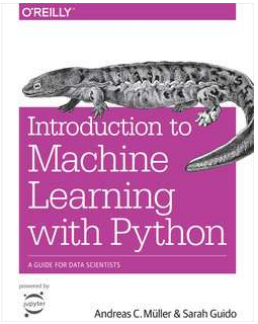
Applied Machine Learning
Support Vector Machines & Neural Networks for Regression

BSc course Informatiekunde 2026
https://staff.fnwi.uva.nl/a.visser/education/AML

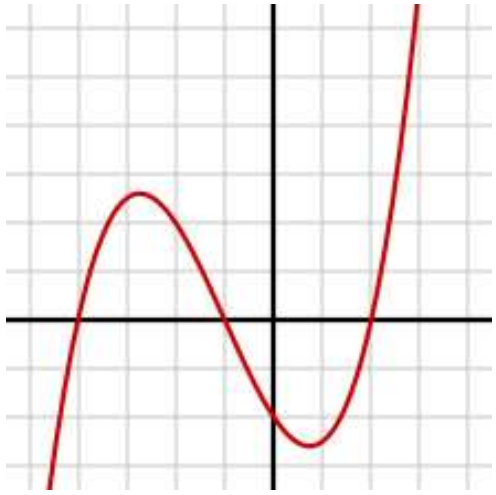
Arnoud Visser
Intelligent Robotics Lab & Computer Vision Lab
Informatics Institute
Universiteit van Amsterdam
A.Visser@uva.nl

Illustrations courtesy of Maarten Marx, Sarah Guido, Yolanda Hagar, and many others.

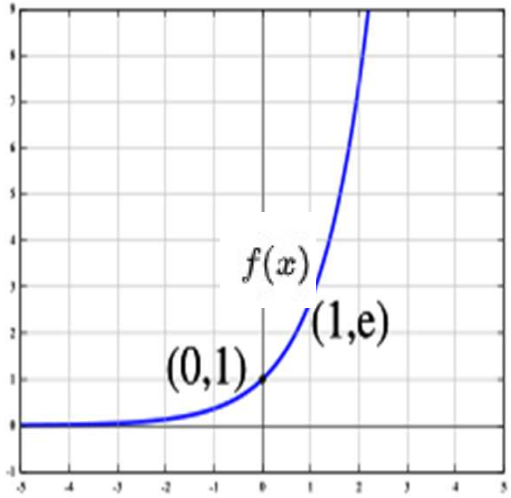


Analytic functions

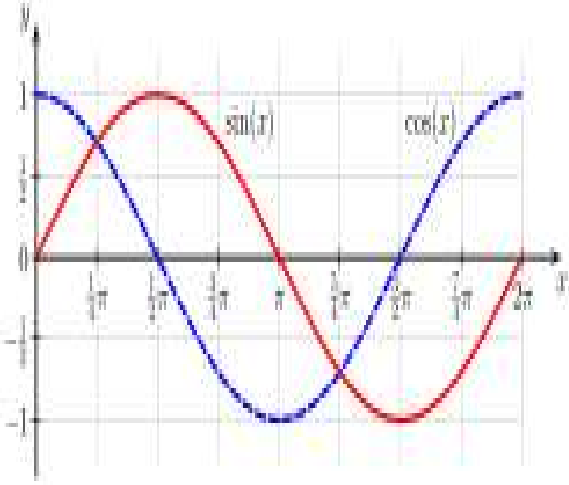
- Smooth functions, functions that are infinitely differentiable, are analytic functions.



Polynomials



Exponential



Trigonometric

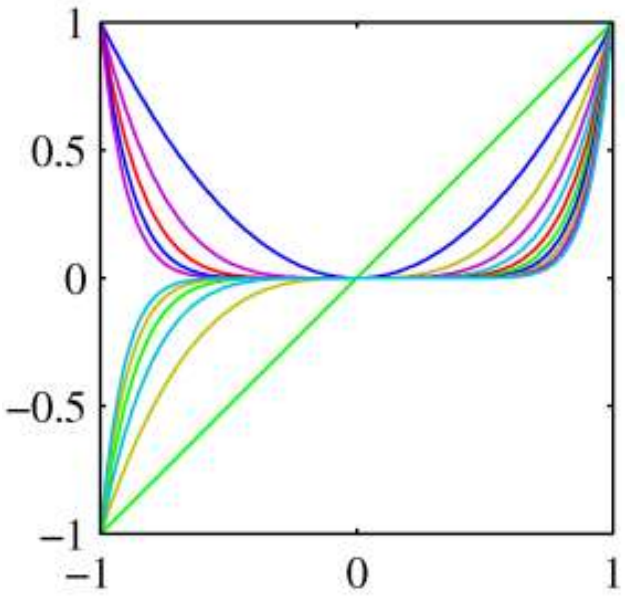
- An analytic function can be written as infinit sum of basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n (x - x_0)^n$$ ← Polynomial

basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n \ \phi_n(x - x_0)$$

- Polynomial basis functions $\phi_n(\boldsymbol{x}) = \boldsymbol{x}^n$



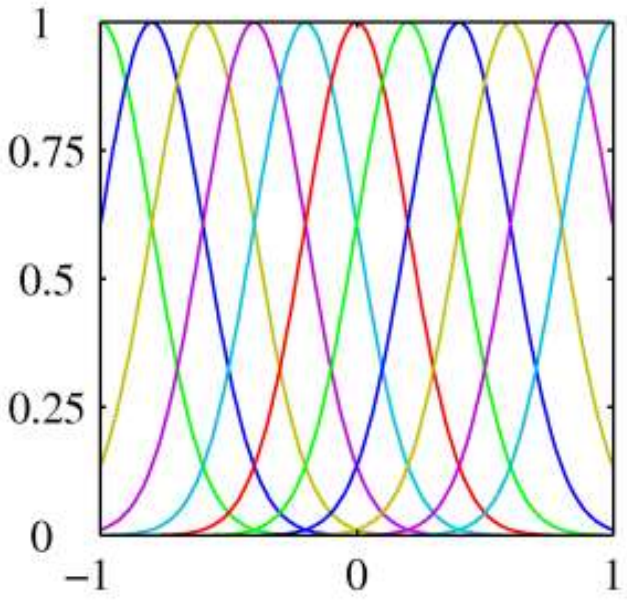
Based on slides from Byron Boots, which was inspired by Christopher Bishop

Other basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n \quad \phi_n(x - x_0)$$

- Gaussian basis functions $\phi_n(\boldsymbol{x}) = e^{-\frac{(x-\mu_n)^2}{2\sigma^2}}$

Mixture of Gaussians



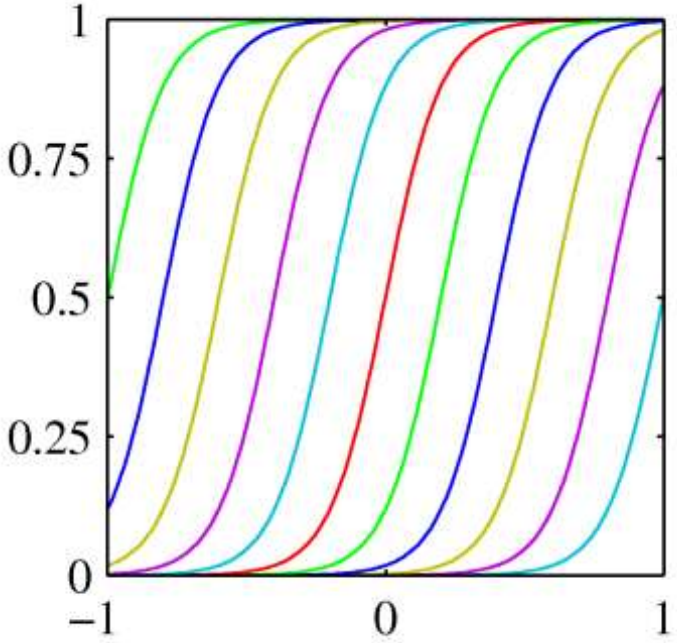
Based on slides from Byron Boots, which was inspired by Christopher Bishop

Other basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n \quad \phi_n(x - x_0)$$

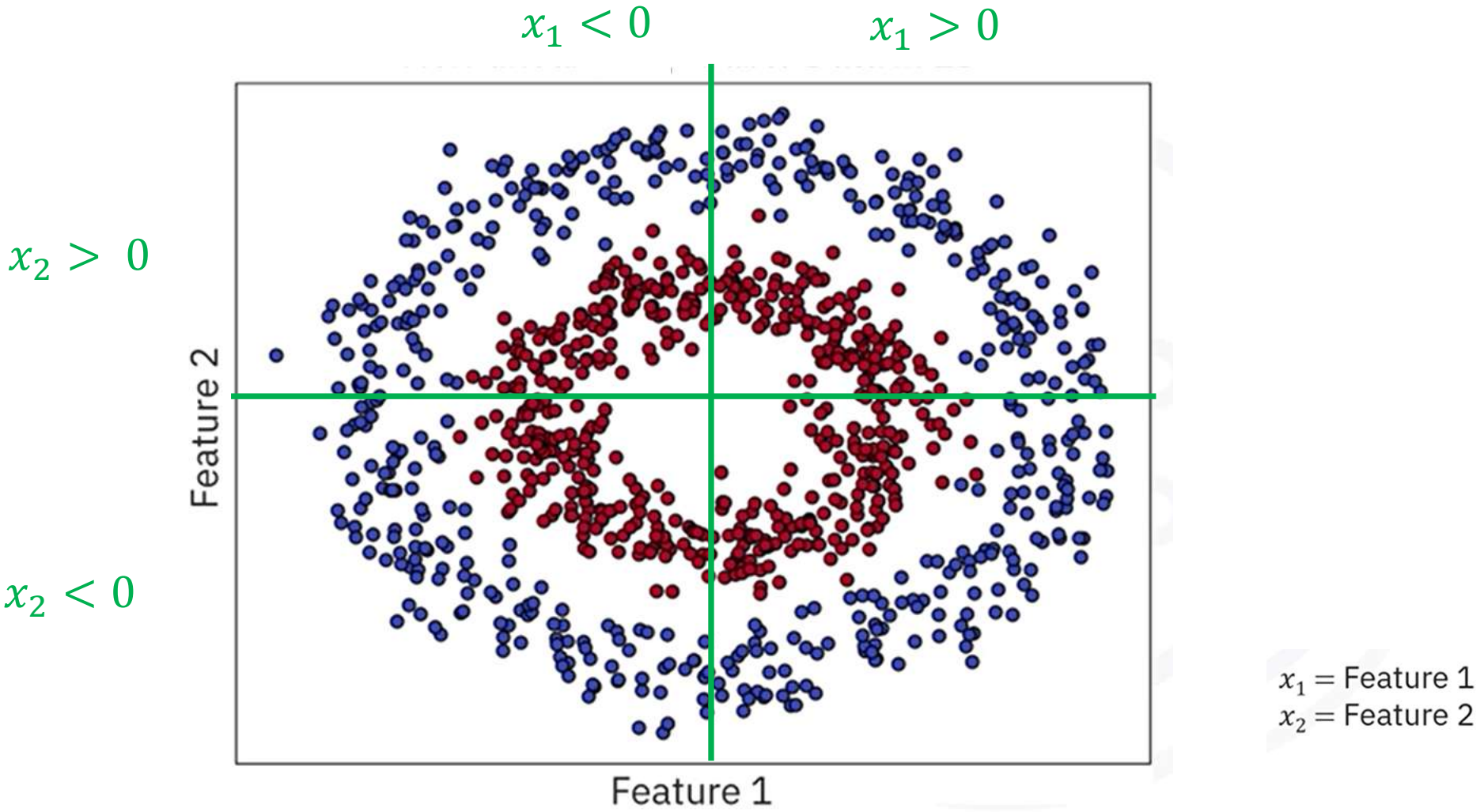
- Sigmoidal basis functions $\phi_n(\boldsymbol{x}) = \varsigma(\frac{\boldsymbol{x}-\mu_n}{\sigma})$ $\zeta(\boldsymbol{x}') = \frac{1}{1+e^{-x\prime}}$

Will come back
when we cover
Support Vector Machines



Based on slides from Byron Boots, which was inspired by Christopher Bishop

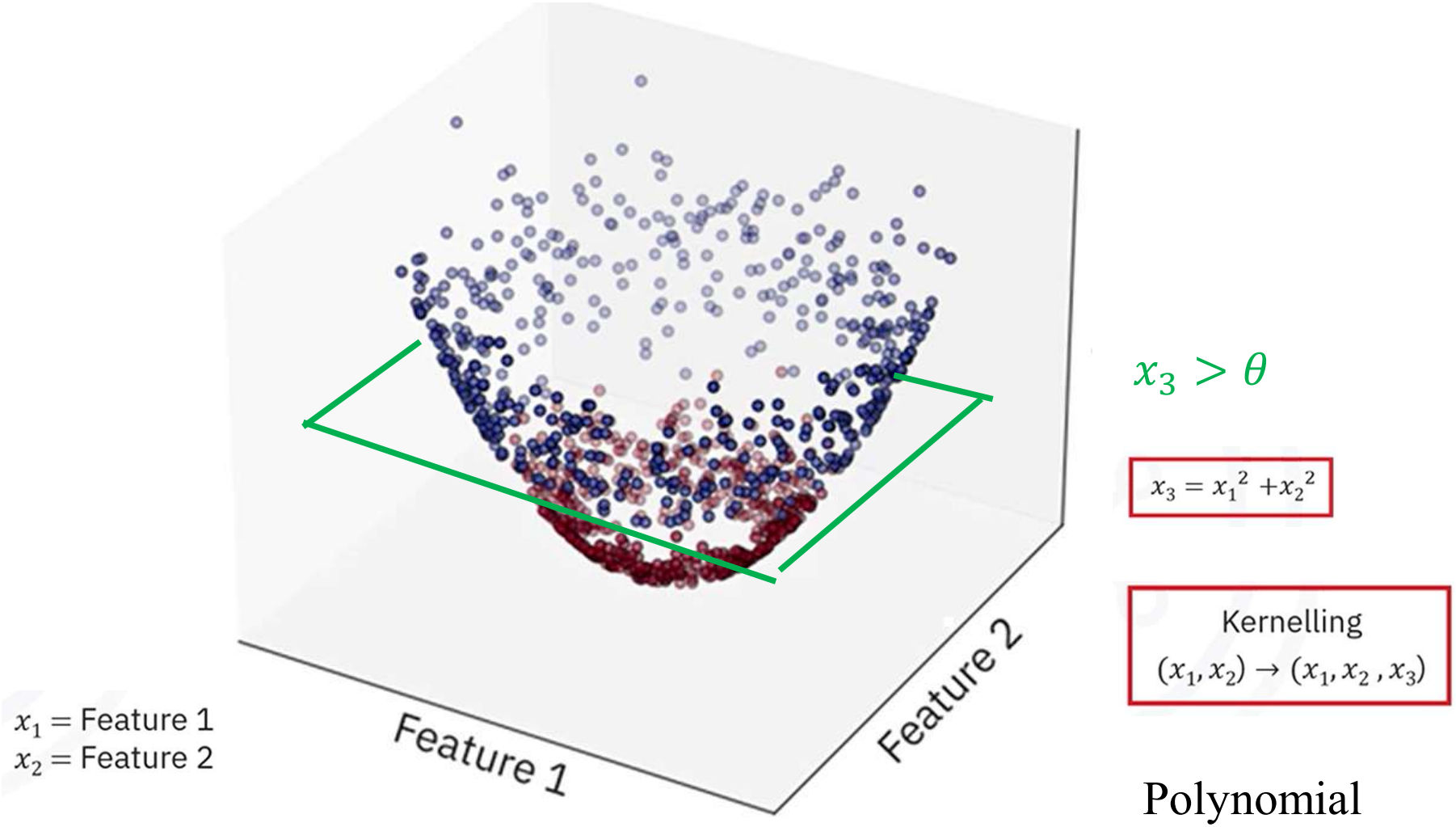
Two subset of samples



- A Decision Tree would have difficulty to split this data

Courtesy Joseph Santarcangelo & Jeff Grossman

Two subset of samples

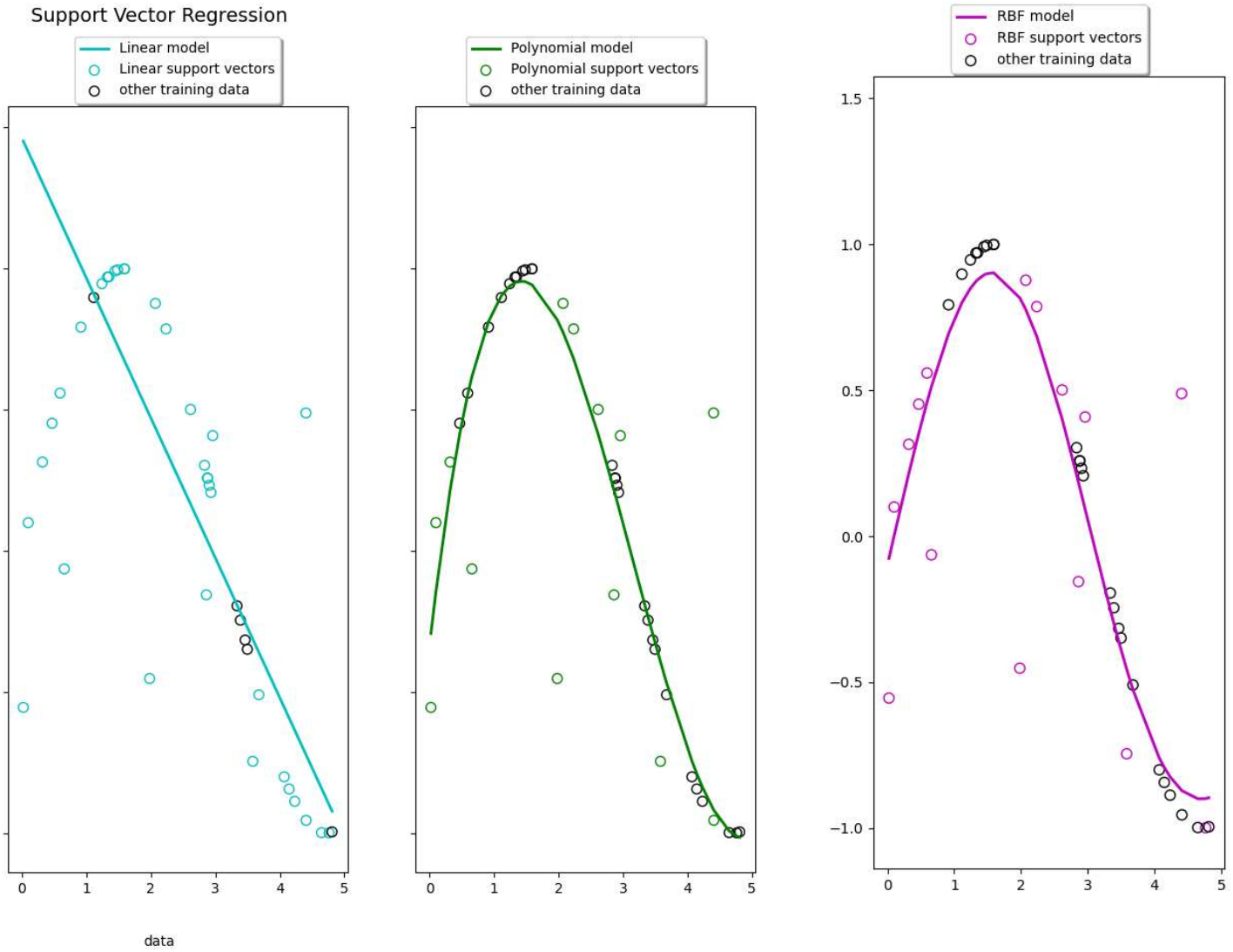


Polynomial option of SVM

- In 3D it would be easy to split this data

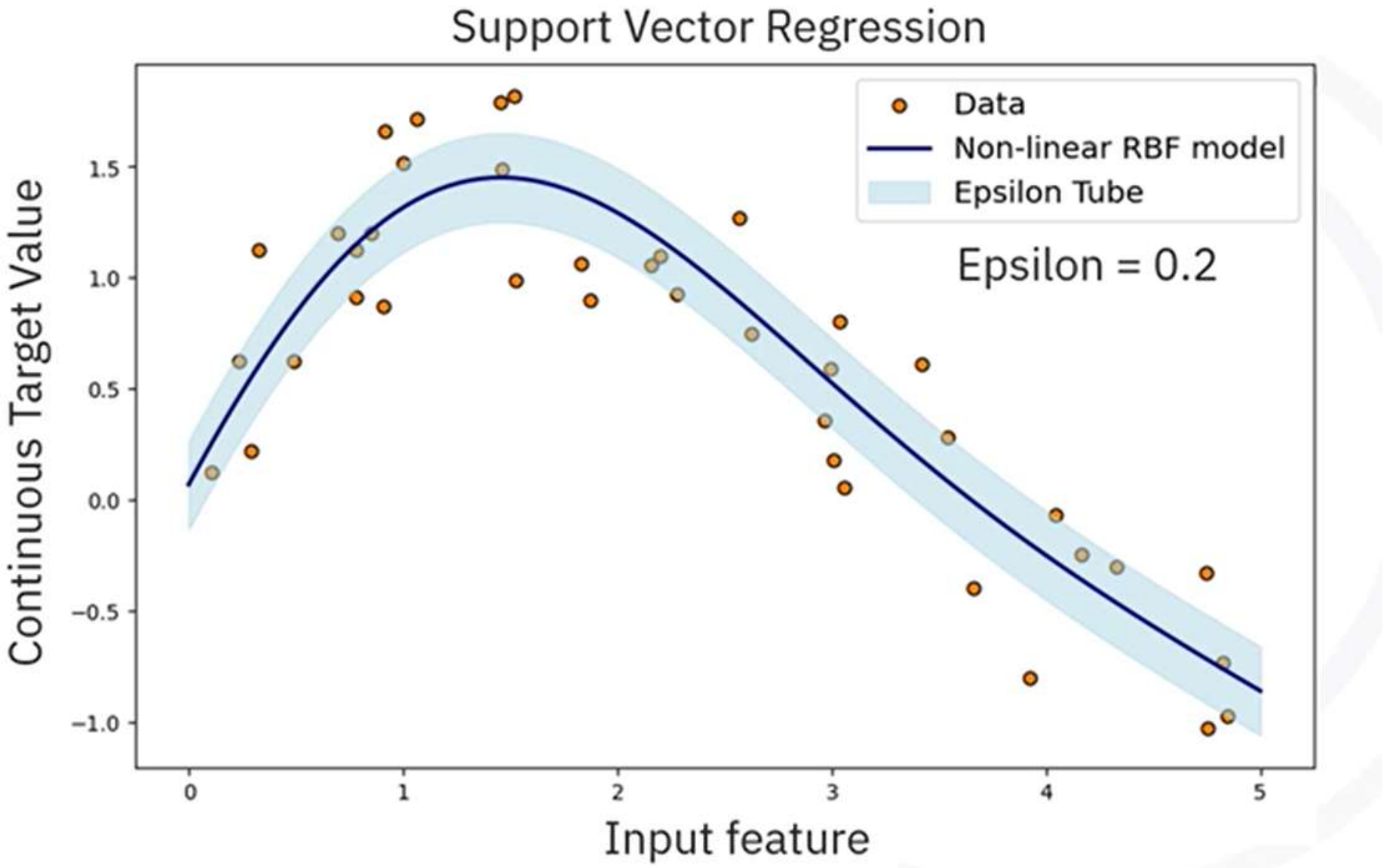
Courtesy Joseph Santarcangelo & Jeff Grossman

SVM can work with any basic function



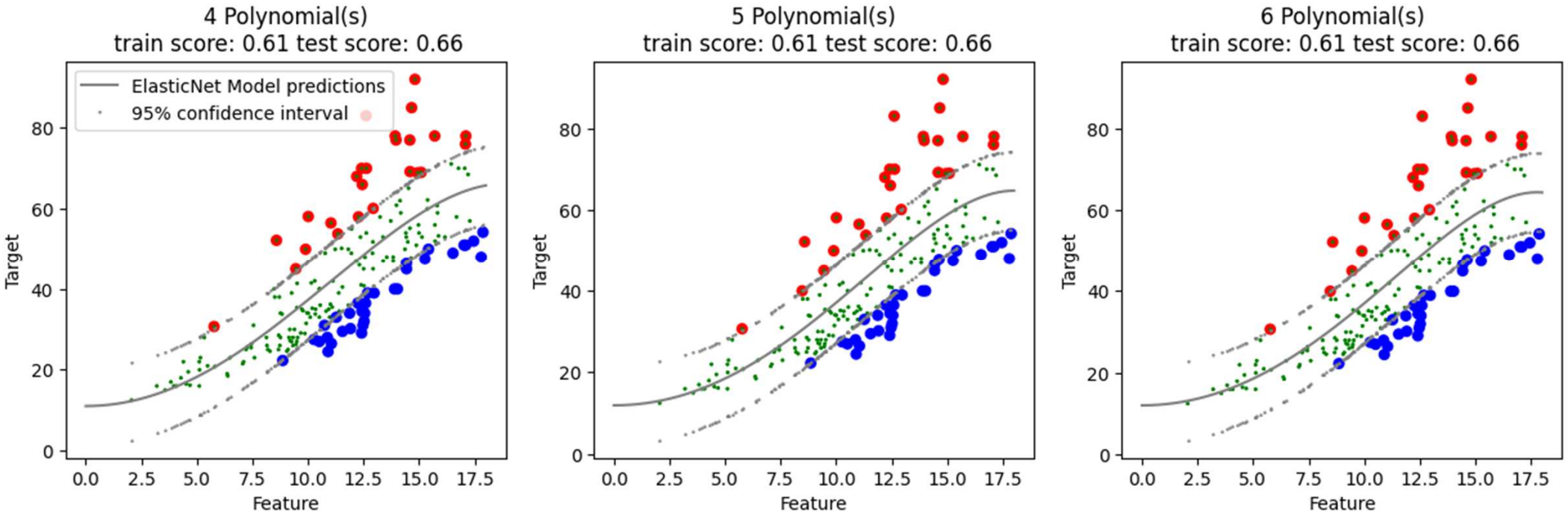
Support Vector Regression

SVM works with a distance ε



Courtesy Joseph Santarcangelo & Jeff Grossman

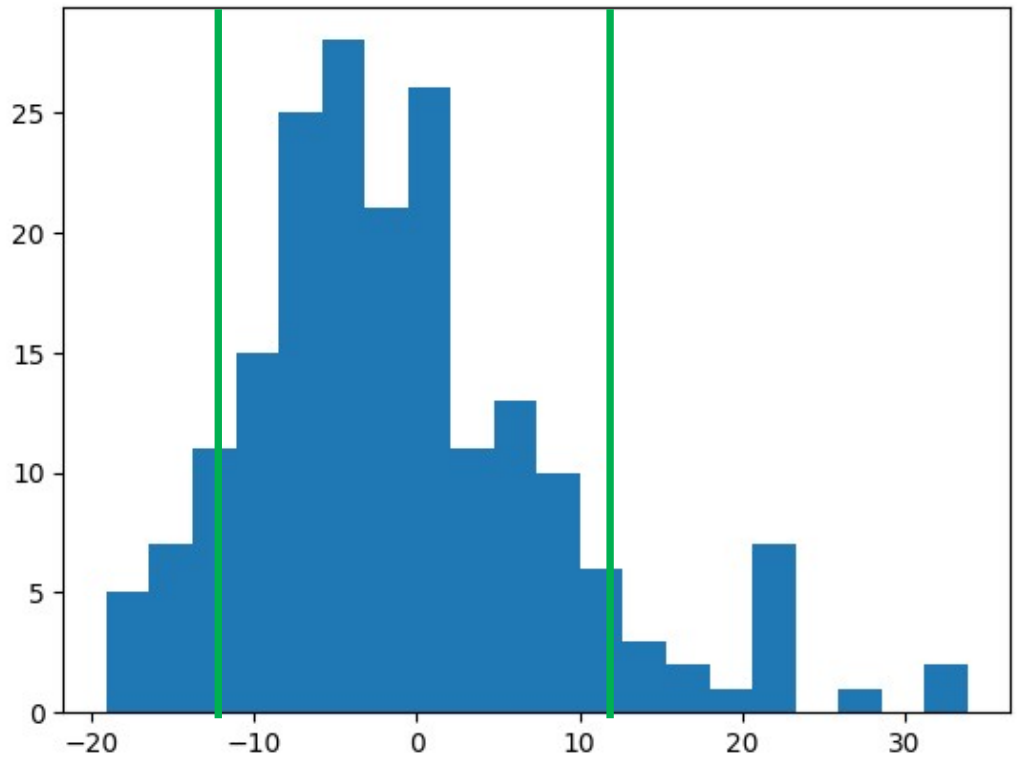
Expert knowledge



12% above the 95% confidence interval
16% below the 95% confidence interval

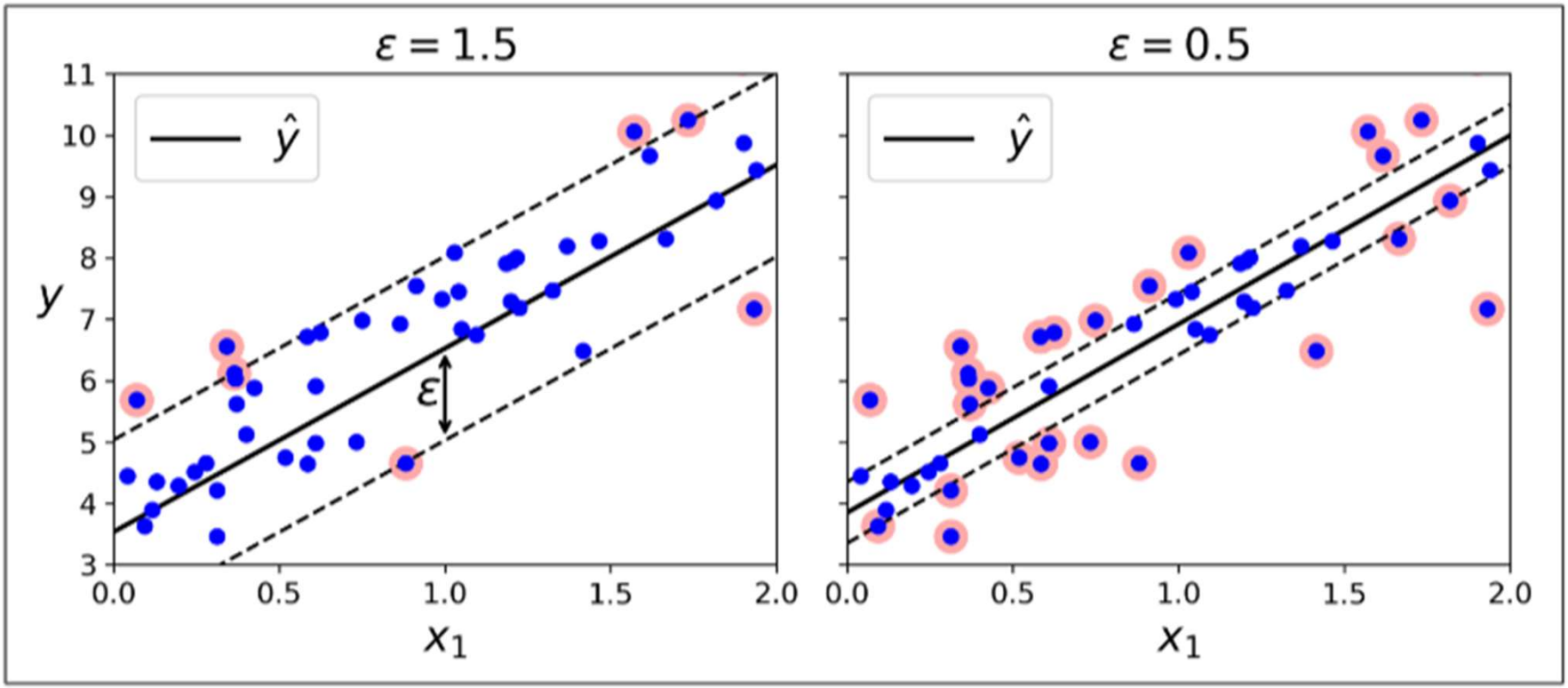
Radial basis functions

$x_1 < -\varepsilon$ $x_1 > \varepsilon$



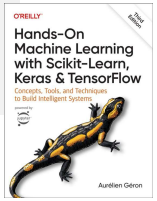
RBF uses a Gaussian kernel, so $\boldsymbol{\varepsilon} \approx$ the standard deviation σ which defines the confidence interval

Linear SVM Regression

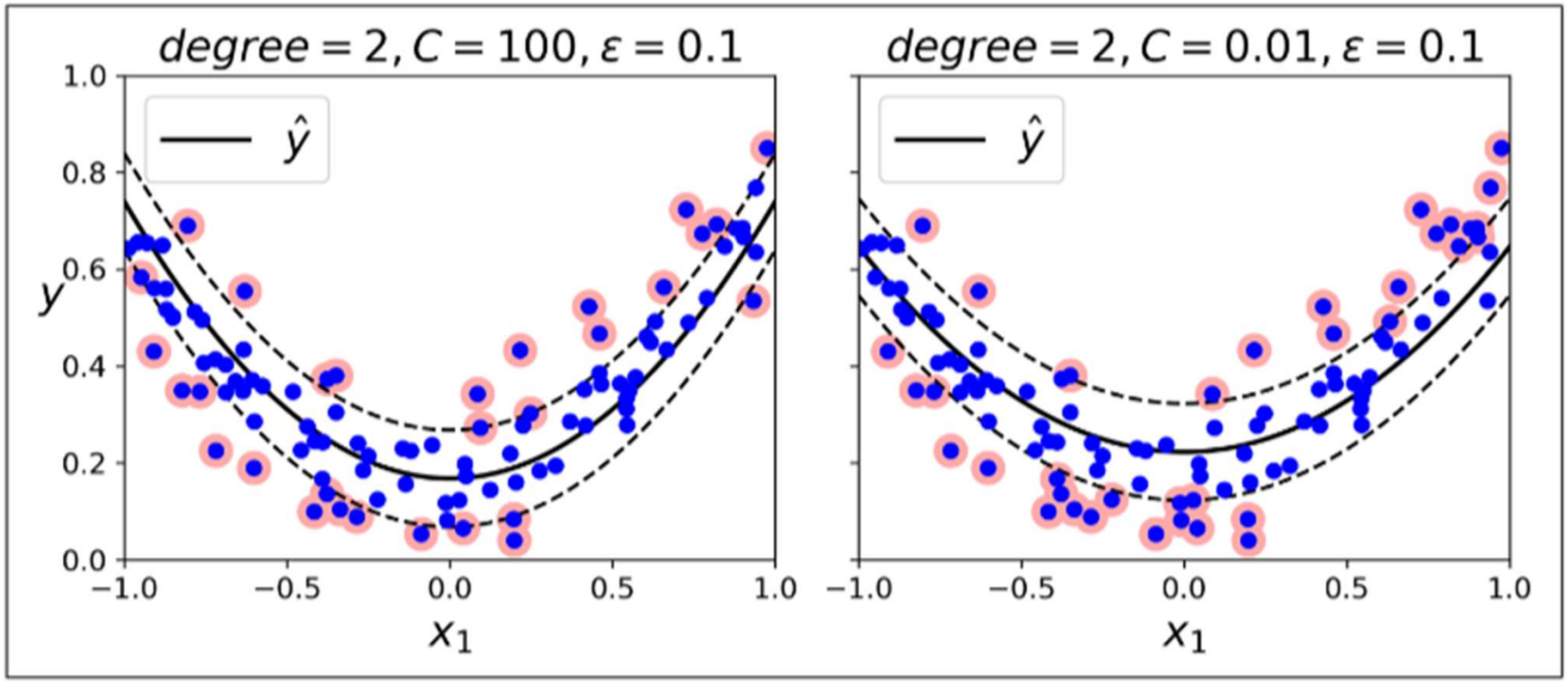



```
from sklearn.svm import LinearSVR

svm_reg = LinearSVR(epsilon=1.5)

svm_reg.fit(X, y)
```

```
svm_reg = LinearSVR(epsilon=0.5)
```

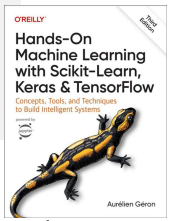


Polynomial SVM Regression



2nd-degree polynomial kernel

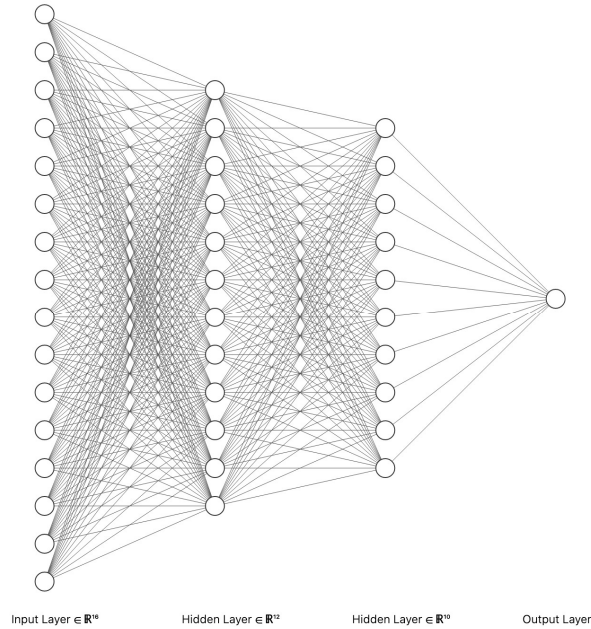
```
from sklearn.svm import SVR

svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)

svm_poly_reg.fit(X, y)
```



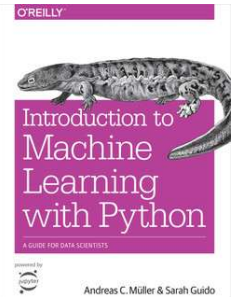
Neural Nets for Regression

MLP Regressor

- Scikit-learn has support for Neural Networks, but only for Multi-Layer Perceptrons.

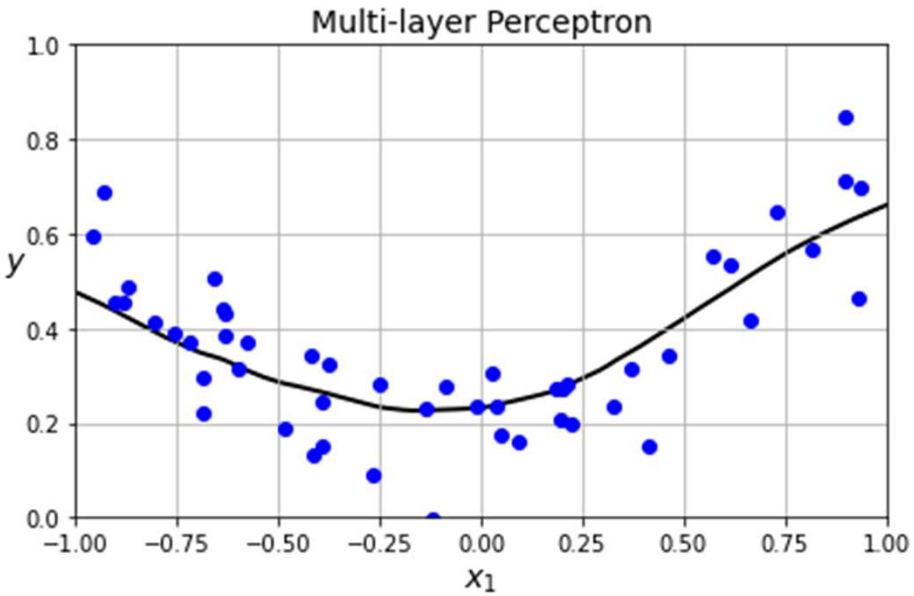


Two hidden layers – (12,10) nodes



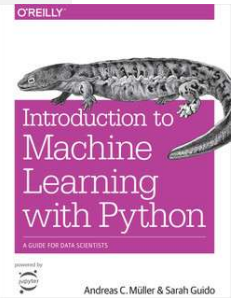

```
from sklearn.neural_network import MLPRegressor
```

Multi-Layer Perceptron Regression



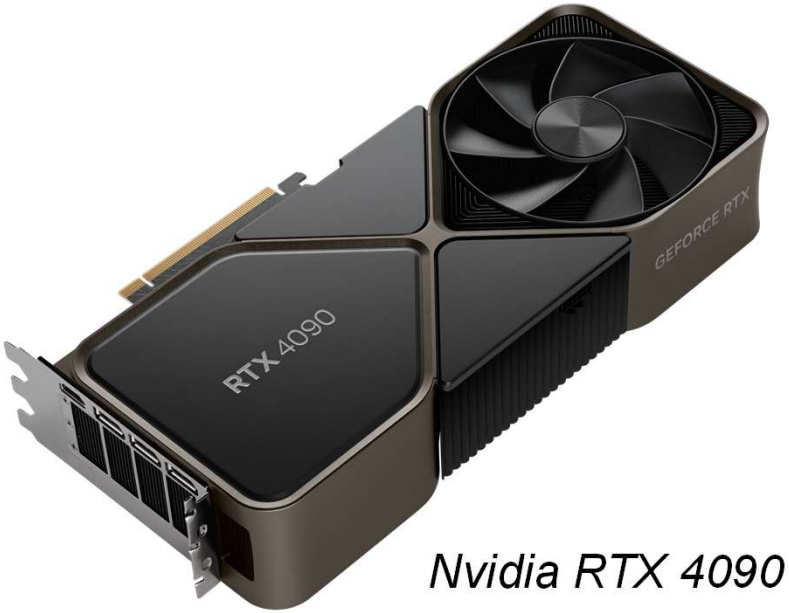
One hidden layer – 100 nodes

```
from sklearn.neural_network import MLPRegressor

mlp_poly_reg = (random_state=1, max_iter=2000, tol=0.1)

mlp_poly _reg.fit(X, y)
```



Matrix operations

PyTorch



Nvidia RTX 4090

scikit learn

GPU support

- Since scikit-learn version 1.8, gpu-support is possible for array-api and PyTorch operations
- Array API support was added to the following estimators: preprocessing.StandardScaler, preprocessing.PolynomialFeatures, linear_model.RidgeCV, linear_model.RidgeClassifierCV, mixture.GaussianMixture and calibration.CalibratedClassifierCV.

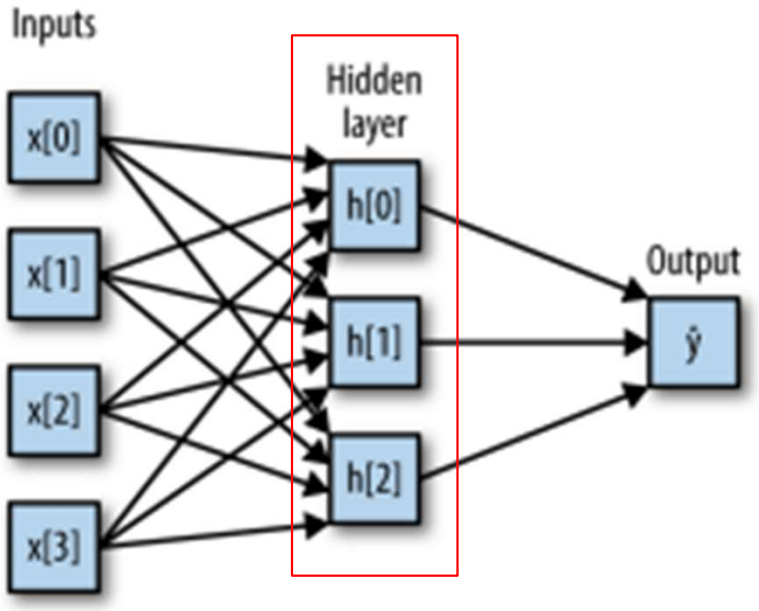
10x as fast

For small NN, works also fine on CPU

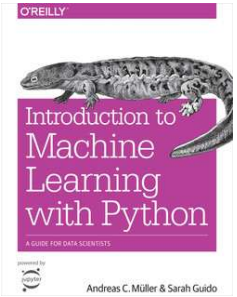
Prediction

- Perceptron regression

$$\hat{y} = w[0] * h[0] + w[1] * h[1] + \ldots + w[p] * h[p] + b$$



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016



Artificial Neural Networks

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH and WALTER H. PITTS

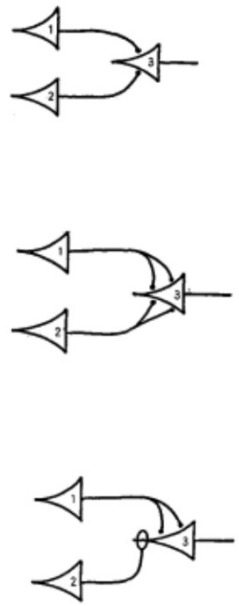
Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic.

. . .

Let c_i be any neuron of $\mathfrak{N}$ with a threshold $\theta_i > 0$, and let c_{i1}, $c_{i2}, \ldots, c_{ip}$ have respectively $n_{i1}, n_{i2}, \ldots, n_{ip}$ excitatory synapses upon it. Let $c_{j1}, c_{j2}, \ldots, c_{jq}$ have inhibitory synapses upon it. Let κ_i be the set of the subclasses of $\{n_{i1}, n_{i2}, \ldots, n_{ip}\}$ such that the sum of their members exceeds θ_i. We shall then be able to write, in accordance with the assumptions mentioned above,

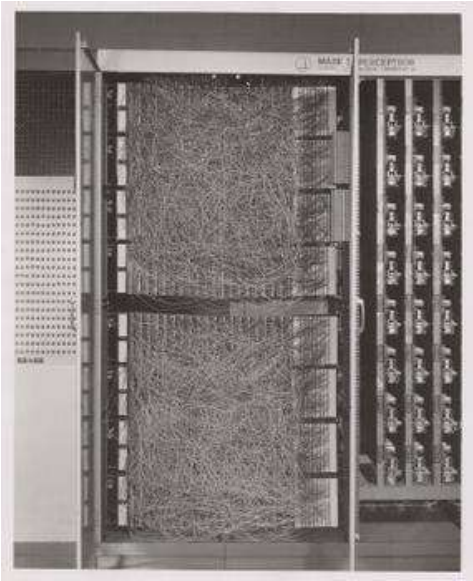
$$N_i(z_1) \cdot \equiv \cdot S\left\{\prod_{m=1}^{q} \sim N_{jm}(z_1) \cdot \sum_{\alpha \in K_i} \prod_{s \in \alpha} N_{is}(z_1)\right\} \qquad (1)$$

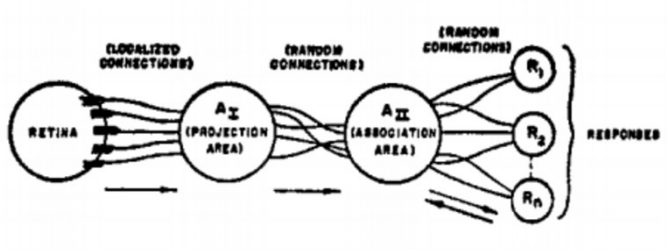
where the '$\sum$' and '$\prod$' are syntactical symbols for disjunctions and conjunctions which are finite in each case.

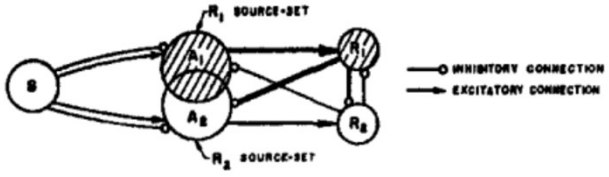


Warren S. McCulloch and Walter Pitts, "A logical calculus of the ideas immanent in nervous activity", The bulletin of mathematical biophysics, Dec. 1943, Volume 5, Issue 4, pp 115–133.

Perceptron







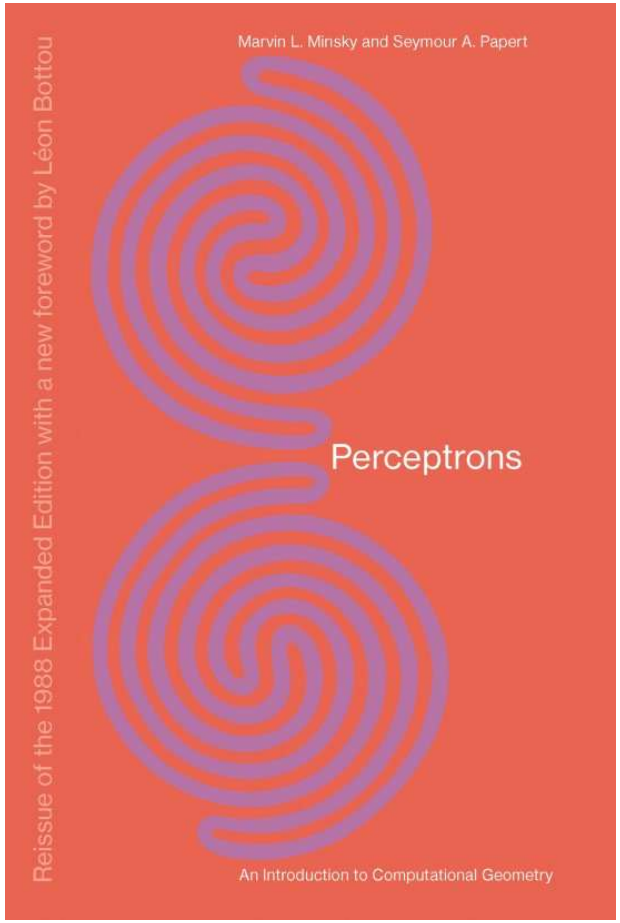
$$P_a = \sum_{e=\theta}^{x} \sum_{i=\theta}^{min(y,\, e-\theta)} P(e,i)$$

(The quantities e and i are the excitatory and inhibitory components of the excitation received by the A-unit from the stimulus. If the algebraic sum $\alpha = e + i$ is equal to or greater than θ, the A-unit is assumed to respond.)

The "Mark 1 perceptron" machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors

Rosenblatt, F. (1958). “The perceptron: A probabilistic model for information storage and organization in the brain.” Psychological Review, 65(6), 386-408.

Single layer perceptrons

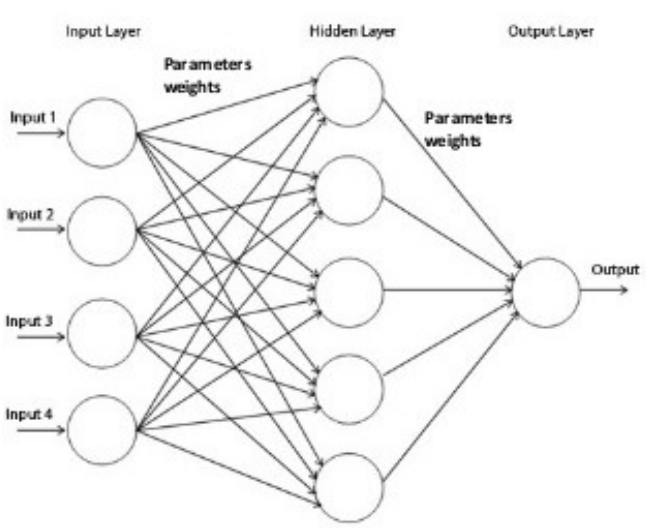


1969

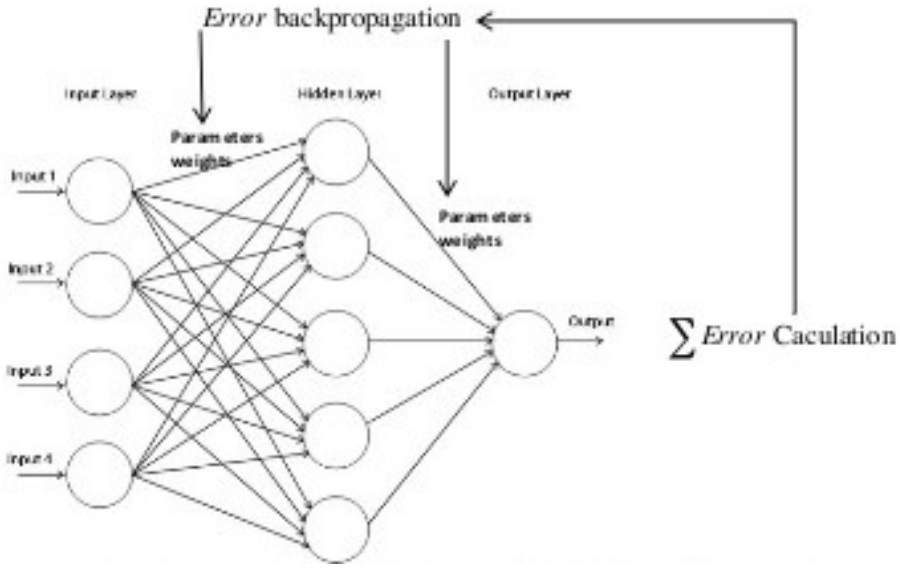
Single layer perceptrons can only linearly separable patterns.

Multi-layer perceptrons can learn any function, yet MLPs are harder to train.

Backpropagation in Neural Networks

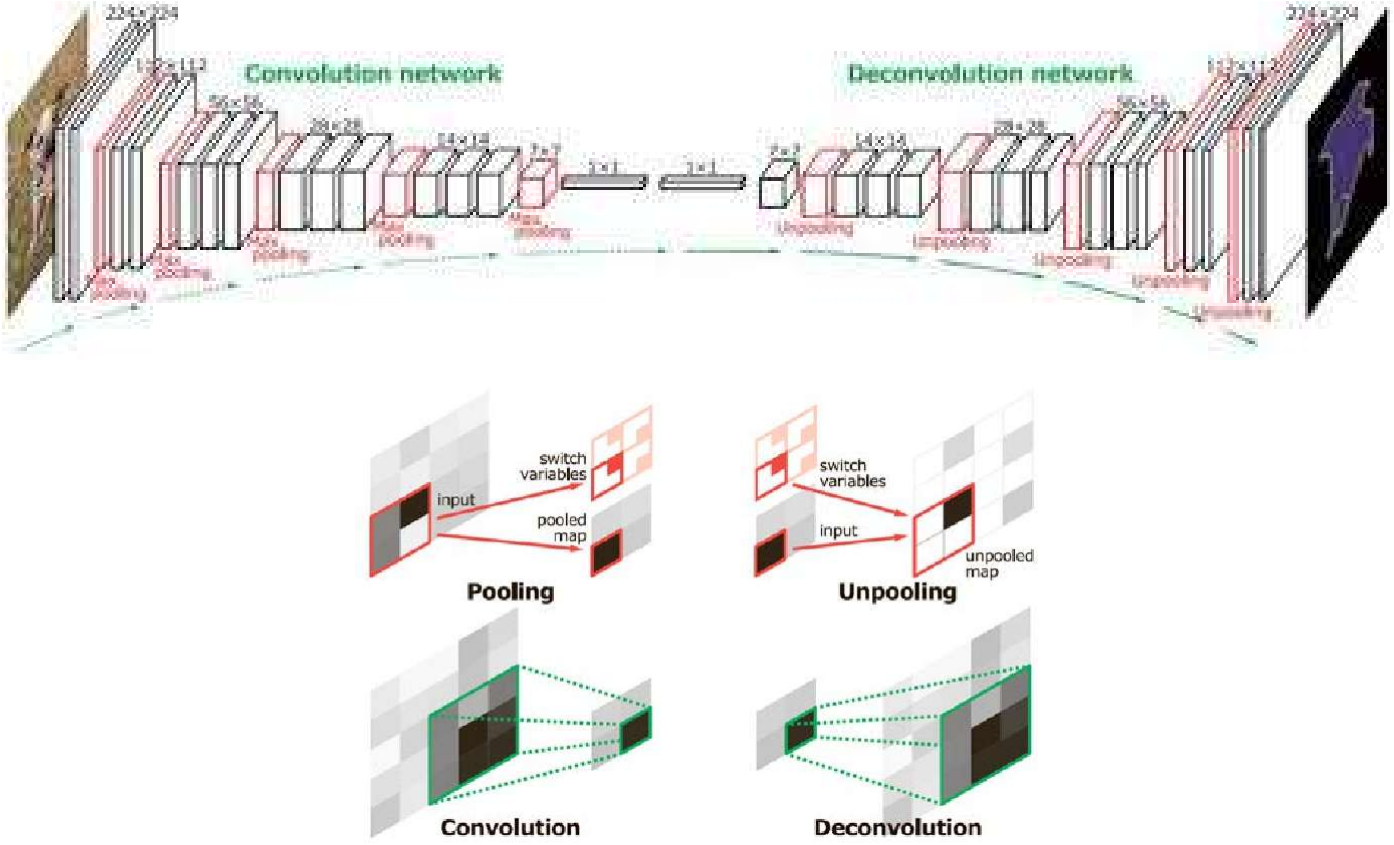


Two-layer feedforward neural network



David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams (1965). “Learning representations by back-propagating errors.” Nature 323, pages 533–536.

Convolutional Neural Nets



No CNNs are part of scikit-learn. Moving to Keras or PyTorch is a good choice, combined with preprocessing of scikit-learn

Keras

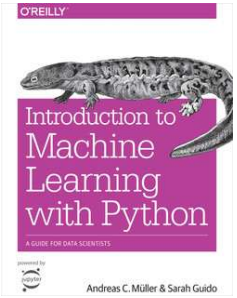
PyTorch

Introduction to Machine Learning

- 2.3.7 Kernelized Support Vector Machines

- Strengths:
  - Effective in high-dimensional spaces
  - Robust to overfitting
  - Works with data that needs more complex models (combination of features)
- Weakness:
  - Slow for large datasets
  - Sensitive to kernel and regularization parameters

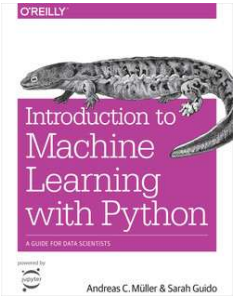
Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016



Introduction to Machine Learning

- Neural Networks – Multi-Layer Perceptrons

- Strengths:
  - Can fit everything
  - Can fit large datasets
- Weakness:
  - Black-box solution
  - Many possible # layers & parameters – hyperparameters
  - Relative slow to train

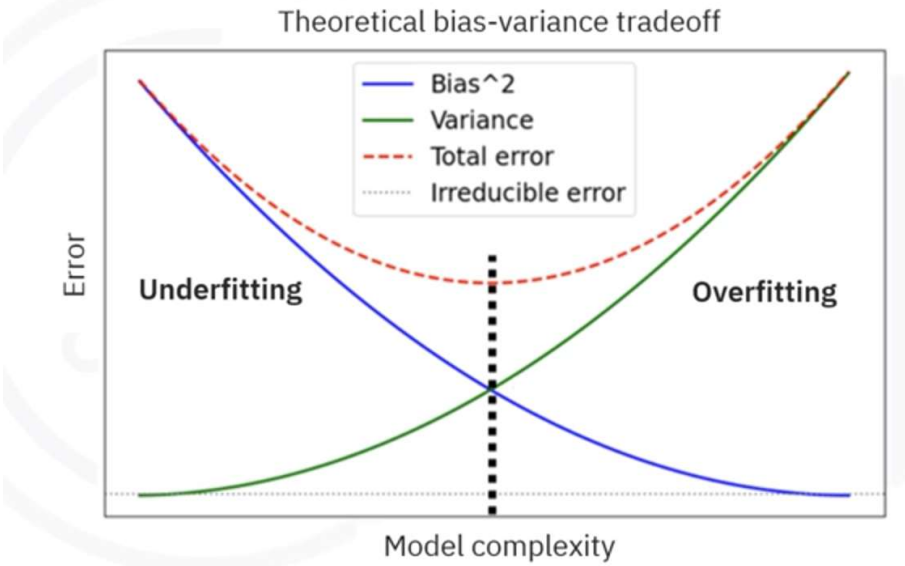


Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

Conclusion

Learning outcomes of this course covered today

- SVMs can be used to make the model more complex
- Ensembles could be used to make the model less complex
- Hidden layers of add exponential more complexity



Feature Engineering

Generalization