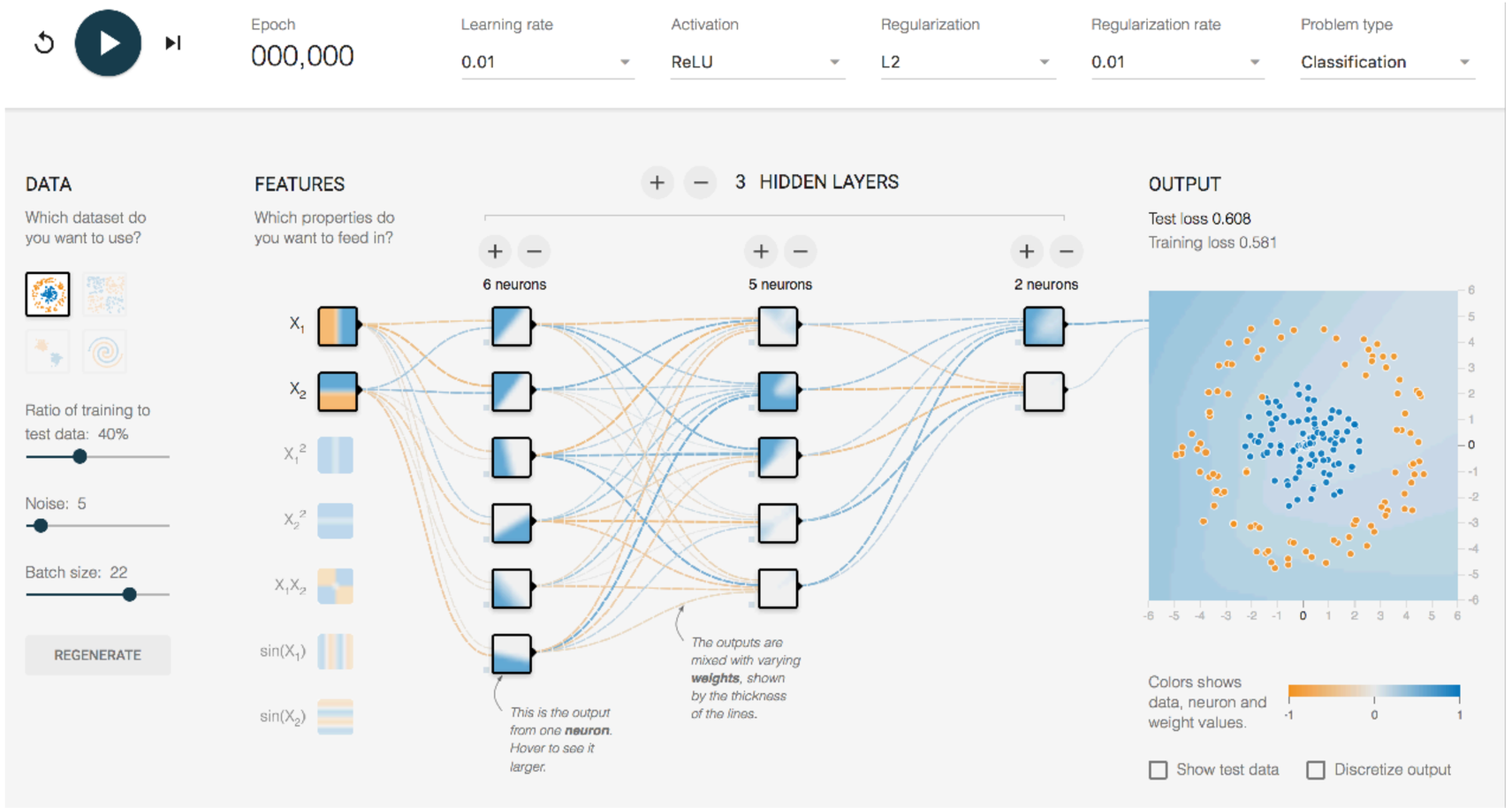
Convnet.js:

- layer_defs
  layer_defs
  layer_defs
  layer_defs
  layer_defs
  net = new
  net.make
  trainer = r
  momentu





One more playground



http://playground.tensorflow.org/

Convolutional Neural Networks

Regular Machine Learning (ML) approach

In Physics:

Describe data by variables => ML => signal vs bckg

In a search engine:

Describe web pages by features => ML => estimation of relevance to a query

This approach works well enough in many areas.
Features can be either computed manually or also be estimated by ML (*stacking*).

When it didn't work (good enough)?

- which features help to identify a drum on the picture?
  or a car?
  or an animal?
- which features are good to detect a particular phoneme in a soundform?

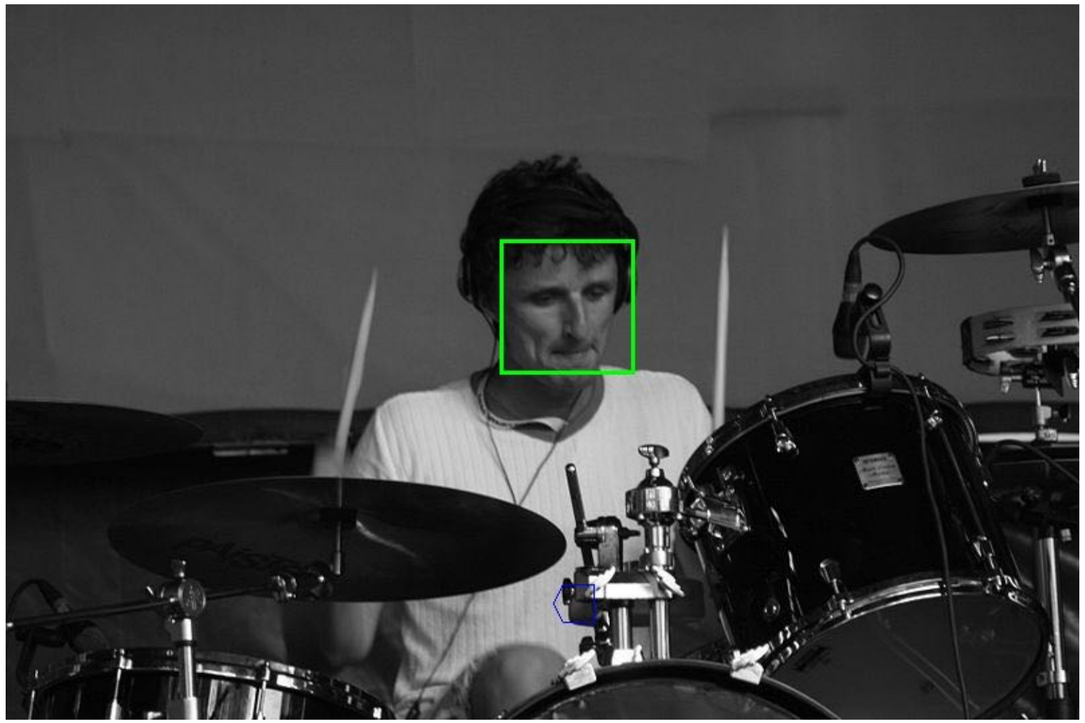
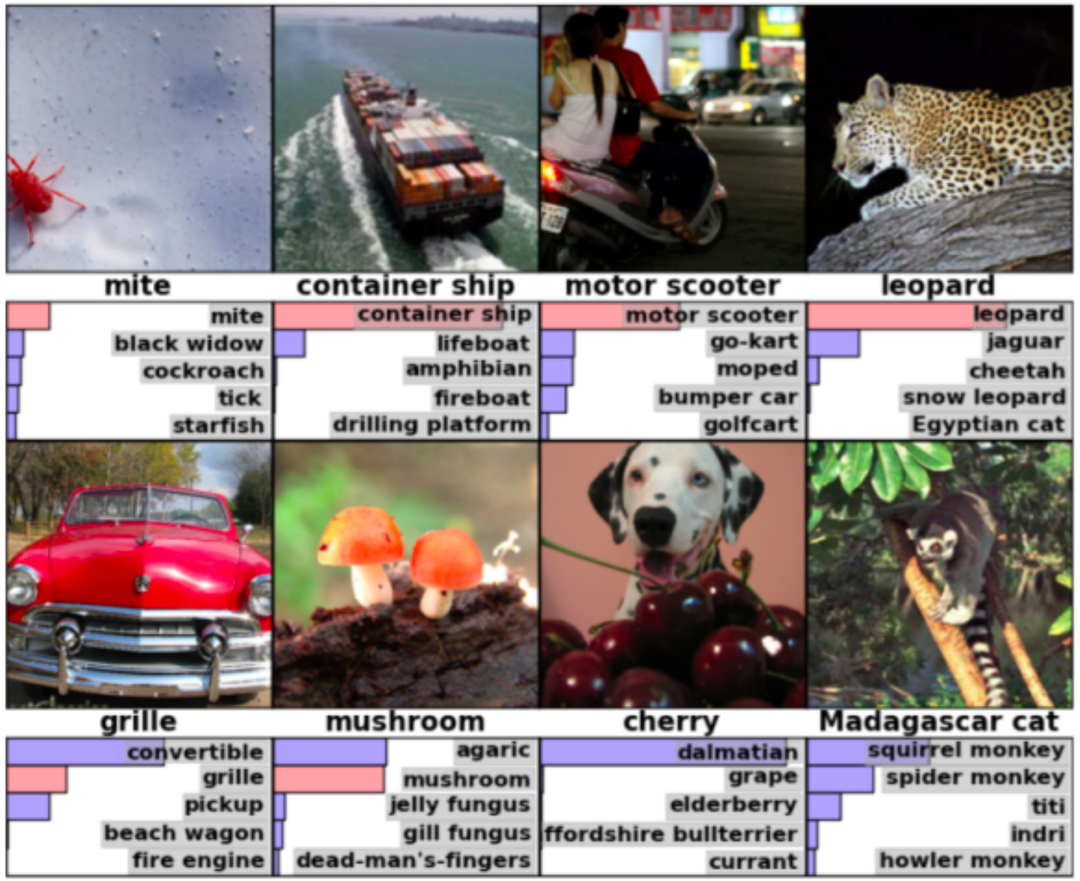


Image recognition today

can classify objects on images with very high accuracy even if those are images of 500 x 500 and we have only 400 output neurones, that's at least $500 * 500 * 400 = 10^8$ parameters

and that's only linear model!

1. can we reduce the number of parameters?
2. is there something we're missing about image data when treating picture as a vector?



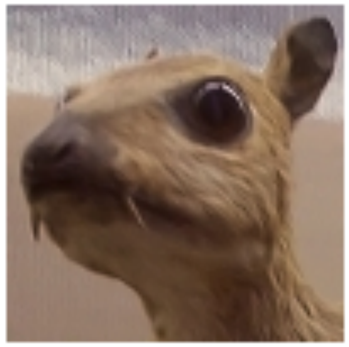
Convolution idea

Convolution is a mathematical operation which describes a rule of how to mix two functions or pieces of information:

- input data and
- the convolution kernel mix together to form

a transformed feature map

Input image



Convolution Kernel

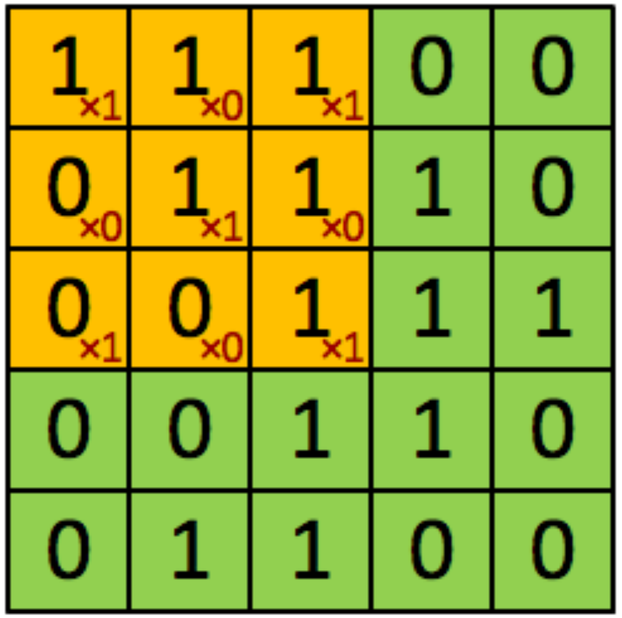
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

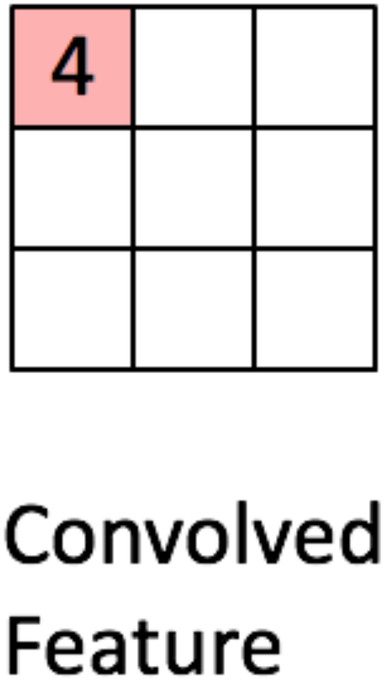
Feature map

Convolution in steps

Calculating convolution by sliding image patches over the entire image.

One image patch (yellow) of the original image (green) is multiplied by the kernel (red numbers in the yellow patch), and its sum is written to one feature map pixel (red cell in convolved feature).

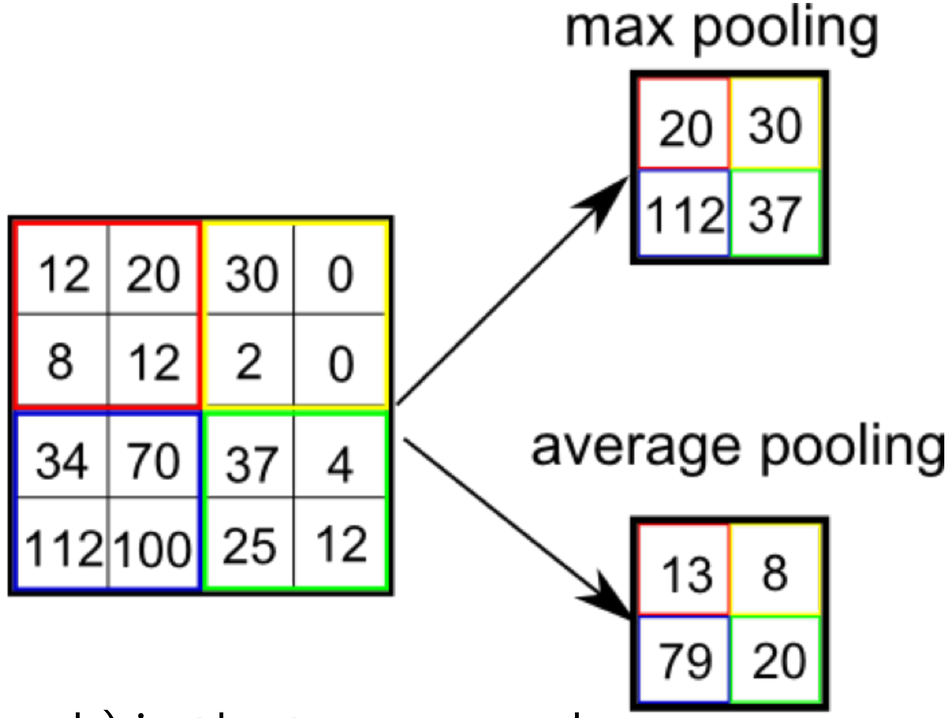




Pooling

Motivation:

- Reduce layer size by a factor
- Make NN less sensitive to small image shifts



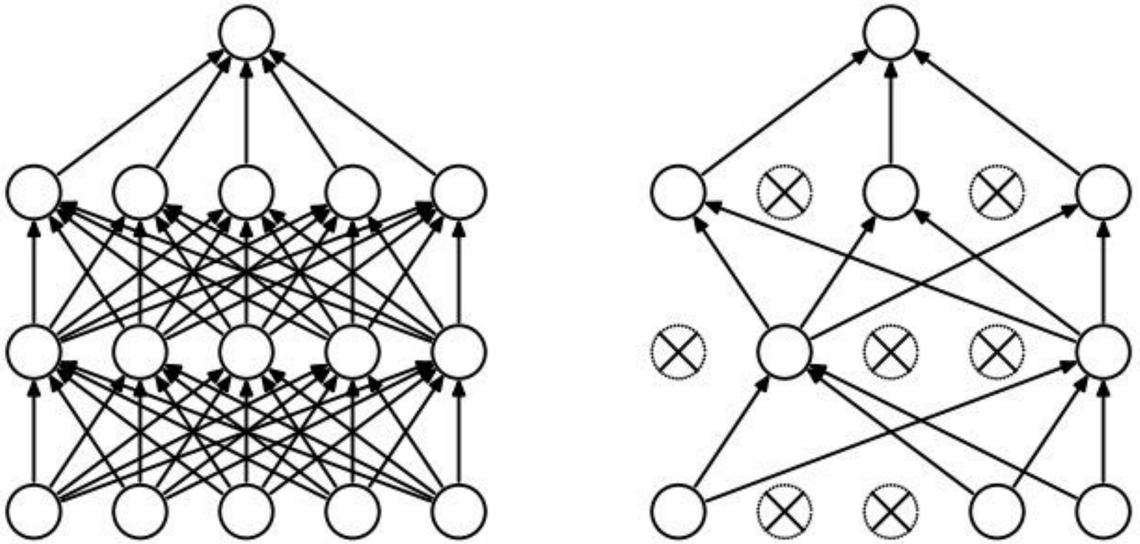
The intuition behind this operation (layer in the network) is that once we know that a specific feature is in the original input volume (there will be a high activation value), its exact location is not as important as its relative location to the other features.

Example: we have a filter for detecting faces. The exact pixel location of the face is less relevant then the fact that there is a face "somewhere at the top"

Regularizations redux

Many parameters → overfitting

L1 , L2 — regularizations are still used



(a) Standard Neural Net (b) After applying dropout.

but this isn't enough!

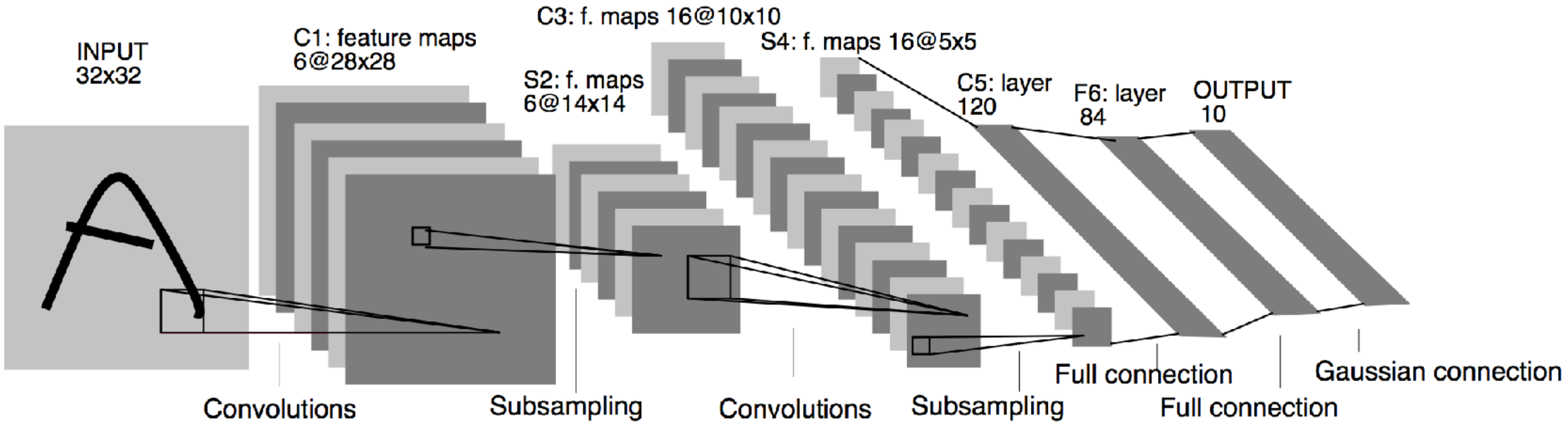
dropout: at each step of optimization some subset of neuron weights is zeroed randomly (dropped)

it prevents co-adaptation of neurons and makes the neural network more stable

Examples of Neural Network layers

- Input
- Convolution
- Pooling
- Fully connected (FC) layers
  Basically, a FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.
- Dropout

Yann LeCun Network (LeNet)



LeNet-5 based on CNNs — was developed in 1998, and how it works.

Quiz: 1) what is the convolution size at the 1st step?
2) what is pooling window size at the 2nd step?

Problems with deep architectures

In deep architectures, the gradient varies between parameters drastically (10^3 – 10^6 times difference and this is not a limit).

This problem usually referred to as 'gradient diminishing'.

Adaptive versions of SGD are used today.

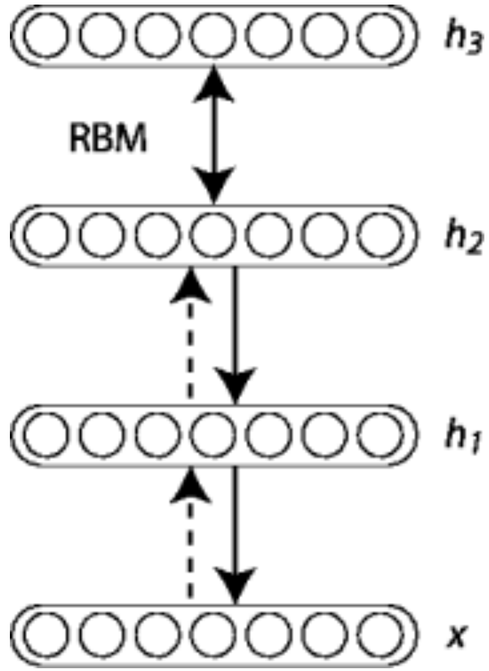
Deep learning original idea (Hinton, 2006)

Deep Belief Network (Hinton et al, 2006) was one of the first ways to train deep structures.

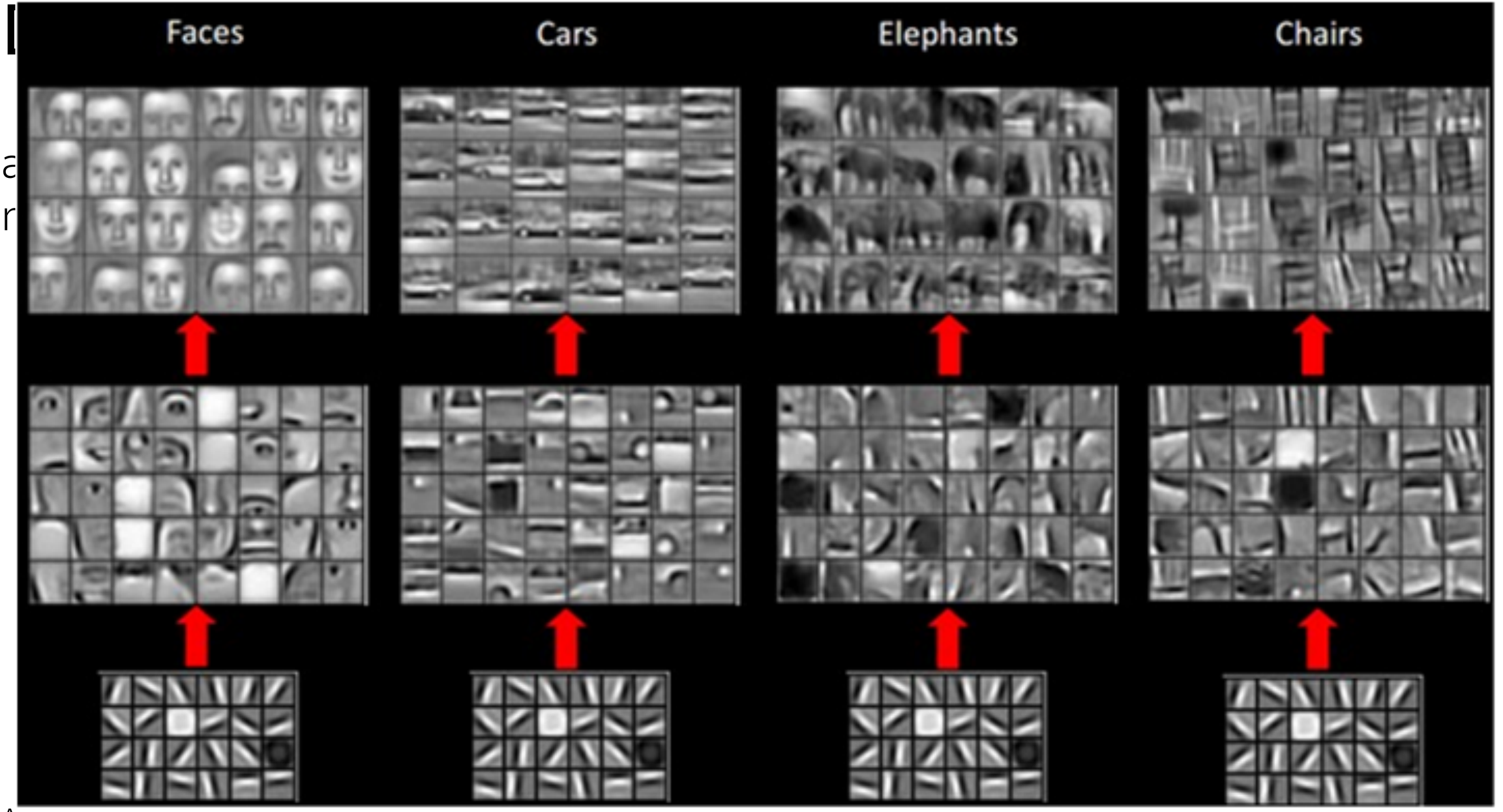
- trained in an unsupervised manner layer-by-layer
- next layer should "find useful dependencies" in previous layer output
- finally, a classification done using output of last layer (optionally, minor fine-tuning of the network might also required)

Why this is important?

- we can recognize objects and learn patterns without supervision gradient
- backpropagation in the brain??



Practical? No, today deep learning relies mostly on stochastic gradient optimization .


Faces
Cars
Elephants
Chairs

Inference of high-level information

Ability of neural networks to extract high-level features can be demonstrated with artistic style transfer.

Image is smoothly modified to preserve the activations on some convolutional layer.

Sketch to picture (texturing)

Online demo to transform a doodle (sketch) into Claude Monet or Vincent Van Gogh:

http://likemo.net/#sketch_canvas

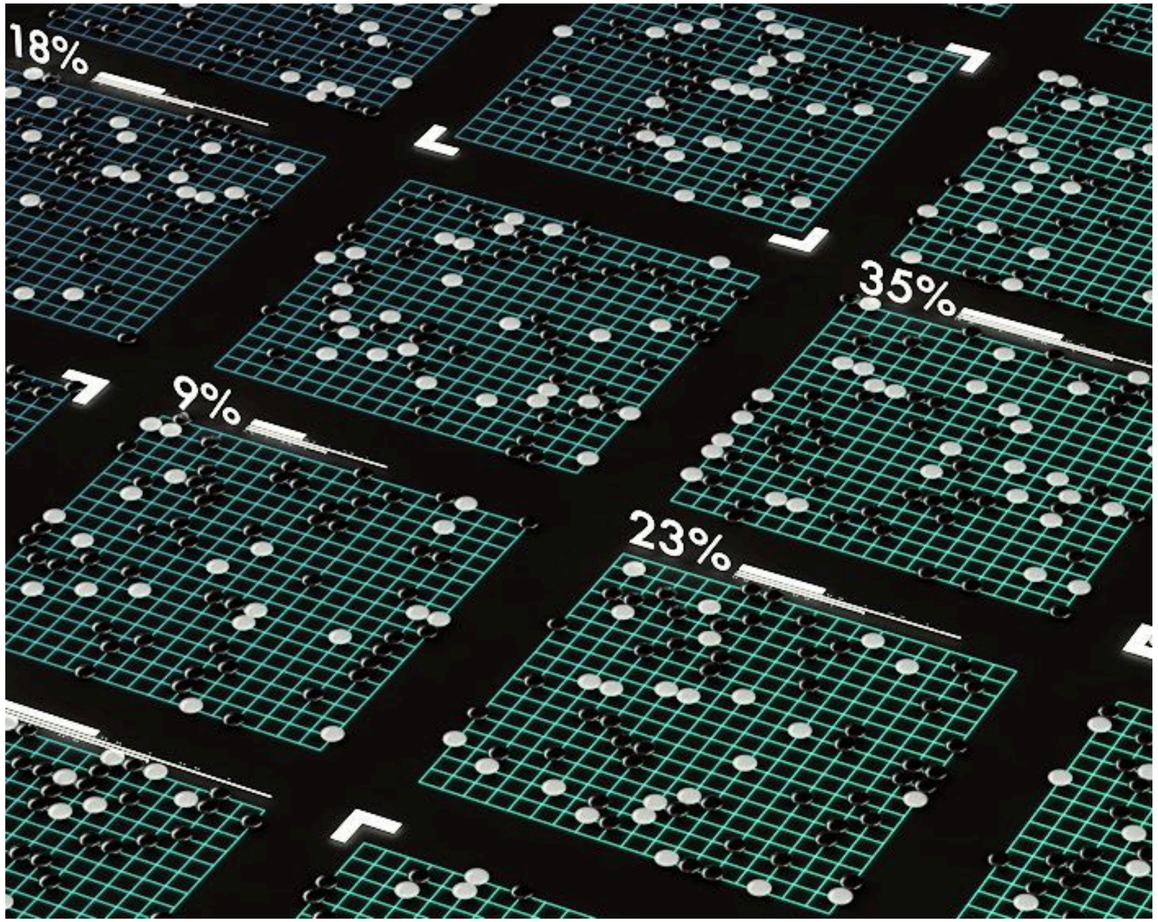
AlphaGo

- too many combinations to check
- no reliable way to assess the quality of position

AlphaGo has 2 CNNs:

1. network predicts probability to win
2. policy network predicts next step of a player

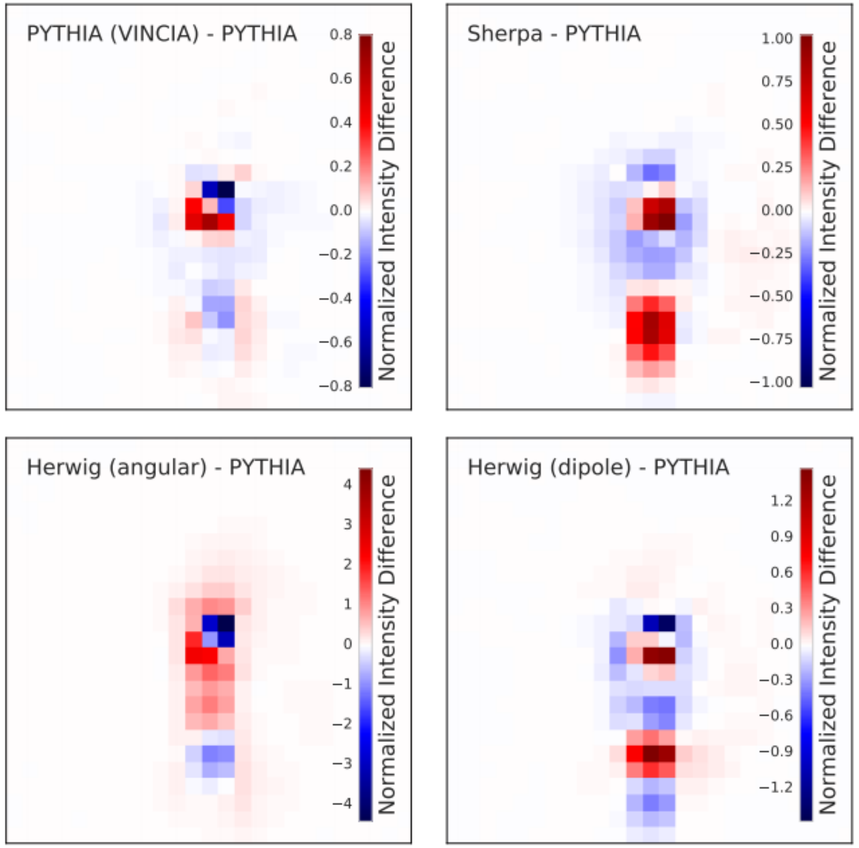
both are given the board (with additional features for each position on the board)



Pitfalls of blind machine learning

CNNs were applied to jet substructure analyses and shown promising results on simulated data, but a study shows model applied to different MC generators images has up to 2x background efficiency.

- thus, learnt dependencies are MC-specific and may turn out to be useless,
- machine learning may use dependencies there are approaches to domain adaptation, that might come handy.

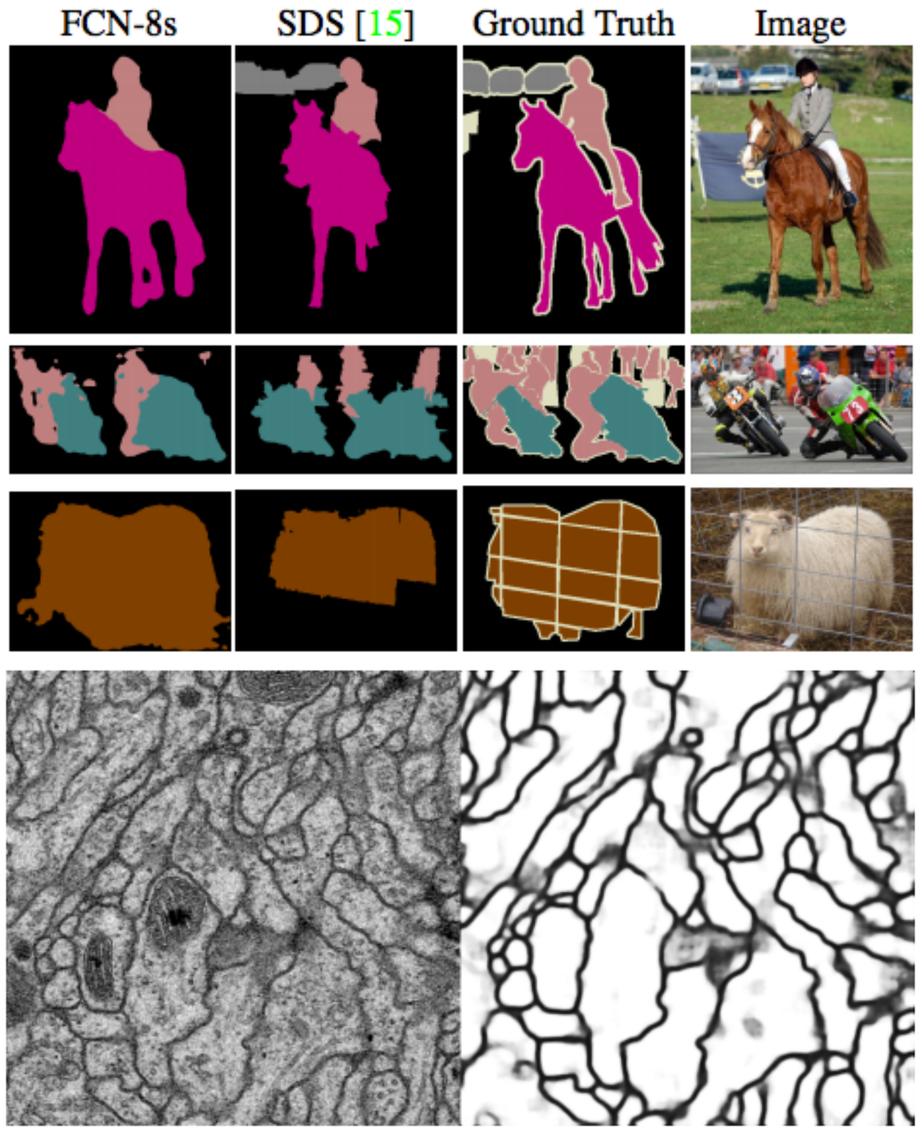


Applications of CNNs

Convolutional neural networks are top-performing on the image and image-like data:

- image recognition and annotation
- object detection
- image segmentation
- segmentation of neuronal membranes

Next big thing: attention control networks

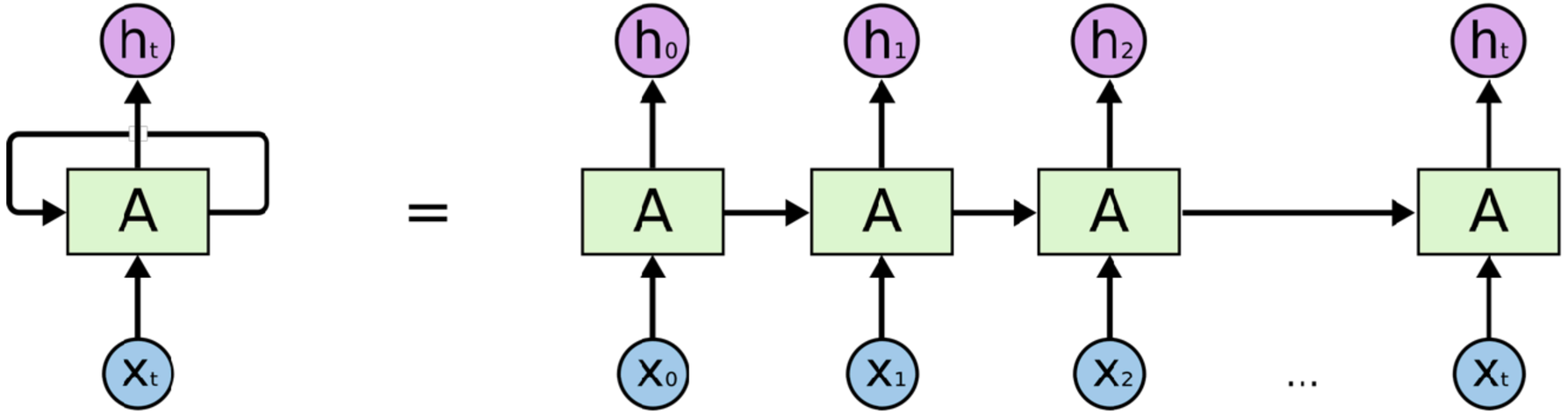


Recurrent Neural Networks

Sequences and Recurrent NN

> sound, text
> any information that is time dependent (daily revenues, stocks prices…)

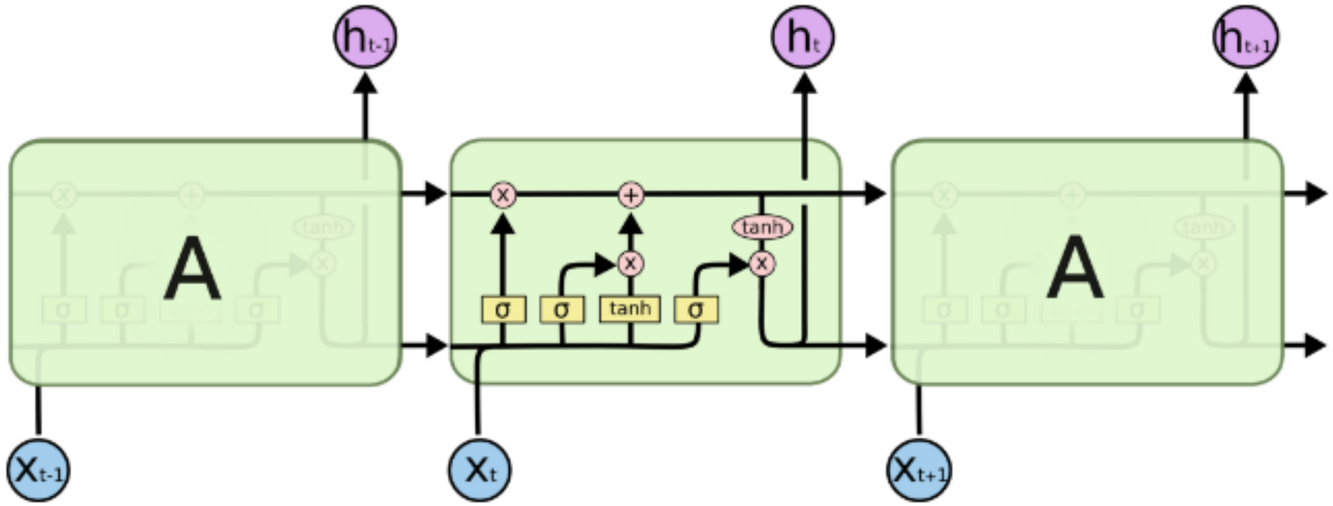
Inspiration: brain contains information about past and is able to process the information given in sequences.



LSTM: a popular recurrent unit

LSTMs have chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

See post by Christopher Olah for details.



RNN for fun, baby names generation

feed the RNN a bunch of 8000 baby names and ask to generate new ones:

Rudi Levette Berice Lussa Hany Mareanne Chrestina Carissy Marylen Hammine Janye Marlise Jacacrie Hendred Romand Charienna Nenotto Ette Dorane Wallen Marly Darine Salina Elvyn Ersia Maralena **Minoria** Ellia Charmin Nerille Chelon Walmor Jeryly Stachon Allisa Anatha Cathanie Geetra Alexie Jerin Cassen Herbett Cossie Velen Daurenge **Robester** Shermond Terisa Licia Roselen Ferine Jayn Lusine Charyanne **Sales** Wallon Martine Merus Jelen Candica Wallin Tel Rachene Tarine Ozila Ketia Shanne Arnande Karella Roselina Alessia Chasty Deland Berther Geamar Jackein Mellisand Sagdy Nenc Lessie Guen Gavi Milea Anneda Margoris Janin Rodelin ZeElyne Janah Ferzina Pey **Castina, Killie, Saddie, R**

RNN for fun, math text generation

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{étale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

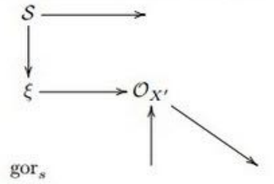
be a morphism of algebraic spaces over S and Y.

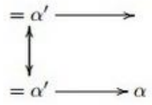
Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram





$\mathrm{Spec}(K_\psi)$ Mor_{Sets} $\mathrm{d}(\mathcal{O}_{\mathcal{X}_{X/k}}, \mathcal{G})$

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $f_\star$. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{étale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

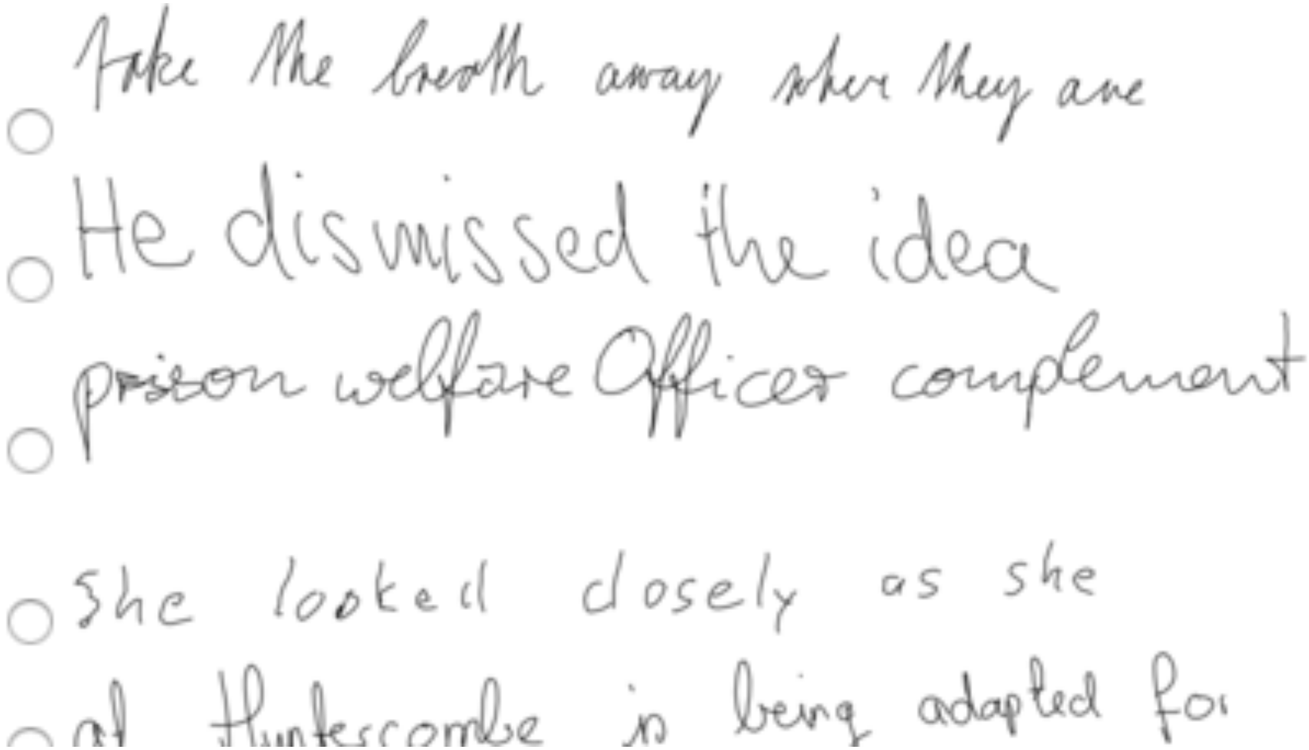
is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

RNN for fun, (Linux) source code generation

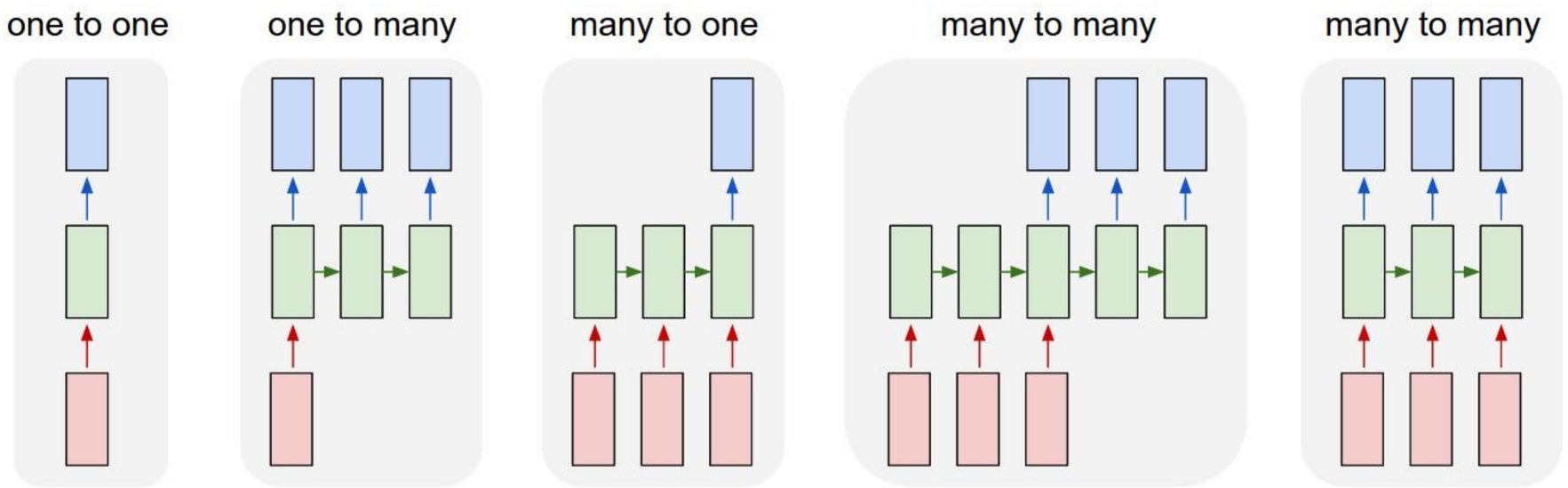
```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

Style in RNNs



Position of the pen in the next moment is predicted (online demo).

Different strategies for prediction

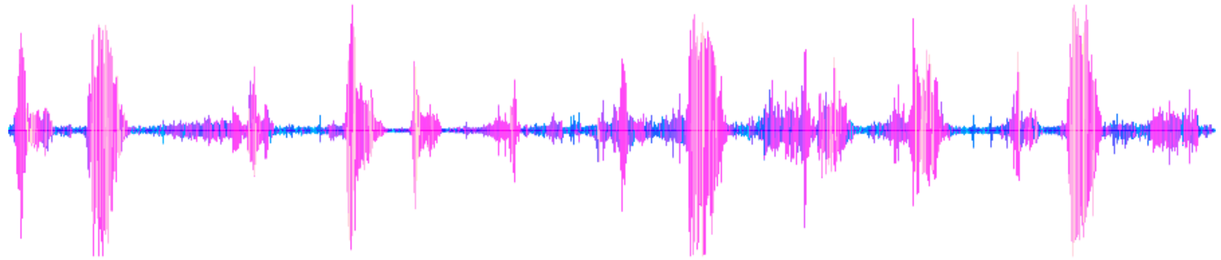


one can predict

> a single observation from sequence or
> map sequence to sequence

Applications of Recurrent NNs

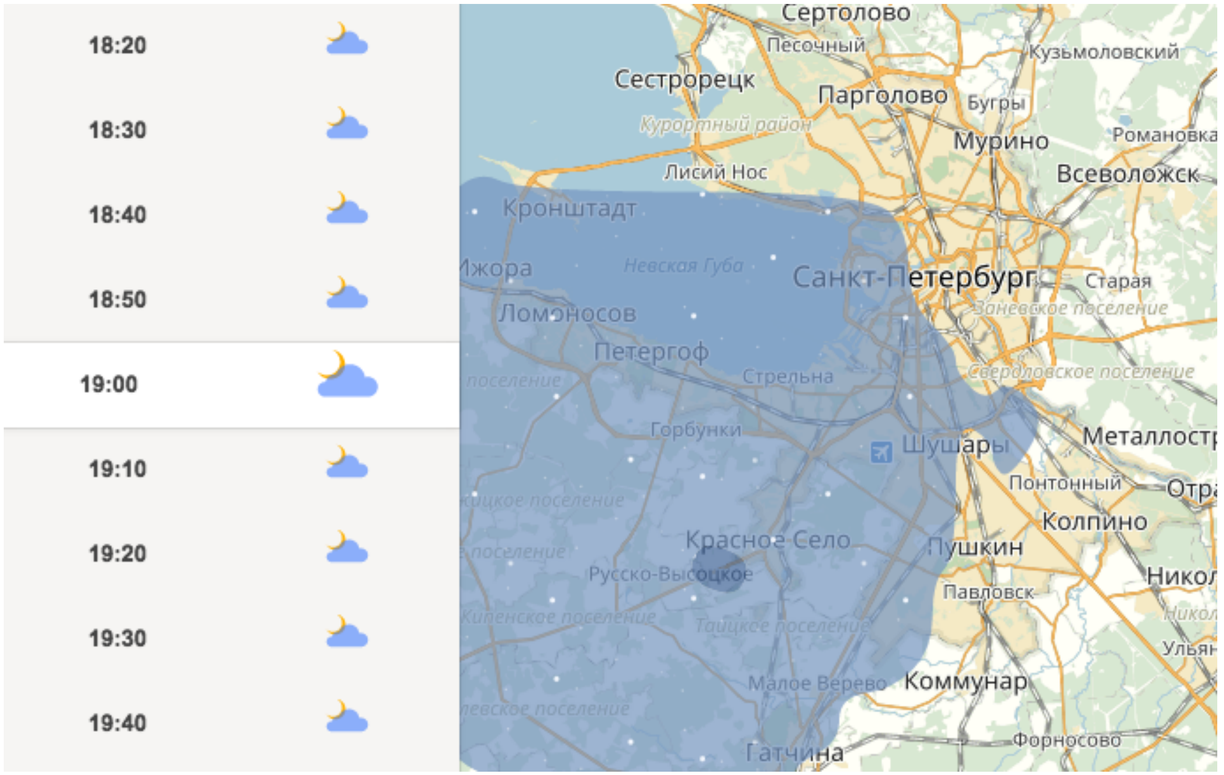
- natural language processing /translation
  speech recognition
  - › sound record is a sequence!



- speech synthesis
  demand forecasting in retail systems /
  stock prices forecasting
- Jet substructure recognition, https://arxiv.org/abs/1702.00748

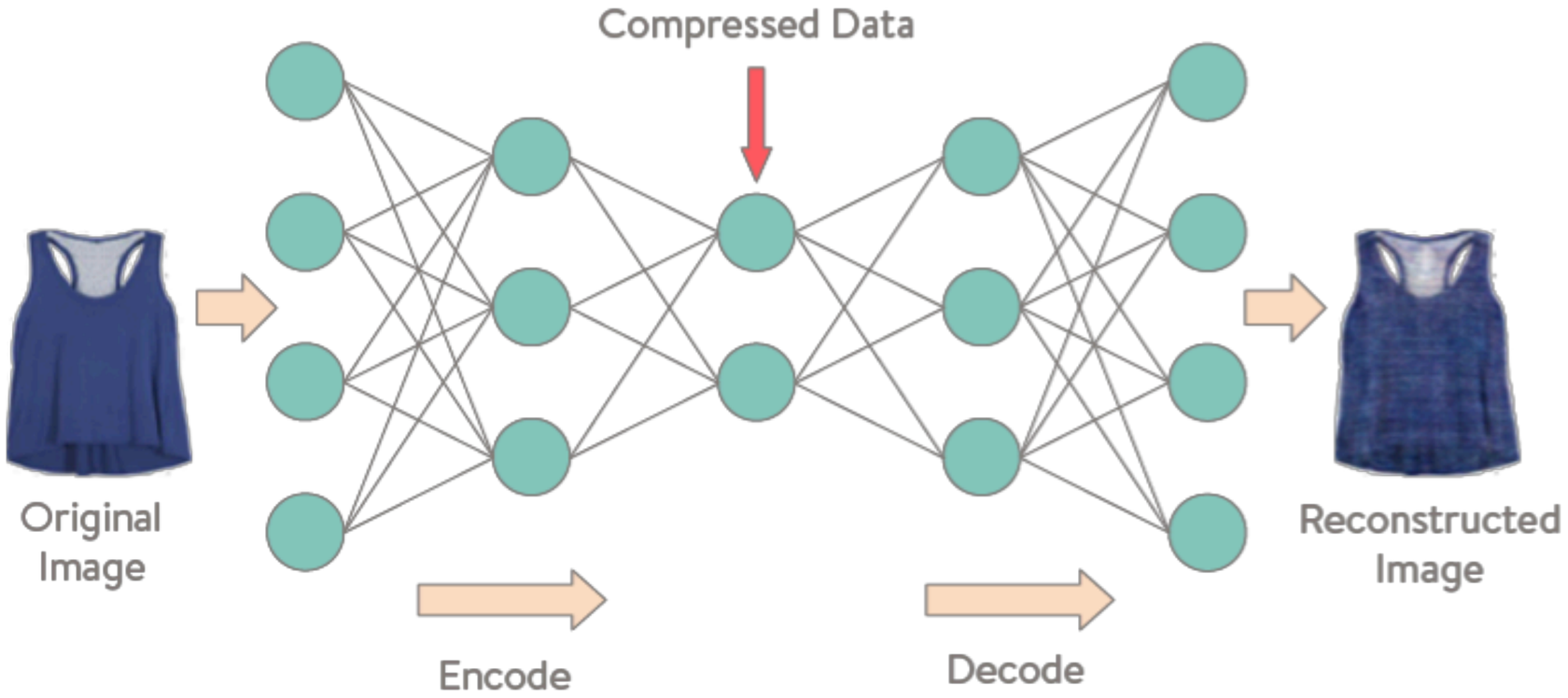
Mixing RNN and CNN

Sometimes you need both to analyze dependencies in space and time. For example, in weather forecasting a sequence of photos from satellites can be used to predict weather conditions, i.e. snow/rain (https://yandex.com/company/blog/77/)



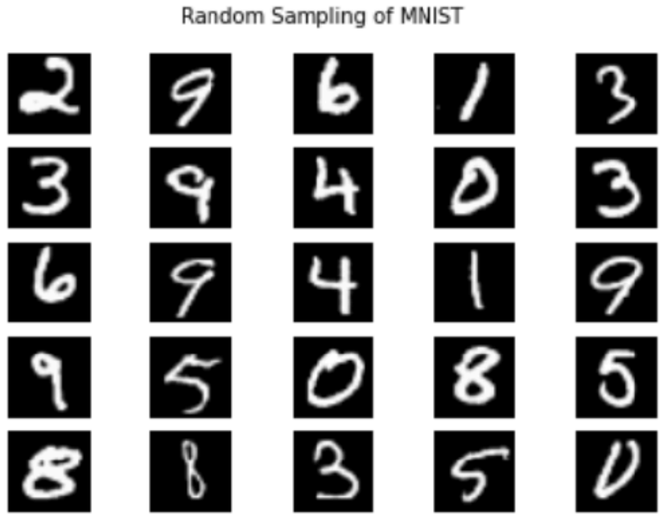
Other directions

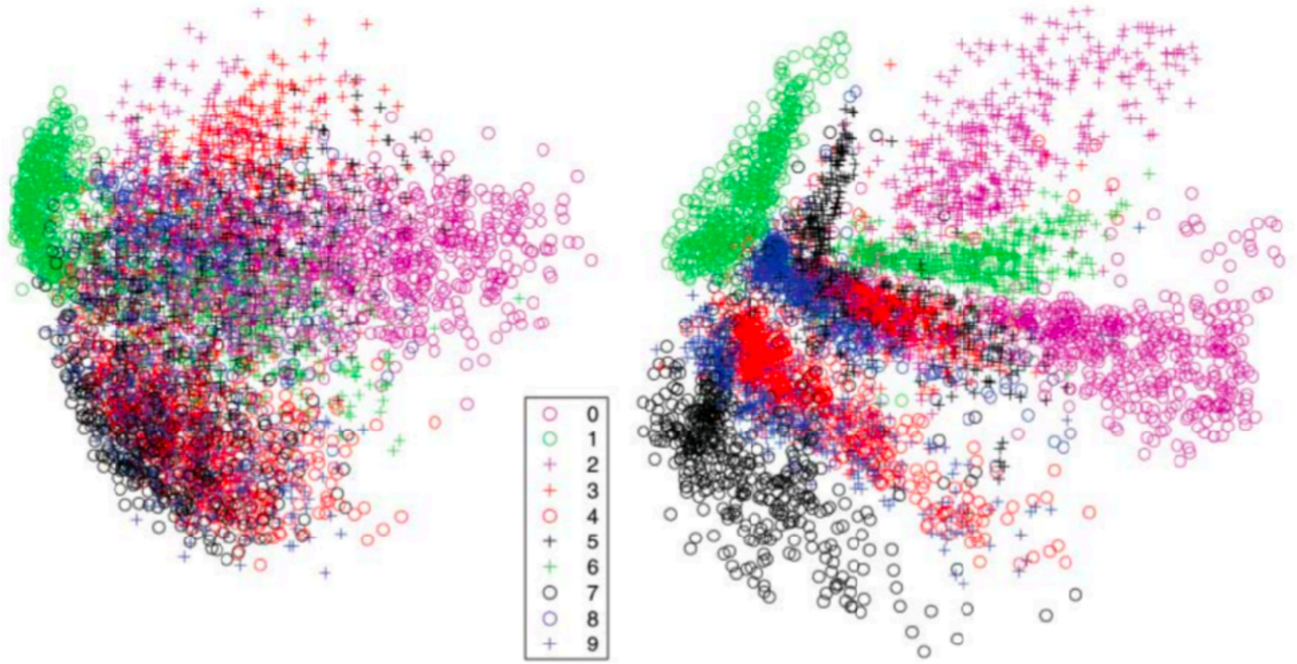
Autoencoding



compressed data is treated as a result of an algorithm
trained representations can be made stable against different noise

Autoencoding





Take MNIST dataset (left) and compare PCA (middle) and autoencoder with 2 neurons in the middle layer (right).

Interactive playground: https://transcranial.github.io/keras-js/#/

Generative Adversarial Networks

how can network learn to generate an image (of a bedroom)?
(https://github.com/Newmu/dcgan_code)

Generative Adversarial Networks

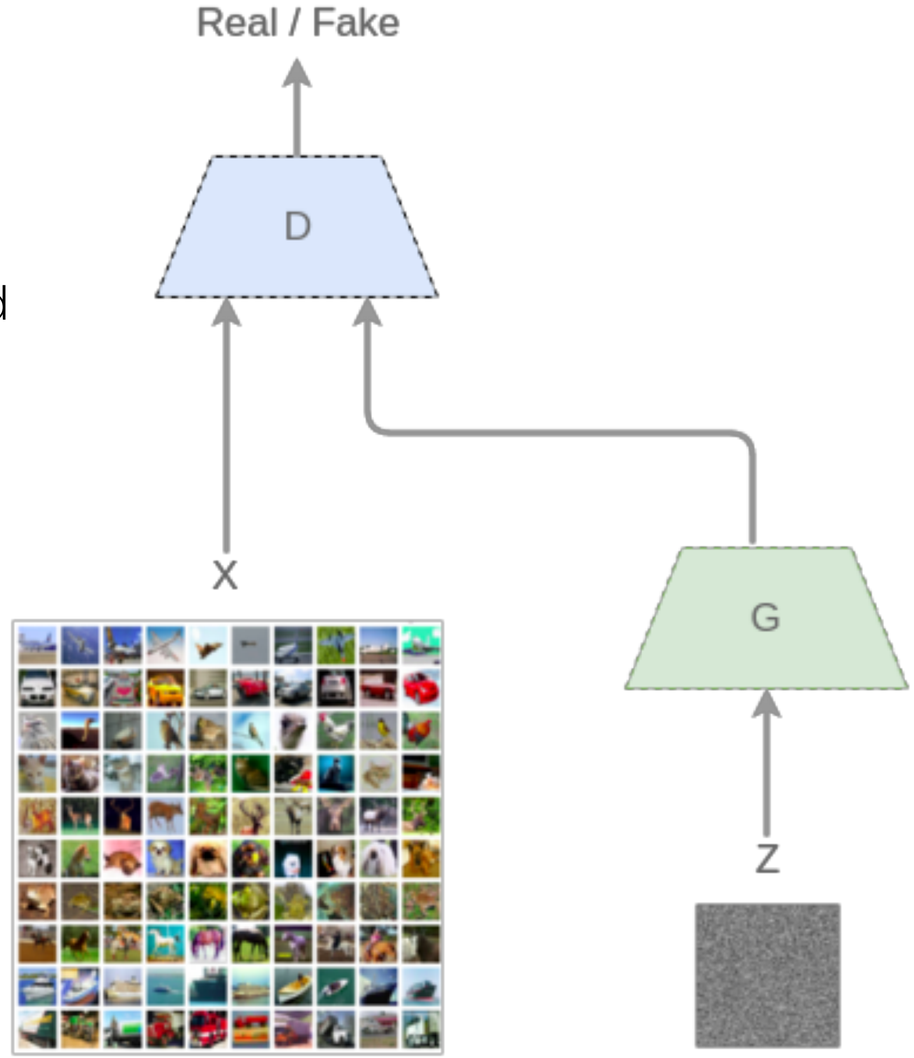
Goal: learn to generate realistic images
Solution: make **G**enerator play cooperative game with **D**iscriminator

- Generator (MLP) converts noise (Z) to image (S)
- Discriminator (MLP) distinguishes real images (X) from generated images (X)

Combined loss function:

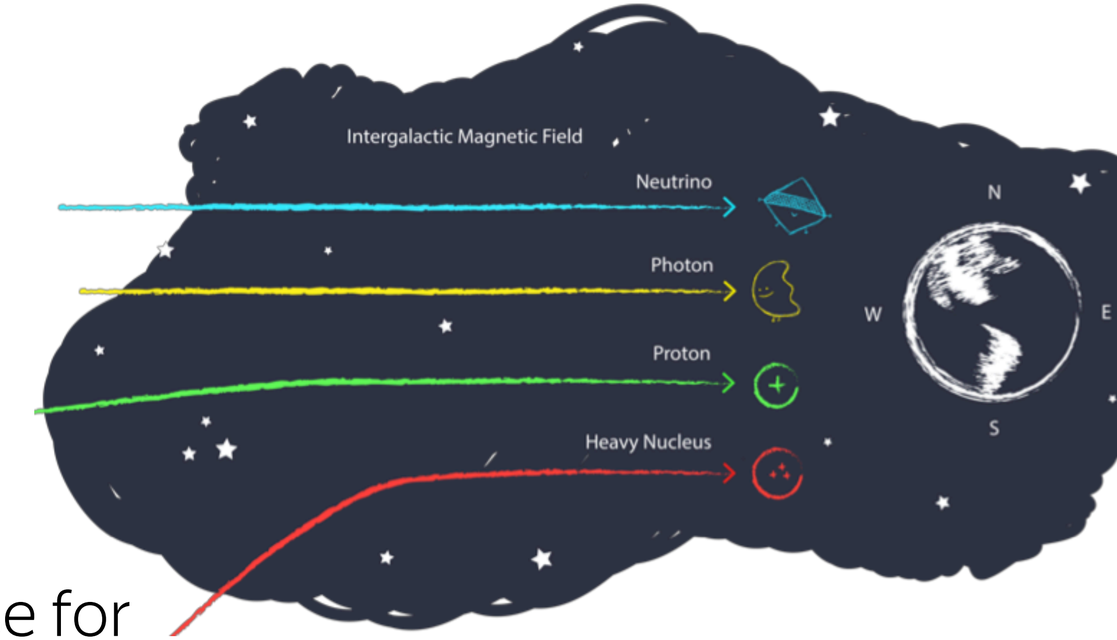
$$\mathcal{L}_{adv} = \underbrace{E[log(P(D(I)=0|I \in S))]}_{\substack{\text{term associated with discriminator} \\ \text{perceiving a generated sample as fake}}} + \underbrace{E[log(P(D(I)=1|I \in X))]}_{\substack{\text{term associated with discriminator} \\ \text{perceiving a real sample as real}}}$$

The game-theoretical basis for this framework ensures that if we extend the space of allowed functions that G and D can draw from to the space of all continuous functions, then there exists some G that exactly recovers f: Z -> X



Applications. Detecting cosmic rays

- cosmic rays coming from space have huge energies
- we don't know their source
- decay products of those can be detected
- need to cover large areas with detectors



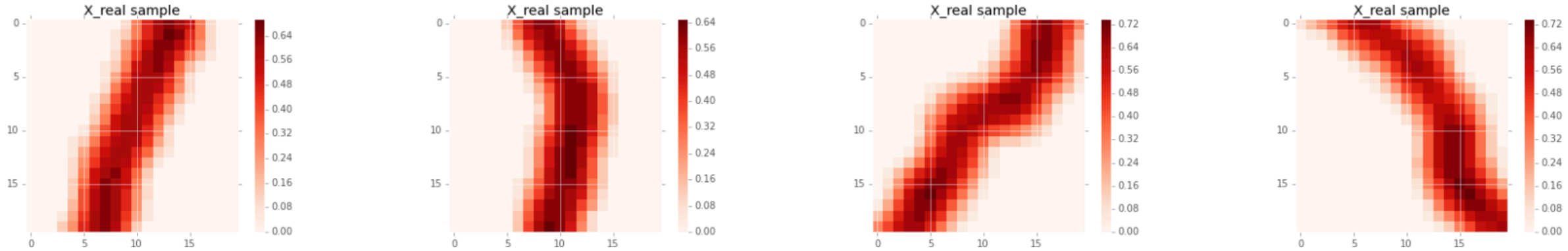
Collaboration: CRAYFIS, https://crayfis.io

Idea: use smartphone camera as a detector

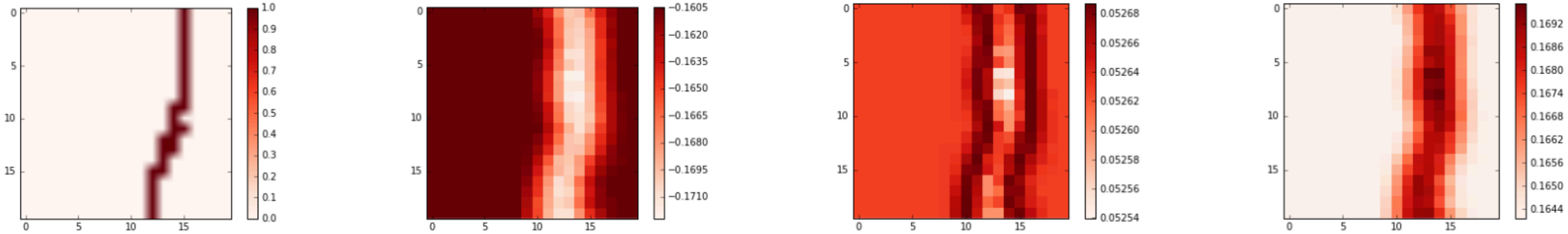
Problem: how to simulate well camera response for the particles?

Transforming adversarial networks

We can 'calibrate' the simulation with adversarial networks. That's **real data images**:



TAN input is not a noise, but output of a simulator. Here is G output by epoch:



More applications to HEP

https://arxiv.org/abs/1705.02355 - M. Paganinin et al, CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks

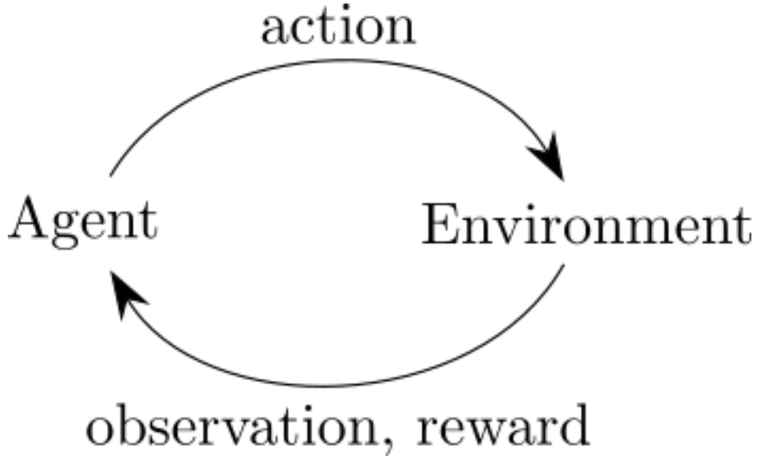
https://arxiv.org/abs/1701.05927 - L. de Oliveira et al, Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis

Reinforcement Learning

Reinforcement Learning (RL) is the branch of machine learning that is concerned with making sequences of decisions. It assumes that there is an ***agent*** that is situated in an ***environment***.

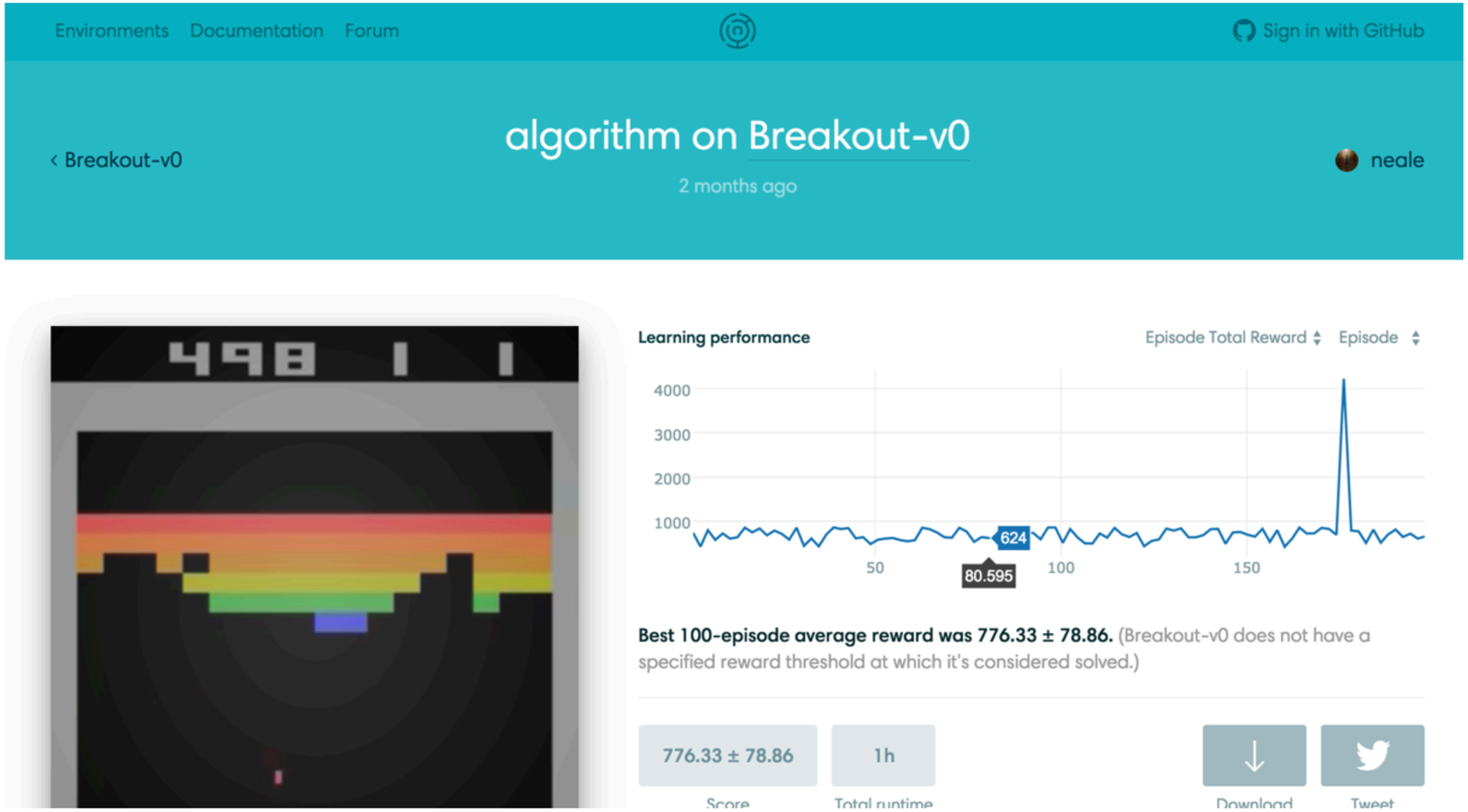
At each step, the agent takes an ***action***, and it receives an ***observation*** and ***reward*** from the environment.

An RL algorithm seeks to maximize the agent's total reward, given a previously unknown environment, through a learning process that usually involves lots of trial and error.



Platform, Typical Problems

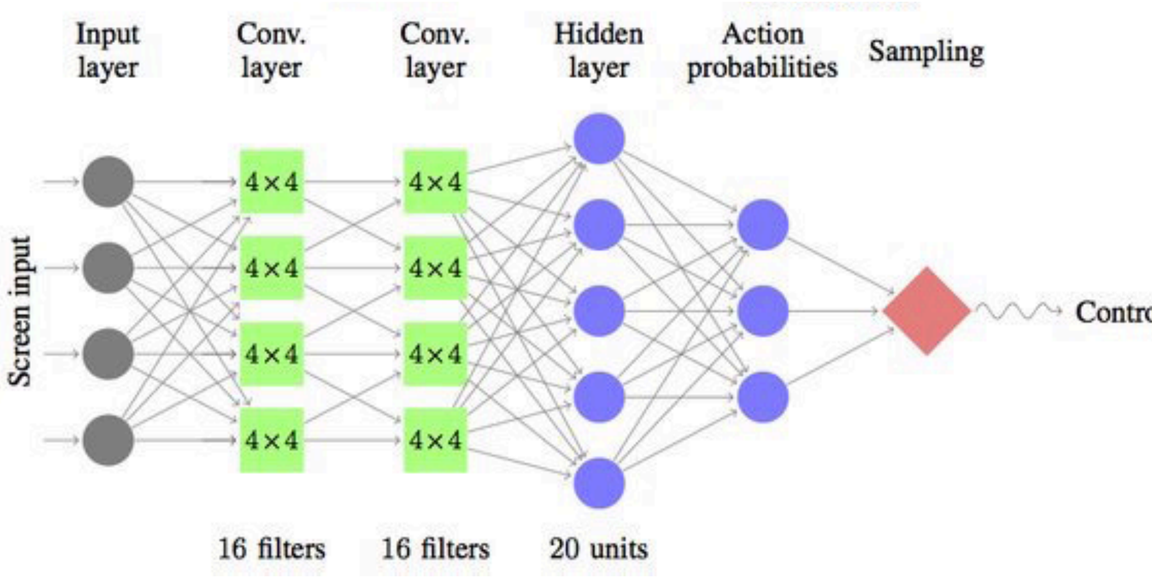
OpenAI - A toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Go.



Why Neural Networks?

Reinforcement learning algorithms are focused on learning a ***policy*** (function that takes in observations and outputs actions (e.g., motor torques)). Other algorithms are focused on learning ***value*** functions, which measure the goodness of states (i.e., the state of the world) and actions.

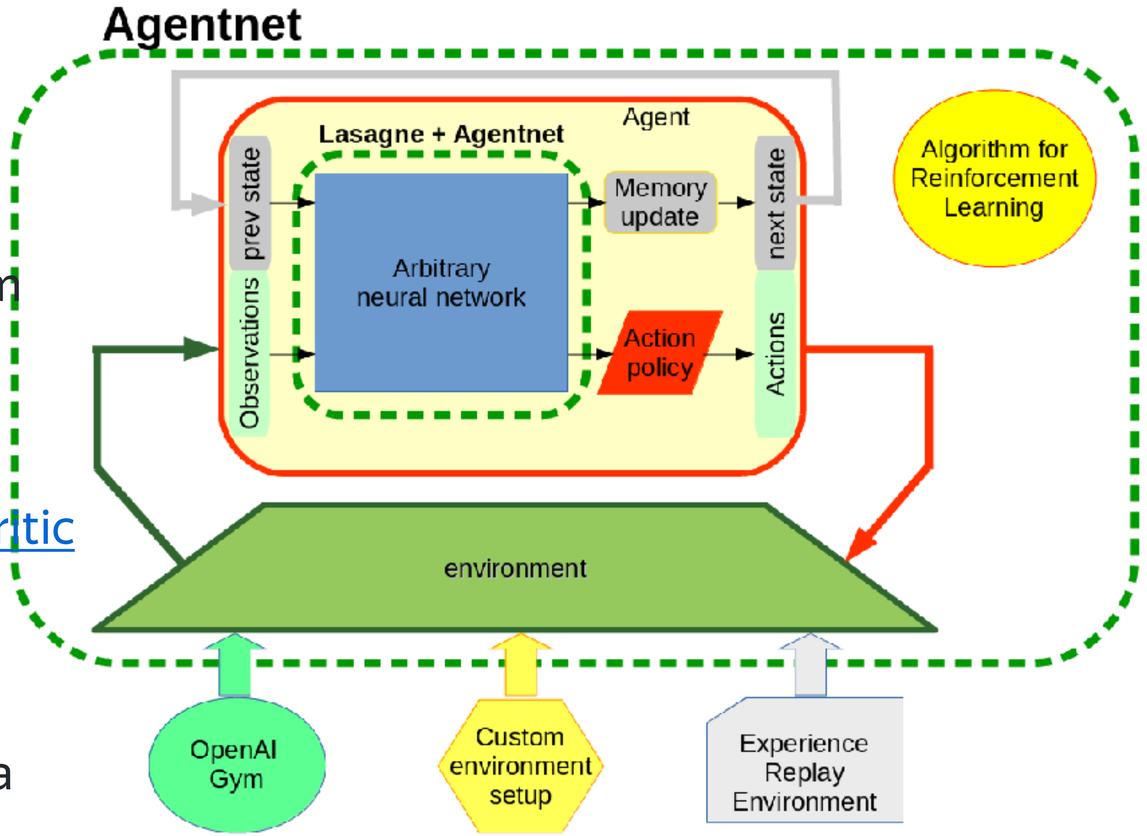
In other words, the Q-function defines a policy. We can use NN to represent the policies and Q-functions. For example, when playing Atari games, the input to these networks is an image of the screen, and there is a discrete set of actions. As a policy for this task, you can use a convolutional neural network with similar architecture, which instead outputs the probability of each action. (Guo et al., 2014, Schulman et al., 2015). (more details)



AgentNet

Deep Reinforcement Learning library for humans

- Playing Atari with Convolutional NN via OpenAI Gym
  - Can switch to any visual game thanks to awesome Gym interface
  - Very simplistic, non-recurrent suffering from atari flickering, etc.
- Deep Recurrent Kung-Fu training with GRUs and actor-critic
  - Uses the "Playing atari" example with minor changes
  - Trains via Advantage actor-critic (value+policy-based)
- Simple Deep Recurrent Reinforcement Learning setup
  - Trying to guess the interconnected hidden factors on a synthetic problem setup



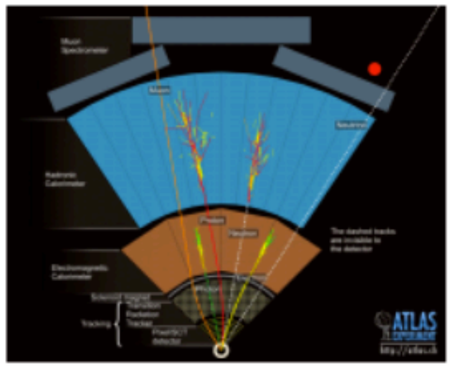
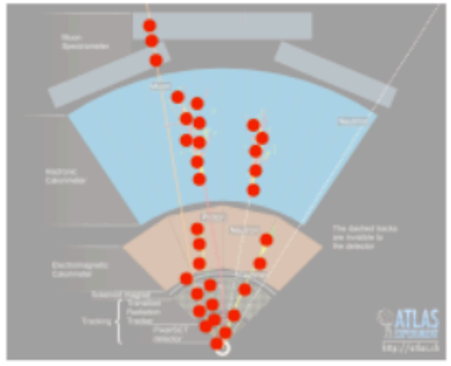
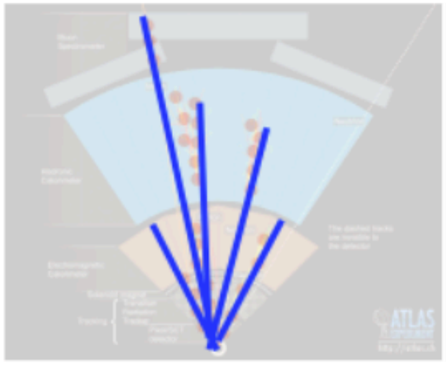
Possible HEP application: detector design optimisation.

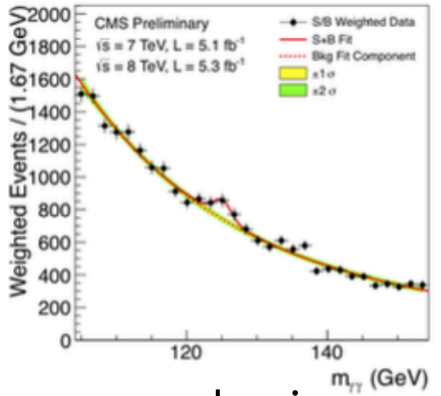
Needs a decent simulator!

Deep Learning application examples for HEP

- Jet flavour identification:
  - https://arxiv.org/abs/1407.5675 - CNN, Josh Cogan et al;
  - https://arxiv.org/abs/1603.09349 - DNN for jets, Pierre Baldi et al;
  - https://arxiv.org/abs/1701.05927 - GAN for jets, Luke de Oliveira et al;
  - https://arxiv.org/abs/1702.00748 - RNN for jets, Gilles Louppe et al;
- Ultimate application:

HEP Feature Engineering down to discovery



hits → clustering → tracking → selection → analysis

~10^7 → ~10^4 → ~50 → ~10 → ~1

10 ms → 100 ms → 1 week-1 month → 10 month

Could it be automated a bit more? or even design part/whole detector *D* for finding *X* (Dark Matter, Sterile Neutrino, etc), so $P(X|D) \rightarrow max$.

Practical Aspects

Popular libraries

- Tensorflow, https://www.tensorflow.org
  - Keras, https://keras.io/
  - Tensorboard
- Lasagna, https://github.com/Lasagne/Lasagne
  Torch, https://github.com/torch
  - PyTorch, http://pytorch.org

If working on big dataset, think of getting a GPU

Potential (popular) caveats

Neural Nets require much more attention to get results

- get enough training data;
- network initialisation (random weights);
- playing with optimisation procedure;
- training sample nature;
- choosing cost function;
- choosing architecture;
- debugging practice/tools.

Helpful resources

- Winter school on Deep Learning: https://github.com/yandexdataschool/CSC_deeplearning
- MLHEP 2016: https://github.com/yandexdataschool/mlhep2016
- MLHEP 2017: bit.ly/mlhep2017 , Reading UK, 17-23 Jul
- Advanced Machine Learning Specialisation at Coursera (to be started fall 2017
- https://github.com/aymericdamien/TensorFlow-Examples
- https://www.tensorflow.org/get_started/get_started

Conclusion

- NNs are very popular today and are a subject of intensive research differentiable operations with tensors provide a very good basis for defining useful predicting models
- Structure of a problem can be effectively used in the structure of the model
- DL requires much data and many resources that became available recently and also numerous quite simple tricks to train it better
- NNs are awesome, but not guaranteed to work well on problems without specific structure

Thank you for attention!

anaderi@yandex-team.ru

anaderiRu @ twitter

Special Thanks to

Tatiana Likhomanenko
Fedor Ratnikov
Denis Derkach
Maxim Borisyak
Mikhail Hushchyn
and all YSDA research team for helping crafting these slides