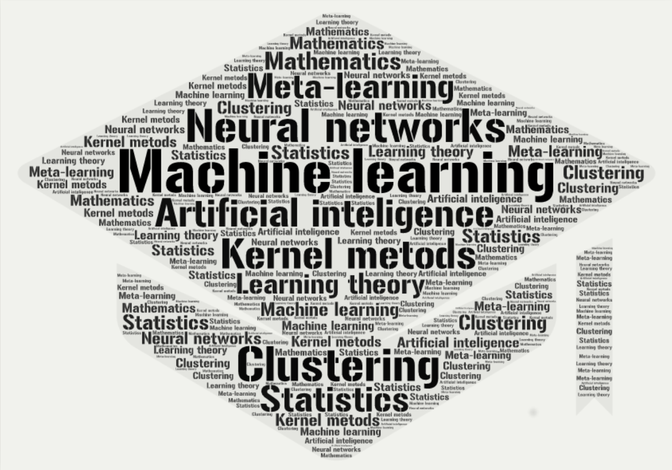
From perceptron to deep neural networks

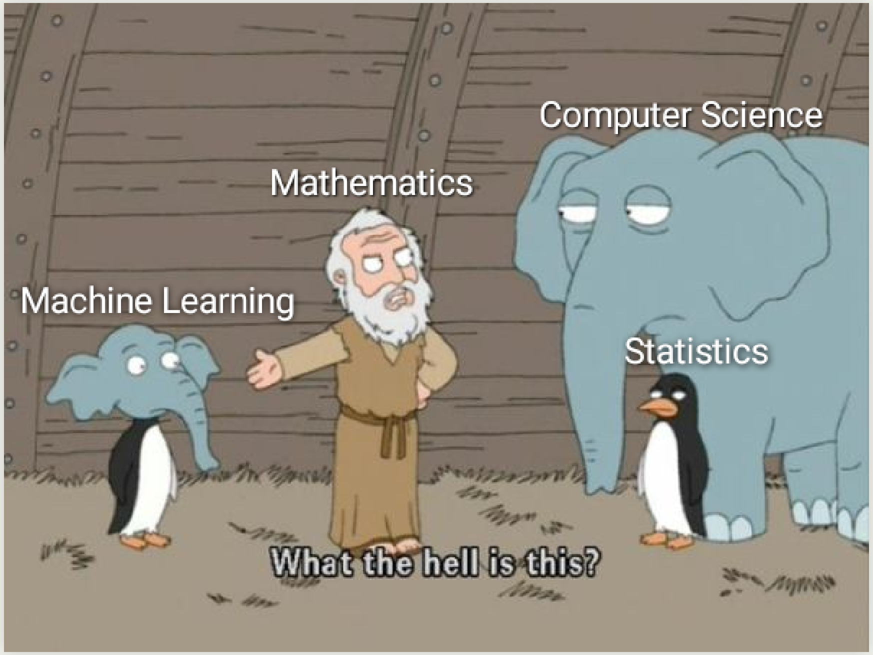
Petra Vidnerová

Ústav informatiky, AV ČR

Intro



- Artificial inteligence
- Machine learning
- Statistics
- Neural networks
- Kernel methods
- Meta-learning
- Evolutionary algorithms
- Learning theory
- ...


Computer Science
Mathematics
Machine Learning
Statistics
What the hell is this?

Outline

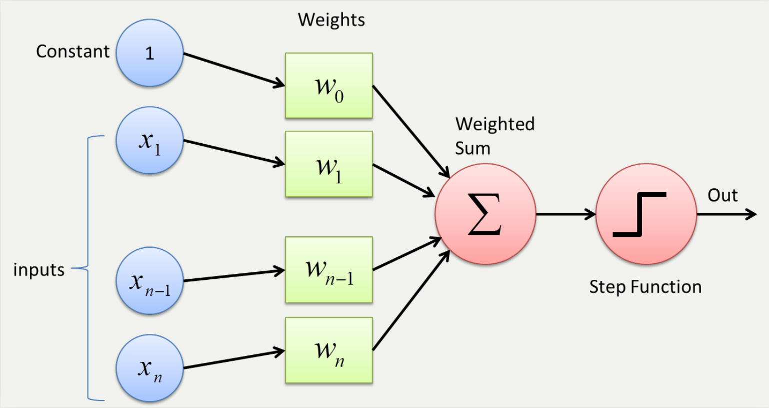
- Neural networks overview
  - First perceptron model
  - MLP, RBF networks, kernel methods
  - Deep and convolutional networks
- Our work
  - quantile estimation by neural networks
  - adversarial examples

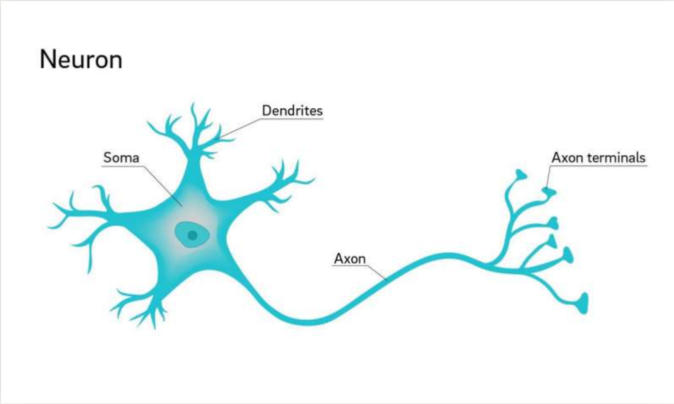
Looking back at history ...

1957 Perceptron

- Frank Rosenblatt introduced a model of neuron with supervised learning algorithm.
- First neural computer for pattern recognition, images 20x20, Mark 1 Perceptron.

Perceptron model





- inputs, weights, bias, potencial
- implement only linearly separable functions

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i - \theta \geq 0 \\ 0 & \text{else} \end{cases}$$

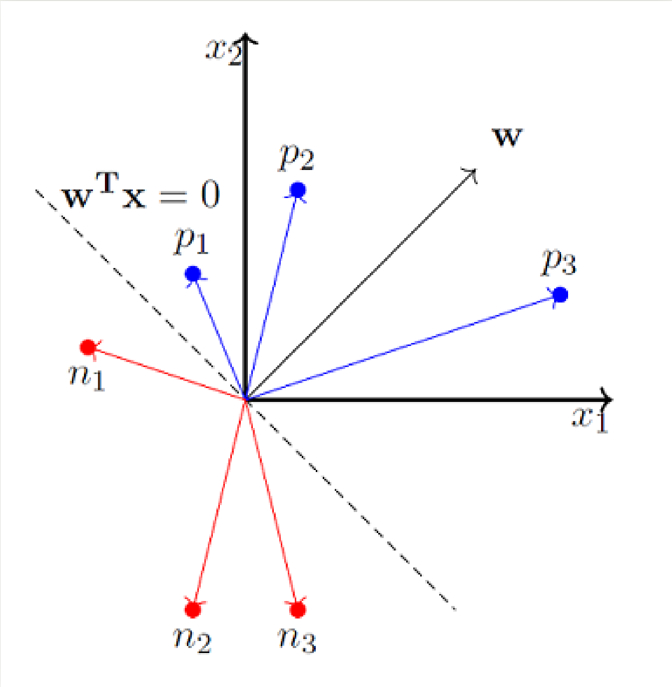
Perceptron learning algorithm

Problem: inputs $x_1, \ldots, x_n$; output `0/1`

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while !*convergence* **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w.x} < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w.x} \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end
//the algorithm converges when all the inputs are classified correctly

Geometric interpretation



$x \in P$ we need $\mathbf{wx} > 0$

i.e. we need angle between $\mathbf{w}$ and $\mathbf{x} < 90°$

$$\cos\alpha = \frac{\mathbf{wx}}{||\mathbf{w}||\cdot||\mathbf{x}||}$$

$$\mathbf{wx} > 0 \rightarrow \cos\alpha > 0 \rightarrow \alpha < 90°$$

Explanation

(α_{new}) when $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

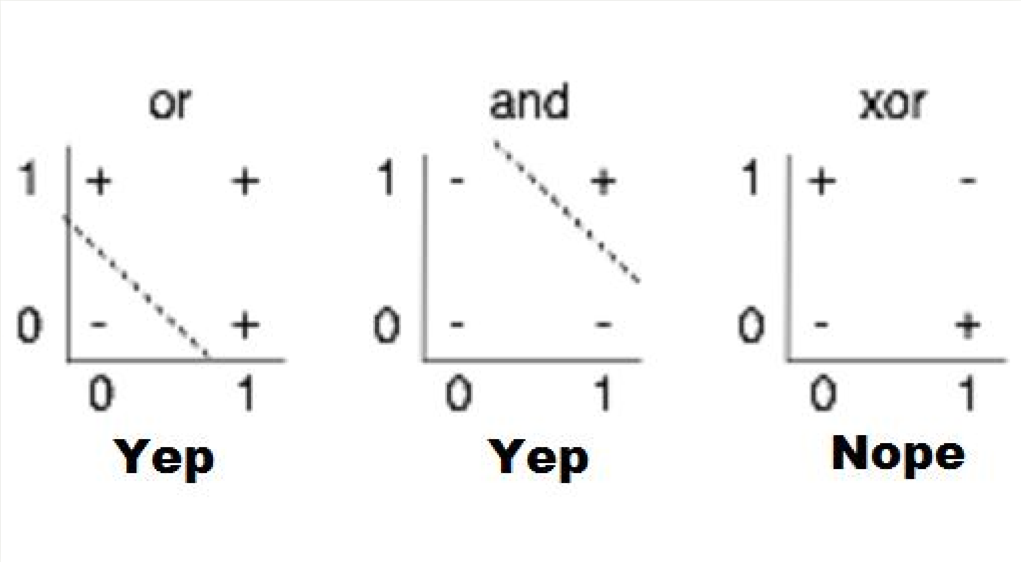
$$
\begin{aligned}
cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T \mathbf{x} \\
&\propto (\mathbf{w} + \mathbf{x})^T \mathbf{x} \\
&\propto \mathbf{w}^T \mathbf{x} + \mathbf{x}^T \mathbf{x} \\
&\propto cos\alpha + \mathbf{x}^T \mathbf{x} \\
cos(\alpha_{new}) &> cos\alpha
\end{aligned}
$$

(α_{new}) when $\mathbf{w_{new}} = \mathbf{w} - \mathbf{x}$

$$
\begin{aligned}
cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T \mathbf{x} \\
&\propto (\mathbf{w} - \mathbf{x})^T \mathbf{x} \\
&\propto \mathbf{w}^T \mathbf{x} - \mathbf{x}^T \mathbf{x} \\
&\propto cos\alpha - \mathbf{x}^T \mathbf{x} \\
cos(\alpha_{new}) &< cos\alpha
\end{aligned}
$$

Was proven that the algorithm converges. See for example Michael Collins proof.

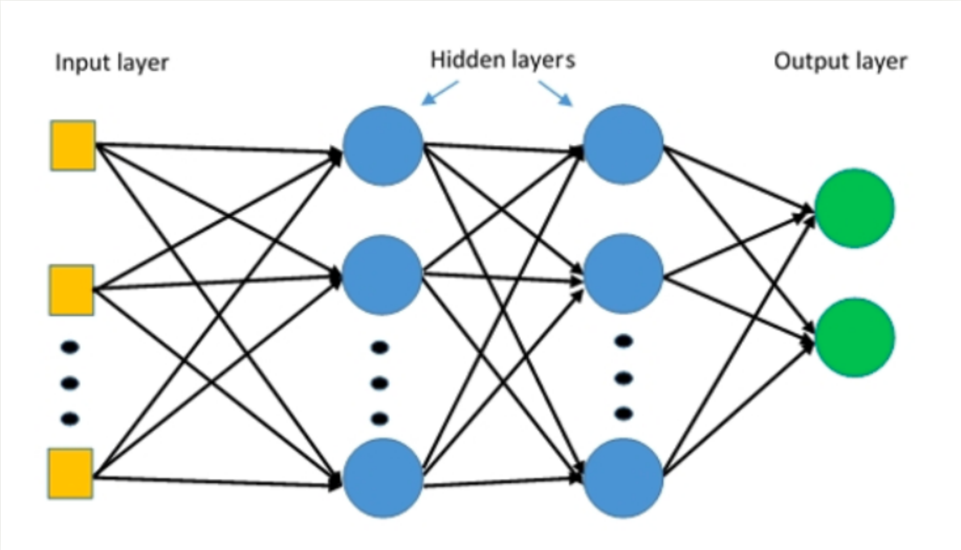
Problem XOR



1969 Minsky

Limitation of perceptron. Cannot solve the problem XOR.
Start of AI winter.

Multi-layer perceptrons (MLP)



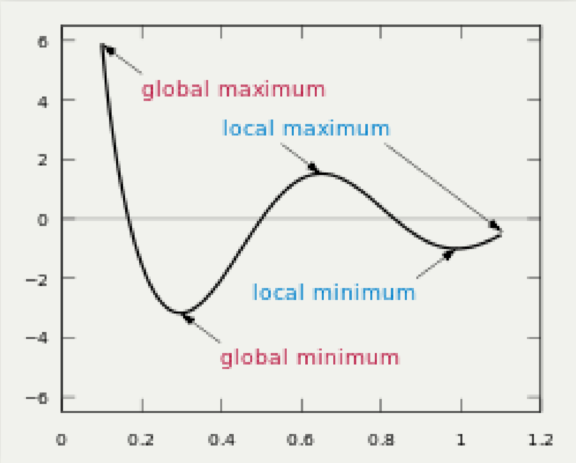
1986 Backpropagation (Rumelhart et al and others)

Least square method for learning neural networks with multiple layers.

Back propagation

- gradient learning of neural networks
  by means of backward error propagation
- optimisation of error/loss function
- how many layers we need (theory)
- how many layers we should use (practice)
- gradient descent algorithm

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$



Derivatives

$$E = \sum_{k=1}^{N} E_k \qquad \triangle w_{ij} = -\varepsilon \frac{\delta E}{\delta w_{ij}}$$

$$\frac{\delta E}{\delta w_{ij}} = \sum_{k=1}^{N} \frac{\delta E_k}{\delta w_{ij}} \qquad \frac{\delta E_k}{\delta w_{ij}} = \frac{\delta E_k}{\delta o_i} \frac{\delta o_i}{\delta \xi_i} \frac{\delta \xi_i}{\delta w_{ij}}$$

$$o_{ij} = \varphi(\sum_{k=1}^{p} (w_{ijk} x_k - \theta_{ij})) \qquad \varphi(z) = \frac{1}{1 + e^{-z}}, \frac{\delta \varphi(z)}{\delta z} = \varphi(z)(1 - \varphi(z))$$

$$\frac{\delta E_k}{\delta w_{ij}} = \frac{\delta E_k}{\delta o_i} o_i (1 - o_i) o_j$$

If o_i is the output neuron: $\frac{\delta E_k}{\delta o_i} = y_i - d_k$

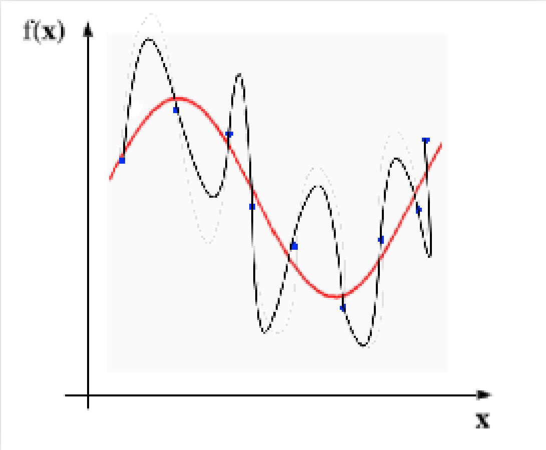
If o_i is an hidden neuron: $\frac{\delta E_k}{\delta o_i} = \sum_{r \in i^{\rightarrow}} \frac{\delta E_k}{\delta o_r} o_r (1 - o_r) w_{ir}$

Loss function and regularization

- Mean square error (MSE):

$$E = \frac{1}{N}\sum_{k=1}^{N}(f(x_k) - d_k)^2 q$$

- Function approximation - ill posed problem



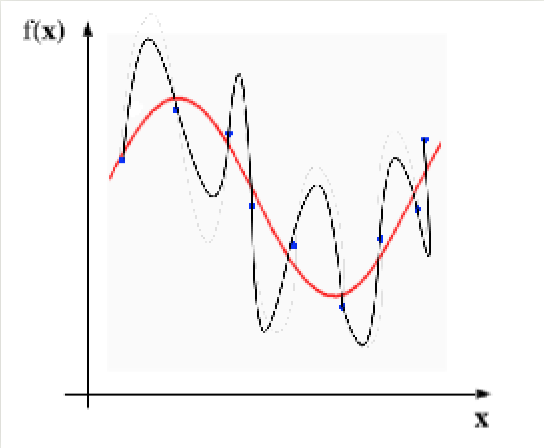
- Add regularization, weight decay

Loss function and regularization

- Mean square error (MSE):

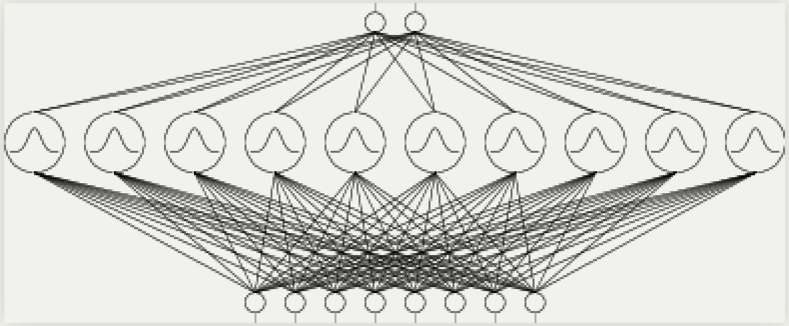
$$E = \frac{1}{N}\sum_{k=1}^{N}(f(x_k) - d_k) + \gamma\Phi[f]$$

- Function approximation - ill posed problem



- Add regularization, weight decay

RBF Networks

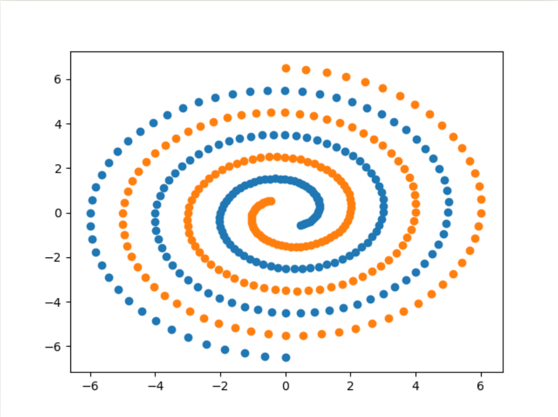


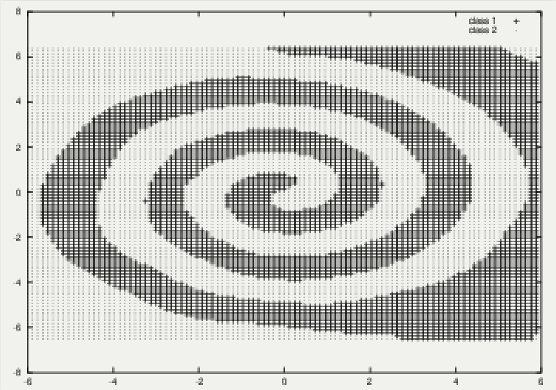
- originated in 1980s, function approximation
- network with one hidden layer
- local units
- alternative to perceptron

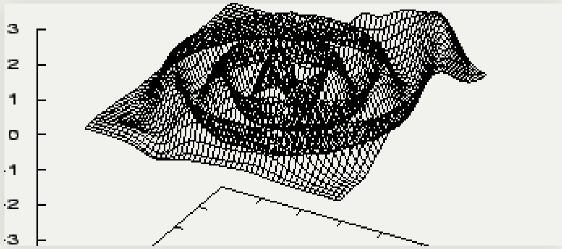
$$\text{RBF unit: } y(\vec{x}) = \varphi\left(\frac{||\vec{x}-\vec{c}||}{b}\right) = e^{-\left(\frac{||\vec{x}-\vec{c}||}{b}\right)^2}$$

$$\text{Network function: } f(\vec{x}) = \sum_{j=1}^{h} w_j \varphi\left(\frac{||\vec{x}-\vec{c}||}{b}\right)$$

Two spirals problem

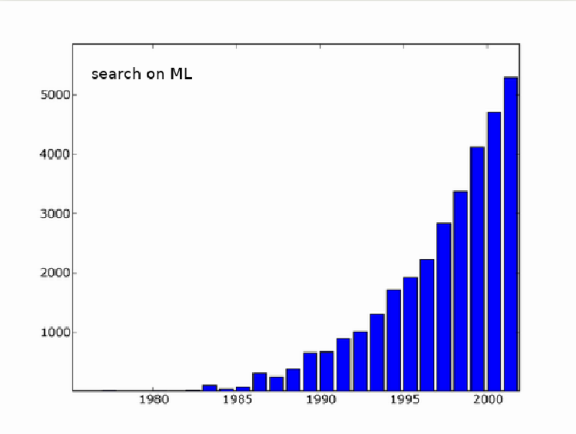


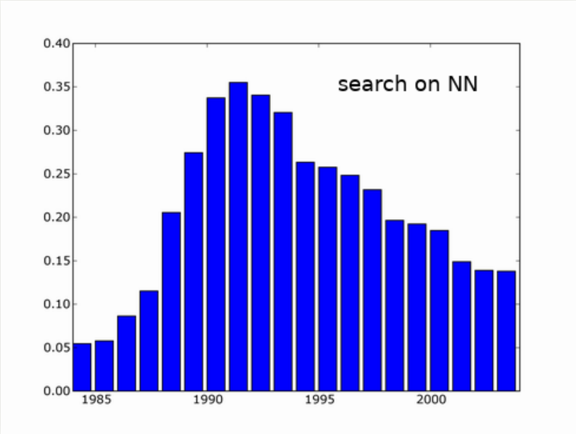


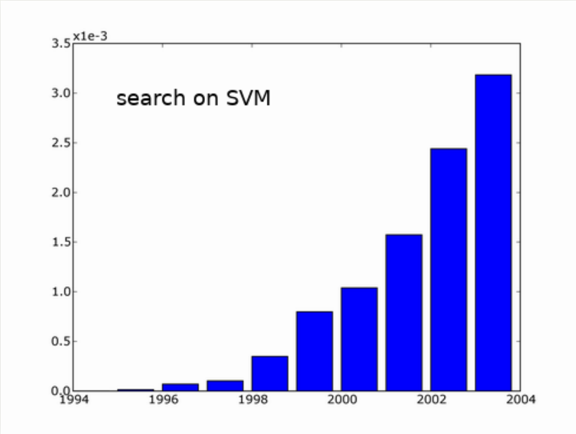


Difficult problem for linear separation.

Kernel methods (SVM)

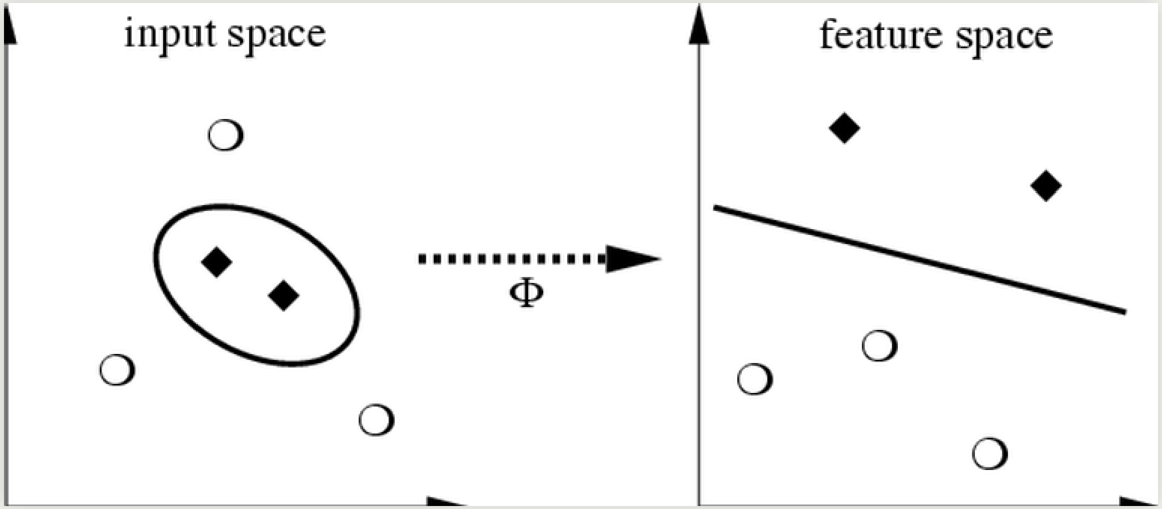






In '90s kernel methods and SVMs were very popular.

Mapping to the feature space



Choose a mapping to a (high dimensional) dot-product space - feature space.

Mercer's condition and kernels

Mercer theorem:

If a symmetric function $K(x, y)$ satisfies

$$\sum_{i,j=1}^{M} a_i a_j K(x_i, x_j) > 0$$

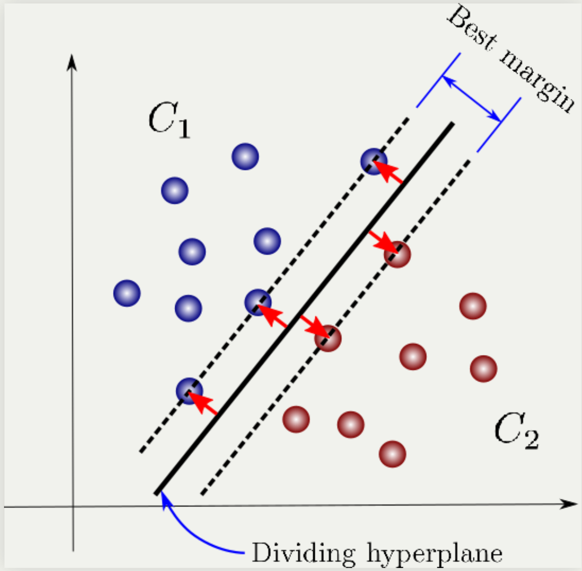
for all $M \in \mathbf{N}$, x_i and $a_i \in \mathbf{R}$, there exists a mapping function ϕ that maps x into the dot-product feature space and

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

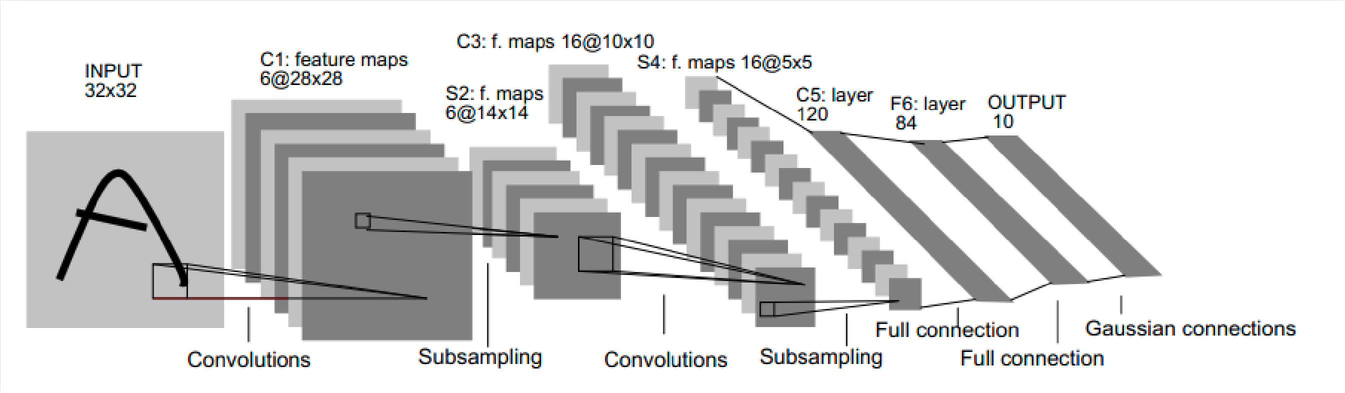
and vise versa. Functio K is called **kernel**.

Support vector machine

Looking for a separating hyperplane with the maximal margin.



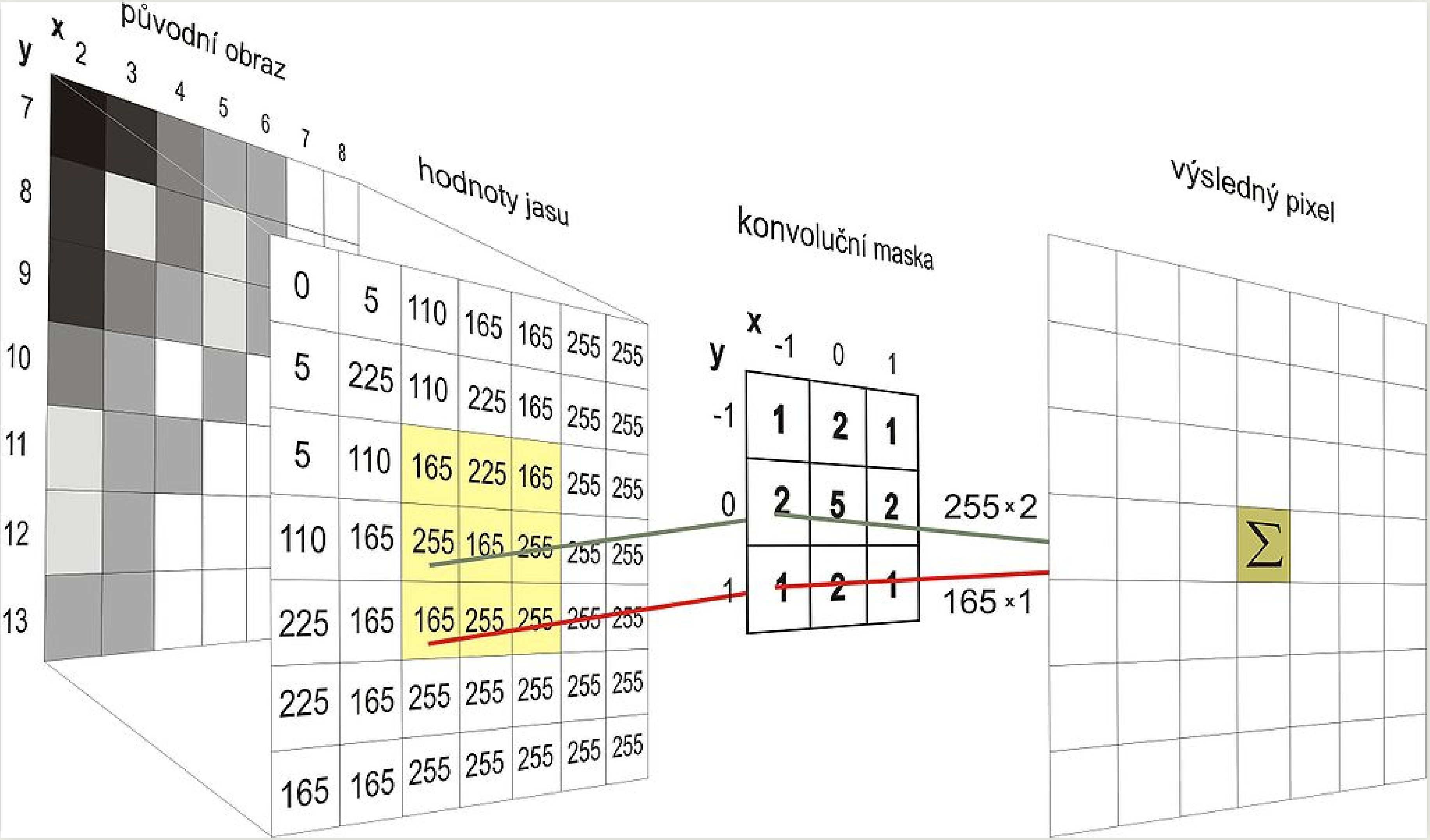
Convolutional Networks



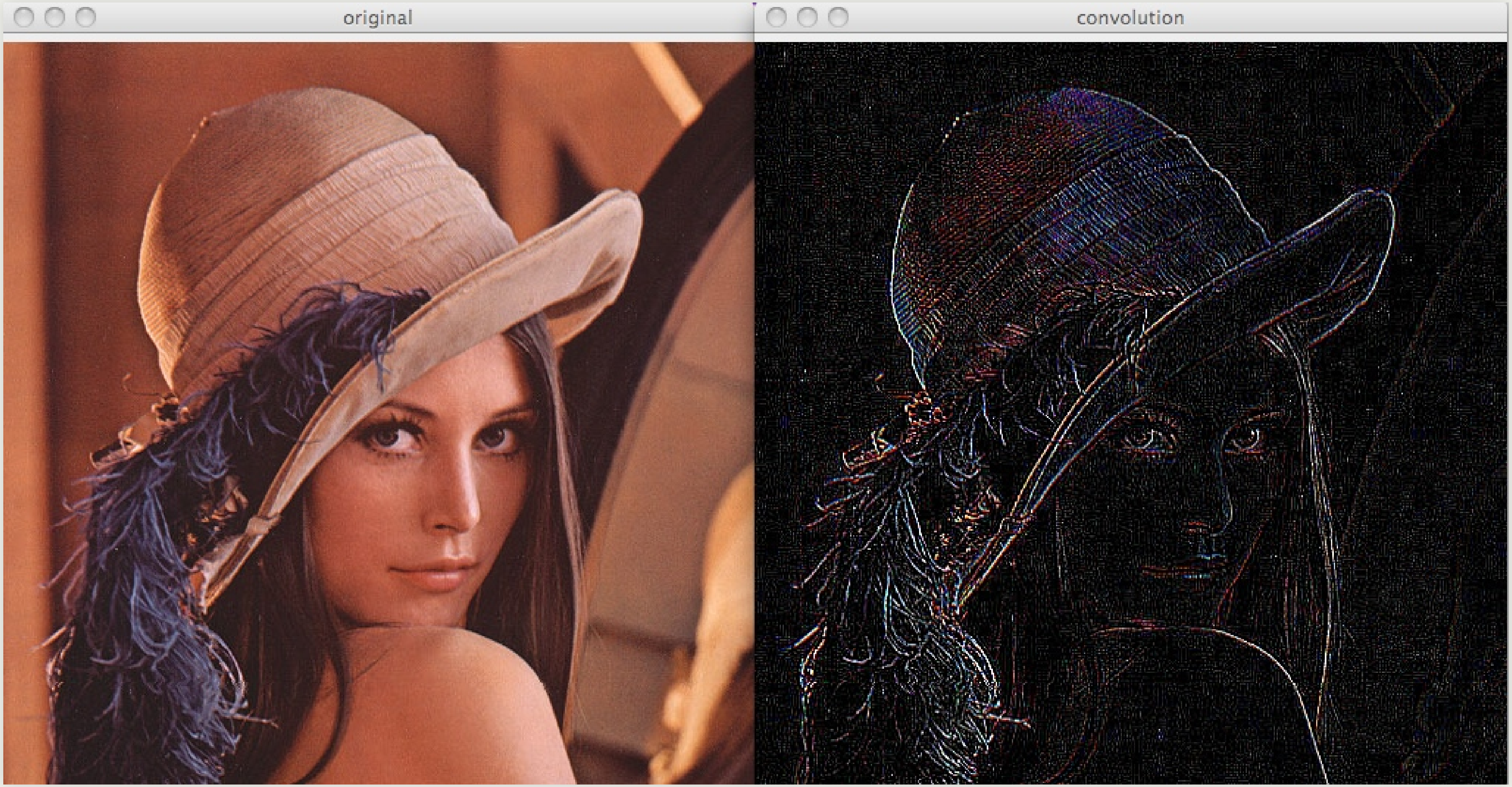
1994 LeNet5 (Yann LeCun)

Convolutional layers for feature extraction. Subsampling layers (max-pool layers).

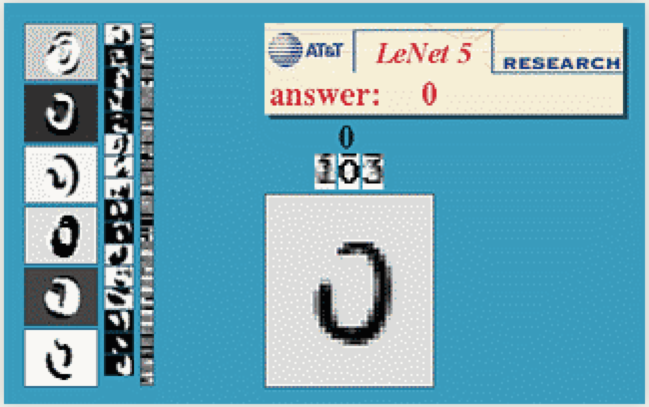
Descrete convolution



Edge detection

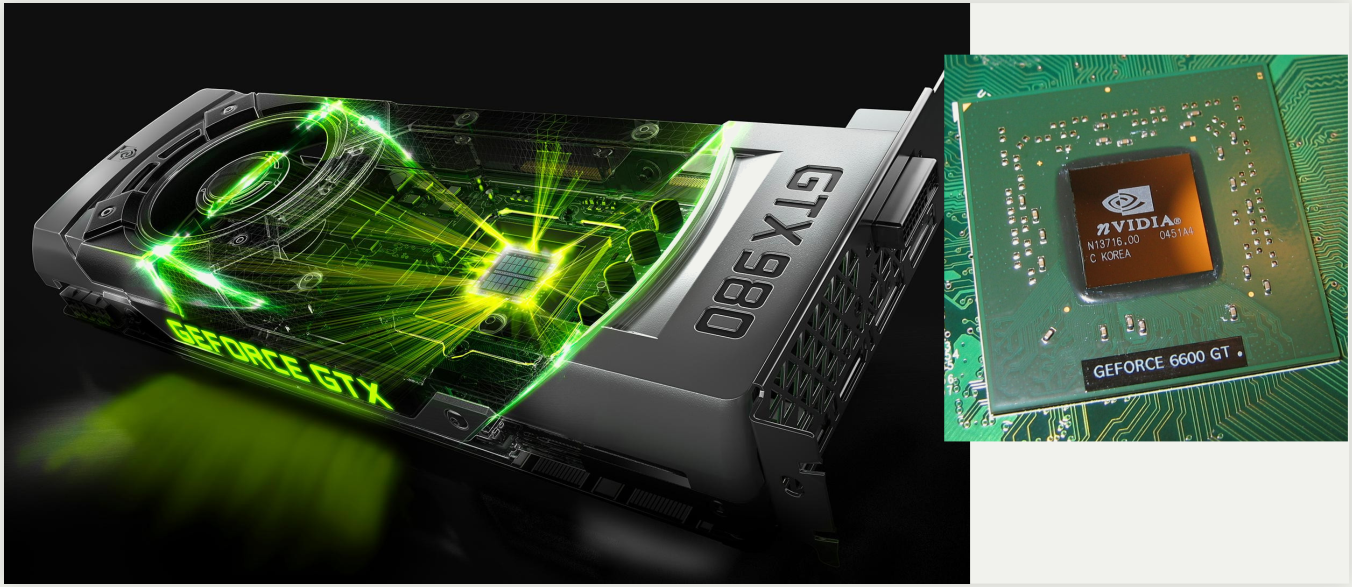


LeNet network Example



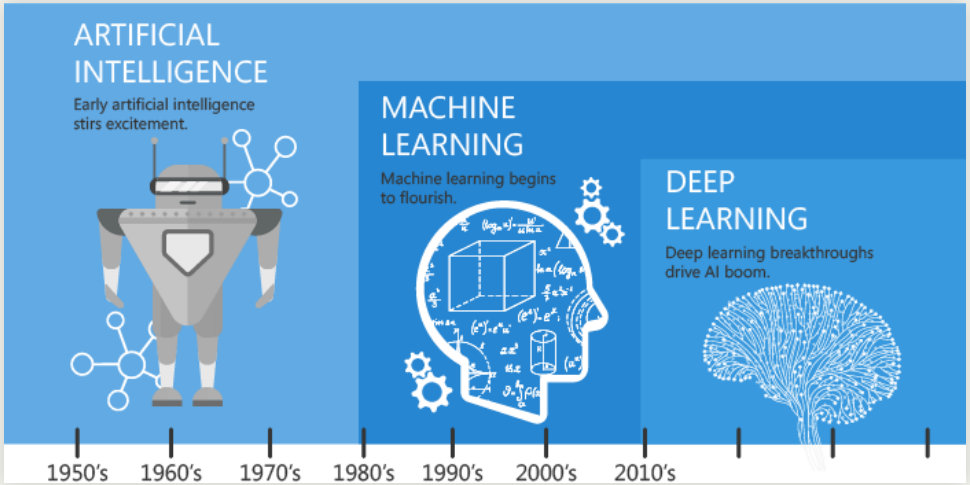
Was applied in several banks for recognition of numbers on cheques.

Deep neural networks



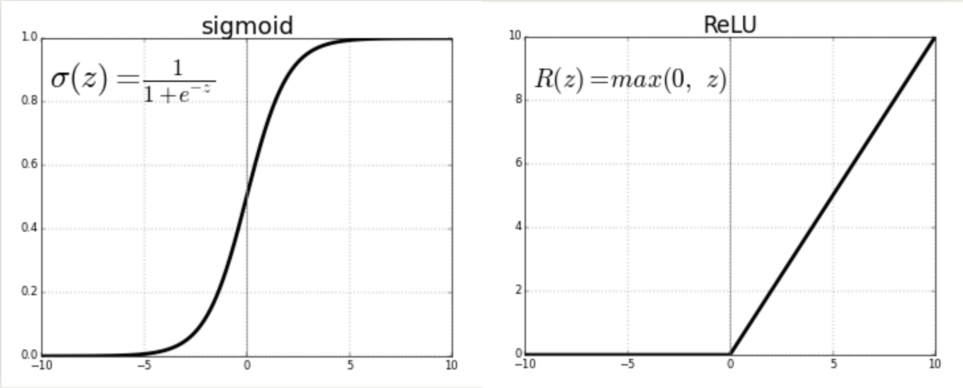
Bengio, Hinton, LeCun (2009)

Big data + GPUs/TPUs. Learning with millions of neurons. New architectures available for computer vision, video processing, NLP.



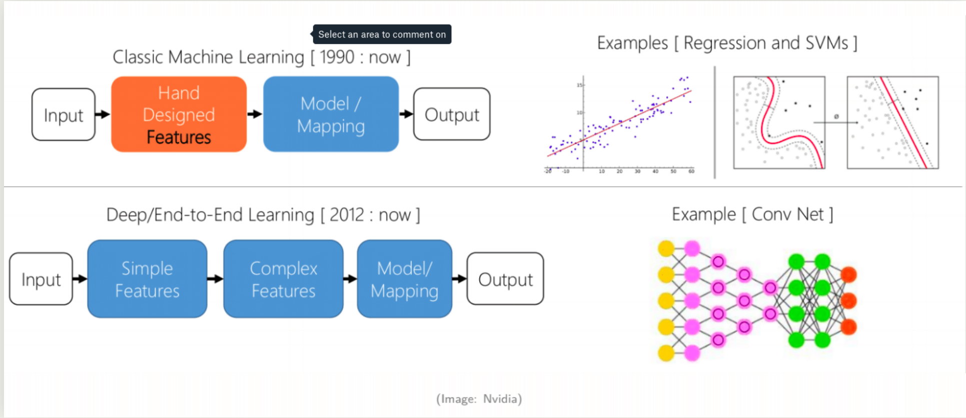
Advances in deep learning ...

- ReLU activation units (vanishing gradient problem)



Advances in deep learning

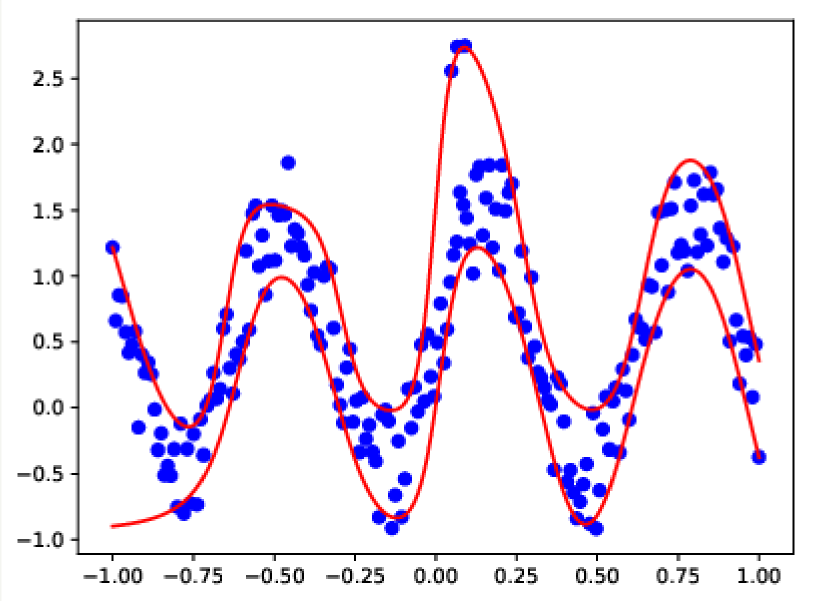
- Dropout - type of regularization, as ansamble
- Learning with mini-batches
- Convolutional layers - adaptive preprocessing



- Transfer learning
- Float types with lower numbers of bits (8bits)
- Missing interpretation

Our Work

Quantile Estimation



- instead of mean trend predict the desired quantile
- joint work with Jan Kalina

Quantile Estimation

- data points $(x_i, y_i), i = 1, \dots, N$
- for each i resudial $\xi_i = y_i - f(x_i)$
- MSE:

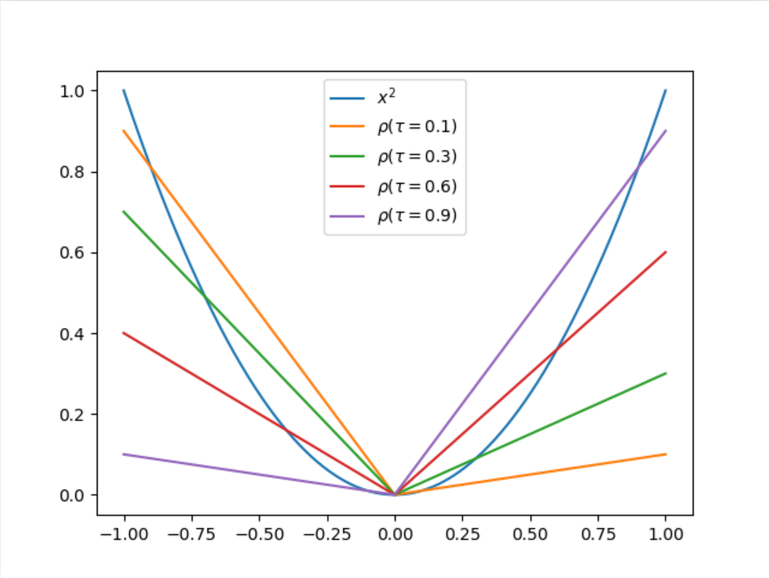
$$E = \sum_{k=1}^{N} \xi_i^2 = \sum_{k=1}^{N} \rho(\xi_i)$$

$$\rho(z) = z^2$$

Quantile Estimation

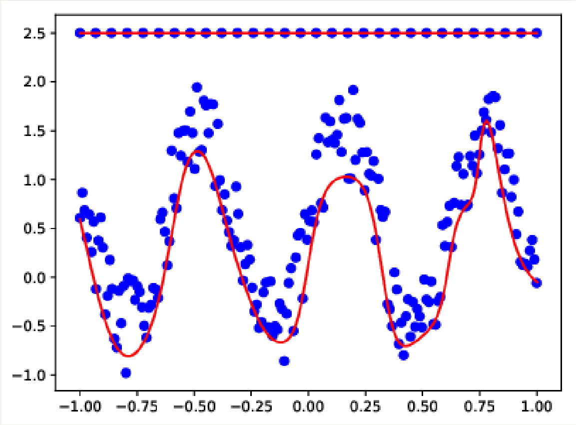
- modify a loss function (R. Koenker)

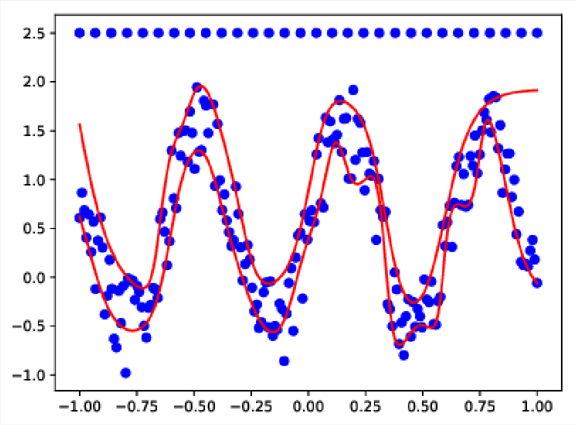
$$\rho(z) = \begin{cases} \tau|x|, & \text{if } x > 0 \\ (1-\tau)|x|, & \text{else} \end{cases}$$

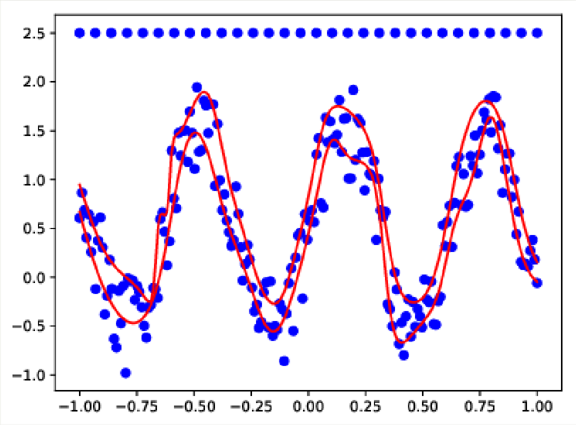


Quantile Estimation - toy example

Simple MLP.







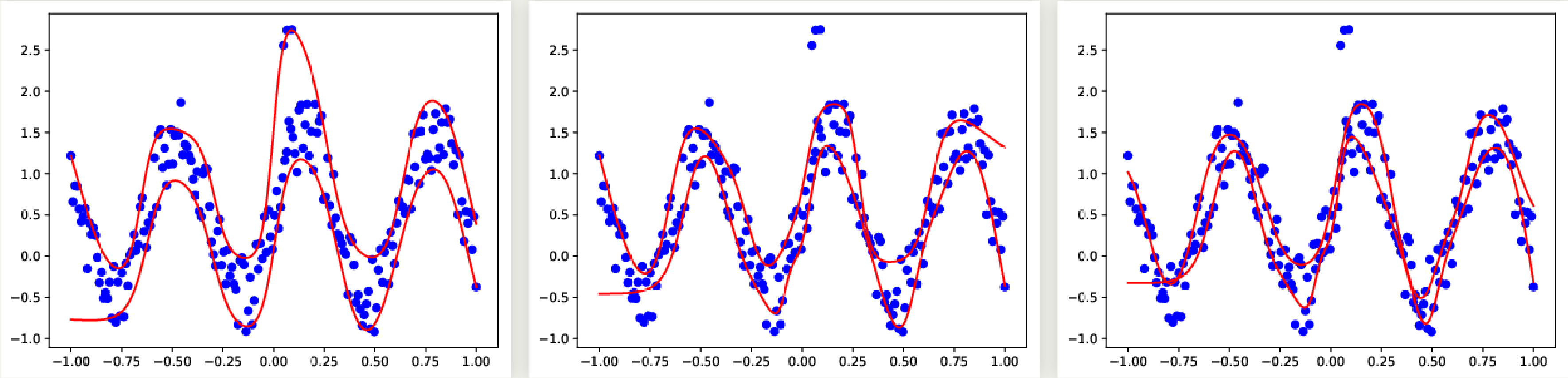
$\tau = 0.1$and 0.9 (left)

$\tau = 0.2$and 0.8 (center)

$\tau = 0.3$and 0.7 (right)

Quantile Estimation - toy example

Simple MLP.



$\tau = 0.1$and 0.9 (left)

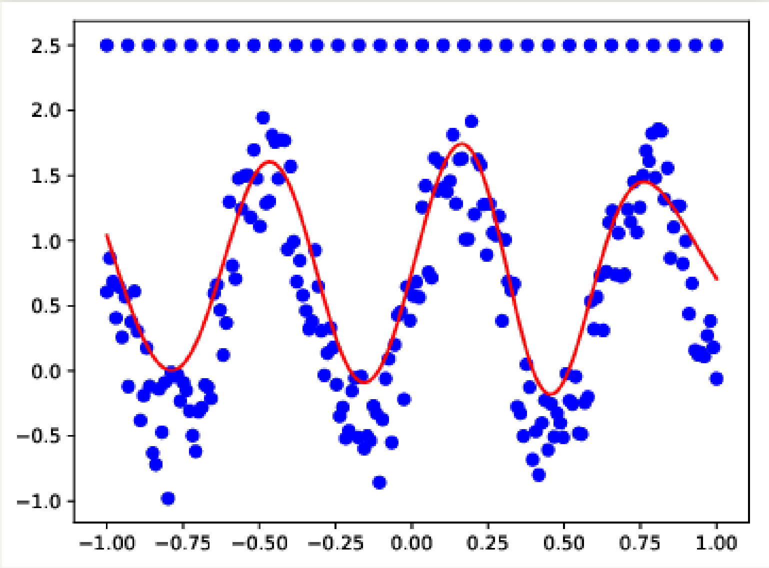
$\tau = 0.2$and 0.8 (center)

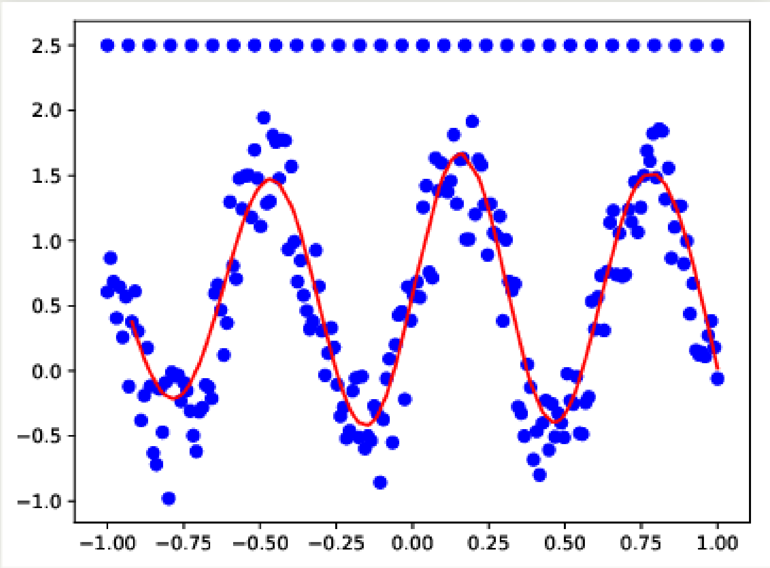
$\tau = 0.3$and 0.7 (right)

Network robustification

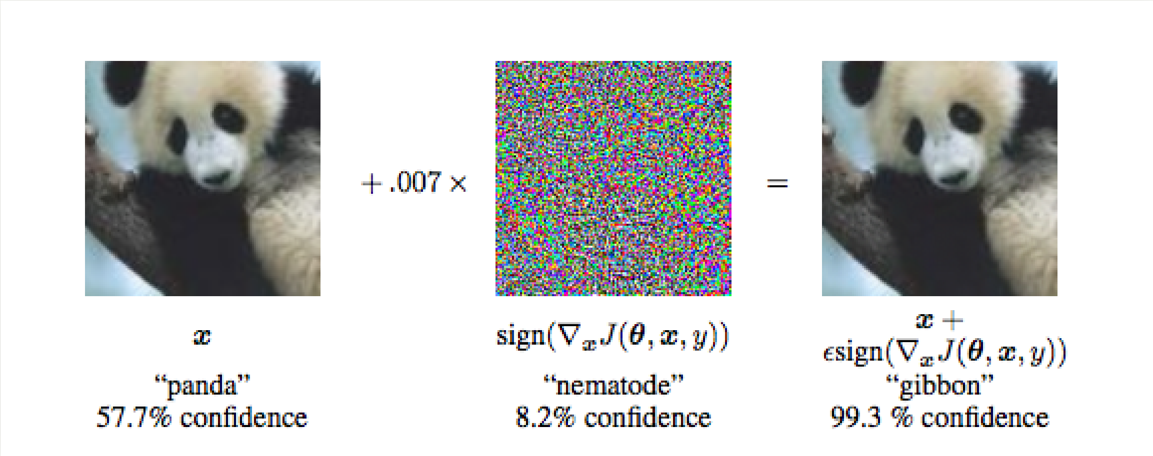
Simple RBF.

Data above and bellow quantiles are omitted.





Adversarial Examples



2015 Goodfellow

- applying an imperceptible non-random pertrubation to an input image, it is possible to arbitrarily change the machine learning model prediction
- for a human eye, adversarial examples seem close to the original examples
- a **security flaw** in a classifier

Crafting adversarial examples

- optimisation problem, w is fixed, x is optimised
- minimize $||r||_2$ subject to $f(x+r) = l$ and $(x+r) \in \langle 0, 1 \rangle^m$
- a box-constrained L-BFGS

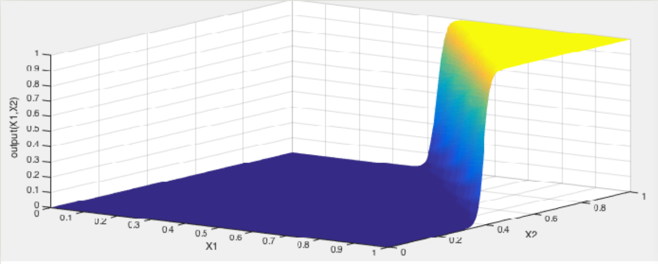
- **gradient sign method** - adding small vector in the direction of the sign of the derivation
- we can linearize the cost function around θ and obtain optimal perturbation

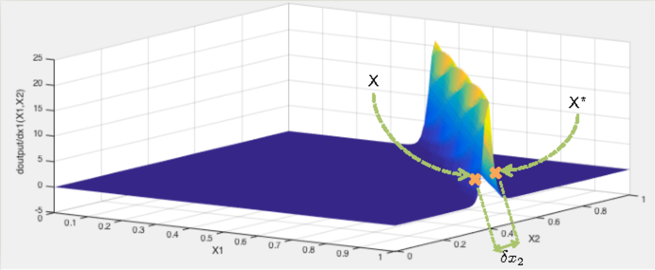
$$\eta = \varepsilon \text{sign}(\Delta_x J(\theta, x, y))$$

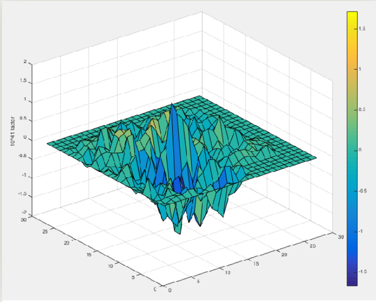
Crafting adversarial examples II.

- **adversarial saliency maps** -- identify features of the input that most significantly impact output classifications (Papernot, 2016)

- motivation: output function (left), forward derivation (right)



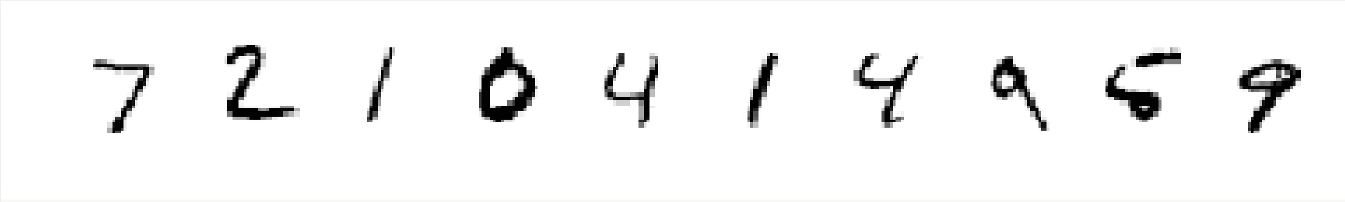




- misclasify X such that it is assigned a target class $t \neq label(X)$, $F_t(X)$ must increase, while $F_j(X), j \neq t$ decrease

$$S(X,t)[i] = \begin{cases} 0 \text{ if } \frac{\delta F_t(X)}{\delta X_i} < 0 \text{ or } \sum_{j \neq t} \frac{\delta F_j(X)}{\delta X_i} > 0 \\ \frac{\delta F_t(X)}{\delta X_i} \left| \sum_{j \neq t} \frac{\delta F_j(X)}{\delta X_i} \right| \text{ otherwise} \end{cases}$$

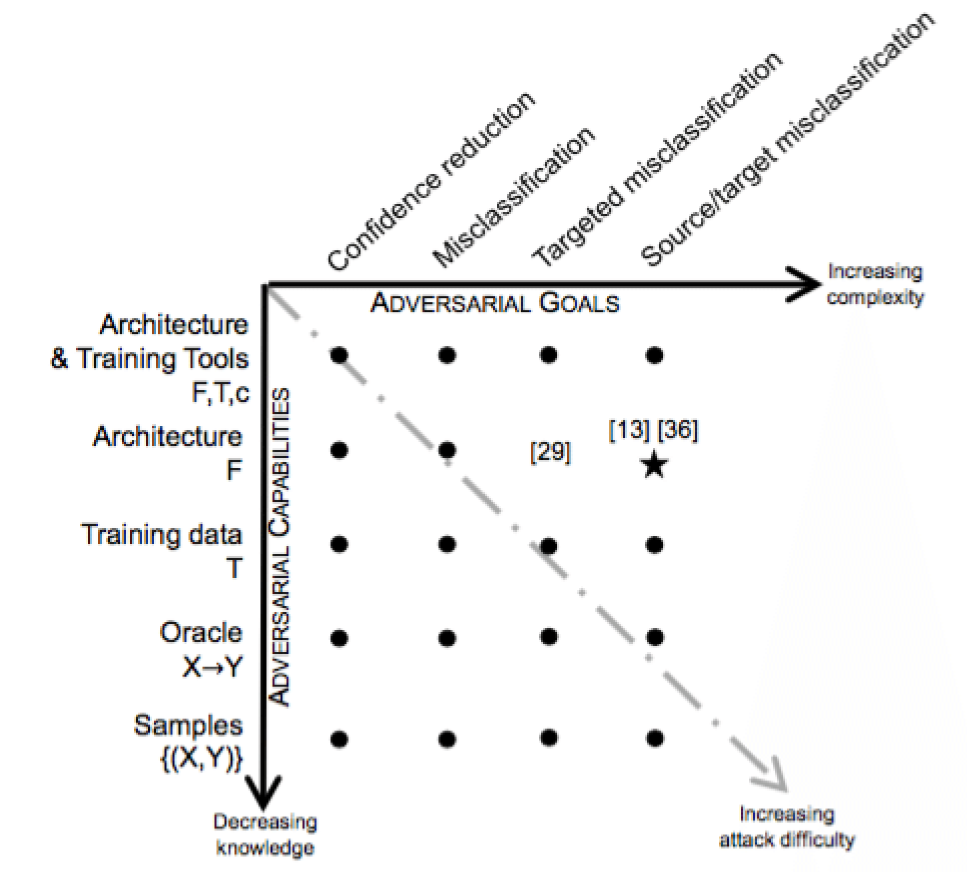
FGSM vs. Saliency maps



Crafted by FGSM:

Crafted by saliency maps:

Attack taxonomy



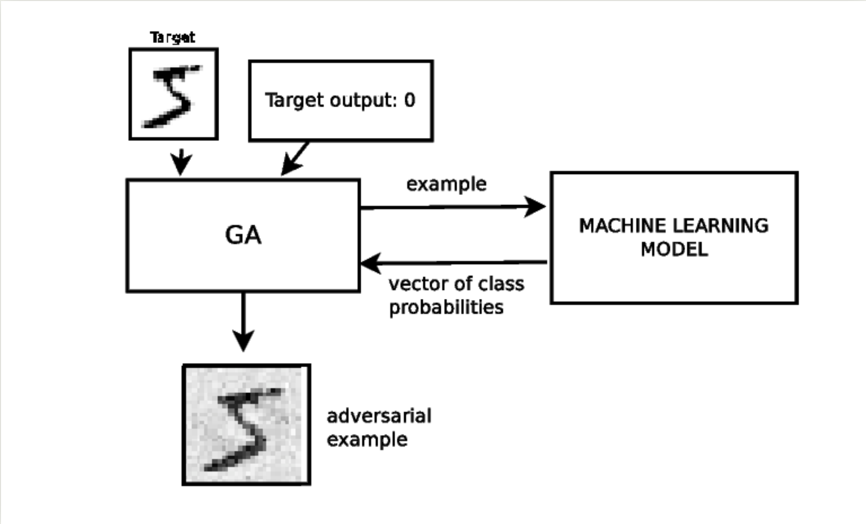
Results on GTSRB

Genetic Crafting Algorithm

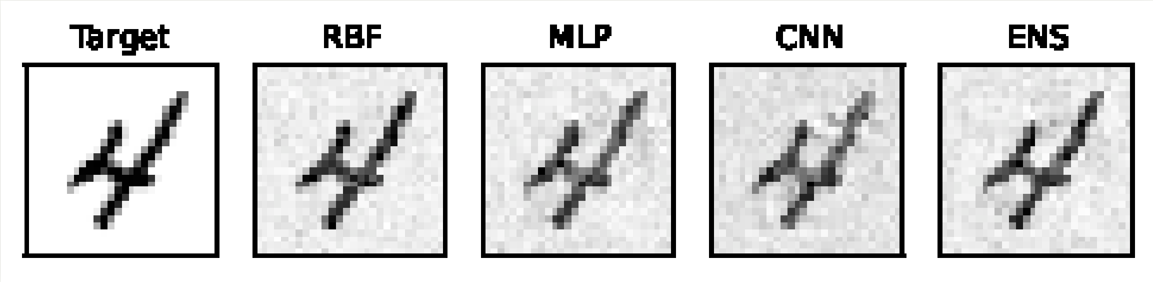
- To obtain an adversarial example for the trained machine learning model, we need to optimize the input image with respect to model output.
- For this task we employ a GA – robust optimisation method working with the whole population of feasible solutions.
- The population evolves using operators of selection, mutation, and crossover.
- The ML model and the target output are fixed.

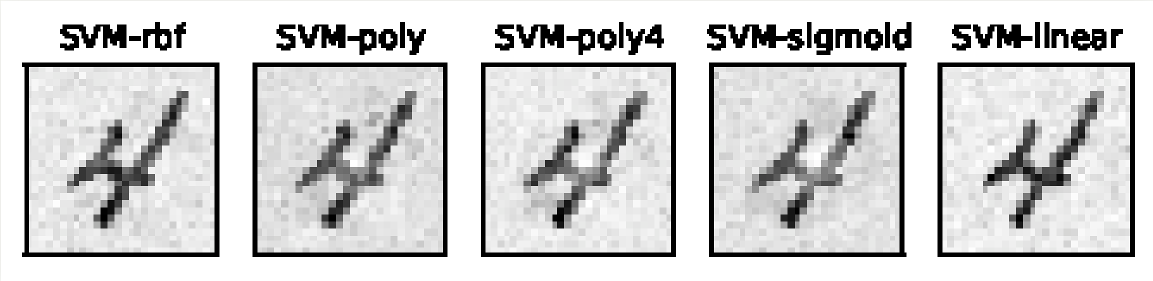
Black-box approach

- genetic algorithms to generate adversarial examples
- machine learning method is a blackbox
- applicable to all methods without the need to acess models parameters (weights)



Results for different ML models





Questions?