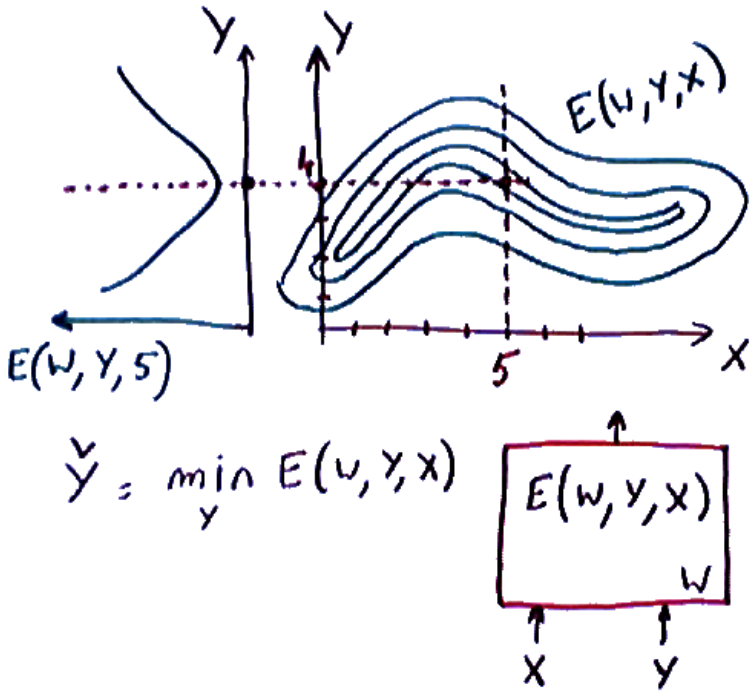
MACHINE LEARNING AND PATTERN RECOGNITION

Fall 2004, Lecture 2:

Energy-Based Models and Loss Functions, Linear Machines

Yann LeCun
The Courant Institute,
New York University
http://yann.lecun.com

Energy-Based Models

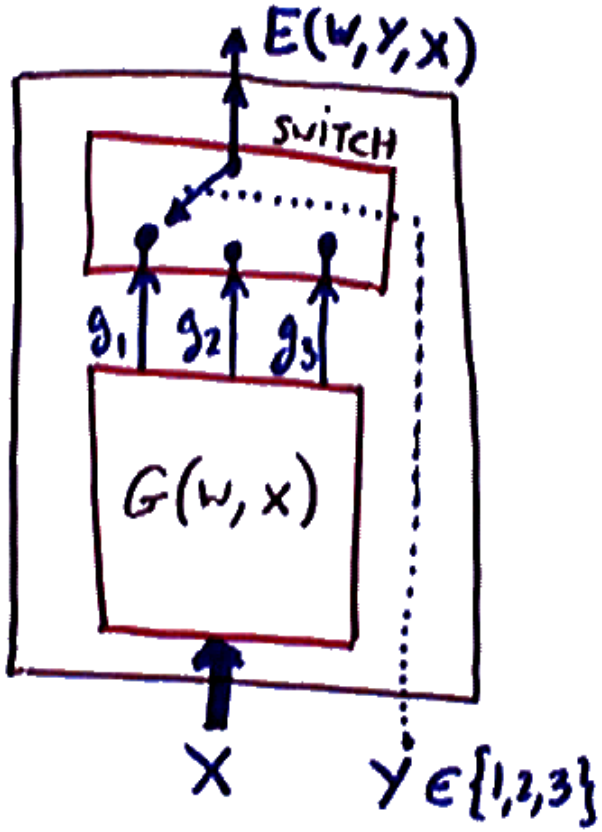


- An energy-based model is a scalar-valued *energy function*: $E(W, Y, X)$.
- X is the input, and Y the variable to be predicted (output).
- W is the parameter vector to be learned.
- X and Y can be discrete variables, scalars, vectors, tensors, sequences, probability distributions, or any other entity.

Minimum Energy Machine: Operating the machine (performing an *inference*), consists in taking an input X, and looking for the value of Y within a permissible set $\{Y\}$, that minimizes $E(W, Y, X)$:

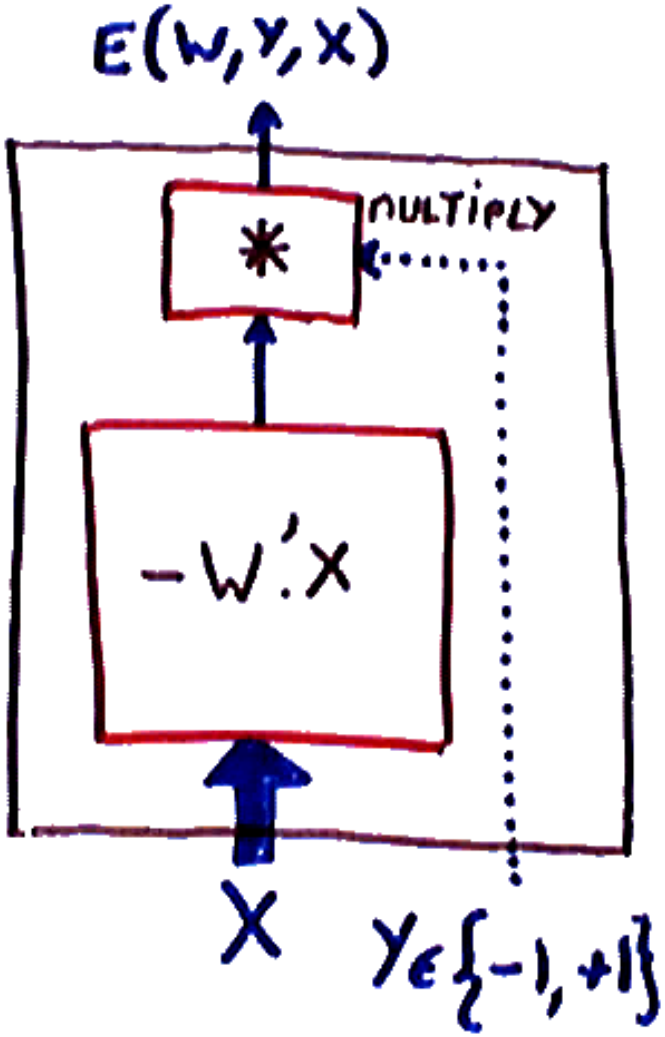
$$\check{Y} = \text{argmin}_{Y \in \{Y\}} E(W, Y, X)$$

Examples of EBM: Classifier



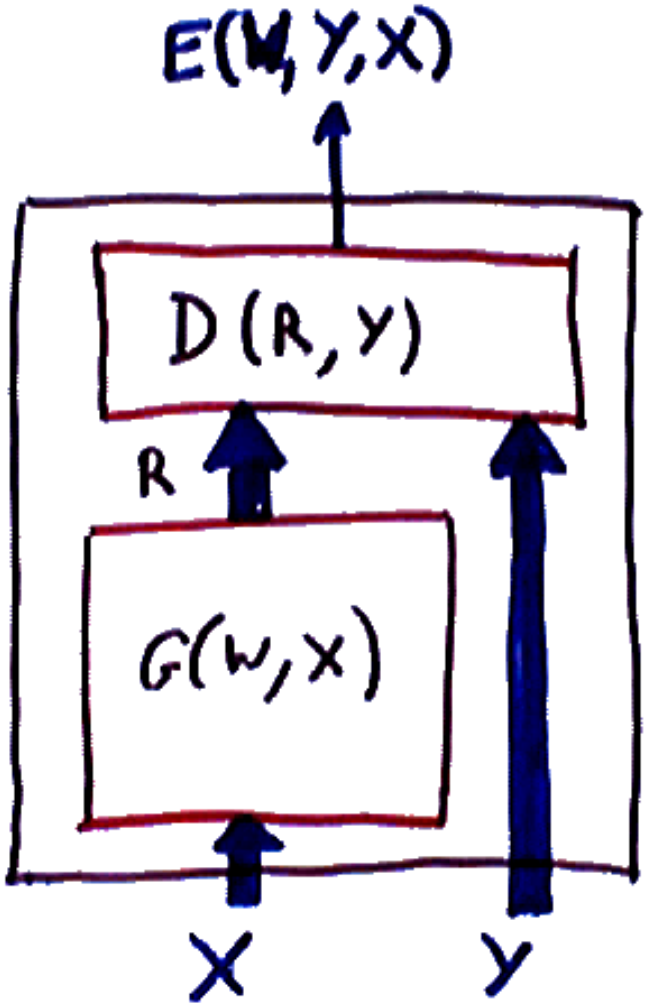
- Y is a discrete variable, $\{Y\} = \{1, 2, 3\}$.
- Energy: $E(W, Y, X) = \sum_k G_k(W, X)\delta(k, Y)$, where $\delta(k, Y) = 1$ iff $k = Y$ and 0 otherwise.
- $G_k(W, X)$, the k-th component of the output vector of $G(W, X)$ is interpreted as the "cost" of classifying X into category k.
- Best output: $\check{Y} = \min_{Y \in \{Y\}} E(W, Y, X) = \min_k G_k(W, X)$.

Examples of EBM Classifier: Perceptron



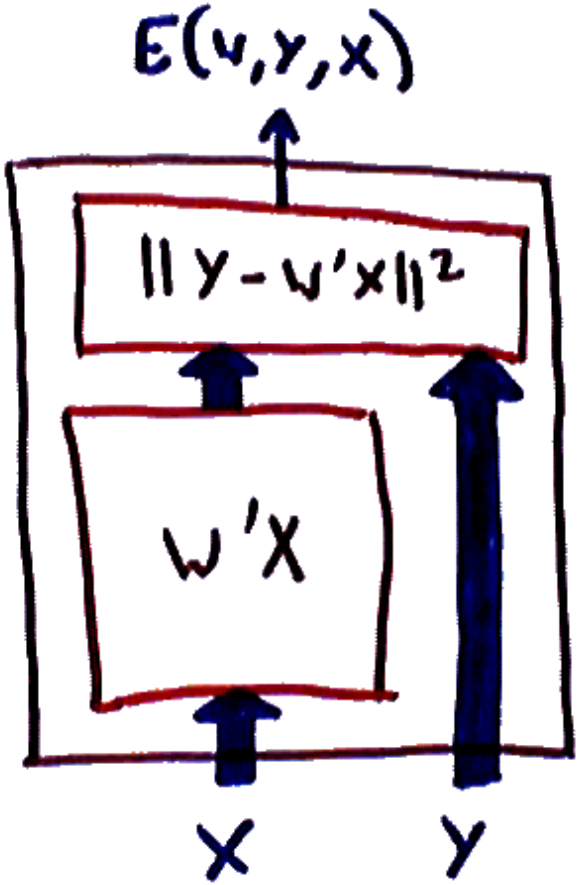
- Y is a discrete variable, $\{Y\} = \{-1, +1\}$.
- Energy: $E(W, Y, X) = -Y.W'X$.
- Best output: $\check{Y} = \text{sign}(W'X)$, where $\text{sign}(R) = +1$ iff $R > 0$ and -1 otherwise.

Examples of EBM: Regressor



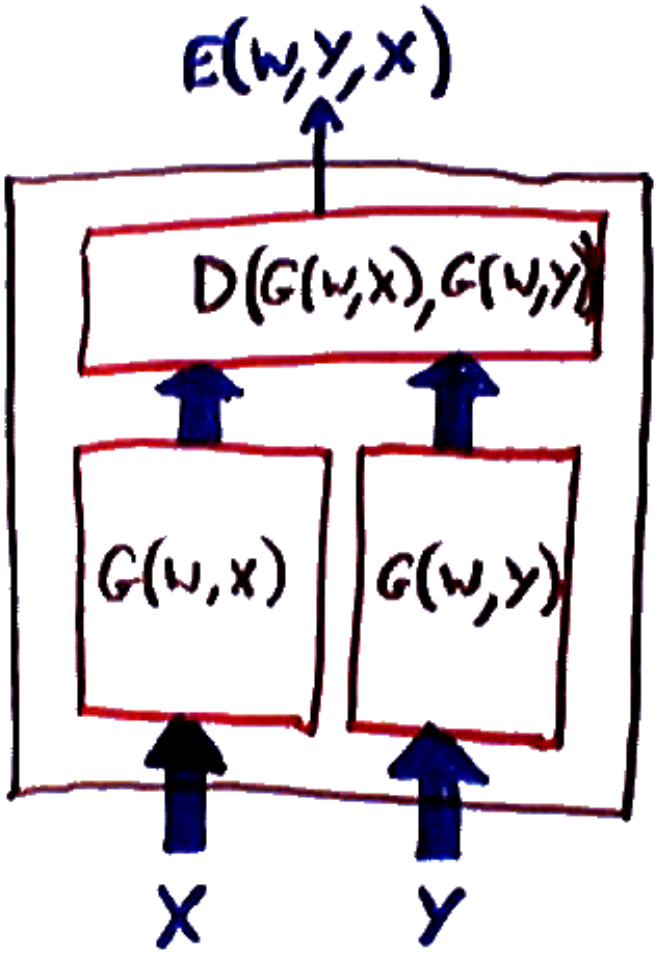
- X and Y are vectors or other entities
- Energy: $E(W, Y, X) = D(Y, G(W, X))$ where $D(Y, R)$ is a distance or dissimilarity measure.
- Best output: $\check{Y} = \min_Y E(W, Y, X) = G(W, X)$.

Examples of EBM Regressor: Linear Regression



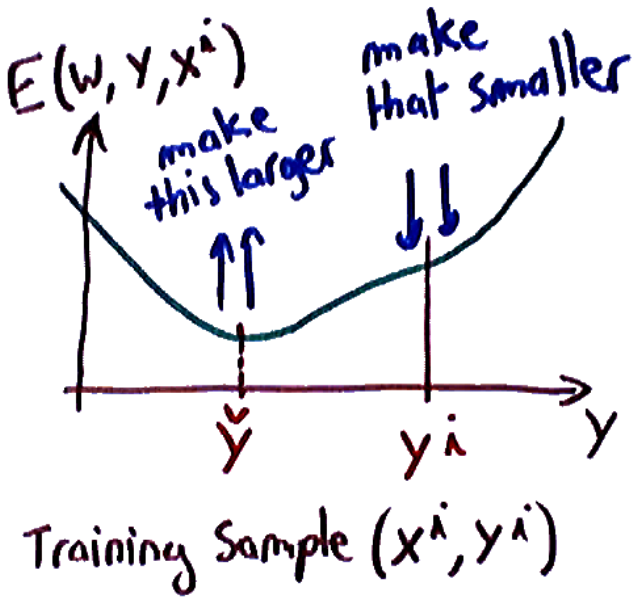
- X and Y are vectors
- Energy: $E(W, Y, X) = ||Y - W'X)||^2$.
- Best output: $\check{Y} = \min_Y E(W, Y, X) = W'X$.

Examples of EBM: Matcher



- X and Y are vectors of the same dimension.
- Energy: $E(W, Y, X) = D(G(W, Y), G(W, X))$ where $D(., .)$ is a distance or dissimilarity measure.
- Best output: $\check{Y} = \min_Y E(W, Y, X) = G^{(} - 1)(G(W, X))$.

Training Energy-Based Models



- To train an EBM, we minimize a **loss function**, which is an average over training samples of a **per-sample loss function** $L(W, Y^i, X^i)$:

$$\mathcal{L}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} L(W, Y^i, X^i)$$

- The loss function must be designed so that minimizing it with respect to W will make the machine approach the desired behavior.

To ensure this, we pick loss functions that, for a given training input X^i, will drive the energies $E(W, Y^i, X^i)$ associated with the desired output Y^i to be lower than the energies associated with all other (undesired) outputs values $E(W, Y, X^i)$ for all $Y \neq Y^i, Y \in \{Y\}$.

Form of the Loss Function

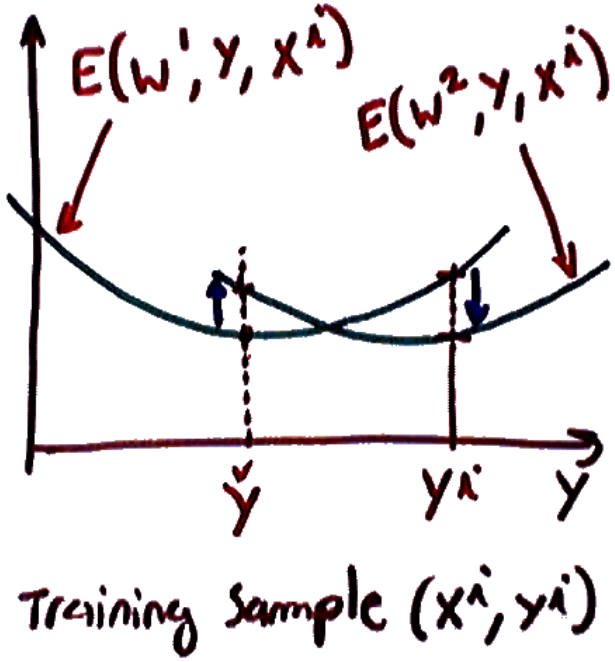
- We assume that the per-sample loss function $L(W, Y^i, X^i)$ has a lower bound over W for all Y^i, X^i.
- We assume that L depends on X^i only indirectly through the set of energies $\{E(W, Y, X^i)\ ,\ Y \in \{Y\}\}$.
- For example, if $\{Y\}$is the set of integers between 0 and $k-1$ (as would be the case for a classifier with k categories), the per-sample loss for sample (X^i, Y^i) should be of the form:

$$L(W, Y^i, X^i) = L(Y^i, E(W, 0, X^i), E(W, 1, X^i), \ldots, E(W, k-1, X^i))$$

- With this assumption, we separate the choice of the loss function from the details of the internal structure of the machine, and limit the discussion to how minimizing the loss function affects the energies.

Examples of Loss: Energy Loss

Energy Loss, the simplest of all losses: $L_{\text{energy}}(W, Y^i, X^i) = E(W, Y^i, X^i)$. This loss only works if $E(W, Y, X^i)$ has a special form which guarantess that making $E(W, Y^i, X^i)$ lower will automatically make $E(W, Y, X^i)$ for $Y \neq Y^i$ larger than the minimum.



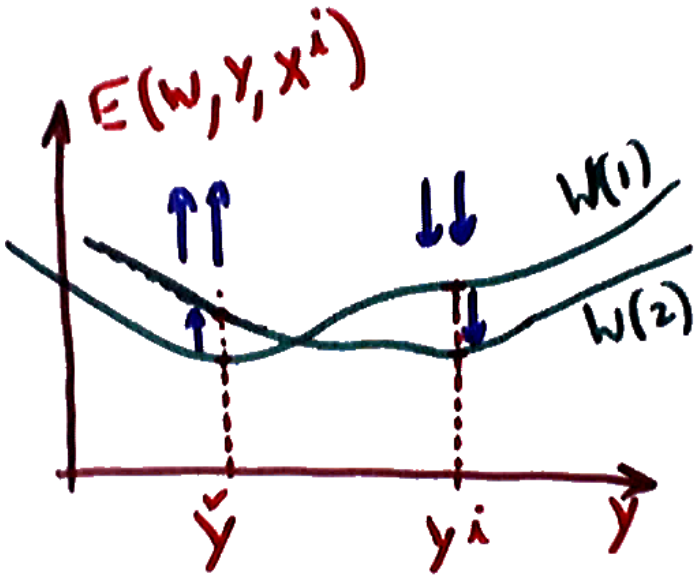
Example: if $E(W, Y, X)$ is quadratic in Y, as is the case for regression with squared error: $E(W, Y, X) = ||Y - G(W, X)||^2$,
Let $W(1)$ is the parameter before a learning update, and $W(2)$ the parameter after the learning update, and let $\check{Y} = \min_Y E(W(1), Y, X)$. Then,

$$E(W(2), Y^i, X^i) - E(W(2), \check{Y}, X^i) < E(W(1), Y^i, X^i) - E(W(1), \check{Y}, X^i)$$

Examples of Loss: Perceptron Loss

Perceptron Loss:

$$L_{\text{perceptron}}(W, Y^i, X^i) = E(W, Y^i, X^i) - \min_{Y \in \{Y\}} E(W, Y, X^i)$$



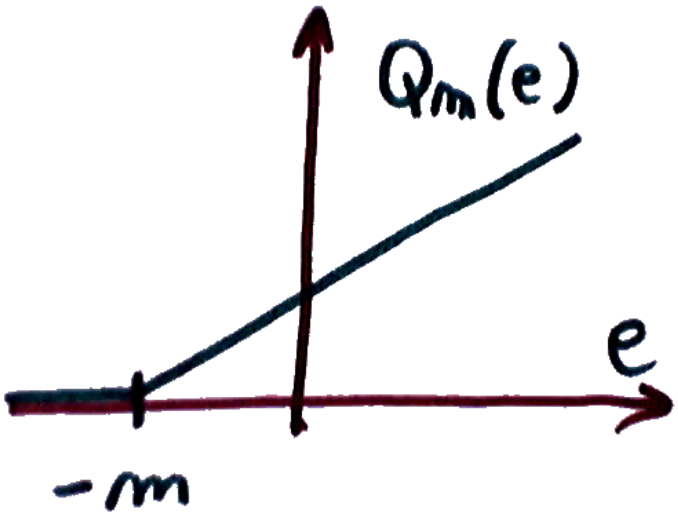
Adjust W so that $E(W, Y^i, X^i)$ gets smaller, while $\check{Y} = \min_{Y \in \{Y\}} E(W, Y, X^i)$ gets bigger (or more precisely, so that the difference decreases).
This algorithm makes no update whenever the energy of the desired Y is lower than all the others.

Examples of Loss: Margin Loss

Margin Loss: for discrete output set $\{Y\}$:

$$L_{\text{margin}}(W, Y^i, X^i) = Q_m \left(E(W, Y^i, X^i) - \min_{Y \in \{Y\}, Y \neq Y^i} E(W, Y, X^i) \right)$$

where $Q_m(e)$ is any function that is monotonically increasing for $e > -m$, where m is a constant called the **margin**.



Adjust W so that $E(W, Y^i, X^i)$ gets smaller, while all $E(W, Y, X^i)$ for which $E(W, Y, X^i) - E(W, Y^i, X^i) < m$ get bigger. This guarantees that the energy of the desired Y will be smaller than all other energies by at least m.

Examples of Loss: Log-Likelihood Loss

Log-Likelihood Loss:

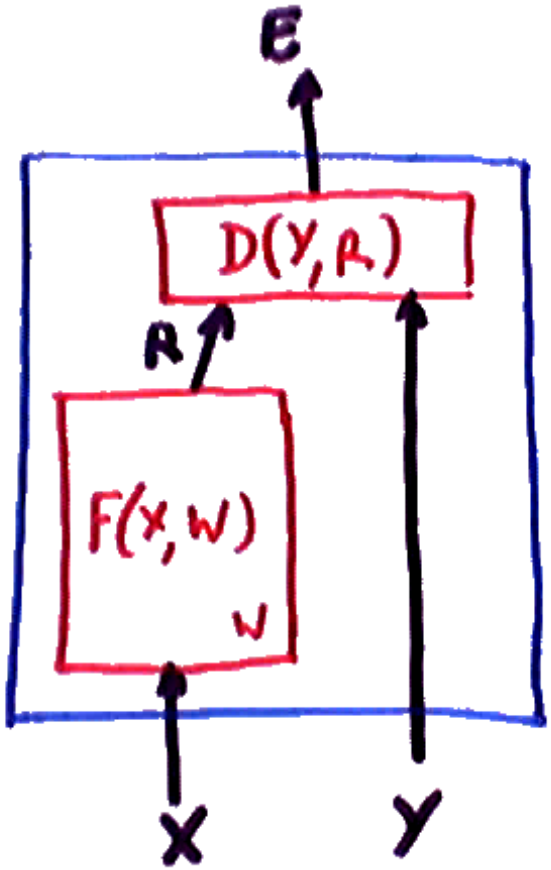
$$L_{\text{ll}}(W, Y^i, X^i) = E(W, Y^i, X^i) + \frac{1}{\beta} \log \left(\sum_{Y \in \{Y\}} \exp(-\beta E(W, Y, X^i)) \right)$$

where β is a positive constant.

- The function $\mathcal{F}_\beta(\{Y\}) = \frac{1}{\beta} \log \left(\sum_{Y \in \{Y\}} \exp(-\beta E(W, Y, X^i)) \right)$ is called the **free energy** of the ensemble $\{Y\}$ for temperature $1/\beta$.
- We define $\mathcal{Z}_\beta(\{Y\}) = \sum_{Y \in \{Y\}} \exp(-\beta E(W, Y, X^i))$ as the **partition function** of ensemble $\{Y\}$.
- Interesting property # 1: $\mathcal{F}_\beta(\{Y\}) = \frac{1}{\beta} \log \mathcal{Z}_\beta(\{Y\})$
- Interesting property # 2: $\lim_{\beta \to \infty} \mathcal{F}_\beta(\{Y\}) = \min_{Y \in \{Y\}} E(W, Y, X^i)$

For very large β, the log-likelihood loss reduces to the Perceptron loss.

Energy-Based Supervised Learning



- A *supervised* system parameterizes $E(W, Y, X)$ as follows:

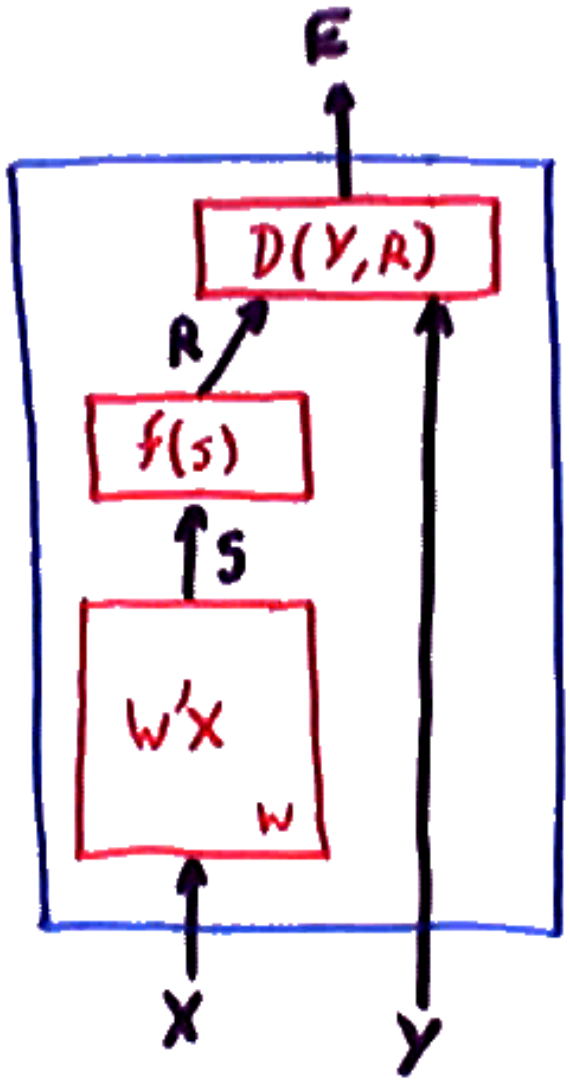
$$E(W, Y, X) = D(Y, F(W, X))$$

where $F(W, X)$ is a suitably chosen *discriminant function* parameterized by W, and D is an appropriately chosen dissimilarity measure.

- A popular example would be

$$E(Y, X, W) = ||Y - F(X, W)||^2$$

Linear Machines



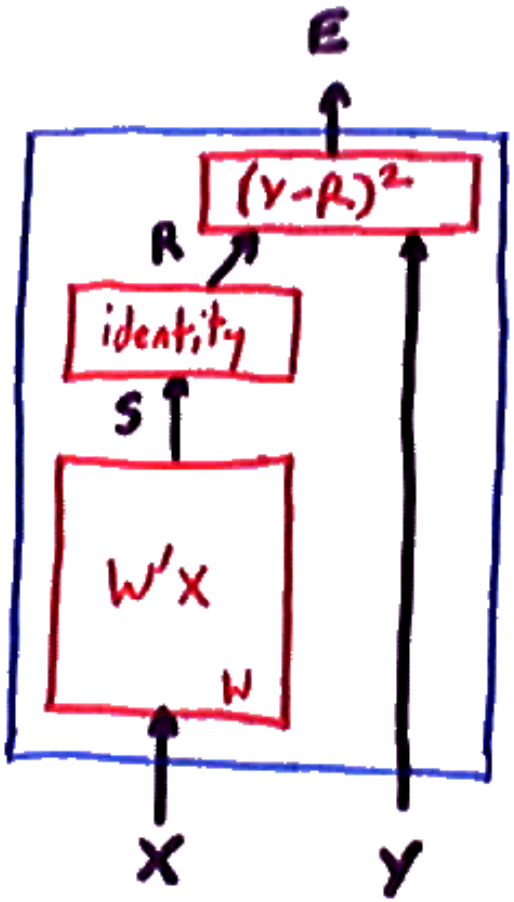
- The learning algorithms we have seen so far (perceptron, linear regression) are of that form, with the assumption that $G(W, X)$ only depends on the dot product of W and X.
- In other words, The E function of 2-class linear classifiers can be written as:

$$E(Y, X, W) = D(Y, f(W'X))$$

  where $W'X$ is the dot product of vectors W and X, and f is a monotonically increasing scalar function.
- in the following, we assume $Y = -1$ for class 1, and $Y = +1$ for class 2.

Linear Regression

Linear regression uses the Energy loss, or (equivalently) the Log-Likelihood loss.

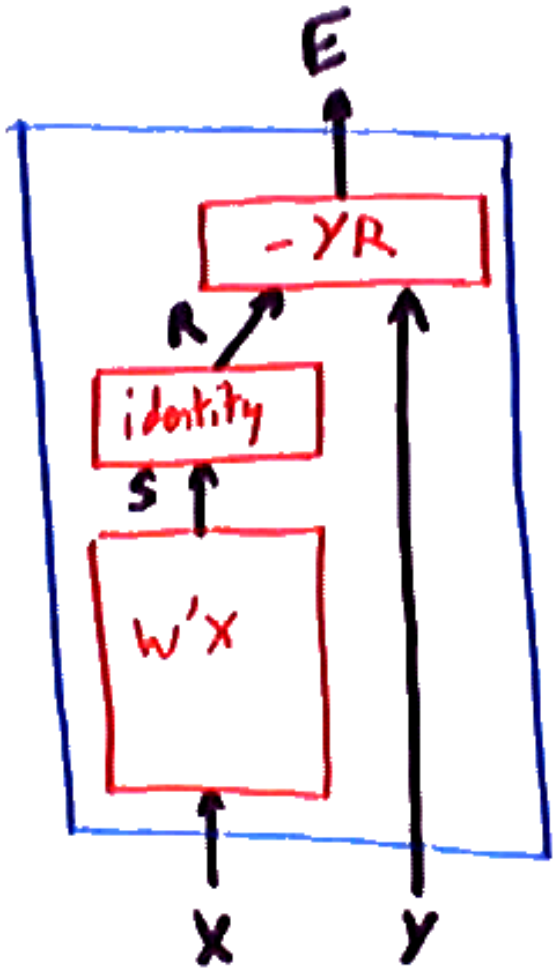


- $R = W'X$
- $E(W, Y, X) = D(Y, R) = \frac{1}{2}||Y - R||^2$
- $L(W, Y^i, X^i) = D(Y^i, W'X^i)$
- $\frac{\partial L}{\partial W} = \frac{\partial D(Y^i, R)}{\partial R} \frac{\partial R}{\partial W}$
- $\frac{\partial L}{\partial W} = \frac{\partial D(Y^i, R)}{\partial R} \frac{\partial (W'X^i)}{\partial W} = (R - Y^i)X^i$
- descent: $W \leftarrow W + \eta(Y^i - R)X^i$

Perceptron

$$L_{\text{perceptron}}(W, Y^i, X^i) = E(W, Y^i, X^i) - \min_{Y \in \{Y\}} E(W, Y, X^i)$$

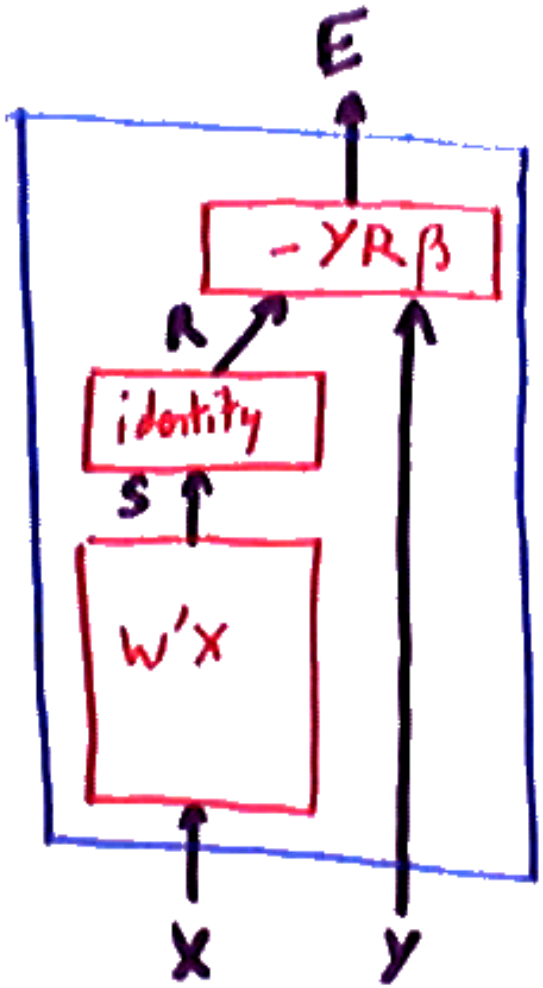
$\{Y\} = \{-1, +1\}$.



- $R = W'X$
- $E(Y, X, W) = D(Y, R) = -YR$
- $Y \in \{-1, +1\}$, hence $\min_Y -YR = -\text{sign}(R)R$ where $\text{sign}(R) = 1$ iff $R > 0$, and -1 otherwise.
- $L(W, Y^i, X^i) = -(Y^i - \text{sign}(R))R$
- $\frac{\partial L}{\partial W} = \frac{\partial -(Y^i - \text{sign}(R))R}{\partial R} \frac{\partial R}{\partial W}$
- $\frac{\partial L}{\partial W} = -(Y^i - \text{sign}(W'X^i))X^i$
- descent: $W \leftarrow W + \eta(Y^i - \text{sign}(W'X^i))X^i$

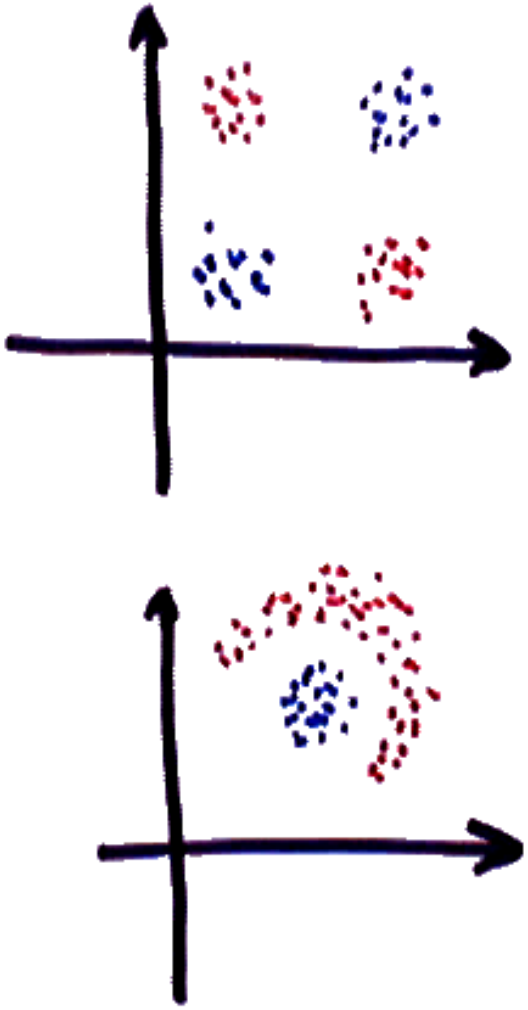
Logistic Regression

$$L_{\text{ll}}(W) = E(Y^i, X^i, W) + \log\left(\sum_{Y \in \{Y\}} \exp(-E(W, Y, X^i))\right)$$



- $R = \frac{1}{2} W'X$
- $E(Y, X, W) = D(Y, R) = -\frac{1}{2} YR = -\frac{1}{2} YW'X$
- $L(W) = \log(1 + \exp(-Y^i W' X^i))$
- $\frac{\partial L}{\partial W} = \frac{\partial D(Y^i, R)}{\partial R} \frac{\partial S}{\partial W}$
- $\frac{\partial L}{\partial W} = -\left(\frac{Y^i + 1}{2} - \frac{1}{1 + \exp(-W' X^i)}\right) X^i$
- descent: $W \leftarrow W + \eta \left(\frac{Y^i + 1}{2} - \frac{1}{1 + \exp(-W' X^i)}\right) X^i$

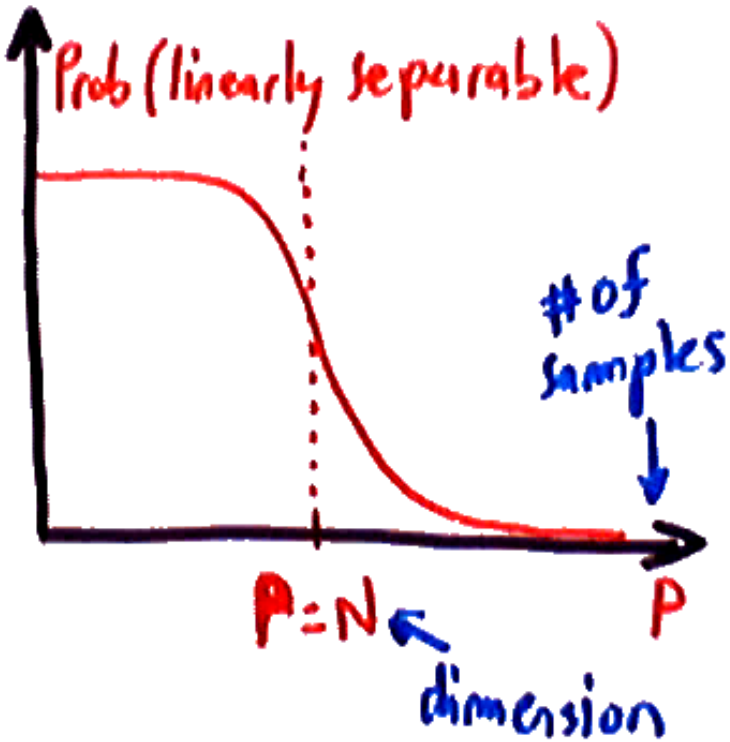
Limitations of Linear Machines



The *Linearly separable* dichotomies are the partitions that are realizable by a linear classifier (the boundary between the classes is a hyperplane).

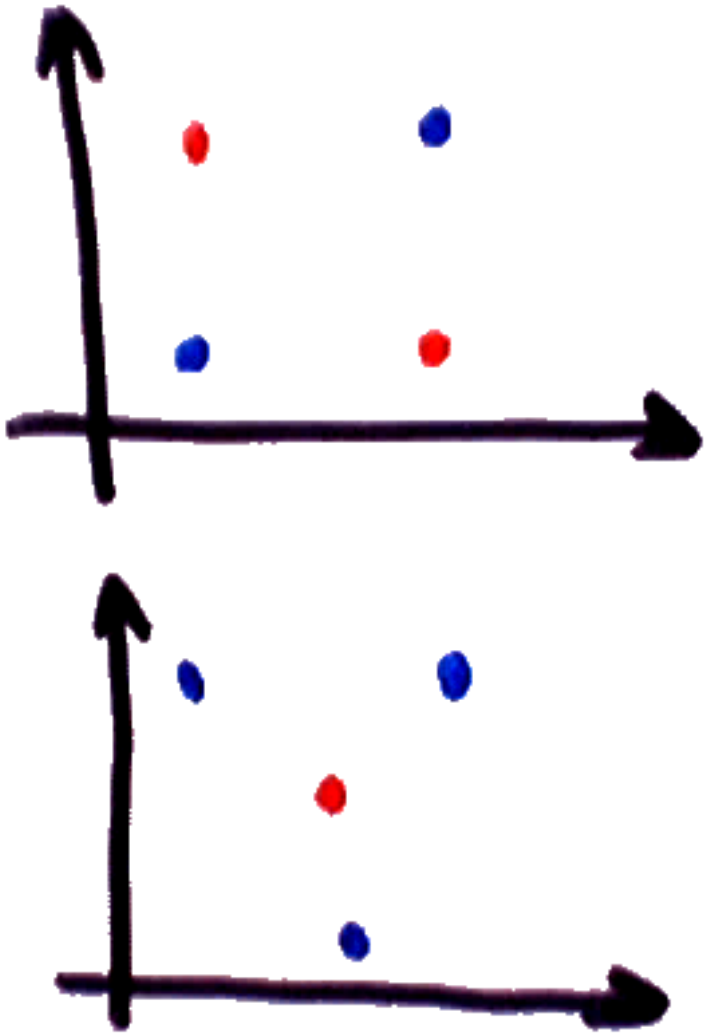
Number of Linearly Separable Dichotomies

The probability that P samples of dimension N are linearly separable goes to zero very quickly as P grows larger than N (Cover's theorem, 1966).



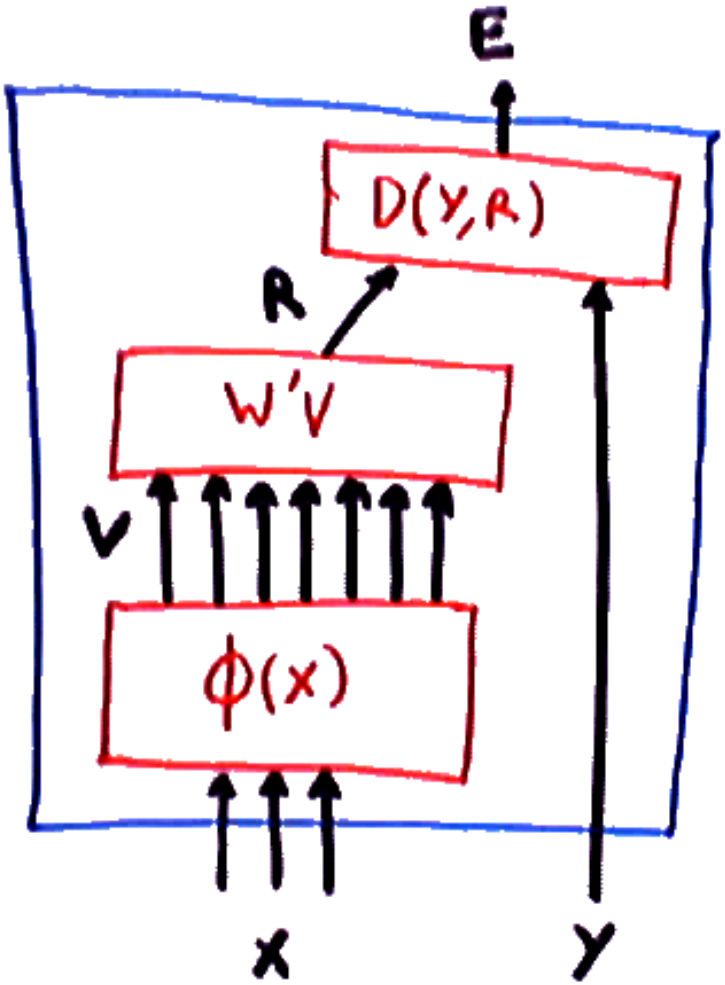
- Problem: there are 2^P possible dichotomies of P points.
- Only about N are linearly separable.
- If P is larger than N, the probability that a random dichotomy is linearly separable is very, very small.

Example of Non-Linearly Separable Dichotomies



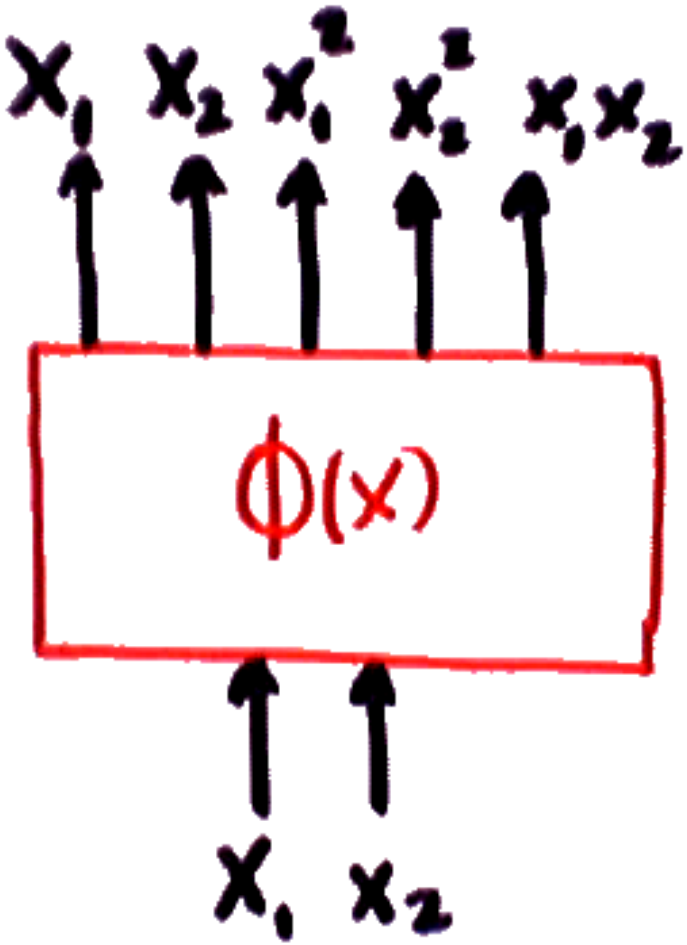
- Some seemingly simple dichotomies are not linearly separable
- Question: How do we make a given problem linearly separable?

Making N Larger: Preprocessing



- Answer 1: we make N larger by augmenting the input variables with new "features".
- we map/project X from its original N-dimensional space into a higher dimensional space where things are more likely to be linearly separable, using a vector function $\Phi(X)$.
- $E(Y, X, W) = D(Y, R)$
- $R = f(W'V)$
- $V = \Phi(X)$

Adding Cross-Product Terms

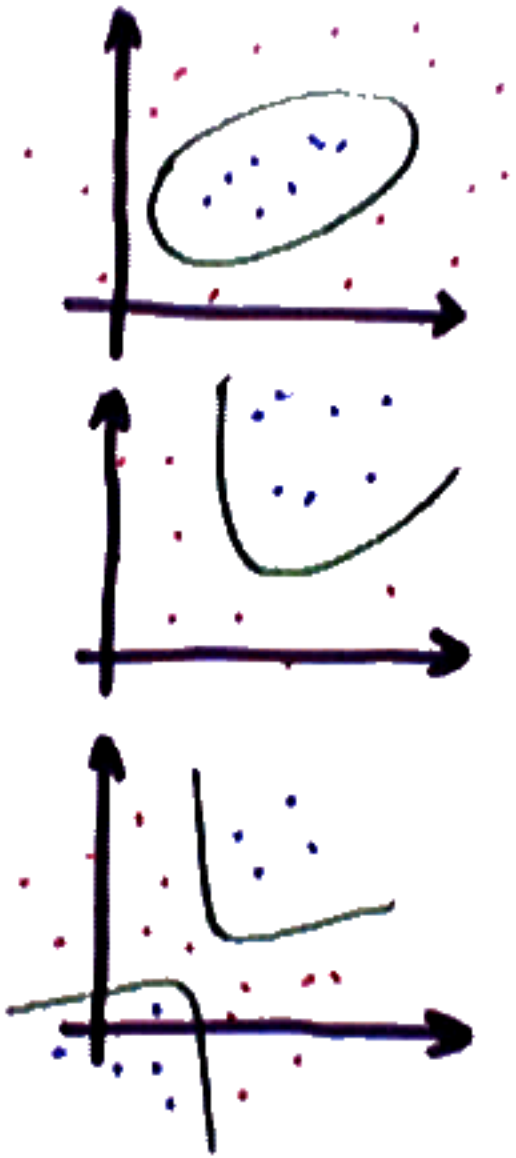


- Polynomial Expansion.
- If our original input variables are $(1, x_1, x_2)$, we construct a new *feature vector* with the following components:

$$\Phi(1, x_1, x_2) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

  i.e. we add all the cross-products of the original variables.
- we map/project X from its original N-dimensional space into a higher dimensional space with $N(N+1)/2$ dimensions.

Polynomial Mapping



- Many new functions are now separable with the new architecture.
- With cross-product features, the family of class boundaries in the original space is the conic sections (ellipse, parabola, hyperbola).
- to each possible boundary in the original space corresponds a linear boundary in the transformed space.
- Because this is essentially a linear classifier with a preprocessing, we can use standard linear learning algorithms (perceptron, linear regression, logistic regression...).

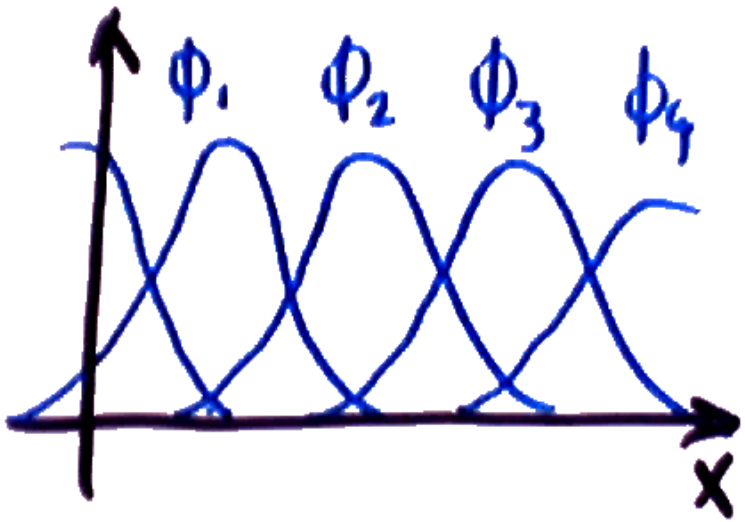
o

Problems with Polynomial Mapping

- We can generalize this idea to higher degree polynomials, adding cross-product terms with 3, 4 or more variables.
- Unfortunately, the number of terms is the number of combinations d choose N, which grows like N^d, where d is the degree, and N the number of original variables.
- In particular, the number of free parameters that must be learned is also of order N^d.
- This is impractical for large N and for $d > 2$.
- Example: handwritten digit recognition (16x16 pixel images). Number of variables: 256. Degree 2: 32,896 variables. Degree 3: 2,796,160. Degre 4: 247,460,160.....

Next Idea: Tile the Space

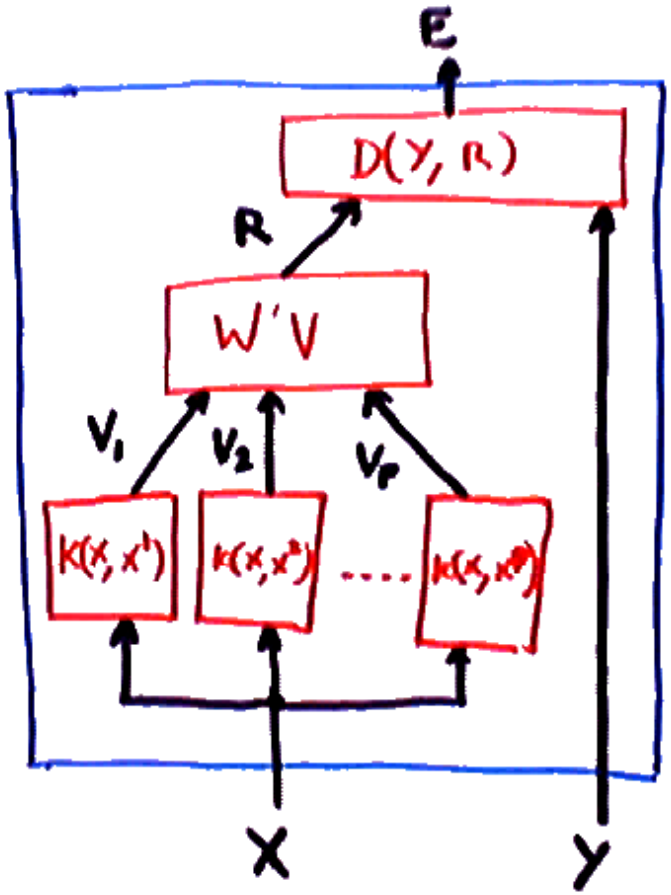
place a number of equally-spaced "bumps" that cover the entire input space.



- For classification, the bumps can be Gaussians
- For regression, the basis functions can be wavelets, sine/cosine, splines (pieces of polynomials)....
- problem: this does not work with more than a few dimensions.
- The number of bumps necessary to cover an N dimensional space grows exponentially with N.

Sample-Centered Basis Functions (Kernels)

Place the center of a basis function around each training sample. That way, we only spend resources on regions of the space where we actually have training samples.

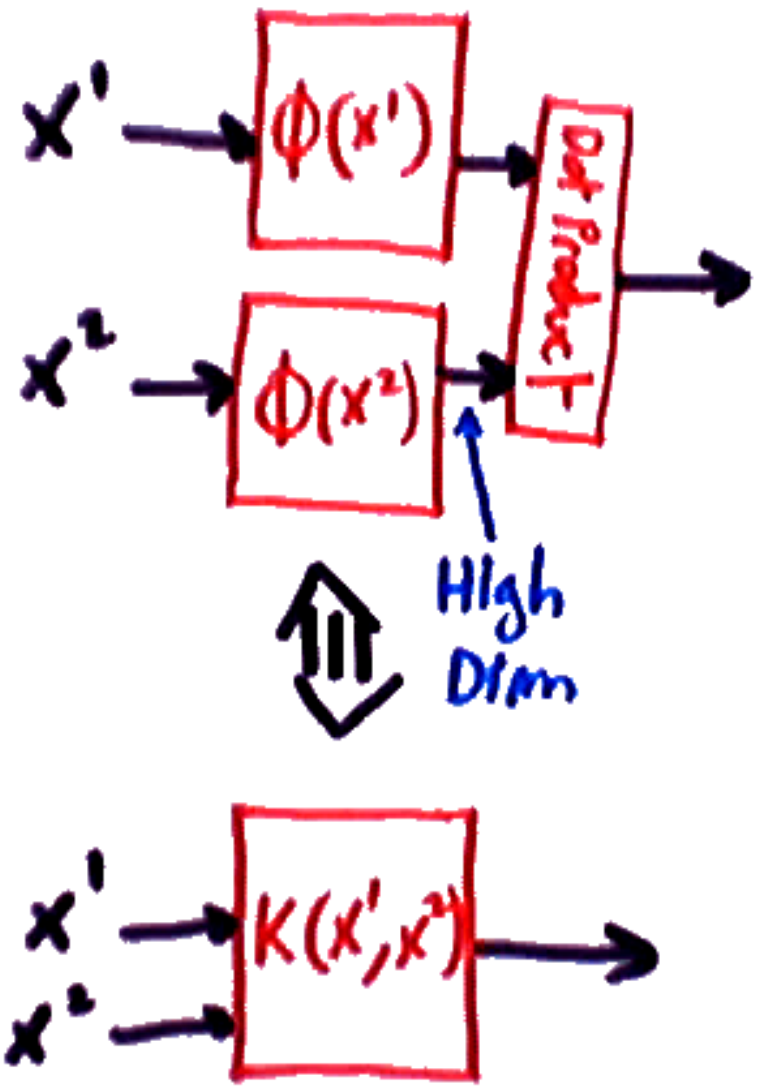


- Discriminant function:

$$f(X, W) = \sum_{k=1}^{k=P} W_k K(X, X^k)$$

- $K(X, X')$ often takes the form of a *radial basis function*:
  $K(X, X') = \exp(b||X - X'||^2)$ or a polynomial $K(X, X') = (X.X' + 1)^m$
- This is a very common architecture, which can be used with a number of energy functions.
- In particular, this is the architecture of the so-called Support Vector Machine (SVM), but the energy function of the SVM is a bit special. We will study it later in the course.

The Kernel Trick



- If the kernel function $K(X, X')$ verifies the *Mercer conditions*, then there exist a mapping Φ, such that $\Phi(X).\Phi(X') = K(X, X')$.
- The Mercer conditions are that K must be symmetric, and must be positive definite (i.e $K(X, X)$ must be positive for all X).
- In other words, if we want to map our X into a high-dimensional space (so as to make them linearly separable), and all we have to do in that space is compute dot products, we can take a shortcut and simply compute $K(X^1, X^2)$ without going through the high-dimensional space.
- This is called the "kernel trick". It is used in many so-called Kernel-based methods, including Support Vector Machines.

Examples of Kernels

- Quadratic kernel: $\Phi(X) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x2, x_1^2, x_2^2)$ then

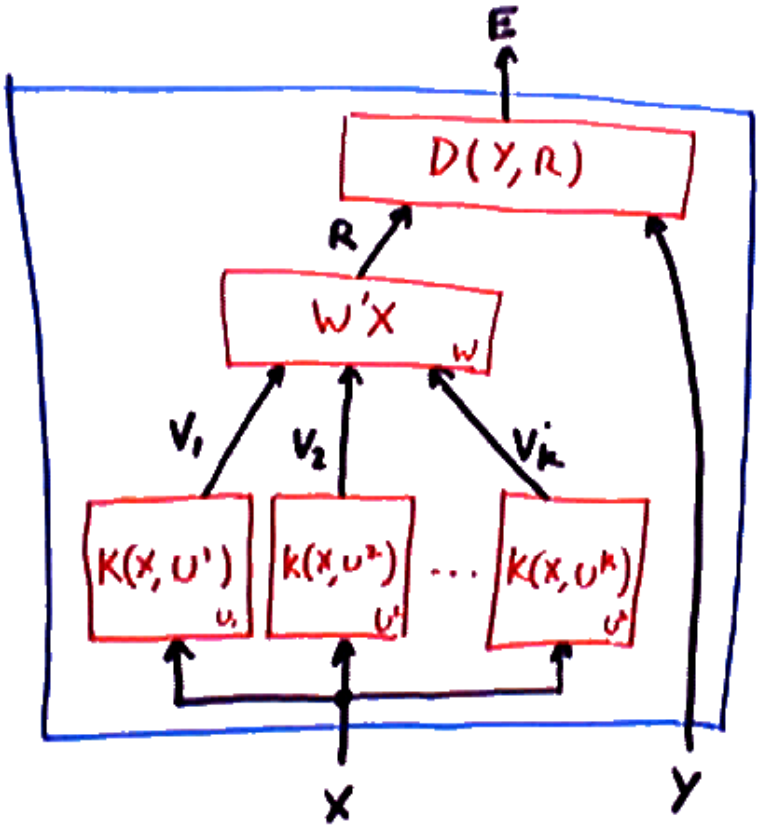
$$K(X, X') = \Phi(X).\Phi(X') = (X.X' + 1)^2$$

- Polynomial kernel: this generalizes to any degree d. The kernel that corresponds to $\Phi(X)$ bieng a polynomial of degree d is
  $K(X, X') = \Phi(X).\Phi(X') = (X.X' + 1)^d$.
- Gaussian Kernel:

$$K(X, X') = \exp(-b||X - X'||^2)$$

  This kernel, sometimes called the Gaussian Radial Basis Function, is very commonly used.

Sparse Basis Functions

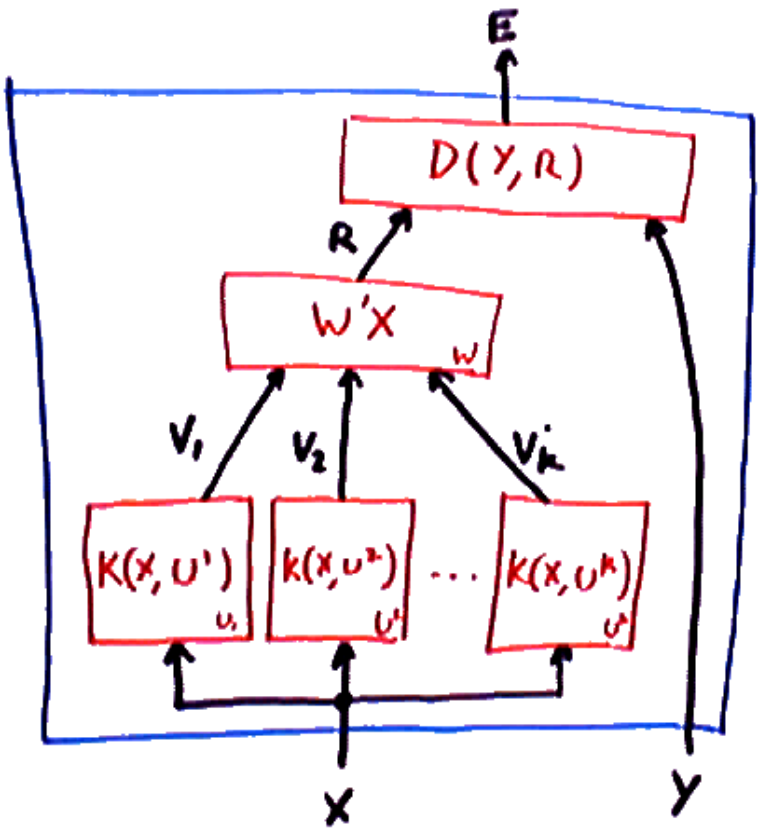


- Place the center of a basis function around areas containing training samples.
- Idea 1: use an unsupervised clustering algorithm (such as K-means or mixture of Gaussians) to place the centers of the basis functions in areas of high sample density.
- Idea 2: adjust the basis function centers through gradient descent in the loss function.

The discriminant function F is:

$$F(X, W, U^1, \ldots, U^K) = \sum_{k=1}^{k=K} W_k K(X, U^k)$$

Supervised Adjustment of the RBF Centers



- To adjust the U's we must compute the partial derivatives of L with respect to the U's.
- by posing and $V_k = K(X, U^k)$, and $R = \sum_{k=1}^{k=K} W_k V_k$ we can write:

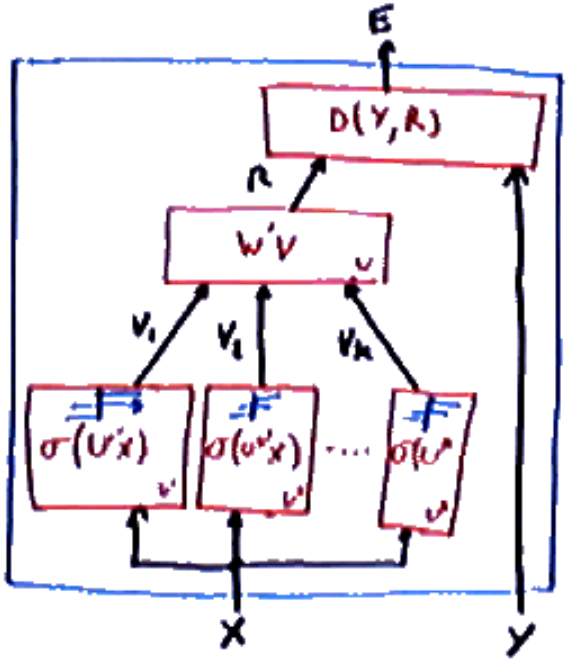
$$\frac{\partial L(W)}{\partial U^j} = \frac{\partial L(W)}{\partial R} \frac{\partial R}{\partial V_j} \frac{\partial V_j}{\partial U_j}$$

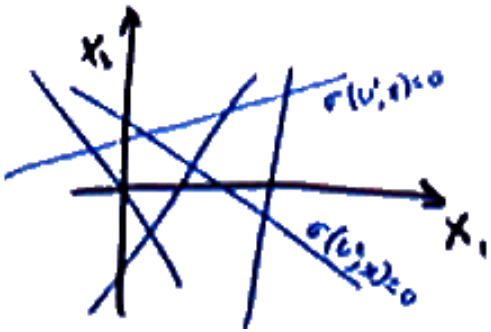
- Which comes down to:

$$\frac{\partial L(W)}{\partial U^j} = \frac{\partial L(W)}{\partial R} W_j \frac{\partial K(X, U_j)}{\partial U_j}$$

Now, there is a *very general method* for dealing with those multiple applications of chain rule. We will see that next time.

Other Idea: Random Directions

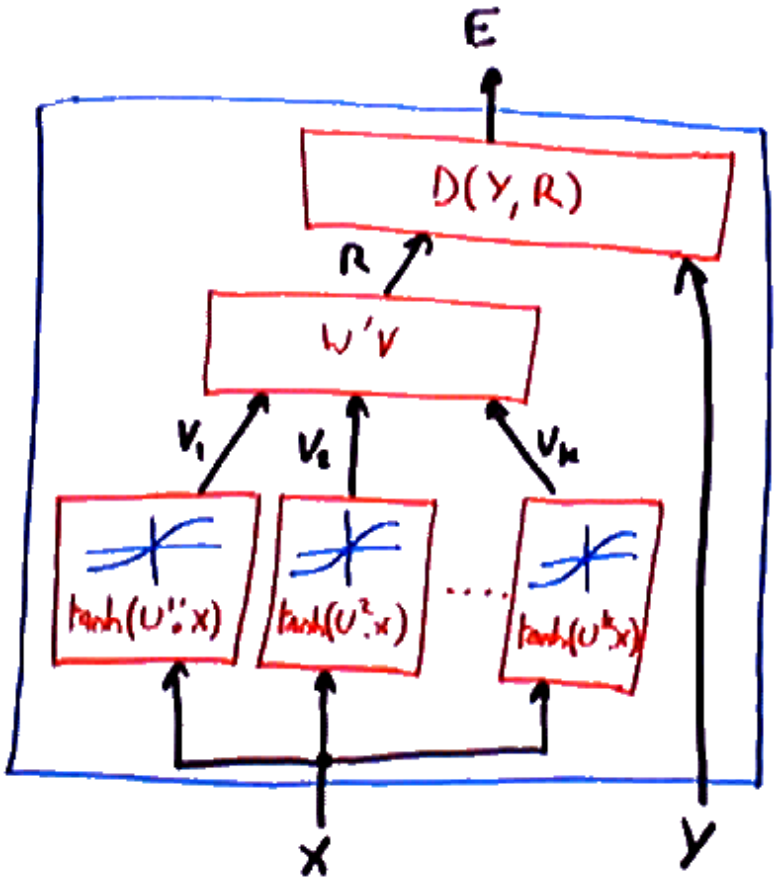




- Partition the space in lots of little domains by randomly placing lits of hyperplanes.
- Use many variables of the type $q(W^k X)$, where q is the threshold function (or some other squashing function) and W_k is a randomly picked vector.
- This is the original Perceptron.
- Without the non-linearity, the whole system would be linear (product of linear operations), and therefore would be no more powerful than a linear classifier.
- problem: a bit of a wishful thinking, but it works occasionally.

Neural Net with a Single Hidden Layer

A particularly interesting type of basis function is the sigmoid unit: $V_k = \tanh(U'^k X)$



- a network using these basis functions, whose output is $R = \sum_{k=1}^{k=K} W_k V_k$ is called a *single hidden-layer neural network*.
- Similarly to the RBF network, we can compute the gradient of the loss function with respect to the U^k:

$$\frac{\partial L(W)}{\partial U^j} = \frac{\partial L(W)}{\partial R} W_j \frac{\partial tanh(U'_j X)}{\partial U_j}$$

$$= \frac{\partial L(W)}{\partial R} W_j tanh'(U'_j X) X'$$

Any well-behaved function can be approximated as close as we wish by such networks (but K might be very large).