Machine Learning and Bayesian Nonparametrics

Tamara Broderick · Peter Orbanz

MIT and Columbia University

Overview

OVERVIEW

Morning	Machine Learning
Afternoon	Bayesian Nonparametrics

OVERVIEW

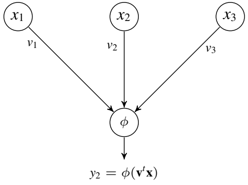
Machine Learning

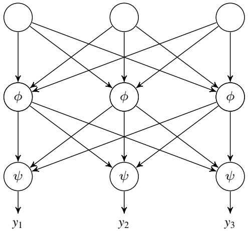
- Many latent variable models (finite mixtures, HMMs, ...)
- Linear classifiers can be regarded as boundary cases of models with hidden variables

Bayesian nonparametrics in Machine Learning

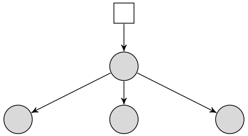
- Useful way to impose structure on latent variables
- BNP can be used as building blocks in hierarchies

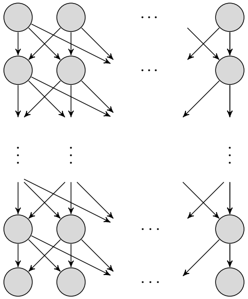
Neural networks





Graphical models





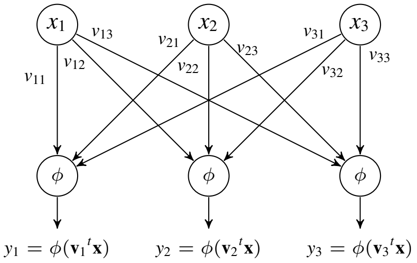
Grouping dependent variables into
layers is *a good thing*.

NEURAL NETWORKS

“Neural networks”

A neural network represents a function $f : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$.

NEURAL NETWORKS

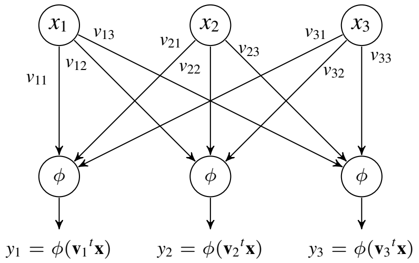


x_1
x_2
x_3
v_{11}
v_{12}
v_{13}
v_{21}
v_{22}
v_{23}
v_{31}
v_{32}
v_{33}
ϕ
ϕ
ϕ
$y_1 = \phi(\mathbf{v}_1{}^t\mathbf{x})$
$y_2 = \phi(\mathbf{v}_2{}^t\mathbf{x})$
$y_3 = \phi(\mathbf{v}_3{}^t\mathbf{x})$

$$y_i = \phi\Big(\sum_j v_{ij}x_j\Big)$$

Typically: $\phi = \sigma$ or $\phi(x) = \mathbb{I}\{\pm x > \tau\}$ or $\phi(x) = x$

Neural networks

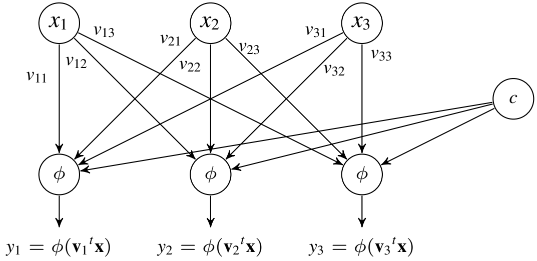


x_1
x_2
x_3
v_{11}
v_{12}
v_{13}
v_{21}
v_{22}
v_{23}
v_{31}
v_{32}
v_{33}
ϕ
ϕ
ϕ
$y_1 = \phi(\mathbf{v}_1{}^t\mathbf{x})$
$y_2 = \phi(\mathbf{v}_2{}^t\mathbf{x})$
$y_3 = \phi(\mathbf{v}_3{}^t\mathbf{x})$

$$y_i = \phi\Big(\sum_j v_{ij}x_j\Big)$$

Typically: $$\phi(x) = \begin{cases} \sigma(x) \text{ (sigmoid)} \\ \mathbb{I}\{\pm x > \tau\} \\ c \text{ (constant)} \\ x \end{cases}$$

Neural networks


x_1
x_2
x_3
c
v_{11}
v_{12}
v_{13}
v_{21}
v_{22}
v_{23}
v_{31}
v_{32}
v_{33}
ϕ
ϕ
ϕ
$y_1 = \phi(\mathbf{v}_1{}^t\mathbf{x})$
$y_2 = \phi(\mathbf{v}_2{}^t\mathbf{x})$
$y_3 = \phi(\mathbf{v}_3{}^t\mathbf{x})$

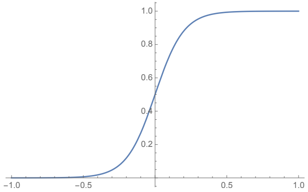
$$y_i = \phi\Big(\sum_j v_{ij}x_j \quad - c\Big)$$

$$\text{Typically:} \qquad \phi(x) = \begin{cases} \sigma(x) \text{ (sigmoid)} \\ \mathbb{I}\{\pm x > \tau\} \\ c \text{ (constant)} \\ x \end{cases}$$

LOGISTIC REGRESSION

Sigmoid function

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



- $\sigma(\theta x)$ is smooth approximation to indicator function $\mathbb{I}\{x \geq 0\}$.
- θ larger $\rightarrow$ closer approximation

Recall: Logistic regression

Covariates $\mathbf{x} \in \mathbb{R}^d$, binary observations y, regressor is logistic function.

$$p(y|\mathbf{x}; \theta) = \text{Bernoulli}(\sigma(\theta^t \mathbf{x}))$$

$\theta \in \mathbb{R}^d$ is parameter to estimate.

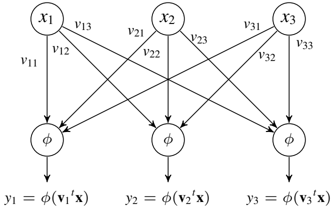
NNs and Graphical Models

Neural networks

- Representation of function using a graph
- Layers:

$$\begin{tikzcd} x \arrow[r] & g \arrow[r] & f \end{tikzcd}$$

"f depends on x only through g"



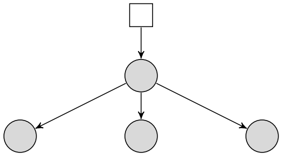
x_1
x_2
x_3
v_{11}
v_{12}
v_{13}
v_{21}
v_{22}
v_{23}
v_{31}
v_{32}
v_{33}
ϕ
ϕ
ϕ
$y_1 = \phi(\mathbf{v}_1{}^t\mathbf{x})$
$y_2 = \phi(\mathbf{v}_2{}^t\mathbf{x})$
$y_3 = \phi(\mathbf{v}_3{}^t\mathbf{x})$

Graphical models

- Representation of a distribution using a graph
- Layers:

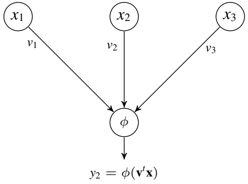
$$\begin{tikzcd} X \arrow[r] & Y \arrow[r] & Z \end{tikzcd}$$

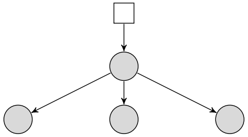
"Z is conditionally independent of X given Y"

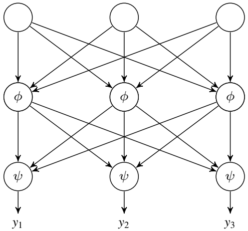


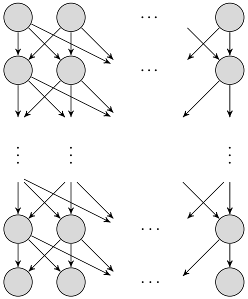
Neural networks

Graphical models

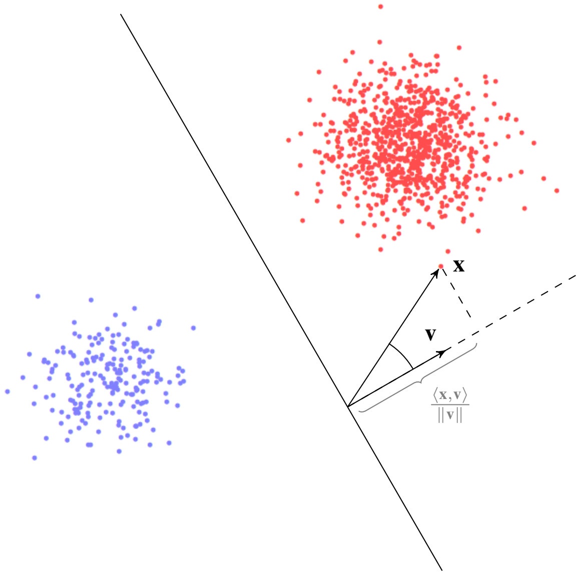








Linear Classification


$\mathbf{x}$
$\mathbf{v}$
$\frac{\langle \mathbf{x}, \mathbf{v} \rangle}{\|\mathbf{v}\|}$

Historical perspective: Perceptron

McCulloch-Pitts neuron model (1943)

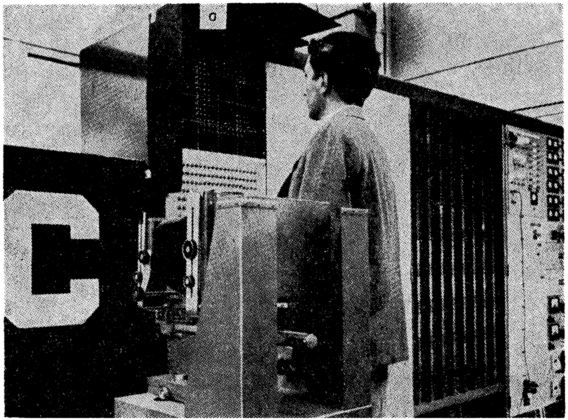
$$y = \mathbb{I}\{\mathbf{v}^t\mathbf{x} > c\} \qquad \text{for some } c \in \mathbb{R} \, .$$

Perceptron (Rosenblatt, 1957)

"Train" McCulloch-Pitts model (that is: estimate $(c, \mathbf{v})$) by applying gradient descent to the function

$$C_p(c, \mathbf{v}) := \sum_{i=1}^{n} \mathbb{I}\{\text{sgn}(\langle \mathbf{v}, \tilde{\mathbf{x}}_i \rangle - c) \neq \tilde{y}_i\} \left| \left\langle \begin{pmatrix} c \\ \mathbf{v} \end{pmatrix}, \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix} \right\rangle \right| \, ,$$

called the **Perceptron cost function**.



Perceptron training

- The neuron model $\mathbb{I}\{\mathbf{v}^t\mathbf{x} > c\}$ is a linear classifier.
- For parameter estimation by optimization, we need an optimization target.
- Choose 0-1 loss as simplest loss for classification.
- Minimize empirical risk (on training data) under this loss.

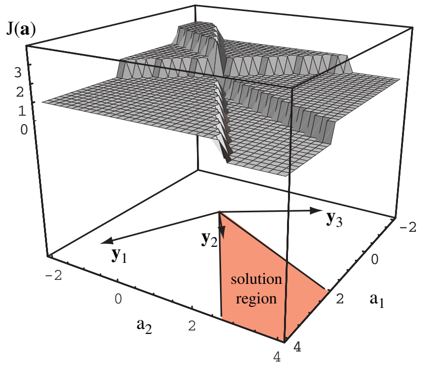
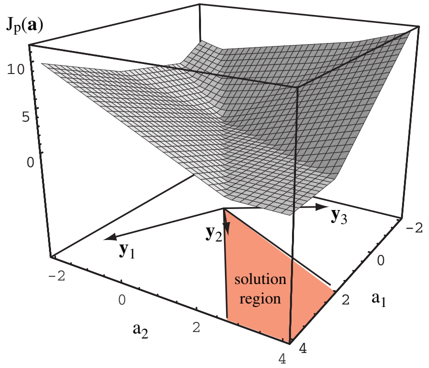


Illustration: R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, Wiley 2001

THE PERCEPTRON CRITERION

- Piece-wise constant function not suitable for numerical optimization.
- Approximate by piece-wise linear function $\rightarrow$ perceptron cost function



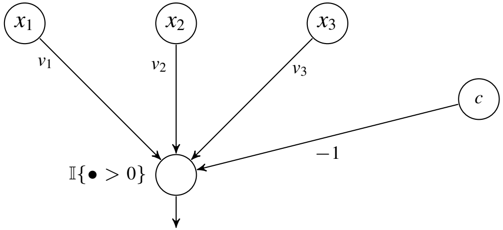
NNs and Linear Classification

Linear classifier:

$$f(x) = \mathrm{sgn}(\mathbf{v}^t\mathbf{x} - c)$$

Equivalently:

$$\mathbb{I}\{\mathbf{v}^t\mathbf{x} - c > 0\}$$



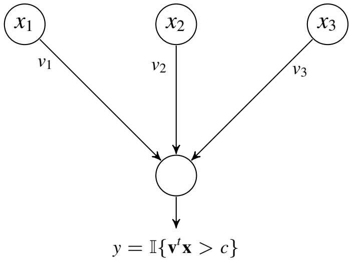
NNs and Linear Classification

Linear classifier:

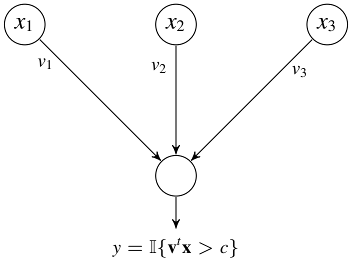
$$f(x) = \operatorname{sgn}(\mathbf{v}^t\mathbf{x} - c)$$

Equivalently:

$$\mathbb{I}\{\mathbf{v}^t\mathbf{x} - c > 0\}$$

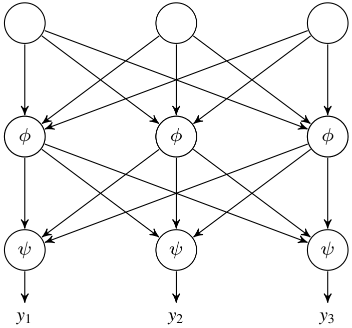


Remarks



- The "neural network" only represents linear classifier, for fixed weights.
- It does not specify the training method.
- "Neural networks" as a field refers to large collection of algorithms and engeneering tricks for function estimation.
- These algorithms typically exploit the layered structure.
- Personally, I would recommend to beware of all claims of "neurologial plausibility".

ADDING LAYERS



$$y_i = \psi\Big(\sum_j w_{ij}\phi\Big(\sum_k v_{jk}x_k\Big)\Big)$$

Training

$$y_i = \psi\Big(\sum_j w_{ij}\phi\Big(\sum_k v_{jk}x_k\Big)\Big)$$

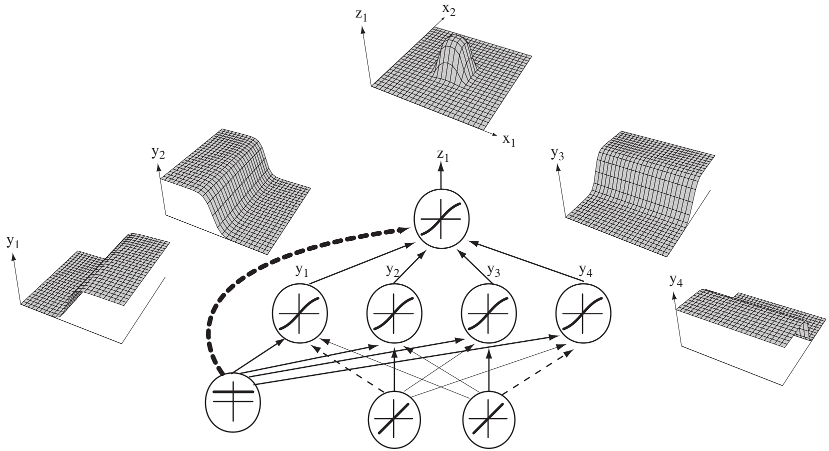
Given: y_i, x_k Estimate: w_{ij}, v_{jk}

Error Backpropagation

- Use chain rule for differentiation to compute dependence of ψ on w, v
- Differential approximates the correction Δv and Δw we have to apply to produce a correction $\Delta\psi$.

Algorithm

1. For current weights, run network on input x
2. Compare output $\psi(\dots)$ to training values y
3. Make corrections Δv, Δw
4. Repeat


z_1
x_2
x_1
y_1
y_2
y_3
y_4
z_1
y_1
y_2
y_3
y_4

Sample theoretical result

Theorem (Barron, 1993)

If $f : \mathbb{R}^d \to \mathbb{R}$ satisfies a smoothness conditions, namely

$$C_f = \int \|\omega\| \hat{f}(\omega) d\omega < \infty \qquad \text{for Fourier transform } \hat{f} \ ,$$

the restriction of f to a disk of radius r can be approximated using N hidden units of simple funcions (sigmoids, threshold or ramp functions) to precision $(2rC_f/\sqrt{N})$.

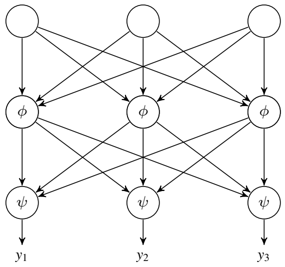
- Note the error rate $2rC_f/\sqrt{N}$ does not seem to depend on dimension d.
- In the wake of this result, a hype declared neural networks can beat the curse of dimensionality.
- Not so: The class of functions satisfying the smoothness condition shrinks with d.
- Much like a filter bank, a NN represents a function basis.

Feature interpretation

Think of applying a 3-layer NN to **x** as:

1. Applying a 2-layer NN to **x**.
2. Applying another 2-layer NN to the output.

The output of the first NN can be regarded as an intermediate representation.

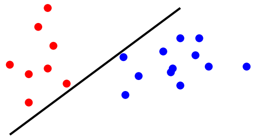


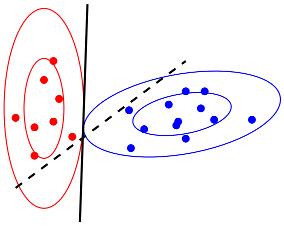
Convolutional NNs

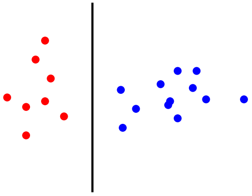
- Suppose NN is applied to an image.
- Each input vertex is a location in the image (a pixel, say).
- Now constrain the weights applied at different locations to be identical at all locations (“tied weights”).
- Result: Output is translation invariant.
- One can also achieve rotation invariance.
- These invariance properties are very valuable in computer vision.

Maximum Margin Classifiers

Generalization Error


Possible Perceptron solution


Good generalization under a specific distribution (here: Gaussian)


Maximum margin solution

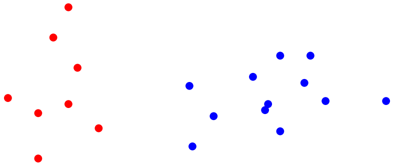
Maximum margin idea

To achieve good generalization (low prediction error), place the hyperplane “in the middle” between the two classes.

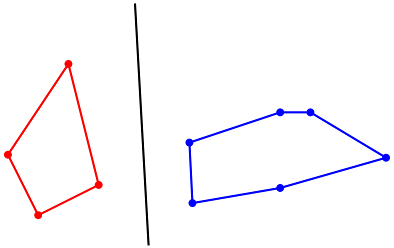
Without distributional assumption: Max-margin classifier

- Philosophy: Without distribution assumptions, best guess is symmetric.
- In the Gaussian example, the max-margin solution would *not* be optimal.

LINEAR SEPARATION AND CONVEXITY



Sets can be separated by a hyperplane if and only if their convex hulls are disjoint.



The distance of the plain to the closest set is called the **margin**.

LINEAR CLASSIFIER WITH MARGIN

Recall: Specifying affine plane

Normal vector $\mathbf{v}_{\mathrm{H}}$.

$$\langle \mathbf{v}_{\mathrm{H}}, \mathbf{x} \rangle - c \begin{cases} > 0 & \mathbf{x} \text{ on positive side} \\ < 0 & \mathbf{x} \text{ on negative side} \end{cases}$$

Scalar $c \in \mathbb{R}$ specifies shift (plane through origin if $c = 0$).

Plane with margin

Demand

$$\langle \mathbf{v}_{\mathrm{H}}, \mathbf{x} \rangle - c > 1 \text{ or } < -1$$

$\{-1, 1\}$ on the right works for any margin: Size of margin determined by $\|\mathbf{v}_{\mathrm{H}}\|$. To increase margin, scale down $\mathbf{v}_{\mathrm{H}}$.

Classification

Concept of margin applies only to training, not to classification. Classification works as for any linear classifier. For a test point $\mathbf{x}$:

$$y = \operatorname{sign}\left(\langle \mathbf{v}_{\mathrm{H}}, \mathbf{x} \rangle - c\right)$$

SUPPORT VECTOR MACHINE

Finding the hyperplane

For n training points $(\tilde{\mathbf{x}}_i, \tilde{y}_i)$ with labels $\tilde{y}_i \in \{-1, 1\}$, solve optimization problem:

$$\begin{aligned} \min_{\mathbf{v}_{\mathrm{H}}, c} \quad & \|\mathbf{v}_{\mathrm{H}}\| \\ \text{s.t.} \quad & \tilde{y}_i(\langle \mathbf{v}_{\mathrm{H}}, \tilde{\mathbf{x}}_i \rangle - c) \geq 1 \quad \text{for } i = 1, \dots, n \end{aligned}$$

Definition

The classifier obtained by solving this optimization problem is called a **support vector machine**.

Dual Optimization Problem

Solving the SVM opimization problem

$$\begin{aligned} \min_{\mathbf{v}_{\mathrm{H}},c} \quad & \|\mathbf{v}_{\mathrm{H}}\| \\ \text{s.t.} \quad & \tilde{y}_i(\langle \mathbf{v}_{\mathrm{H}}, \tilde{\mathbf{x}}_i \rangle - c) \geq 1 \quad \text{for } i = 1, \ldots, n \end{aligned}$$

is difficult, because the constraint is a function. It is possible to transform this problem into a problem which seems more complicated, but has simpler constraints:

$$\begin{aligned} \max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad & W(\boldsymbol{\alpha}) := \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j \langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle \\ \text{s.t.} \quad & \sum_{i=1}^{n} \tilde{y}_i \alpha_i = 0 \\ & \alpha_i \geq 0 \quad \text{for } i = 1, \ldots, n \end{aligned}$$

This is called the optimization problem **dual** to the minimization problem above. It is usually derived using Lagrange multipliers. We will use a more geometric argument.

Deriving the Dual Problem

Idea

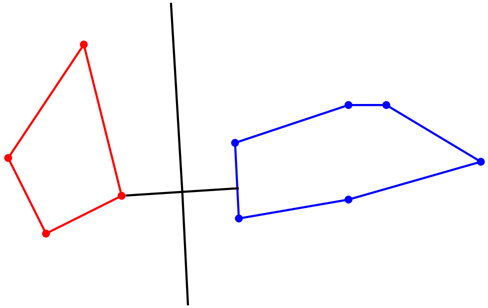
As a consequence of duality on previous slide, we can find the maximum-margin plane as follows:

1. Find shortest line connecting the convex hulls.
2. Place classifier orthogonal to line in the middle.

Convexity of sets ensures that this classifier has correct orientation.

As optimization problem

$$\min_{\substack{\mathbf{u}\in\mathrm{conv}(\mathcal{X}_\ominus)\\ \mathbf{v}\in\mathrm{conv}(\mathcal{X}_\oplus)}} \quad \|\mathbf{u}-\mathbf{v}\|^2$$



Dual optimization problem

Dual problem

$$\begin{aligned}
\|\sum_{i\in\mathcal{X}_\ominus}\alpha_i\tilde{\mathbf{x}}_i-\sum_{i\in\mathcal{X}_\oplus}\alpha_i\tilde{\mathbf{x}}_i\|_2^2 &= \|\sum_{i\in\mathcal{X}_\ominus}\tilde{y}_i\alpha_i\tilde{\mathbf{x}}_i+\sum_{i\in\mathcal{X}_\oplus}\tilde{y}_i\alpha_i\tilde{\mathbf{x}}_i\|_2^2 \\
&= \left\langle\sum_{i=1}^n\tilde{y}_i\alpha_i\tilde{\mathbf{x}}_i\,,\,\sum_{i=1}^n\tilde{y}_i\alpha_i\tilde{\mathbf{x}}_i\right\rangle=\sum_{i,j}\tilde{y}_i\tilde{y}_j\alpha_i\alpha_j\left\langle\tilde{\mathbf{x}}_i,\tilde{\mathbf{x}}_j\right\rangle
\end{aligned}$$

Note: Minimizing this term under the constraints is equivalent to *maximizing*

$$\sum_i\alpha_i-\frac{1}{2}\sum_{i,j}\tilde{y}_i\tilde{y}_j\alpha_i\alpha_j\left\langle\tilde{\mathbf{x}}_i,\tilde{\mathbf{x}}_j\right\rangle$$

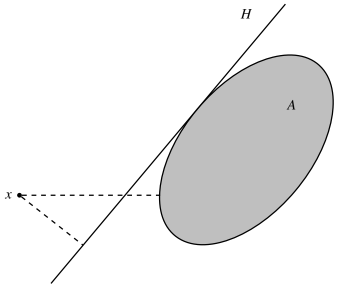
under the same constraints, since $\sum_i\alpha_i=2$ is constant. That is just the dual problem defined four slides back.

Sets and Planes

Many dual relations in convex optimization can be traced back to the following fact:

The closest distance between a point **x** *and a convex set A is the maximum over the distances between* **x** *and all hyperplanes which separate* **x** *and A.*

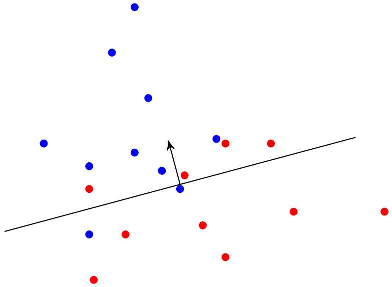
$$d(\mathbf{x}, A) = \sup_{H \text{ separating}} d(\mathbf{x}, H)$$



Ensemble Classifiers

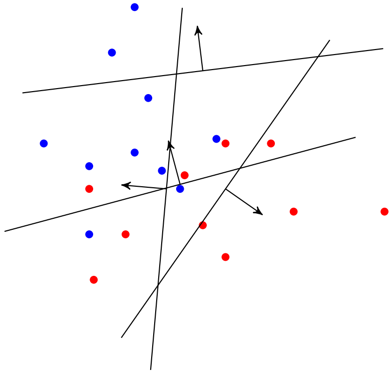
ENSEMBLES

A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



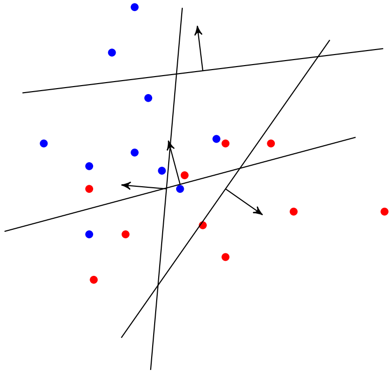
ENSEMBLES

A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



Ensembles

A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



- Many random hyperplanes combined by majority vote: Still 0.5.
- A single classifier slightly better than random: $0.5 + \varepsilon$.
- What if we use *m* such classifiers and take a majority vote?

VOTING

Decision by majority vote

- m individuals (or classifiers) take a vote. m is an odd number.
- They decide between two choices; one is correct, one is wrong.
- After everyone has voted, a decision is made by simple majority.

Note: For two-class classifiers $f_1, \dots, f_m$ (with output ± 1):

$$\text{majority vote } = \operatorname{sgn}\Big(\sum_{j=1}^{m} f_j\Big)$$

Assumptions

Before we discuss ensembles, we try to convince ourselves that voting can be beneficial. We make some simplifying assumptions:

- Each individual makes the right choice with probability $p \in [0, 1]$.
- The votes are *independent*, i.e. stochastically independent when regarded as random outcomes.

Does the Majority Make the Right Choice?

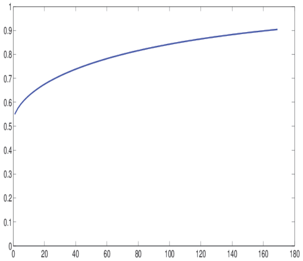
Condorcet's rule

If the individual votes are independent, the answer is

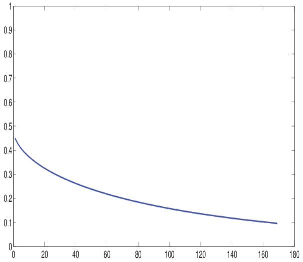
$$\Pr\{\text{ majority makes correct decision }\} = \sum_{j=\frac{m+1}{2}}^{m} \frac{m!}{j!(m-j)!} p^j (1-p)^{m-j}$$

This formula is known as **Condorcet's jury theorem**.

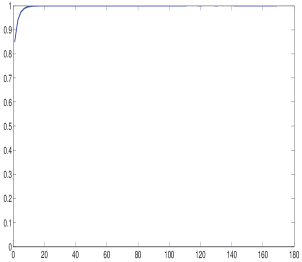
Probability as function of the number of votes



$p = 0.55$



$p = 0.45$



$p = 0.85$

ENSEMBLE METHODS

Terminology

- An **ensemble method** makes a prediction by combining the predictions of many classifiers into a single vote.
- The individual classifiers are usually required to perform only slightly better than random. For two classes, this means slightly more than 50% of the data are classified correctly. Such a classifier is called a **weak learner**.

Strategy

- We have seen above that if the weak learners are random and independent, the prediction accuracy of the majority vote will increase with the number of weak learners.
- Since the weak learners all have to be trained on the training data, producing random, independent weak learners is difficult.
- Different ensemble methods (e.g. Boosting, Bagging, etc) use different strategies to train and combine weak learners that behave relatively independently.

AdaBoost

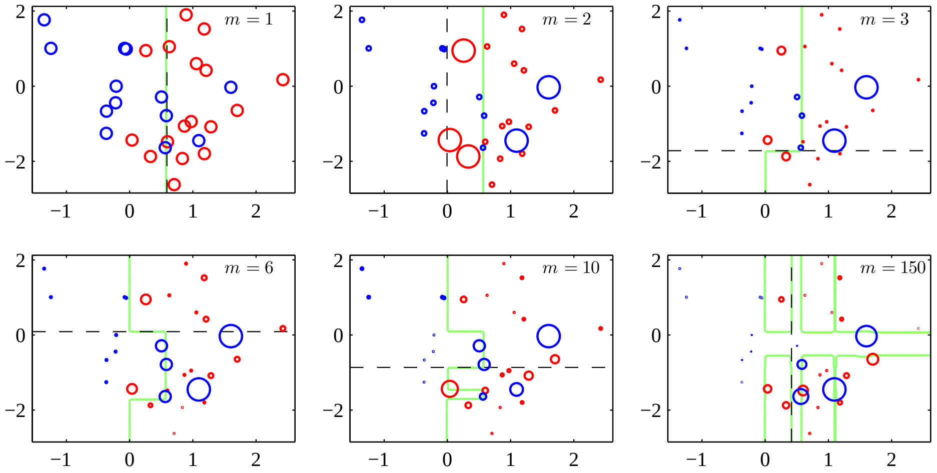
Input

- Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \dots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$
- Algorithm parameter: Number M of weak learners

Training algorithm

```
1. Initialize the observation weights $w_i = \frac{1}{n}$ for $i = 1, 2, ..., n$.
2. For $m = 1$ to $M$:
    2.1 Fit a classifier $g_m(x)$ to the training data using weights $w_i$.
    2.2 Compute
        $$\mathrm{err}_m := \frac{\sum_{i=1}^{n} w_i \mathbb{I}\{y_i \neq g_m(x_i)\}}{\sum_i w_i}$$
    2.3 Compute $\alpha_m = \log(\frac{1-\mathrm{err}_m}{\mathrm{err}_m})$
    2.4 Set $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot \mathbb{I}(y_i \neq g_m(x_i)))$ for $i = 1, 2, ..., n$.
3. Output
    $$f(x) := \mathrm{sign}\left(\sum_{m=1}^{M} \alpha_m g_m(x)\right)$$
```

ILLUSTRATION



Circle = data points, circle size = weight.

Dashed line: Current weak learner. Green line: Aggregate decision boundary.

Illustration: C.M. Bishop, *Pattern Recognition and Machine Learning*, Springer 2006

Bootstrapping weak learners

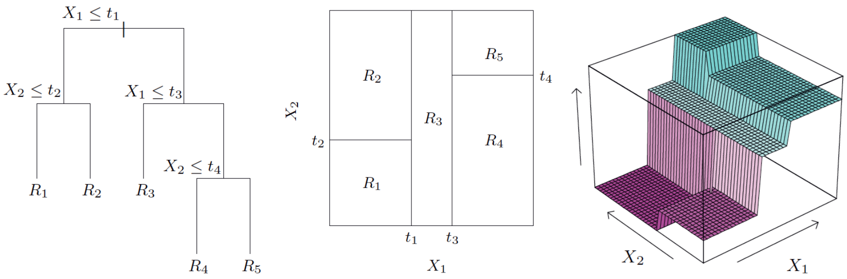
Idea

- Recall Boosting: Weak learners are deterministic, but selected to exhibit high variance.
- Strategy now: Randomly distort data set by resampling.
- Train weak learners on resampled training sets.
- Resulting algorithm: **Bagging** (= **B**ootstrap **agg**regation)

Bagging tree classifiers

- Draw a bootstrap resample of the traninig data
- Train a tree classifier on the resample
- Repeat

TREES



- Each leaf of the tree corresponds to a region R_m of $\mathbb{R}^d$.
- Classes $k \in \{1, \ldots, K\}$ (not restricted to two classes).
- Training: Greedy splits

BAGGED TREES

- Two classes, each with Gaussian distribution in $\mathbb{R}^5$.
- Note the variance between bootstrapped trees.

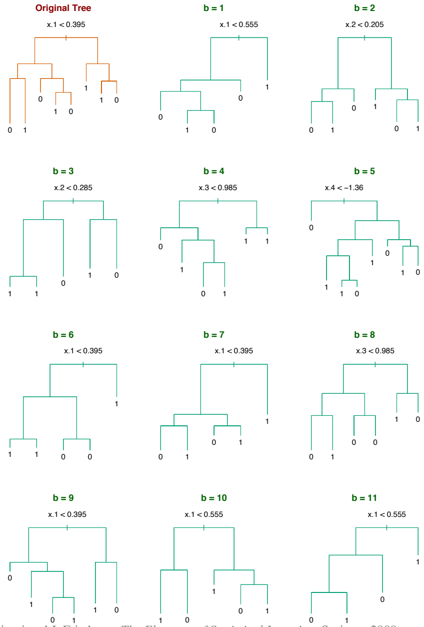


Illustration: T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, Springer 2009

Random Forests

Bagging vs. Boosting

- Bagging works particularly well for trees, since trees have high variance.
- Boosting typically outperforms bagging with trees.
- The main culprit is usually dependence: Boosting is better at reducing correlation between the trees than bagging is.

Random Forests

Modification of bagging with trees designed to further reduce correlation.

- Tree training optimizes each split over all dimensions.
- Random forests choose a different subset of dimensions *at each split*.
- Optimal split is chosen within the subset.
- The subset is chosen at random out of all dimensions $\{1, \ldots, d\}$.

RANDOM FORESTS: ALGORITHM

Training

Input parameter: m (positive integer with $m < d$)

For $b = 1, \dots, B$:

1. Draw a bootstrap sample $\mathcal{B}_b$ of size n from training data.
2. Train a tree classifier f_b on $\mathcal{B}_b$, where each split is computed as follows:
   - Select m axes in $\mathbb{R}_d$ at random.
   - Find the best split (j^*, t^*) on this subset of dimensions.
   - Split current node along axis j^* at t^*.

Classification

Exactly as for bagging: Classify by majority vote among the B trees. More precisely:

- Compute $f_{\text{avg}}(\mathbf{x}) := (p_1(\mathbf{x}), \dots, p_k(\mathbf{x})) := \frac{1}{B} \sum_{b=1}^{B} f_b(\mathbf{x})$
- The Random Forest classification rule is

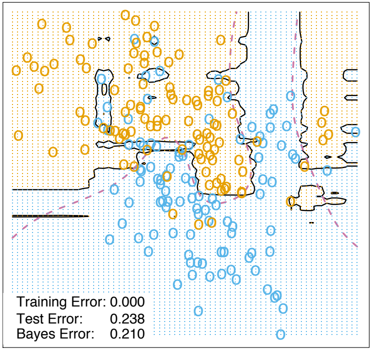
$$f_{\text{Bagging}}(\mathbf{x}) := \arg\max_k \{p_1(\mathbf{x}), \dots, p_k(\mathbf{x})\}$$

Remarks

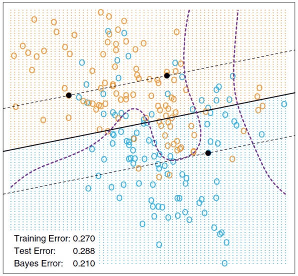
- Recommended value for m is $m = \lfloor\sqrt{d}\rfloor$ or smaller.
- RF typically achieve similar results as boosting. Implemented in most packages, often as standard classifier.

Example: Synthetic Data

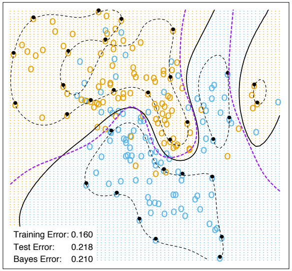
- This is the RF classification boundary on the synthetic data we have already seen a few times.
- Note the bias towards axis-parallel alignment.



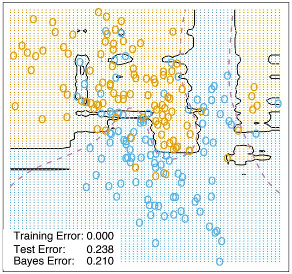
COMPARISON: CLASSIFIERS



Linear SVM



RBF SVM



Random forest

Illustration: T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, Springer 2009

Comparison: Classifiers

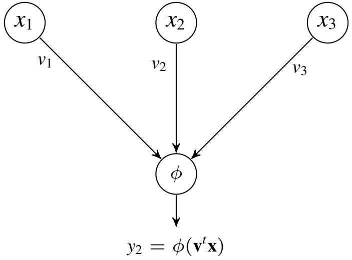
Perhaps best off-the-shelve choices

- SVM (with RBF kernel)
- Random Forest

Every method has its idiosyncrasies

- SVM: Treats feature space *geometrically*
- RF:
  - Approximates correlation by piece-wise constant decision surface
  - Applicable to almost any type of data
  - Works naturally with multiple classes
- Boosting (with decision stumps = AdaBoost): Weights provide built-in feature selection

	Decision boundary	Feature space	Tuning parameters
Perceptron	Linear	Euclidean	"Learning rate"
SVM (linear)	Linear	Euclidean	Slack
SVM (RBF)	Nonlinear	Euclidean	Slack Bandwidth
Random forest	Nonlinear	Flexible	Tree depth # resamples # dimensions



Markov models

Markov Models

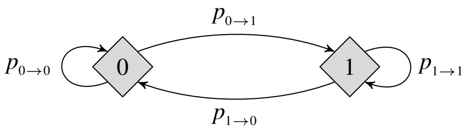
Markov models

The sequence $(X_n)_n$ is called a (first-order) **Markov chain** if

$$P(X_n|X_{n-1},\ldots,X_1) = P(X_n|X_{n-1}) \ .$$

Example: Random binary seqeunce

$$\begin{tikzcd}
0 \arrow[loop left, "p_{0\to0}"] \arrow[r, bend left, "p_{0\to1}"] & 1 \arrow[loop right, "p_{1\to1}"] \arrow[l, bend left, "p_{1\to0}"]
\end{tikzcd}$$



Parametrization

$$P_{\text{init}} := \begin{pmatrix} \Pr\{X_0 = 1\} \\ \dots \\ \Pr\{X_0 = d\} \end{pmatrix} \qquad \mathbf{p} := (p_{i\to j})_{j,i\leq d} = \begin{pmatrix} p_{1\to 1} & \cdots & p_{d\to 1} \\ \vdots & & \vdots \\ p_{1\to d} & \cdots & p_{d\to d} \end{pmatrix}$$

The chain $(X_n)_{n\in\mathbb{N}}$ is **stationary** if $p_{s\to t}$ does not depend on n.

General case

If X_n takes values in infinite set, $\mathbf{p}$ becomes a probability kernel.

State Probabilities

Probability after $n = 1$ steps

$$P_1(s_1) = \sum_{s_0 \in \mathbf{X}} p_{s_0 \to s_1} P_{\text{init}}(s_0) \qquad \text{hence} \qquad P_1 = \mathbf{p} \cdot P_{\text{init}} \ .$$

Probability after n steps

$$P_n = \mathbf{p}^n P_{\text{init}}$$

Limiting distribution

$$P_\infty := \lim_{n \to \infty} P_n = \lim_{n \to \infty} \mathbf{p}^n P_{\text{init}}$$

(need not exist!)

INVARIANT DISTRIBUTION

Observation

If the limit P_∞ exists, then

$$\mathbf{p} \cdot P_\infty = \mathbf{p} \cdot \lim_{n \to \infty} \mathbf{p}^n P_{\text{init}} = \lim_{n \to \infty} \mathbf{p}^n P_{\text{init}} = P_\infty \ .$$

Therefore, P_∞ is called an **equilibrium** or **invariant distribution** of the chain.

Theorem

Suppose a Markov chain $(\mathbf{p}, P_{\text{init}})$ is stationary, and for each state $s \in \mathbf{X}$:

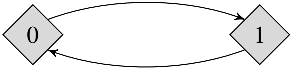
1. There is a path (with non-zero probability) from s to every other state (i.e. the chain is irreducible).
2. $p_{s \to s} > 0$ (i.e. the chain is aperiodic).

Then:

- The limit distribution P_∞ exists.
- The limit distribution is also the equlibrium distribution.
- The equilibrium distribution is unique.

What Can Go Wrong?

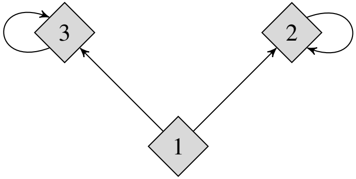
Problem 1: The limit may not exist



$$P_n \text{ oscillates between } P_{\text{even}} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \text{and} \qquad P_{\text{odd}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} .$$

Calculus analogy: $\lim_n (-1)^n$ does not exist.

Problem 2: The equilibrium distribution may not be unique



Both $P = (0, 1, 0)$ and $P' = (0, 0, 1)$ are valid equilibria.

Application: Internet search

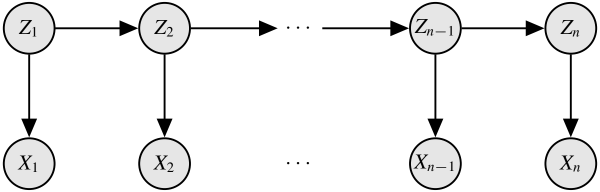
Random walks as Markov chains

- Simple random walk on a connected graph is a Markov chain.
- Its state space is the vertex set.
- Its equilibrium is the degree-biased distribution.

Internet search

- Idea: Popularity of a web page is a good proxy for whether it is interesting.
- A web crawler can determine which web page links which.
- It cannot determine how often a link is followed.
- "Random surfer model": Popularity score = probability that a random web surfer would land on the page.
- Random surfer is modeled as simple random walk.
- Google's PageRank algorithm approximates the equilibrium distribution of simple random walk on the web graph.

APPLICATION: HMMS



- In a HMM, the *latent* variables (Z_n) form a Markov chain.
- Since these are not observed, the sequence (X_n) can exhibit long-range dependencies.
- This is another example of how multiple layers of simple structures express more complicated structures.

REVERSIBILITY

Reversible Markov chains

A Markov chain $(X_1, \dots, X_n)$ is **reversible** if

$$(X_1, \dots, X_n) \overset{\mathrm{d}}{=} (X_n, \dots, X_1)$$

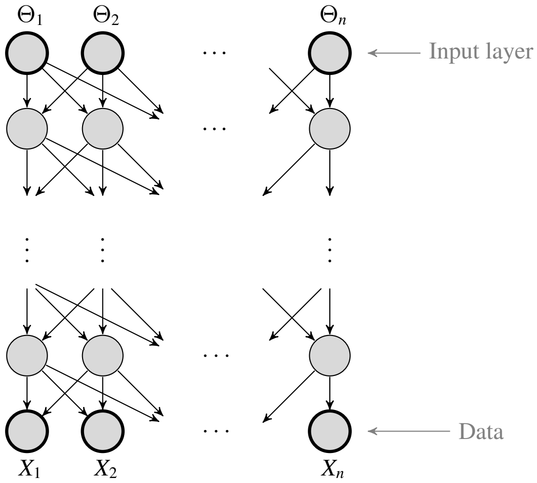
Detailed balance condition

$$P_\infty(dx) P(X_n \in dx' | X_{n-1} = x) = P_\infty(dx') P(X_n \in dx | X_{n-1} = x')$$

If a stationary Markov chain satisfied detailed balance, it is reversible.

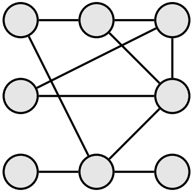
MANY HIDDEN LAYERS

DIRECTED GRAPHICAL MODEL



Graphical model distribution on $\{0,1\}^n$ with joint law

$$P(x_1,\ldots,x_n) = \frac{e^{\mathbf{x}^t W \mathbf{x} + \mathbf{c}^t \mathbf{x}}}{Z(W,\mathbf{c})}$$

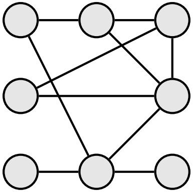


- This is a Markov random field.
- The Markov blanket of X_i are those X_j with $W_{ij} \neq 0$.
- For $\mathbf{x} \in \{-1,1\}^n$: “Potts model with external magnetic field”.
- Statisticians might call this an exponential family with sufficient statistics $\mathbb{E}[X_i X_j]$ and $\mathbb{E}[X_i]$.

Boltzmann machine

Graphical model distribution on $\{0,1\}^n$ with joint law

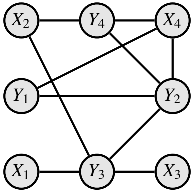
$$P(x_1,\dots,x_n) = \frac{e^{\mathbf{x}^t W \mathbf{x} + \mathbf{c}^t \mathbf{x}}}{Z(W,\mathbf{c})}$$



- This is a Markov random field.
- The Markov blanket of X_i are those X_j with $W_{ij} \neq 0$.
- For $\mathbf{x} \in \{-1,1\}^n$: “Potts model with external magnetic field”.
- Statisticians might call this an exponential family with sufficient statistics $\mathbb{E}[X_i X_j]$ and $\mathbb{E}[X_i]$.

We dispense with tedious accuracy and call P a **Boltzmann machine**.

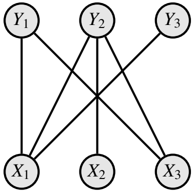
Restricted Boltzmann machine



With observations

If some some vertices represent observation variables Y_i:

$$P(x_1,\ldots,x_n,y_1,\ldots,y_m) = \frac{e^{(\mathbf{x},\mathbf{y})^t W(\mathbf{x},\mathbf{y}) + \mathbf{c}^t\mathbf{x} + \tilde{\mathbf{c}}^t\mathbf{y}}}{Z(W,\mathbf{c},\tilde{\mathbf{c}})}$$



Recall our hierarchical design approach

- Only permit layered structure.
- Obvious grouping: One layer for X, one for Y.
- As before: No connections *within* layers.
- Since the graph is undirected, that makes it bipartite.

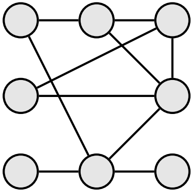
A **restricted Boltzmann machine (RBM)** is a BM that is bipartite.

GIBBS SAMPLING BOLTZMANN MACHINES

Full conditionals: General case

$$P(\mathbf{X} = \mathbf{x}) = \frac{e^{\mathbf{x}^t W \mathbf{x} + \mathbf{c}^t \mathbf{x}}}{Z(W, \mathbf{c})}$$

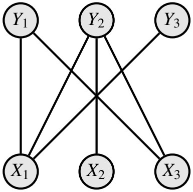
$$P(X_i = 1 | x_{(i)}) = \sigma(W_i^t \mathbf{x} + c_i)$$



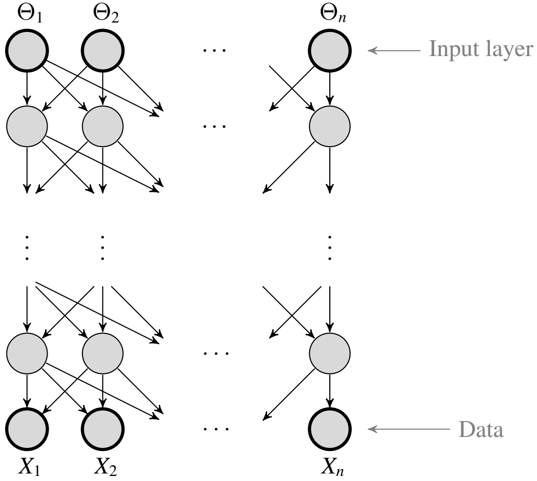
Full conditionals: RBM

- X's are conditional independent given Y's and vice versa
- Two groups of conditionals: $X|Y$ and $Y|X$
- Blocked Gibbs samplers

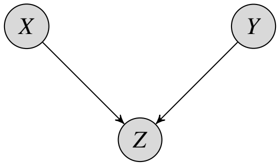
$$P(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}) = \sigma(W^t \mathbf{y} + \mathbf{c}')$$
$$P(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x}) = \sigma(W^t \mathbf{x} + \mathbf{c})$$



DIRECTED GRAPHICAL MODEL



Recall: Explaining away



$$\begin{tikzcd} X \arrow[dr] & & Y \arrow[dl] \\ & Z & \end{tikzcd}$$

Conditioning on Z makes X and Y dependent.

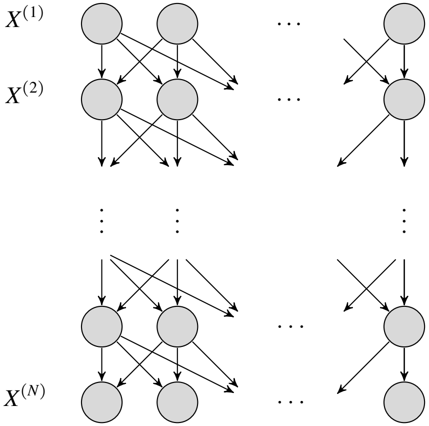
Complementary Prior

Observation

$X^{(1)}, X^{(2)}, \dots, X^{(N)}$ is a Markov chain.

Complementary prior idea

- Suppose Markov chain is reversible.
- Then all arrows can be reversed.
- Now: Inference easy.



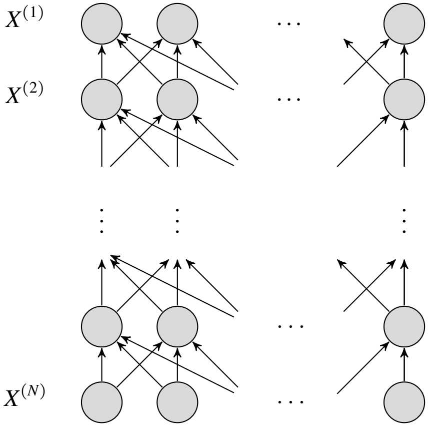
Complementary Prior

Observation

$X^{(1)}, X^{(2)}, \dots, X^{(N)}$ is a Markov chain.

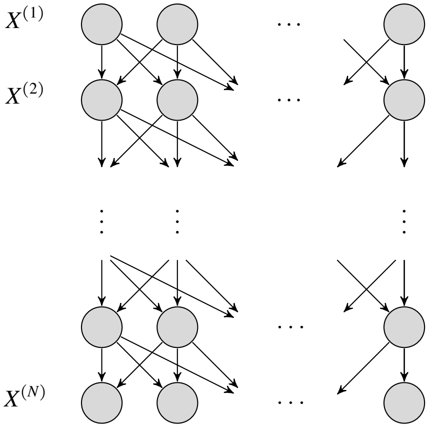
Complementary prior idea

- Suppose Markov chain is reversible.
- Then all arrows can be reversed.
- Now: Inference easy.



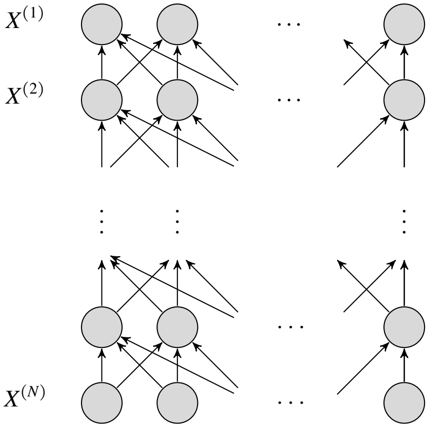
Building a Complementary Prior

- Find reversible Markov chain with
$$P^{(1)} = P_\infty$$
- Let $\mathbf{p}_{\mathrm{T}}$ be its transition kernel
- Choose
$$P^{(n+1)}(\bullet|X^{(n)} = \mathbf{x}) = \mathbf{p}_{\mathrm{T}}(\bullet|\mathbf{x})$$
- Then $P^{(2)} = \ldots = P^{(n)} = P_\infty$
- Since chain is reversible,
$$P^{(n)}(\bullet|X^{(n+1)} = \mathbf{x}) = \mathbf{p}_{\mathrm{T}}(\bullet|\mathbf{x})$$
and edges flip.



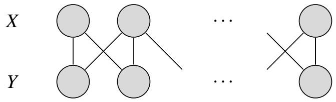
Building a Complementary Prior

- Find reversible Markov chain with
$$P^{(1)} = P_\infty$$
- Let $\mathbf{p}_{\mathrm{T}}$ be its transition kernel
- Choose
$$P^{(n+1)}(\bullet|X^{(n)} = \mathbf{x}) = \mathbf{p}_{\mathrm{T}}(\bullet|\mathbf{x})$$
- Then $P^{(2)} = \ldots = P^{(n)} = P_\infty$
- Since chain is reversible,
$$P^{(n)}(\bullet|X^{(n+1)} = \mathbf{x}) = \mathbf{p}_{\mathrm{T}}(\bullet|\mathbf{x})$$
and edges flip.



WHERE DO WE GET THE MARKOV CHAIN?

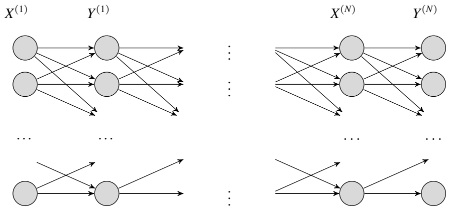
Start with an RBM



Blocked Gibbs sampling alternates between X and Y

$$X^{(1)} \to Y^{(1)} \to X^{(2)} \to Y^{(2)} \to \ldots \to X^{(N)} \to Y^{(N)}$$

"Roll off" this chain into a graphical model



The *Gibbs sampler* for the RBM becomes the *model* for the directed network.

Remarks

- “Deep learning” refers to hierarchies with multiple hidden layers.
- There are other methods to build/train “deep” hierarchies.
- Many training methods are of the form: Initialize with method (a), fine-tune with method (b), etc
- “Deep” now refers to everything that is hierarchical and has more than two layers.
- Some deep networks are indeed deep: ImageNet now has >100 layers

How deep is deep?

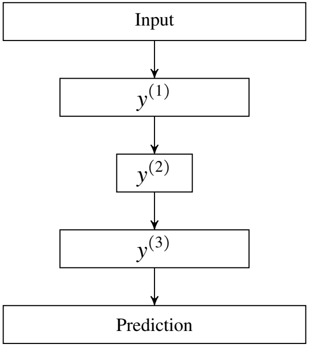
- Hinton/Osindero/Teh, 2006: Input + output + 3 hidden layers.
- Models currently used for computer vision problems: >100 layers.

AUTOENCODERS

$$\begin{tikzcd}
\boxed{\text{Input}} \arrow[d] \\
\boxed{y^{(1)}} \arrow[d] \\
\boxed{y^{(2)}} \arrow[d] \\
\boxed{y^{(3)}} \arrow[d] \\
\boxed{\text{Prediction}}
\end{tikzcd}$$

$$\text{Prediction} \quad = \quad f(\text{Input})$$

AUTOENCODERS



$$\text{Prediction} \quad = \quad f(\text{Input})$$

AUTOENCODERS

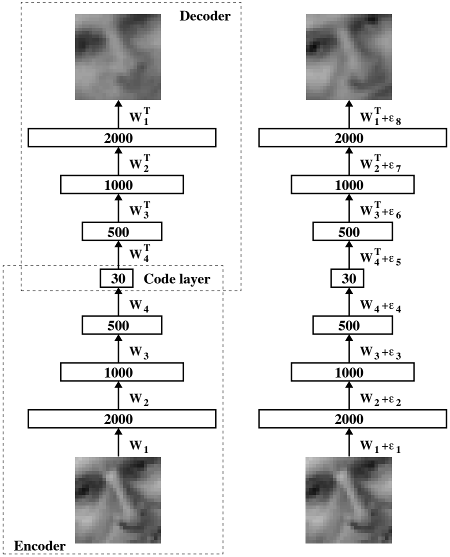


Illustration: K. Murphy, *Machine Learning: A Bayesian perspective*, MIT Press 2012