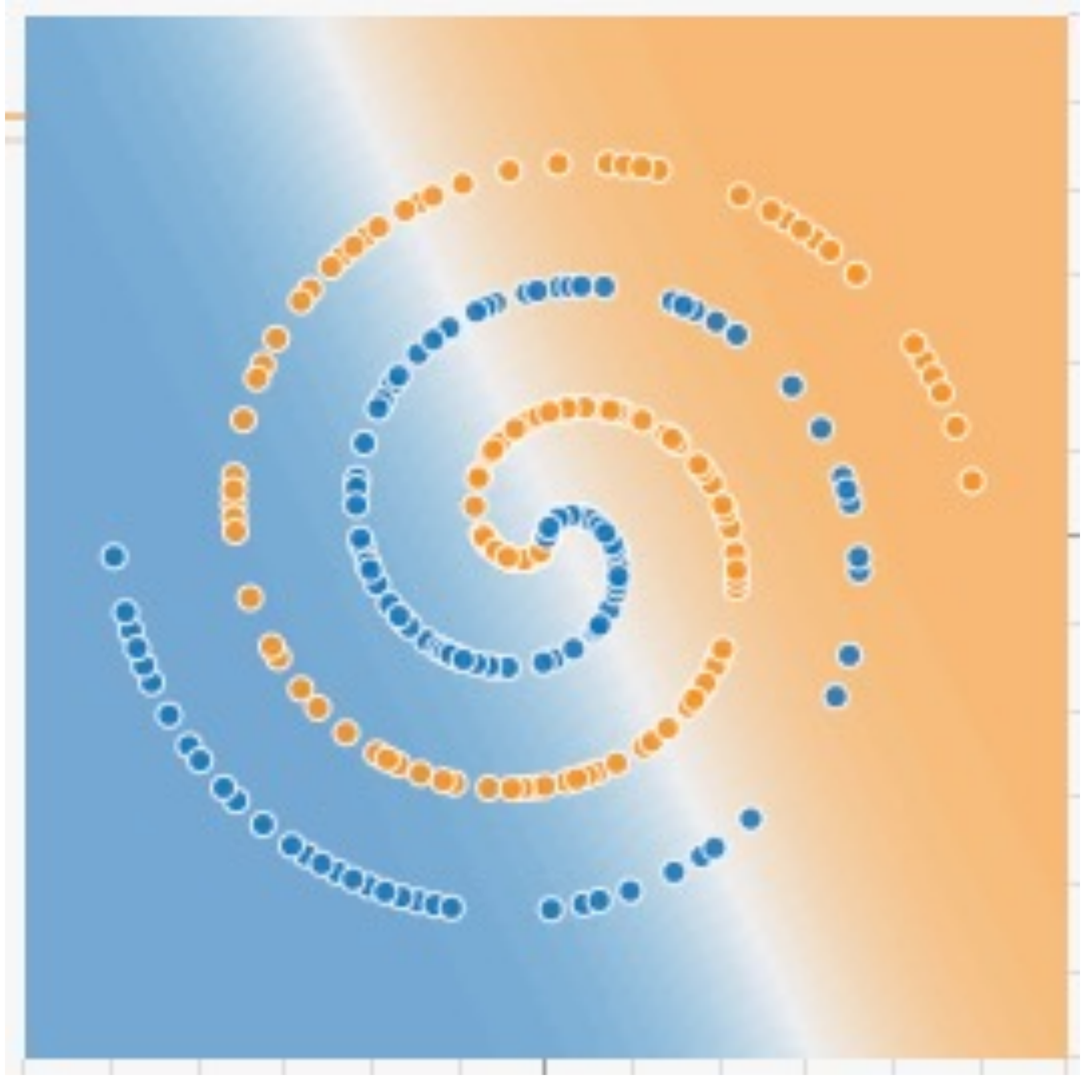
Feature engineering, Nonlinear classifiers, bias-variance tradeoff

Saurabh Gupta



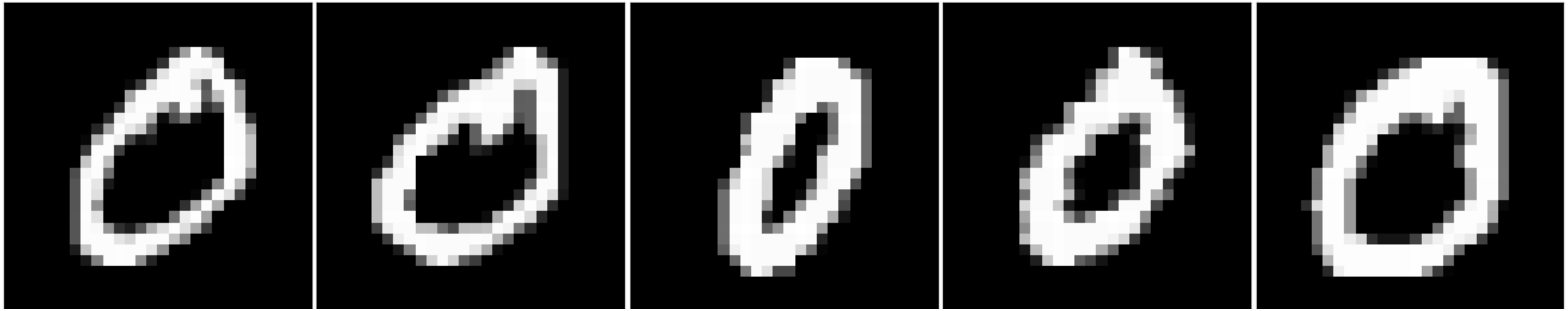
Many slides from Lana Lazebnik

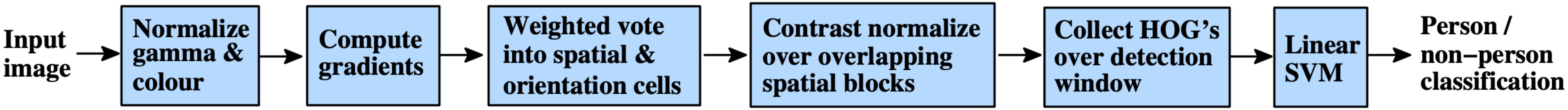
Overview

- Feature Design
- Nonlinear classifiers
  - “Shallow” approach: Kernel support vector machines (SVMs)
  - “Deep” approach: Multi-layer neural networks
- Controlling classifier complexity
  - Hyperparameters
  - Bias-variance tradeoff
  - Overfitting and underfitting
  - Hyperparameter search in practice

Hand-designing Feature Spaces using Domain Knowledge

- Edges / gradients more useful than raw pixel values
- Invariance to local deformations
  - Spatial pooling
  - Quantization into coarse bins

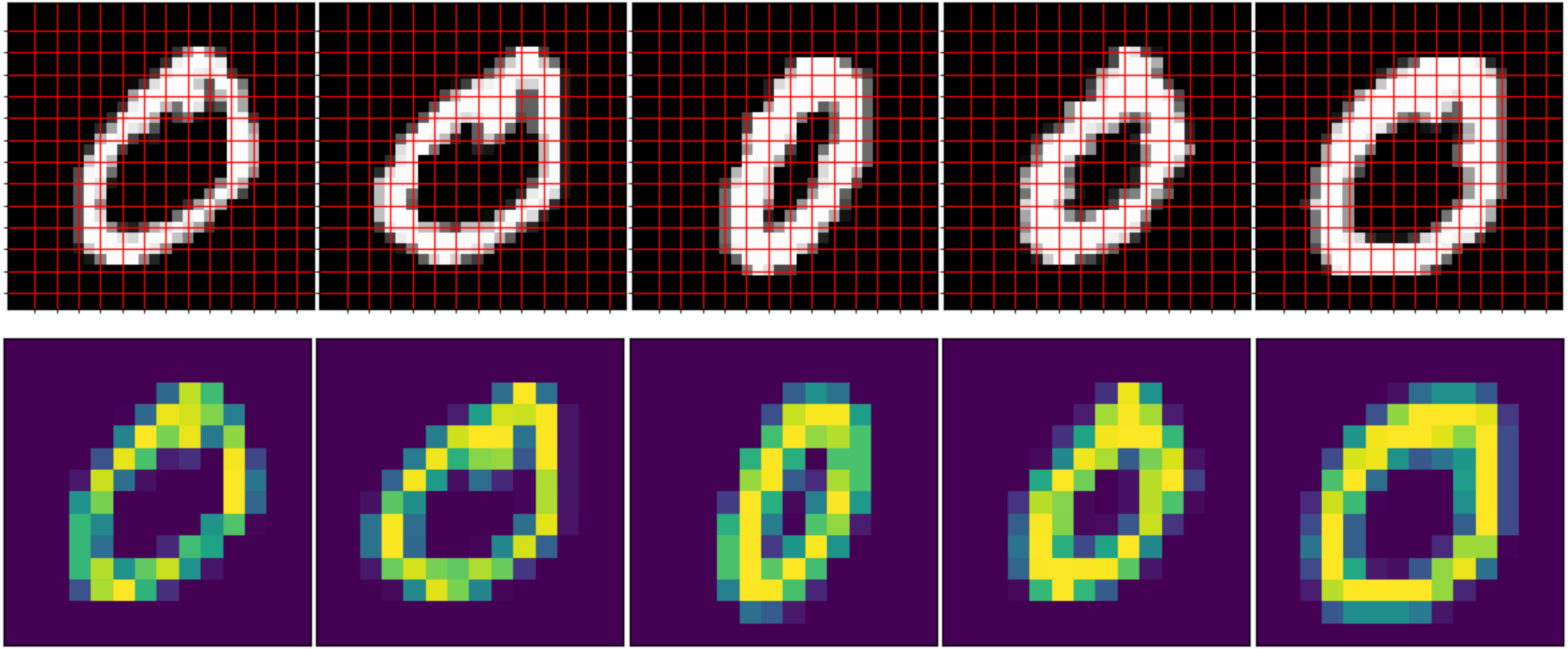




Input image → Normalize gamma & colour → Compute gradients → Weighted vote into spatial & orientation cells → Contrast normalize over overlapping spatial blocks → Collect HOG's over detection window → Linear SVM → Person / non−person classification

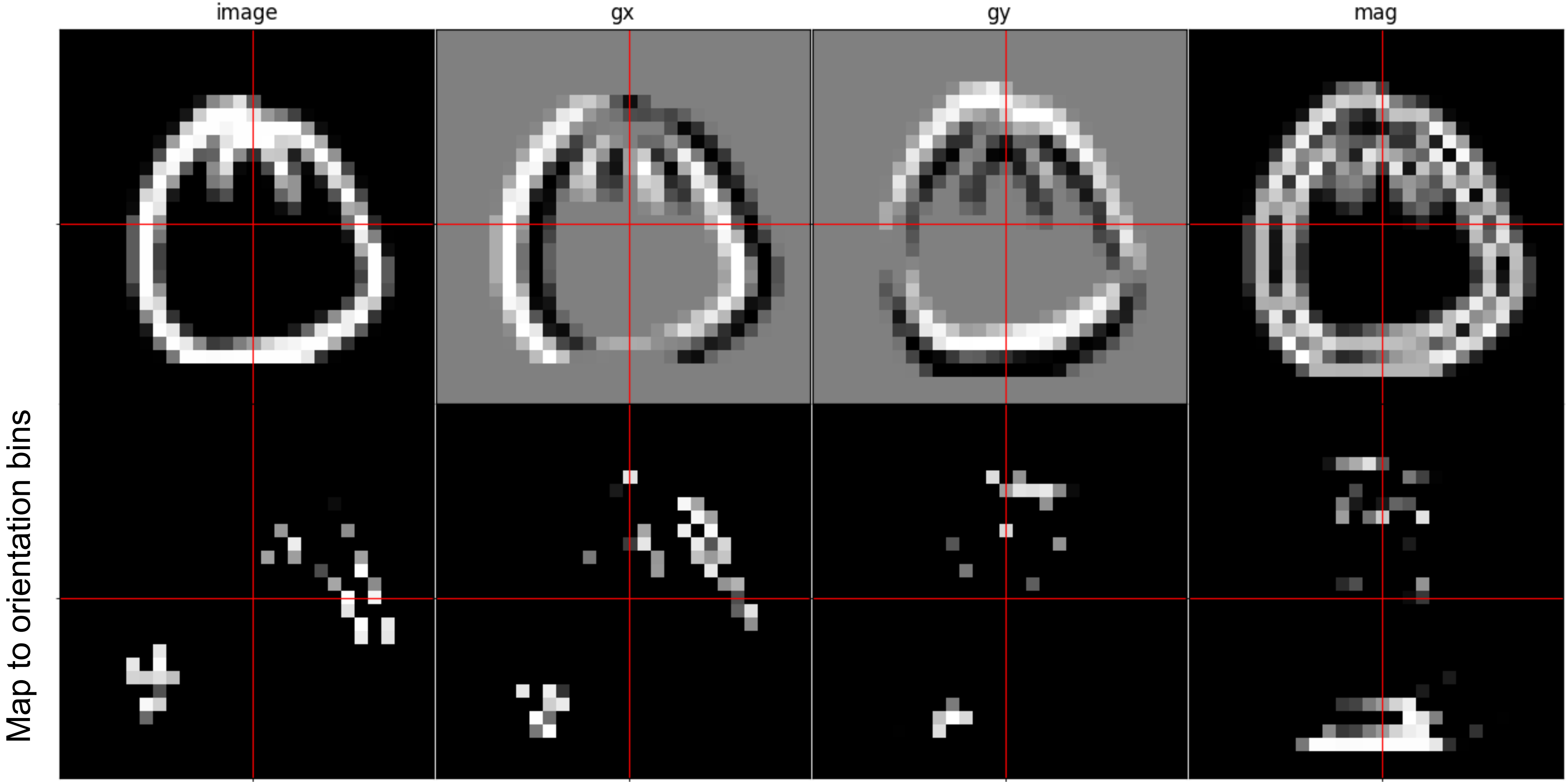
Hand-designing Feature Spaces using Domain Knowledge

- E.g. Spatial pooling of raw pixels



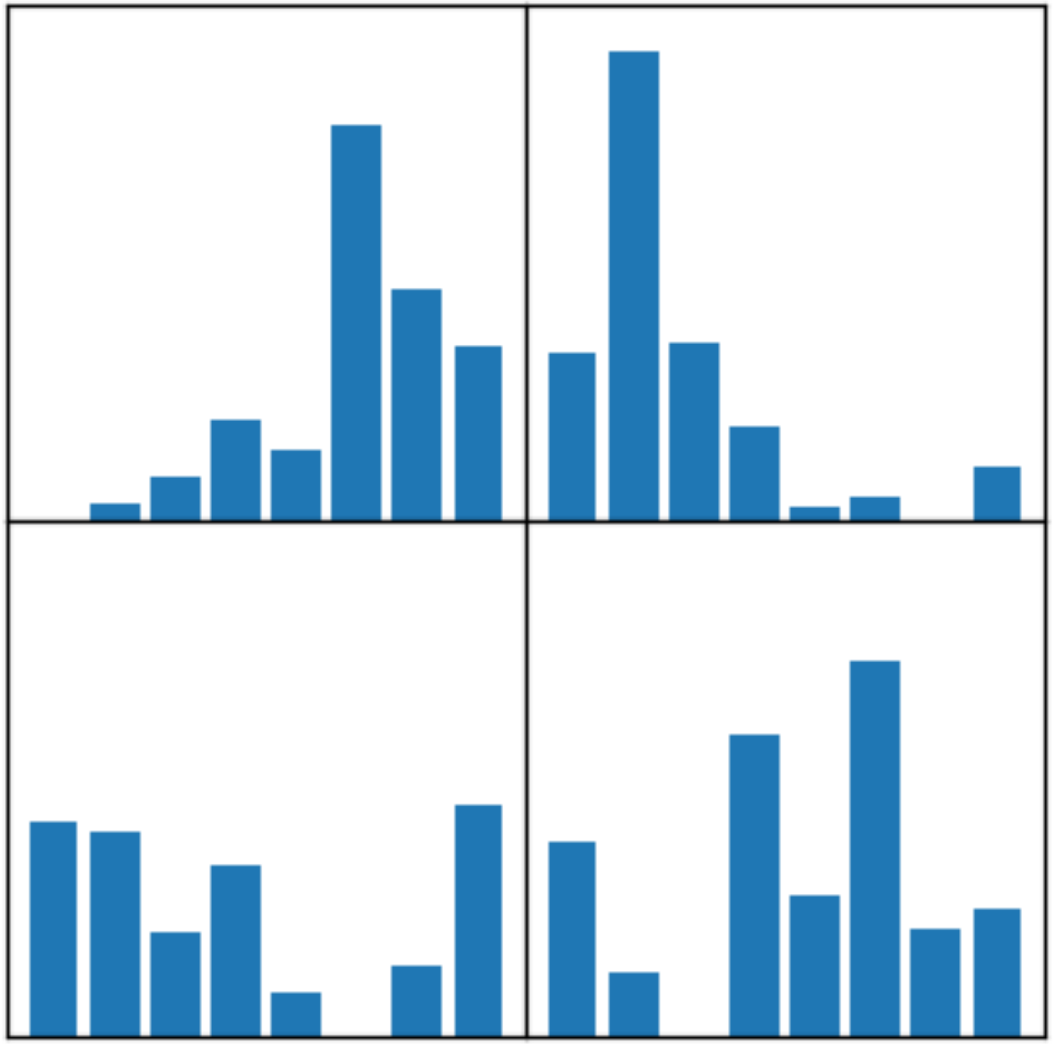
Hand-designing Feature Spaces using Domain Knowledge

- E.g. Histogram of Oriented Gradients

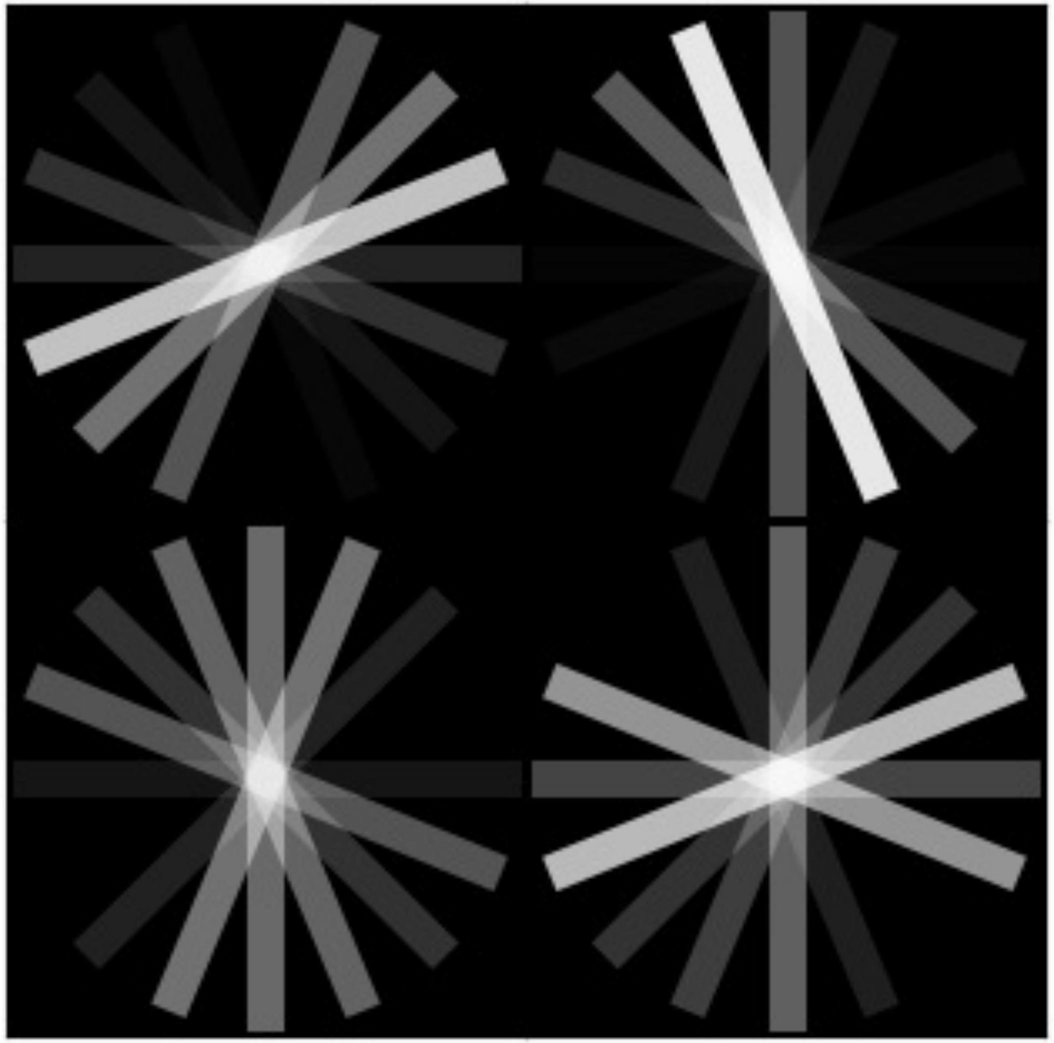


Hand-designing Feature Spaces using Domain Knowledge

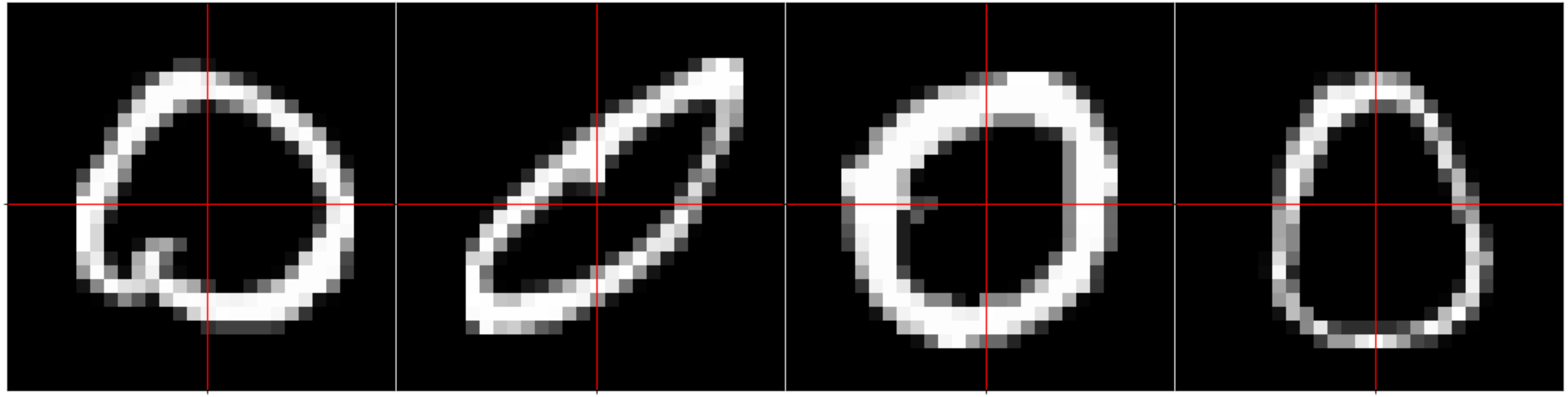
Histogram of Oriented Gradients

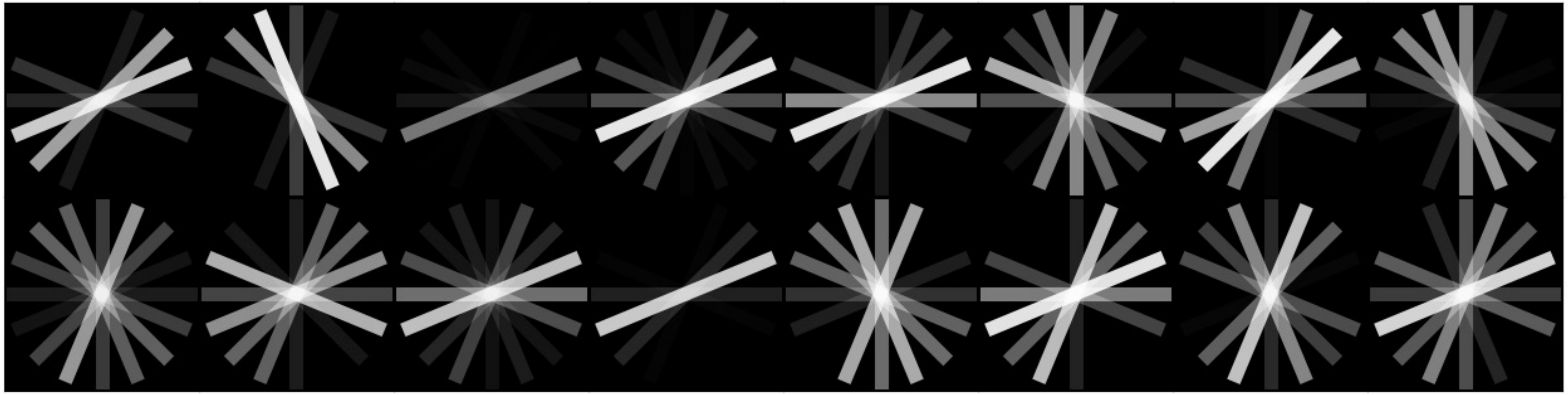


Histogram of Oriented Gradients

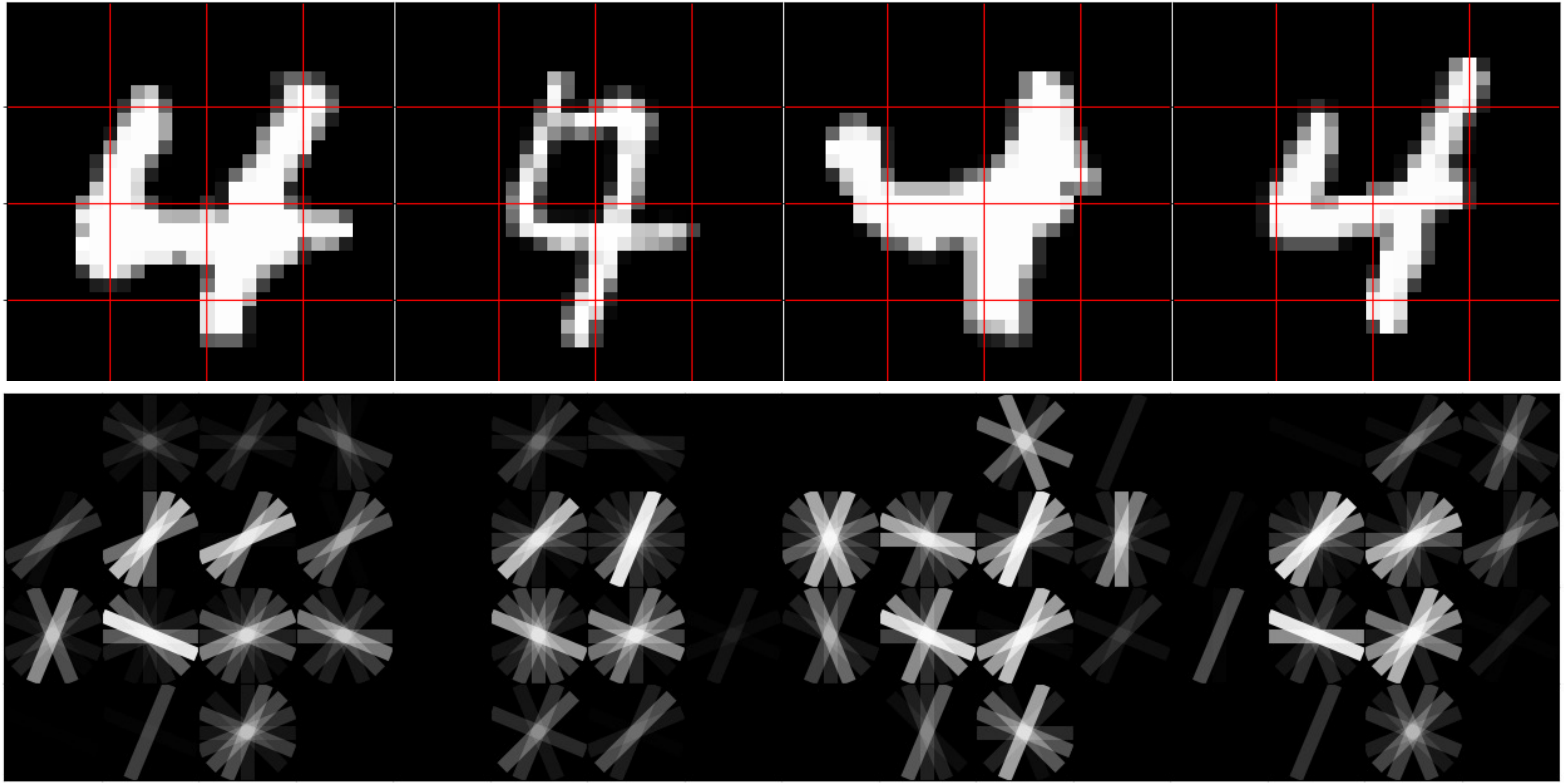


Hand-designing Feature Spaces using Domain Knowledge

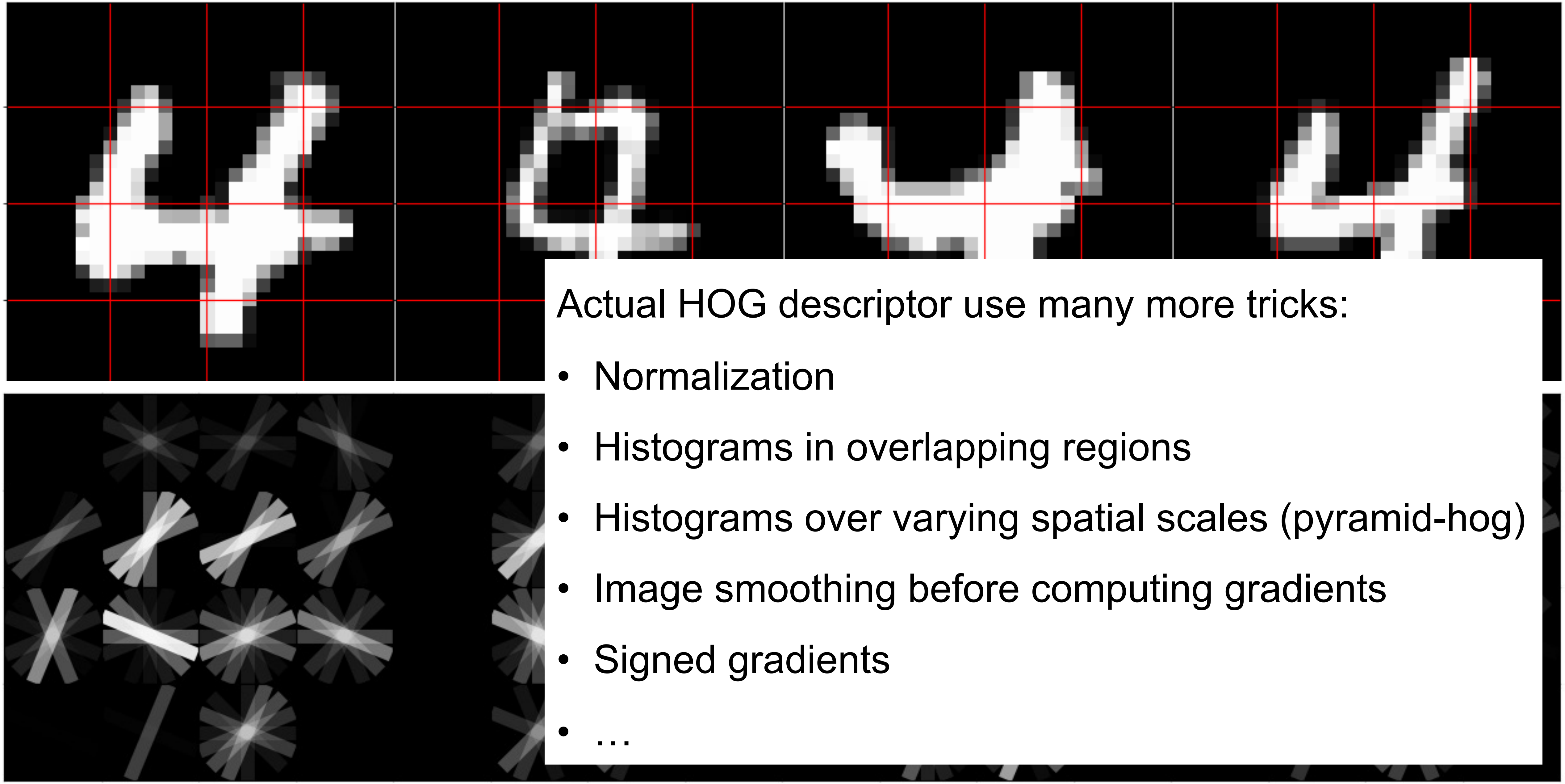




Hand-designing Feature Spaces using Domain Knowledge



Hand-designing Feature Spaces using Domain Knowledge



Actual HOG descriptor use many more tricks:

- Normalization
- Histograms in overlapping regions
- Histograms over varying spatial scales (pyramid-hog)
- Image smoothing before computing gradients
- Signed gradients
- …

Hand-designing Feature Spaces using Domain Knowledge

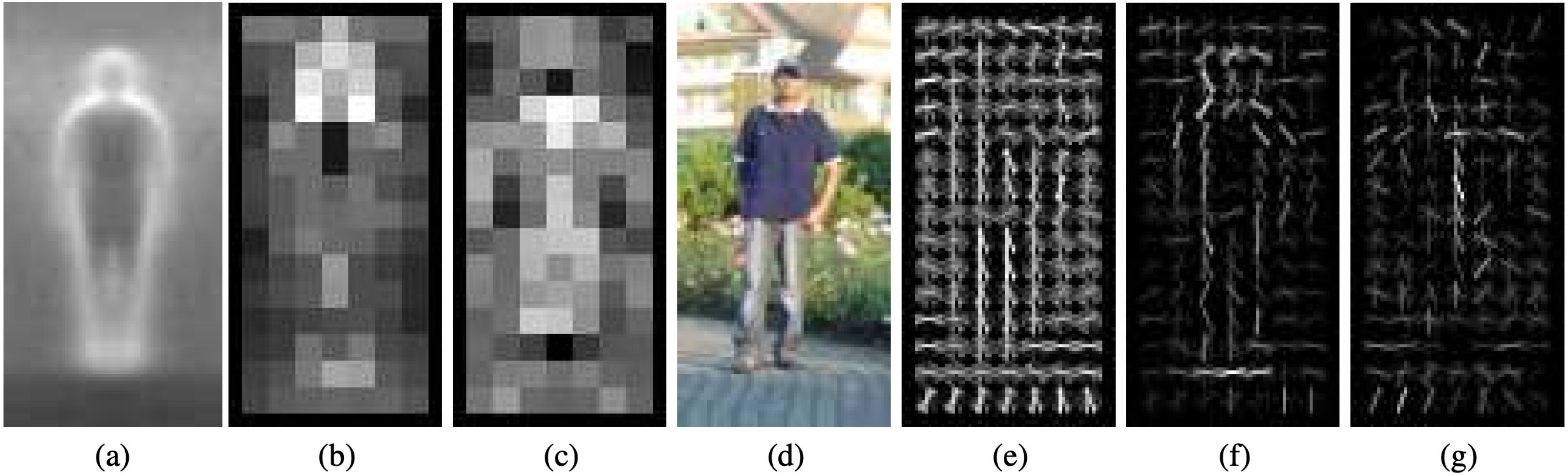
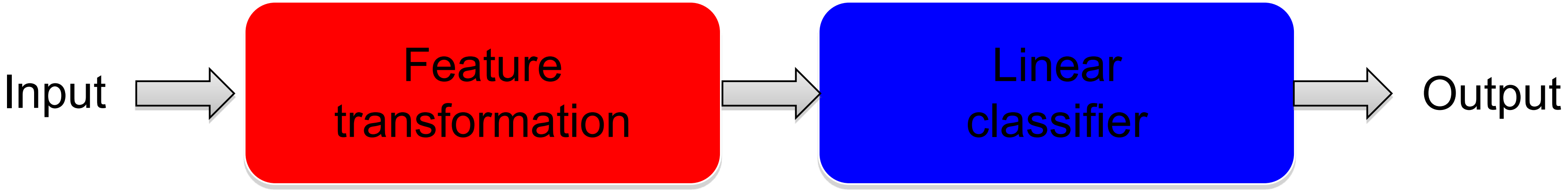


Figure 6. Our HOG detectors cue mainly on silhouette contours (especially the head, shoulders and feet). The most active blocks are centred on the image background just *outside* the contour. (a) The average gradient image over the training examples. (b) Each "pixel" shows the maximum positive SVM weight in the block centred on the pixel. (c) Likewise for the negative SVM weights. (d) A test image. (e) It's computed R-HOG descriptor. (f,g) The R-HOG descriptor weighted by respectively the positive and the negative SVM weights.

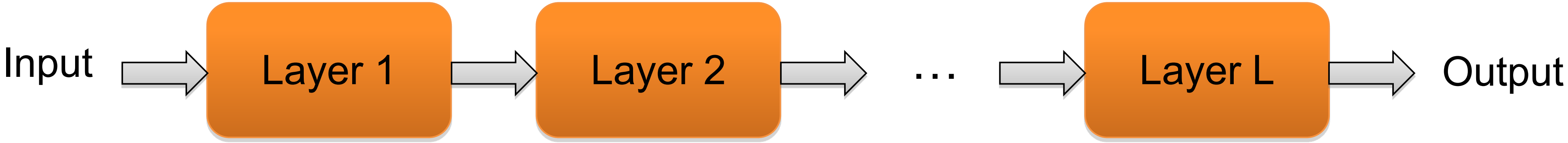
Dalal and Triggs. **Histograms of Oriented Gradients for Human Detection**. CVPR 2005.

Beyond Linear Decision Boundaries

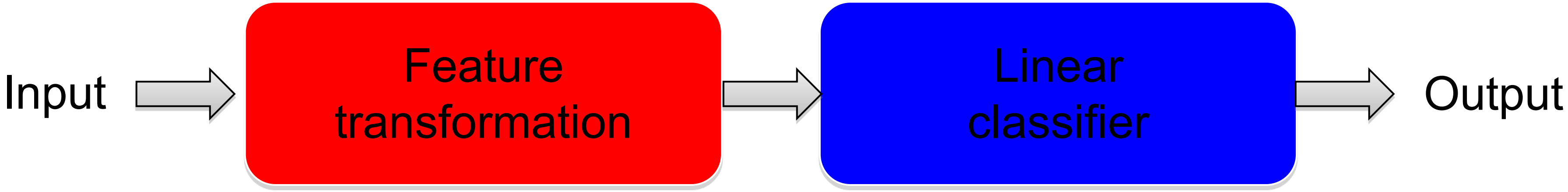
- Feature design approach: design features that work well with linear classifiers
- Non-linear classifier approach:
    - **“Shallow” approach:** nonlinear feature transformation followed by linear classifier



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



Shallow approach



Input

Feature transformation

Linear classifier

Output

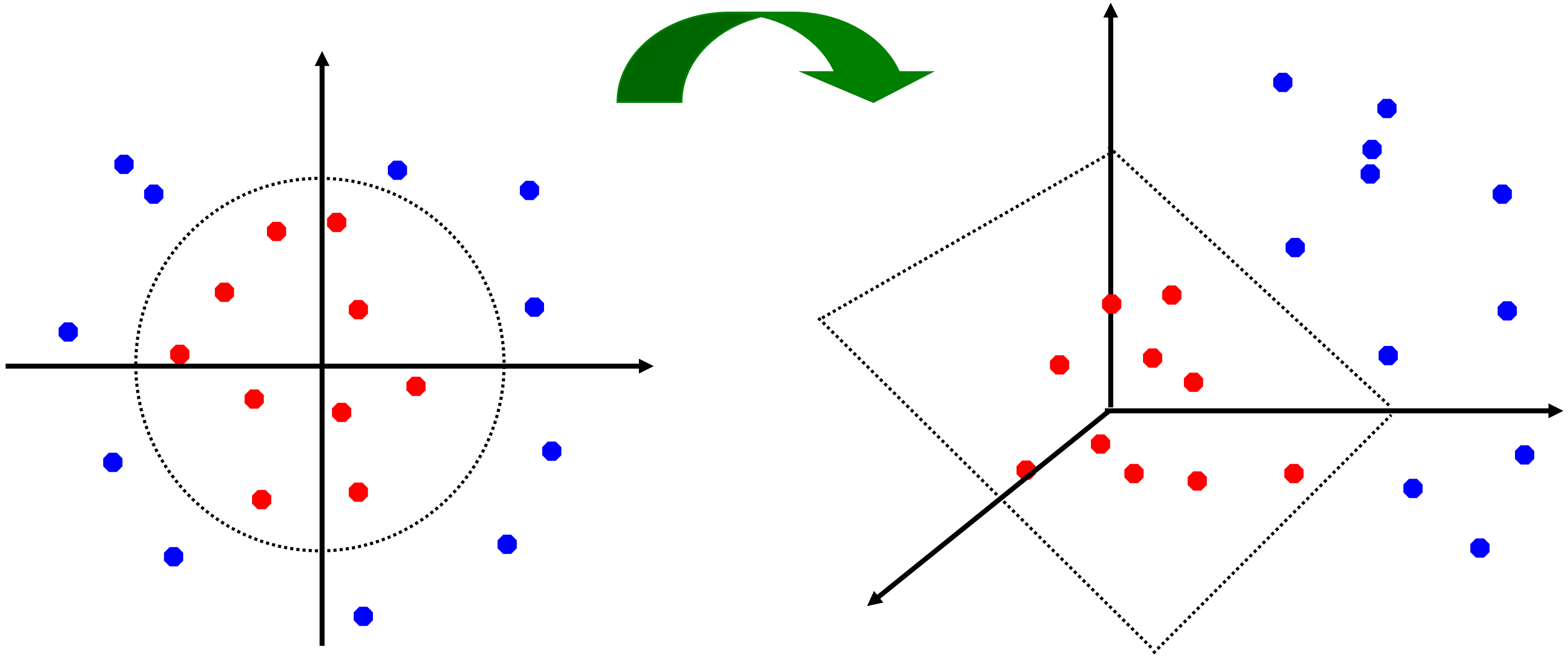


Image credit: Andrew Moore

Nonlinear SVMs

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

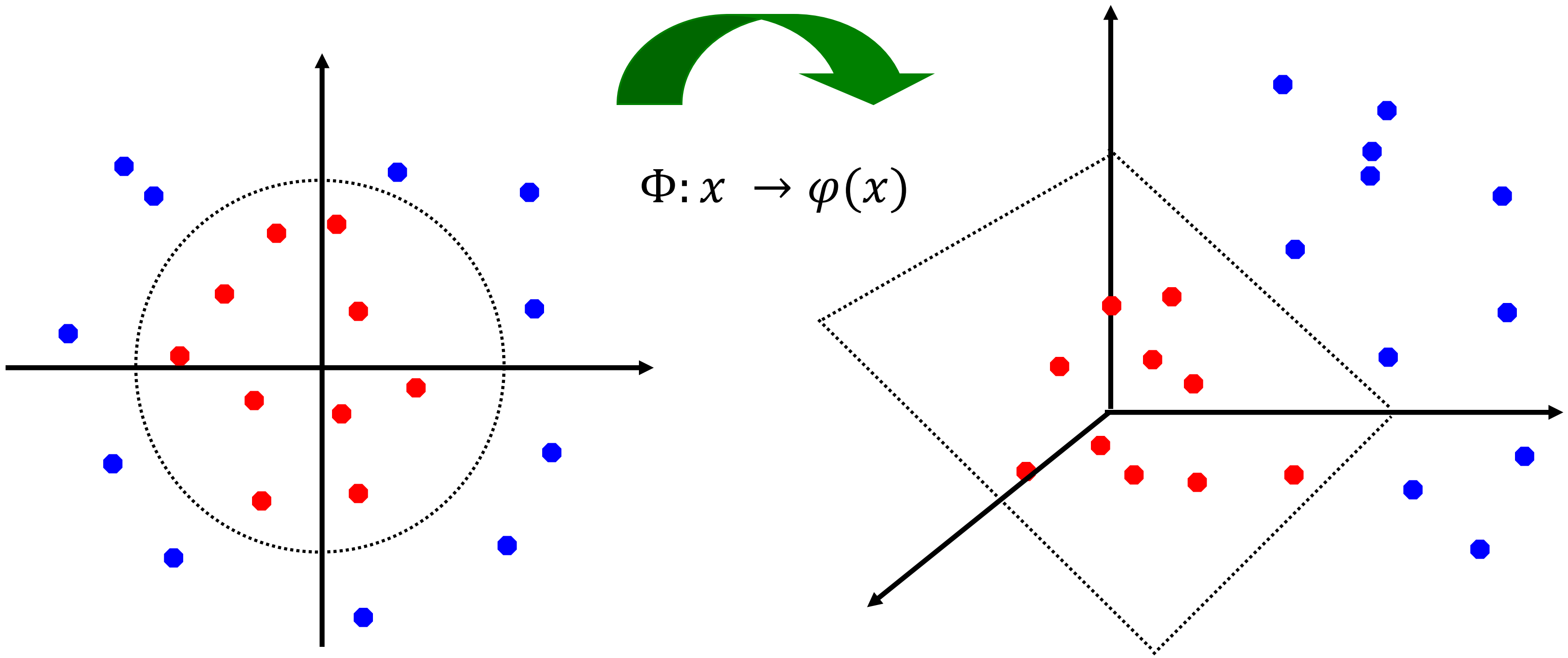


Image credit: Andrew Moore

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* α_i associated with individual training points:
  - $\arg\max_\alpha \sum_i \alpha_i \quad - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j : \sum_i \alpha_i y_i = 0, 0 \leq \alpha_i \leq \frac{1}{\lambda}$
  - At the optimum, α_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*
- This gives a classifier of the form:

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

Dual SVM formulation

- $\arg\max_{\alpha} \sum_i \alpha_i \quad -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i^T} \boldsymbol{x_j} : \sum_i \alpha_i y_i = 0, 0 \le \alpha_i \le \frac{1}{\lambda}$
- $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i^T} \boldsymbol{x_j}.$
- $\arg\max_{\alpha} \sum_i \alpha_i \quad -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) : \sum_i \alpha_i y_i = 0, 0 \le \alpha_i \le \frac{1}{\lambda}$
- This gives a classifier of the form:

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

- How about we compute similarity in a different space φ?

$$K(x_i, x_j) = \varphi(\boldsymbol{x_i})^T \varphi(\boldsymbol{x_j}).$$

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 must be positive-definite and symmetric)

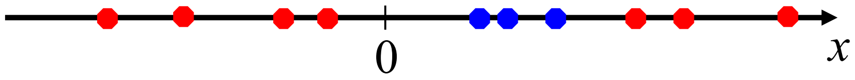
- The learned classifier takes the form

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

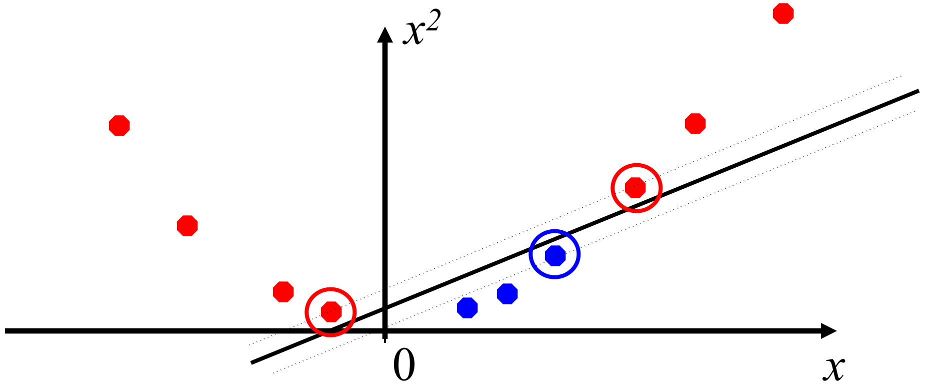
  - This gives a nonlinear decision boundary in the original feature space

Toy example

- Non-separable data in 1D:



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



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

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 $d = 2$:

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

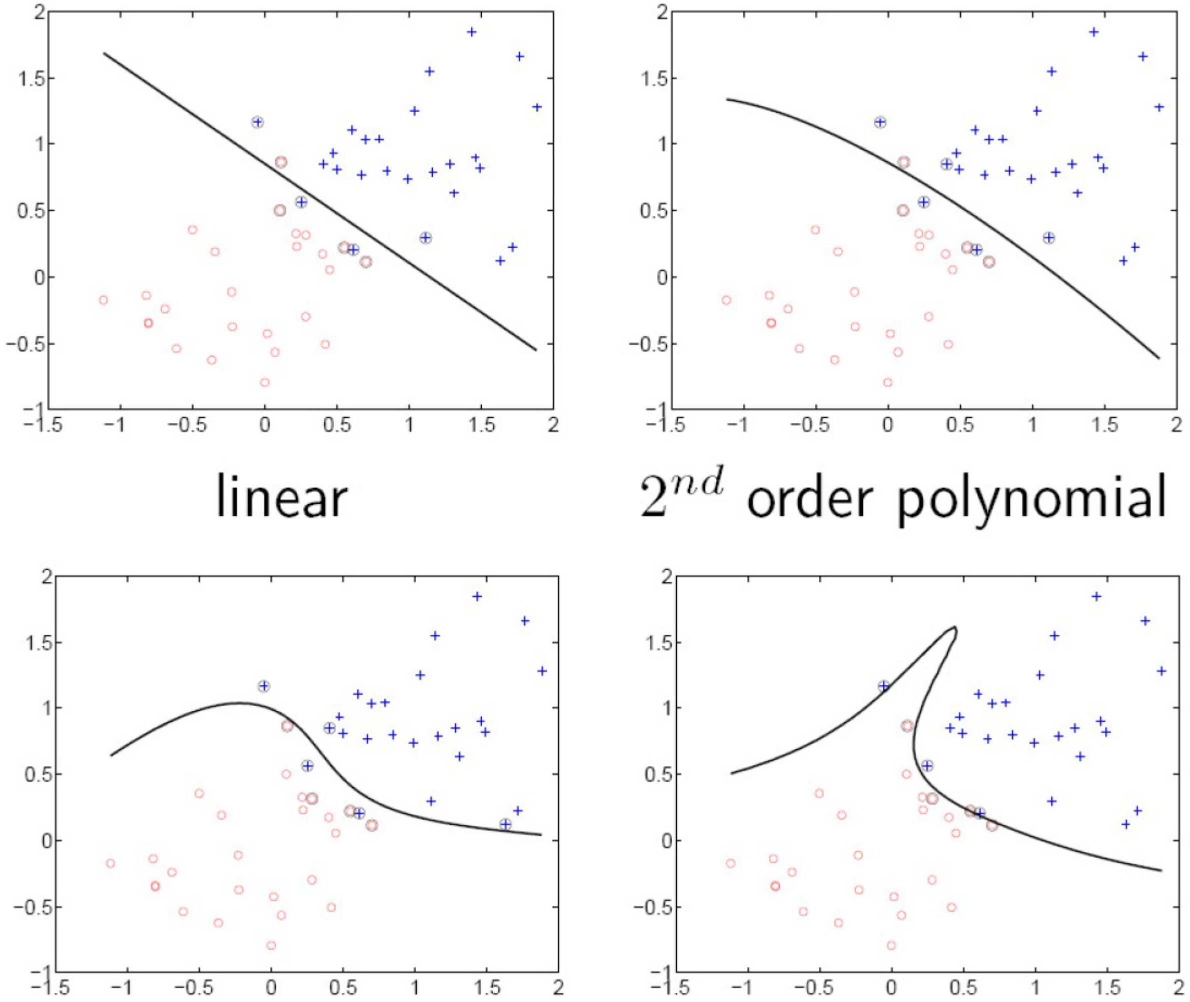
$$K(x, x') = (uu' + vv' + c)^2$$

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

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

- 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

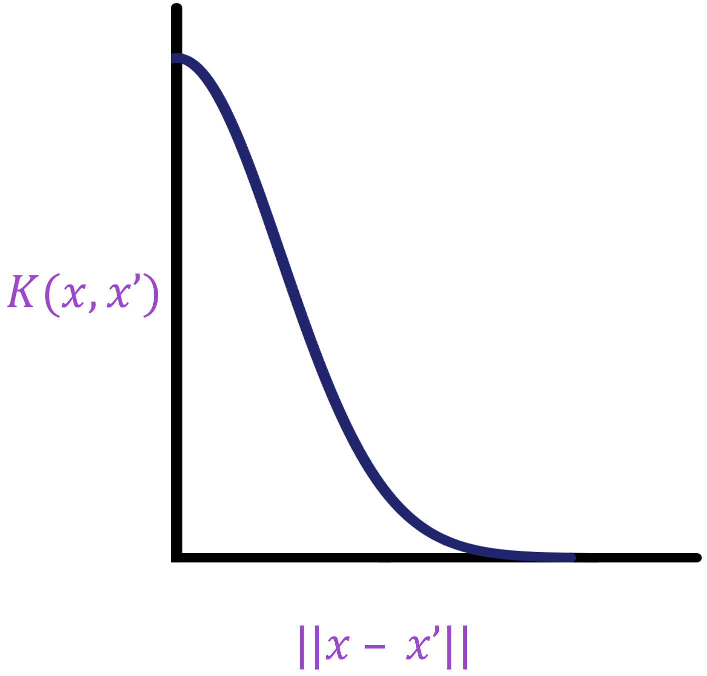
8^{th} order polynomial

Kernel example 2: Gaussian

- Gaussian kernel with bandwidth σ:

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

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

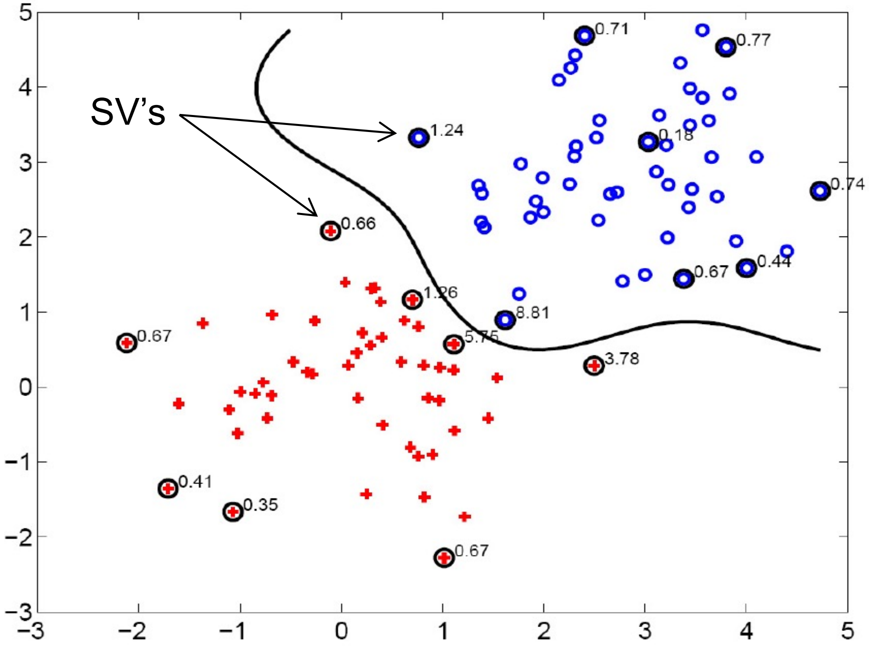


Kernel example 2: Gaussian

- Gaussian kernel with bandwidth σ:

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

  - It’s also called the Radial Basis Function (RBF) kernel
- 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

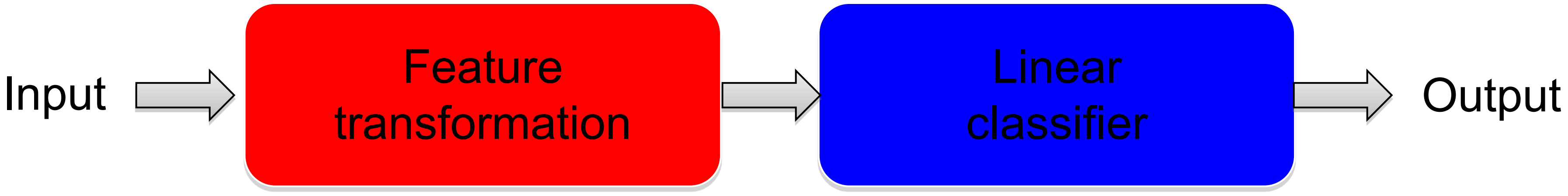


SVM: Pros and cons

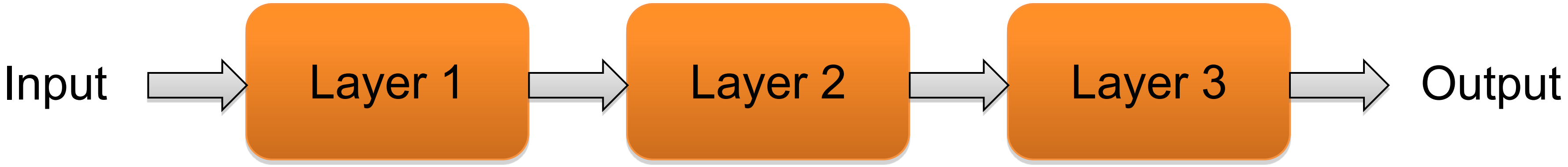
- Pros
  - Margin maximization and kernel trick are elegant, amenable to convex optimization and theoretical analysis
  - Kernel SVMs are flexible, can be used with problem-specific kernels
  - SVM loss gives very good accuracy in practice
  - Perfect “off-the-shelf” classifier, many packages are available
  - Linear SVMs can scale to large datasets
- Con
  - Kernel SVM training does not scale to large datasets: memory cost is quadratic and computation cost even worse

Overview

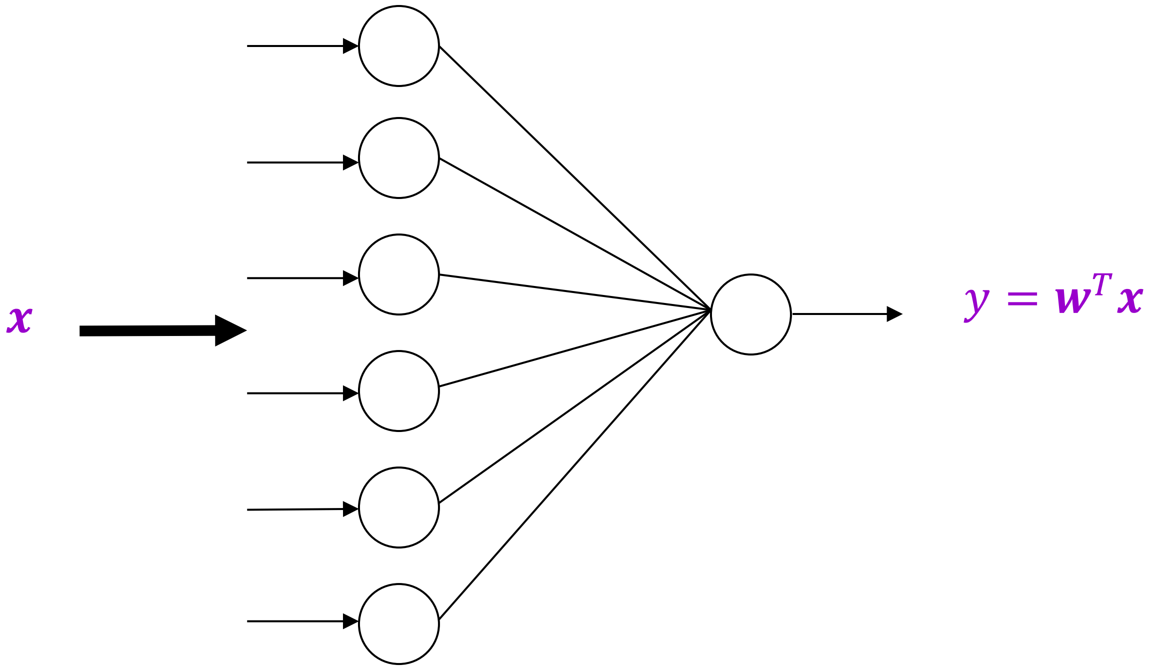
- Feature Design
- Nonlinear classifiers
  - “Shallow” approach: Kernel support vector machines (SVMs)



  - “Deep” approach: Multi-layer neural networks

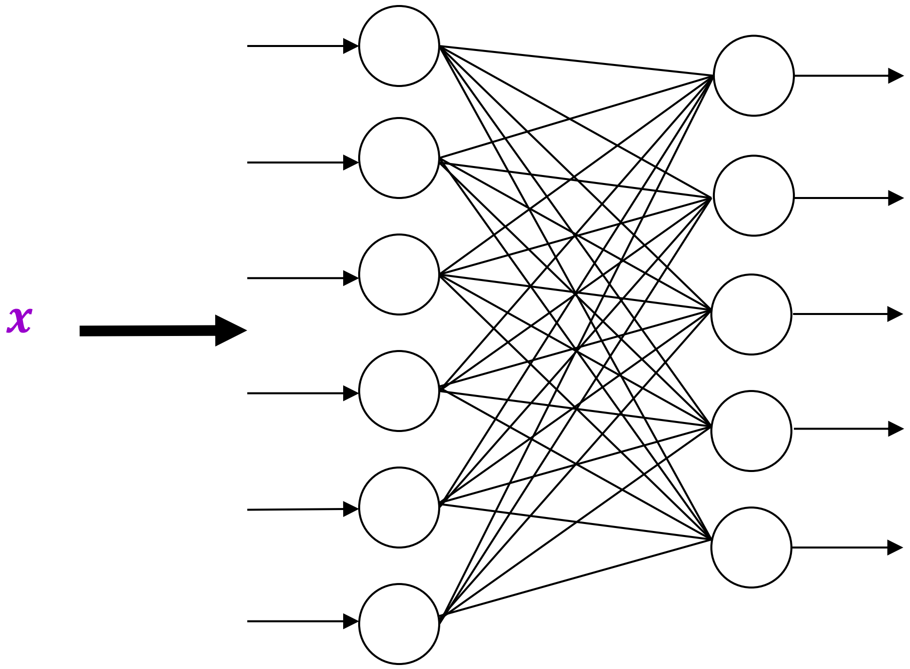


From linear classifiers to multi-layer networks



From linear classifiers to multi-layer networks

Linear layer

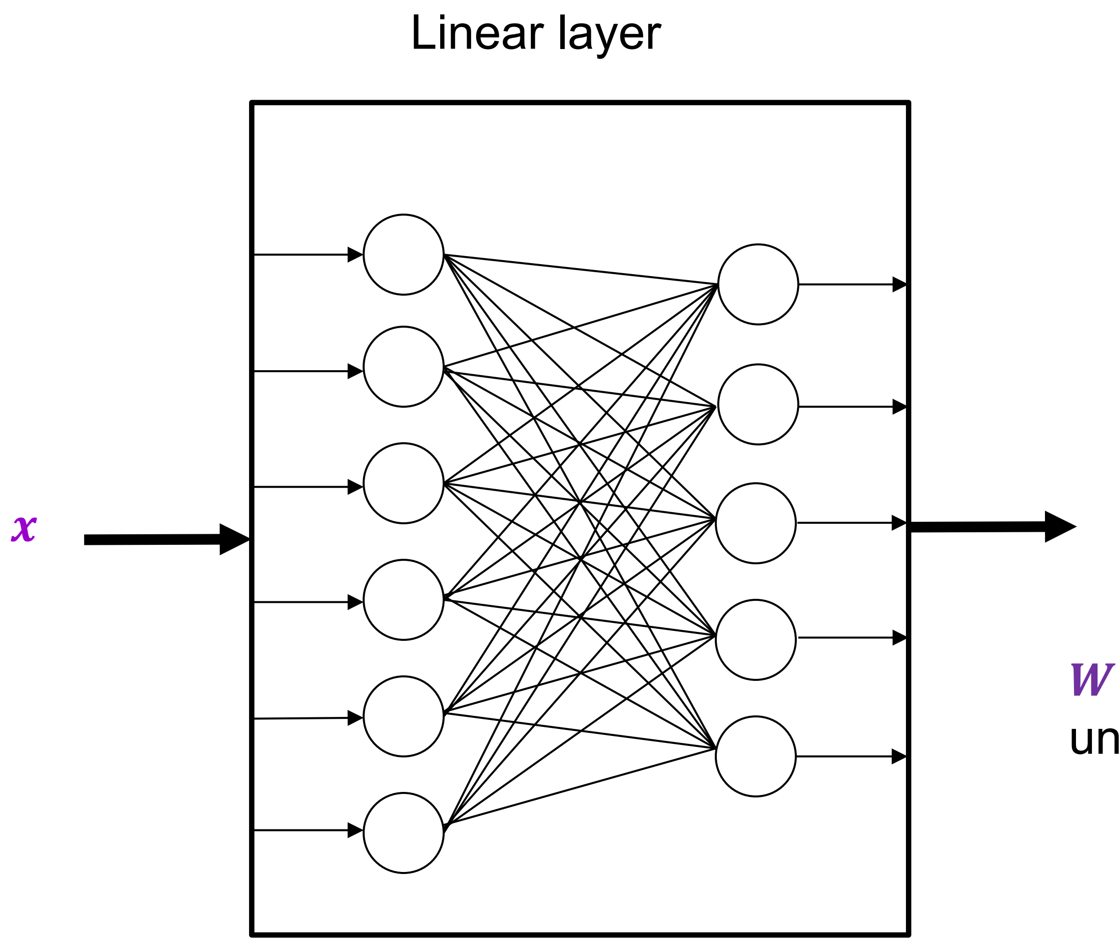


$$y^{(1)} = \boldsymbol{w}^{(1)} \cdot \boldsymbol{x}$$

$$y^{(2)} = \boldsymbol{w}^{(2)} \cdot \boldsymbol{x}$$

$$\vdots$$

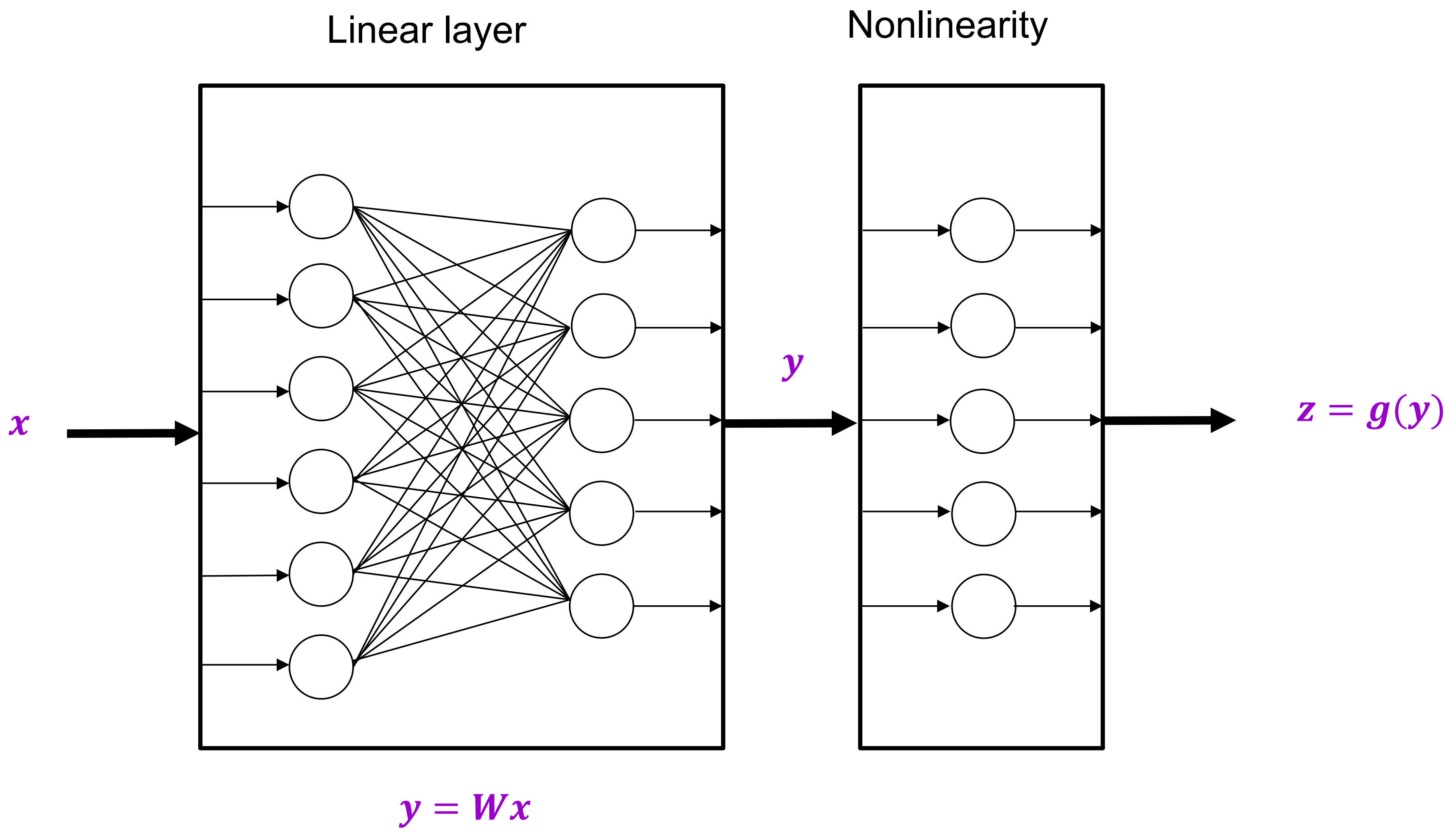
From linear classifiers to multi-layer networks



$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$

$\boldsymbol{W}$: matrix whose rows are weights of output units $\boldsymbol{w}^{(k)}$

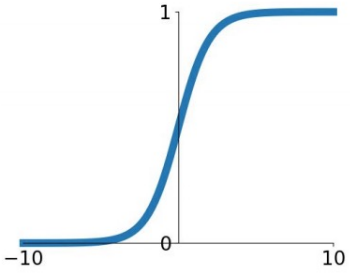
From linear classifiers to multi-layer networks


Linear layer
Nonlinearity
$\boldsymbol{x}$
$\boldsymbol{y}$
$\boldsymbol{z} = \boldsymbol{g}(\boldsymbol{y})$
$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$

Common nonlinearities (or *activation functions*)

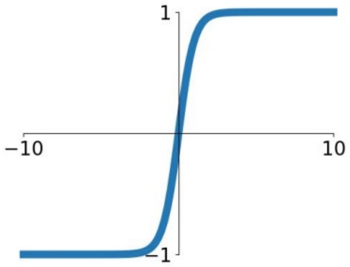
Sigmoid

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



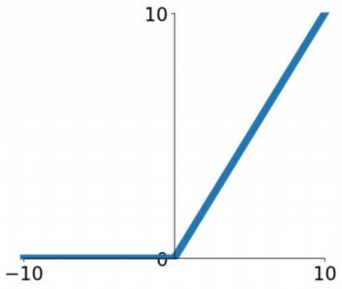
tanh

$\tanh(x)$



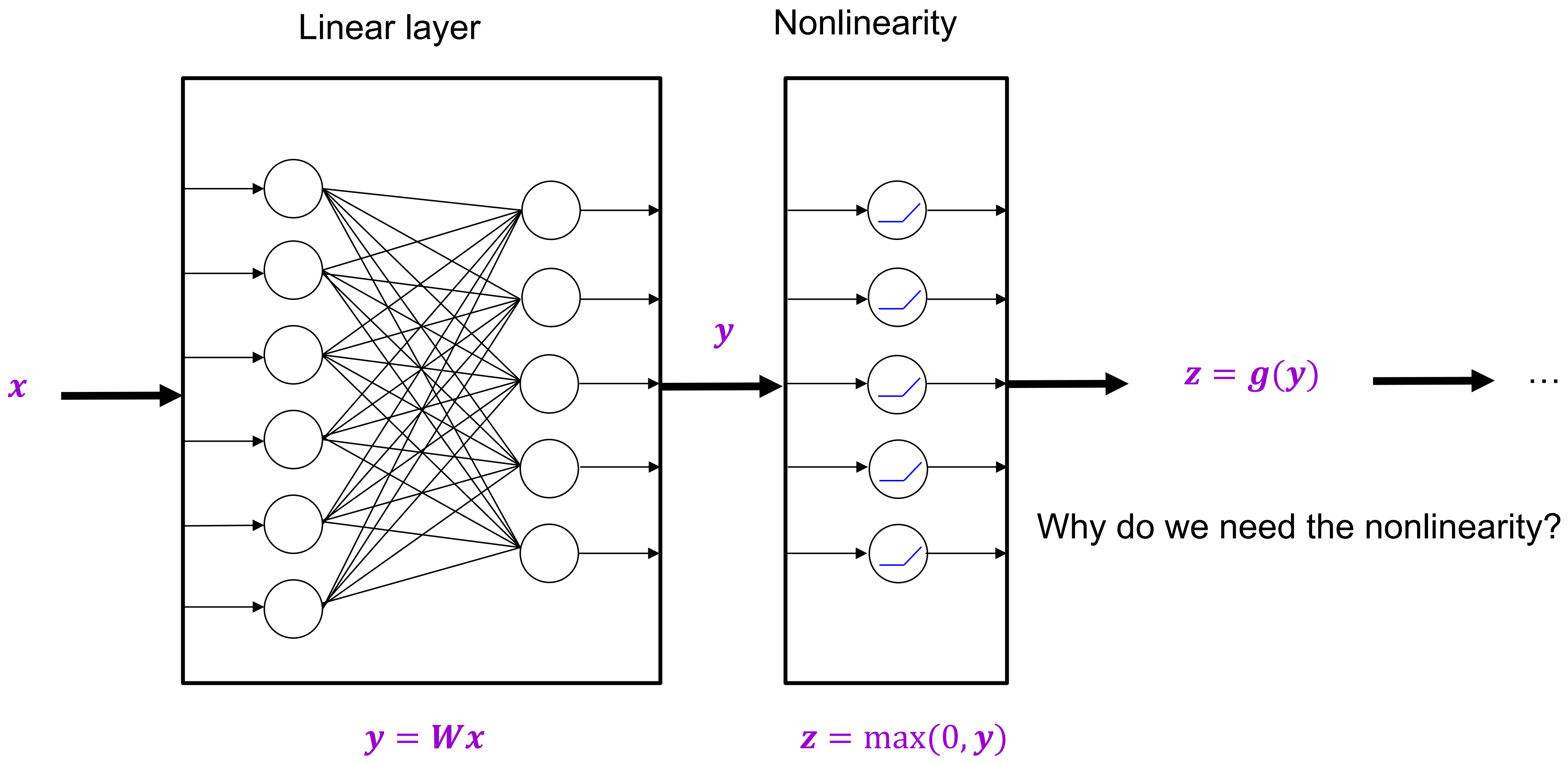
ReLU

$\max(0, x)$



Source: Stanford 231n

From linear classifiers to multi-layer networks



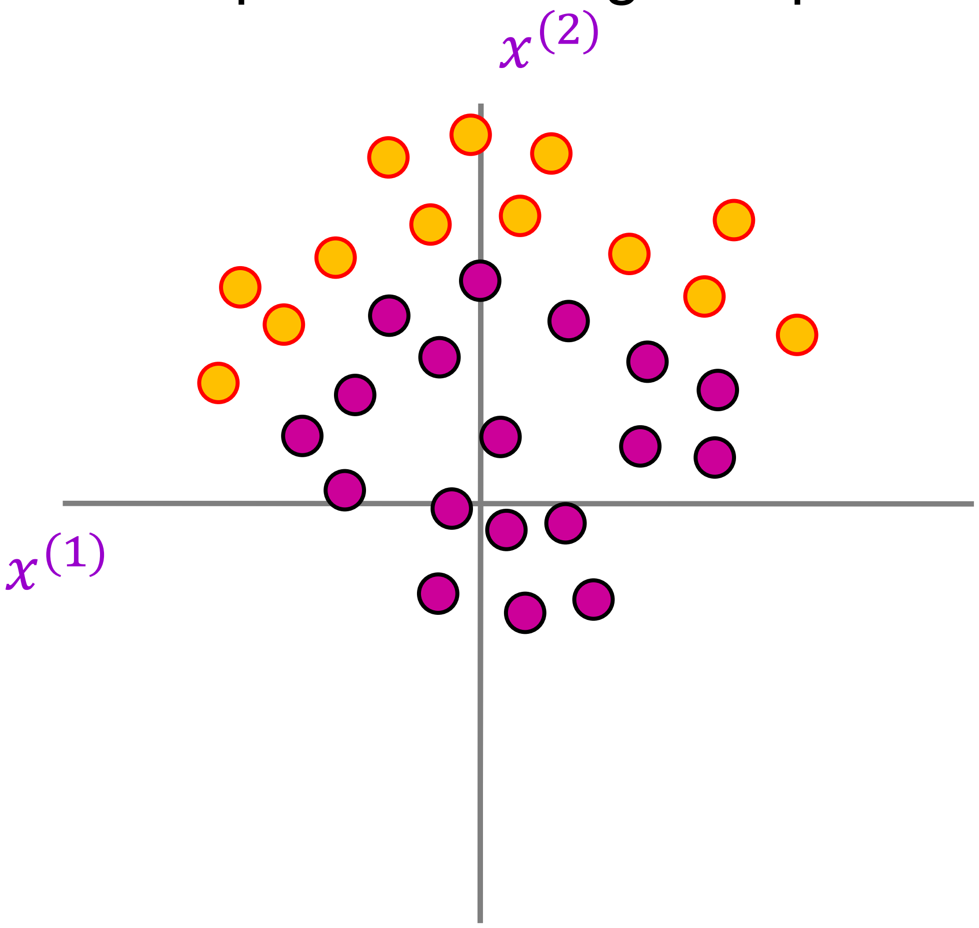
$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$

$\boldsymbol{z} = \max(0, \boldsymbol{y})$

Why do we need the nonlinearity?

The power of nonlinearities

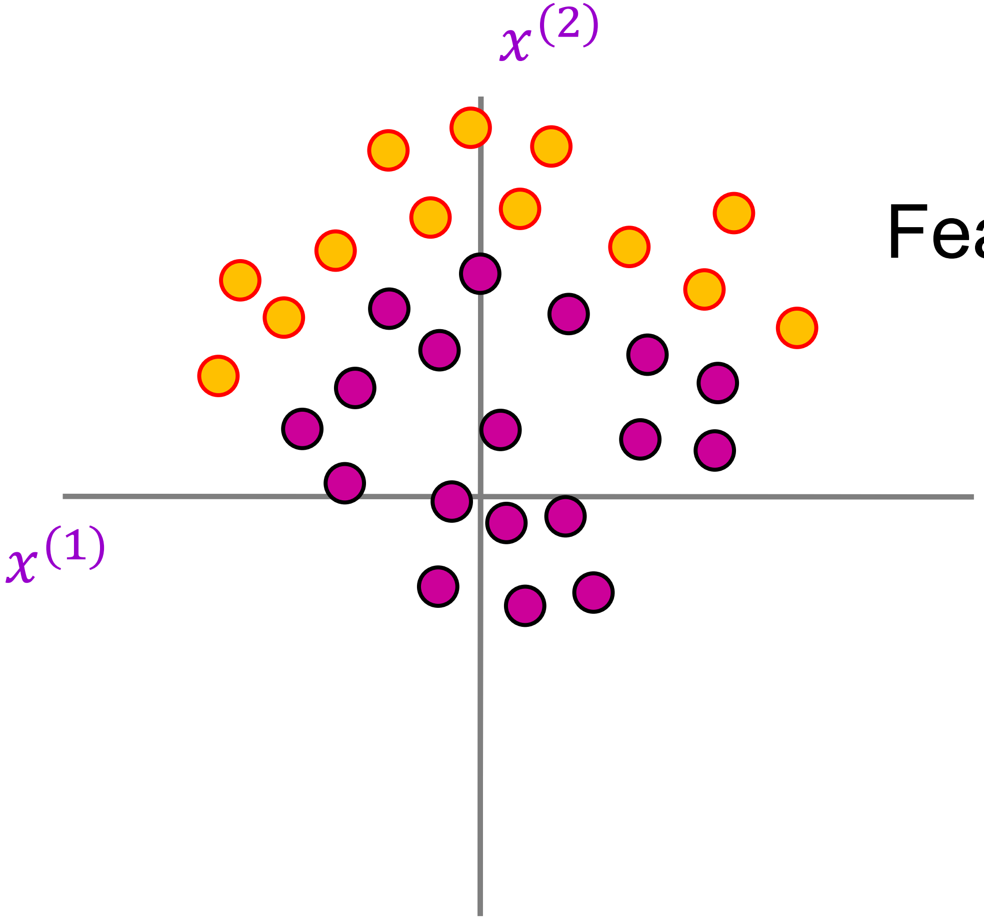
Points not linearly separable in original space



Source: J. Johnson

The power of nonlinearities

Points not linearly separable in original space



Consider a linear transform: $h = Wx + b$
Where x, h, b are 2-dimensional

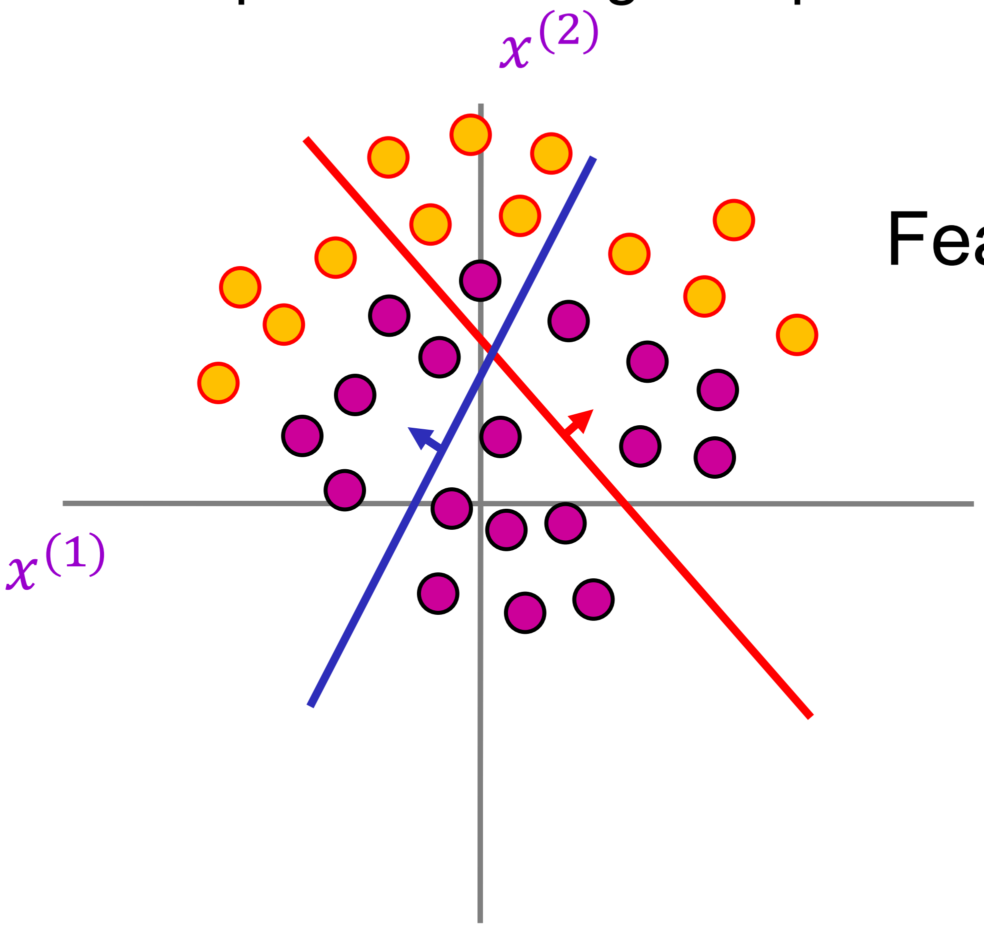
Feature transform:
$h = Wx + b$

Source: J. Johnson

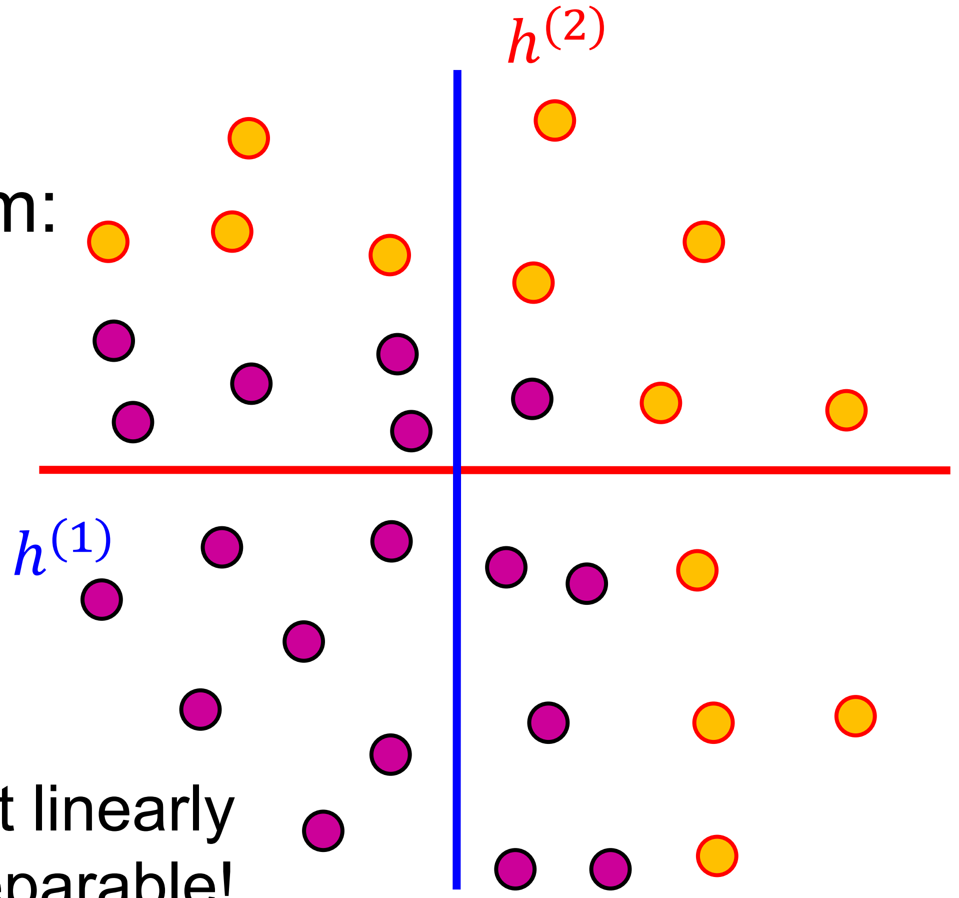
The power of nonlinearities

Points not linearly separable in original space

Consider a linear transform: $h = Wx + b$
Where x, h, b are 2-dimensional



Feature transform:
$h = Wx + b$



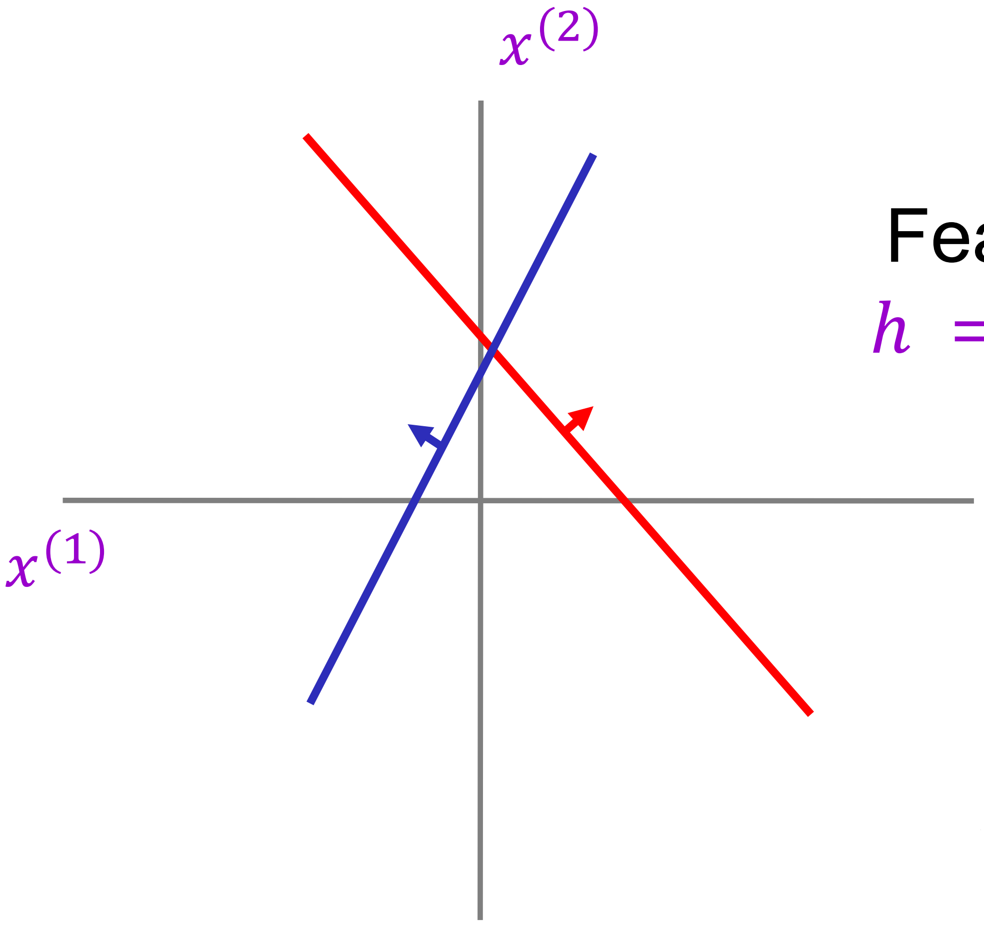
Still not linearly separable!

Source: J. Johnson

The power of nonlinearities

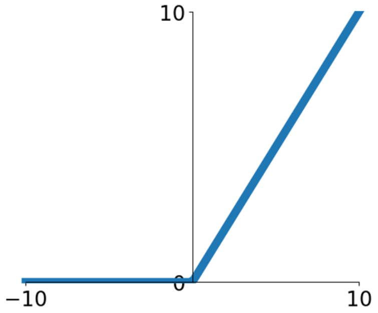
Let's add a nonlinearity:

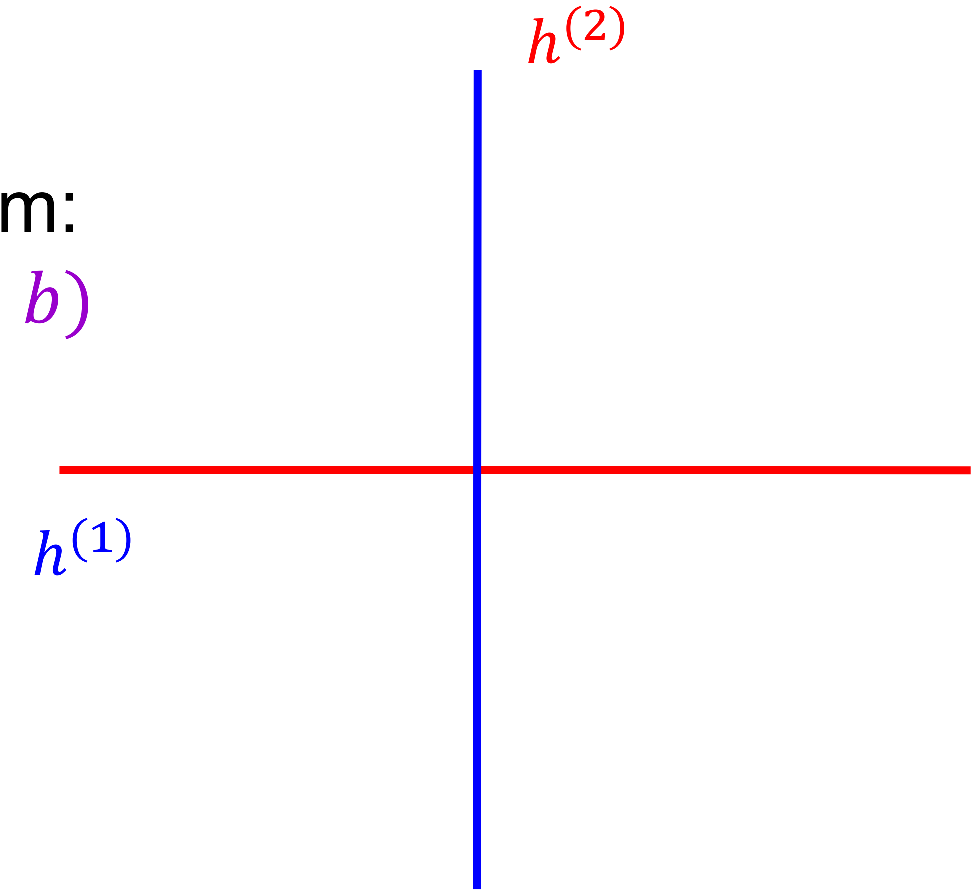
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

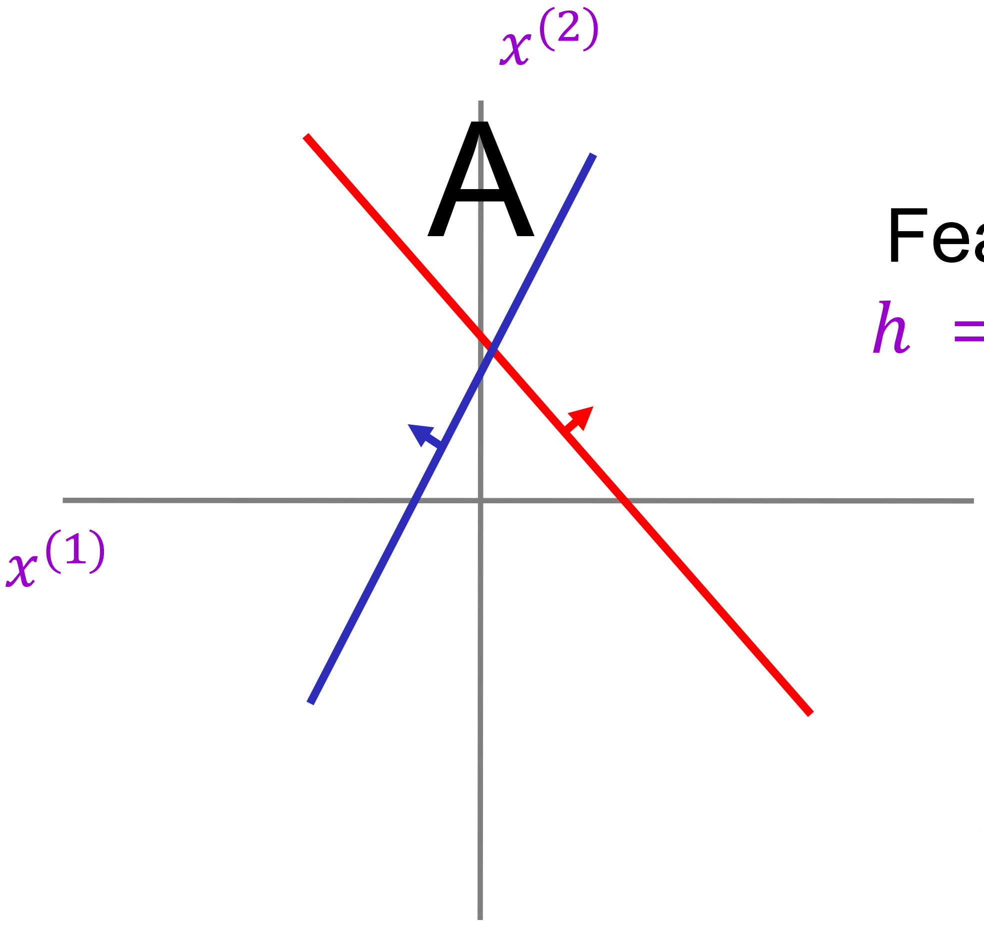
$$h = \mathrm{ReLU}(Wx + b)$$





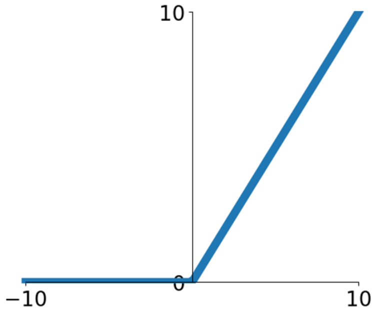
Source: J. Johnson

The power of nonlinearities



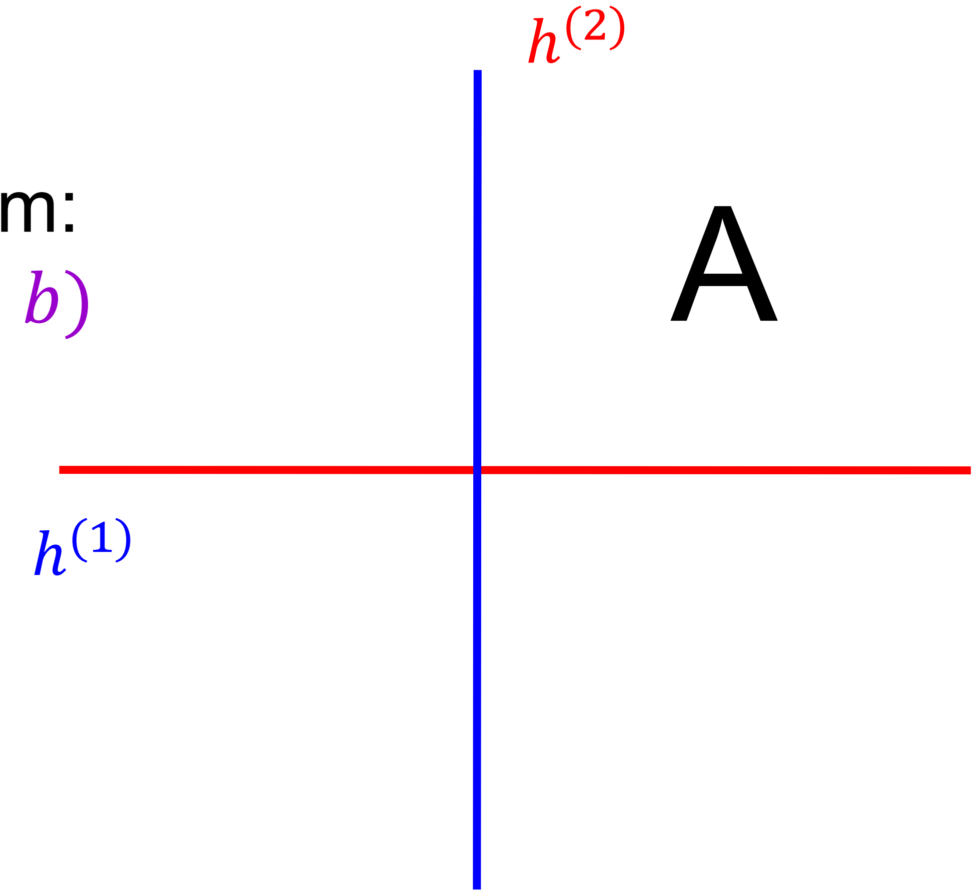
Feature transform:

$$h = \text{ReLU}(Wx + b)$$



Let’s add a nonlinearity:

$$h = \text{ReLU}(Wx + b) = \max(0, Wx + b)$$

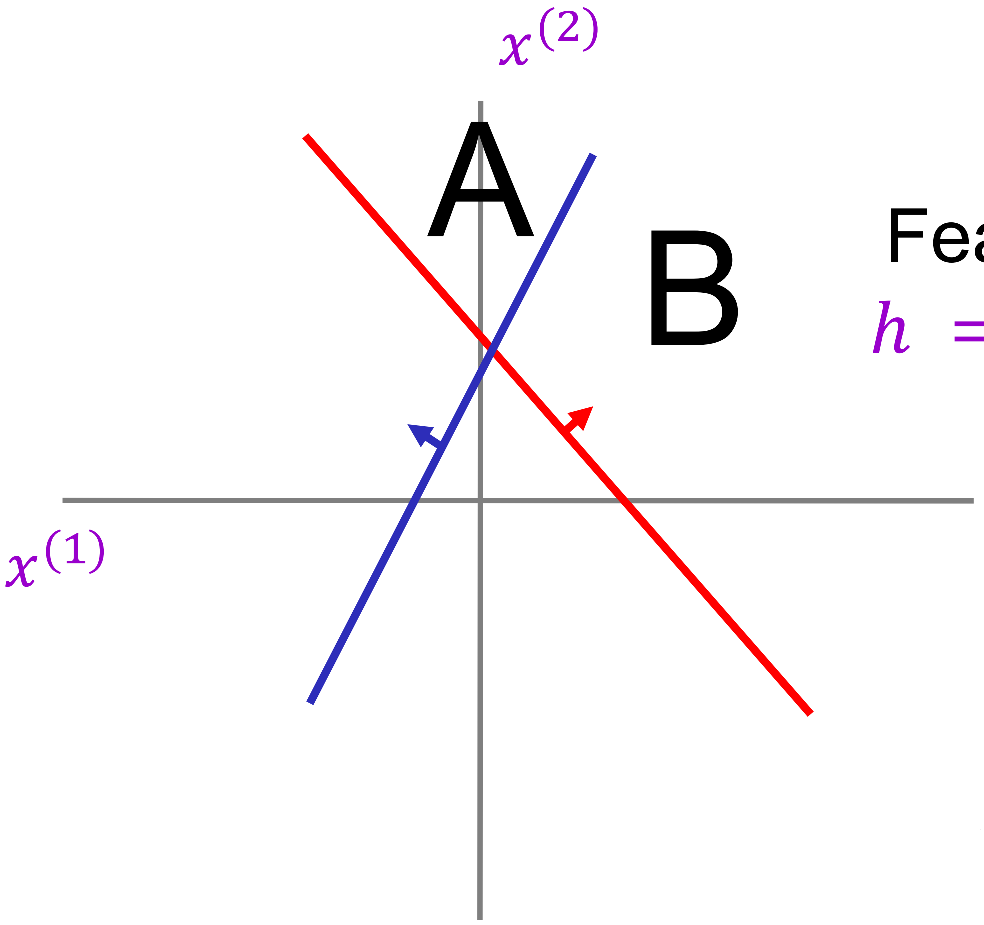


Source: J. Johnson

The power of nonlinearities

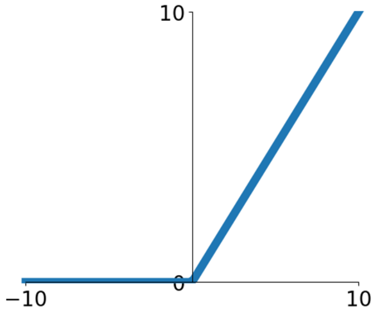
Let’s add a nonlinearity:

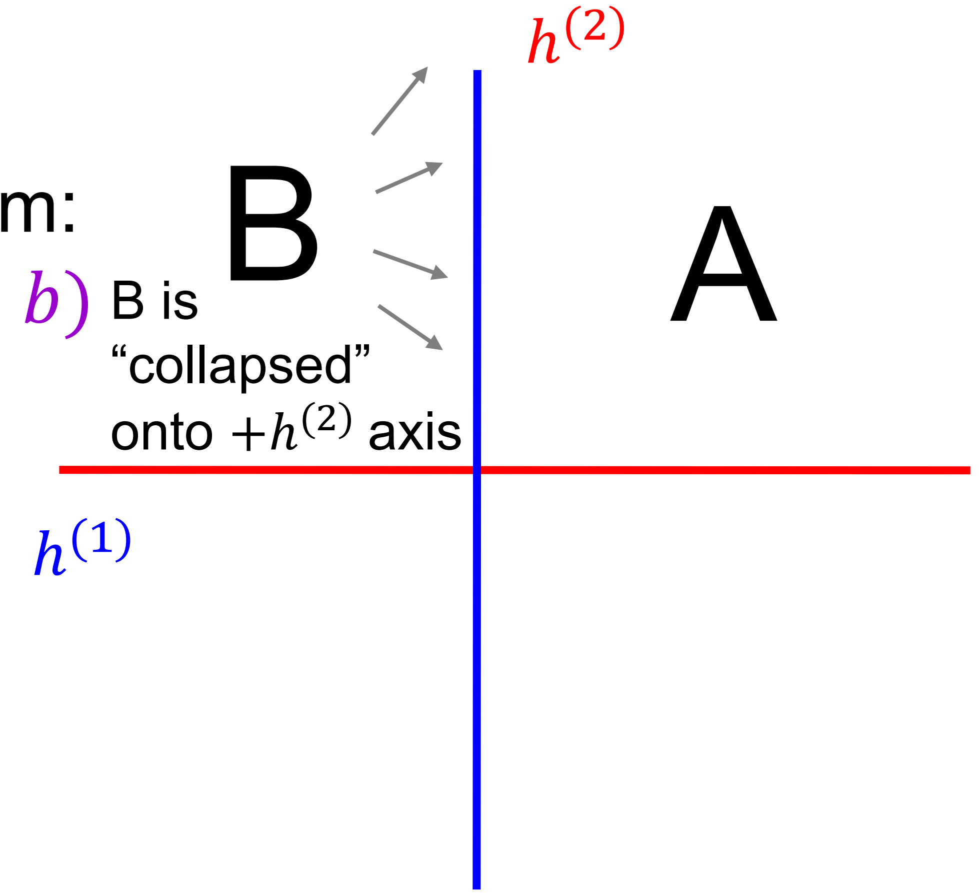
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$





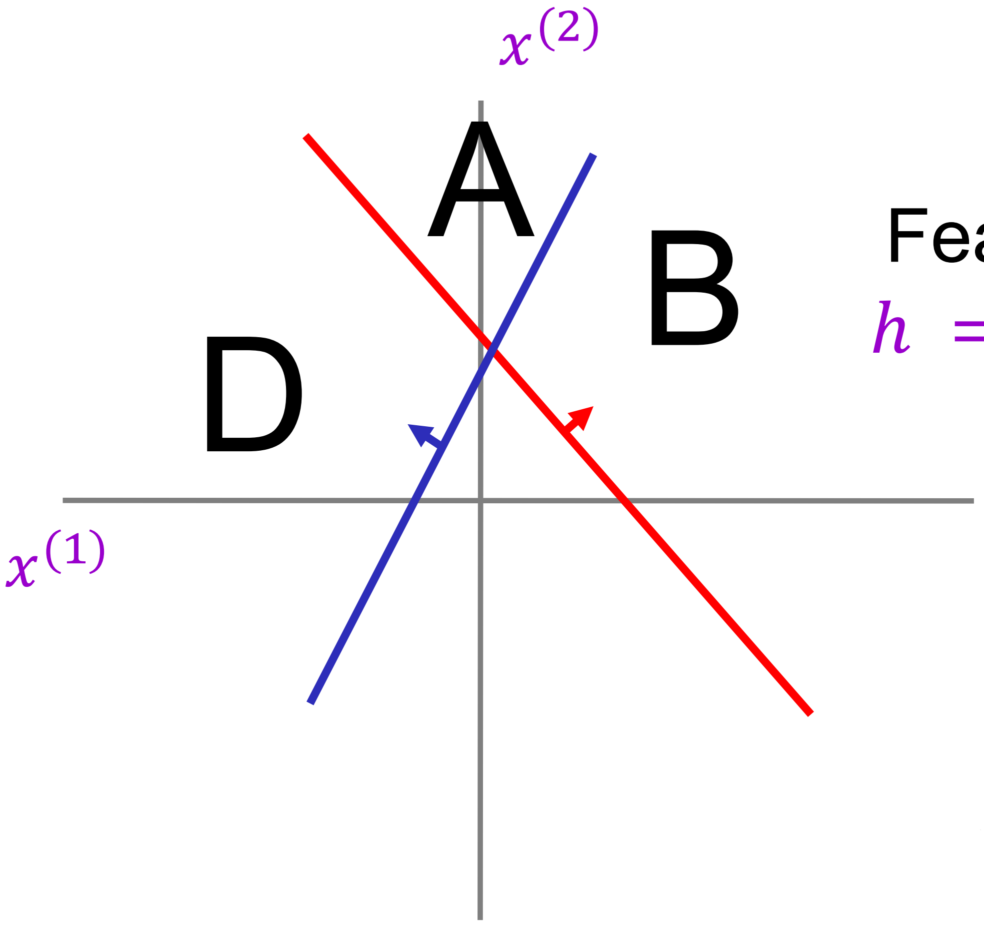
B is “collapsed” onto $+h^{(2)}$ axis

Source: J. Johnson

The power of nonlinearities

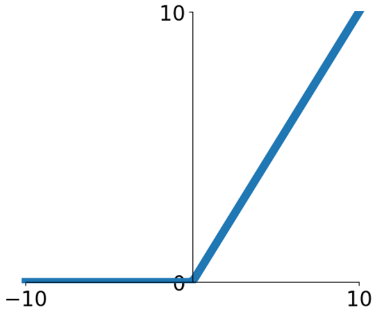
Let's add a nonlinearity:

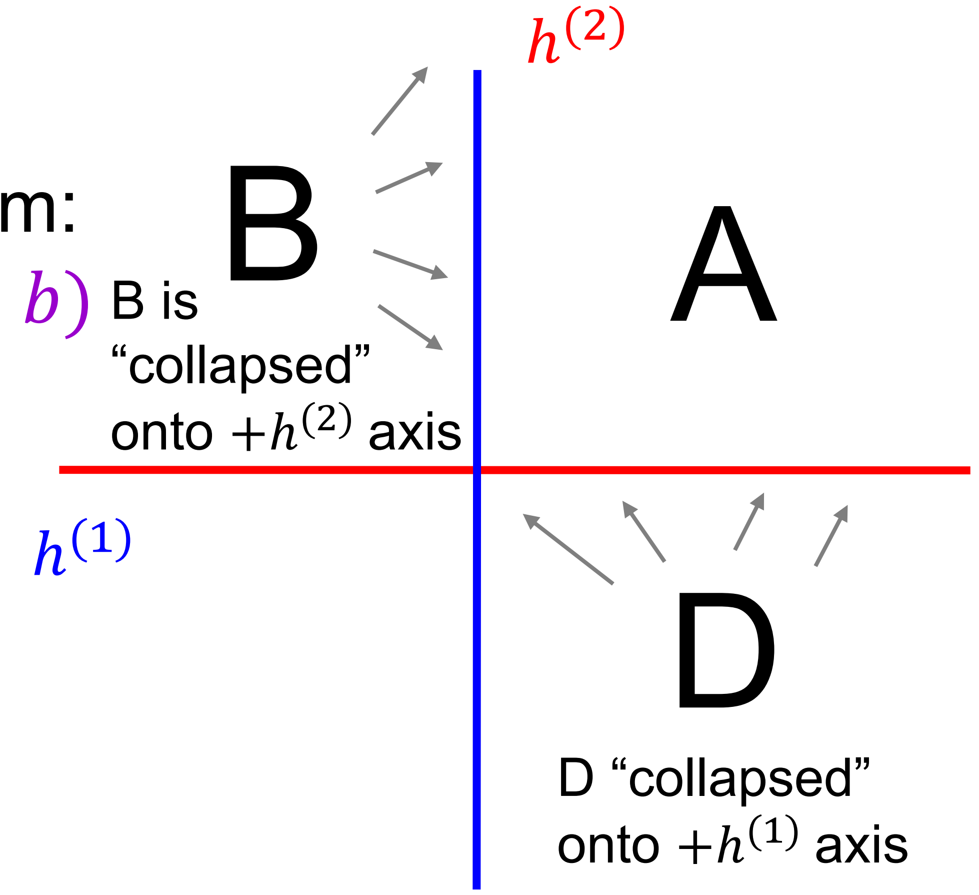
$$h = \text{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

$$h = \text{ReLU}(Wx + b)$$





B is "collapsed" onto $+h^{(2)}$ axis

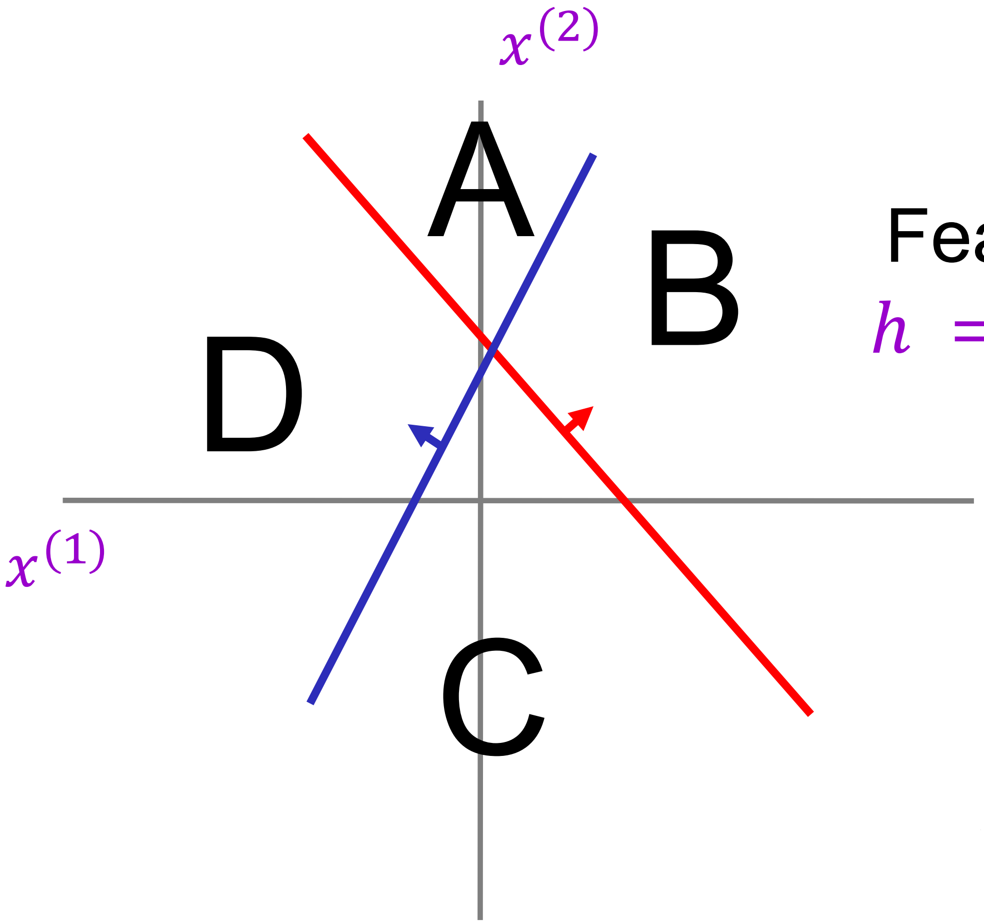
D "collapsed" onto $+h^{(1)}$ axis

Source: J. Johnson

The power of nonlinearities

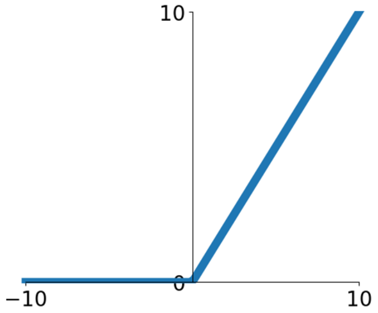
Let's add a nonlinearity:

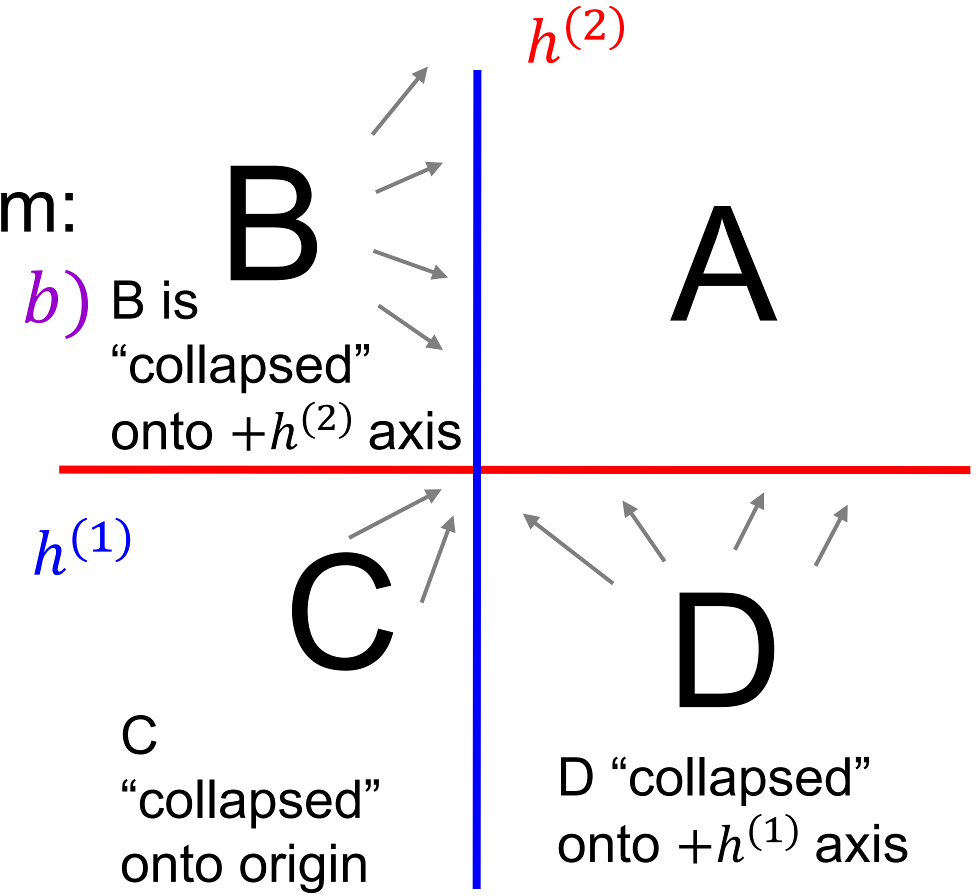
$$h = \mathrm{ReLU}(Wx+b) = \max(0, Wx+b)$$



Feature transform:

$$h = \mathrm{ReLU}(Wx+b)$$





B is "collapsed" onto $+h^{(2)}$ axis

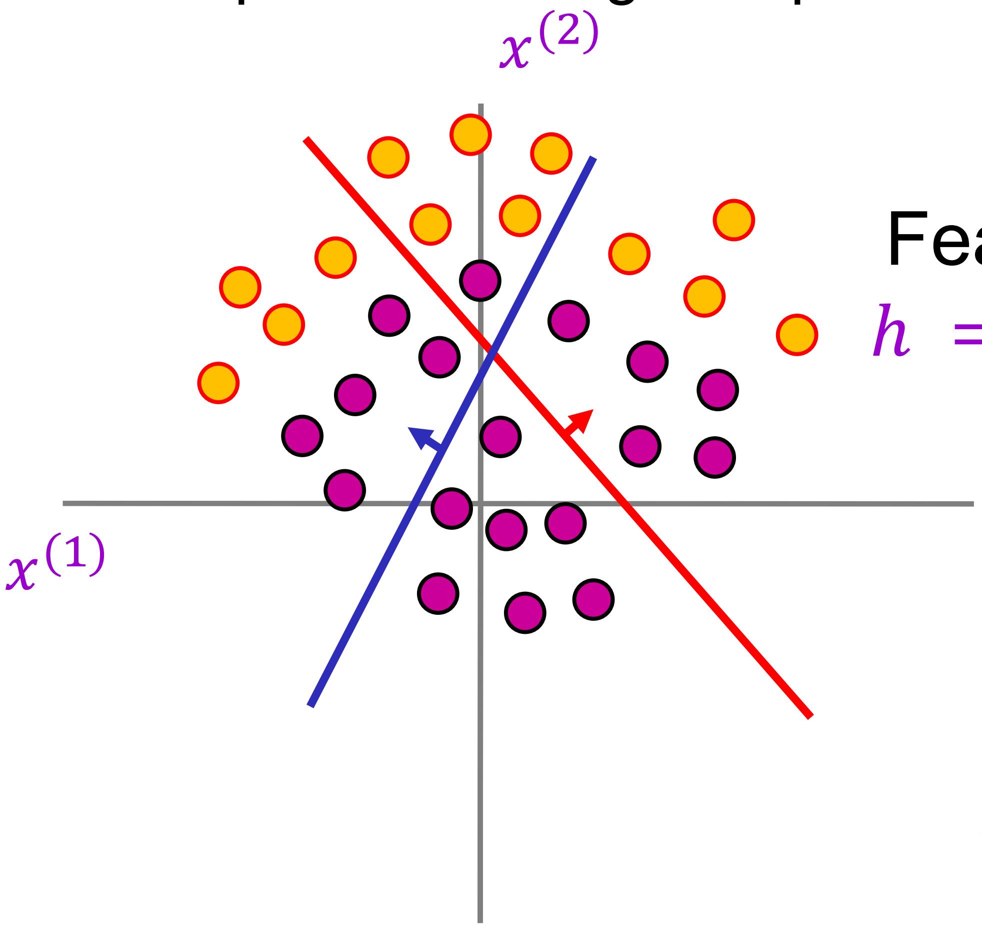
C "collapsed" onto origin

D "collapsed" onto $+h^{(1)}$ axis

Source: J. Johnson

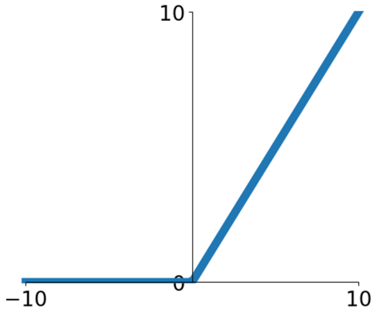
The power of nonlinearities

Points not linearly separable in original space



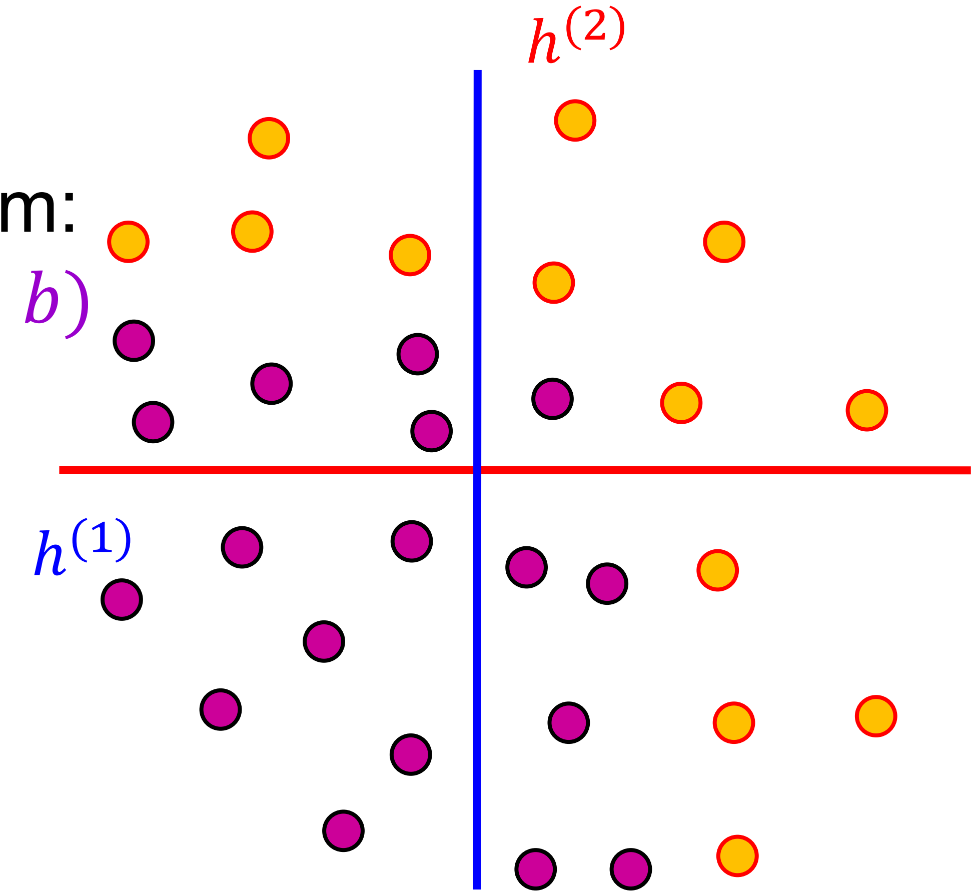
Feature transform:

$h = \mathrm{ReLU}(Wx + b)$



Let’s add a nonlinearity:

$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$



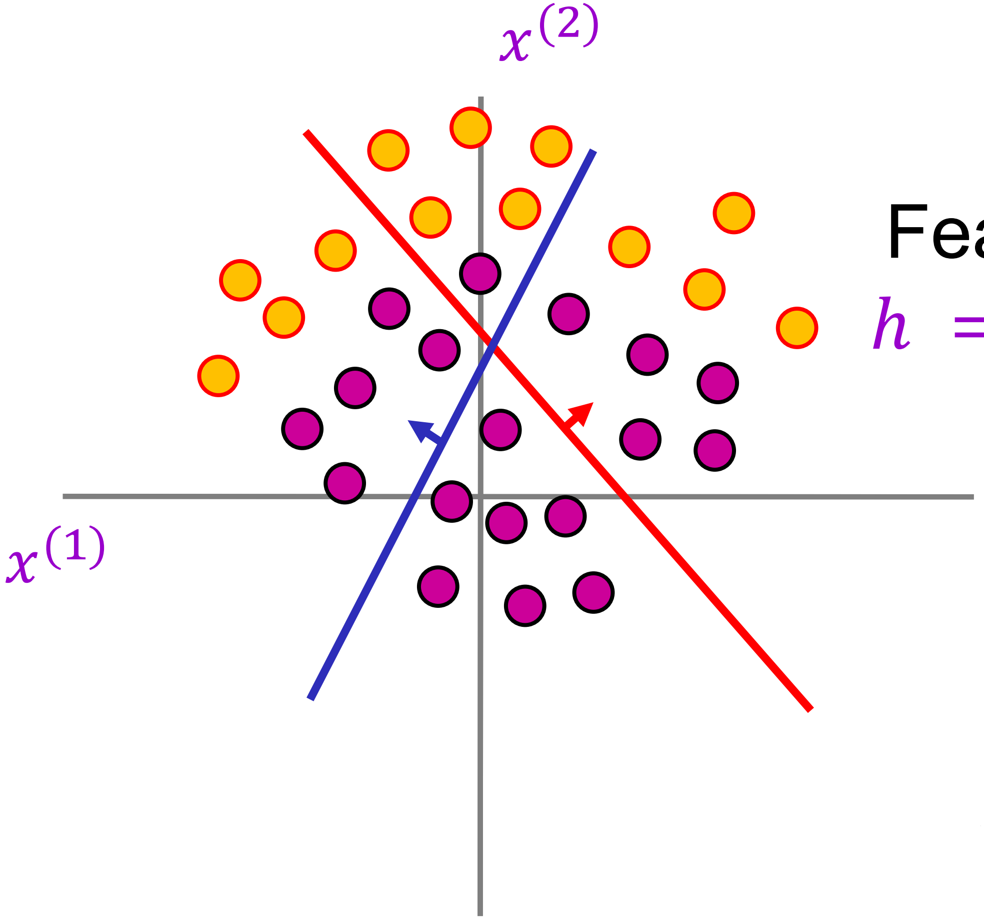
Source: J. Johnson

The power of nonlinearities

Points not linearly separable in original space

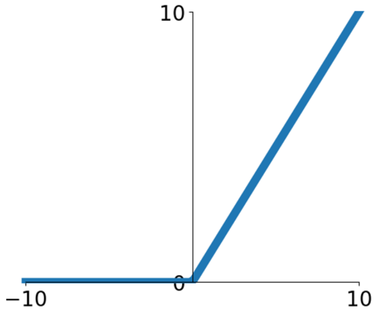
Let’s add a nonlinearity:

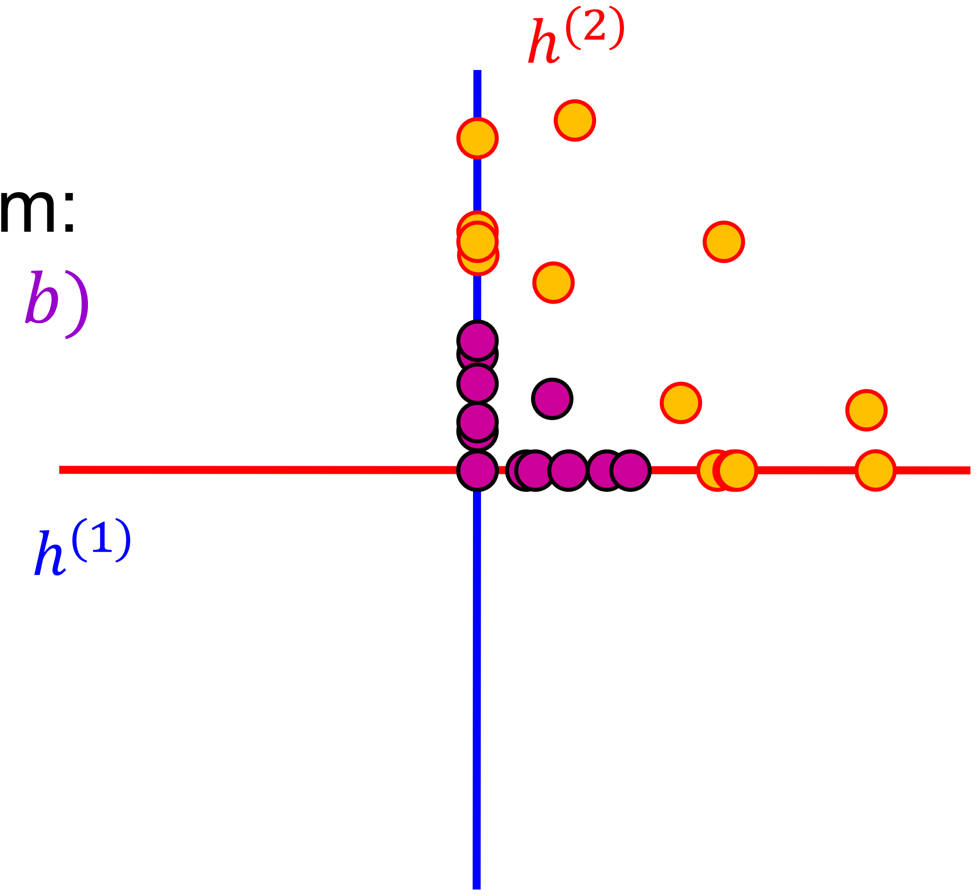
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$





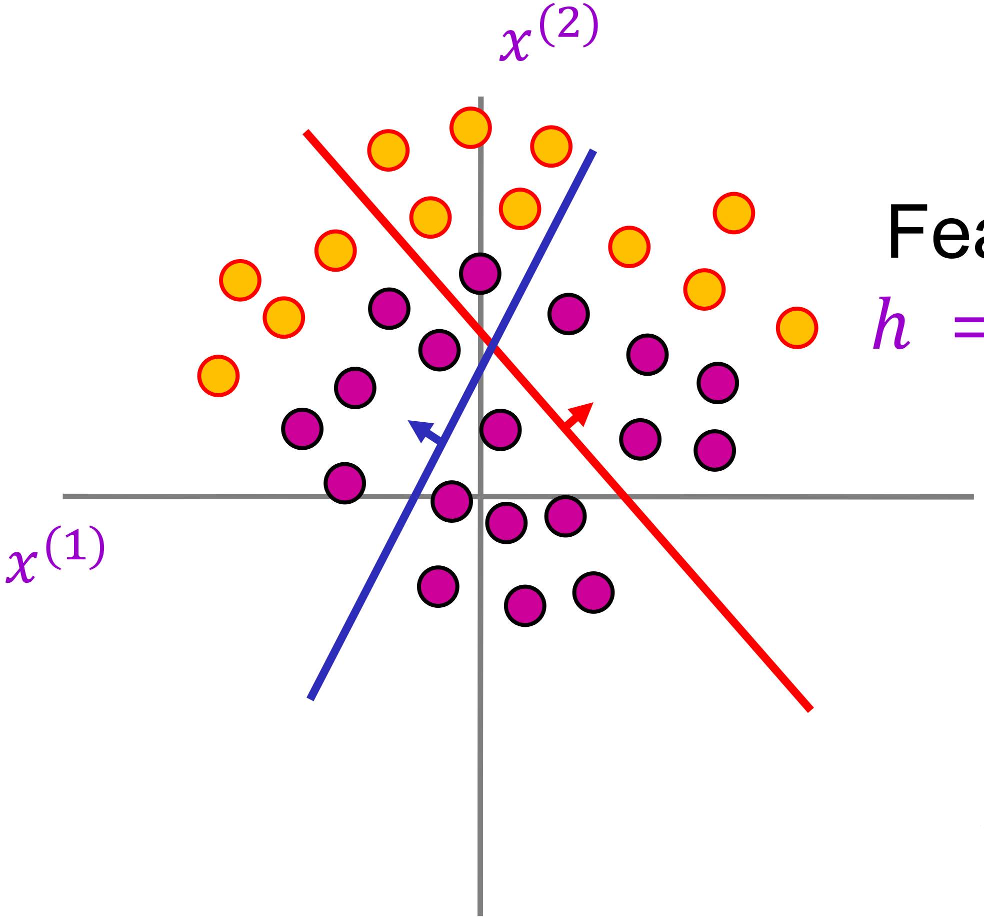
Source: J. Johnson

The power of nonlinearities

Points not linearly separable in original space

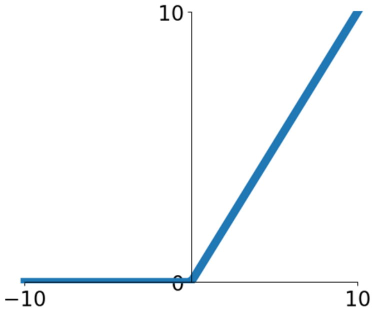
Let’s add a nonlinearity:

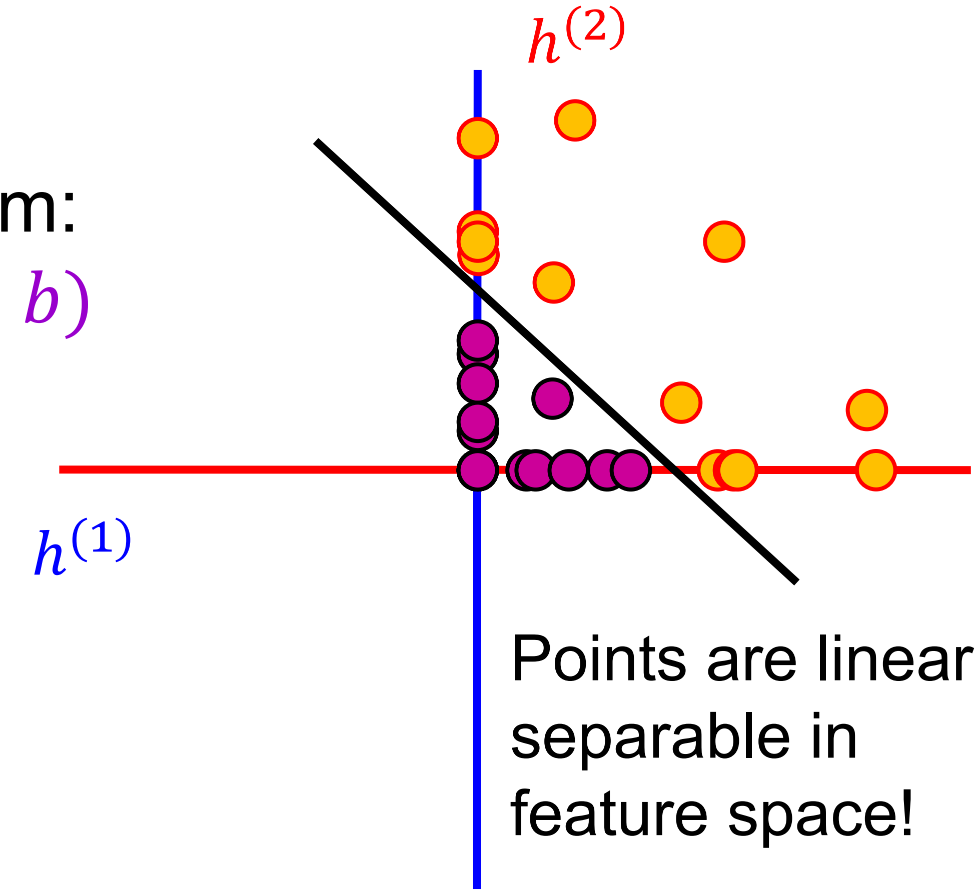
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$





Points are linearly separable in feature space!

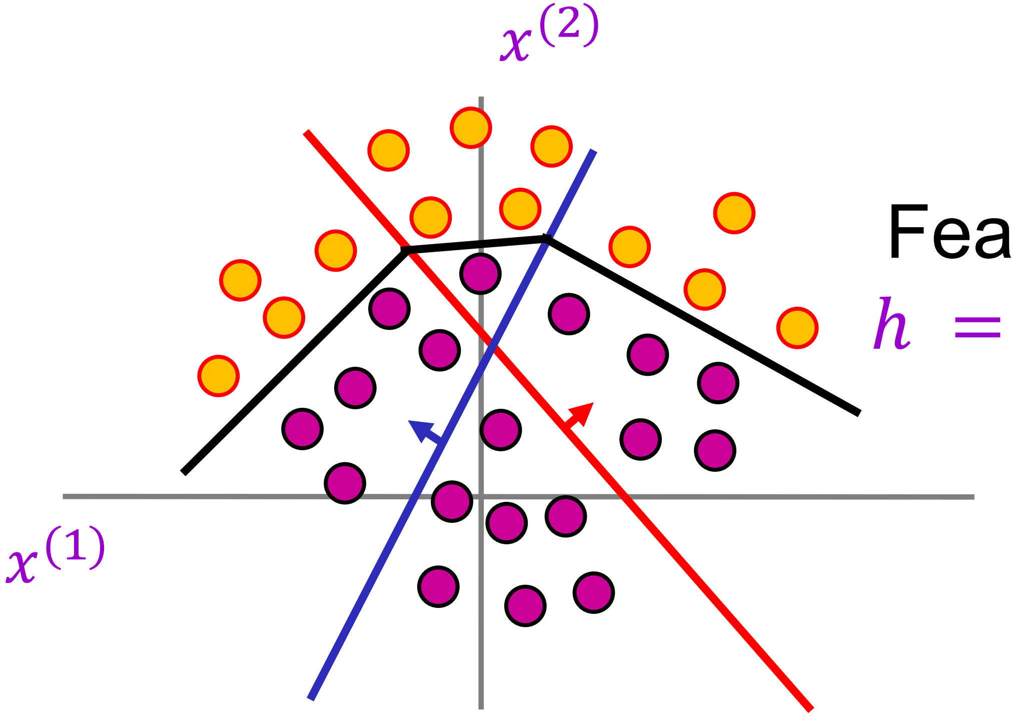
Source: J. Johnson

The power of nonlinearities

Points not linearly separable in original space

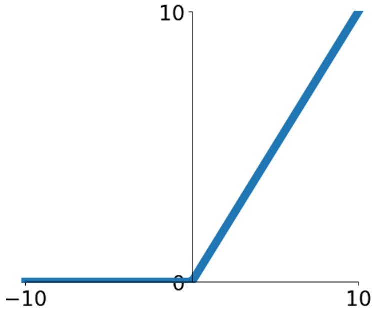
Let’s add a nonlinearity:

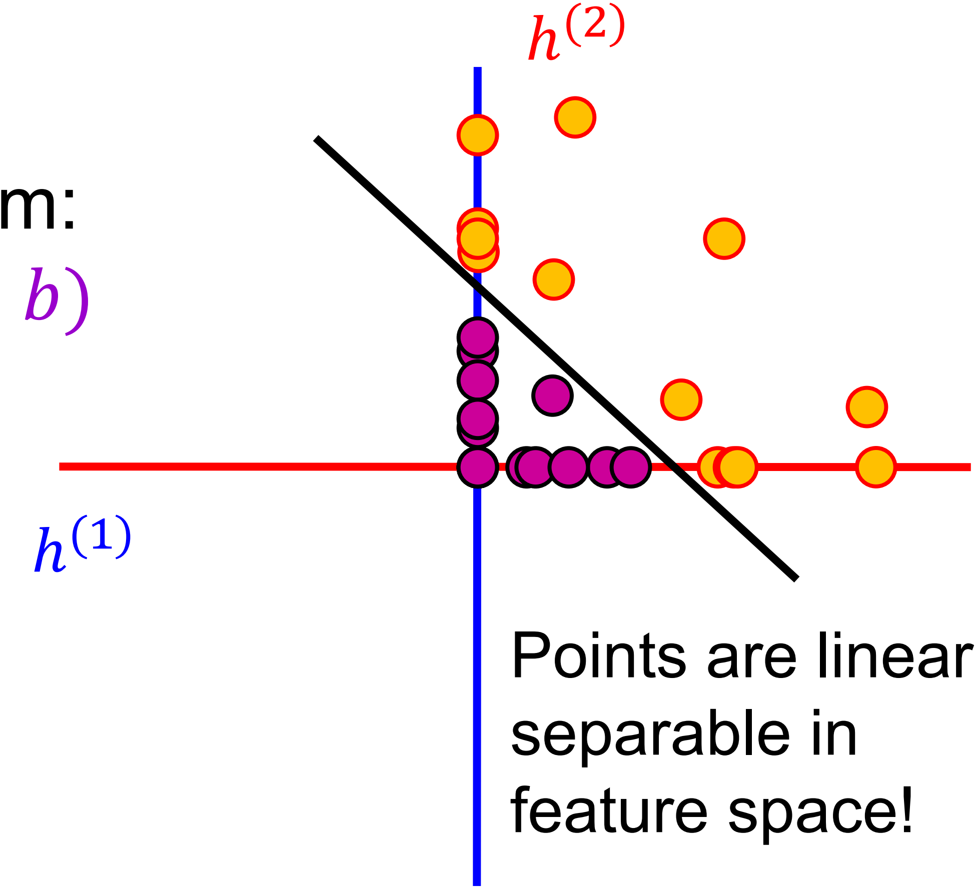
$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$



Feature transform:

$h = \mathrm{ReLU}(Wx + b)$



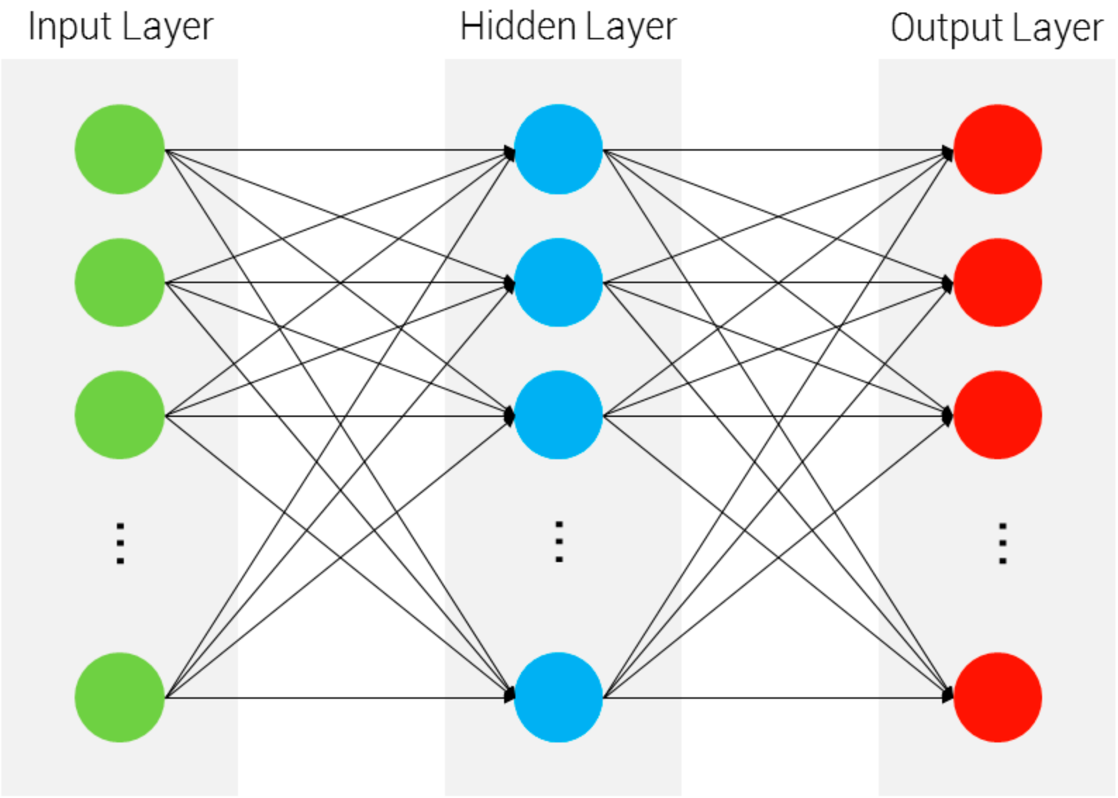


Linear classifier in feature space gives nonlinear classifier in original space

Points are linearly separable in feature space!

Source: J. Johnson

Two-layer neural network



Individual dimensions of $\boldsymbol{x}$

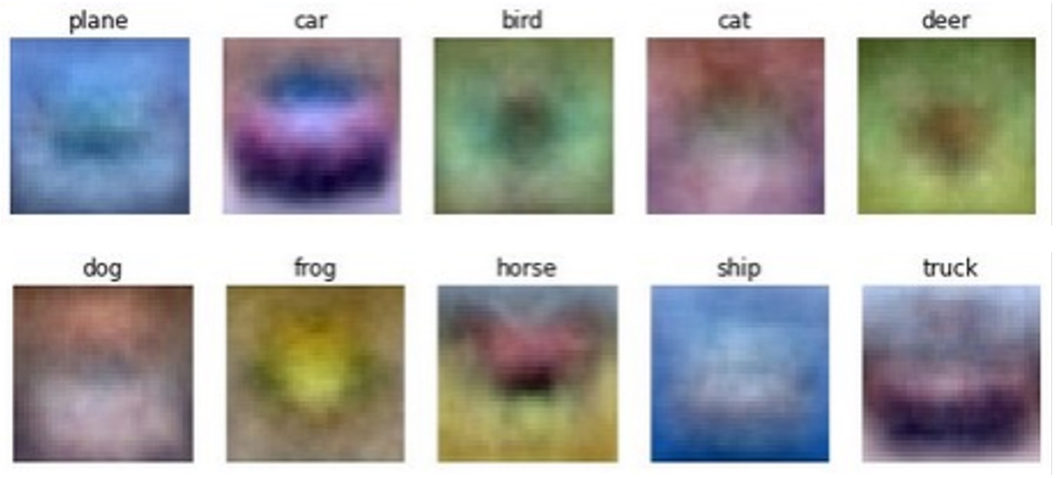
Output of hidden layer: $\boldsymbol{g}(\boldsymbol{W}_1\boldsymbol{x})$

Final output: $\boldsymbol{g}(\boldsymbol{W}_2\boldsymbol{g}(\boldsymbol{W}_1\boldsymbol{x}))$

Image source

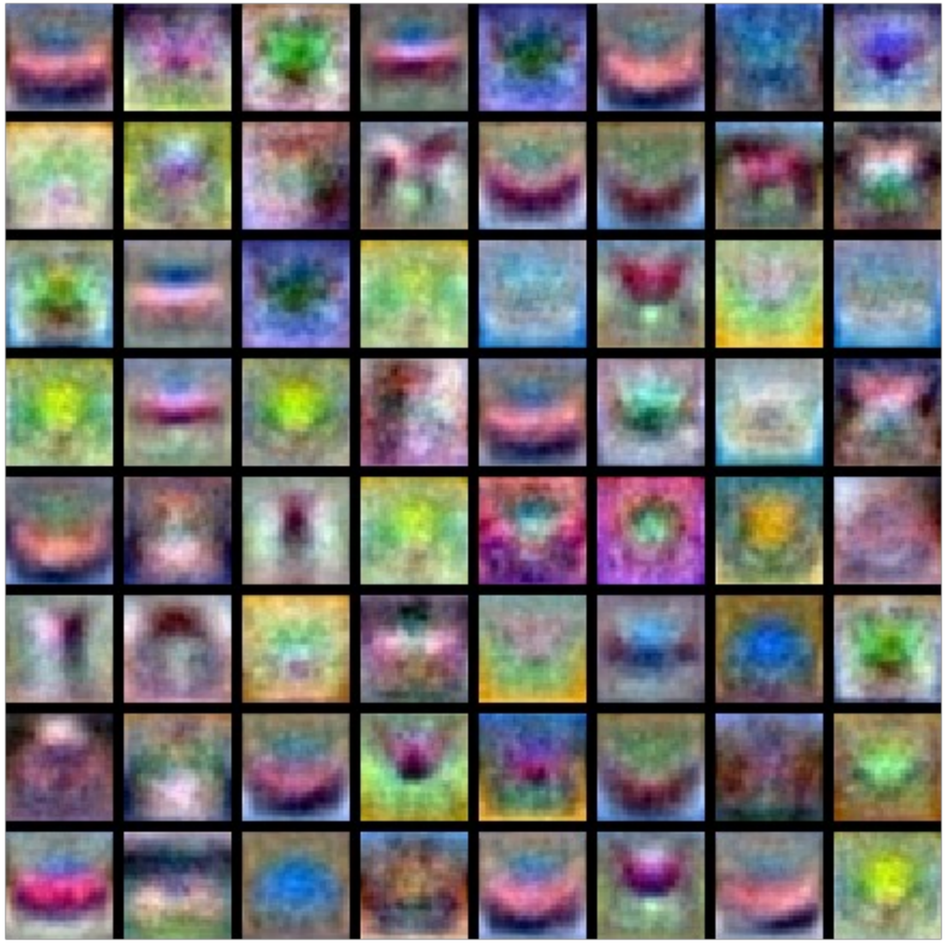
Two-layer networks as combinations of templates

Linear classifier: One template per class



Two-layer networks as combinations of templates

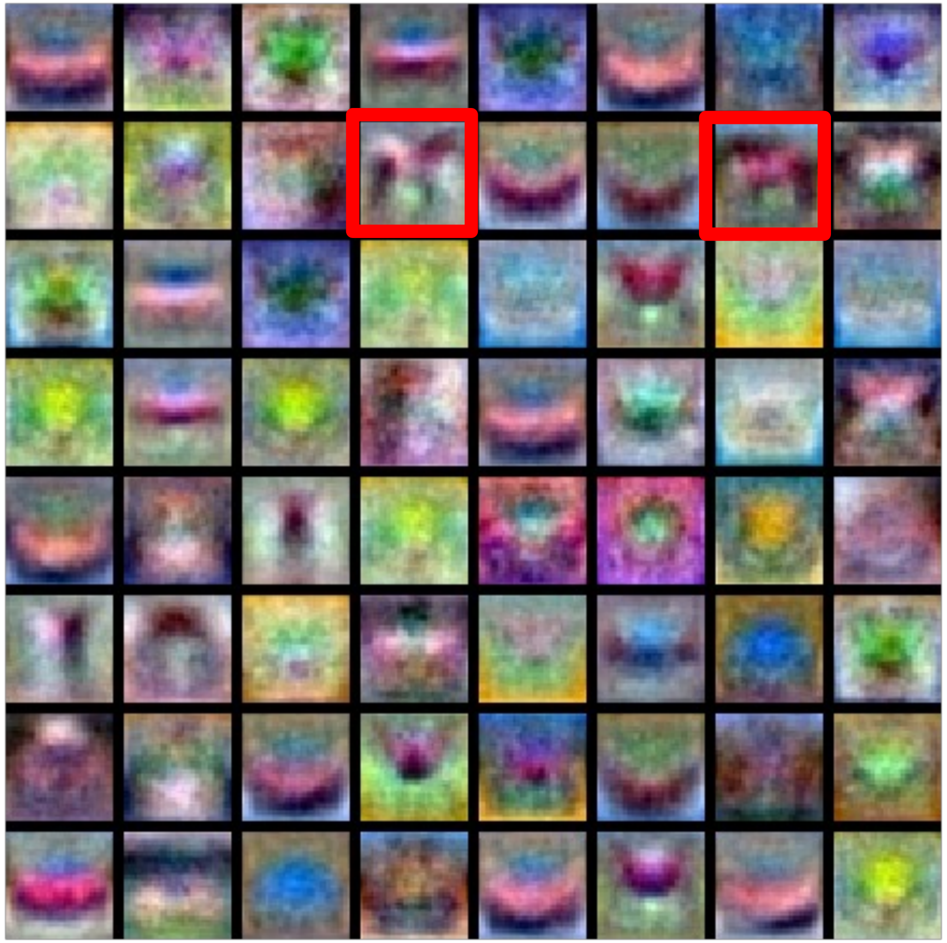
First layer: bank of templates
Second layer: recombines templates



Source: J. Johnson

Two-layer networks as combinations of templates

First layer: bank of templates
Second layer: recombines templates



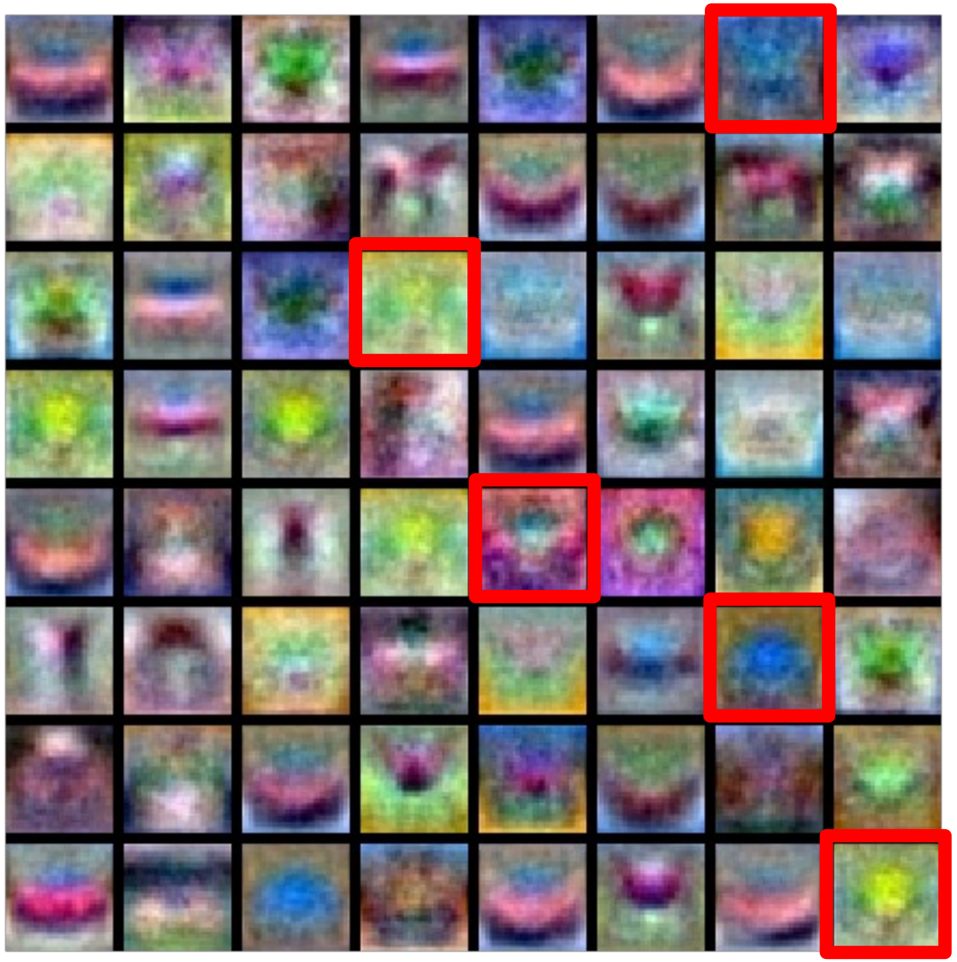
Can use different templates to cover multiple *modes* of a class

Source: J. Johnson

Two-layer networks as combinations of templates

First layer: bank of templates

Second layer: recombines templates



It’s a “distributed” representation: Most templates are not interpretable

Source: J. Johnson

Expressiveness of two-layer networks

- How complex can we make the decision boundary in a two-layer network?
- The bigger the hidden layer, the more complex the model
- A two-layer network is a *universal function approximator*
  - But the hidden layer may need to be huge

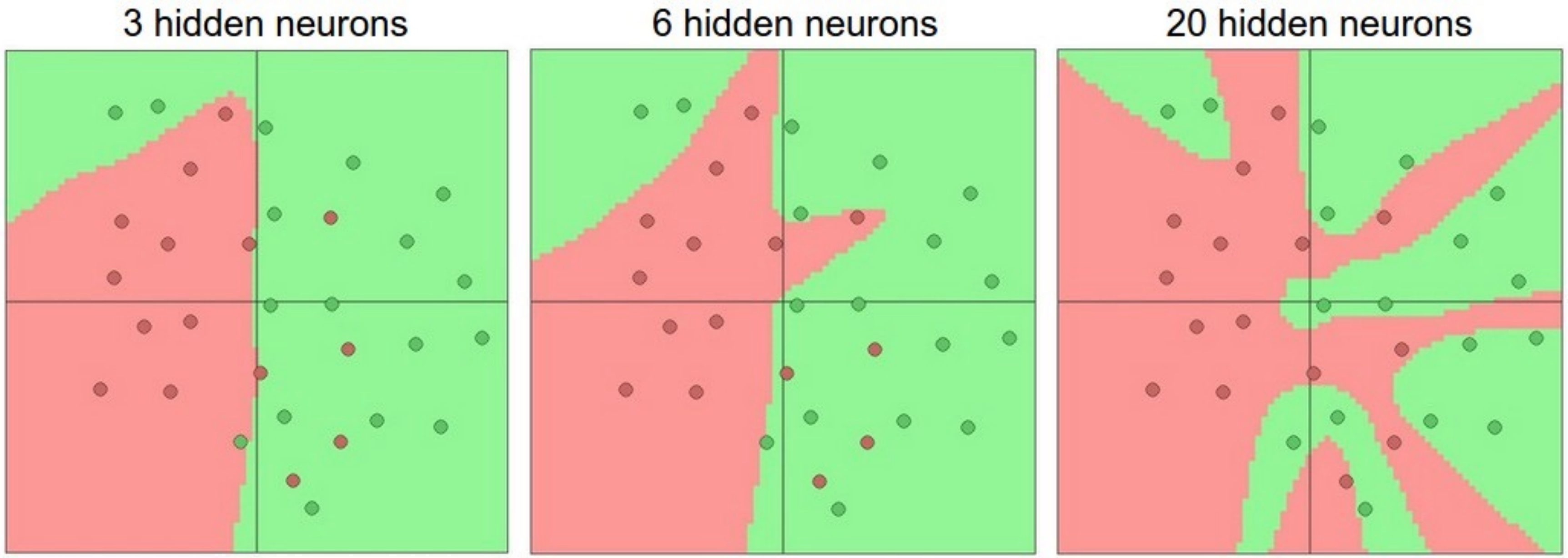
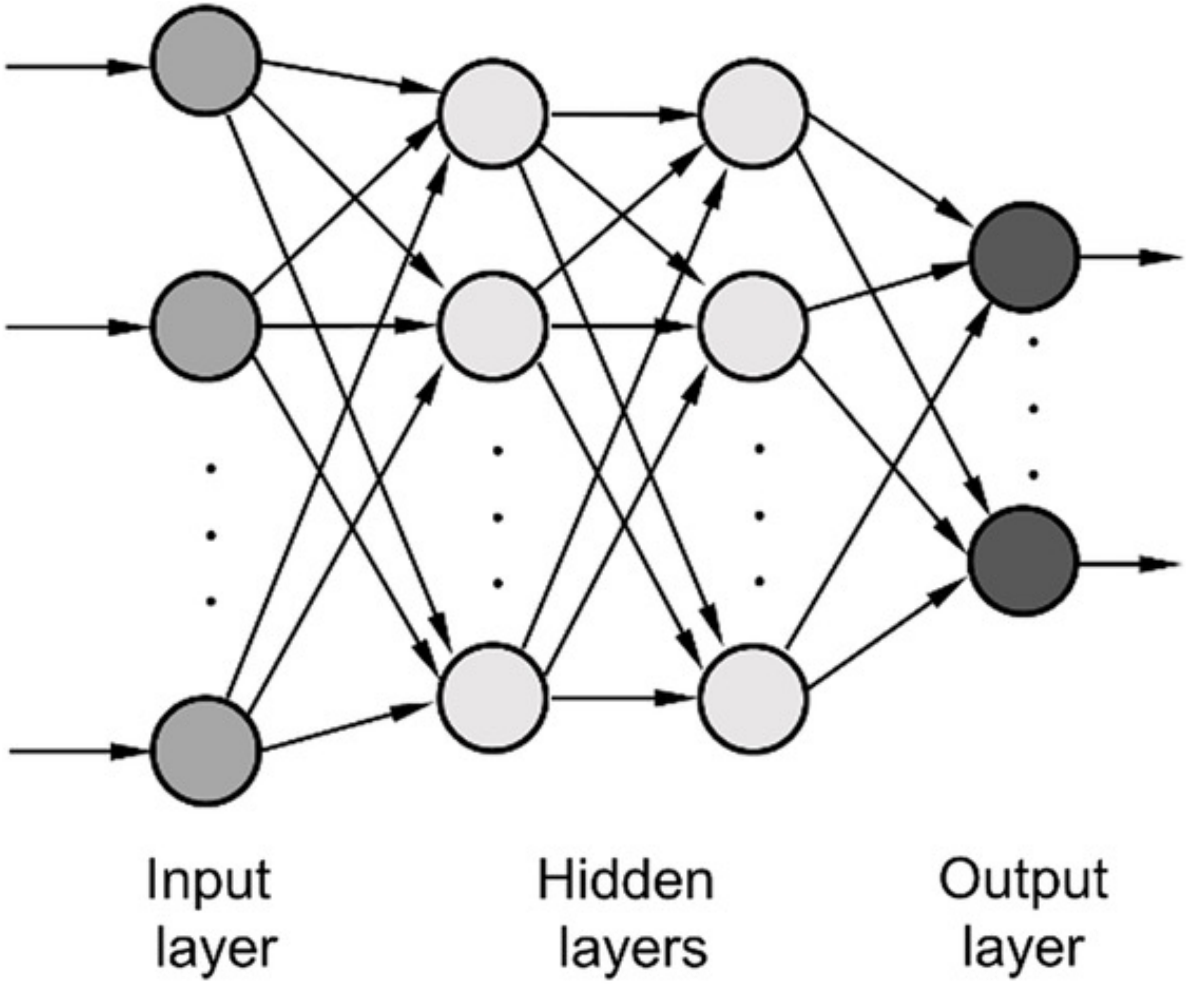


Figure source

Neural networks beyond two layers

Individual dimensions of x

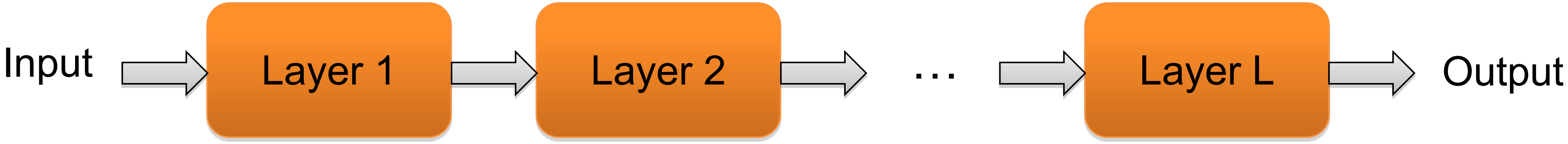


Output:

$$g_L(W_L \dots g_2(W_2\, g_1(W_1 x)) \dots)$$

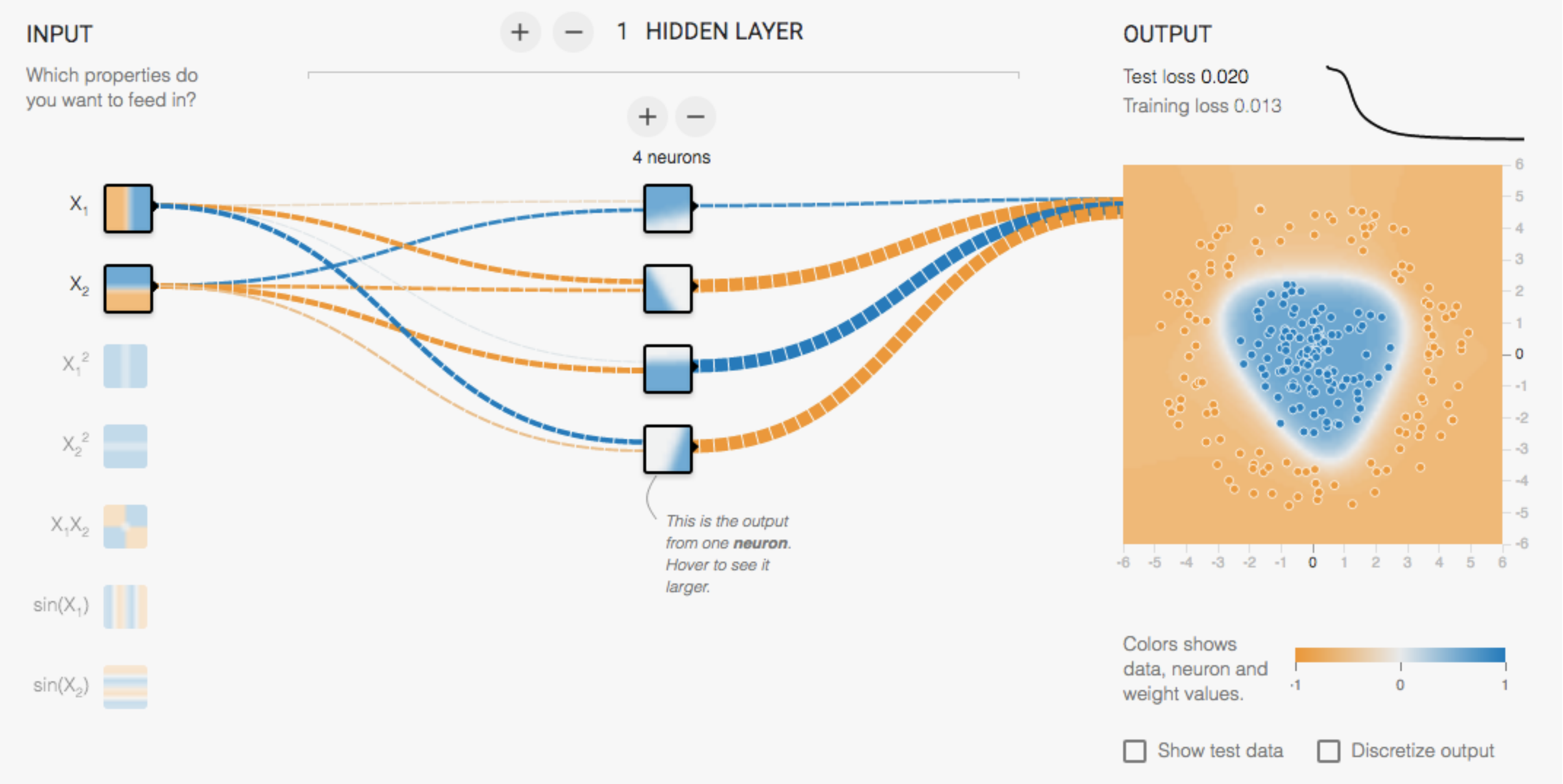
Image source

“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/

Overview

- Feature Design
- Nonlinear classifiers
  - “Shallow” approach: Kernel support vector machines (SVMs)
  - “Deep” approach: Multi-layer neural networks
- Controlling classifier complexity
  - Hyperparameters
  - Bias-variance tradeoff
  - Overfitting and underfitting
  - Hyperparameter search in practice

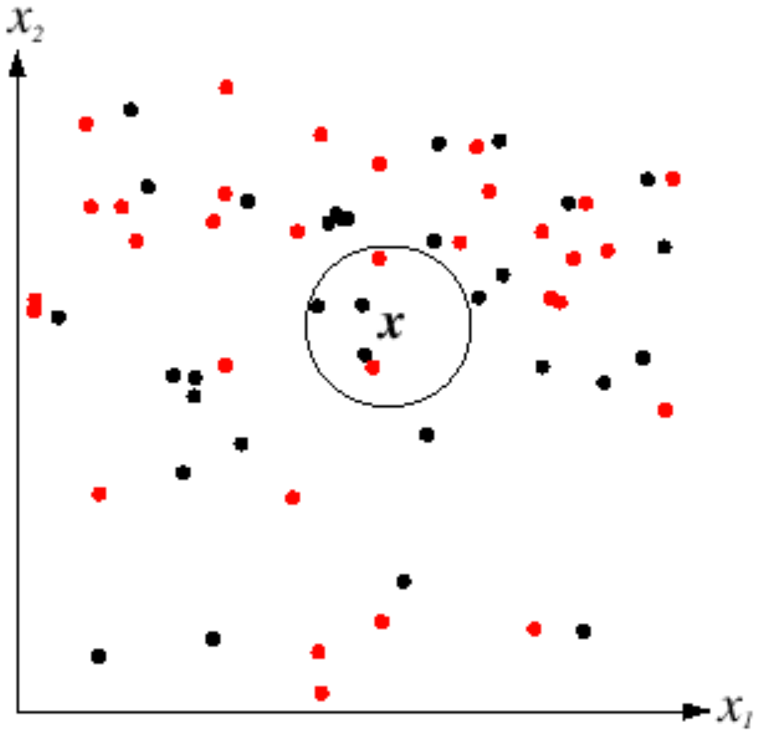
Supervised learning outline revisited

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

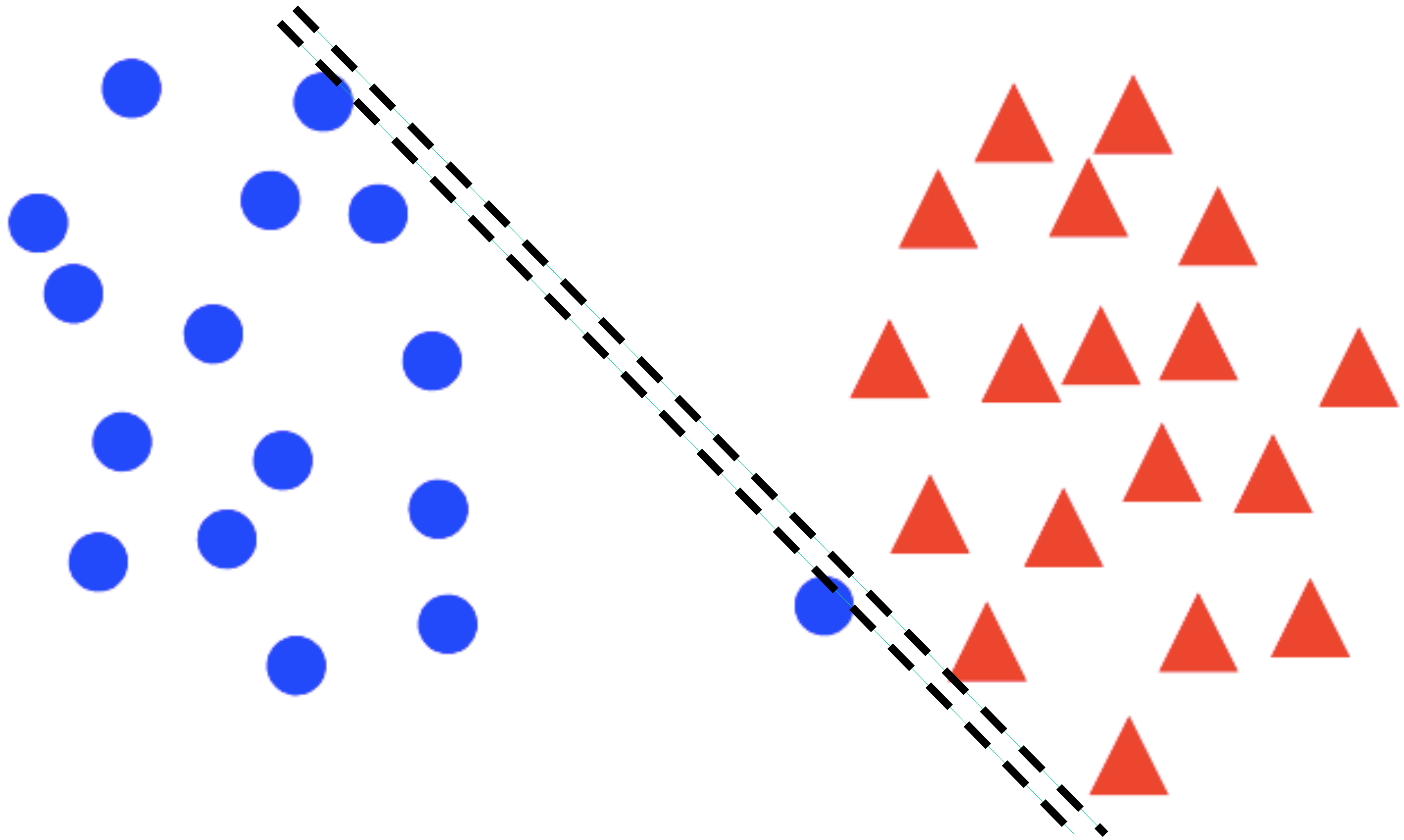
- Regularization constant λ
  - Recall: SVM optimization

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

  - What if λ is too large?
  - What if λ is too small?

Hyperparameters

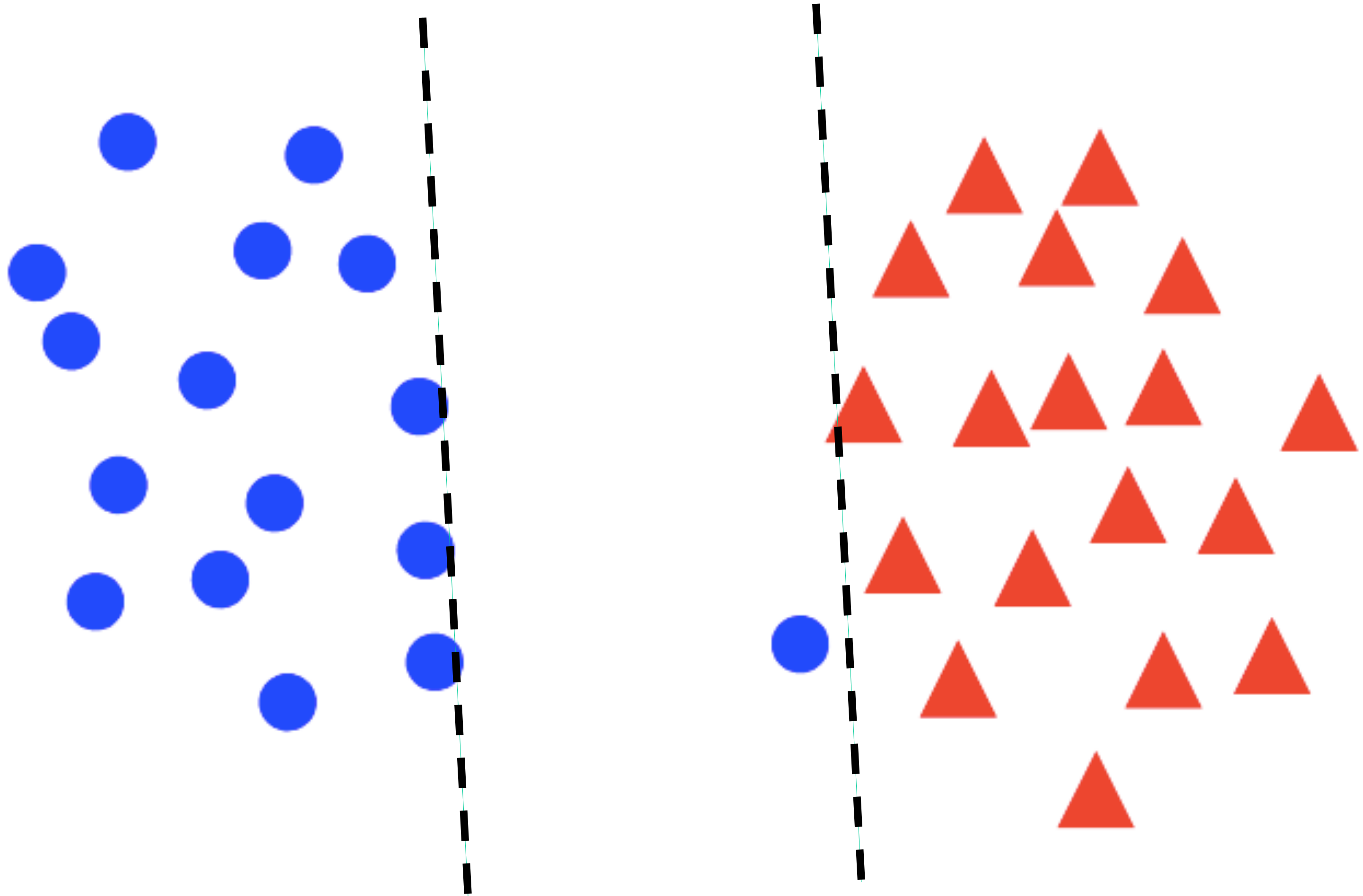
- Regularization constant λ
  - Tradeoff between margin and classification errors



Source

Hyperparameters

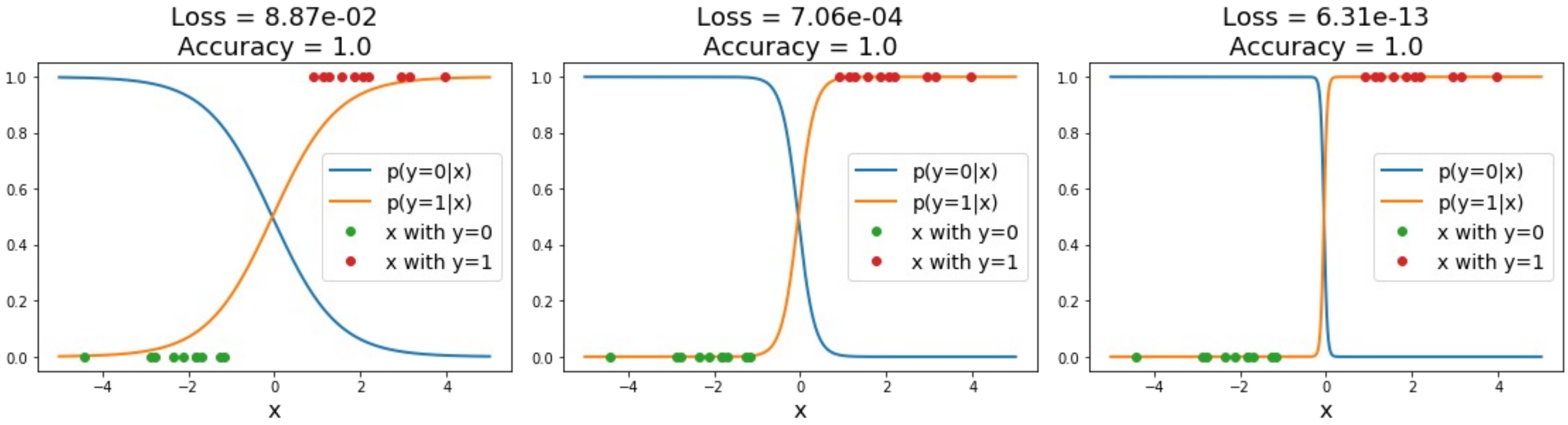
- Regularization constant λ
  - Tradeoff between margin and classification errors



Source

Hyperparameters

- Regularization constant λ
  - Related: preventing the classifier from getting over-confident



Loss = 8.87e-02
Accuracy = 1.0

Loss = 7.06e-04
Accuracy = 1.0

Loss = 6.31e-13
Accuracy = 1.0

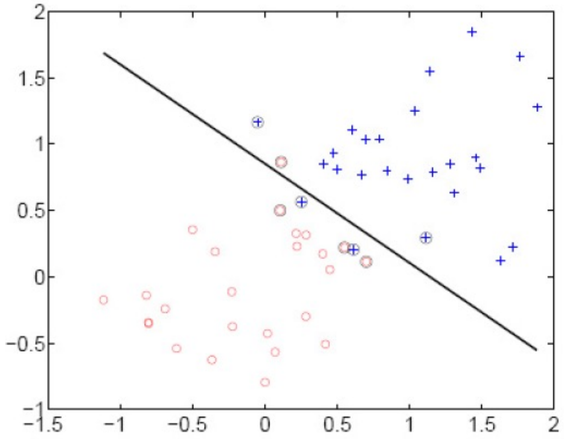
Sigmoid classifier, logistic loss

Source: J. Johnson

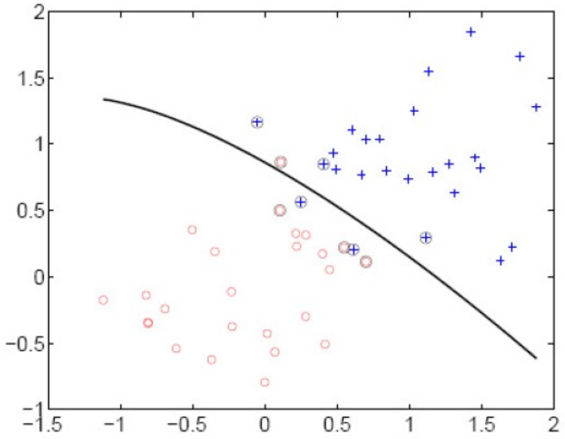
Hyperparameters

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

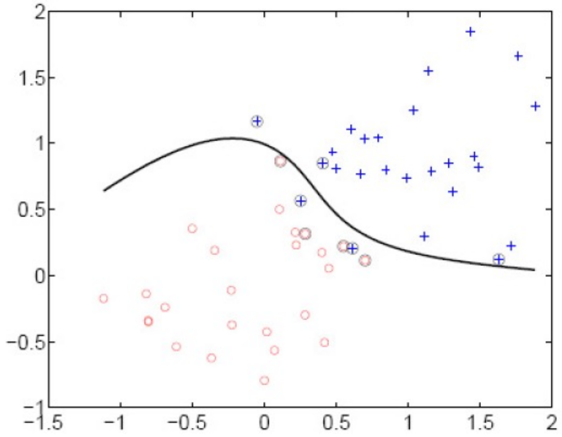
Polynomial kernel: $K(x, x') = (x^T x' + c)^d$



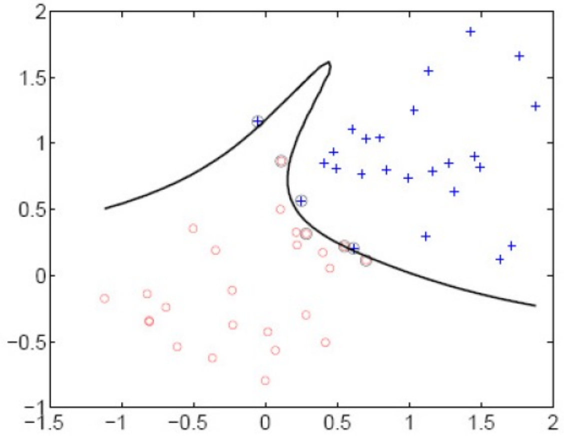
linear



2^{nd} order polynomial



4^{th} order polynomial



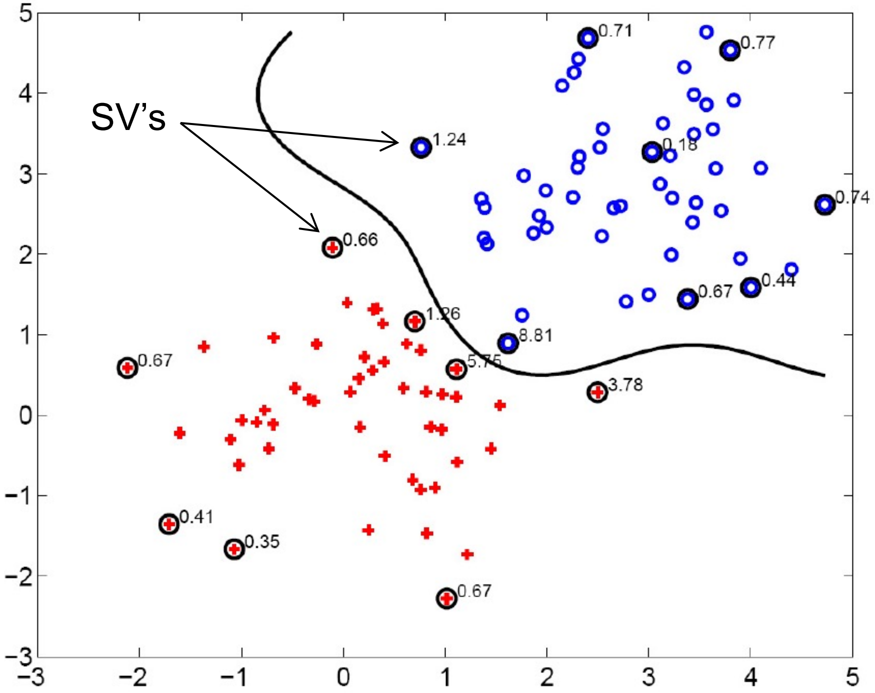
8^{th} order polynomial

Gaussian kernel

- Gaussian kernel with bandwidth σ:

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

- 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

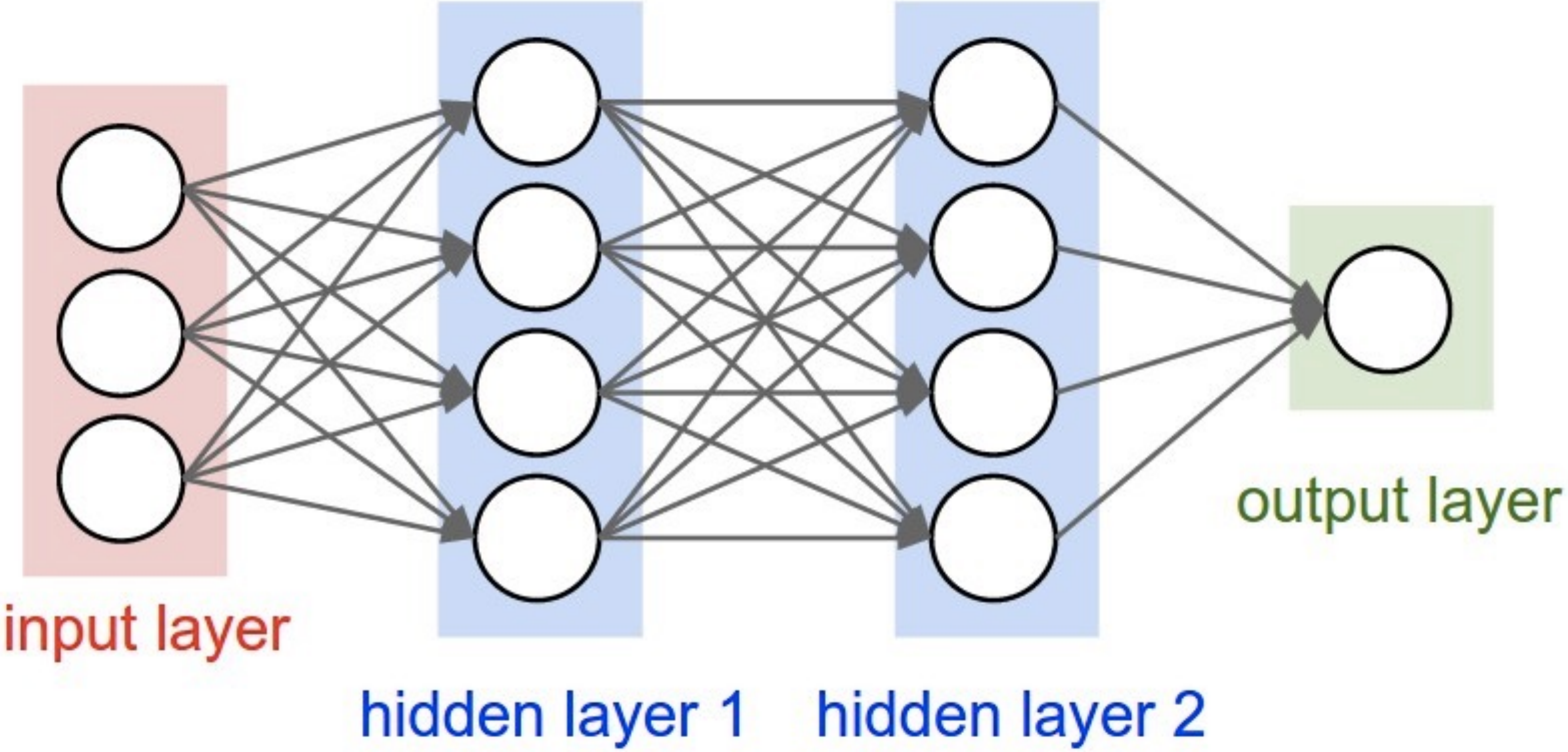
- Gaussian kernel with bandwidth σ:

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

- 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 σ affect the behavior of the predictor?
  - What if σ is close to zero?
  - What if σ is very large?

Hyperparameters in multi-layer networks

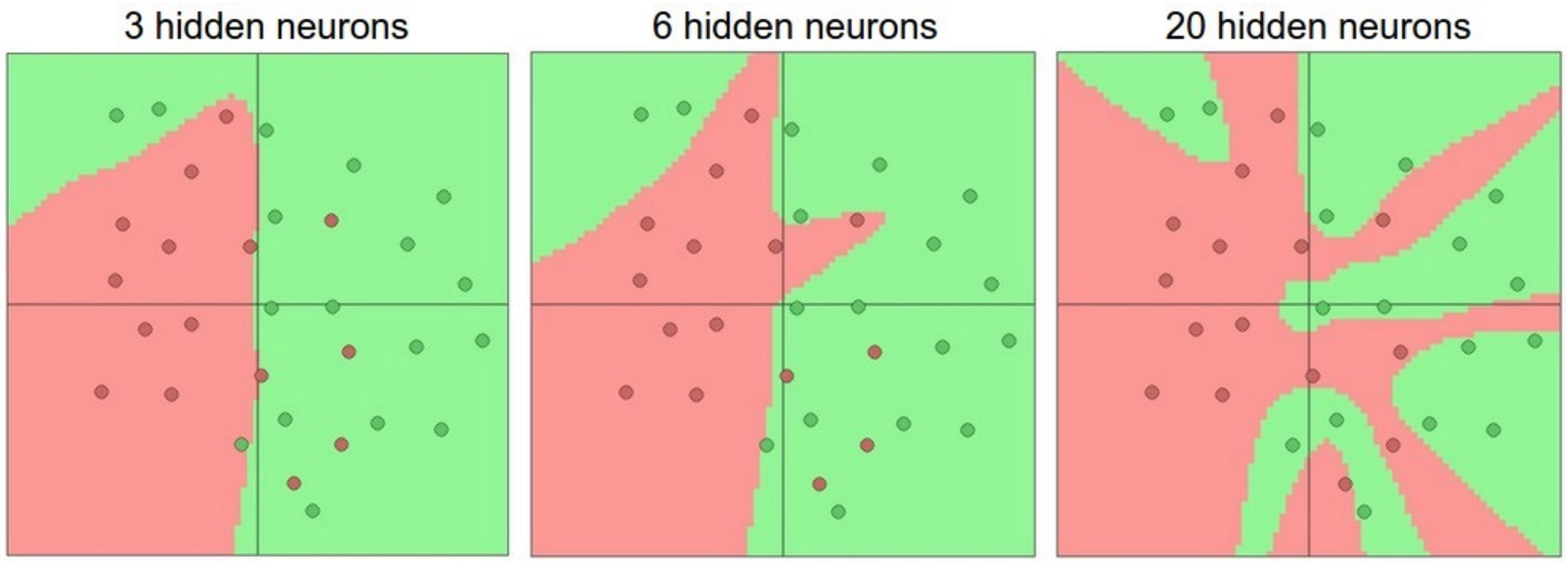
- Number of layers, number of units per layer



Source: Stanford 231n

Hyperparameters in multi-layer networks

- Number of layers, number of units per layer

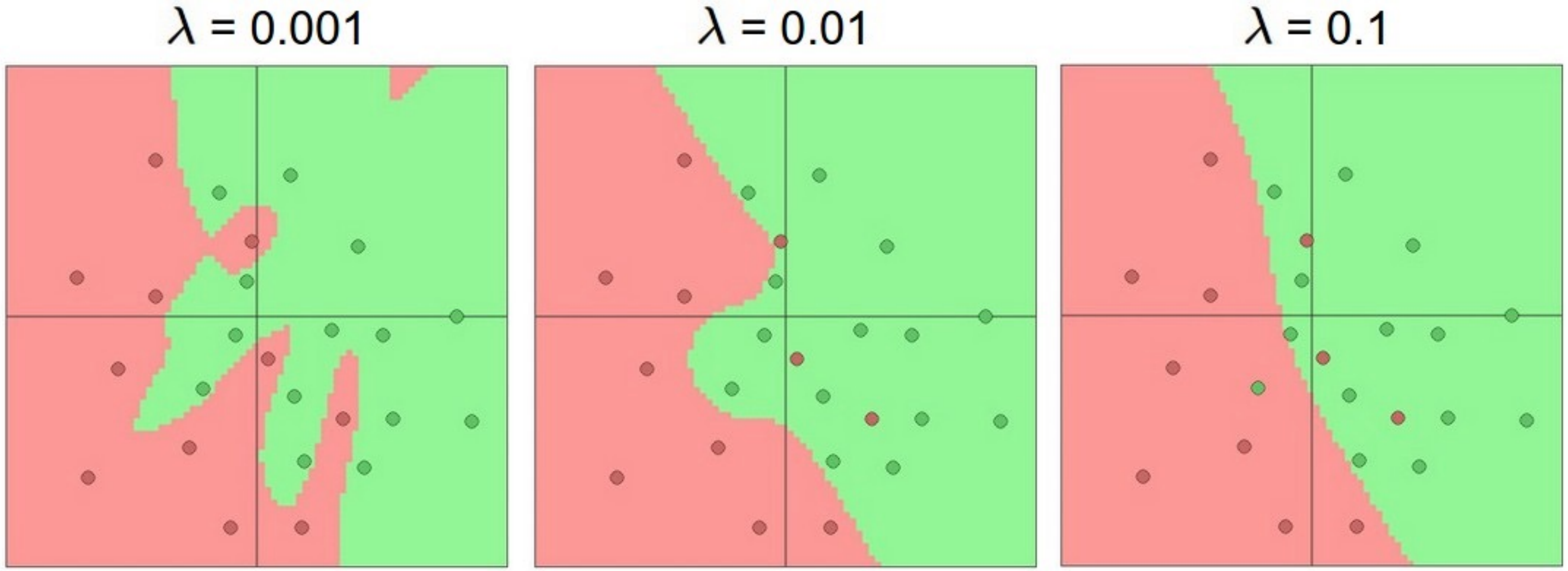


Number of hidden units in a two-layer network

Source: Stanford 231n

Hyperparameters in multi-layer networks

- Number of layers, number of units per layer
- Type of nonlinearity
- Type of loss function
- Regularization constant



Source: Stanford 231n

Hyperparameters in multi-layer networks

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

Summary: Hyperparameters

- 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, type of nonlinearity, type of loss function, regularization constant
  - SGD settings: learning rate schedule, number of epochs, minibatch size, etc.

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

Overview

- Nonlinear classifiers
  - Kernel support vector machines (SVMs)
  - Multi-layer neural networks
- Controlling classifier complexity
  - Hyperparameters
  - Bias-variance tradeoff
  - Overfitting and underfitting
  - Hyperparameter search in practice

Model complexity and generalization

- Generalization (test) error of learning algorithms can be broken down into three components (see notes):
  - **Noise:** unavoidable error
  - **Bias:** error due to simplifying model assumptions
  - **Variance:** error due to randomness of training set

"Simple" model

High bias, low variance

"Intermediate" model

"Complex" model

Low bias, high variance

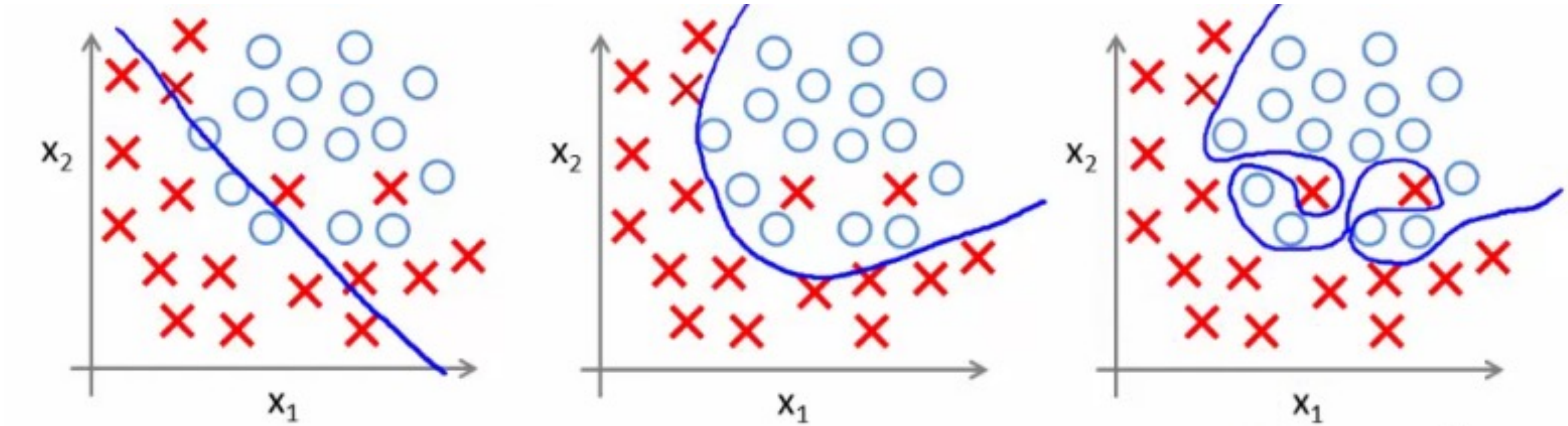
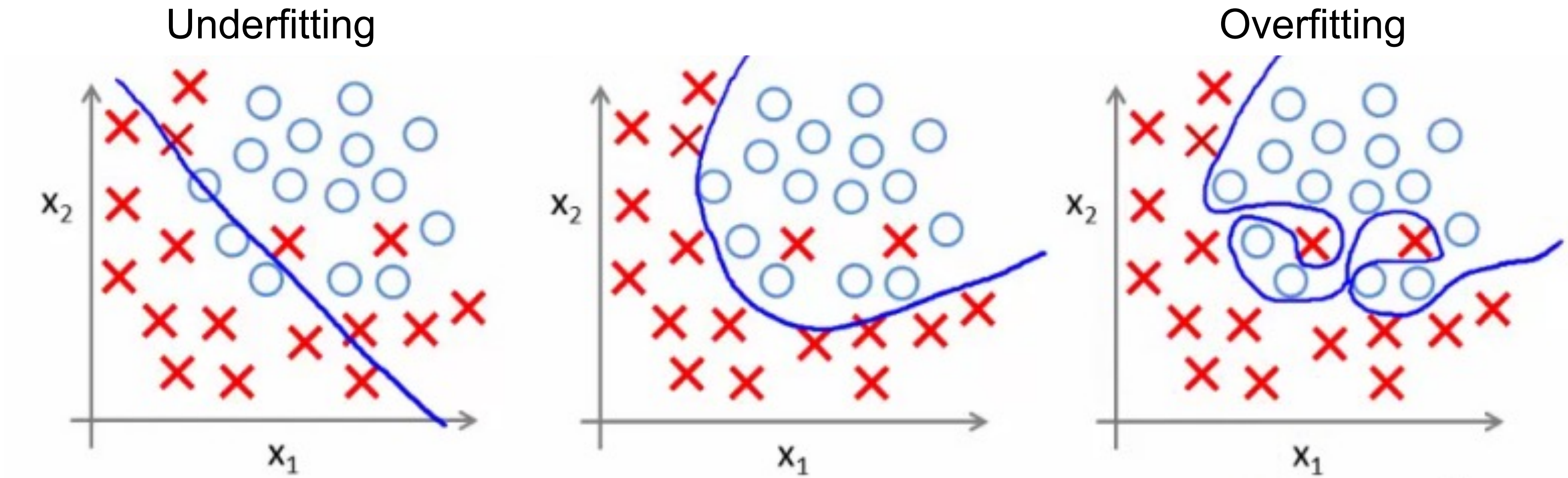


Figure source

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?


Underfitting

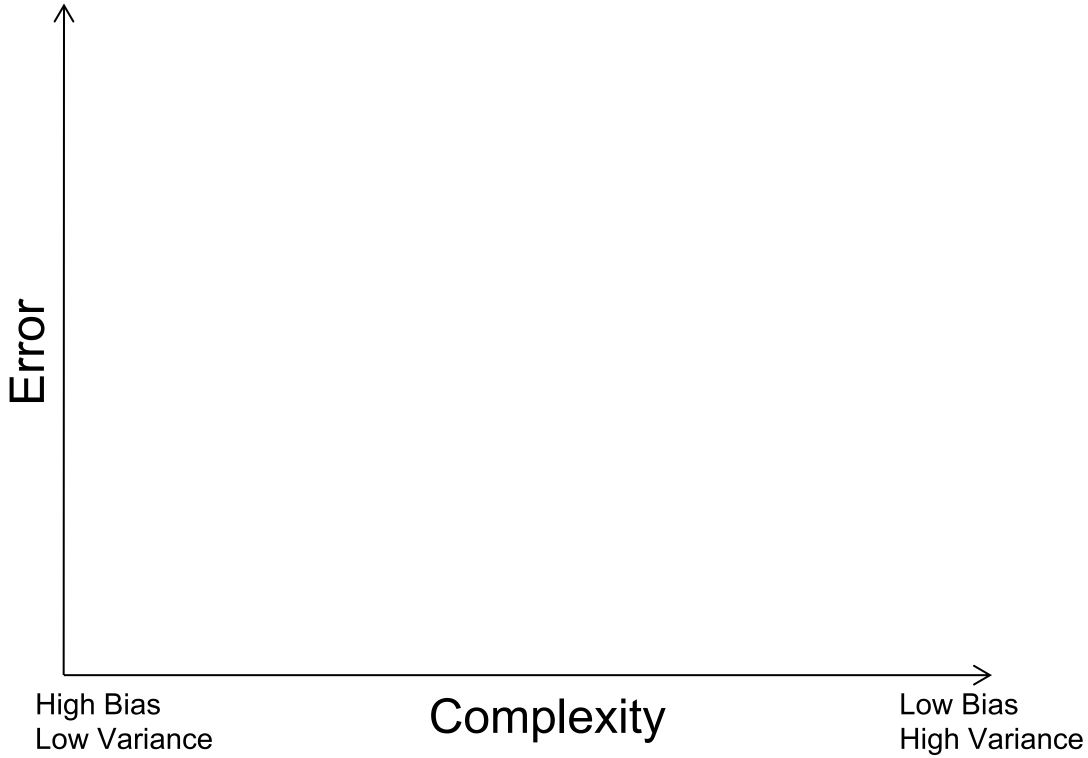
Overfitting

Figure source

Bias-variance tradeoff

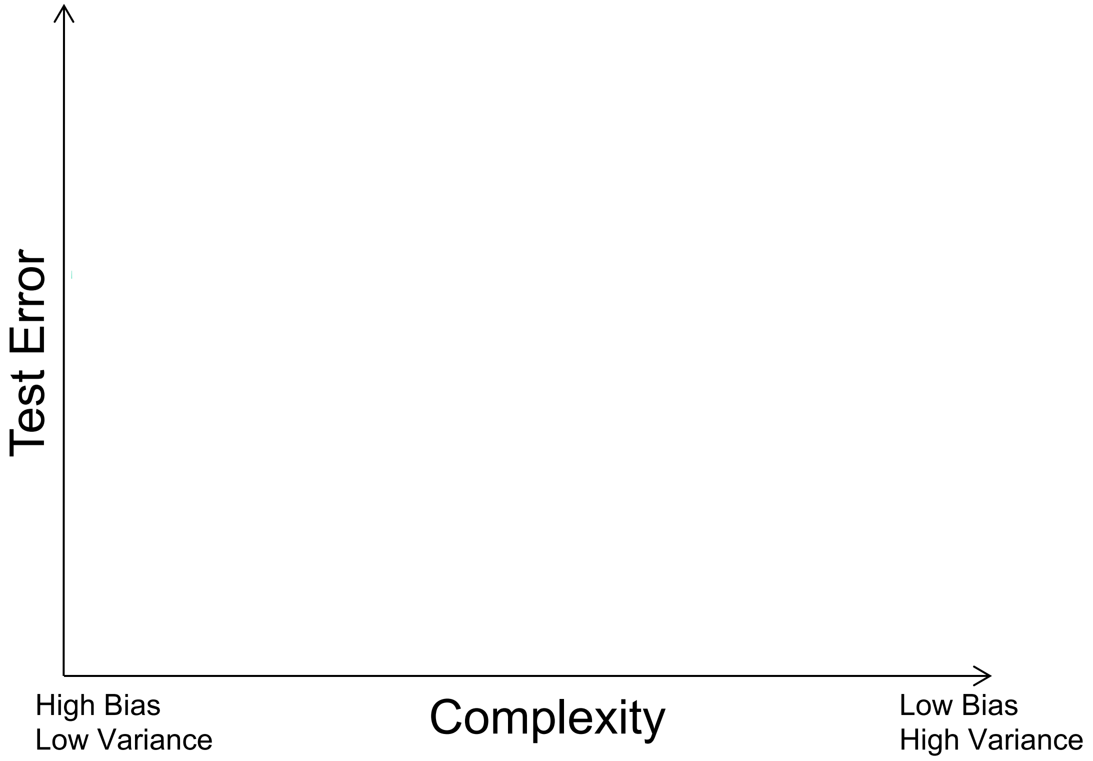
- What if your model **bias** is too high?
    - Your model is **underfitting** – it is incapable of capturing the important characteristics of the training data
- What if your model **variance** is too high?
    - Your model is **overfitting** – it is fitting noise and unimportant characteristics of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

Behavior of training and test error



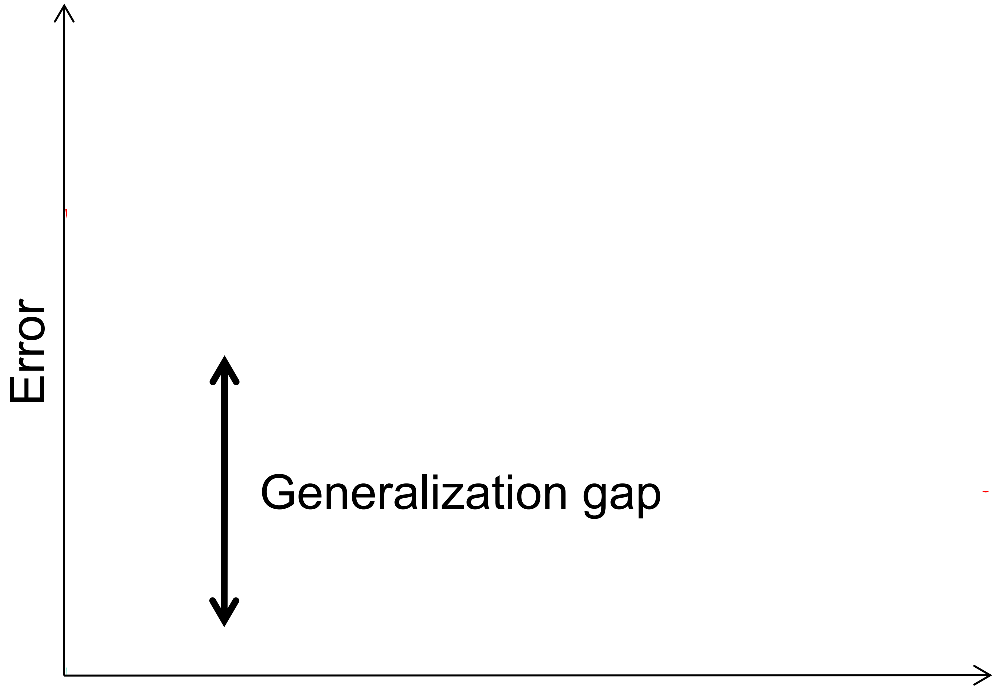
Source: D. Hoiem

Dependence on training set size



Source: D. Hoiem

Dependence on training set size



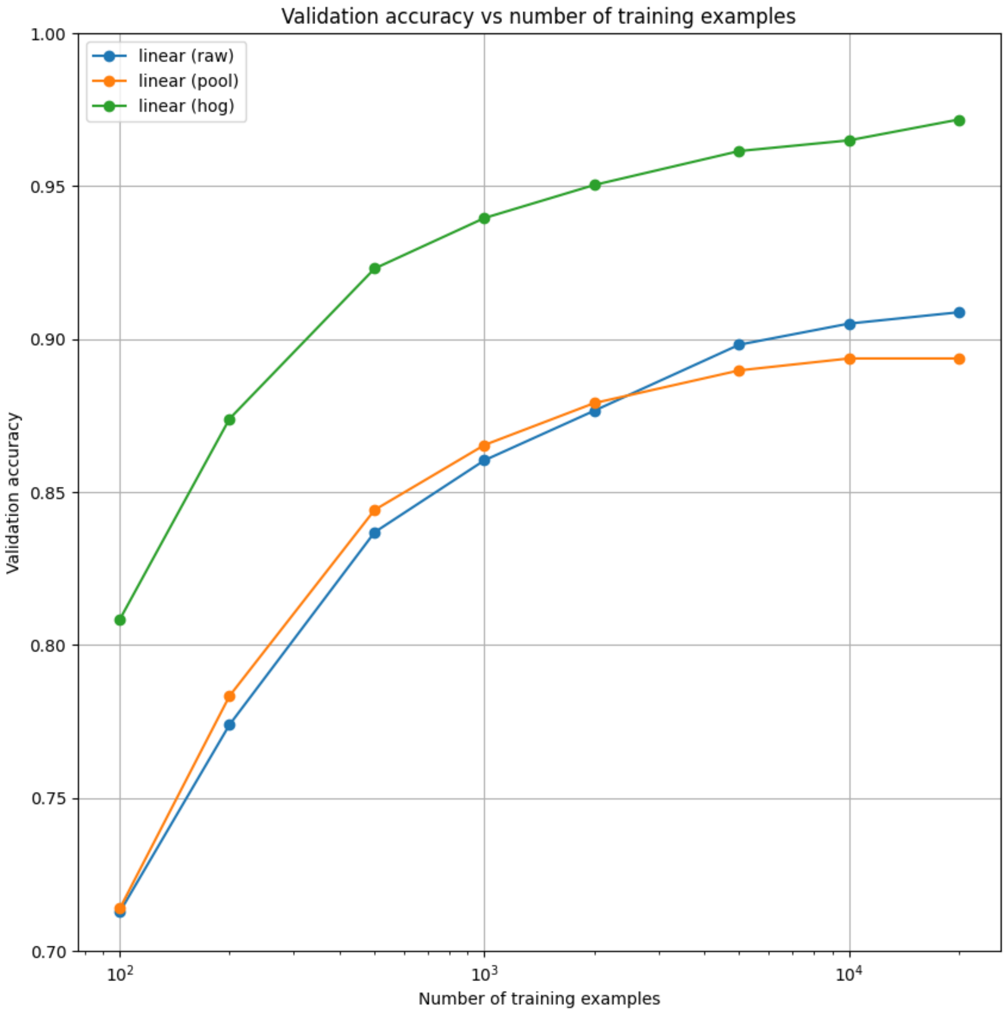
Number of training examples

(fixed model)

Source: D. Hoiem

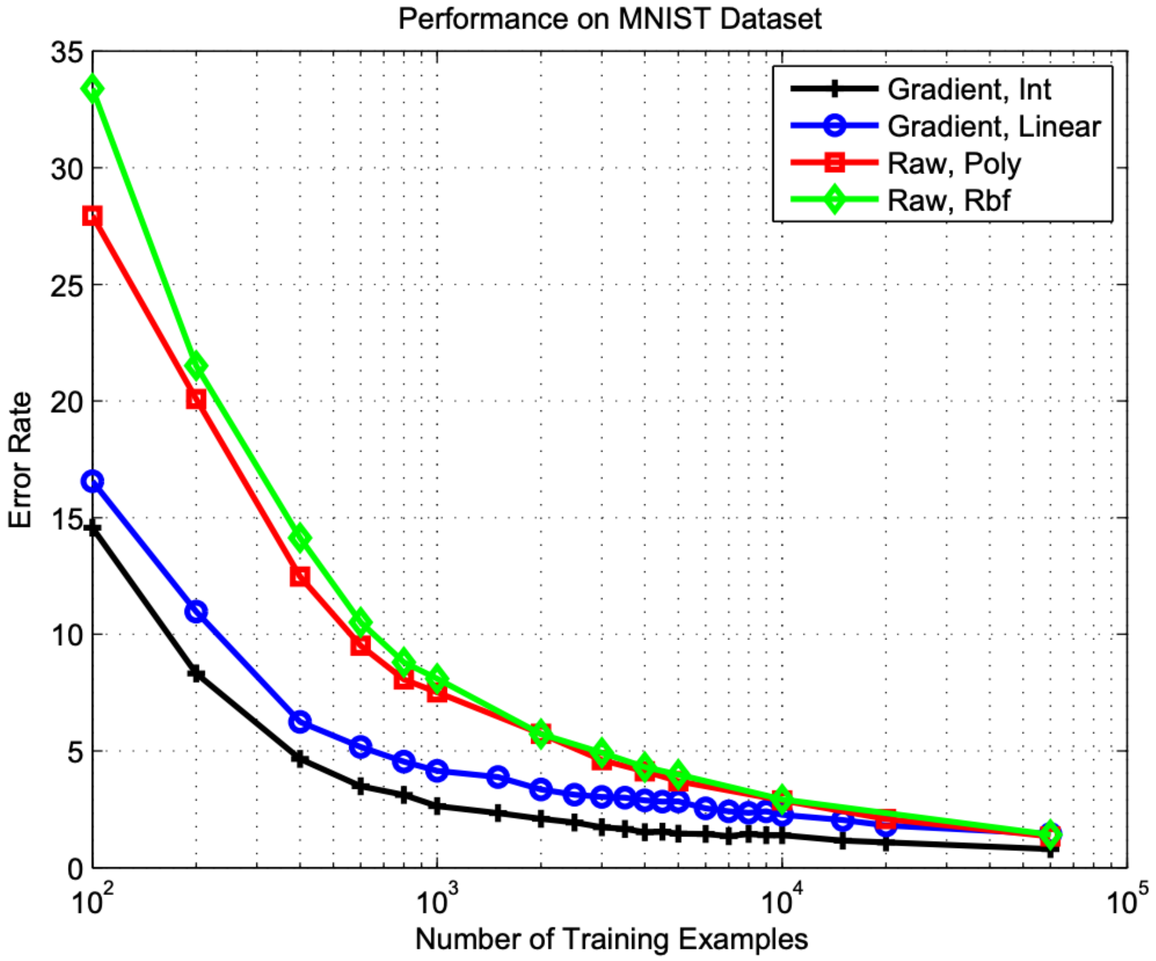
Dependence on training set size

- Digit classification case study



Dependence on training set size

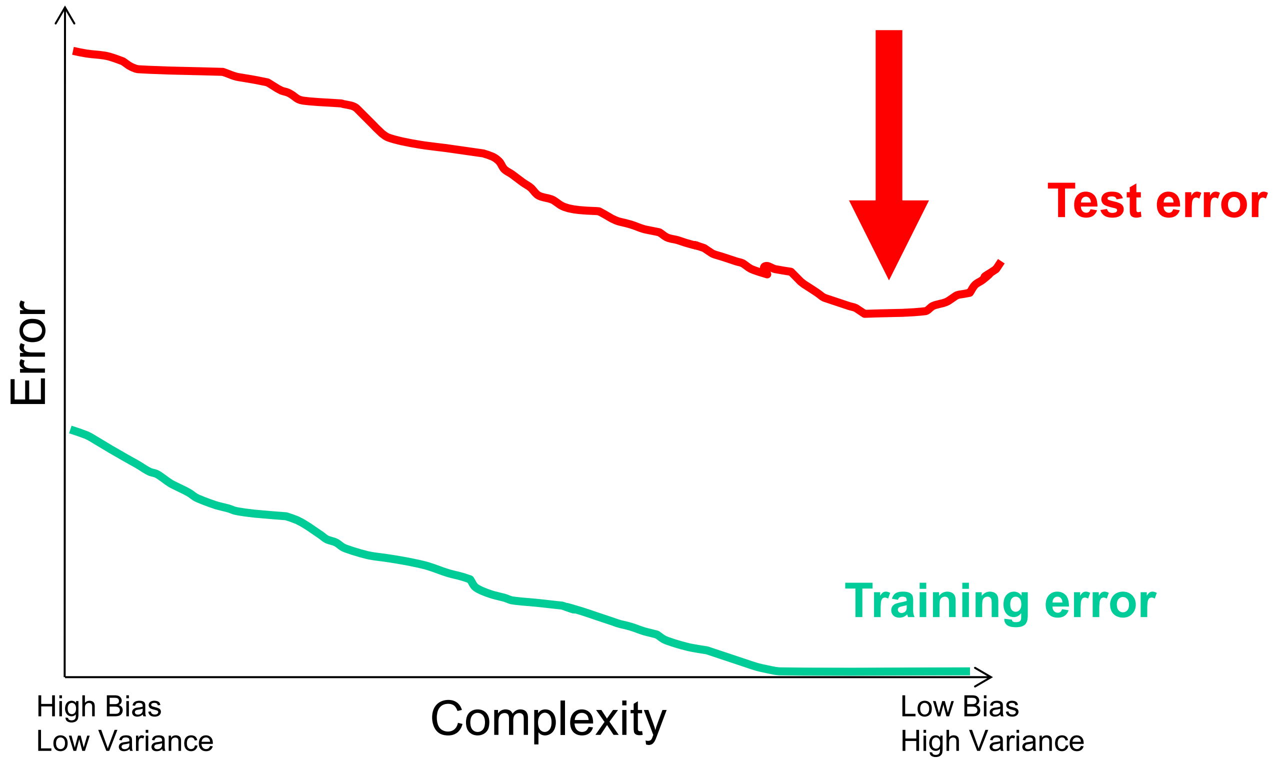
- Digit classification case study



Maji and Malik. 2009 **Fast and Accurate Digit Classification**

Looking at training and test error

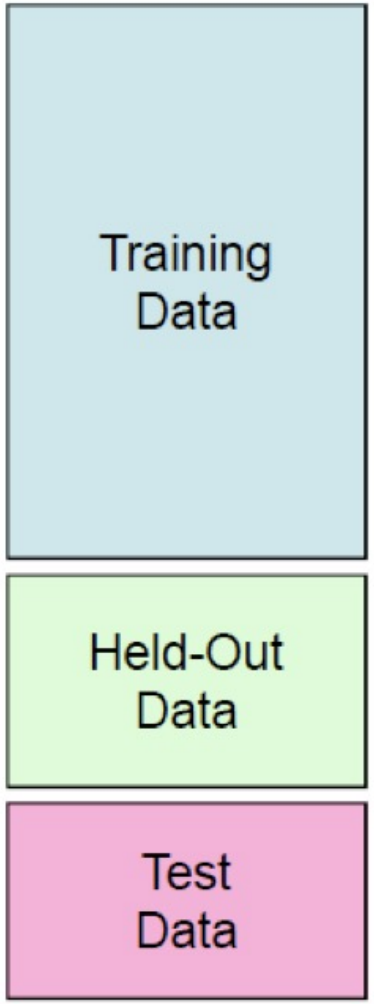
- In most practical situations, you are faced with a fixed dataset and have to find the hyperparameter settings that give you the best generalization performance



Source: D. Hoiem

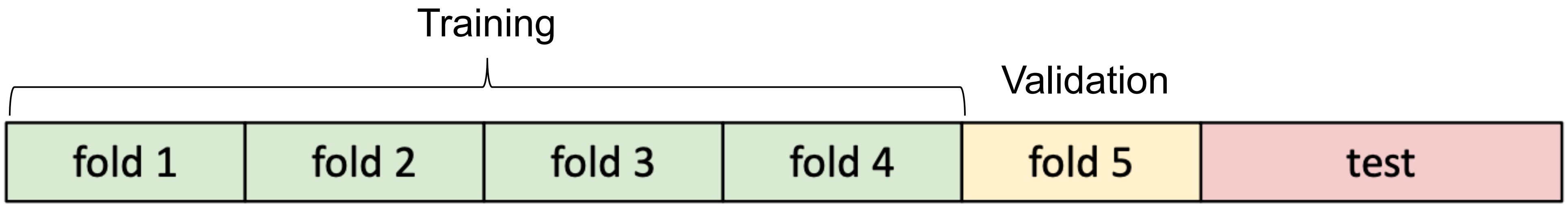
Hyperparameter search in practice

- 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 entire training set into K groups
  - In each run (or fold), select one of the groups as the validation set and train on the other K-1 groups. At the end, average the accuracies across the K folds
  - Typically not used for deep learning due to computational expense



Training

Validation

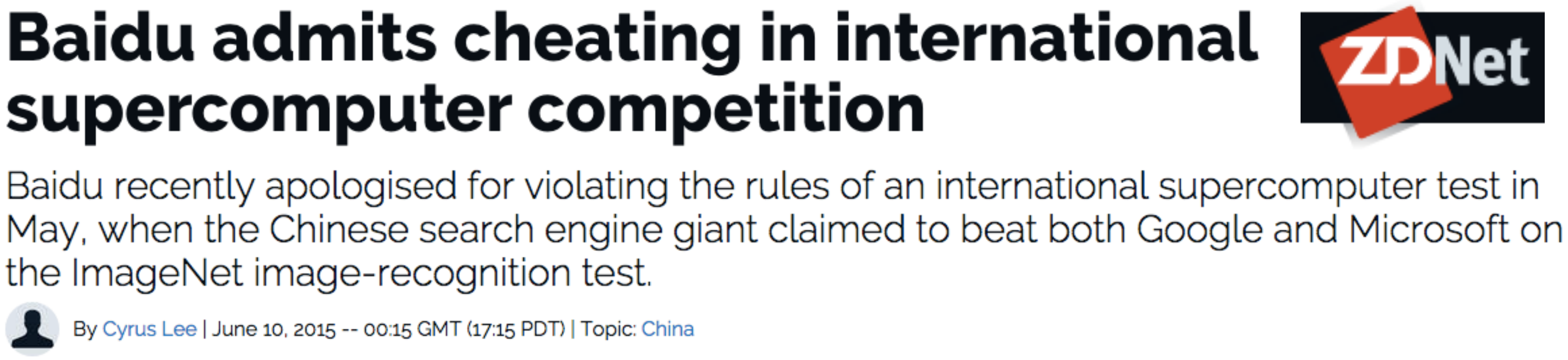
fold 1 | fold 2 | fold 3 | fold 4 | fold 5 | test

Source: J. Johnson

What’s the big deal?

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

What’s the big deal?

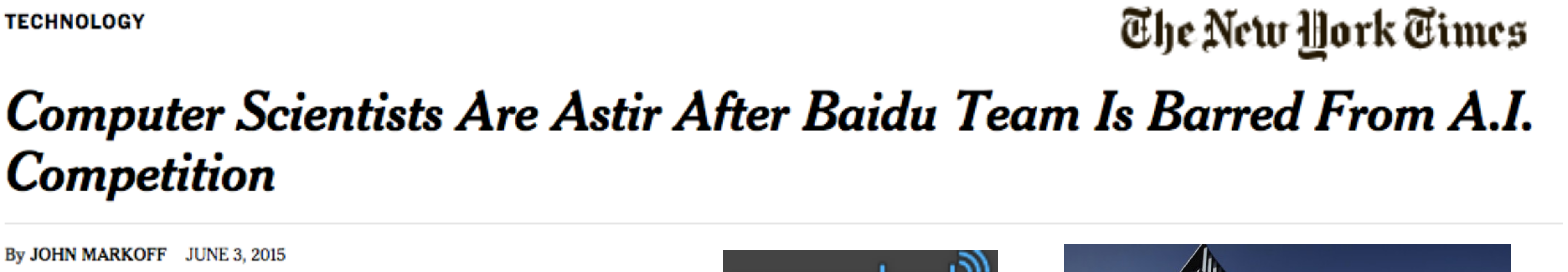


Baidu admits cheating in international supercomputer competition

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



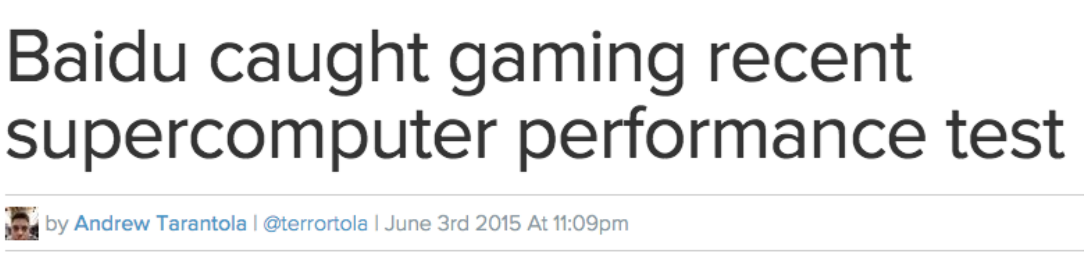
TECHNOLOGY

The New York Times

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

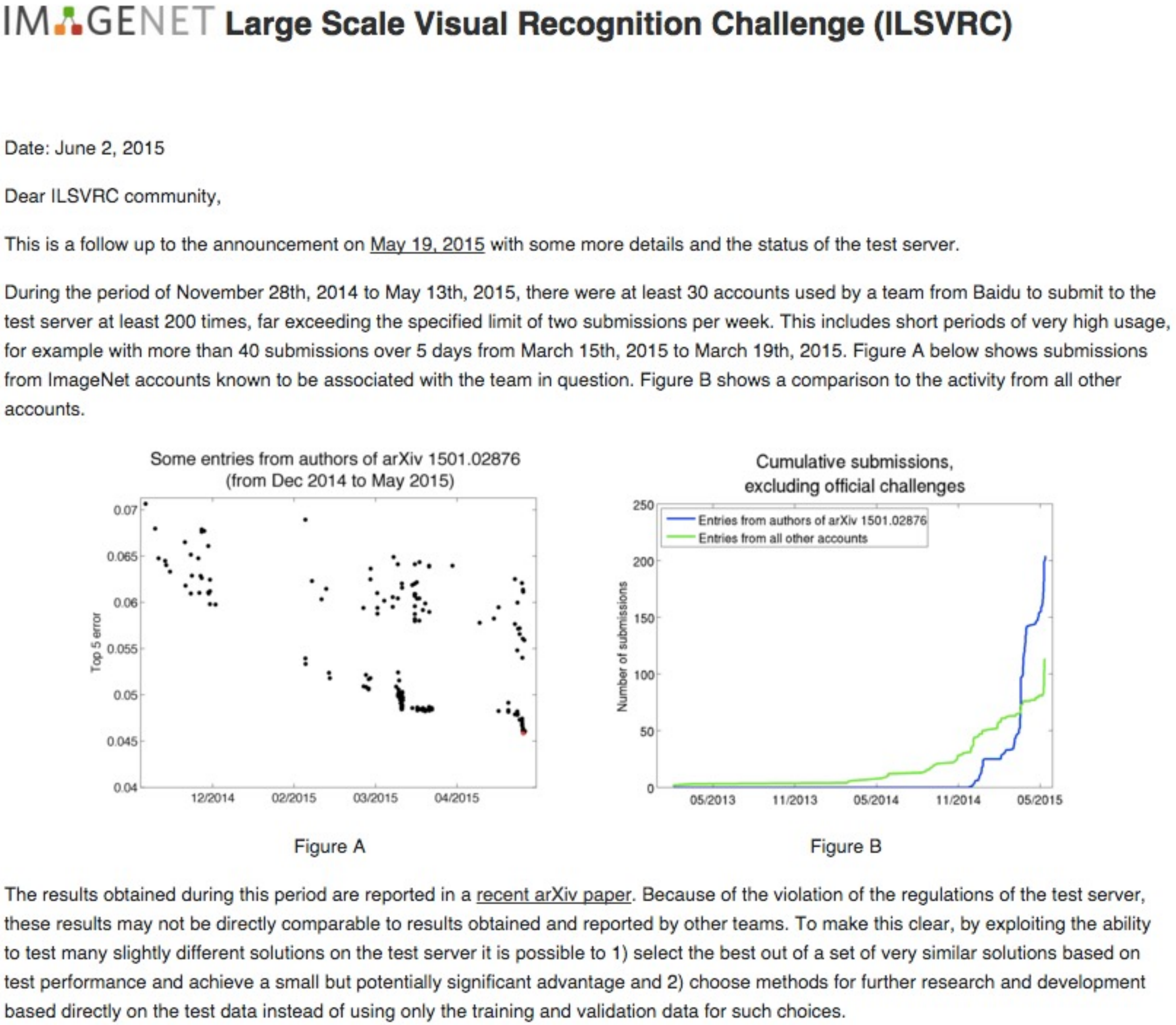
By JOHN MARKOFF JUNE 3, 2015

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.

Some entries from authors of arXiv 1501.02876
(from Dec 2014 to May 2015)

Figure A

Cumulative submissions,
excluding official challenges

Entries from authors of arXiv 1501.02876
Entries from all other accounts
Number of submissions
250
200
150
100
50
0
05/2013
11/2013
05/2014
11/2014
05/2015

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