A Brief Introduction to Machine Learning

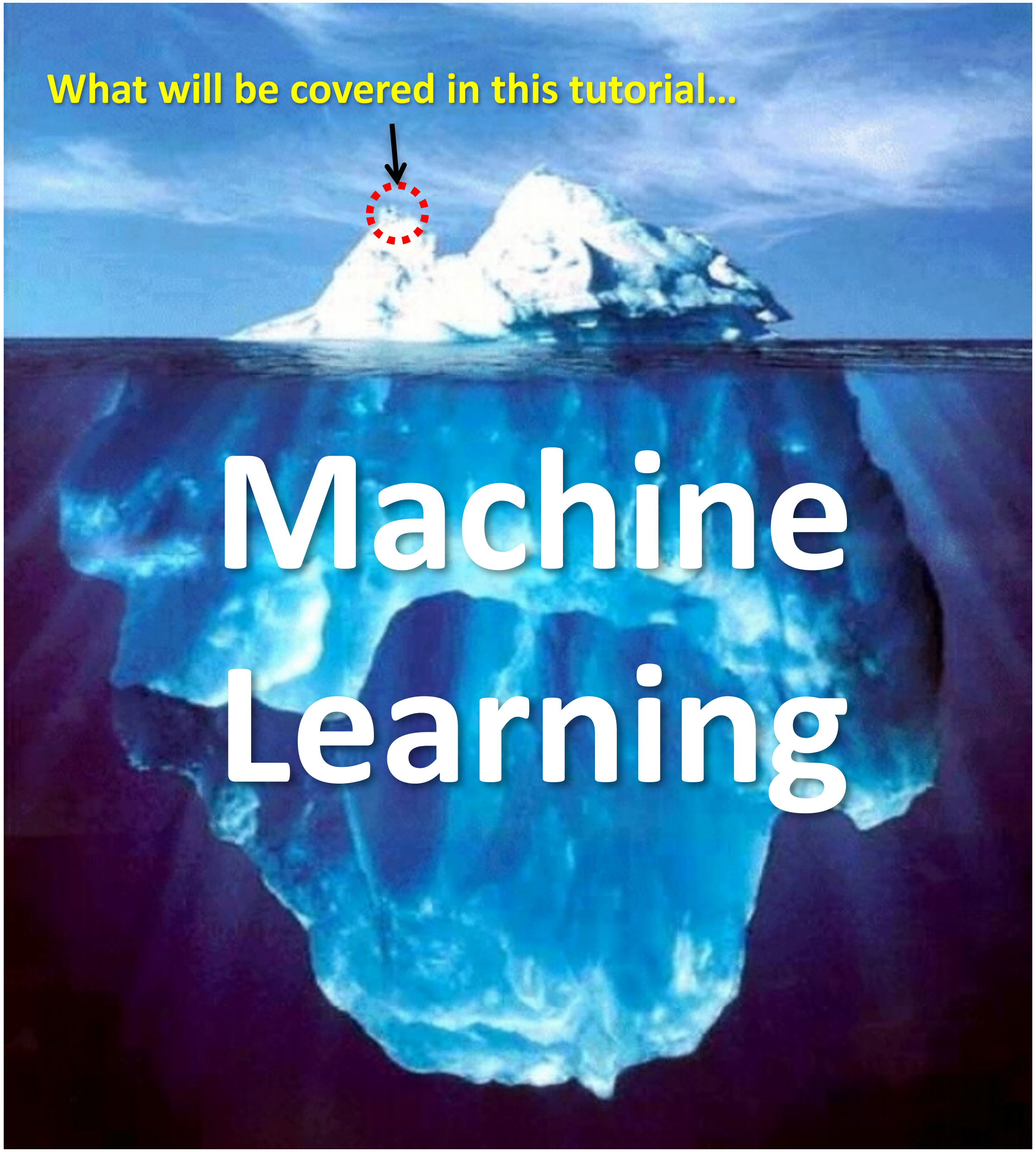
Weishan Dong 董维山

Big Data Lab, Baidu Research

About me

- Weishan Dong 董维山
  - Sr. Data Scientist, BDL, Baidu Research, 2016 - now
  - Research Leader/Manager, IBM Research – China (aka CRL), 2009 - 2016
  - PhD, Institute of Automation, CAS, 2009
  - Joint PhD study, University of Birmingham, UK, 2008
  - B.E., USTC, 2004

- Experience
  - 40+ publications, 50+ patents
  - Tech Leader of Baidu Medical Brain project, Baidu AI Technical Committee member (20 in all)
  - IBM Master Inventor (1/3000 per year), 10x IBM Invention Achievement Awards, 4x IBM Research Accomplishments, 1x IBM Research Division Award
  - Best Paper Award, IEEE INFORMS SOLI (2011)
  - Reviewer/PC member of TKDE, TEVC, JCST, IJCAI, ICDM, etc.


What will be covered in this tutorial…
Machine
Learning

Acknowledgements

- Materials of this tutorial are partly from
  - DragonStar 2012 summer course by Kai Yu
  - UFLDL online course by Andrew Ng
- and with reorganization

The BIG DATA era…

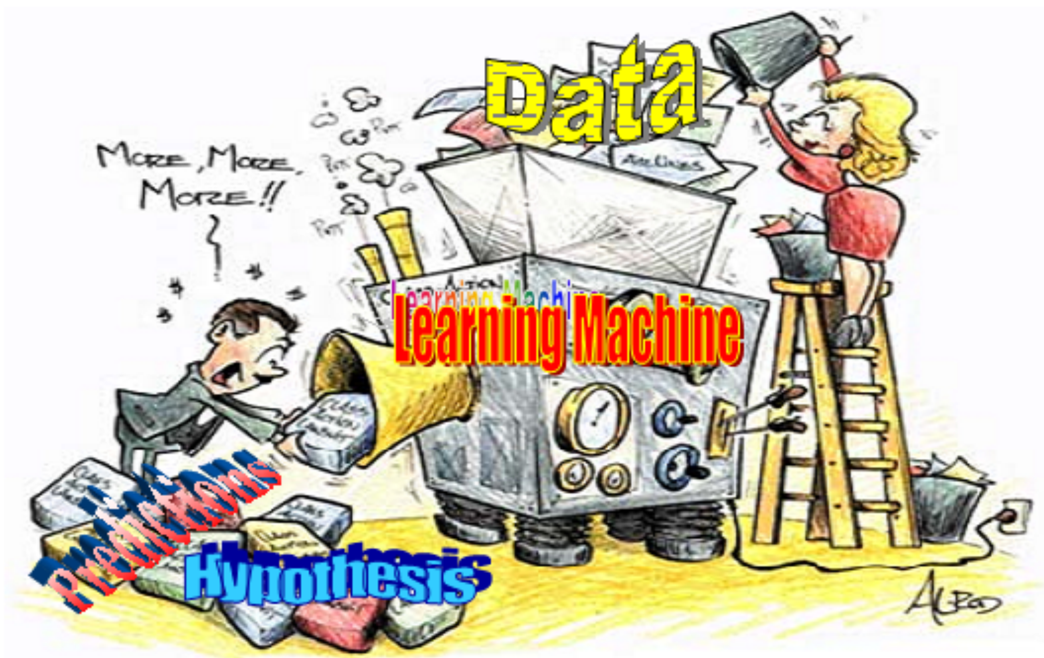
- Web: estimated Google index 45 billion pages
- Click-stream data: 10-100 TB/day
- Transaction data: 5-50 TB/day
- Satellite image feeds: ~1TB/day/satellite
- Biological data: 1-10TB/day/sequencer
- TV: 2TB/day/channel; YouTube 4TB/day uploaded
- Photos: 1.5 billion photos/week uploaded
- Digitized telephony: ~100 petabytes/day

- How to better utilize the value of data?

Machine Learning (ML)

WIKIPEDIA
The Free Encyclopedia

"... a scientific discipline that deals with the construction and study of algorithms that can learn from data. "



ML applications are everywhere

- Recommendation
- Web Search
- Computer Vision
- Driverless Car
- Speech Recognition
- Natural Language Processing (NLP)
- …

Recommendation

Amazon.com recommends products based on purchase history

Linder et al., 2003

Recommendation contributes 35% of sales in Amazon

Web Search

Search engine

-Query classification

-Ranking

-Spam detection

-…

Computational advertising

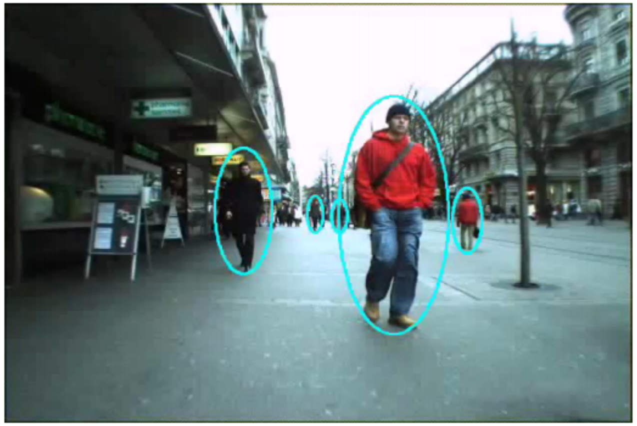
- Estimate click-through rate

- Optimal ads placement

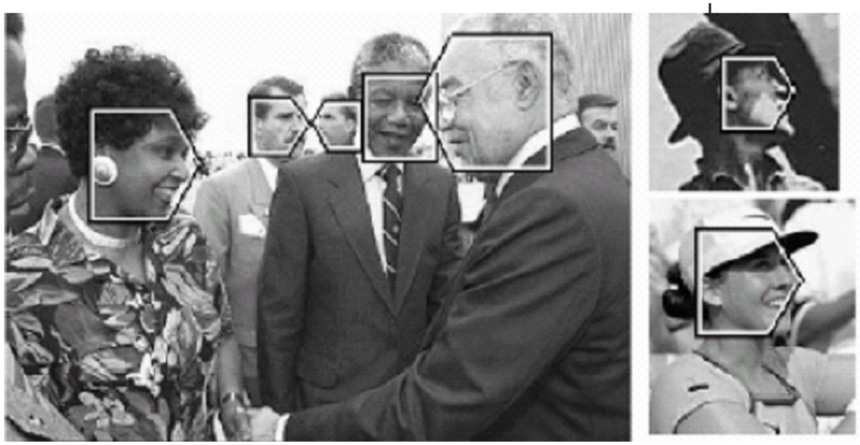
- …

Computer Vision

- Object recognition, detection, tracking
- Scene segmentation, understanding
- Action/behavior recognition
- Image tagging and search
- Optical character recognition (OCR)

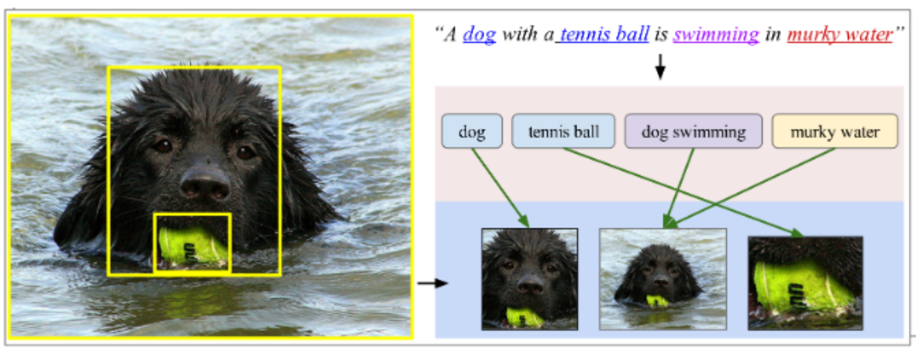


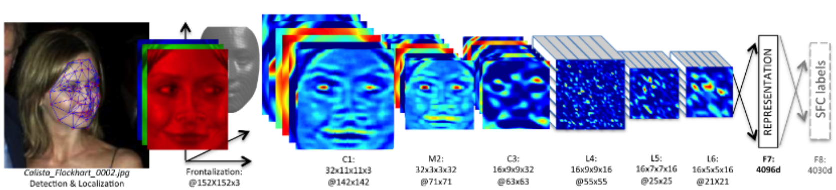
ImageNet Challenge: 1000 categories, 1.2 million images for training



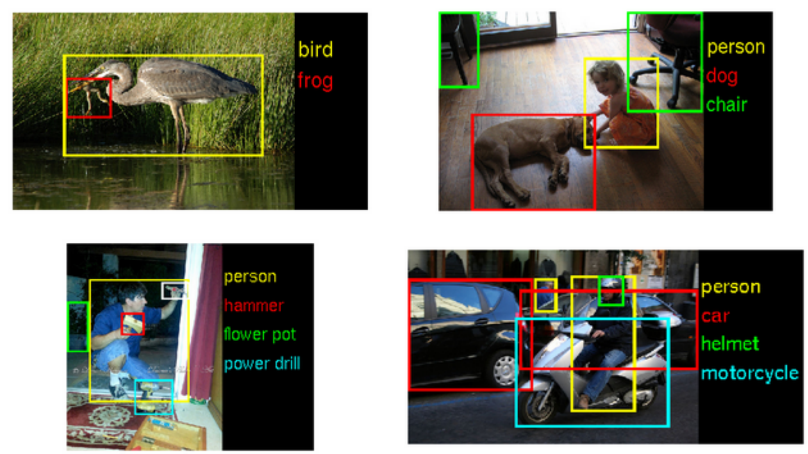
Records Set by Deep Learning

- ImageNet classification: < 5% top-5 error
- Face recognition: 98% accuracy
- Handwritten digit recognition: 0.23% error
- And, even scene understanding: Computer-generated image descriptions





IMAGENET



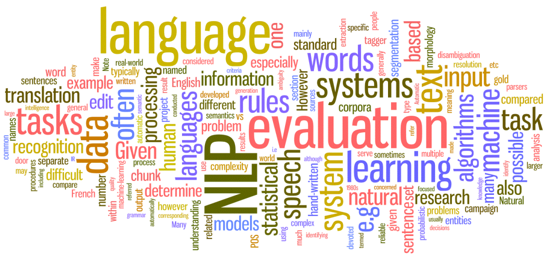

为了教计算机看懂图片并生成句子，

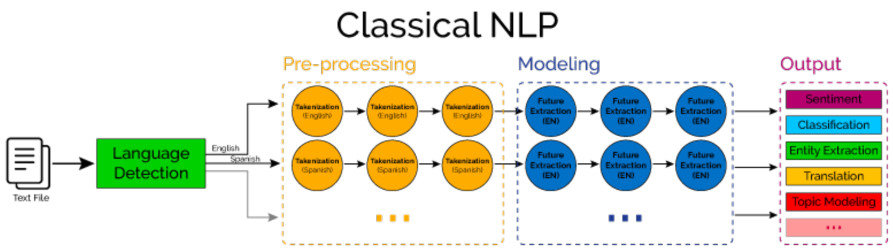
Driverless Car

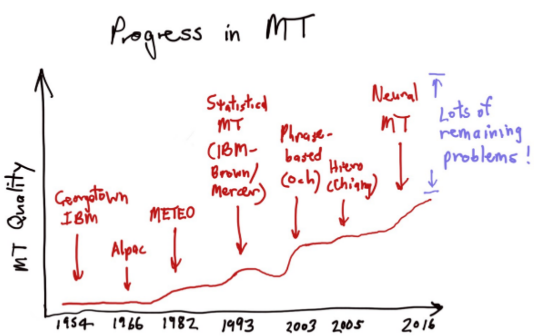
Speech Recognition

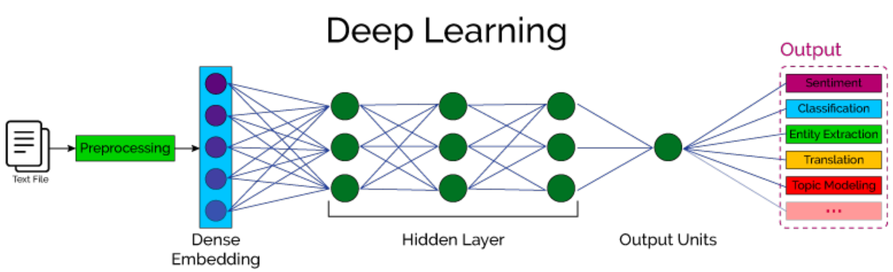
Natural Language Processing (NLP)

- Machine translation
- Information Extraction
- Information Retrieval, question answering
- Text classification, spam filtering, etc….









Paradigms of Machine Learning

- Supervised learning:
  - **Given $\{x_i, y_i\}$, learn $y=f(x; \theta)$**
  - Classification: y is categorical, e.g. digit recognition
  - Regression: y is continuous, e.g. temperature, stock price
  - Ranking: y is ordinal

- Unsupervised learning:
  - **Given $\{x_i\}$, learn $y=f(x; \theta)$**
  - Clustering: y is cluster label
  - Anomaly detection: y is abnormality

An Example of Binary Classification

x





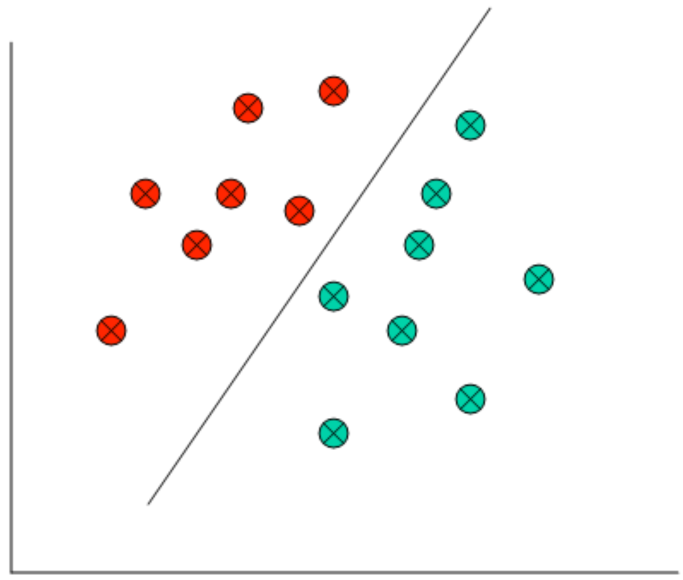

y

长颈鹿 长颈鹿 长颈鹿 神兽 神兽 神兽

x =

f(x) = ?

Linear Classifier



- A simplest classification model
- Help to understand nonlinear models
- Arguably the most useful classification method!

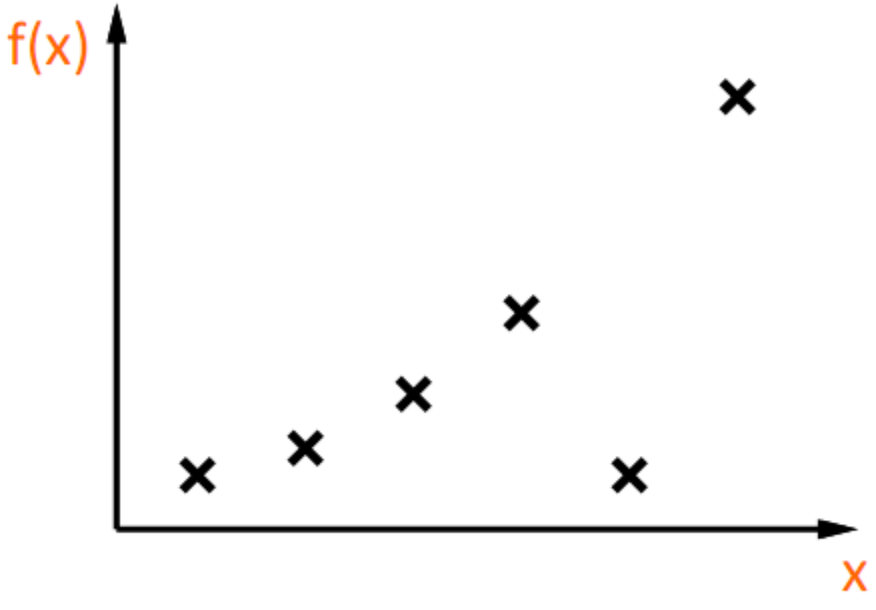
Two classical linear models

- Support Vector Machines (SVM)
- Logistic Regression (LR)

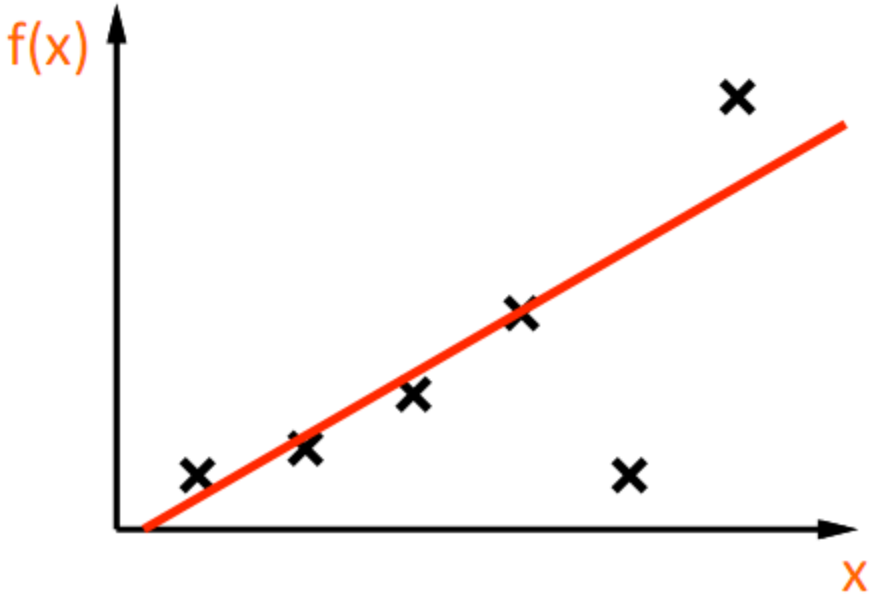
Support Vector Machines (SVM)

- A powerful method for 2-class classification
  - Become very hot since late 90's
- Key ideas
  - Use kernel function to transform low dimensional training samples to higher dimensions
  - Use quadratic programming (QP) to find the best classifier boundary hyperplane
- Better generalization (less overfitting)
  - What is 'overfitting'?

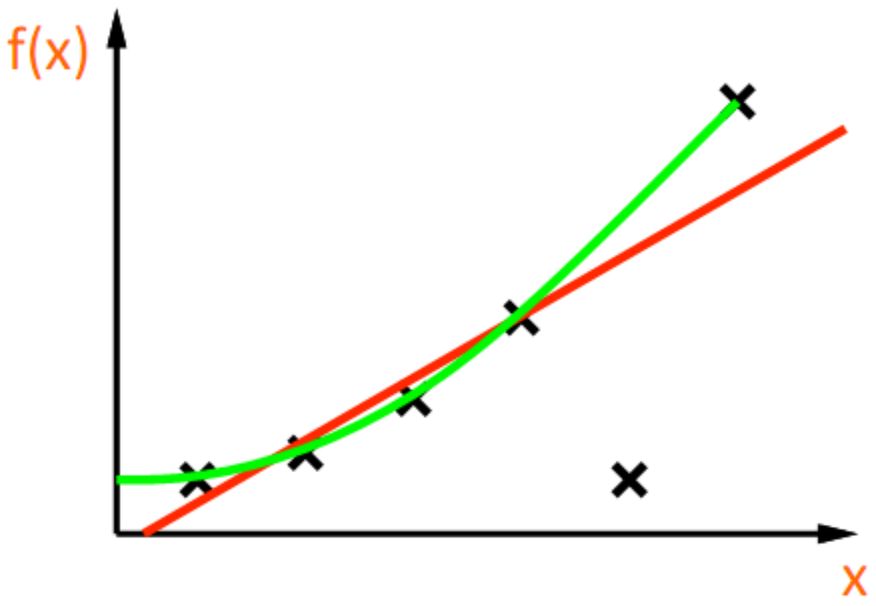
Overfitting



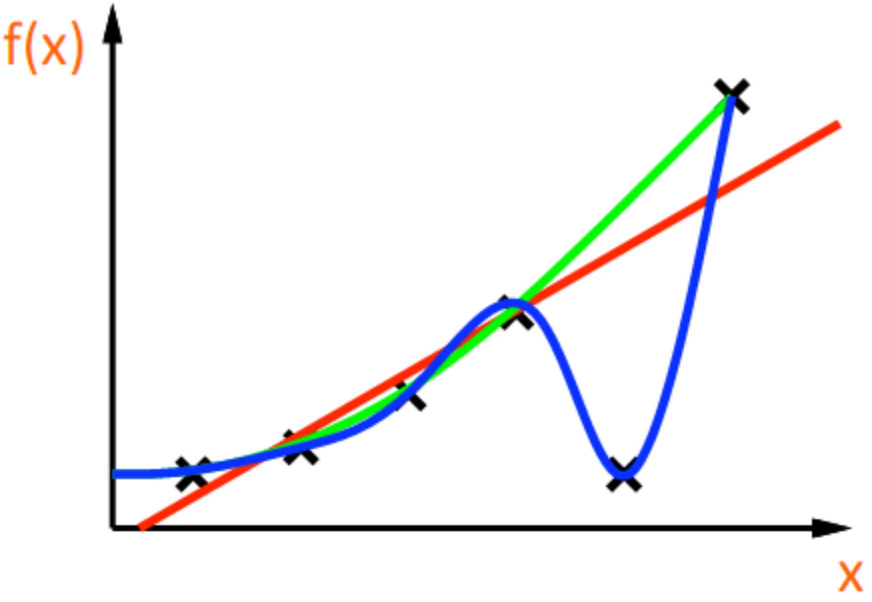
Overfitting



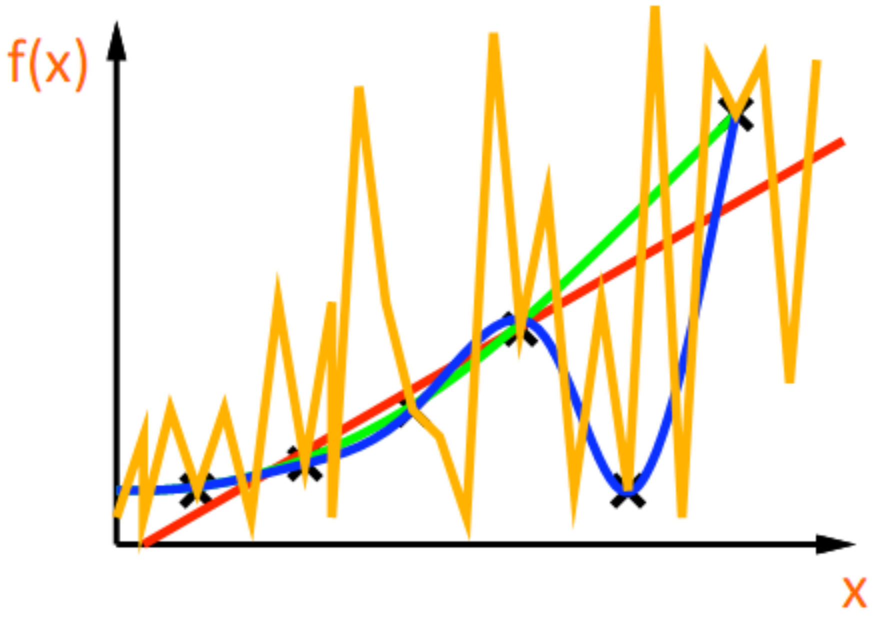
Overfitting



Overfitting



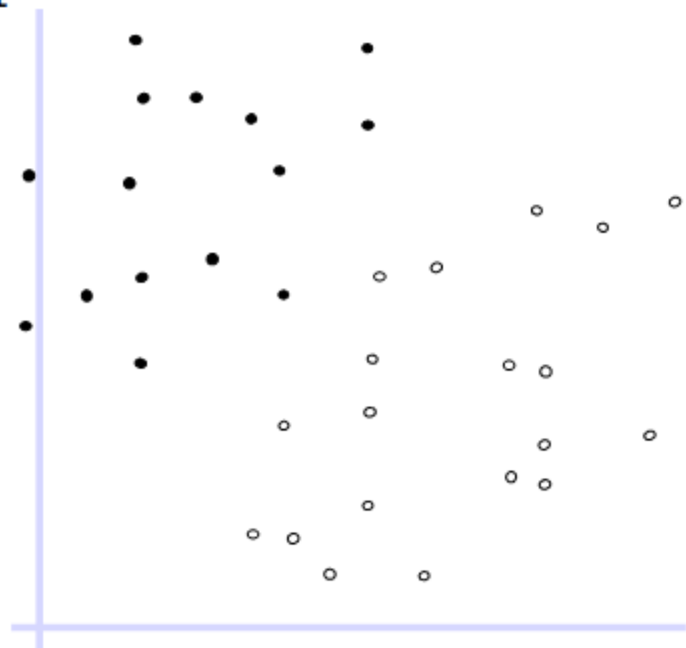
Overfitting



Which curve is desired ?

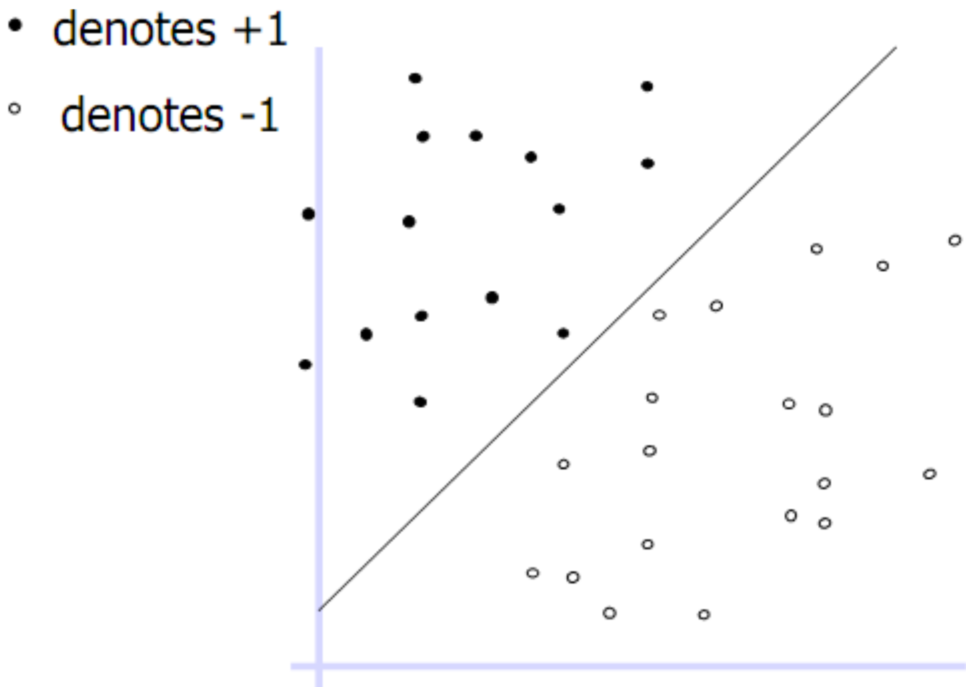
Linear Classifiers

- denotes +1
- ° denotes -1



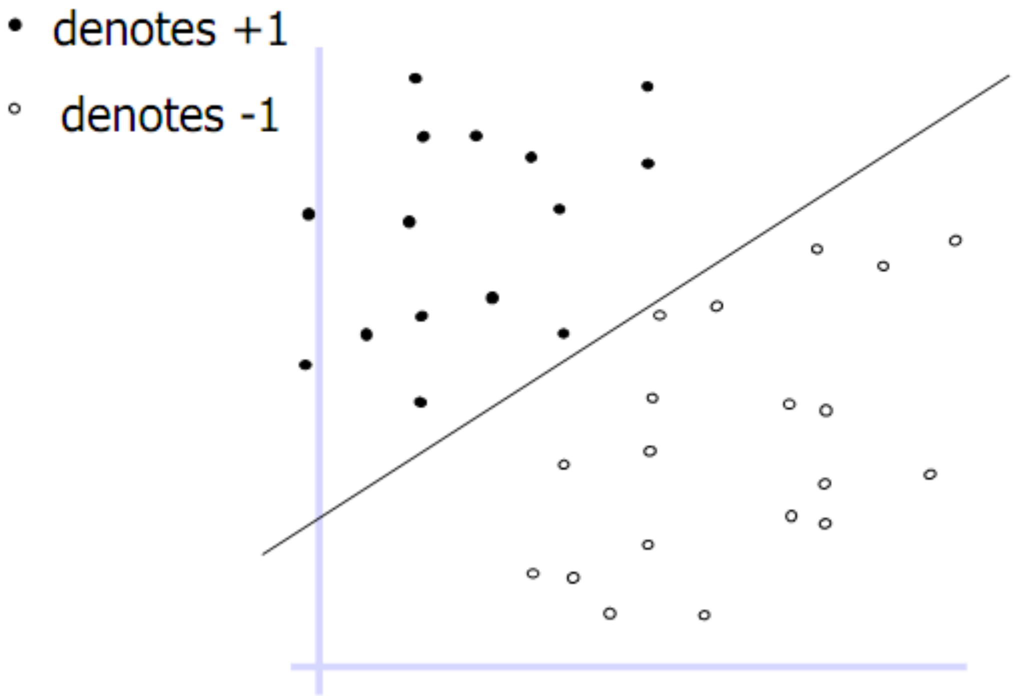
How would you classify this data?

Linear Classifiers



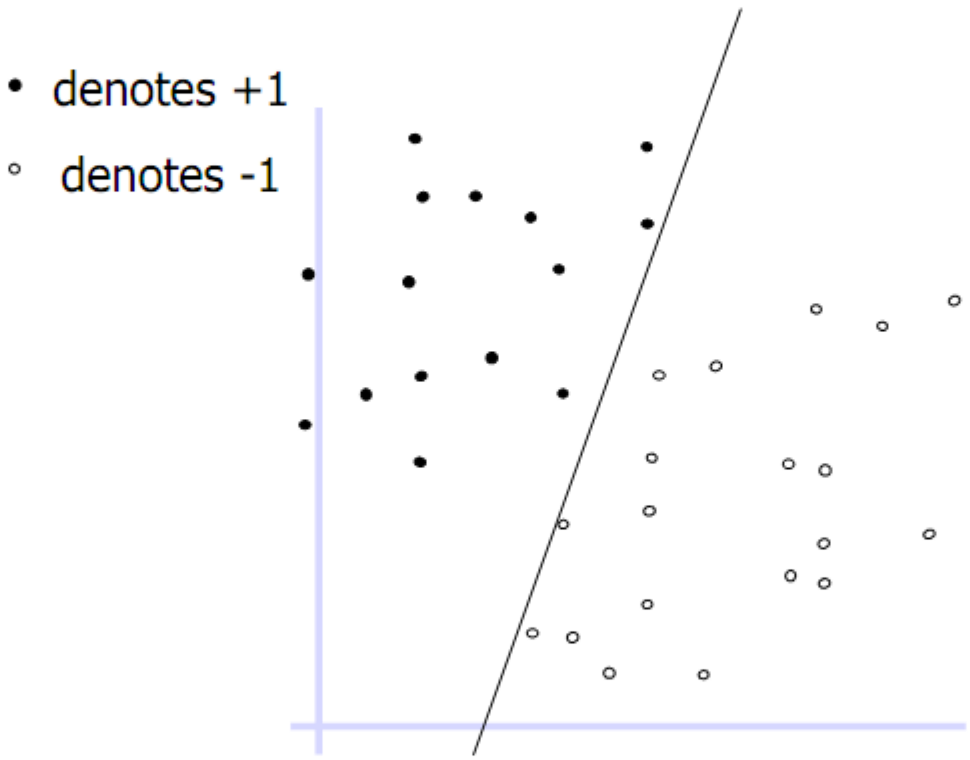
How would you classify this data?

Linear Classifiers



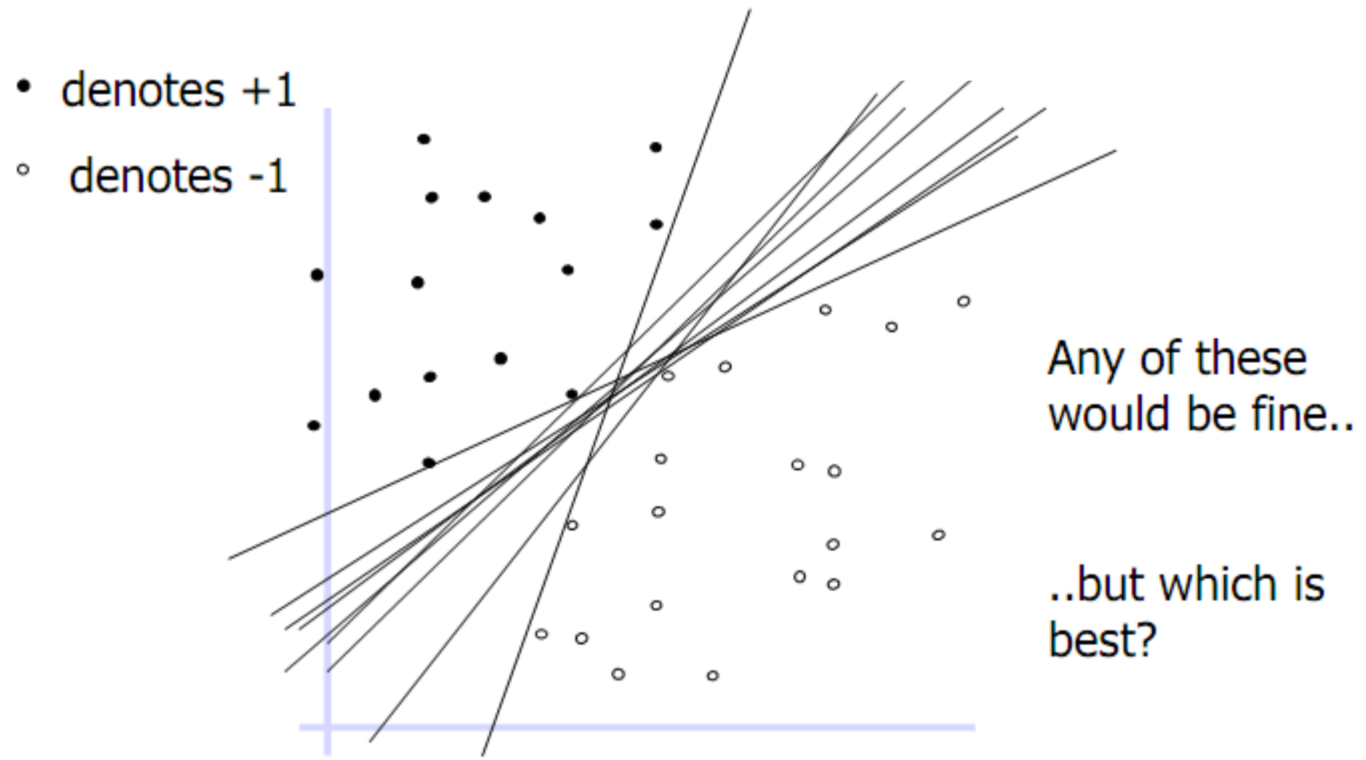
How would you classify this data?

Linear Classifiers



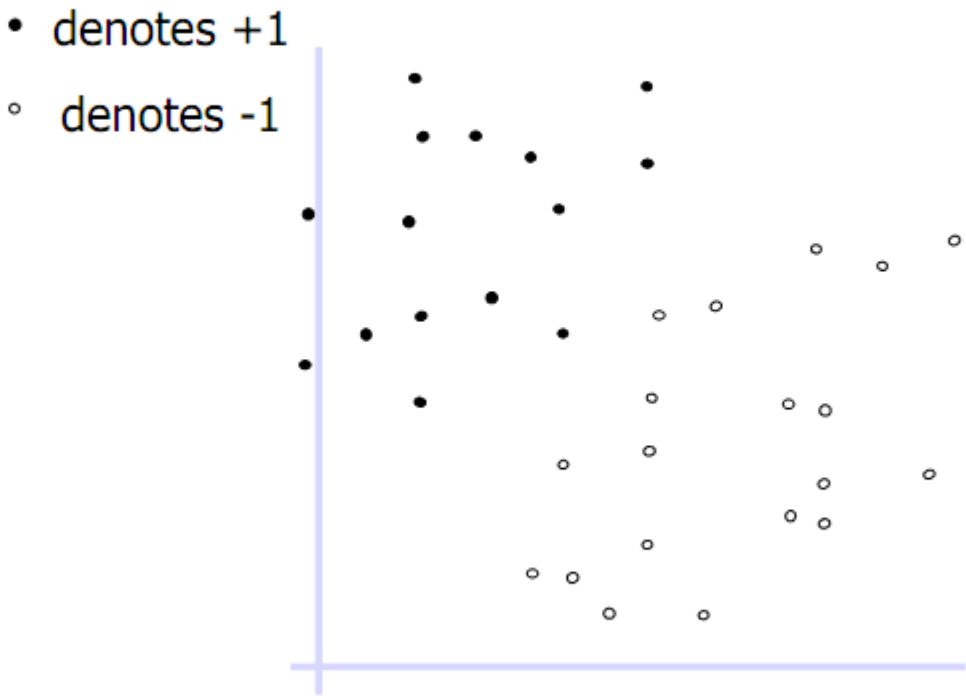
How would you classify this data?

Linear Classifiers



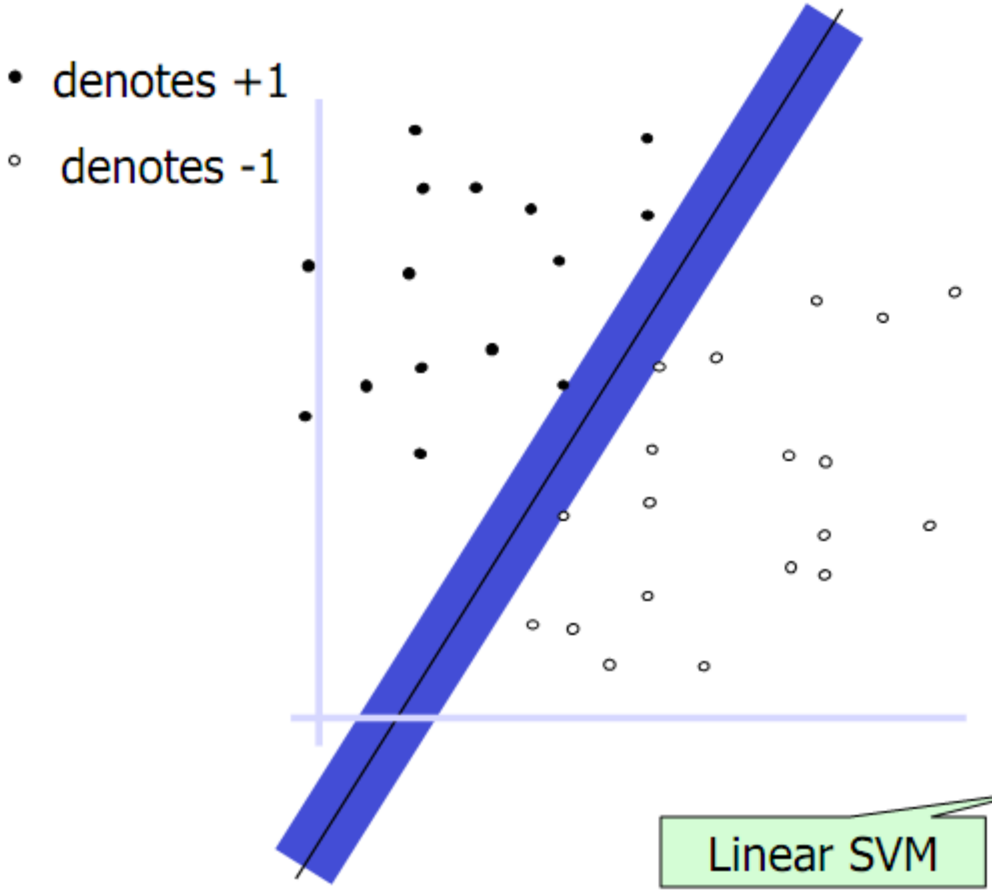
- Essentially, optimization

About ‘Margin’



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

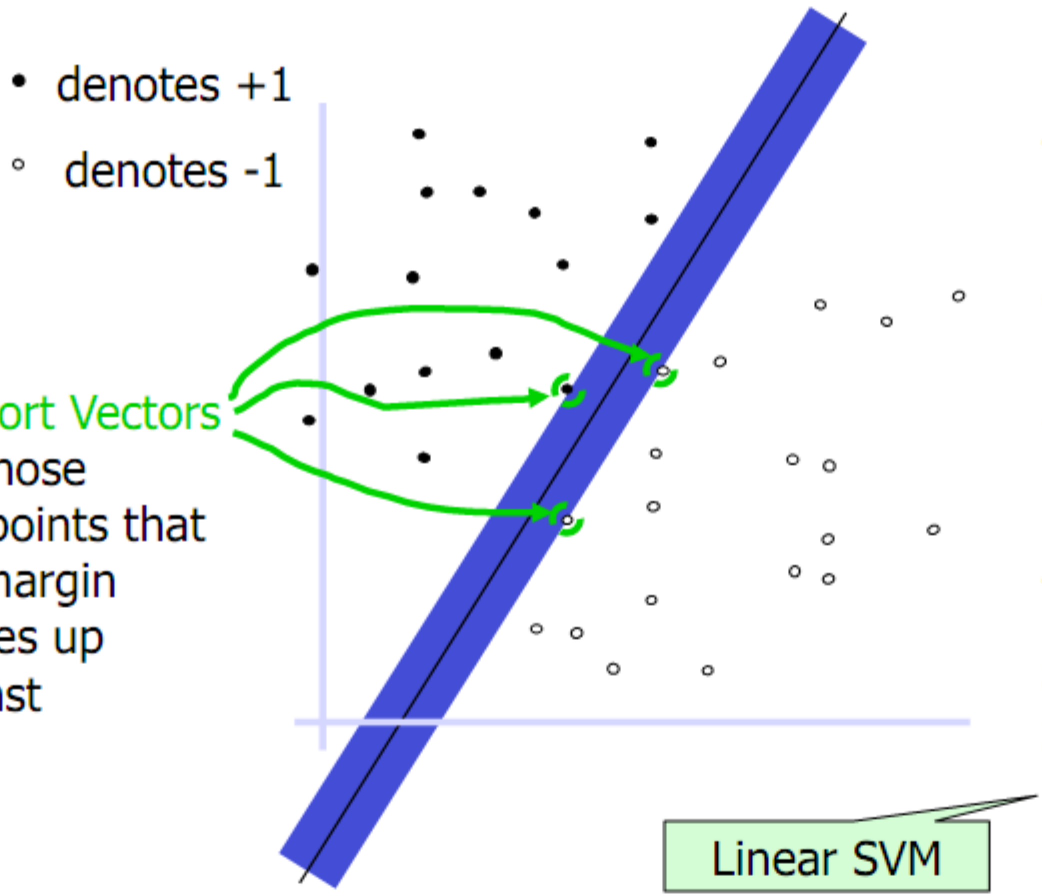
Maximum Margin



The maximum margin linear classifier is the linear classifier with maximum margin.

This is the simplest kind of SVM (Called an LSVM)

… and Support Vectors

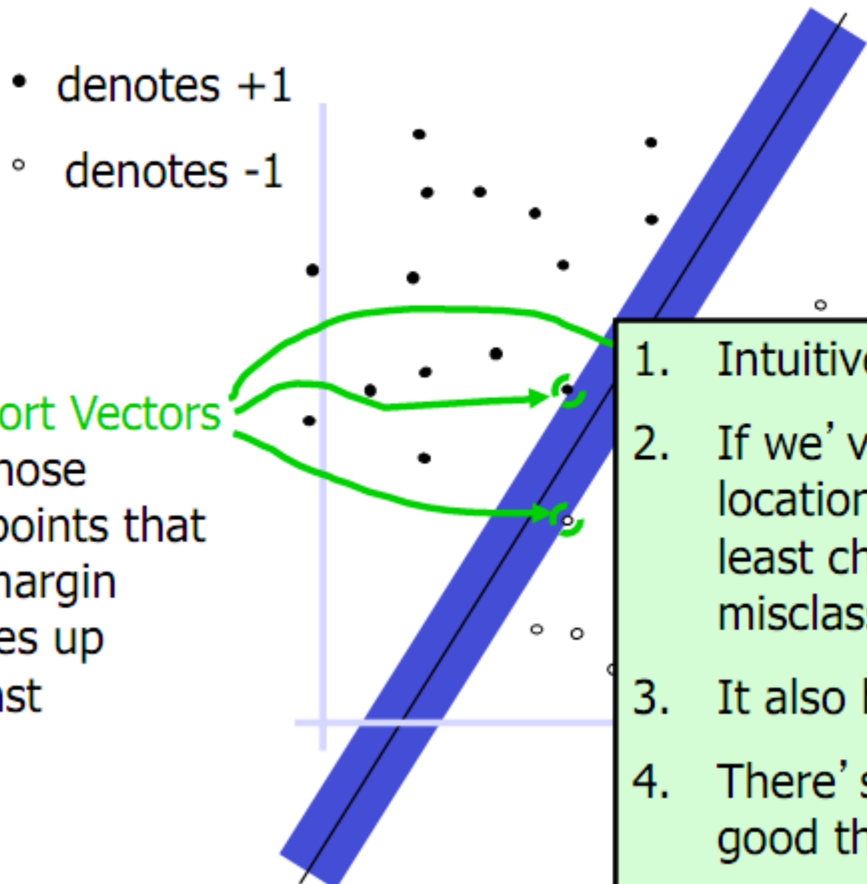


Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier with maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Why Maximum Margin?

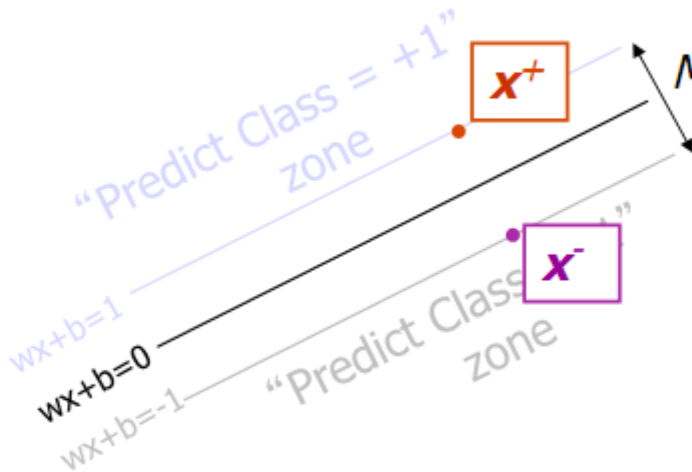


Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the

1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary this gives us least chance of causing a misclassification.
3. It also helps generalization
4. There's some theory that this is a good thing.
5. Empirically it works very very well.

Obtain the Maximum Margin Classifier



M = Margin Width = $\frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

Minimize it!

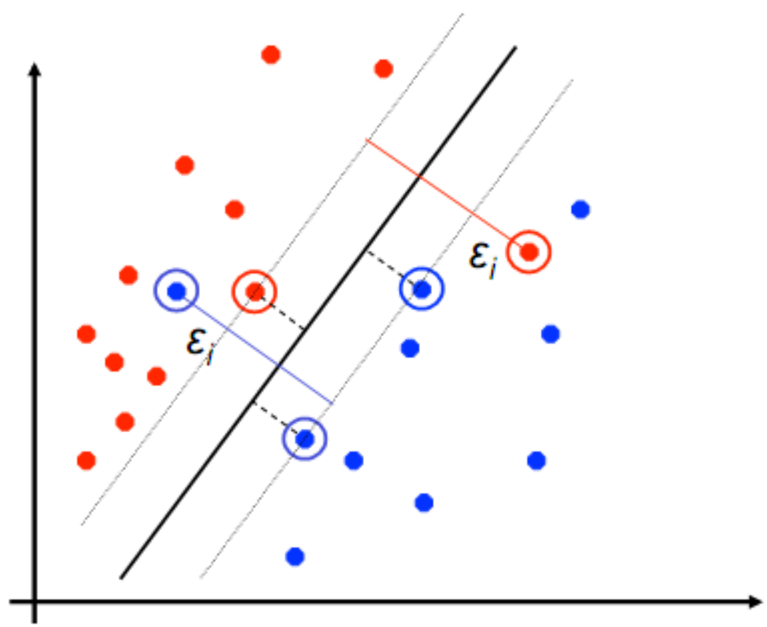
Given a guess of ***w*** and *b* we can

- Compute whether all data points in the correct half-planes
- Compute the width of the margin

So now we just need to write a program to search the space of **w**' s and *b*' s to find the widest margin that matches all the data points. *How?* Quadratic Programming (QP)

Soft Margin

- What if the training set is not linearly separable?
- *Slack variables* ε_i can be added to allow misclassification of difficult or noisy examples, resulting so-called *soft margin.*



Soft Margin Classification Mathematically

- The old formulation:

Find **w** and b such that
$\boldsymbol{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

Find **w** and b such that
$\boldsymbol{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$, $\xi_i \geq 0$

- Parameter C can be viewed as a way to control overfitting: it "trades off" the relative importance of maximizing the margin and fitting the training data.

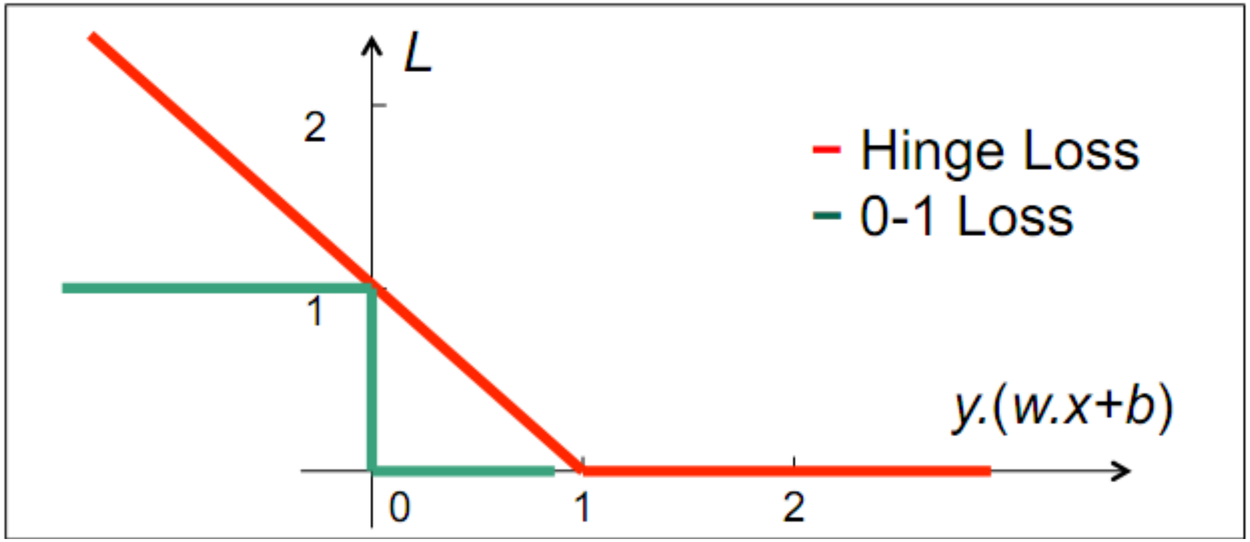
From Another View…

- The soft margin SVM is equivalent to applying a hinge loss

$$L(w, b) := \sum_{i=1}^{n} \max(1 - y_i(w^T x_i + b), 0)$$

- Equivalent unconstrained optimization formulation

$$\min_{\{\mathbf{w},b\}} L(\mathbf{w},b)+\lambda||\mathbf{w}||^2 \qquad \lambda=0.5/C$$



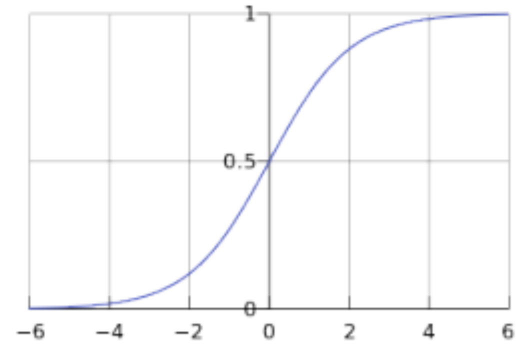
Logistic Regression (LR)

- Binary response: Y ={+1, -1}

$$Y_i|X_i \sim \text{Bernoulli}(p_i)$$

where p_i is the probability of Y_i=1

$$p_i = \frac{1}{1+\exp(-W^T X_i)}$$



- Likelihood

$$\prod_{i=1}^{n} P(Y_i|X_i) = \prod_{i=1}^{n} \left(\frac{1}{1+\exp(-Y_i X_i^T W)}\right)$$

Logistic Regression (LR)

- Maximum likelihood estimator (MLE) becomes logistic regression

$$\min_W \sum_{i=1}^{n} - \ln p(Y_i|X_i) = \sum_{i=1}^{n} \ln(1 + \exp(Y_i X_i^T W))$$

- Convex optimization problem in terms of W

- MAP is regularized logistic regression

$$\min_W \sum_{i=1}^{n} \ln(1 + \exp(Y_i X_i^T W)) + \lambda \|W\|^2$$

General Formulation of Linear Classifiers

$$\min_{\{\mathbf{w},b\}} L(\mathbf{w},b)+\lambda||\mathbf{w}||^2$$

- The objective: empirical loss + regularization
- The regularization term is usually *L2* norm, but also often *L1* norm for sparse models
- The empirical loss can be hinge loss, logistic loss, smooth hinge loss, … or your own invention

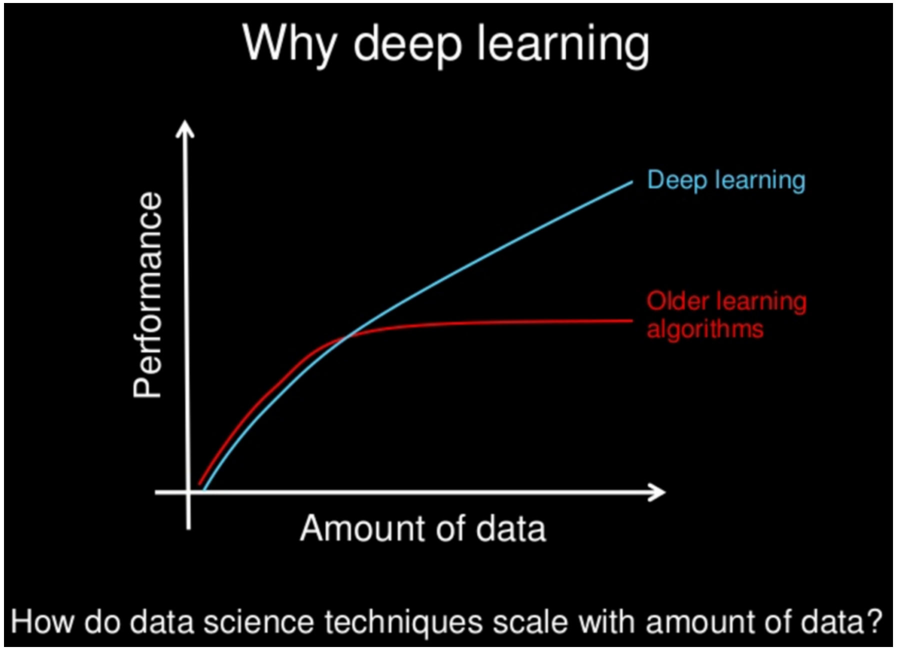
Summary so far…

- ML applications
- ML definition
- ML paradigms
  - Supervised learning
    - Linear classifiers
      - SVM
      - LR
      - General formulation: $\min_{\{\mathbf{w},b\}} L(\mathbf{w},b)+\lambda||\mathbf{w}||^2$
  - Unsupervised learning

Neural Network & Deep Learning

- "**Deep learning** is the application to learning tasks of artificial **neural networks** (ANNs) that contain more than one hidden layers…
- … is part of a broader family of machine learning methods based on learning data representations, as opposed to task specific algorithms …"

WIKIPEDIA
The Free Encyclopedia



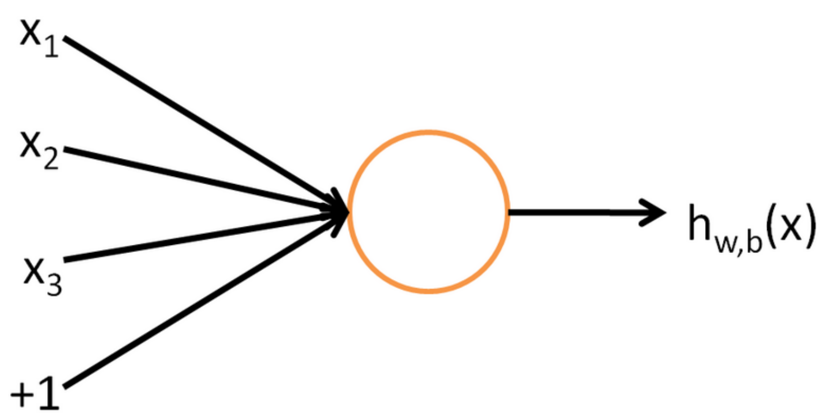
Deep learning algorithms often perform better with more data.
Stronger computing power (e.g. GPU, cloud computing) also matters.

Neural Networks (NN)

- Consider a supervised learning problem
  - Labeled training examples $(x^{(i)},y^{(i)})$

- Neural networks give a way of defining a complex, non-linear form of hypotheses $h_{W,b}(x)$, with parameters W, b that we can fit to our data.

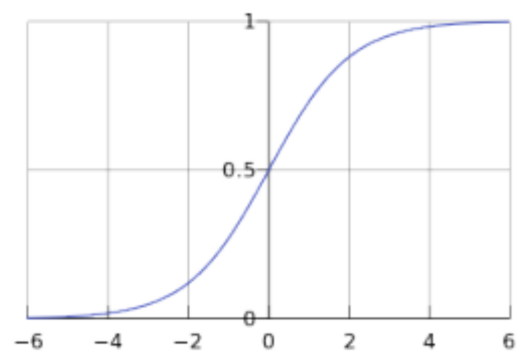
Start from a Single Neuron…



$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$$

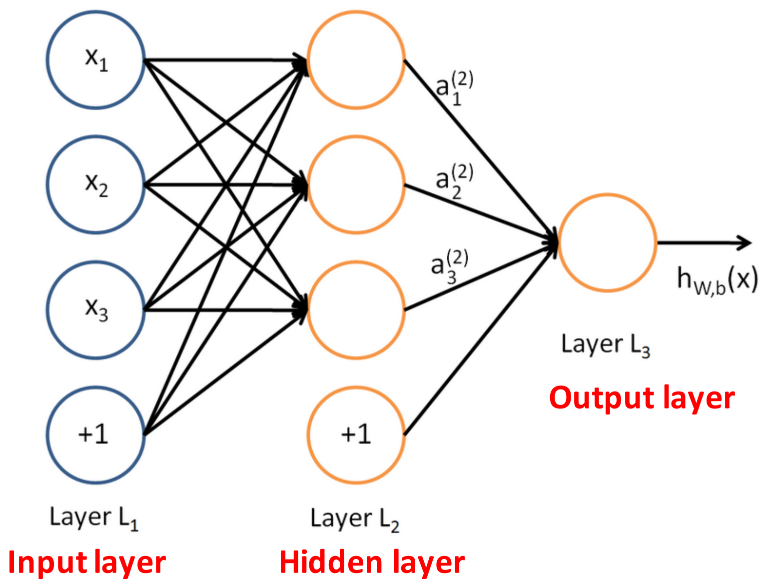
- *f*: **activation function**
  - e.g. the sigmoid function:

$$f(z) = \frac{1}{1 + \exp(-z)}.$$



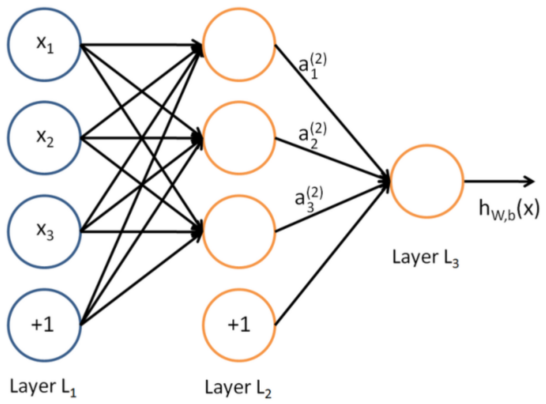
Feedforward Neural Network

- Aka. Multi-Layer Perceptron (MLP)
- Put together by hooking together many simple “neurons”
- Output of a neuron can be input of another



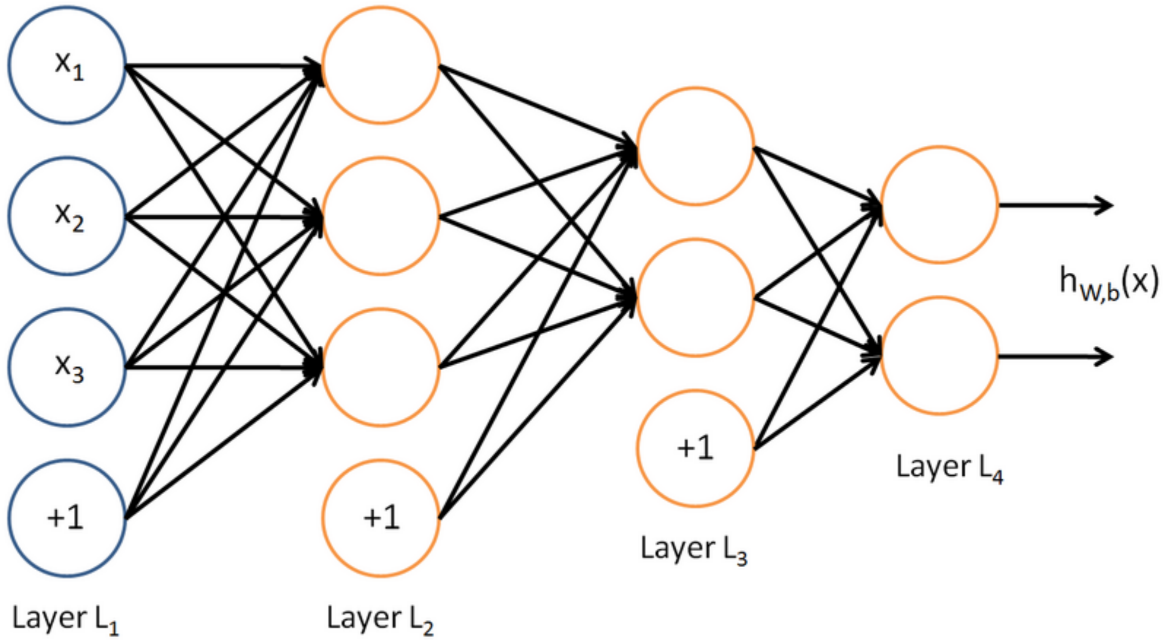
Forward Propagation

- Let $a_i^{(l)}$ denote the **activation** (output value) of unit *i* in layer *l*



$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

Can also have multiple output units



- OK, well… so how to train NN?

First, define cost function

- Given a fixed training dataset

$$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$$

- Cost function with respect to a single example:

$$J(W,b;x,y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2.$$

- Overall cost function (to be minimized):

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m} \sum_{i=1}^m J(W,b;x^{(i)},y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W^{(l)}_{ji} \right)^2 \\ &= \left[\frac{1}{m} \sum_{i=1}^m \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right] + \underbrace{\frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W^{(l)}_{ji} \right)^2}_{\text{weight decay}} \end{aligned}$$

Then, minimize it

- Batch gradient descent update

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$
$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b)$$

  - α: learning rate
  - Iteratively search for better *W* and *b* until a stopping condition is met
  - Susceptible to local optima; however, in practice it usually works fairly well

- How to efficiently compute these partial derivatives?
  - Backpropagation (BP) algorithm

Backpropagation (BP) Algorithm

- In one iteration…

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}.
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \; \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

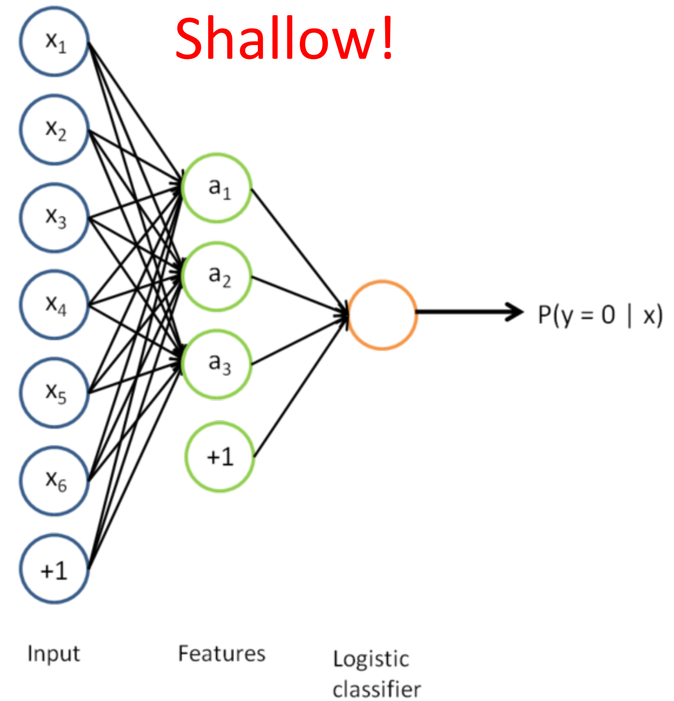
4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$

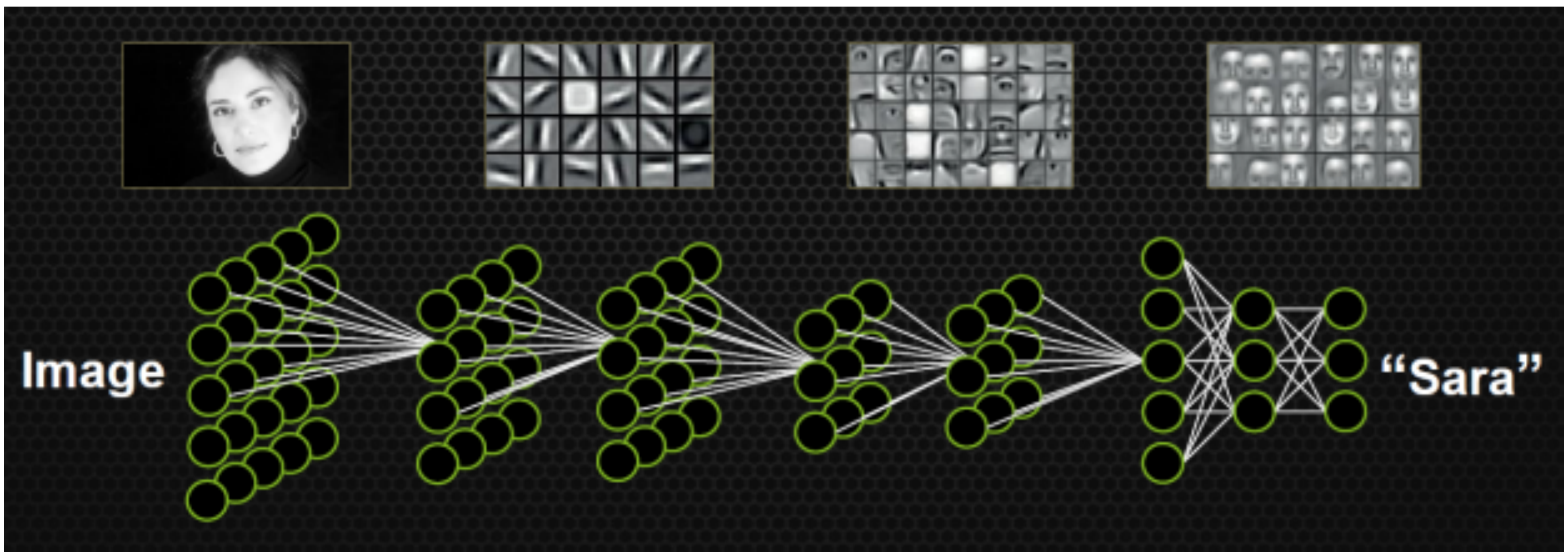
Deep Network

- **Deep** = Multiple hidden layers
- A deep network can have significantly greater representational power than a shallow one
  - can learn significantly more complex functions



In the case of images…

- Deep networks can learn part-whole decompositions



Popular Deep Neural Nets

- Autoencoder (AE)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Generative Adversarial Network (GAN)

Popular Deep Neural Nets

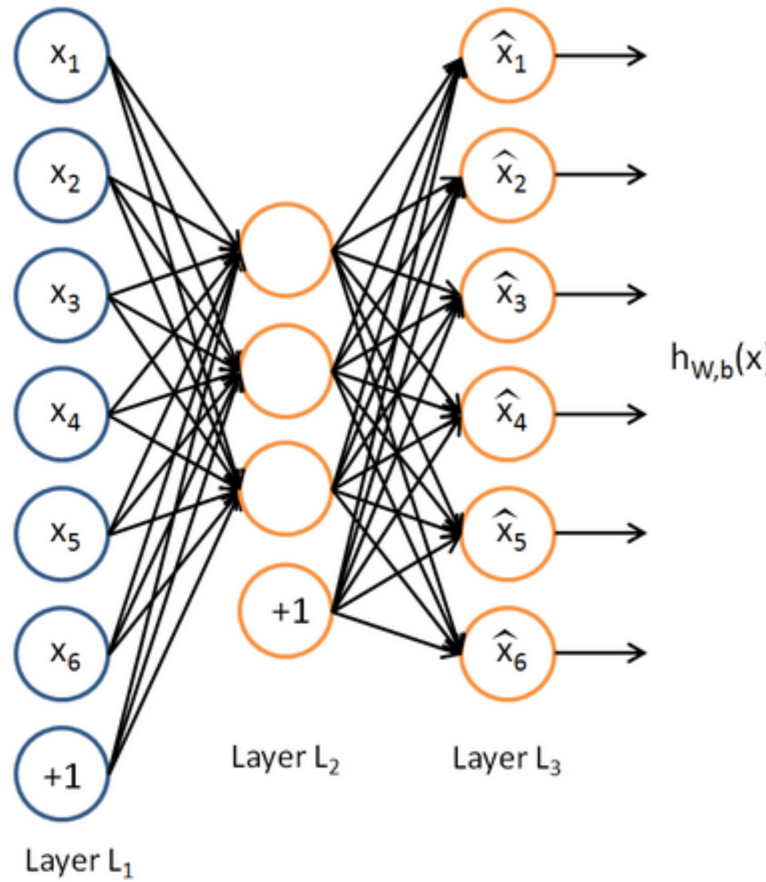
- Autoencoder (AE)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Generative Adversarial Network (GAN)

Autoencoder

- **Unsupervised** learning
  - try to learn a function

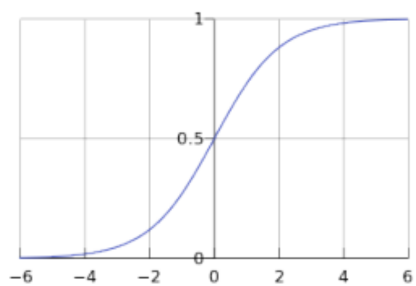
$$h_{W,b}(x) \approx x$$

- By placing **sparsity** constraints on the network, we can discover interesting **structure** about the data
  - E.g. learn a *compressed* representation of input data



Sparsity Constraint

- Think of a neuron (assuming a sigmoid activation function)
  - as being "active" if its output value is close to 1
  - as being "inactive" if its output value is close to 0



- We'd like to constrain neurons to be **inactive** most of the time
  - Activation becomes **sparse** in the network

Mathematically

- Average activation of hidden unit *j*
  - averaged over the training set

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^{(2)}(x^{(i)})\right]$$

- Approximately enforce the constraint

$$\hat{\rho}_j = \rho,$$

  - *ρ* is a **sparsity parameter**, typically a small value close to zero (say 0.05)
  - To satisfy this constraint, hidden unit's activations must mostly be near zero

Define a penalty term

- A typical form:

$$\sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}.$$

  – summing over the hidden units in network

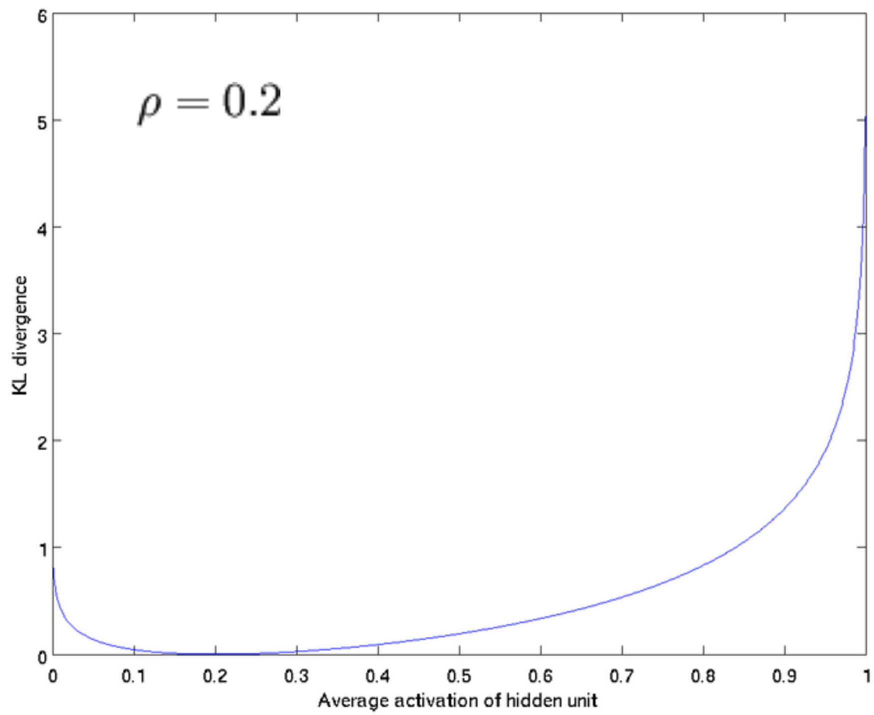
- It is the Kullback-Leibler (KL) divergence between a Bernoulli random variable with mean ρ and a Bernoulli random variable with mean $\hat{\rho}_j$

$$\sum_{j=1}^{s_2} \mathrm{KL}(\rho || \hat{\rho}_j),$$

$$\mathrm{KL}(\rho || \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

Property of KL divergence

$$\mathrm{KL}(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho)\log \frac{1-\rho}{1-\hat{\rho}_j}$$



Cost Function of Autoencoder

- The overall cost function with adding the sparsity penalty:

$$J_{\text{sparse}}(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} \text{KL}(\rho \| \hat{\rho}_j),$$

  - β controls weight of sparsity penalty

- To incorporate the KL-divergence term into derivative calculation in BP:
  - Replace $\delta_i^{(2)} = \left(\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} \right) f'(z_i^{(2)}),$

    with $\delta_i^{(2)} = \left(\left(\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} \right) + \beta \left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i} \right) \right) f'(z_i^{(2)}).$

- Then, we can use BP to train an autoencoder

What does Autoencoder learn?

- Compressed representation (or say, features)
  - Outputs of hidden units

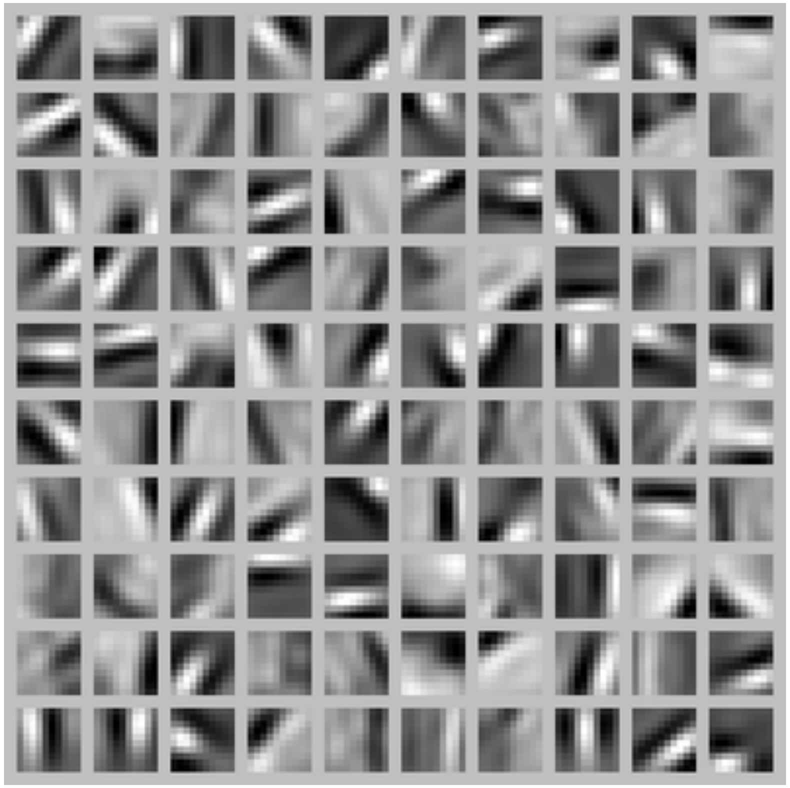
$$a_i^{(2)} = f\left(\sum_{j=1}^{100} W_{ij}^{(1)} x_j + b_i^{(1)}\right).$$

- What does it look like? Any intuitive interpretations?
  - Suppose input training data are 10x10 (n=100) images
  - The image would cause $a_i^{(2)}$ to be maximally activated:
    - j=1,…,100

$$x_j = \frac{W_{ij}^{(1)}}{\sqrt{\sum_{j=1}^{100} (W_{ij}^{(1)})^2}}.$$

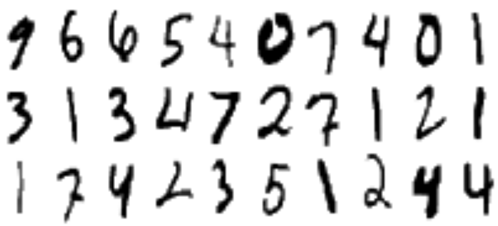
Visually…

- Suppose an autoencoder with 100 hidden units
  - One image per hidden unit
- Different hidden units have learned to **detect edges**
  - at different positions and orientations in the image
- These features are useful for object recognition and other vision tasks

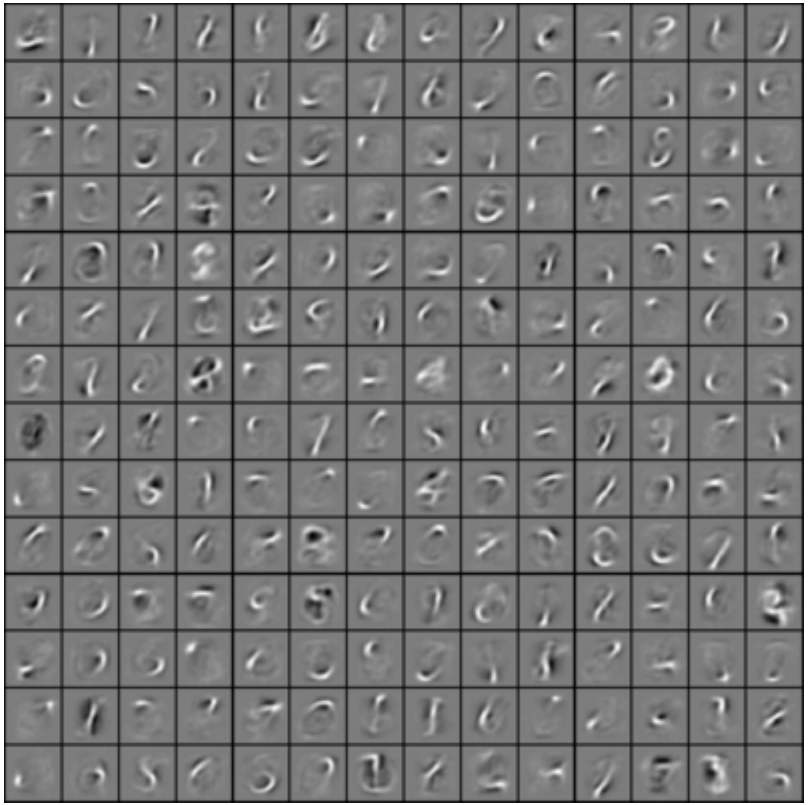


Handwritten Digit Recognition

- On MNIST dataset

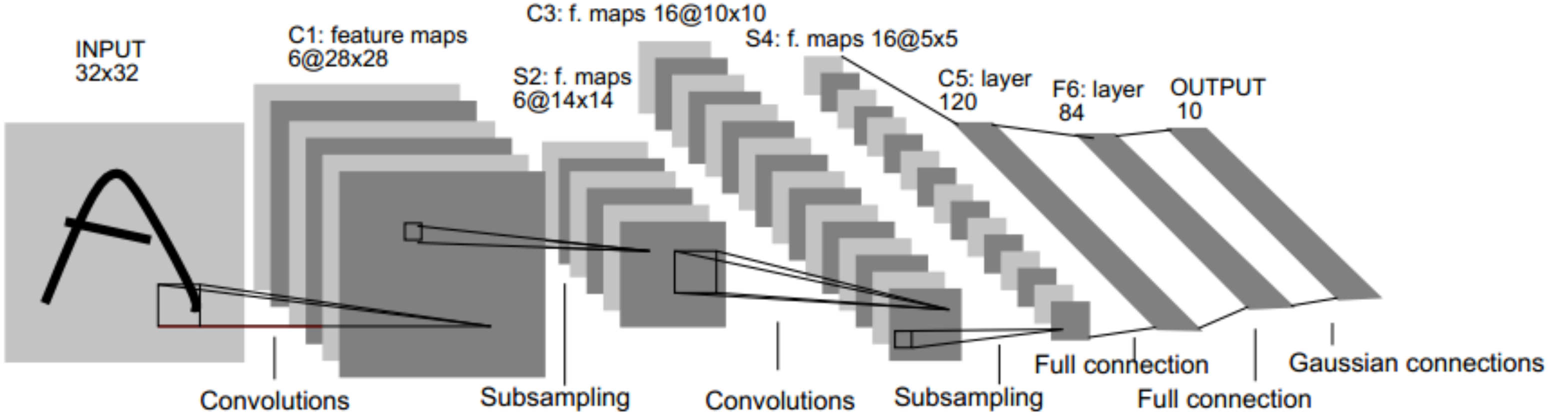


- Features learned by autoencoder
  - **Pen strokes**



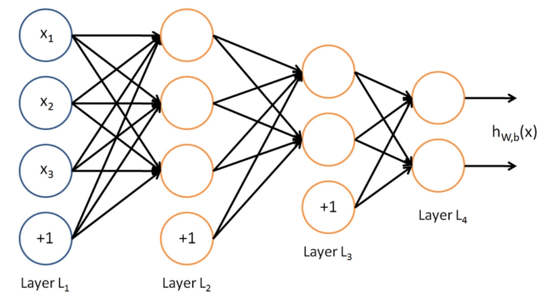
Convolutional Neural Network (CNN)

- **Supervised** learning
  - Typically for image classification tasks
  - Output is a softmax: Generalization of LR for multi-class problems
- LeNet (LeCun et al. 1998)
  - Sparse, **convolutional** layers and **max-pooling** are at the heart of the LeNet family of models



Fully Connected V.S. Locally Connected

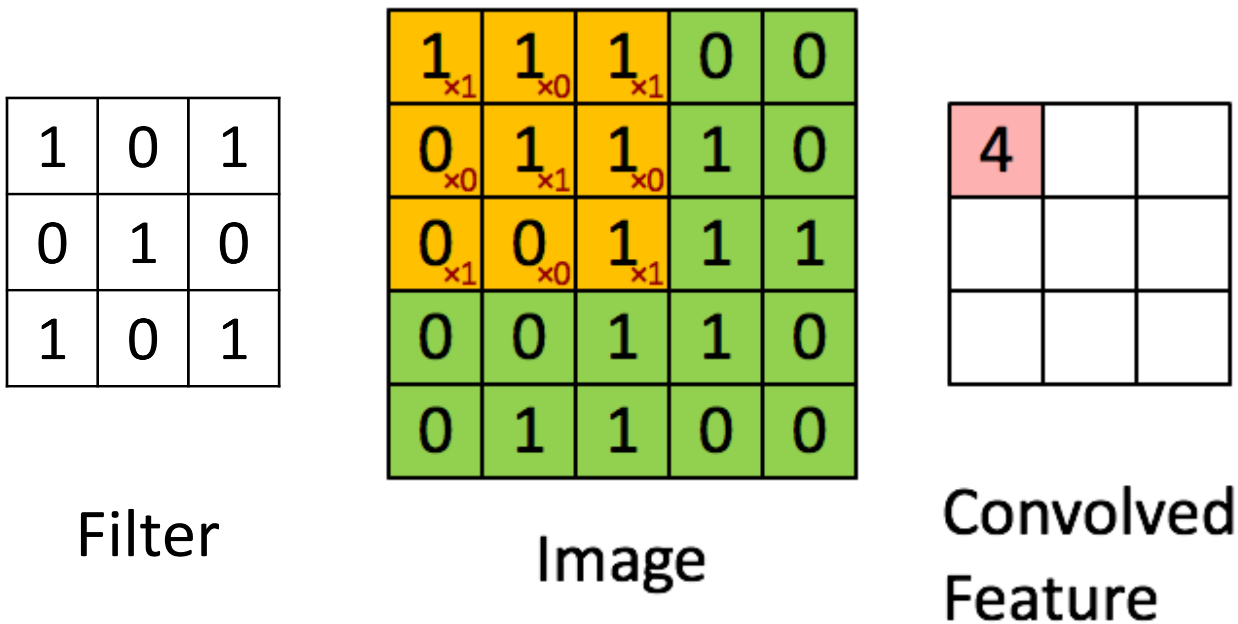
- **Fully connected** networks
  - Traditional NN
    - "fully connect" all hidden units to all input units
  - Computationally expensive
    - Many weights to learn



- **Locally connected** networks
  - Sparse connectivity:
    - Allow each hidden unit to connect to only a small subset of input units
  - Draws inspiration from how the early visual system is wired up in biology
    - neurons in the visual cortex have **localized receptive fields** (i.e., they respond only to stimuli in a certain location)

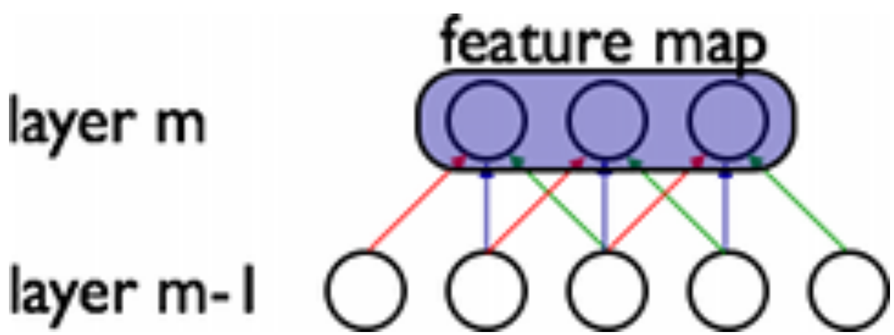
Feature Extraction using Convolution: A Simple Example

- Apply a 3x3 feature detector (or say, filter, convolution kernel) anywhere in input image
  - 3x3: Receptive field size (i.e. filter shape)
- **Convolve** the filter with the larger image to obtain a different feature activation value at each location in the image



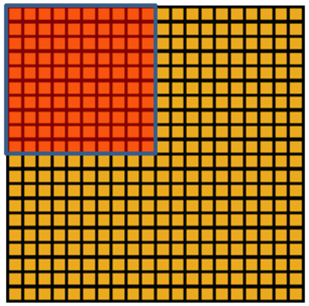
Identical to "Shared Weights"

- Each filter is replicated across the entire visual field
  - They share the same weight vector, and form a *feature map*
  - Weights of the same color are shared—constrained to be identical
  - Locally connected

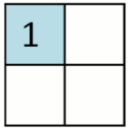


Pooling

- Convolved features could be too many
  - High dimension makes learning hard
  - Prone to overfitting
- Pooling
  - Sub-sampling
  - Dimension reduction
  - Translation invariant
  - Max/mean/random pooling



Convolved feature



Pooled feature

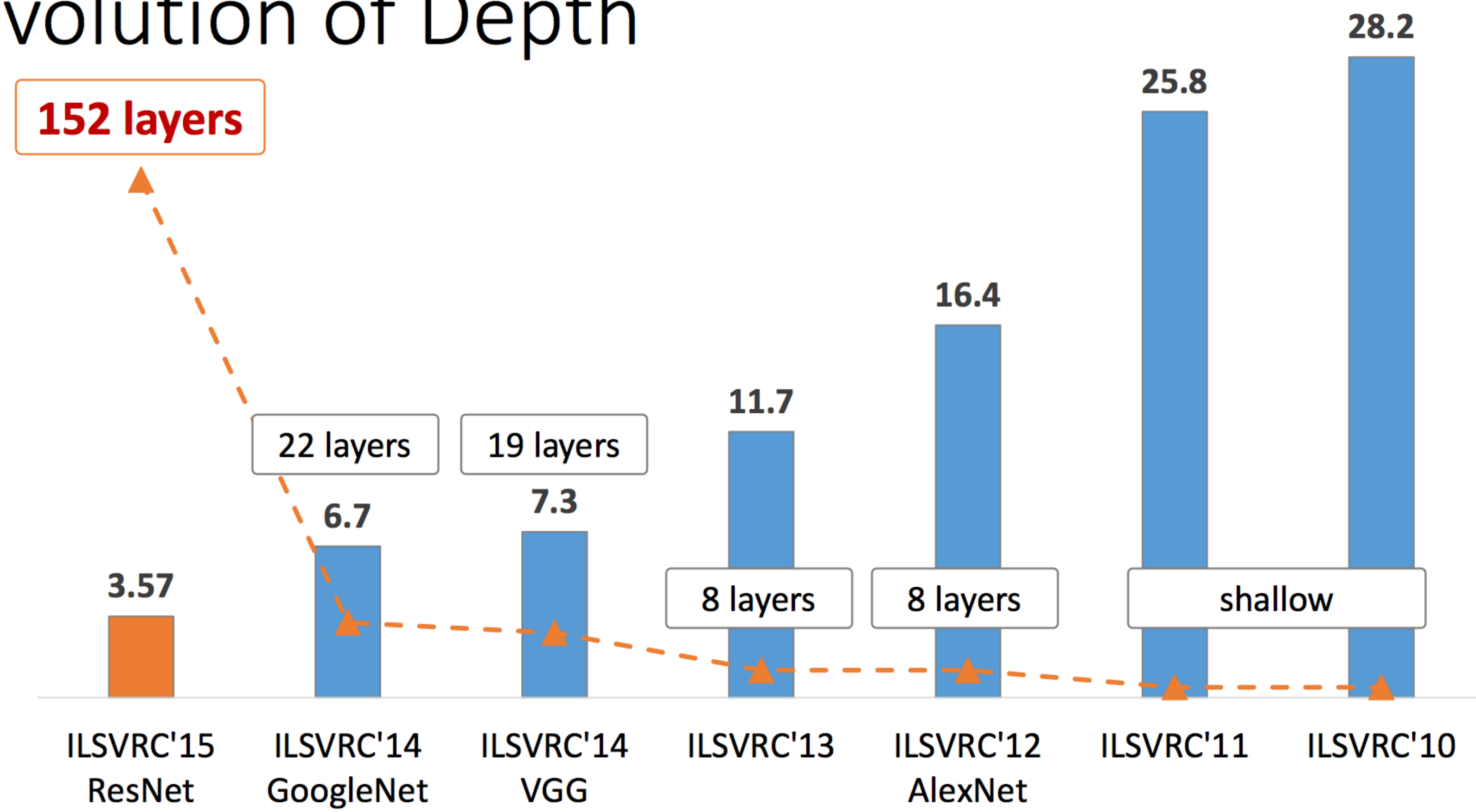
Some Classical CNNs

- AlexNet (Krizhevsky et al. 2012)
  - Winner of ImageNet LSVRC 2012, top-5 error 15.3%
  - 8 layers: 5 conv (2 with LRN, 3 with pooling) + 3 FC (2 with dropout) layers

- GoogLeNet (Szegedy et al. 2015)
  - Winner of ImageNet LSVRC 2014, top-5 error 6.67%
  - 22 layers

- ResNet (He et al. 2016)
  - Winner of ImageNet ILSVRC & COCO 2015, top-5 error 3.57%
  - "Ultra-deep", 152 layers

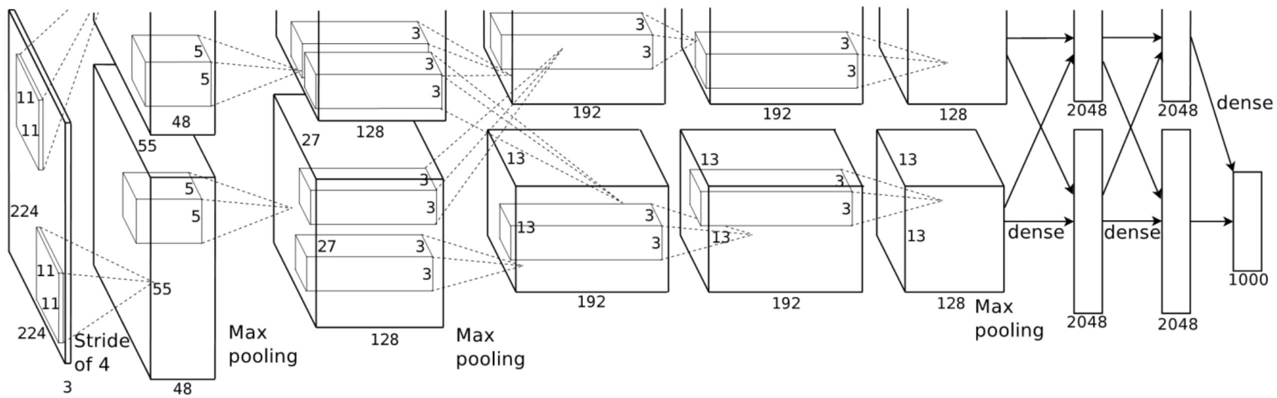
Revolution of Depth

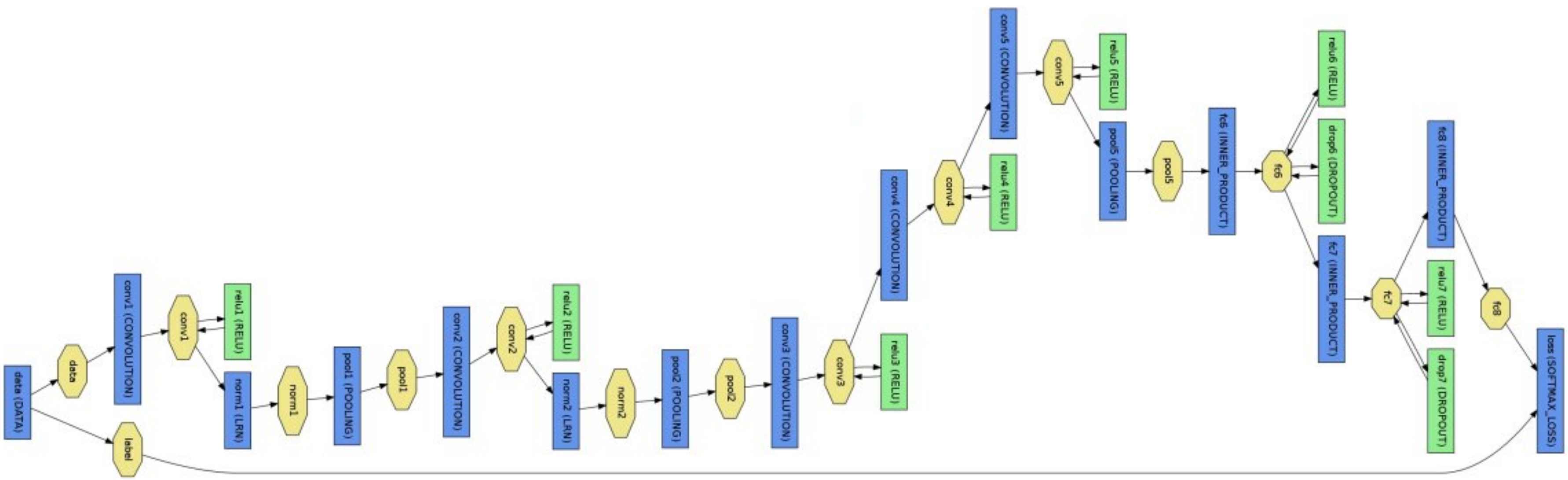


ImageNet Classification top-5 error (%)

Figure source: Kaiming He

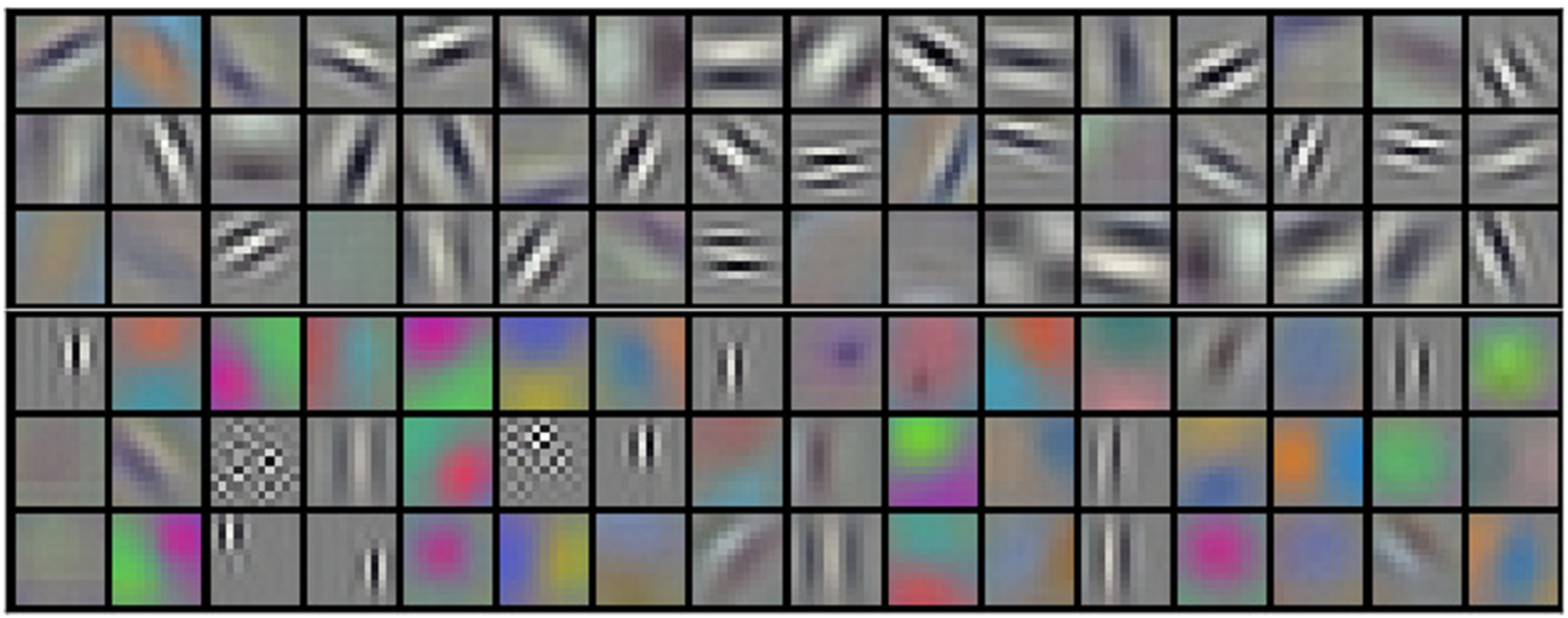
AlexNet Architecture

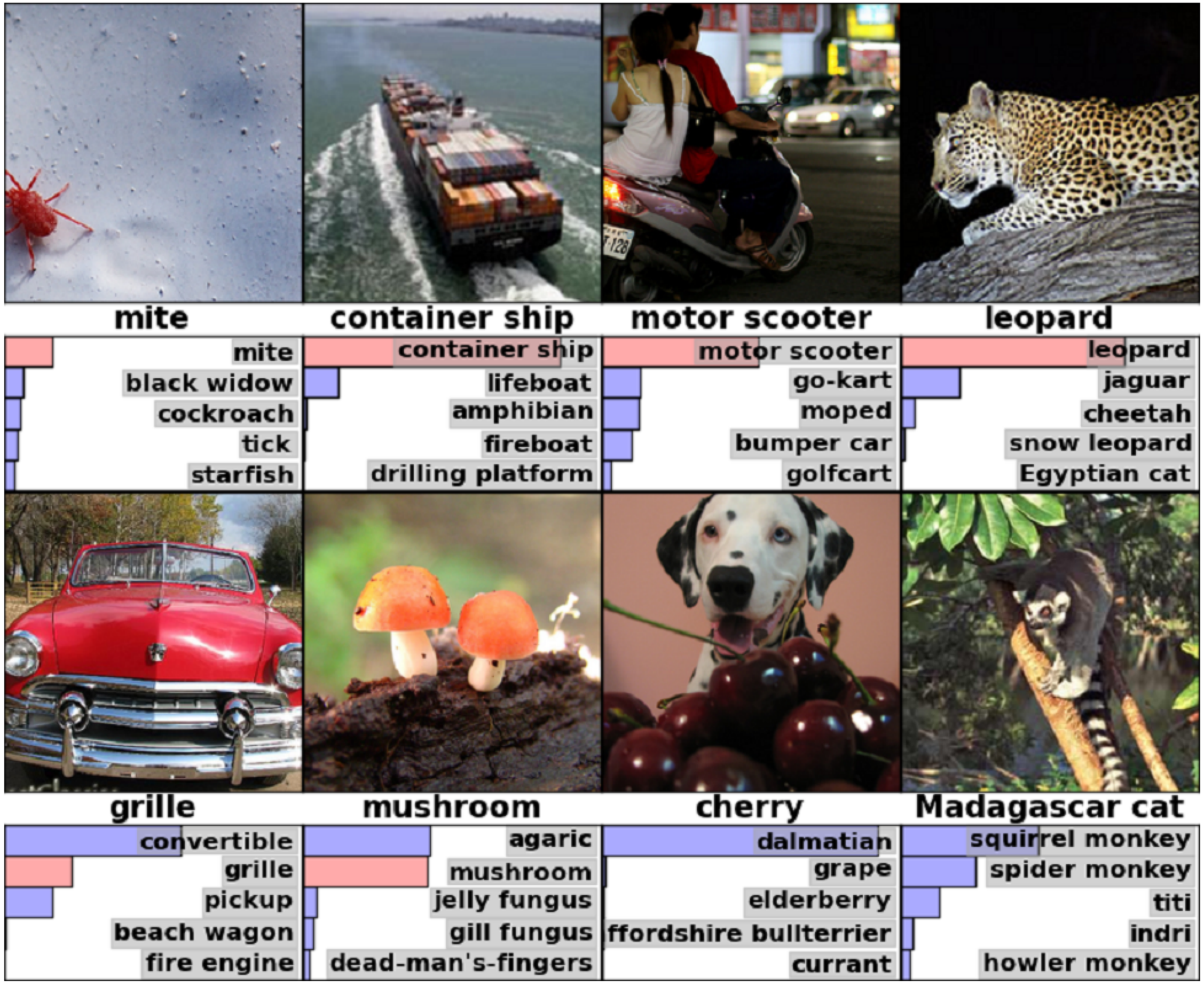




Features Learned

- … by the first convolutional layer




mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

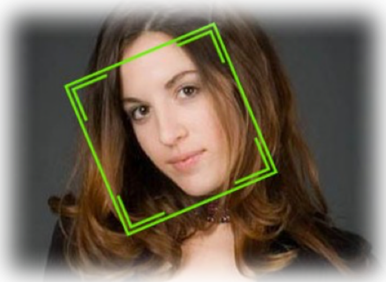
The last 4096d activations…

- Can be used for image retrieval

Many tricks & details!

- Initialization of weights
- Selection of activation functions
- #layers
- #feature maps
- Filter size & shape, pooling shape
- Normalization
- Learning rate
- Data augmentation
- Parameter tuning
- …
- More like **black box** than traditional ML methods!

Deep Learning is now leading the “AI Revolution”



Face Recognition

Question Answering

Self-Driving Car

Voice Recognition

Game

Healthcare

Summary

- From a single neuron to feedforward NN (MLP)
  - BP algorithm
  - Deep vs. Shallow
- Popular deep nets
  - Autoencoder
  - CNN

- Deep learning models are being used for very difficult problems and making progress

- Deep learning is hot, it is delivering results and now is the time to get involved!

Thanks!