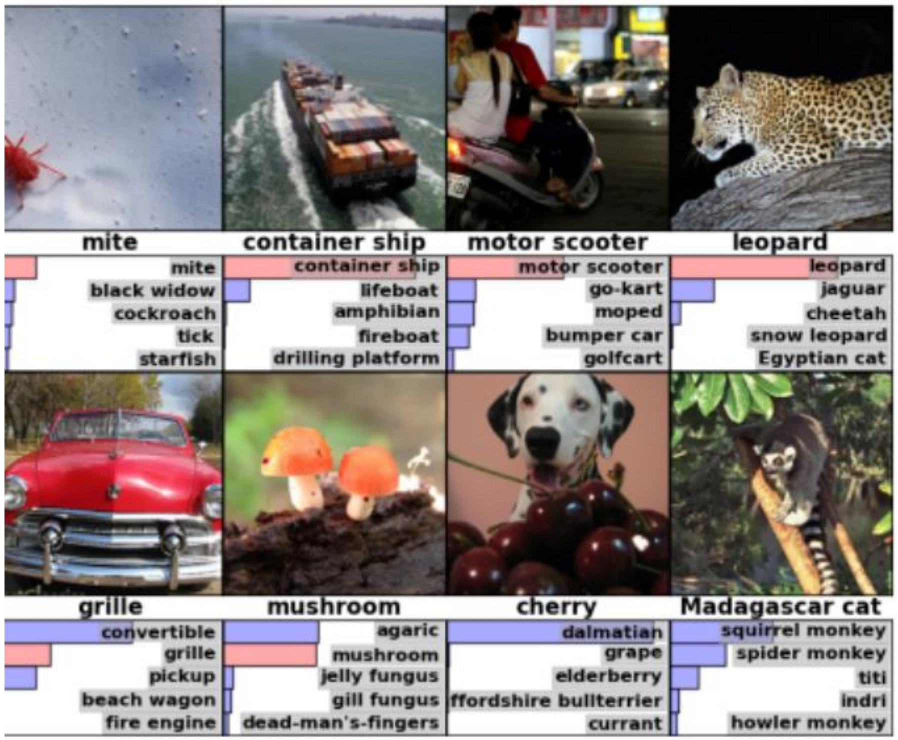
COMP 424 - Artificial Intelligence
Lecture 19: Supervised Learning

Instructor: Jackie CK Cheung (jcheung@cs.mcgill.ca)

Readings: R&N Ch 18

Machine learning at work

- ImageNet dataset: predict object type among 10,000 categories.



(Krizhevsky et al., 2012)

- Error rates now down a few percent, for top-5 results.

Object Detection in Images

- Locate and detect objects in images

E.g.

ILSVRC2014_train_00040375

- accordion
- sunglasses
- microphone
- guitar
- person
- cello
- violin

- LSVRC 2014 Challege: http://image-net.org/challenges/LSVRC/2015/ui/det.html
- Performance on classification and localization in 2017: <3%

Applications of machine learning

- Spam filtering
- Fraud detection
- Weather prediction
- Customer segmentation
- Categorization of news articles by topic
- …

Many successful approaches do not use probabilistic (Bayesian) models.

General learning problem

- Given a set of **labelled examples** $<x_1, x_2, x_3, \ldots, x_n, y>$
  where x_j are **input variables** and y is the **desired output**

- We want to learn a function $h : X_1 \times X_2 \times \ldots \times X_n \rightarrow Y$
  which maps the input variables onto the output domain

- How does that differ from learning parameters for Bayes nets?

Supervised learning - Classification

Goal: Learning a function for a **categorical** output.

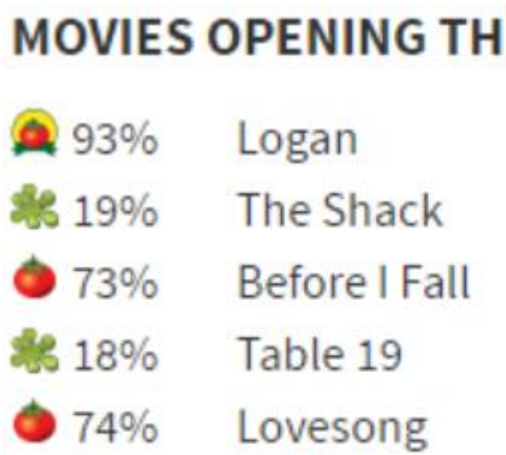
E.g.: Spam filtering. The output ("Spam?") is binary.

	Sender in address book?	Header keyword	**Word 1**	**Word 2**	**…**	**Spam?**
x1	Yes	Schedule	Hi	Profesor	…	No
x2	Yes	meeting	Jackie	I	…	No
x3	No	urgent	Unsecured	Business	…	Yes
x4	No	offer	Hello	I	…	Yes
x5	No	cash	We'll	Help	…	Yes
x6	No	comp-424	Dear	Professor	…	No
…						

Supervised learning - Regression

Sometimes the output value is **continuous**, rather than discrete.

E.g.: Predict the sentiment score associated with a movie, predict stock prices



Terminology

- Inputs called **input variables** or **features** or **attributes**.
- Predictions called **output variables** or **targets**.
- A **data set** consists of **training examples** or **instances**.

e.g., let's identify these components in the table:

	Sender in address book?	Header keyword	**Word 1**	**Word 2**	**…**	**Spam?**
x1	Yes	Schedule	Hi	Profesor	…	No
x2	Yes	meeting	Jackie	I	…	No
x3	No	urgent	Unsecured	Business	…	Yes
x4	No	offer	Hello	I	…	Yes
x5	No	cash	We'll	Help	…	Yes
x6	No	comp-424	Dear	Professor	…	No
…						

Notation

- A training example i has the form $<x_{i,1}, x_{i,2}, \ldots x_{i,n}, y_i>$, where n is the number of attributes.
- Notation $\boldsymbol{x}_i$ denotes the column vector with elements $x_{i,1}, \ldots, x_{i,n}$.
- The training set consists of m training examples.
- Let $X = X_1 \times X_2 \times \ldots \times X_n$ denote the space of input values.
- Let Y denote the space of output values.

Supervised learning problem

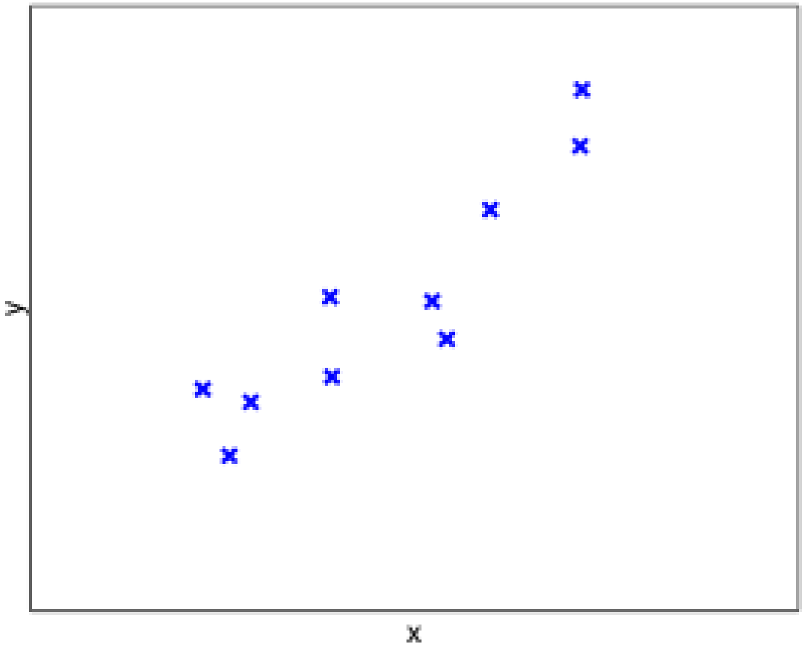
- Given a dataset $D = X \times Y$, find a function: $h : X \rightarrow Y$ such that $h(\mathbf{x})$ is a good predictor for the value of y.

- Formally, h is called the **hypothesis**.

- Output Y can have many types:
  - If $Y = \Re$, this problem is called **regression**.
  - If Y is a finite discrete set, the problem is called **classification**.
  - If Y has 2 elements, the problem is called **binary classification**.

Supervised Learning Example

- What **hypothesis class** should we pick?



x	y
0.86	2.49
0.09	0.83
-0.85	-0.25
0.87	3.10
-0.44	0.87
-0.43	0.02
-1.1	-0.12
0.40	1.81
-0.96	-0.83
0.17	0.43

Linear hypothesis

- Suppose Y is a **linear function** of $\boldsymbol{X}$:

$$f_{\mathbf{W}}(\boldsymbol{X}) = w_0 + w_1 x_1 + \ldots + w_m x_m$$
$$= w_0 + \textstyle\sum_{j=1:m} w_j x_j$$

  - The w_j are called **parameters** or **weights**
  - m = the dimension of observation space, i.e. number of features.

- To simplify notation, we add an attribute $x_0=1$ to the m other attributes (also called **bias term** or **intercept**).

How should we pick the *weights*?

Least-squares solution method

- The linear regression problem: $f_{\mathbf{w}}(X) = w_0 + \sum_{j=1:m} w_j x_j$
- **Goal**: Find the **best** linear model given the data.
- Many different possible **evaluation** criteria!
- Most common choice is to find the ***w*** that minimizes:

$$Err(w) = \sum_{i=1:n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

(A note on notation: Here ***w*** and ***x*** are column vectors of size $m+1$.)

Least-squares solution method

- Re-write in matrix notation: $f_{\mathbf{w}}(X) = X\mathbf{w}$

$$Err(\mathbf{w}) = (Y - X\mathbf{w})^T(Y - X\mathbf{w})$$

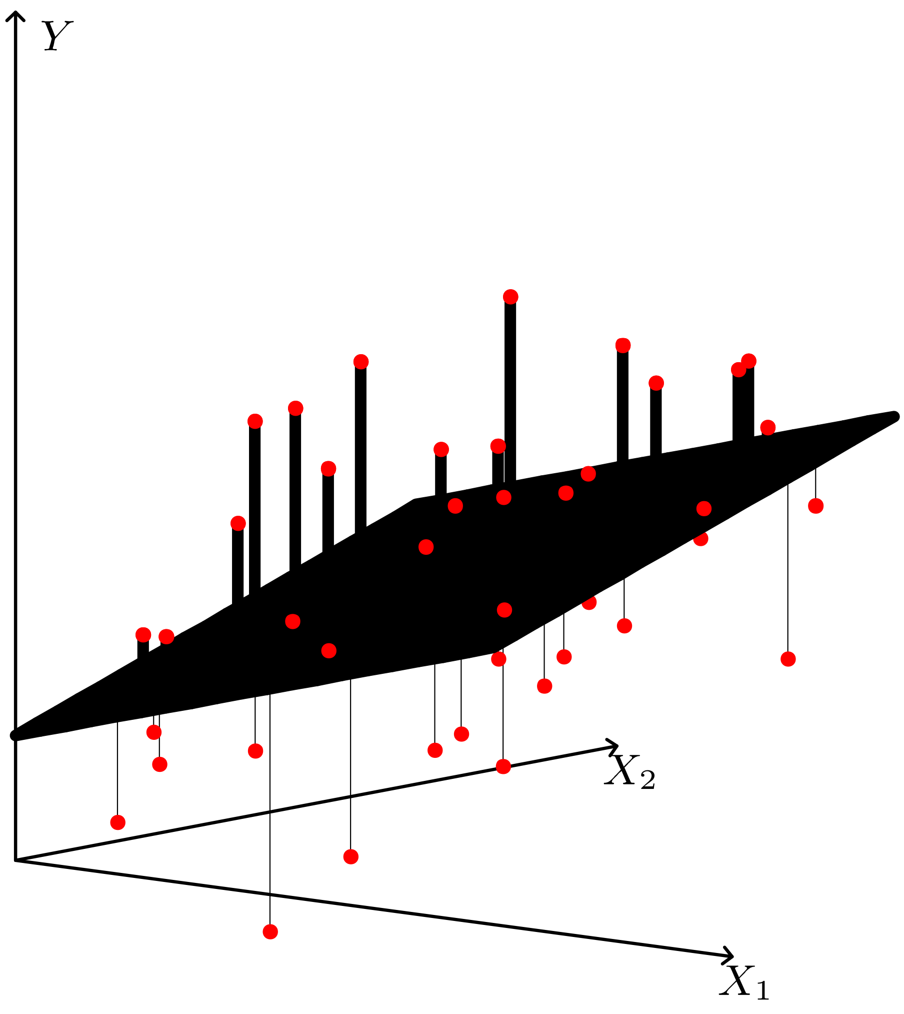
where X is the $n \times m$ matrix of input data,
Y is the $n \times 1$ vector of output data,
$\mathbf{w}$ is the $m \times 1$ vector of weights.

- To minimize, take the derivative w.r.t. $\mathbf{w}$:

$$\partial Err(\mathbf{w})/\partial \mathbf{w} = -2\, X^T (Y - X\mathbf{w})$$

  - You get a system of m equations with m unknowns.
- Set these equations to 0: $X^T (Y - X\mathbf{w}) = 0$

Least-squares solution for $X \in \Re^2$



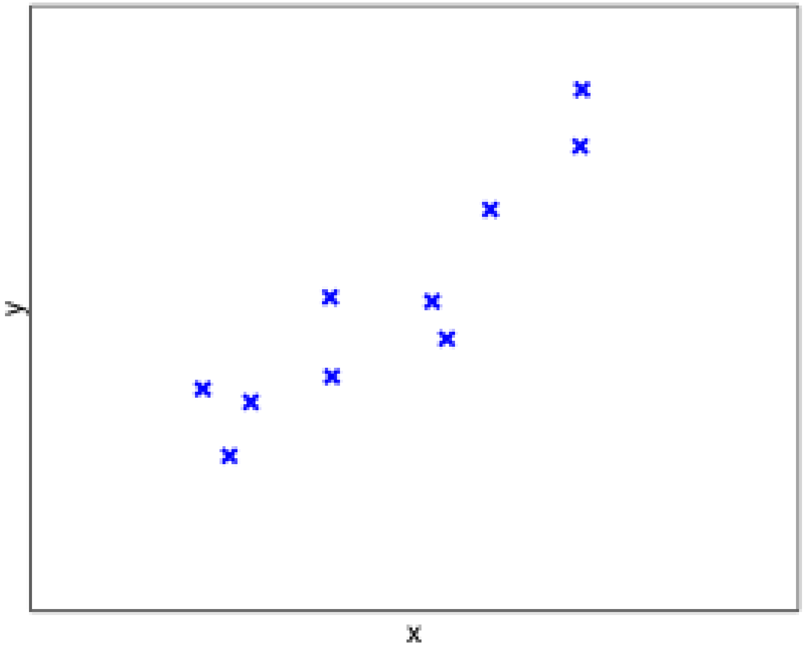
Least-squares solution method

- We want to solve for $\mathbf{w}$: $X^T (Y - X\mathbf{w}) = 0$
- Try a little algebra: $X^T Y = X^T X \mathbf{w}$

  $\hat{\mathbf{w}} = (X^T X)^{-1} X^T Y$

  ($\hat{\mathbf{w}}$ denotes the estimated weights)
- The fitted data: $\hat{Y} = X\hat{\mathbf{w}} = X (X^T X)^{-1} X^T Y$
- To predict new data $X' \rightarrow Y'$: $Y' = X'\hat{\mathbf{w}} = X' (X^T X)^{-1} X^T Y$

Example of linear regression



x	y
0.86	2.49
0.09	0.83
-0.85	-0.25
0.87	3.10
-0.44	0.87
-0.43	0.02
-1.10	-0.12
0.40	1.81
-0.96	-0.83
0.17	0.43

What is a plausible estimate of ***w*** ?

Data matrices

$$X^T X =$$

$$\left[\begin{array}{cccccccccc} 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \times \left[\begin{array}{cc} 0.86 & 1 \\ 0.09 & 1 \\ -0.85 & 1 \\ 0.87 & 1 \\ -0.44 & 1 \\ -0.43 & 1 \\ -1.10 & 1 \\ 0.40 & 1 \\ -0.96 & 1 \\ 0.17 & 1 \end{array}\right]$$

$$= \left[\begin{array}{cc} 4.95 & -1.39 \\ -1.39 & 10 \end{array}\right]$$

Data matrices

$$X^T Y =$$

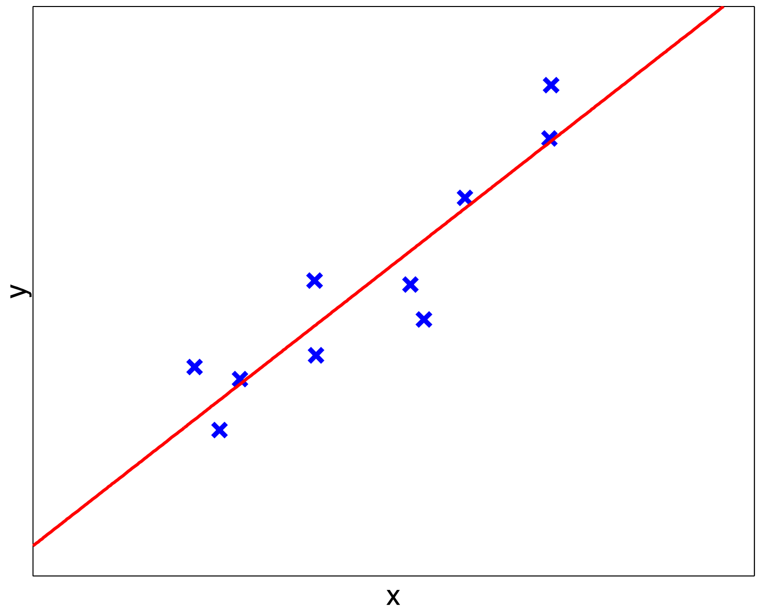
$$\begin{bmatrix} 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{bmatrix}$$

$$= \begin{bmatrix} 6.49 \\ 8.34 \end{bmatrix}$$

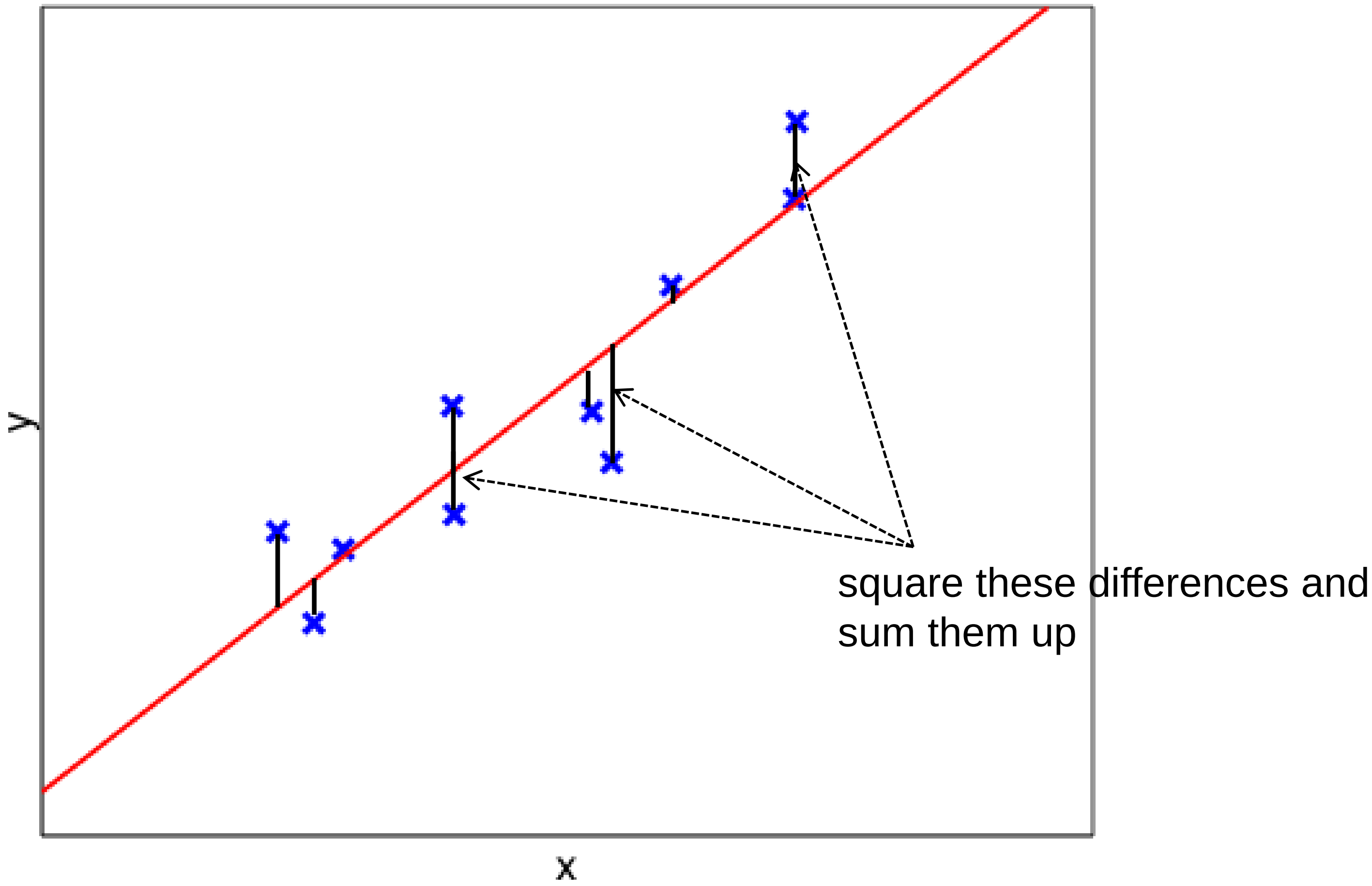
Solving the problem

$$\mathbf{w} = (X^T X)^{-1} X^T Y = \begin{bmatrix} 4.95 & -1.39 \\ -1.39 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 6.49 \\ 8.34 \end{bmatrix} = \begin{bmatrix} 1.60 \\ 1.05 \end{bmatrix}$$

So the best fit line is $y = 1.60x + 1.05$.



Sum of squared errors



Computational cost of linear regression

- What operations are necessary?
  - Matrix multiplication:
    - For A^{nxm} x B^{mxp}: We need *nmp* operations.
  - Matrix inversion:
    - For A^{mxm}: We need m^3 operations.
  - In total: 1 matrix inversion + 3 matrix multiplications

- So we can do linear regression in polynomial time, i.e. $O(n^3)$.

Steps in supervised learning

1. Decide what the input-output pairs are.
2. Decide how to encode inputs and outputs.
   - This defines the input space ***X*** and output space *Y*.
3. Choose a class of hypotheses / representations *H*.
   - e.g., linear functions.
4. Choose an error function (cost function) to define best hypothesis.
   - e.g., Least-mean squares.
5. Choose an algorithm for searching through space of hypotheses.
   - e.g., Taking derivatives of the error function wrt parameters of the hypothesis, setting to 0 and solving the resulting system of equations.

Polynomial fits

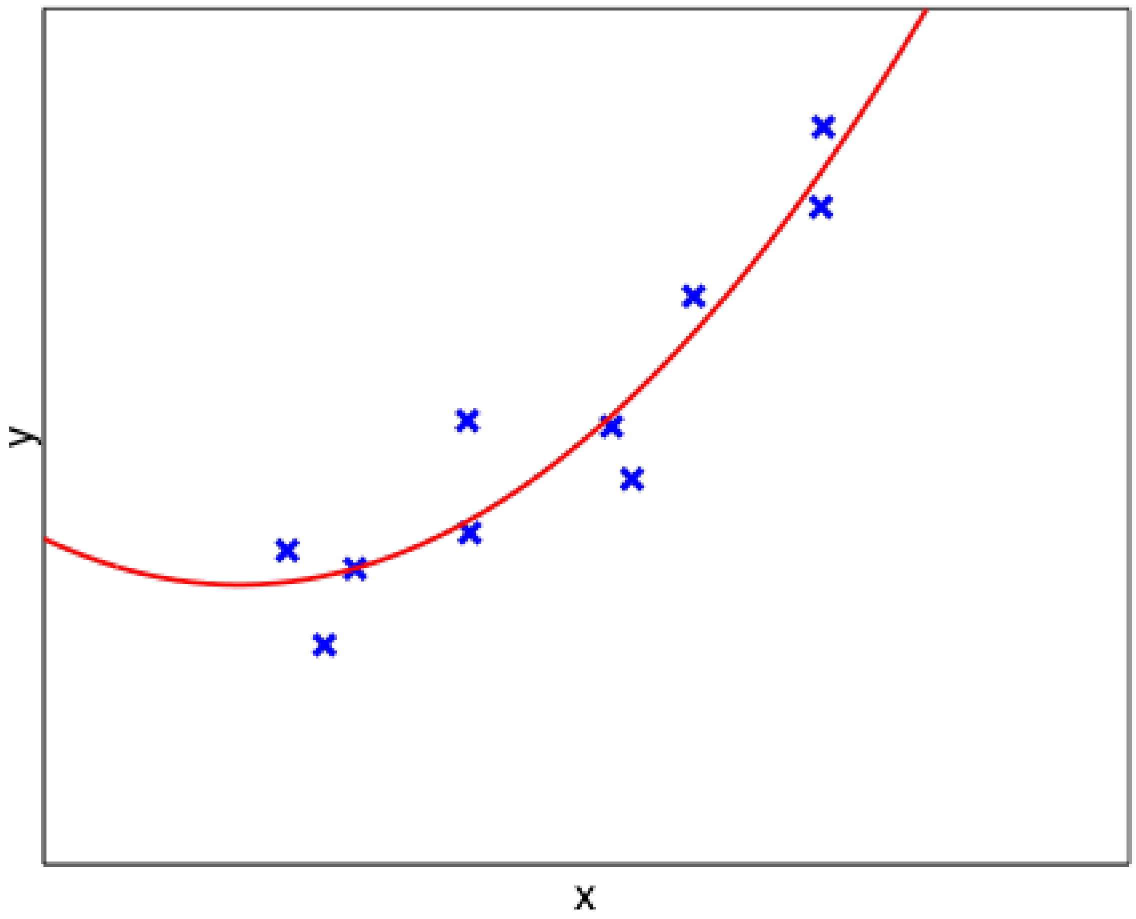
- Suppose we want to fit a higher-degree polynomial to the data.

  e.g., $y = w_0 + w_1 x + w_2 x^2$

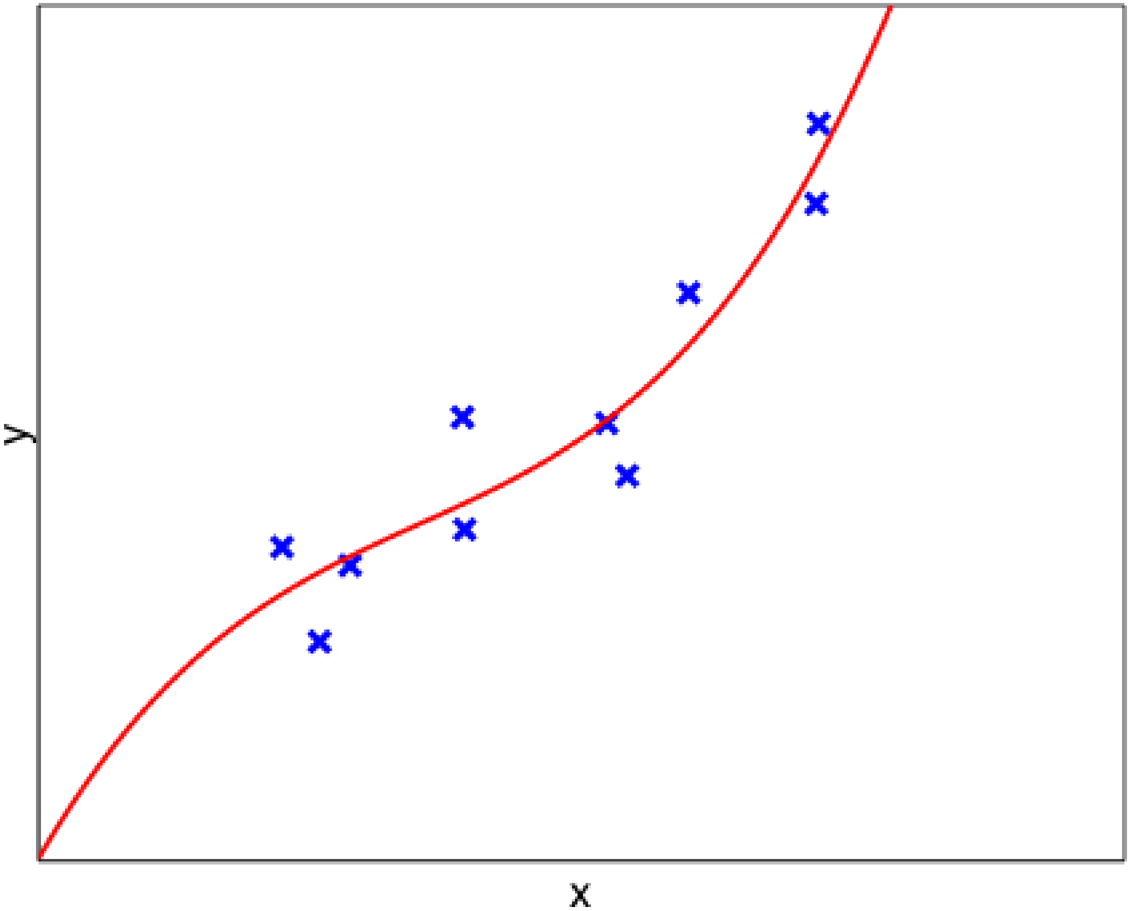
- Suppose for now that there is a single input variable, x. Do we need to change our method?
  - No! Output is still a linear function with respect to the weights!
  - Only difference is that now we have more inputs (x^2 is an input feature).
- If we have more than one input variable, cross factors also have to be considered:

  e.g., $y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2$

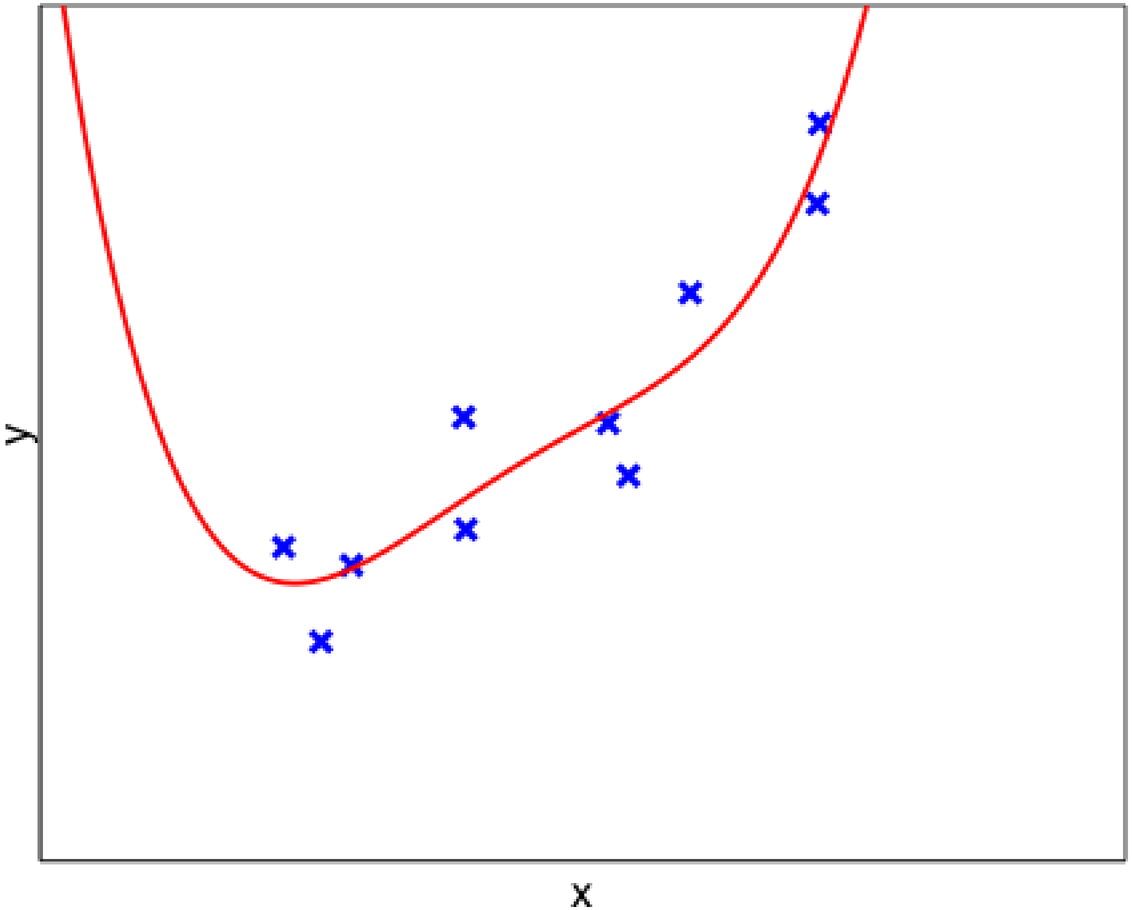
Example: $y = 0.68\,x^2 + 1.74\,x + 0.73$



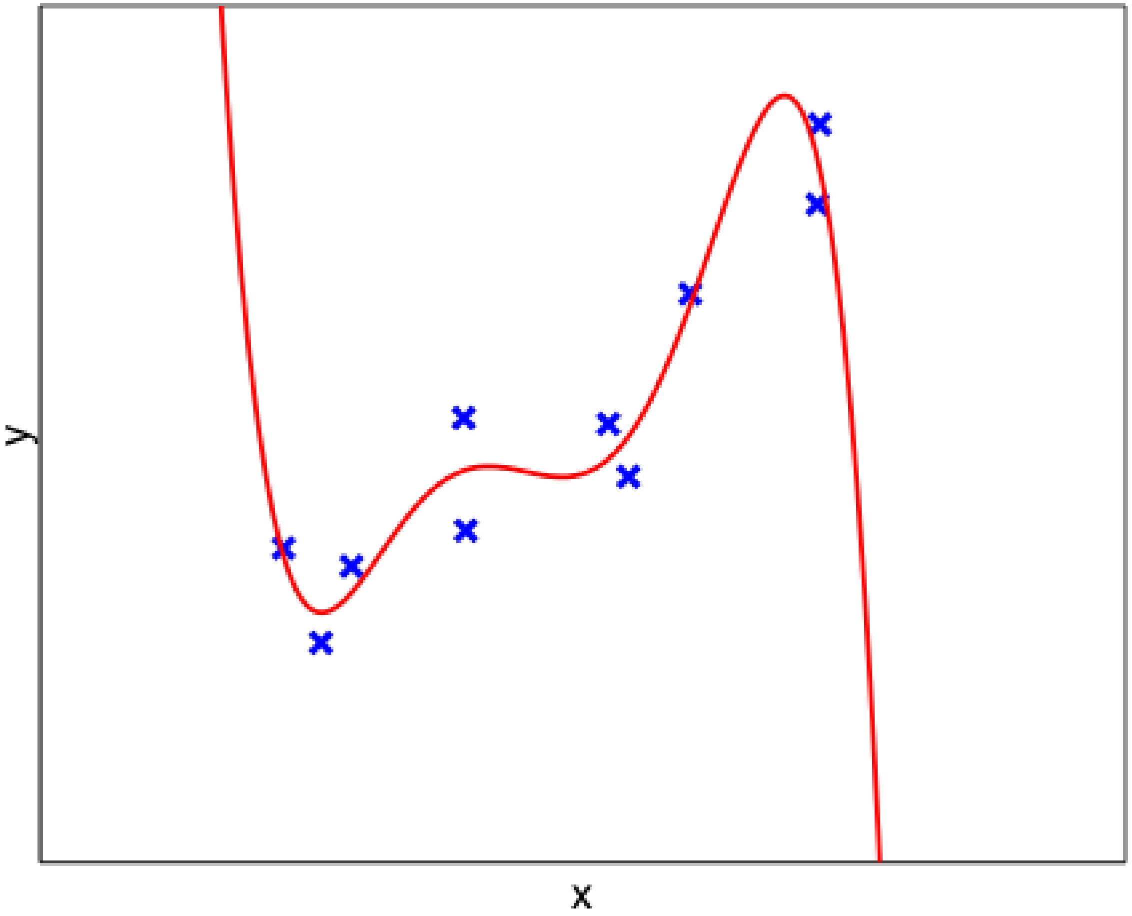
Order-3 fit



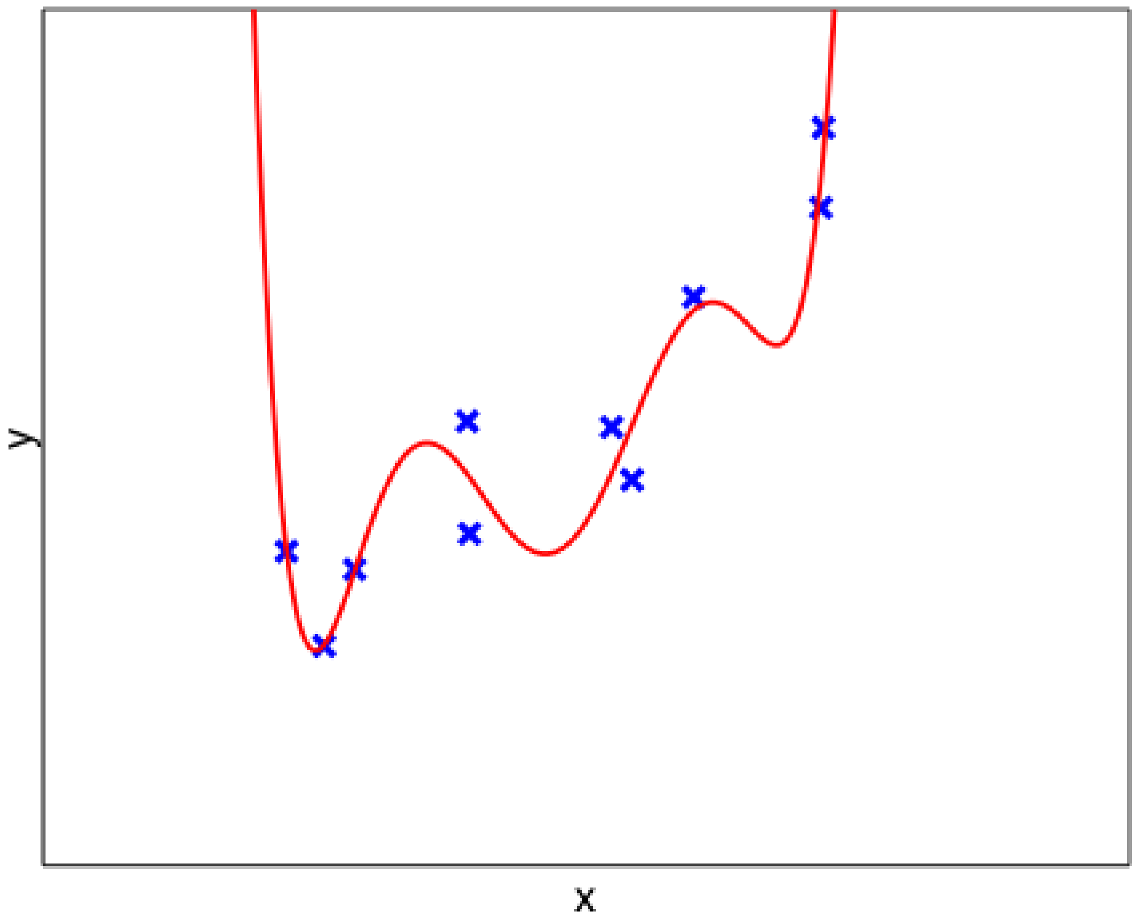
Order-4 fit



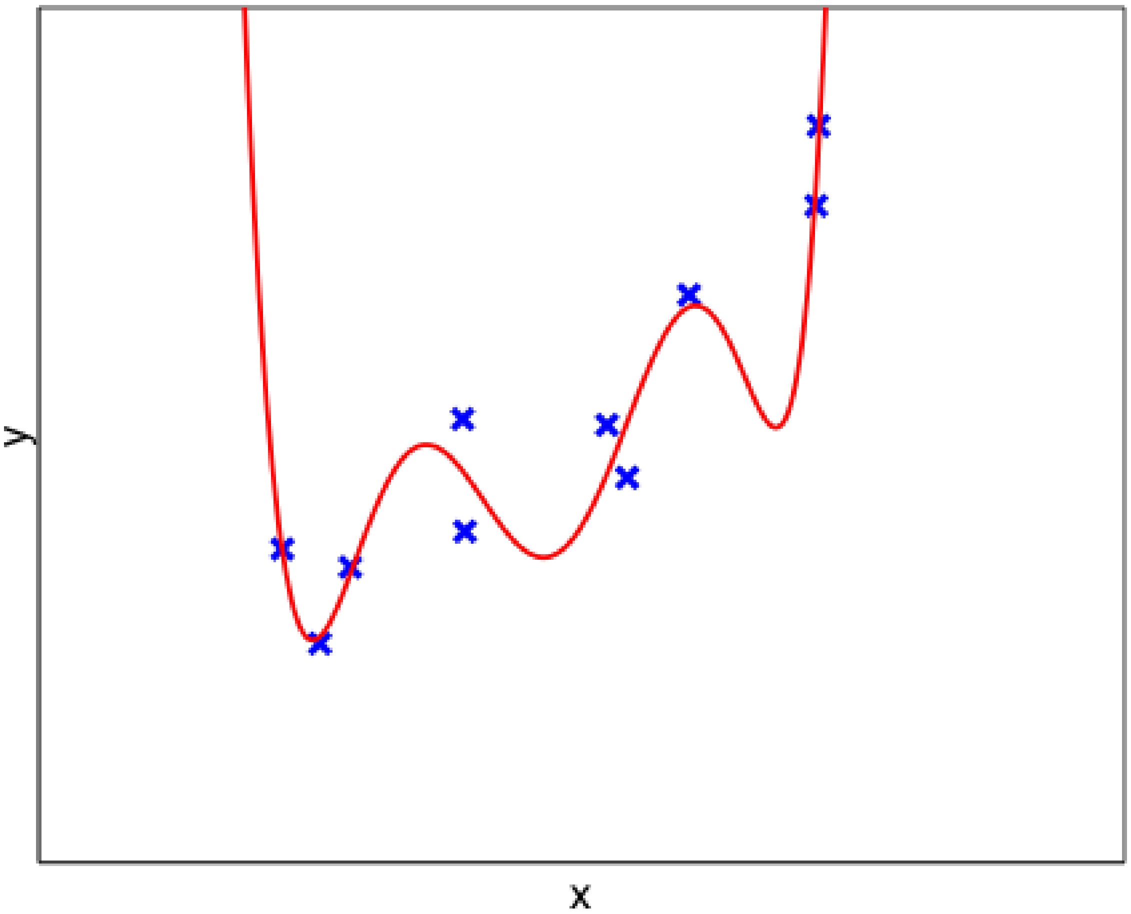
Order-5 fit



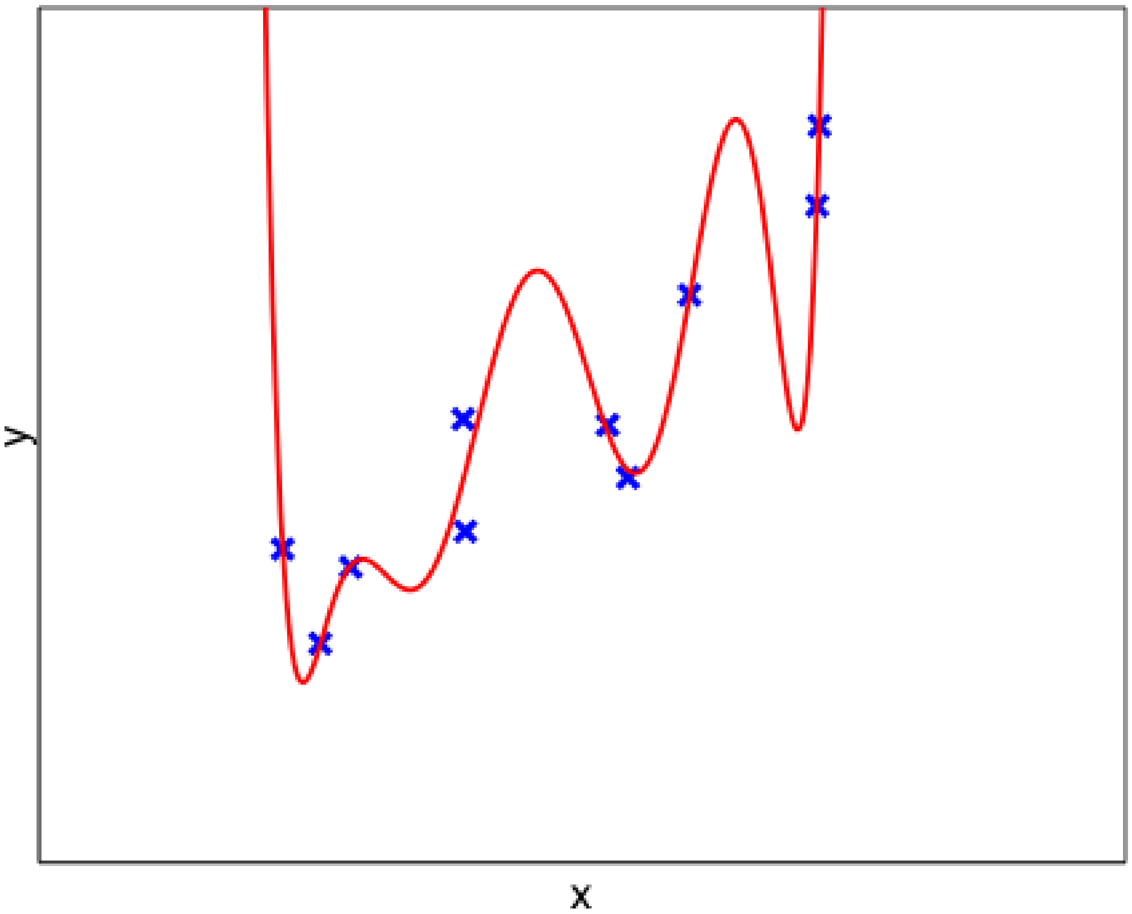
Order-6 fit



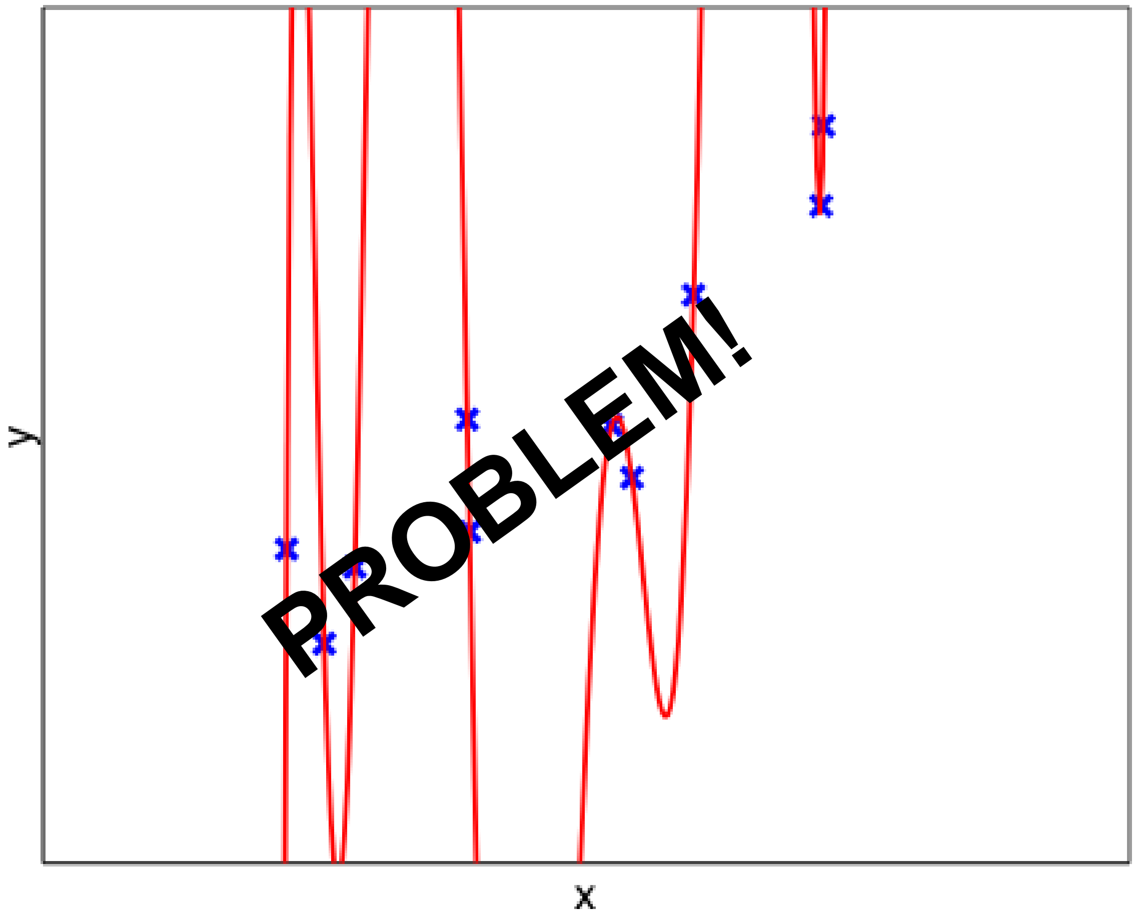
Order-7 fit



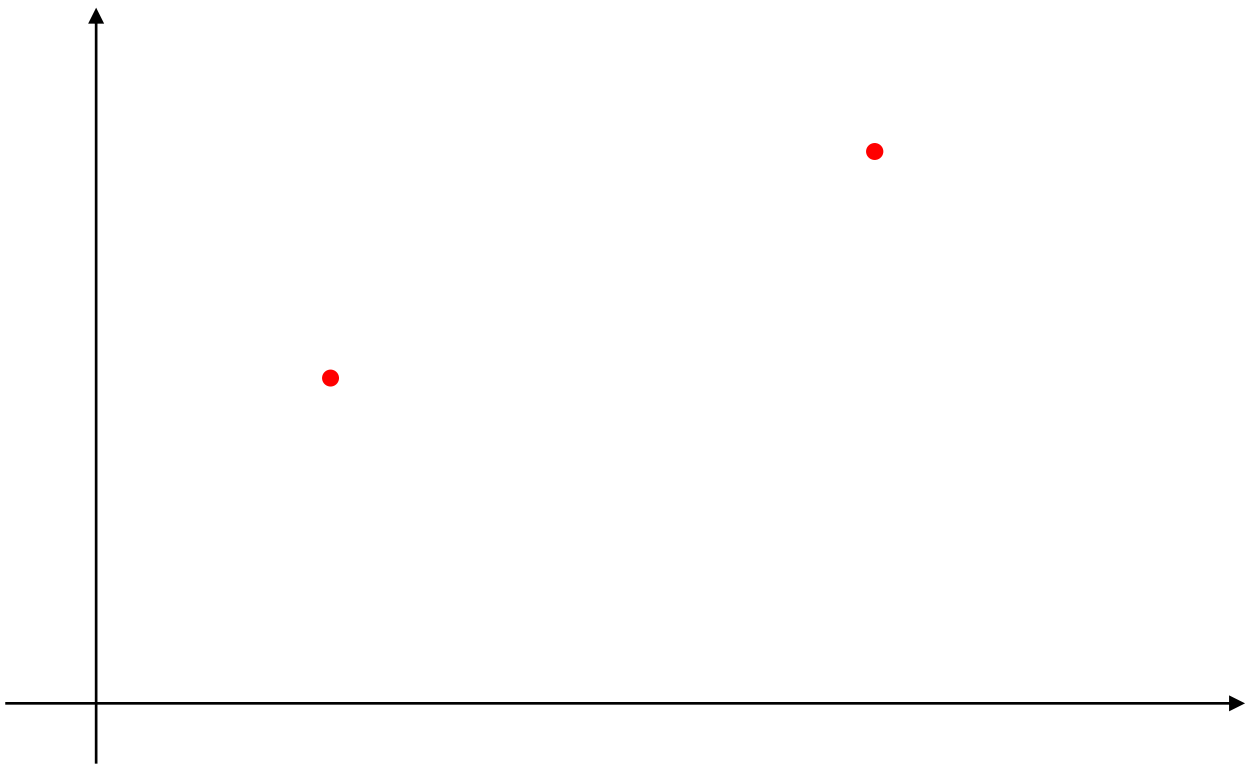
Order-8 fit



Order-9 fit



"There is a linear relationship between these points"



- Something's wrong with this argument…

Overfitting

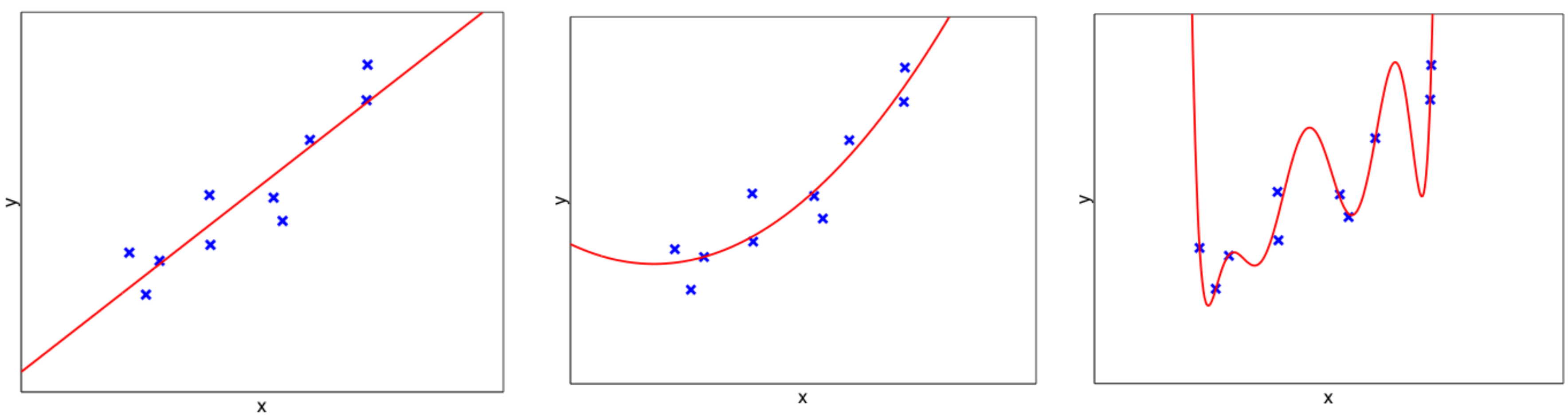
- We can find a hypothesis that explains perfectly the training data, but *does not generalize* well to new data.
  - E.g. a look-up table of all training examples!

- In the example above, there are enough parameters for the hypothesis to "memorize" the data points, **but it is "wild" everywhere else!**

- A general, *very important* problem for *all* machine learning algorithms.

Overfitting

- The *d*=degree polynomial with *d*=8 has zero training error!
  - Looking at the data and the different hypotheses, we see *d*=1 and *d*=2 are better fits (and suspect they have lower error.)



Results on training set vs validation set

- Hold out a **validation set** (from available data) to estimate true error.

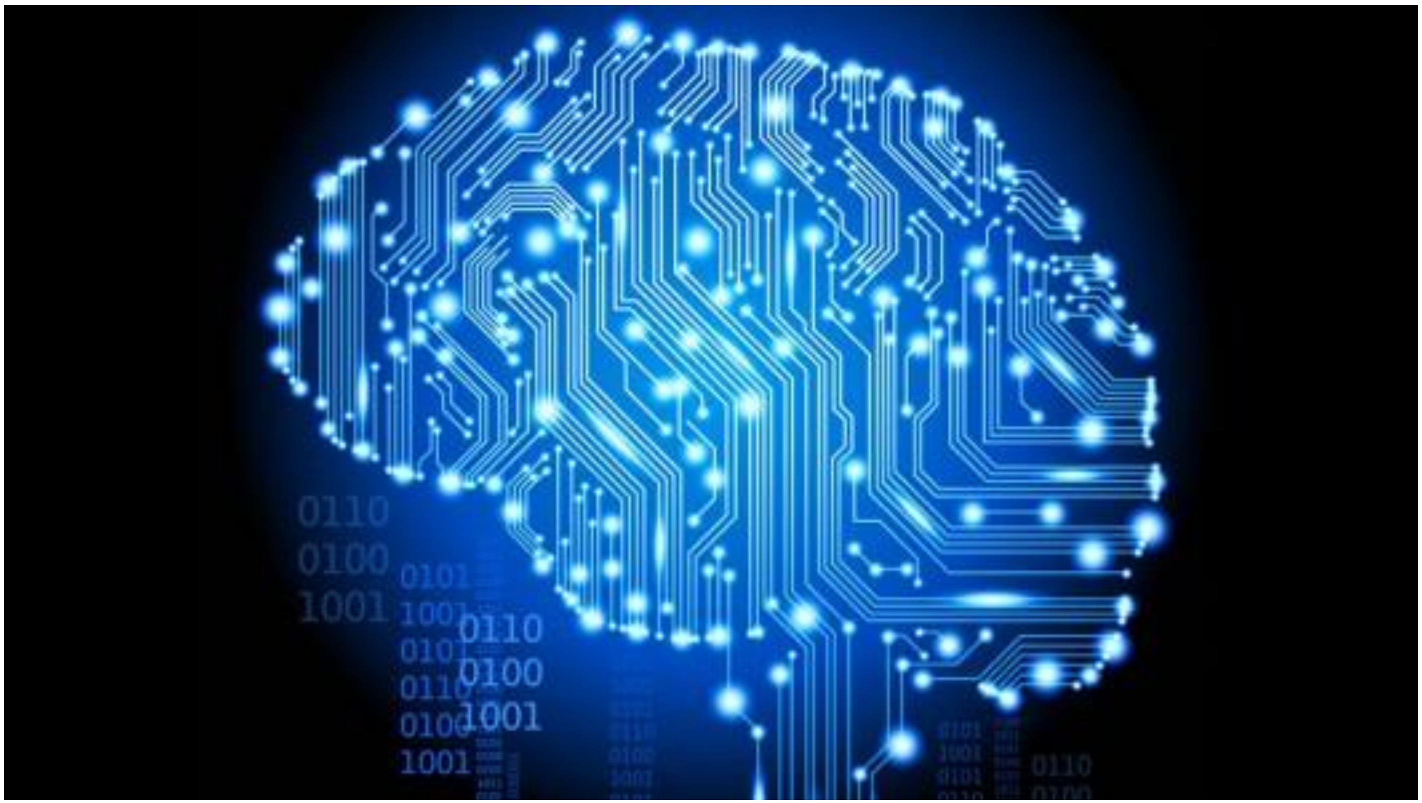
Underfitting for d < 2.

Optimal choice: *d*=2.

Overfitting for *d* > 2.

d	$\text{Error}_{\text{train}}$	$\text{Error}_{\text{valid}}$
1	0.2188	0.3558
2	0.1504	0.3095
3	0.1384	0.4764
4	0.1259	1.1770
5	0.0742	1.2828
6	0.0598	1.3896
7	0.0458	38.819
8	0.0000	6097.5

Learning complex non-linear functions



Connectionist models

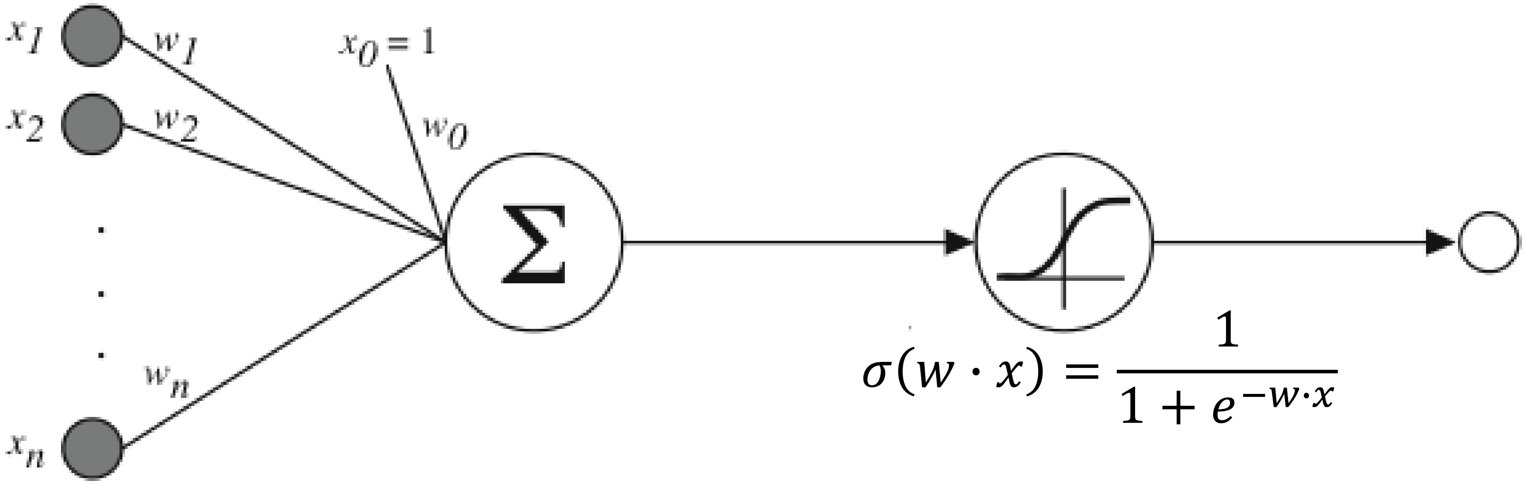
Hypothesis: A computational architecture similar to the brain could duplicate (at least some of its) wonderful abilities.

Properties of **Artificial Neural Networks (ANNs)**:

- Many neuron-like threshold switching units.
- Many weighted interconnections among units.
- Highly parallel, distributed process.
- Emphasis on tuning weights automatically.

Many different kinds of architectures, motivated both by biology and mathematics/efficiency of computation.

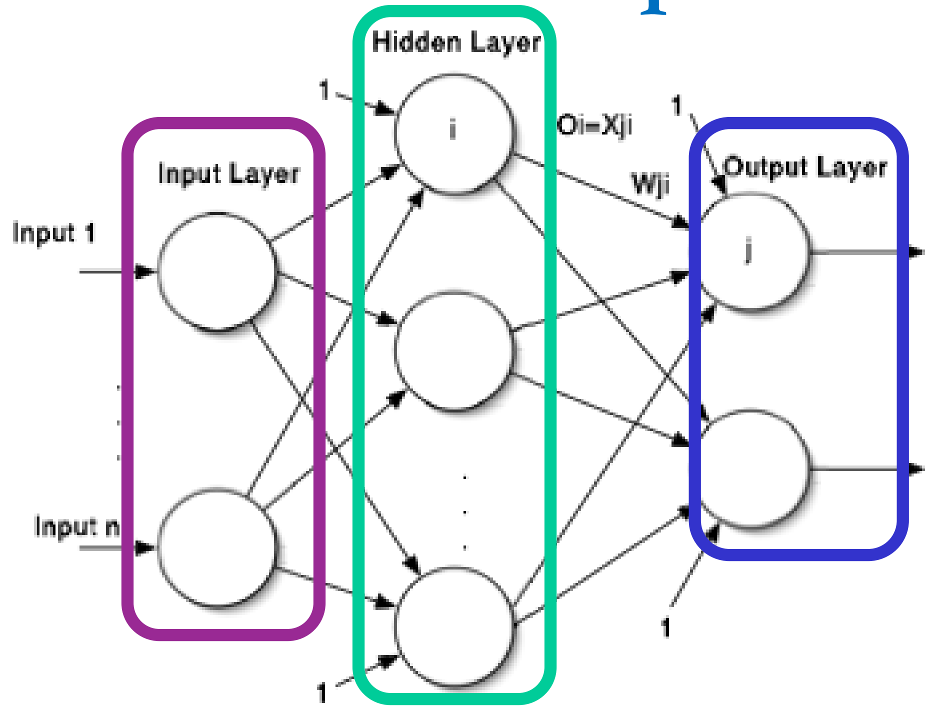
Sigmoid computation unit



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

- σ is the sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}}$
- It has the following nice property: $\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$
  - Why would we care what the derivative is?

Networks of simple units



- **Neurons** with sigmoid activation, arranged in layers.
  - Layer 0 is the input layer, its units just copy the input.
  - Last layer (layer K) is the output layer, its units provide the output.
  - Layers *1, .., K-1* are hidden layers, cannot be detected outside of network.

Feed-forward neural networks

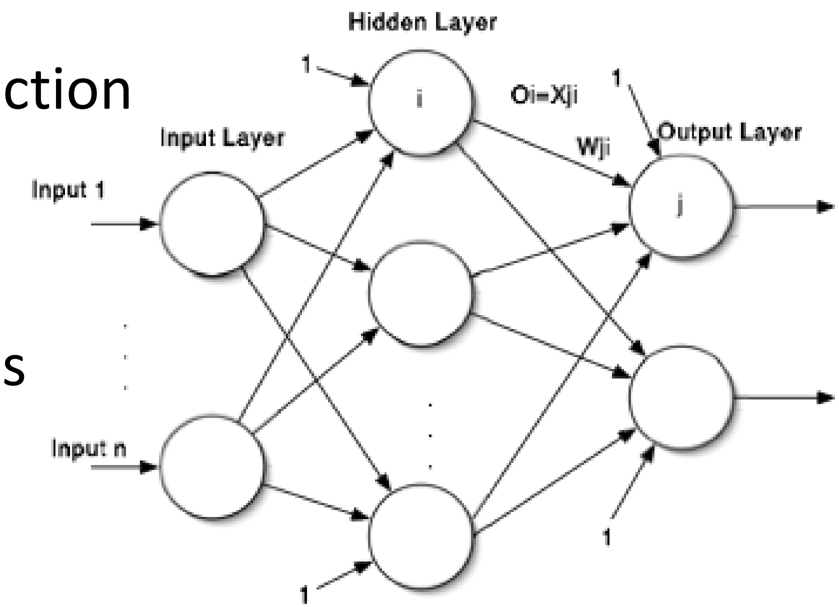
Notation:

- w_{ji} denotes weight on connection from unit i to unit j.
- By convention, $x_{j0} = 1, \ \forall j$
- Output of unit j, denoted o_j is computed using a sigmoid:

  $$o_j = \sigma(\mathbf{w}_j \cdot \mathbf{x}_j)$$

  where $\mathbf{w}_j$ is vector of weights entering unit j

  $\mathbf{x}_j$ is vector of inputs to unit j
- By definition, $x_{ji} = o_i$.



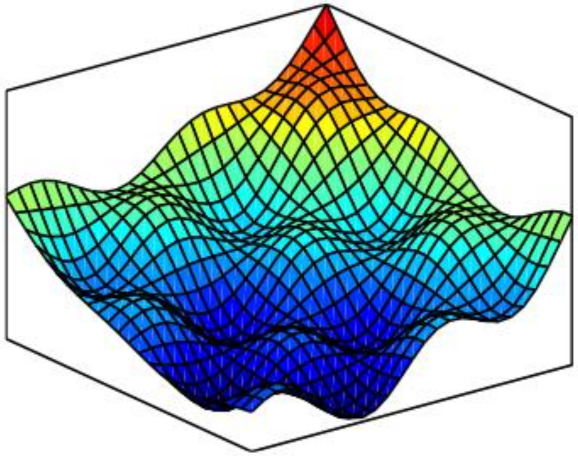
Computations with Neural Networks

1. Given the input, compute the output
    - Go through network from inputs to output neurons, passing the signals through the activation functions

2. Learn good weights for the neural network
    - What is good? Need an **objective function**
        - e.g., mean squared error loss just as in regression
        - There are also alternatives for discrete outcomes, such as categorical cross-entropy.
    - Also need an learning algorithm. Popular choice: **gradient descent**

Gradient descent

- The gradient of *f* at a point $<w_0, w_1, \ldots, w_n>$ can be thought of as a vector indicating which way is “uphill”.
  - We’re doing hill climbing in continuous space!



- If this is an error function, we want to move “downhill” on it, i.e. in the direction opposite to the gradient.

Gradient descent

- Standard operation: $\mathbf{w}^{k+1} = \mathbf{w}^k - \alpha_k \nabla f(\mathbf{w}^k)$

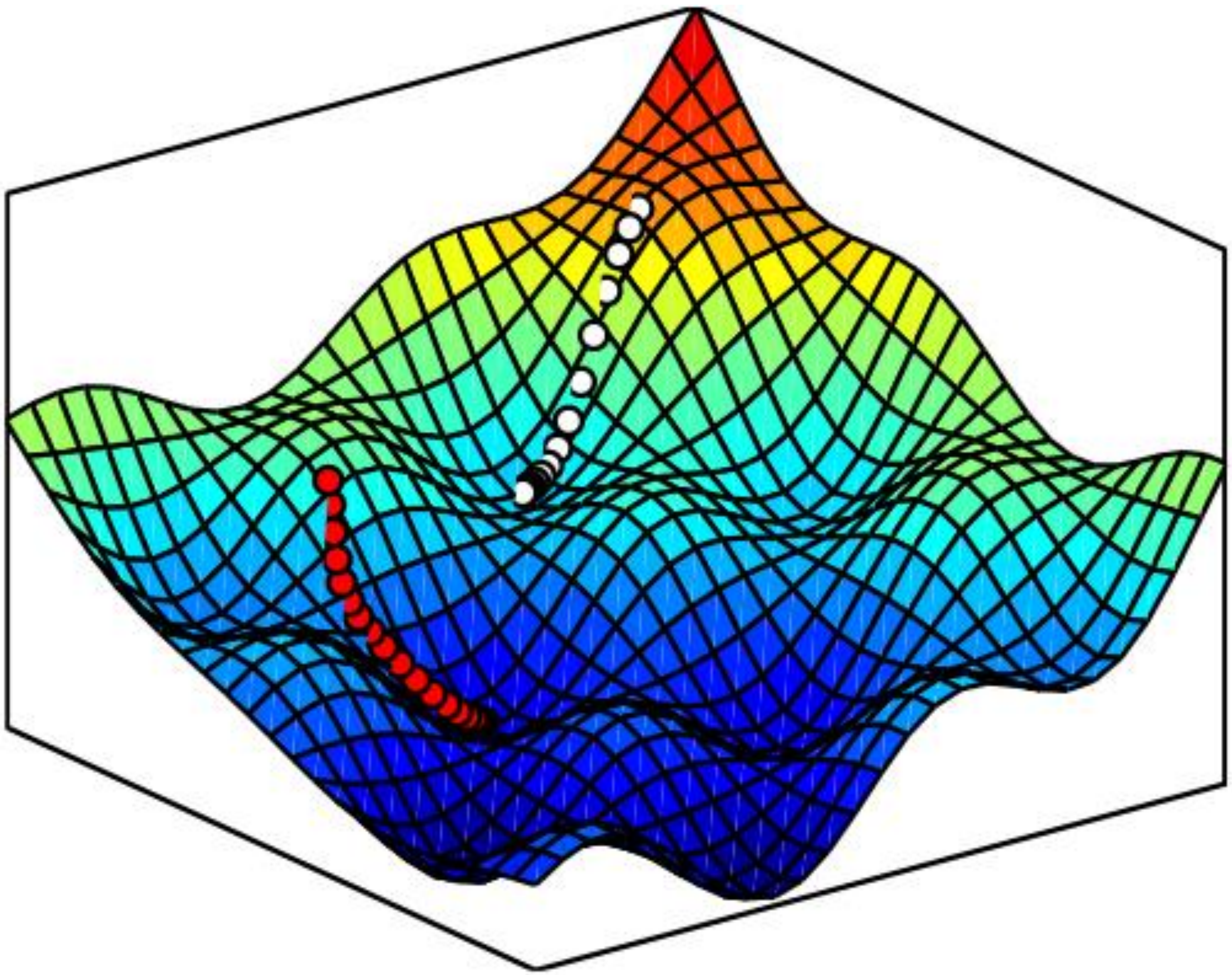
  where $\alpha_k > 0$ is the **step size** or **learning rate** for iteration k.

- The basic algorithm assumes that $\nabla f(w)$ is computable.
  - Need partial derivative w.r.t. each weight w_i :
  - Easy to compute for a linear function!
  - Also easy to compute for sigmoid functions!

$$\nabla f = \left\langle \frac{\partial}{\partial w_0} f, \frac{\partial}{\partial w_1} f, \ldots, \frac{\partial}{\partial w_n} f \right\rangle$$

- Produces a sequence of vectors $\mathbf{w}^1, \mathbf{w}^2, \mathbf{w}^3, \ldots$ with the goal that:
  - $f(\mathbf{w}^1) > f(\mathbf{w}^2) > f(\mathbf{w}^3) > \ldots$
  - $\lim_{k \to \infty} \mathbf{w}^k = \mathbf{w}$, locally optimal

Example gradient descent traces



Backpropagation algorithm

- Just do gradient descent over all the weights in the network!

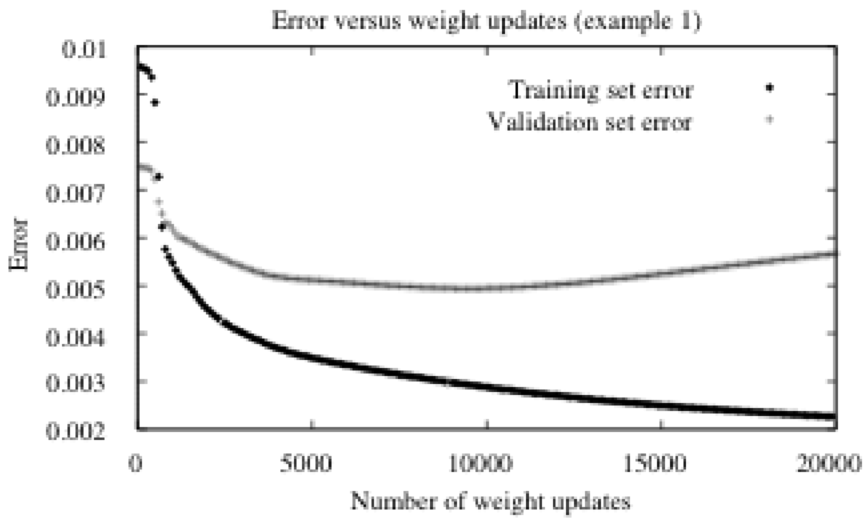
- We put together two phases:
  1. **Forward pass**: Compute the output of all units in the network, o_k, $k=N+1,\ldots, N+H+1$, going in increasing order or layers.

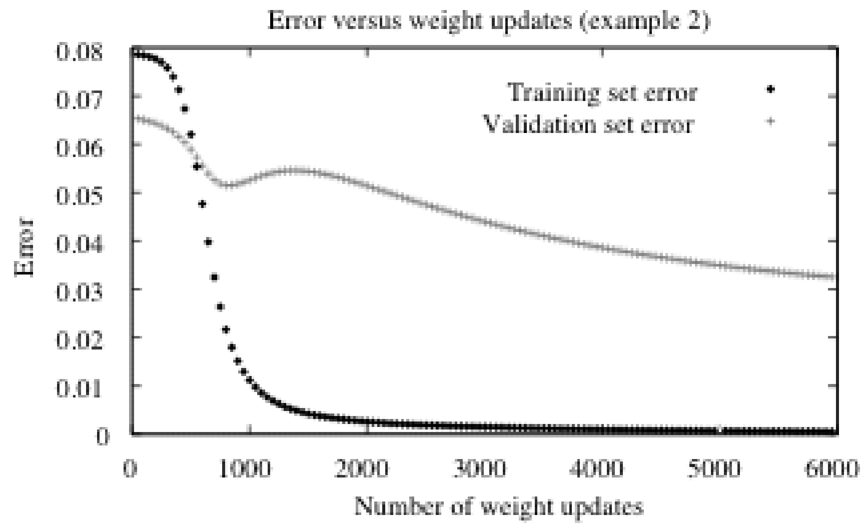
  2. **Backward pass**: Compute the δ_k updates described before, going from $k=N+H+1$ down to $k=N+1$ (in decreasing order of the layers).

  3. **Update** all the weights in the network:

$$w_{i,j} \leftarrow w_{i,j} + \alpha_{i,j}\delta_i x_{i,j}$$

Overfitting in feed-forward networks



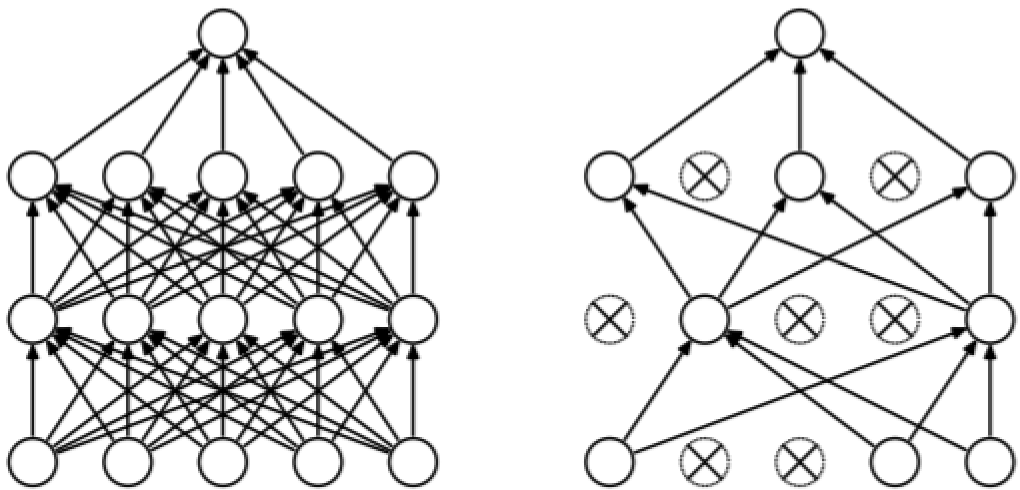


Overfitting in neural networks comes from three sources:

- Too many weights.
- Training for too long.
- Weights that have become too extreme.

Dropout (Srivastava et al., 2014)

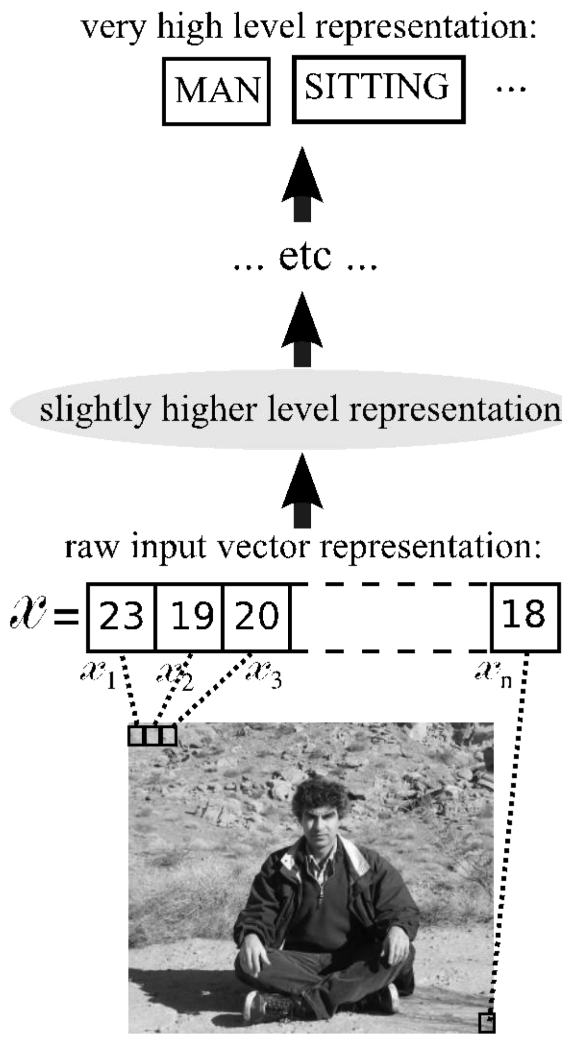
- Randomly drop units and connections in the network during training to fight overfitting



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

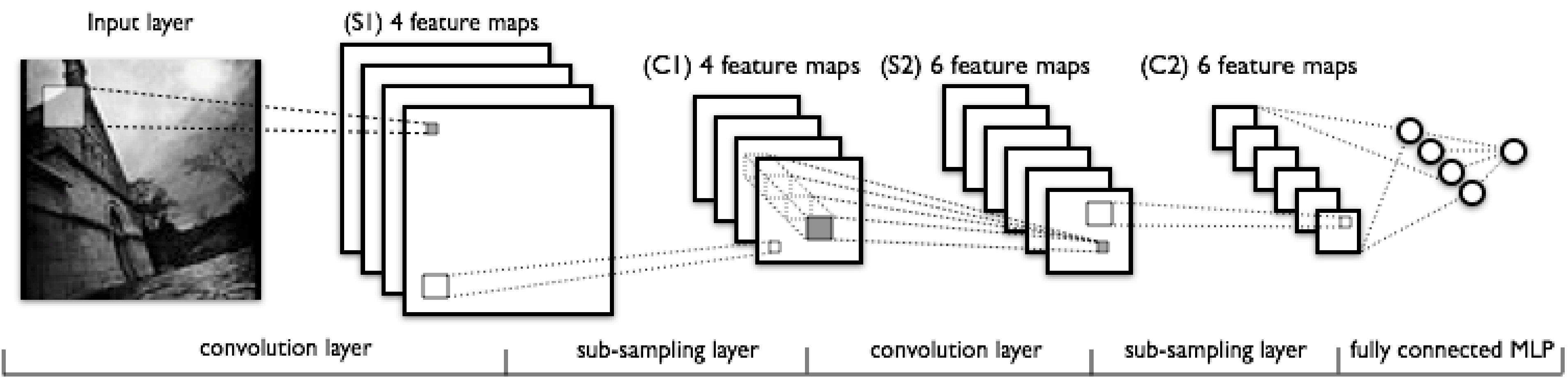
- Helps prevent network from "co-adapting" to the training set, memorizing specific cues in it which do not generalize

The deep learning objective



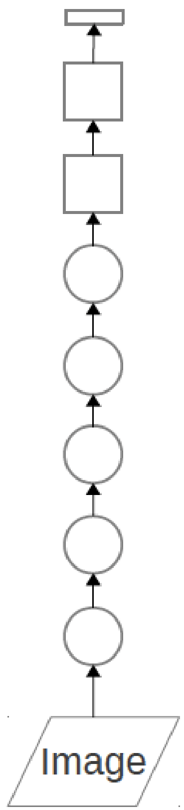
Convolutional neural nets

- S1: Convolve filters (one per feature map) over the image space.
  - Filter: Weight vector + sigmoid.
- C1: Aggregate locally data from feature maps.
  - E.g. Average / max function.
- S2 / C2: Repeat.
- Fully connected layer at the end. Train full network using backprop.

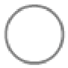


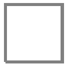
Empirical results: ImageNet

- Basic model:



- **Deep**: 7 hidden "weight" layers
- **Learned**: all feature extractors initialized at white Gaussian noise and learned from the data
- Entirely supervised
- **More data = good**

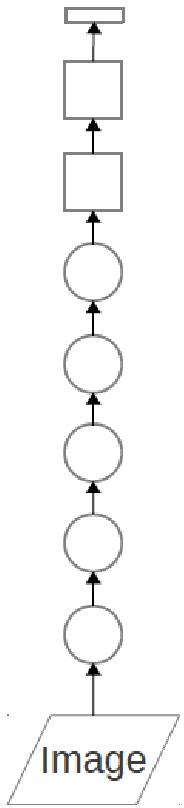

Convolutional layer: convolves its input with a bank of 3D filters, then applies point-wise non-linearity


Fully-connected layer: applies linear filters to its input, then applies point-wise non-linearity

From: *http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf*

Empirical results: ImageNet

- Basic model:



- Trained with stochastic gradient descent on two NVIDIA GPUs for about a week
- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections
- **Final feature layer:** 4096-dimensional

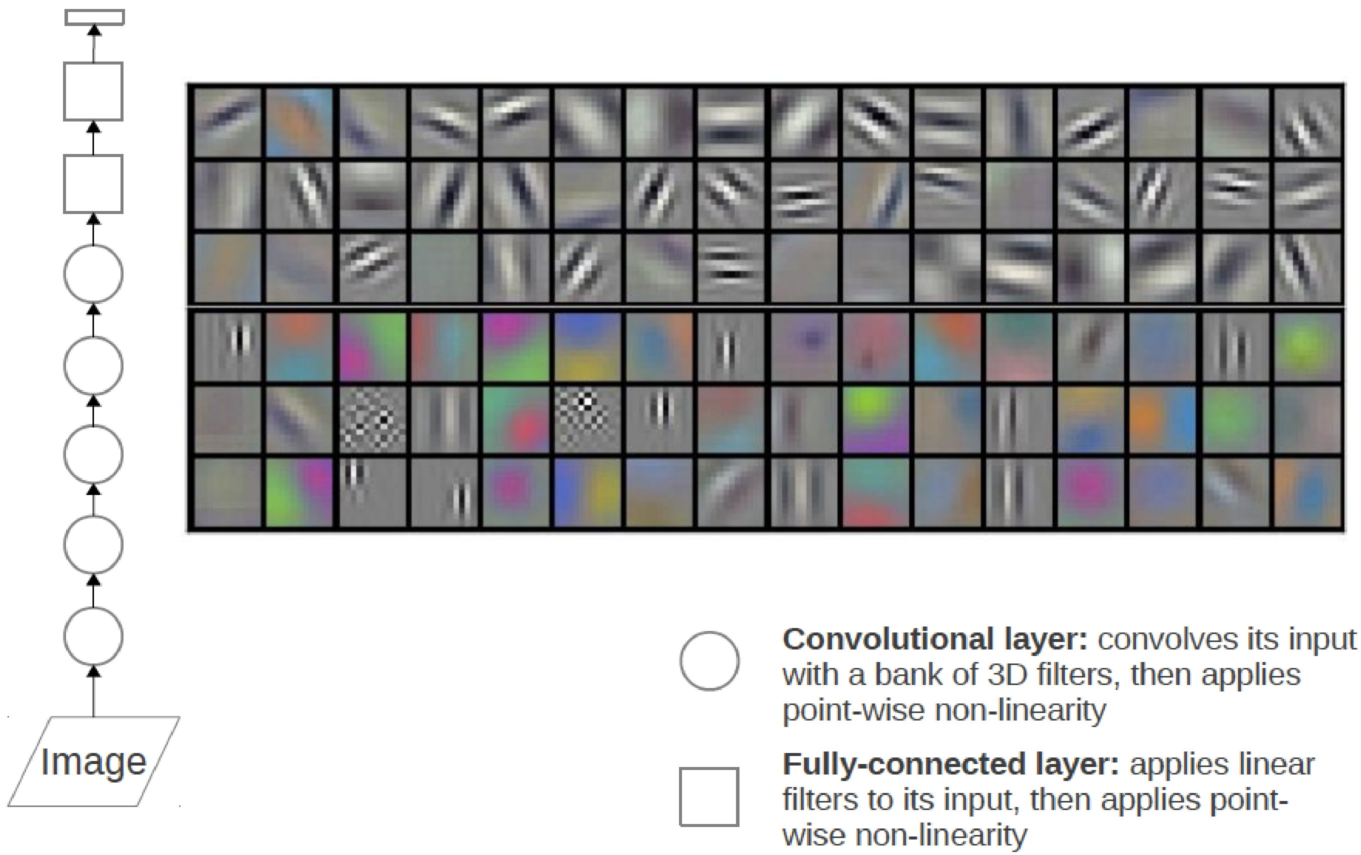
Convolutional layer: convolves its input with a bank of 3D filters, then applies point-wise non-linearity

Fully-connected layer: applies linear filters to its input, then applies point-wise non-linearity

From: *http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf*

Empirical results: ImageNet

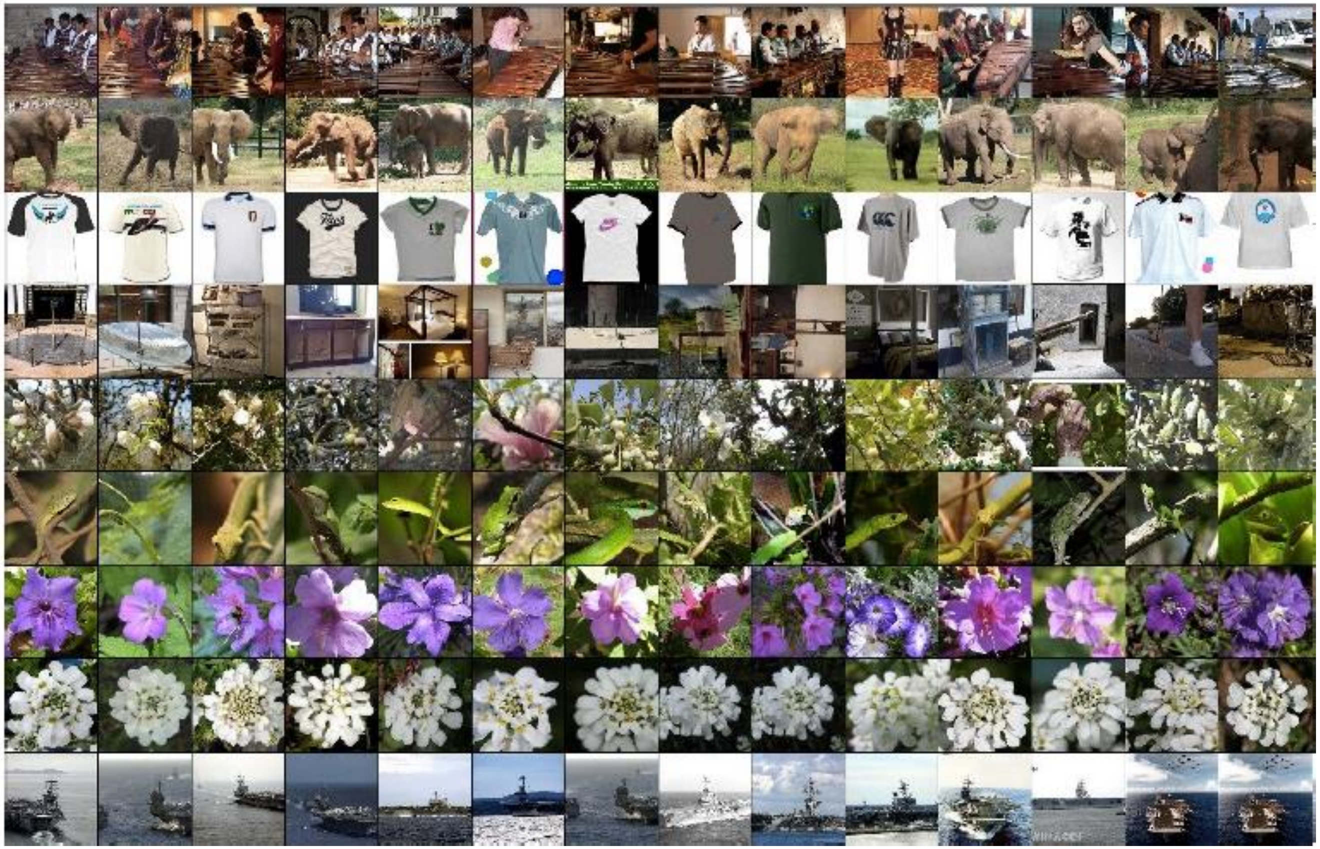
- Training results: 96 learned low-level filters



From: *http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf*

Empirical results: ImageNet

- Results for image retrieval (query items in leftmost column):



From: *http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf*

What you should know

- Formal problem definition for supervised learning.
- Linear regression (hypothesis class, cost function, algorithm).
- Polynomial regression (hypothesis class, cost function, algorithm).
- Underfitting and overfitting
- High-level idea of neural networks (not detailed mathematical formulation.)