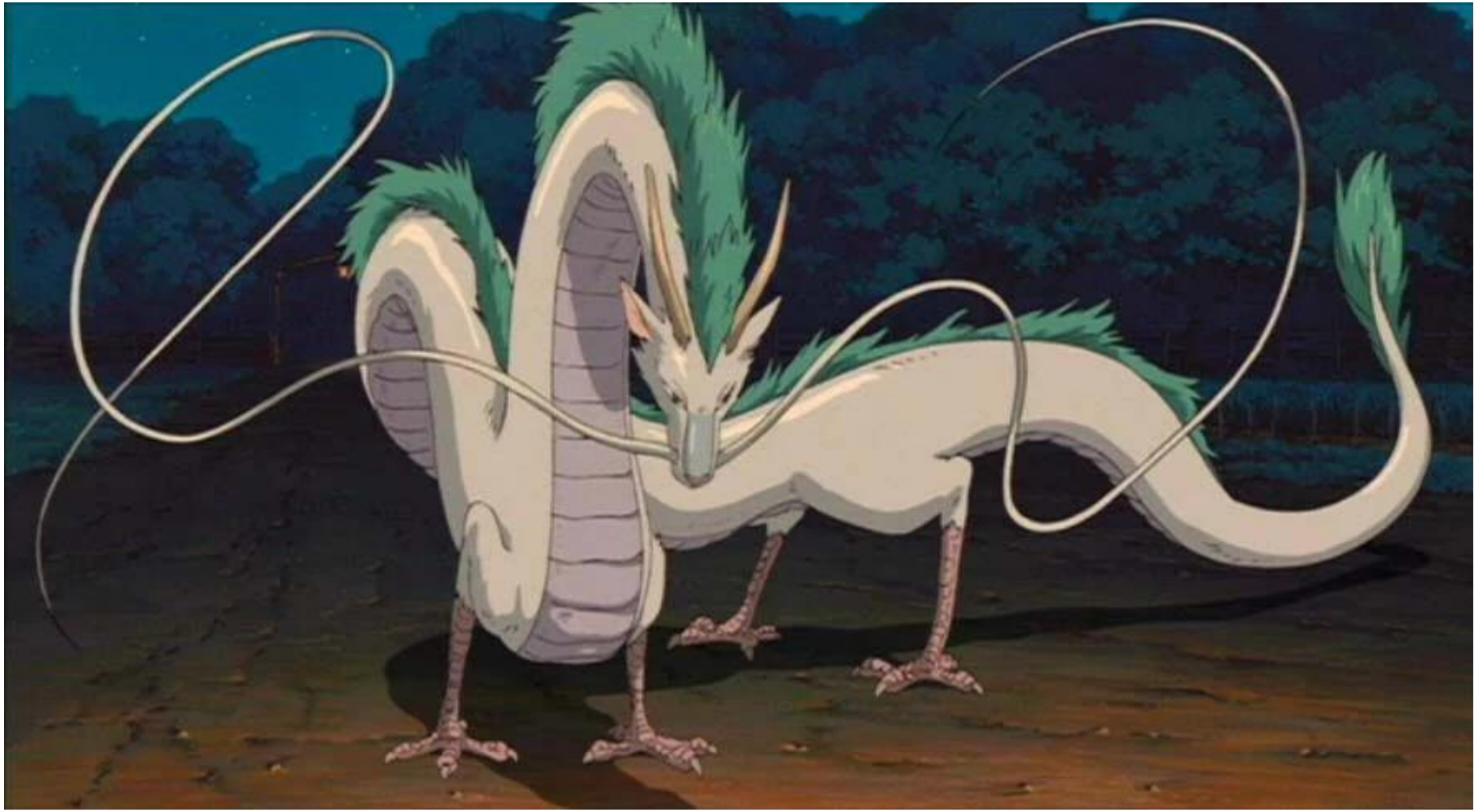
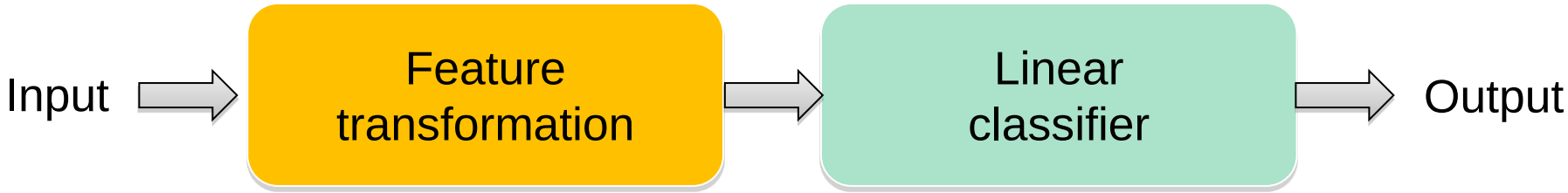
# Nonlinear classifiers, bias-variance tradeoff

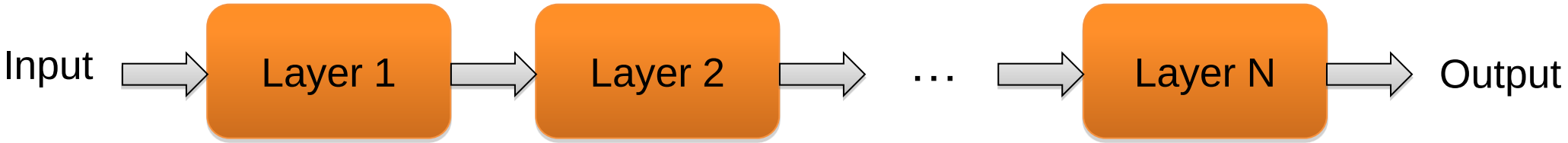


# From linear to nonlinear classifiers

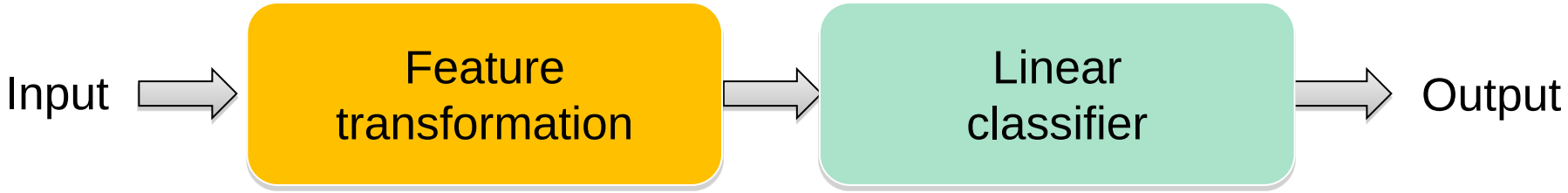
- To achieve good accuracy on challenging problems, we need to be able to train *nonlinear* models
- Two strategies for making nonlinear predictors out of linear ones:
  - **"Shallow" approach:** nonlinear feature transformation followed by linear classifier




  - **"Deep" approach:** stack multiple layers of linear predictors (interspersed with nonlinearities)



# Shallow approach: Nonlinear SVMs



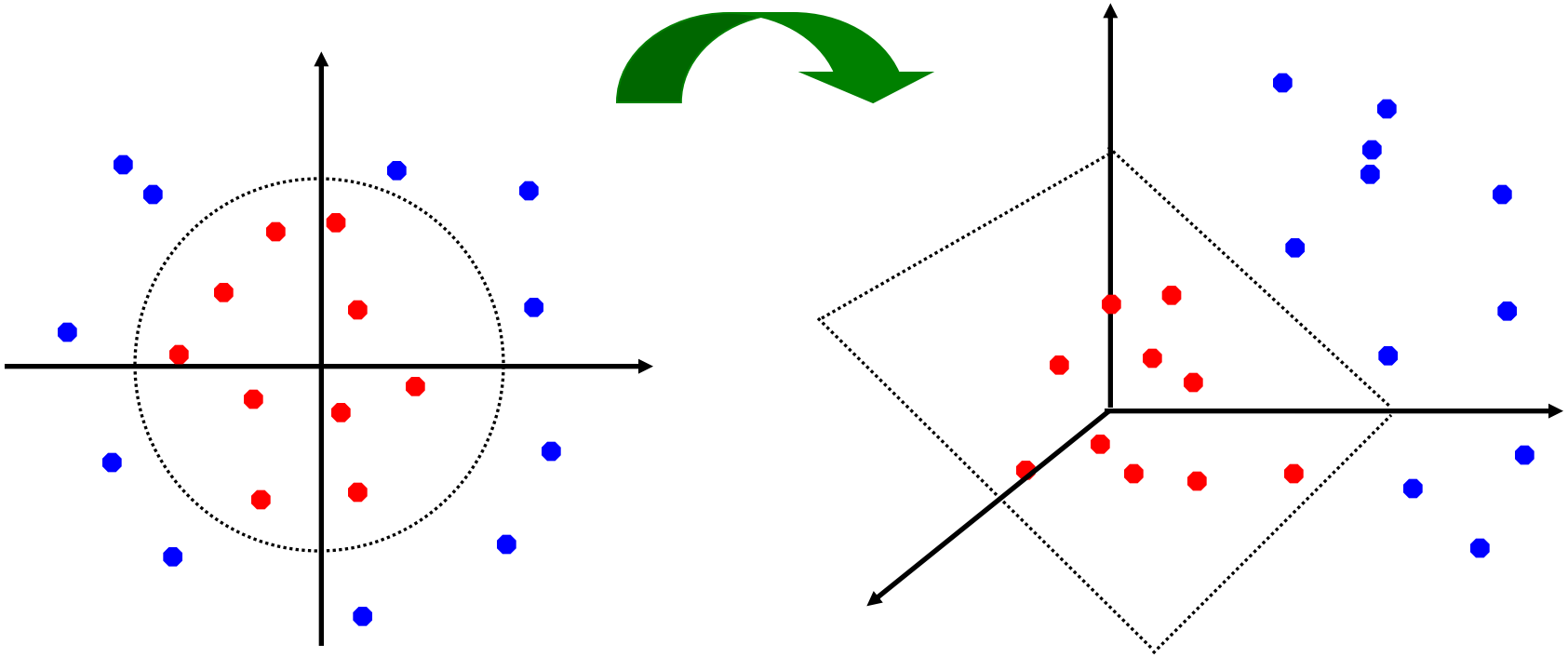



Image credit: Andrew Moore

# Nonlinear SVMs

- General idea: the original feature space can be mapped to some higher-dimensional space where the training data is separable
  - Because of the special properties of SVM optimization, this can be done without explicitly performing the lifting transformation

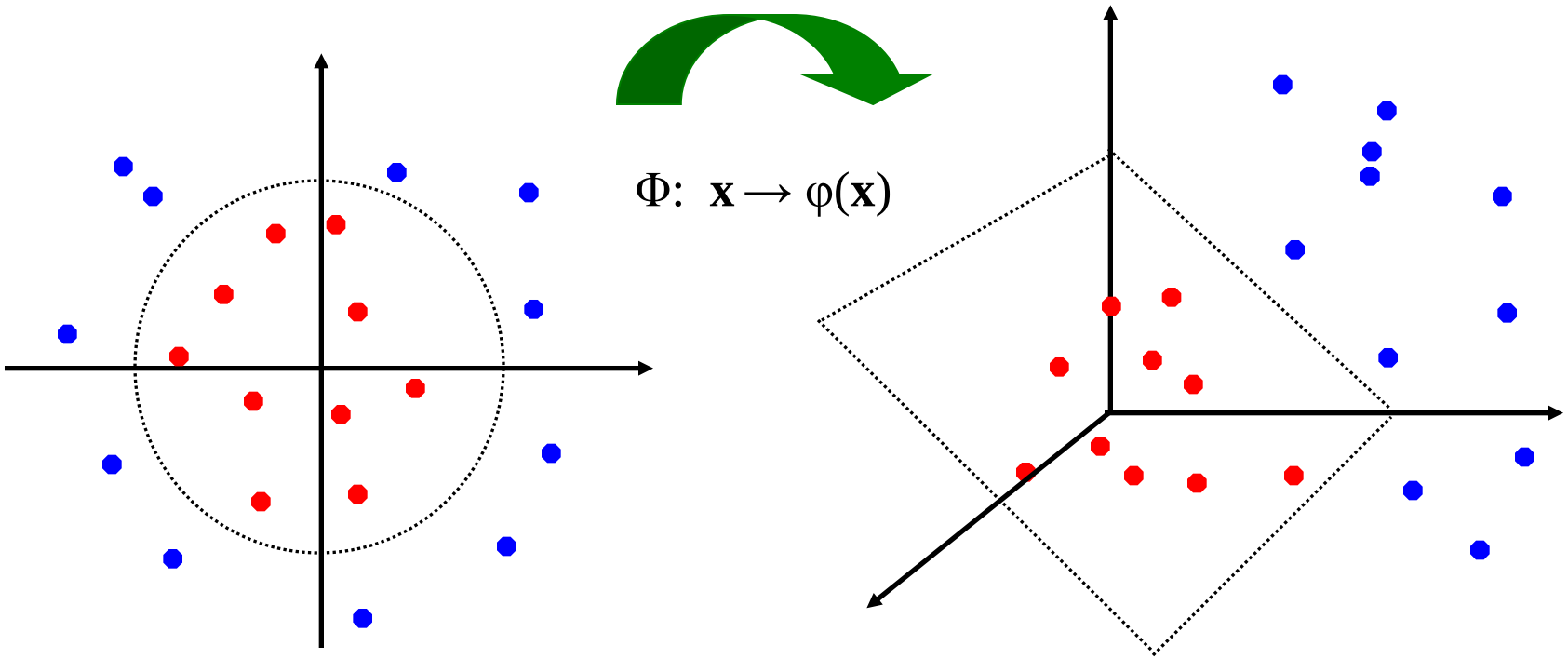



Image credit: Andrew Moore

# Dual SVM formulation

- Directly solving the SVM objective for $w$ is called the *primal* approach:

$$\arg\min_w \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{n} \max[0, 1 - y_i w^T x_i]$$

- An equivalent formulation is solve a *dual* optimization problem over Lagrange multipliers $\alpha_i$ associated with individual training points. This gives a classifier of the form

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i x_i^T x \ .$$

  - At the optimum, $\alpha_i$ are nonzero only for *support vectors*
  - In the dual optimization algorithm, training points appear only inside dot products $x_i^T x_j$ and this enables nonlinear SVMs via the *kernel trick*

# Kernel SVMs

- *The kernel trick*: instead of explicitly computing the lifting transformation $\varphi(x)$, define a *kernel function*

$$K(x, x') = \varphi(x)^T \varphi(x')$$

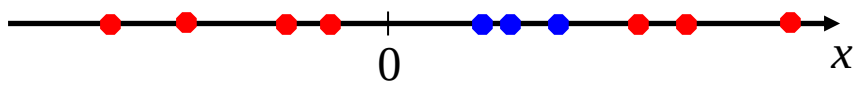
  - To be valid, the kernel function must satisfy *Mercer's condition* (kernel matrices have to be positive-definite and symmetric)

- This gives a nonlinear decision boundary in the original feature space:

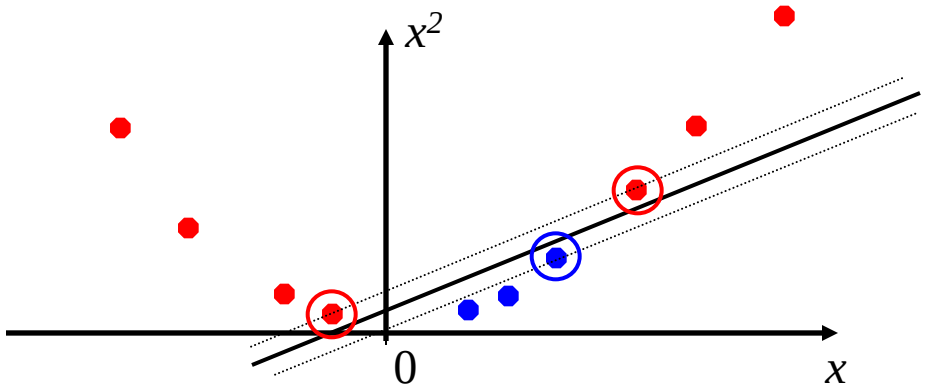
$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \varphi(x_i)^T \varphi(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x)$$

# Example

- Non-separable data in 1D:




- Apply mapping $\varphi(x) = (x, x^2)$:




$$\varphi(x)^T \varphi(x') = K(x, x') = xx' + x^2 x'^2$$

# Kernel example 1: Polynomial

- Polynomial kernel with degree $d$ and constant $c$:

$$K(x, x') = (x^T x' + c)^d$$

- What this looks like for $d = 2$:

$$x = (u, v), \qquad x' = (u', v')$$

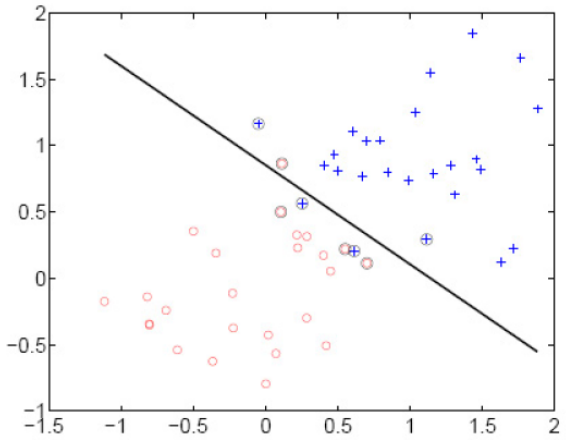
$$K(x, x') = (uu' + vv' + c)^2$$

$$= u^2u'^2 + v^2v'^2 + 2uu'vv' + cuu' + cvv' + c^2$$

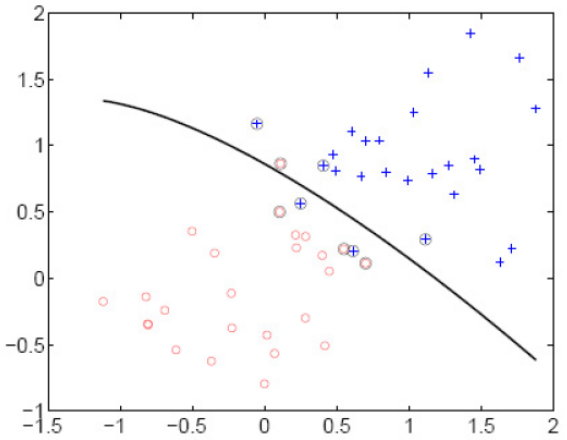
$$\varphi(x) = (u^2, v^2, \sqrt{2}uv, \sqrt{c}u, \sqrt{c}v, c)$$

- Thus, the explicit feature transformation consists of all polynomial combinations of individual dimensions of degree up to $d$

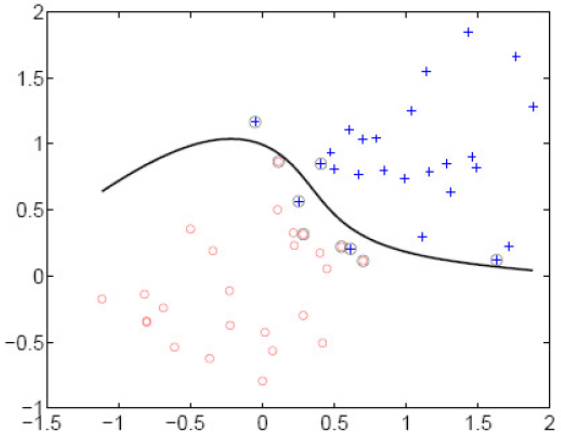
# Kernel example 1: Polynomial



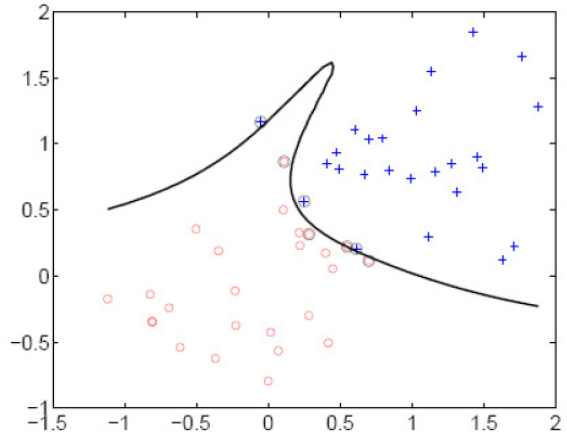
linear



$2^{nd}$ order polynomial



$4^{th}$ order polynomial



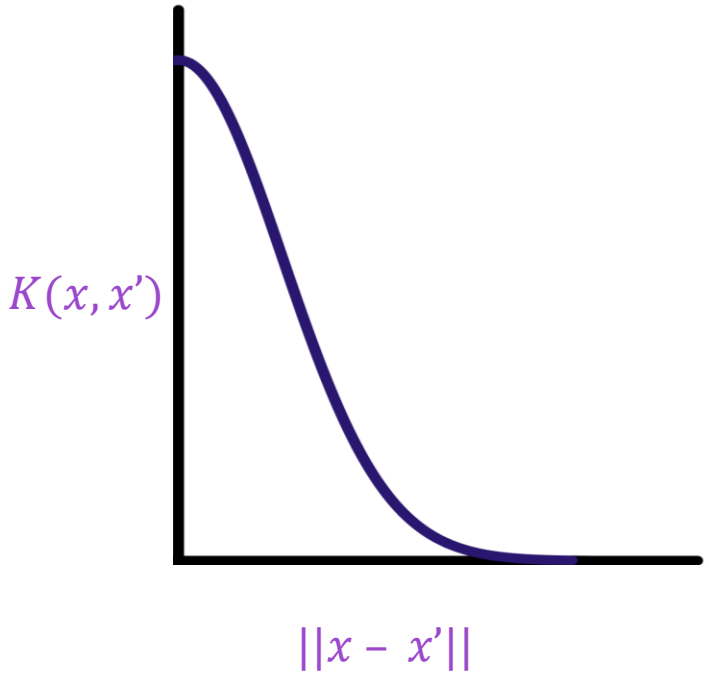
$8^{th}$ order polynomial

# Kernel example 2: Gaussian

- Gaussian kernel with bandwidth $\sigma$:

$$K(x, x') = \exp\left(-\frac{1}{\sigma^2}\|x - x'\|^2\right)$$

- Fun fact: the corresponding mapping $\varphi(x)$ is infinite-dimensional!

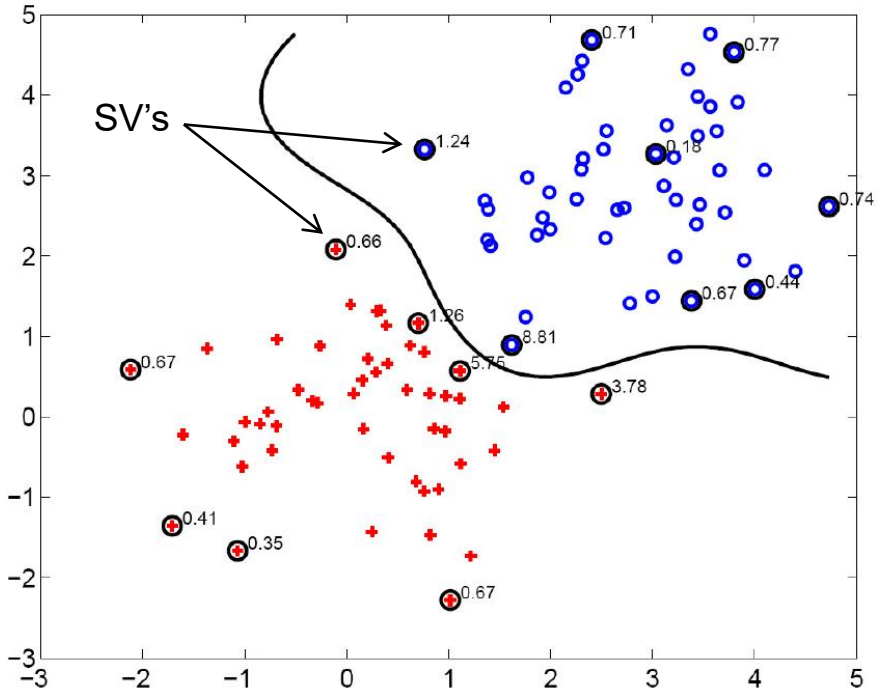


# Kernel example 2: Gaussian

- Gaussian kernel with bandwidth $\sigma$:

$$K(x, x') = \exp\left(-\frac{1}{\sigma^2}\|x - x'\|^2\right)$$

- The predictor $f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x)$ is a sum of "bumps" centered on support vectors



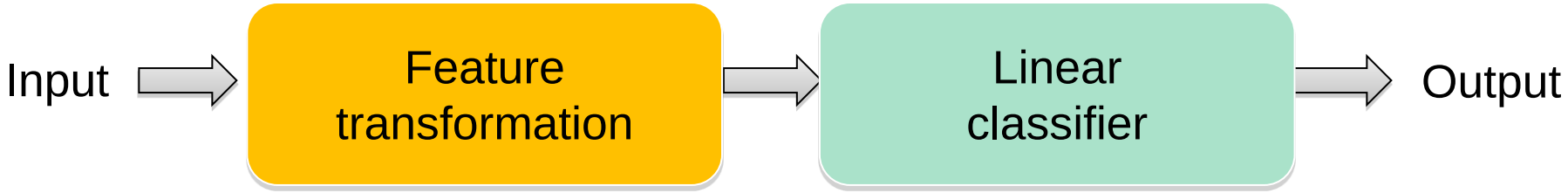

It's also called a *Radial Basis Function* (RBF) kernel

# SVM: Pros and cons

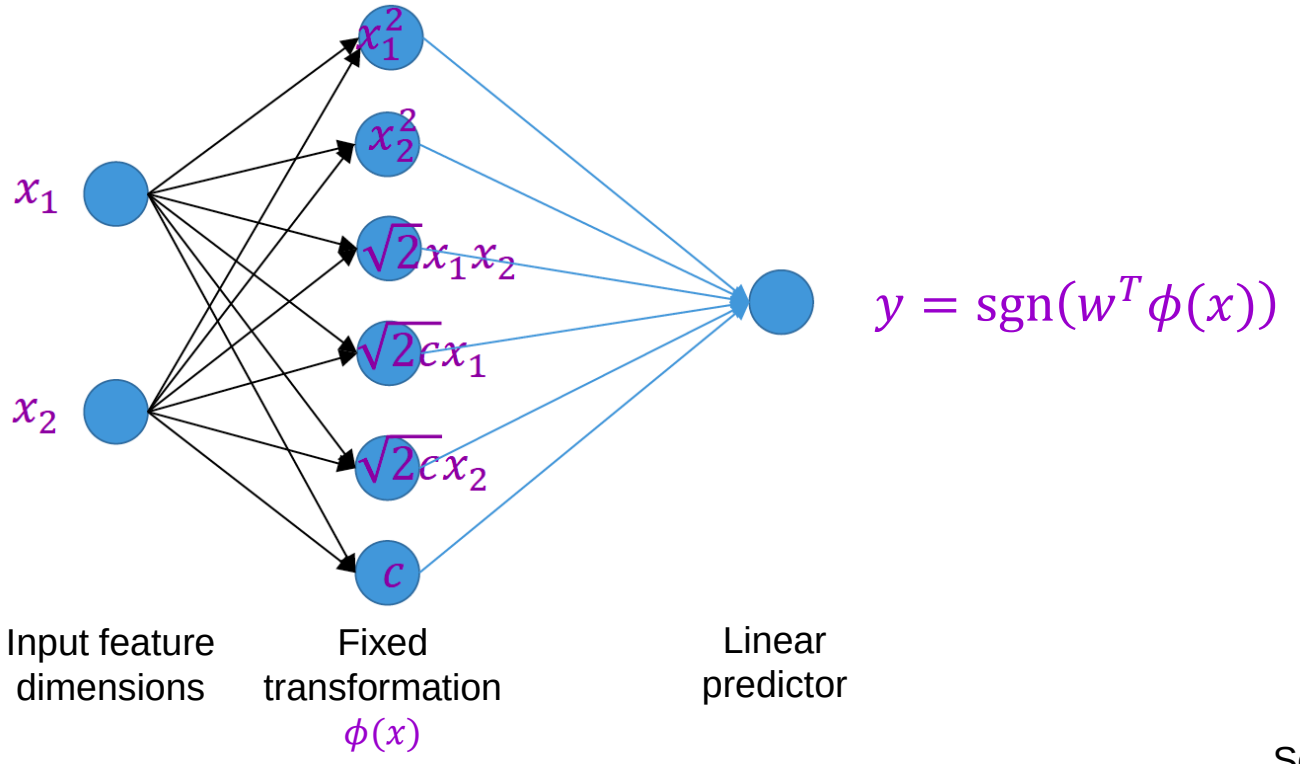
- Pros
  - Margin maximization and kernel trick are elegant, amenable to convex optimization and theoretical analysis
  - Kernel SVMs are flexible, can be used with problem-specific kernels
  - SVM loss gives very good accuracy in practice
  - Perfect “off-the-shelf” classifier, many packages are available
  - Linear SVMs can scale to large datasets

- Cons
  - Kernel SVM training does not scale to large datasets: memory cost is quadratic and computation cost even worse

# Nonlinear SVM as two-layer mapping



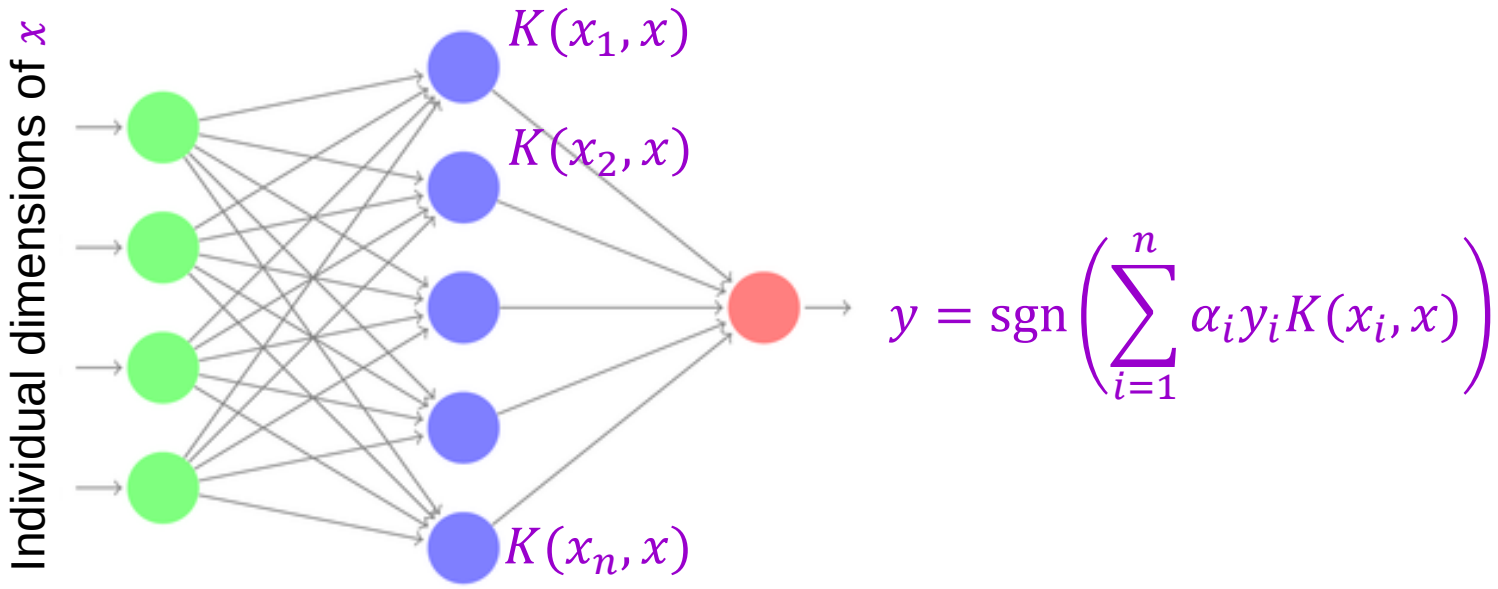

- Example: predictor for polynomial kernel of degree 2




Source: Y. Liang

# Nonlinear SVM as two-layer mapping

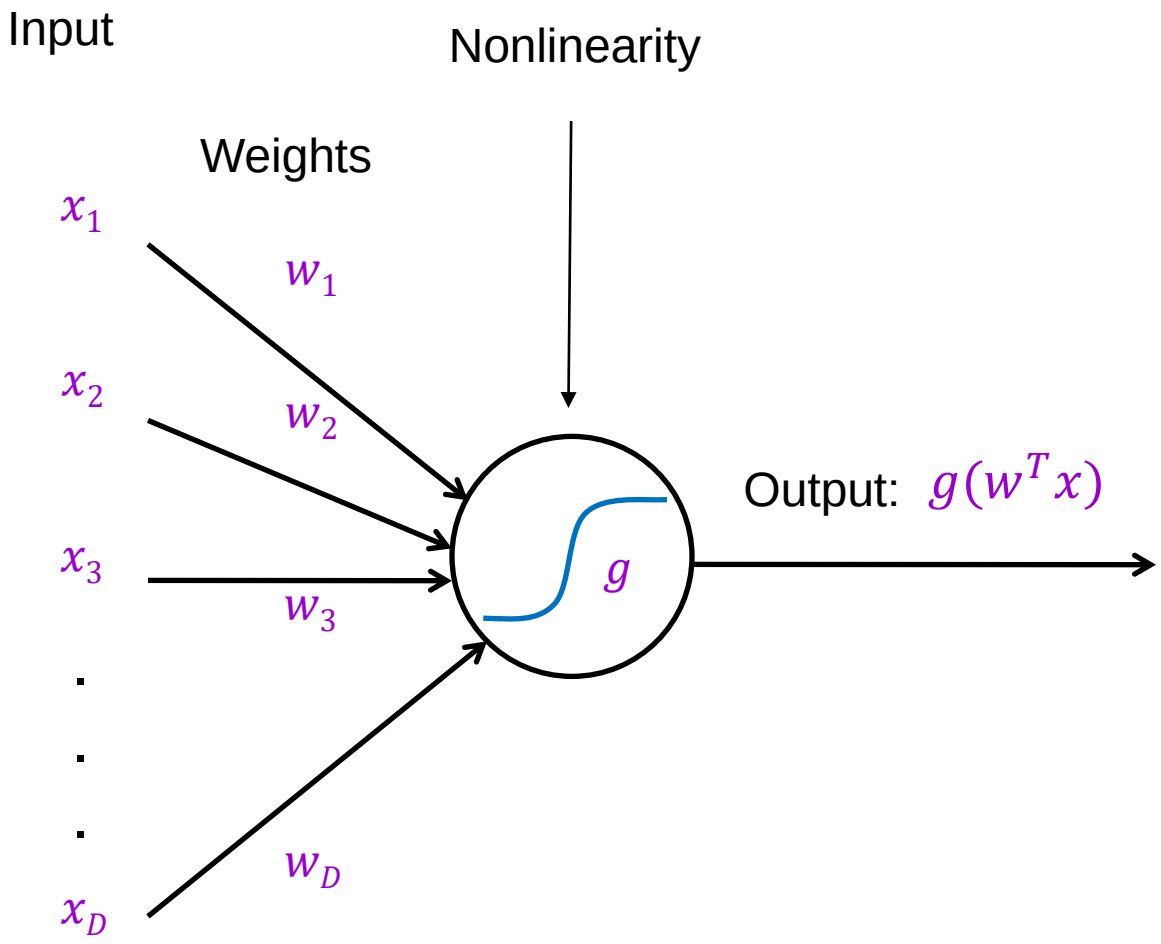
- Dual view: compute kernel function value of input with every support vector, apply linear classifier



# Multi-layer neural networks

- “Deep” approach: stack multiple layers of linear predictors (perceptrons) interspersed with nonlinearities

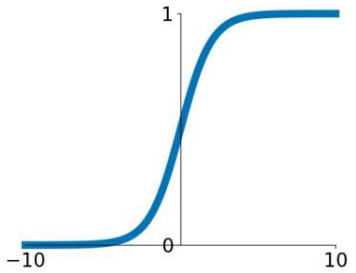
# Recall: Single perceptron



# Common nonlinearities (or *activation functions*)

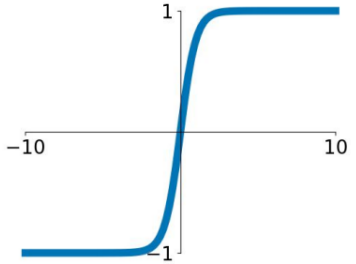
**Sigmoid**

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



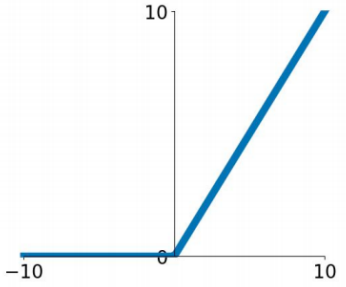

**tanh**

$$\tanh(x)$$




**ReLU**

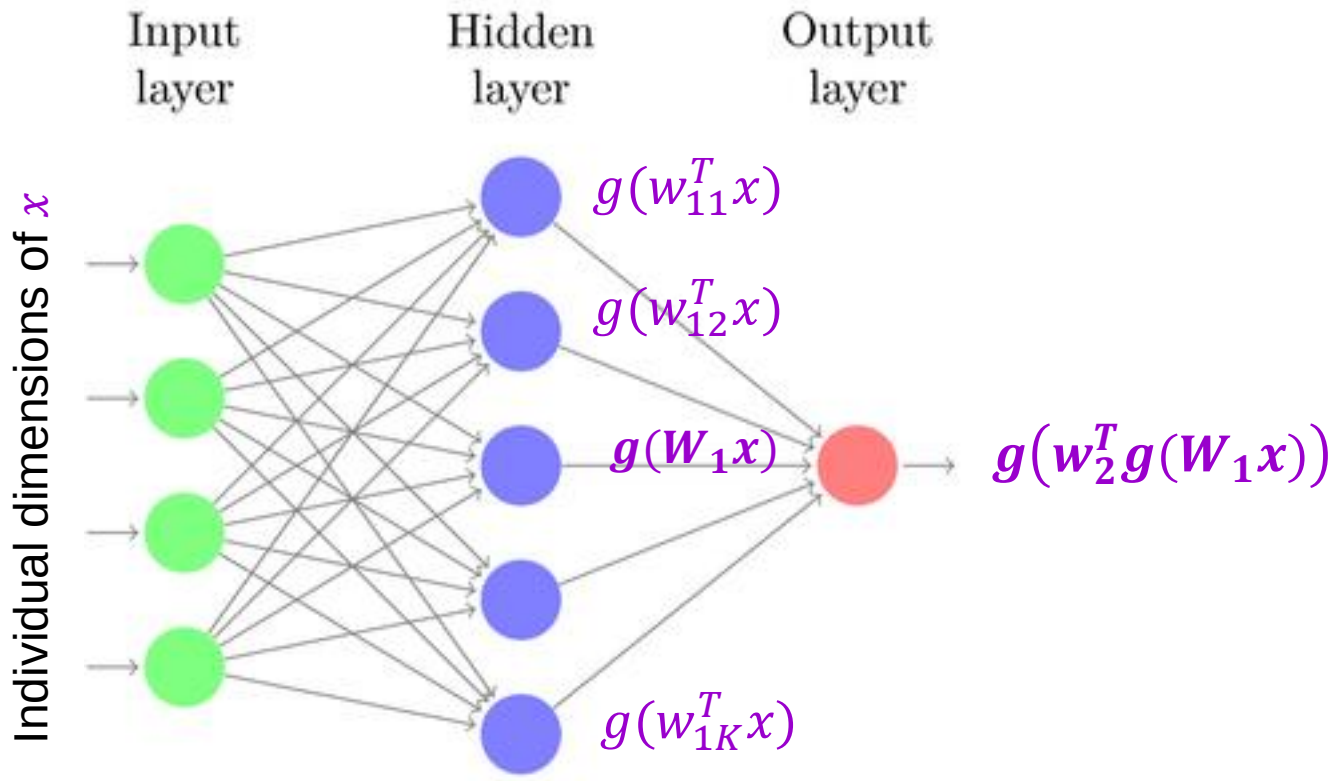
$$\max(0, x)$$




Source: Stanford 231n

# Two-layer neural network

- Introduce a *hidden layer* of perceptrons computing linear combinations of inputs followed by nonlinearities




$\boldsymbol{W_1}$ – matrix with rows $w_{1j}^T$

# Two-layer neural network

- Introduce a *hidden layer* of perceptrons computing linear combinations of inputs followed by nonlinearities
- The bigger the hidden layer, the more expressive the model
- A two-layer network is a *universal function approximator*
  - But the hidden layer may need to be huge

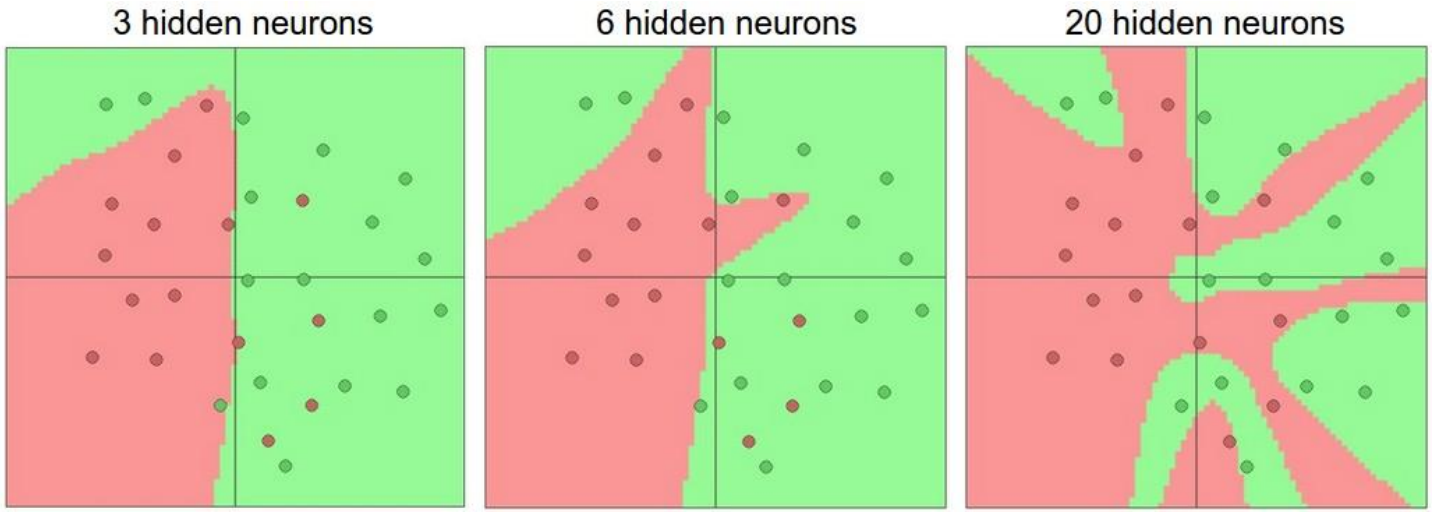
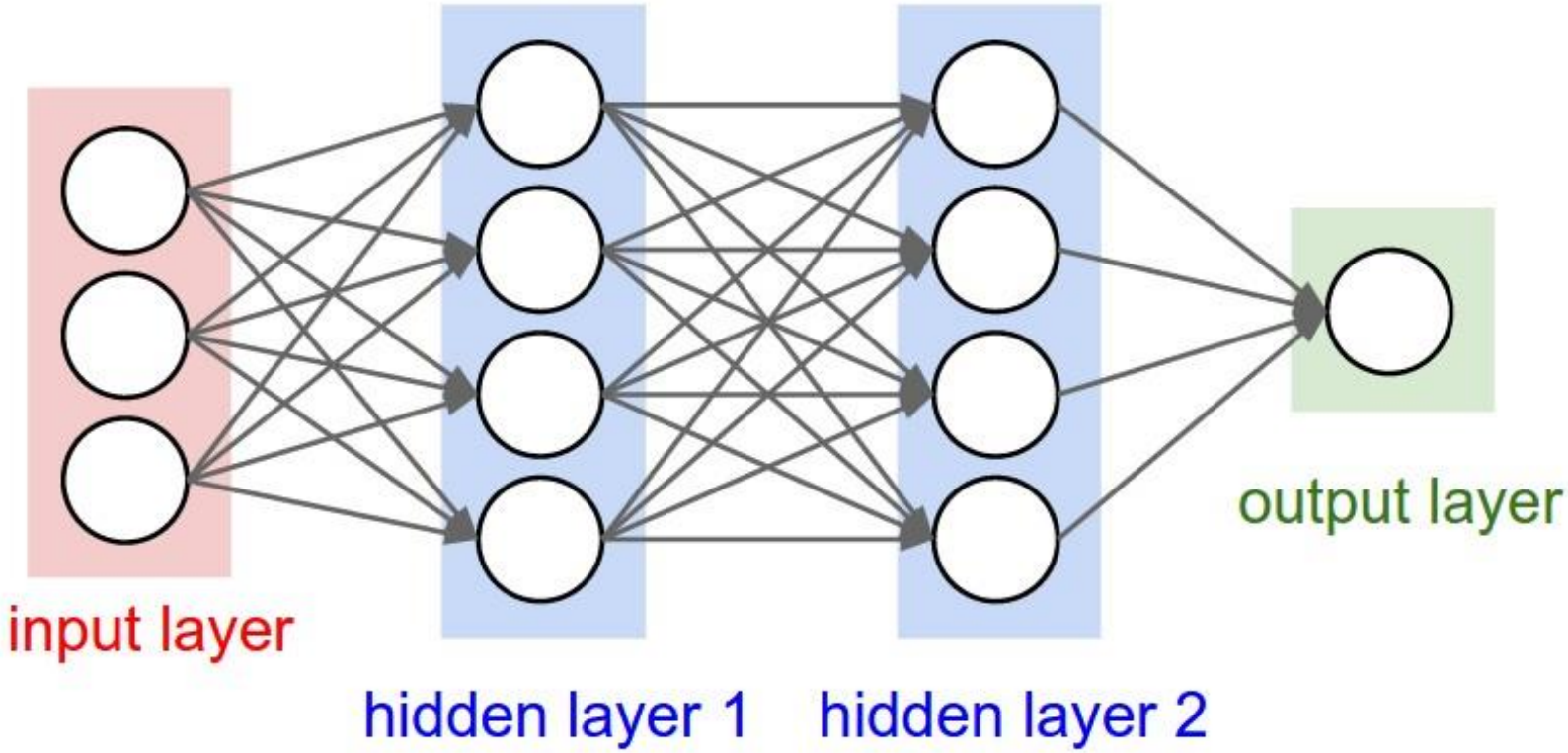



Figure source

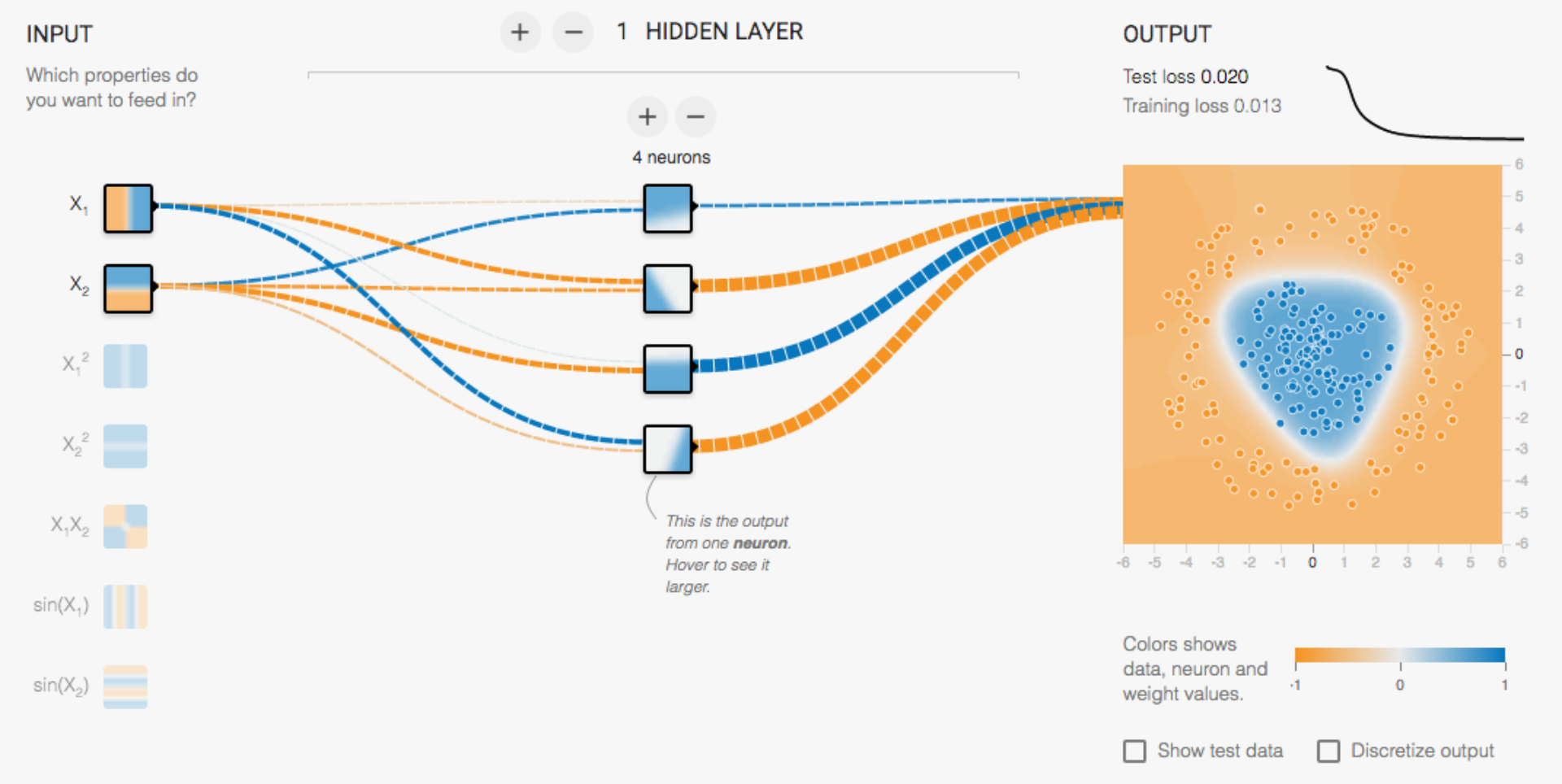
# Beyond two layers



# “Deep” pipeline

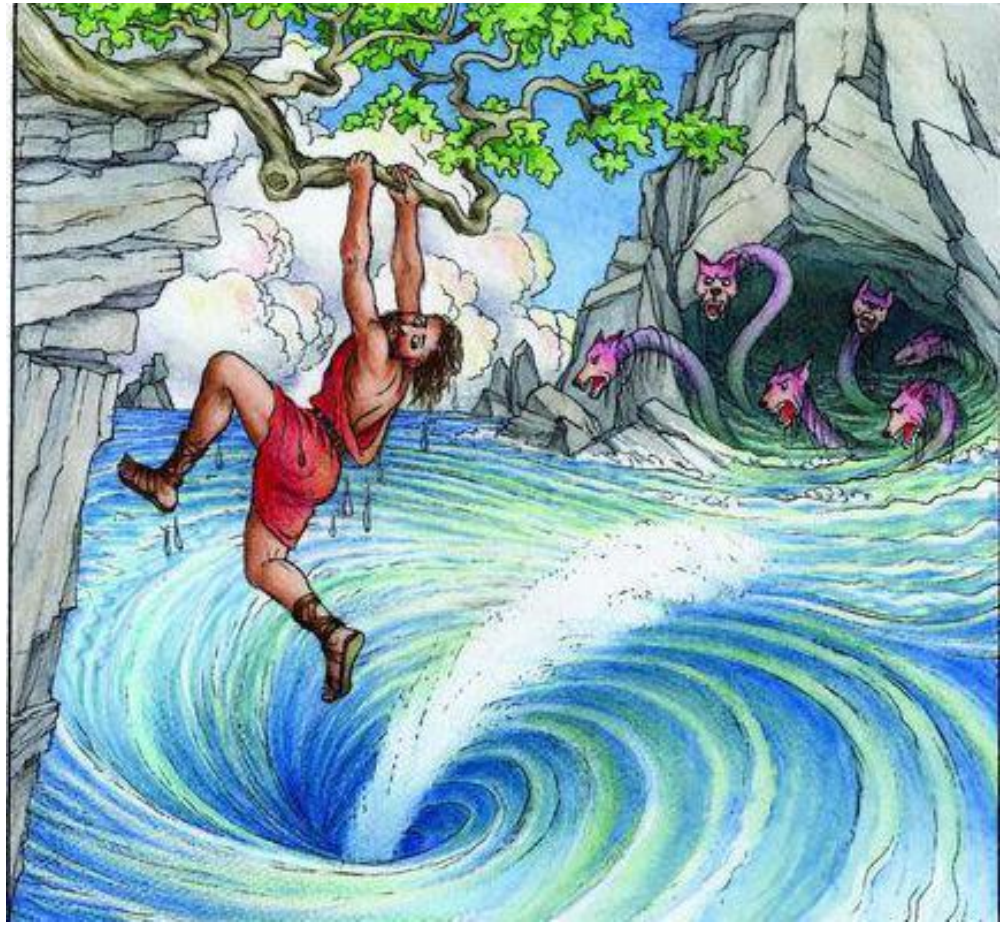
- Learn a *feature hierarchy*
- Each layer extracts features from the output of previous layer
- All layers are trained jointly

# Multi-Layer network demo




http://playground.tensorflow.org/

# Hyperparameters, bias-variance tradeoff, validation



# Supervised learning outline revisited

1. **Collect data and labels**
2. **Specify model:** select model class and loss function
3. **Train model:** find the parameters of the model that minimize the empirical loss on the training data

This involves *hyperparameters* that affect the generalization ability of the trained model

# Hyperparameters

- $K$ in $K$-nearest-neighbor
  - What if $K$ is too large?
  - What if $K$ is too small?

# Hyperparameters

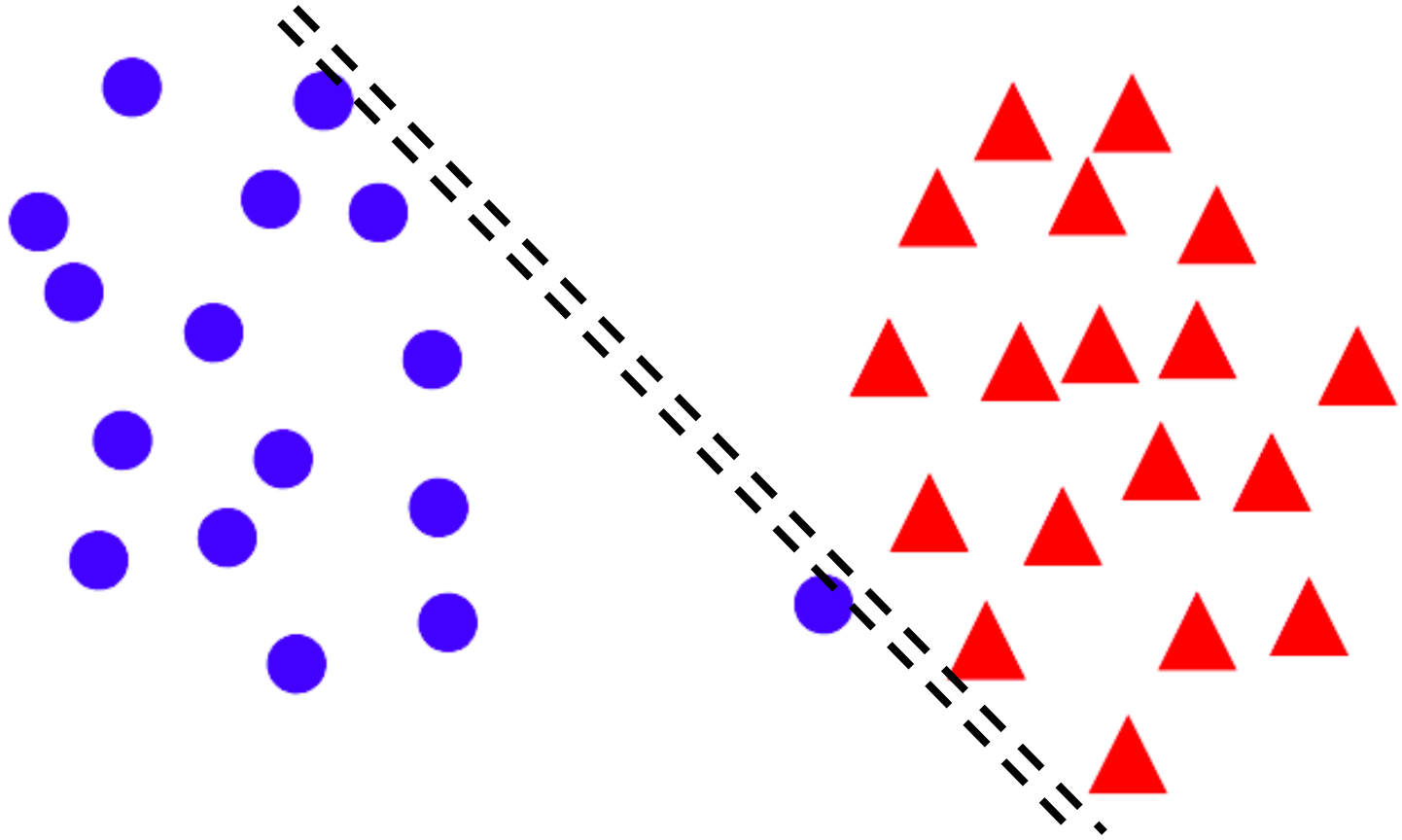
- Regularization constant $\lambda$ in SVM optimization

$$\min_w \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{n} \max[0, 1 - y_i w^T x_i]$$

  - What if $\lambda$ is too large?
  - What if $\lambda$ is too small?

# Hyperparameters

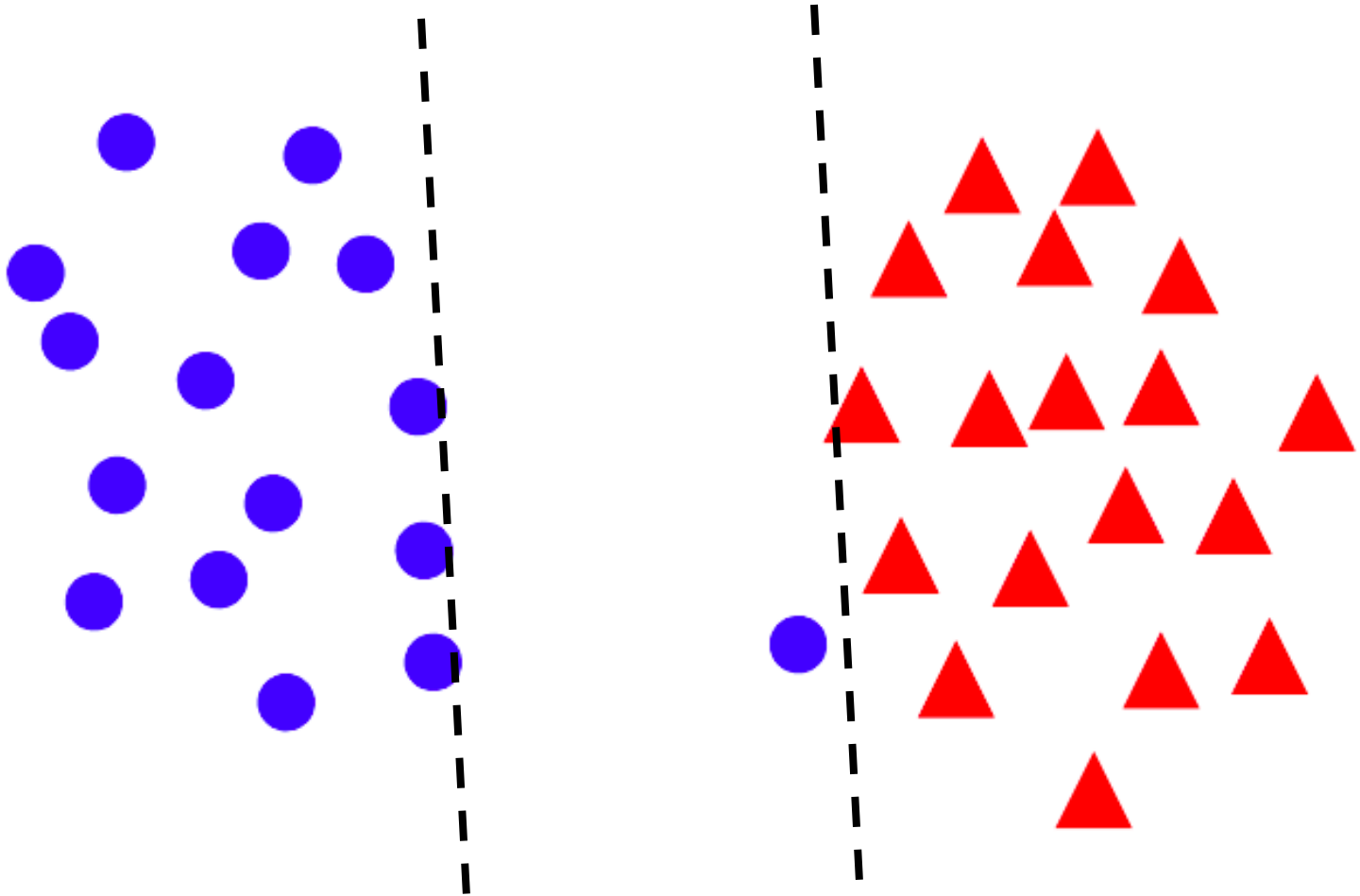
- Regularization constant $\lambda$ in SVM optimization
- Recall: tradeoff between margin and constraint violations



Source

# Hyperparameters

- Regularization constant $\lambda$ in SVM optimization
- Recall: tradeoff between margin and constraint violations

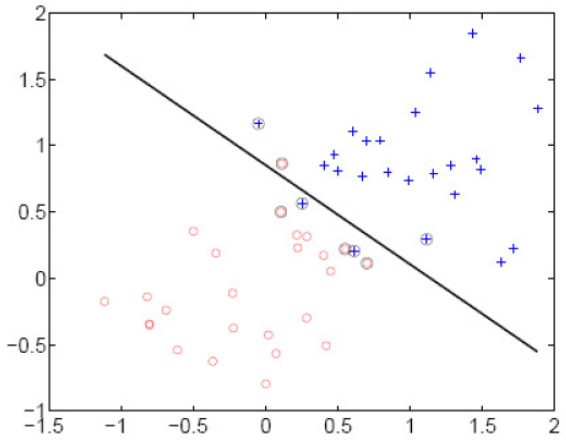


Source

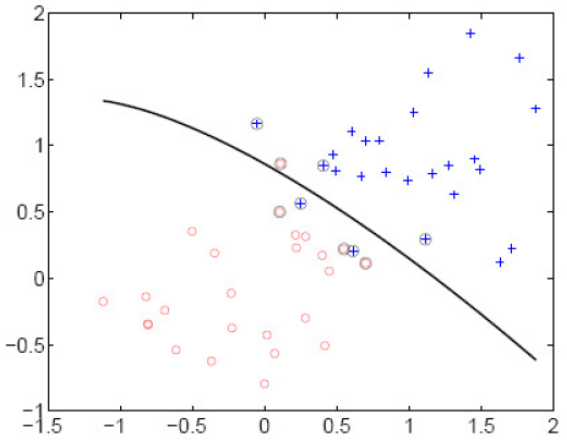
# Hyperparameters

- What about nonlinear SVMs?
  - Choice of kernel (and any associated constants)

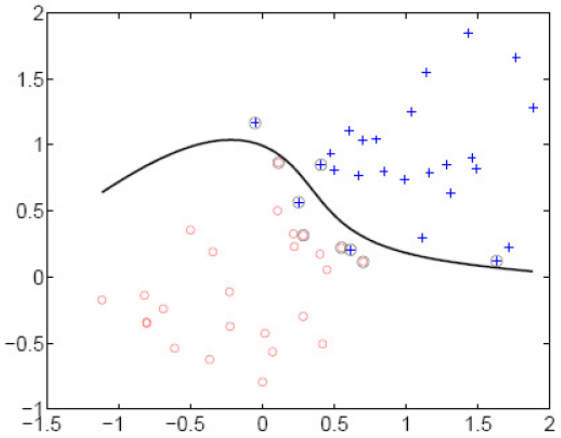
# Polynomial kernel: $K(x, x') = (x^T x' + c)^d$



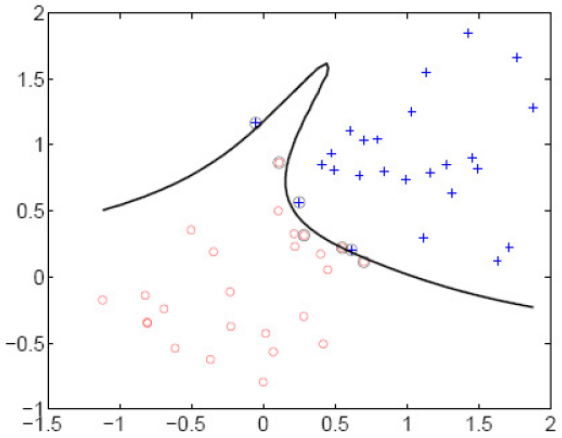

linear




$2^{nd}$ order polynomial




$4^{th}$ order polynomial



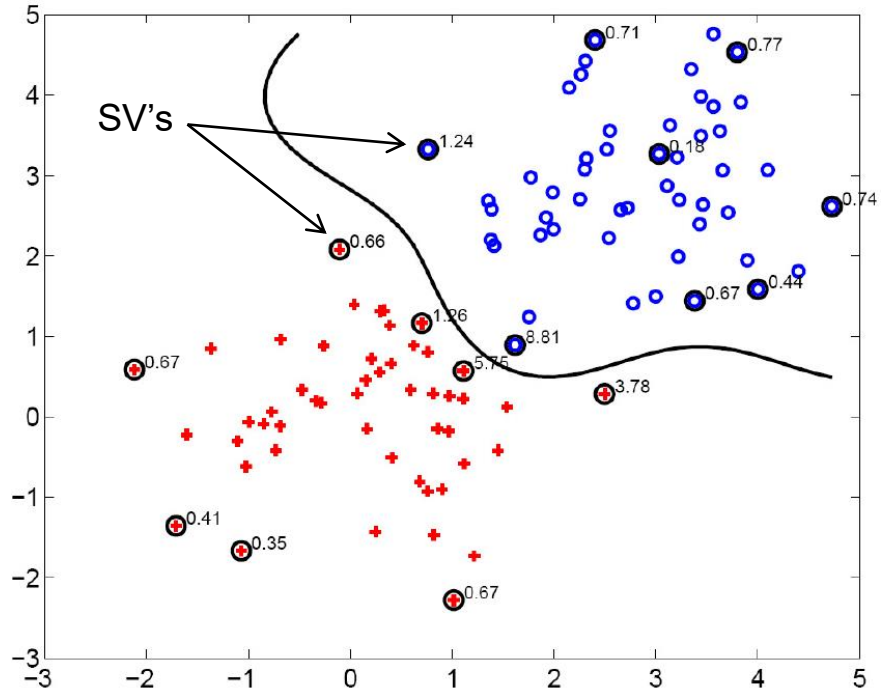

$8^{th}$ order polynomial

# Gaussian kernel

- Gaussian kernel with bandwidth $\sigma$:

$$K(x, x') = \exp\left(-\frac{1}{\sigma^2}\|x - x'\|^2\right)$$

- Recall: the predictor $f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x)$ is a sum of "bumps" centered on support vectors



# Gaussian kernel

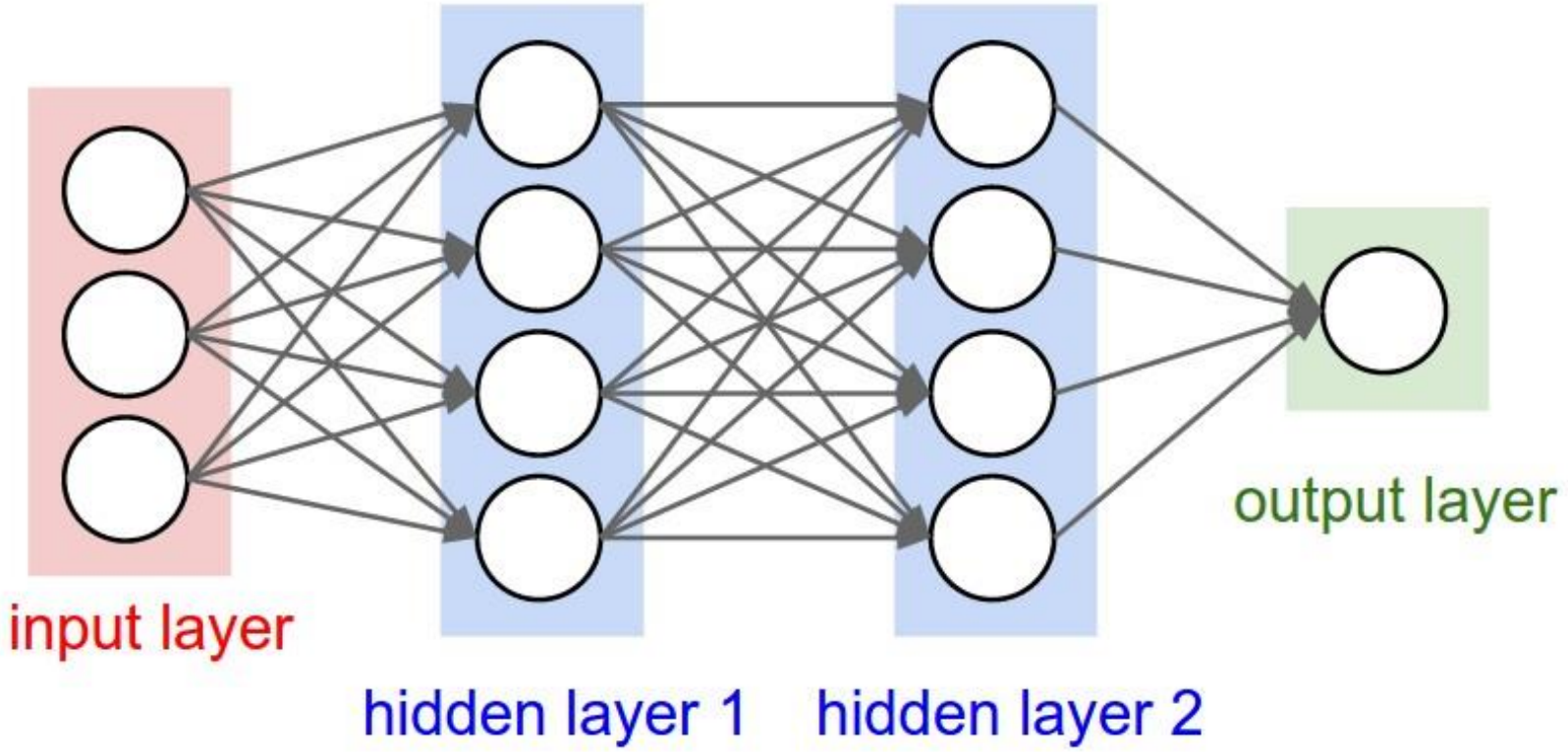
- Gaussian kernel with bandwidth $\sigma$:

$$K(x, x') = \exp\left(-\frac{1}{\sigma^2}\|x - x'\|^2\right)$$

- Recall: the predictor $f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x)$ is a sum of "bumps" centered on support vectors
- How does the value of $\sigma$ affect the behavior of the predictor?
  - What if $\sigma$ is close to zero?
  - What if $\sigma$ is very large?

# Hyperparameters in multi-layer networks

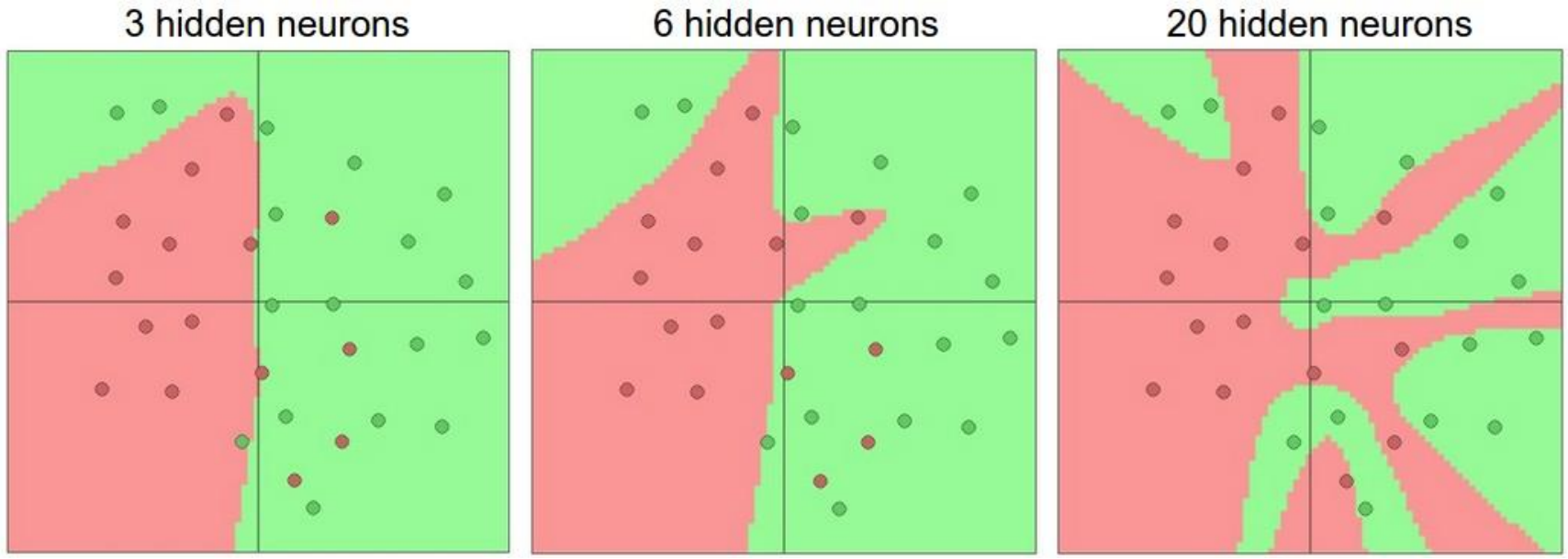
- Number of layers, number of units per layer




Source: Stanford 231n

# Hyperparameters in multi-layer networks

- Number of layers, number of units per layer

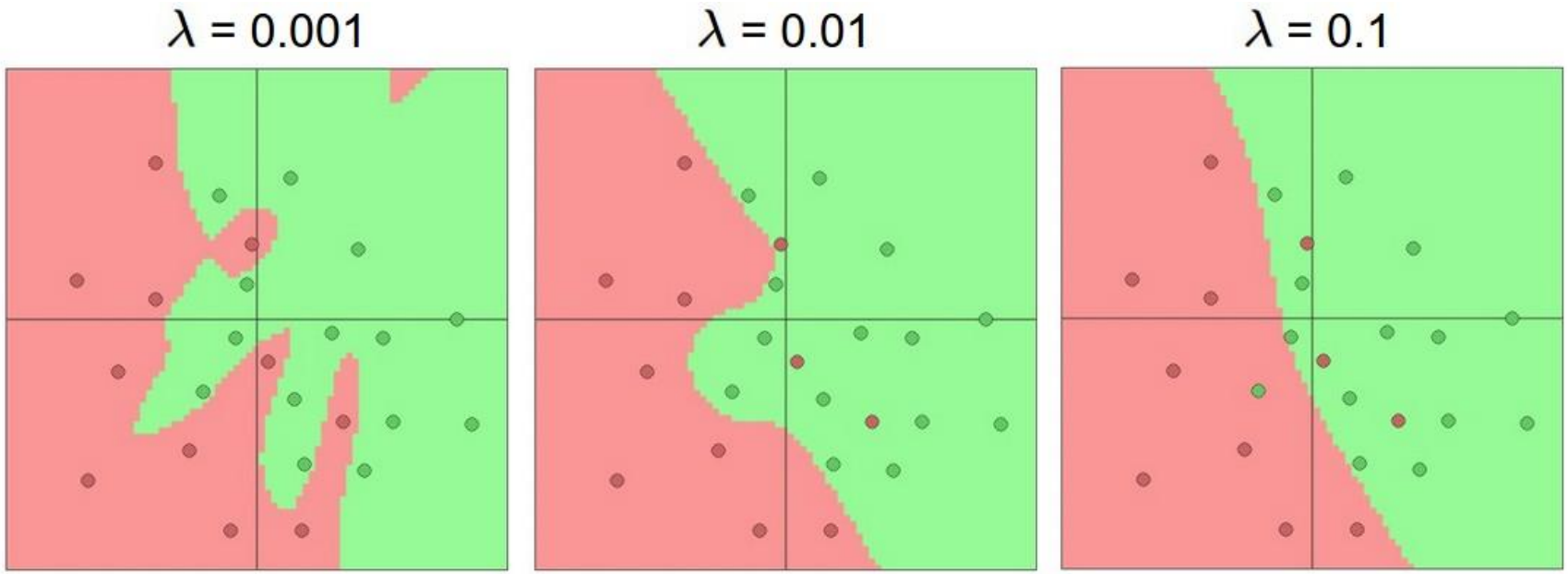



Number of hidden units in a two-layer network

Source: Stanford 231n

# Hyperparameters in multi-layer networks

- Number of layers, number of units per layer
- Regularization constant




Source: Stanford 231n

# Hyperparameters in multi-layer networks

- Number of layers, number of units per layer
- Regularization constant
- SGD settings: learning rate schedule, number of epochs, minibatch size, etc.

# Review: Hyperparameters

- What are some examples of hyperparameters?
  - K in K-NN
  - In SVMs: regularization constant, kernel type and constants
  - In neural networks: number of layers, number of units per layer, regularization
  - SGD settings: learning rate schedule, number of epochs, minibatch size, etc.

- We can think of our hyperparameter choices as defining the “complexity” of the model and controlling its generalization ability

# Model complexity and generalization

- Generalization (test) error of learning algorithms has two main components:
  - **Bias:** error due to simplifying model assumptions
  - **Variance:** error due to randomness of training set

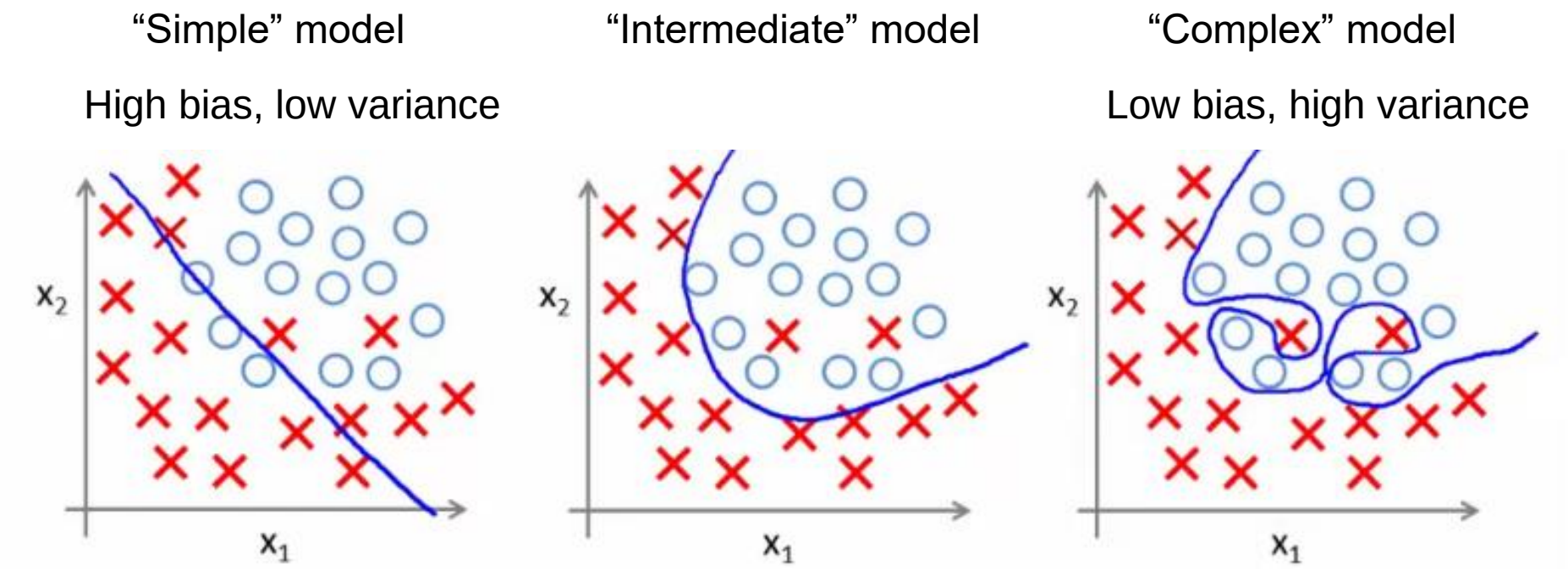



Figure source

# Bias-variance tradeoff

- What if your model **bias** is too high?
  - Your model is **underfitting** – it is incapable of capturing the important characteristics of the training data
- What if your model **variance** is too high?
  - Your model is **overfitting** – it is fitting noise and unimportant characteristics of the data
- How to recognize underfitting or overfitting?

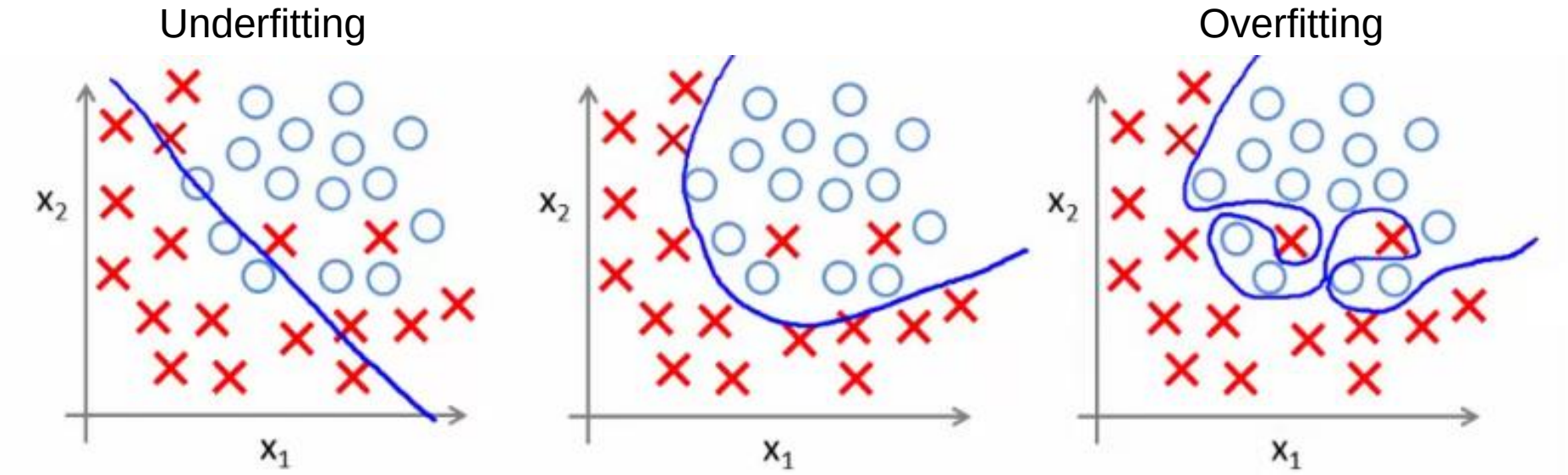
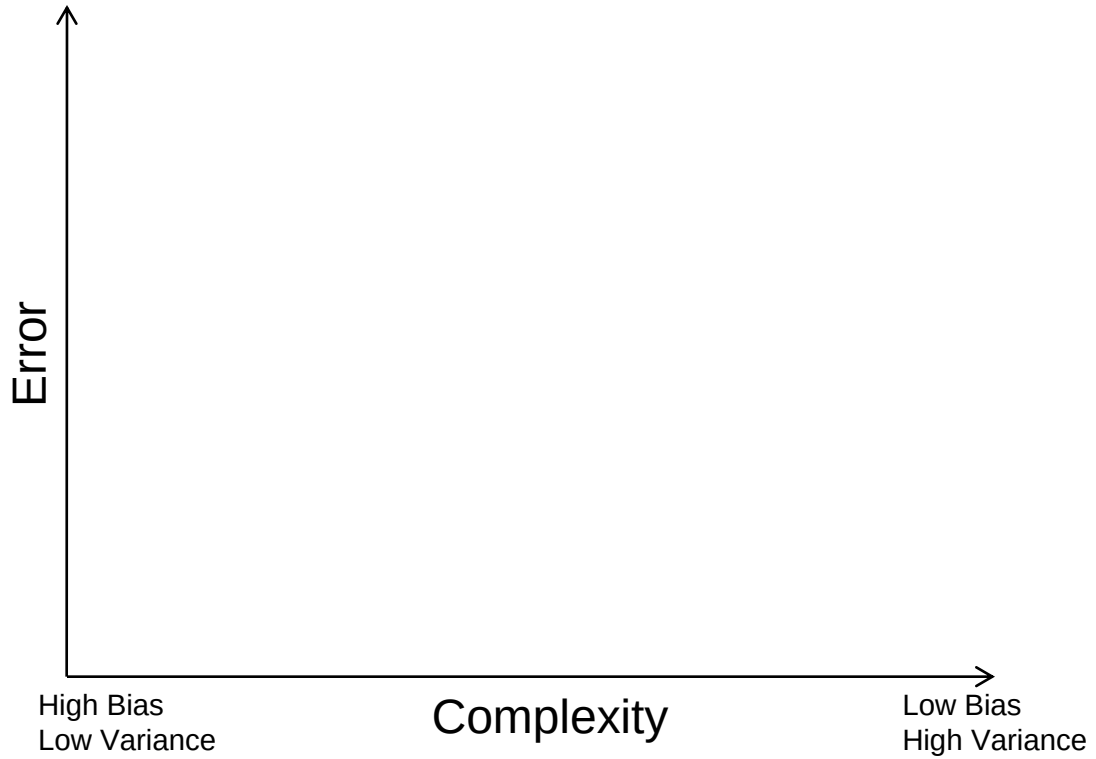



Figure source

# Bias-variance tradeoff

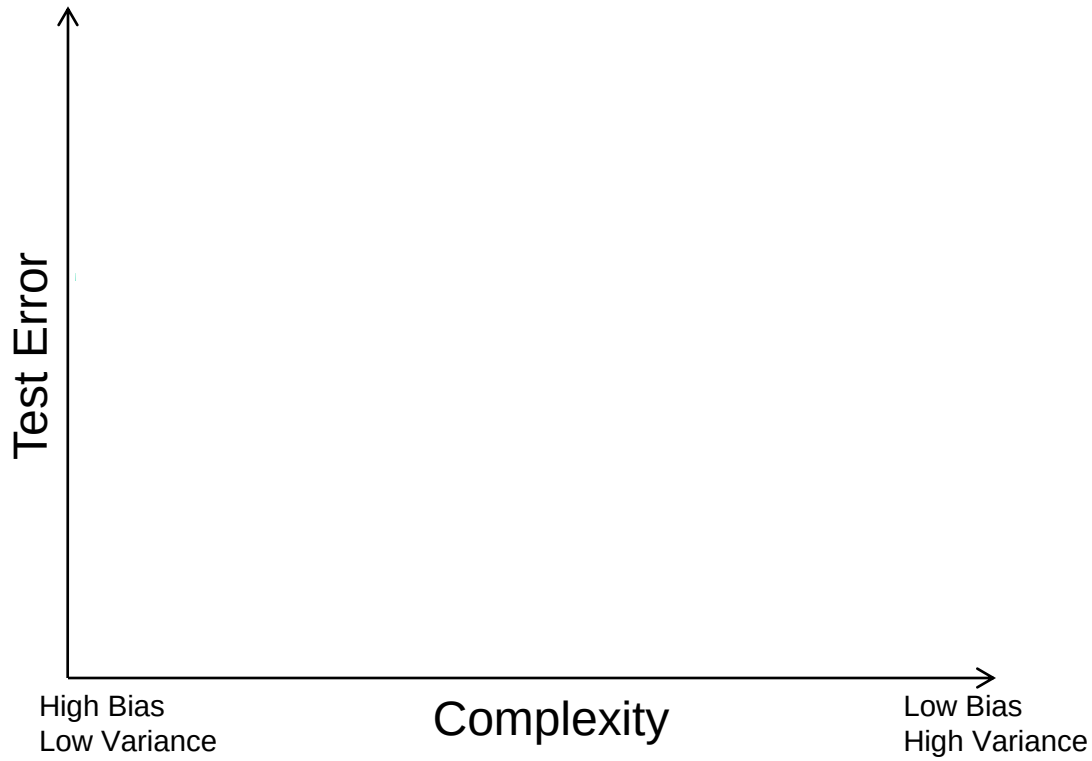
- What if your model **bias** is too high?
  - Your model is **underfitting** – it is incapable of capturing the important characteristics of the training data
- What if your model **variance** is too high?
  - Your model is **overfitting** – it is fitting noise and unimportant characteristics of the data
- How to recognize underfitting or overfitting?
  - Need to look at both training and test error
  - **Underfitting:** training and test error are both *high*
  - **Overfitting:** training error is *low*, test error is *high*

# Looking at training and test error



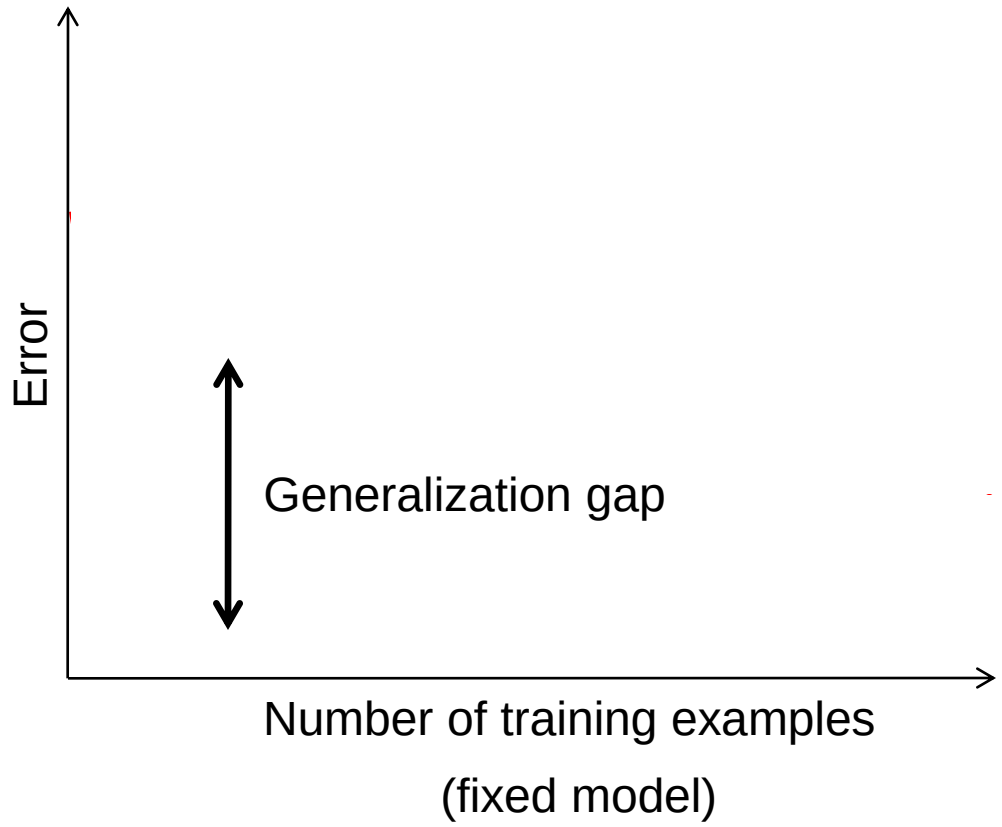

Source: D. Hoiem

# Dependence on training set size




Source: D. Hoiem

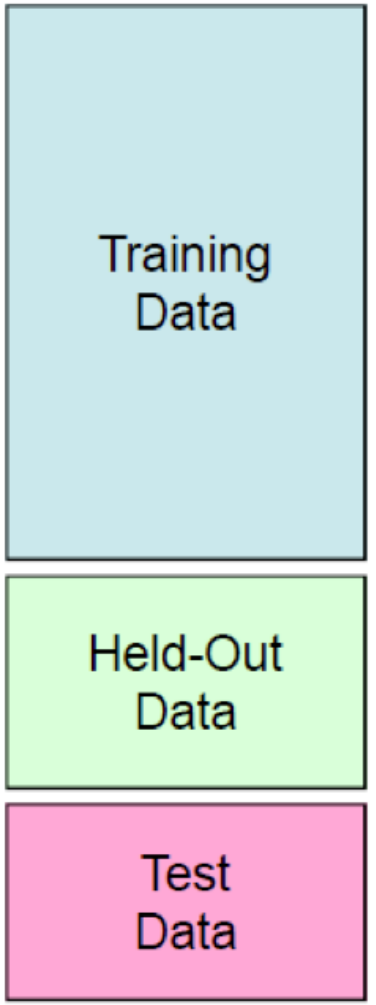
# Dependence on training set size




Source: D. Hoiem

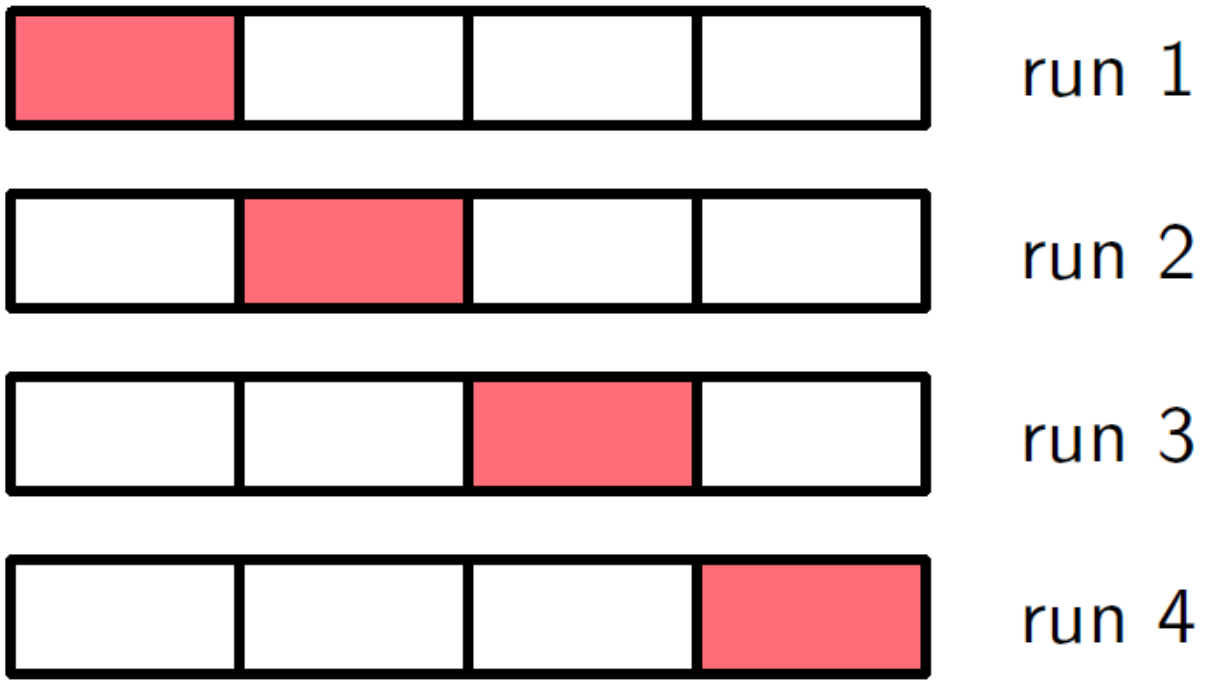
# Hyperparameter search in practice

- For a range of hyperparameter choices, iterate:
  - Learn parameters on the ***training data***
  - Measure accuracy on the ***held-out*** or ***validation data***
- Finally, measure accuracy on the ***test data***
- **Crucial:** do not peek at test set during hyperparameter search!
  - The test set needs to be used sparingly since it is supposed to represent *never before seen data*



# Hyperparameter search in practice

- Variant: *K-fold cross-validation*
  - Partition the data into K groups
  - In each run, select one of the groups as the validation set



# What’s the big deal?

- If you don’t maintain proper training-validation-test hygiene, you will be fooling yourself or others (professors, reviewers, customers)
- It may even cause a public scandal!

# What's the big deal?

## Baidu admits cheating in international supercomputer competition

ZDNet

Baidu recently apologised for violating the rules of an international supercomputer test in May, when the Chinese search engine giant claimed to beat both Google and Microsoft on the ImageNet image-recognition test.

By Cyrus Lee | June 10, 2015 -- 00:15 GMT (17:15 PDT) | Topic: China

TECHNOLOGY

The New York Times

## *Computer Scientists Are Astir After Baidu Team Is Barred From A.I. Competition*

By JOHN MARKOFF JUNE 3, 2015

engadget

## Baidu caught gaming recent supercomputer performance test

by Andrew Tarantola | @terrortola | June 3rd 2015 At 11:09pm

# IMAGENET Large Scale Visual Recognition Challenge (ILSVRC)

Date: June 2, 2015

Dear ILSVRC community,

This is a follow up to the announcement on May 19, 2015 with some more details and the status of the test server.

During the period of November 28th, 2014 to May 13th, 2015, there were at least 30 accounts used by a team from Baidu to submit to the test server at least 200 times, far exceeding the specified limit of two submissions per week. This includes short periods of very high usage, for example with more than 40 submissions over 5 days from March 15th, 2015 to March 19th, 2015. Figure A below shows submissions from ImageNet accounts known to be associated with the team in question. Figure B shows a comparison to the activity from all other accounts.

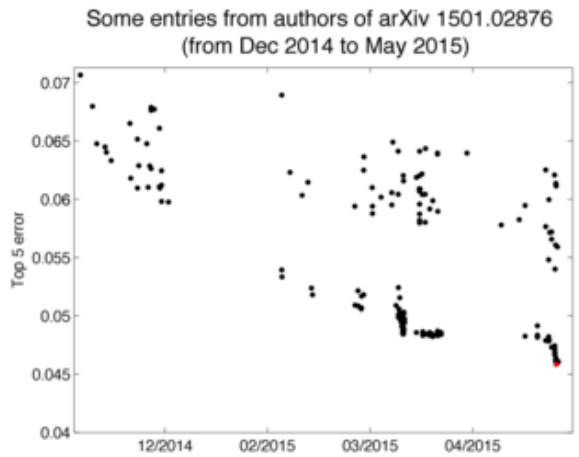



Figure A

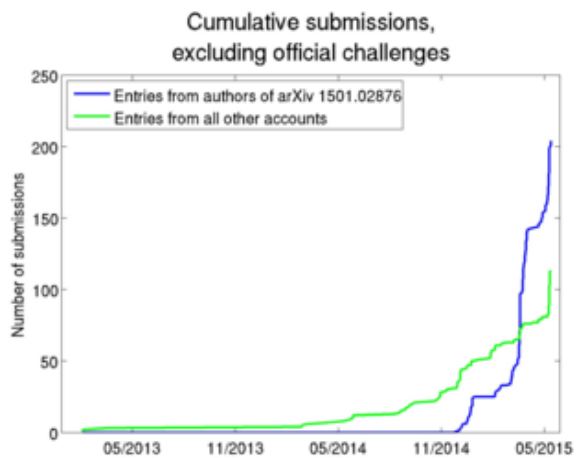



Figure B

The results obtained during this period are reported in a recent arXiv paper. Because of the violation of the regulations of the test server, these results may not be directly comparable to results obtained and reported by other teams. To make this clear, by exploiting the ability to test many slightly different solutions on the test server it is possible to 1) select the best out of a set of very similar solutions based on test performance and achieve a small but potentially significant advantage and 2) choose methods for further research and development based directly on the test data instead of using only the training and validation data for such choices.

http://www.image-net.org/challenges/LSVRC/announcement-June-2-2015

# Acknowledgement


- Deep Learning, Stanford University
- Introduction to Deep Learning, University of Illinois at Urbana-Champaign
- Introduction to Deep Learning, Carnegie Mellon University
- Convolutional Neural Networks for Visual Recognition, Stanford University
- Natural Language Processing with Deep Learning, Stanford University