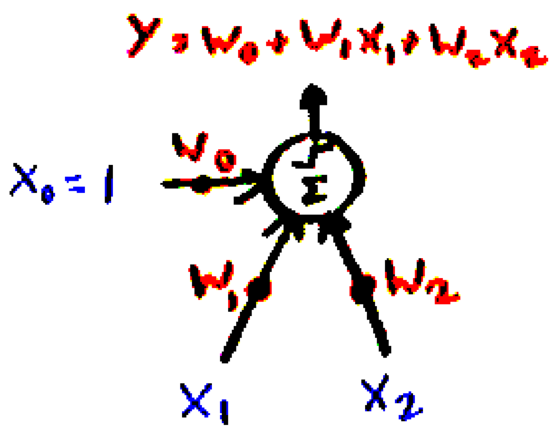
Energy-Based Supervised Learning. Structured Output Models

Yann LeCun,
The Courant Institute of Mathematical Sciences
New York University

http://yann.lecun.com
http://www.cs.nyu.edu/~yann

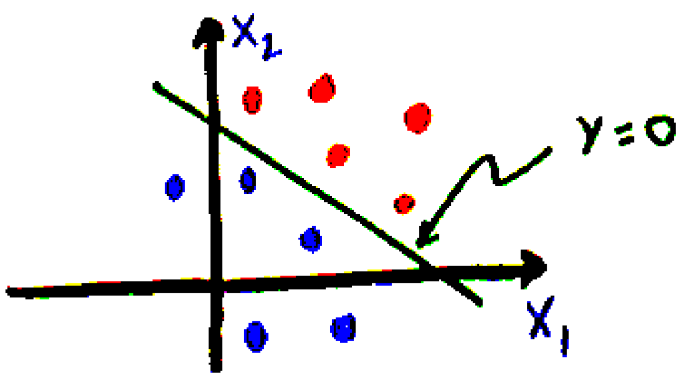
The Linear Classifier

Historically, the Linear Classifier was designed as a highly simplified model of the neuron (McCulloch and Pitts 1943, Rosenblatt 1957):



$$y = f(\sum_{i=0}^{i=N} w_i x_i)$$

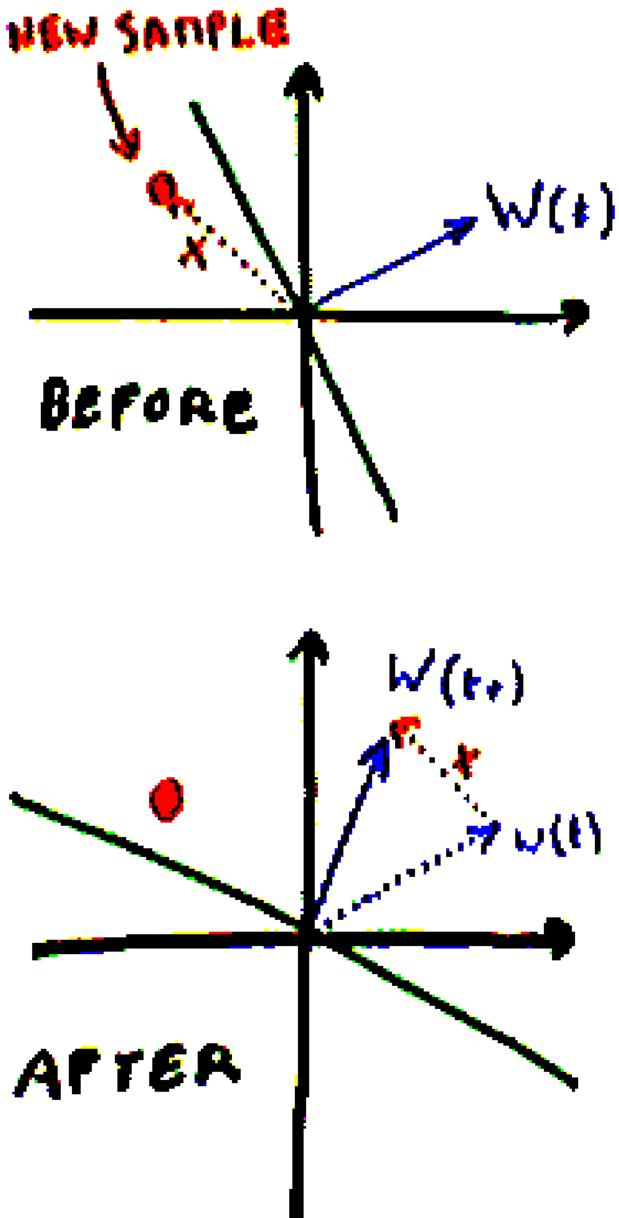
With f is the threshold function: $f(z) = 1$ iff $z > 0$, $f(z) = -1$ otherwise. x_0 is assumed to be constant equal to 1, and w_0 is interpreted as a bias.
In vector form: $W = (w_0, w_1 .... w_n), X = (1, x_1 ... x_n)$:



$$y = f(W'X)$$

The hyperplane $W'X = 0$ partitions the space in two categories. W is orthogonal to the hyperplane.

A Simple Idea for Learning: Error Correction



We have a **training set** $\mathcal{S}$ consisting of P input-output pairs: $\mathcal{S} = (X^1, y^1), (X^2, y^2), ...(X^P, y^P)$.

A very simple algorithm:

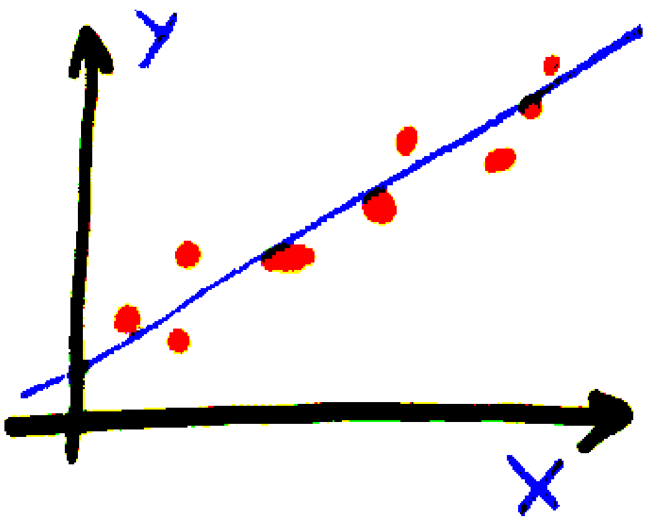
- show each sample in sequence repetitively
- if the output is correct: do nothing
- if the output is -1 and the desired output +1: increase the weights whose inputs are positive, decrease the weights whose inputs are negative.
- if the output is +1 and the desired output -1: decrease the weights whose inputs are positive, increase the weights whose inputs are negative.

More formally, for sample p:

$$w_i(t+1) = w_i(t) + (y_i^p - f(W'X^p))x_i^p$$

This simple algorithm is called the Perceptron learning procedure (Rosenblatt 1957).

Regression, Mean Squared Error



Regression or function approximation is finding a function that approximates a set of samples as well as possible.

Classic example: linear regression. We are given a **training set** $\mathcal{S}$ of input/output pairs $\mathcal{S} = \{(X^1, y^1), (X^2, y^2)....(X^P, y^P)\}$, and we must find the parameters of a linear function that best predicts the y's from the X's in the least square sense. In other words, we must find the parameter W that minimizes the quadratic **loss function** $\mathcal{L}(W, \mathcal{S})$:

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

where the **per-sample loss function** $L(W, y^i, X^i)$ is defined as:

$$L(W, y^i, X^i) = \frac{1}{2}(y^i - W'X^i)^2$$

Regression: Solution

$$\mathcal{L}(W) = \frac{1}{P}\sum_{i=1}^{P}\frac{1}{2}(y^i - W'X^i)^2$$

$$W^* = \text{argmin}_W \mathcal{L}(W) = \text{argmin}_W \frac{1}{P}\sum_{i=1}^{P}\frac{1}{2}(y^i - W'X^i)^2$$

At the solution, W satisfies the extremality condition:

$$\frac{d\mathcal{L}(W)}{dW} = 0$$

$$\frac{d\left[\frac{1}{P}\sum_{i=1}^{P}\frac{1}{2}(y^i - W'X^i)^2\right]}{dW} = 0$$

$$\sum_{i=1}^{P}\frac{d\left[\frac{1}{2}(y^i - W'X^i)^2\right]}{dW} = 0$$

Regression: Solution

The gradient of $\mathcal{L}$(W) is:

$$\frac{d\mathcal{L}(W)}{dW} = \sum_{i=1}^{P} \frac{d\left[\frac{1}{2}(y^i - W'X^i)^2\right]}{dW} = \sum_{i=1}^{P} -(y^i - W'X^i)X^{i'}$$

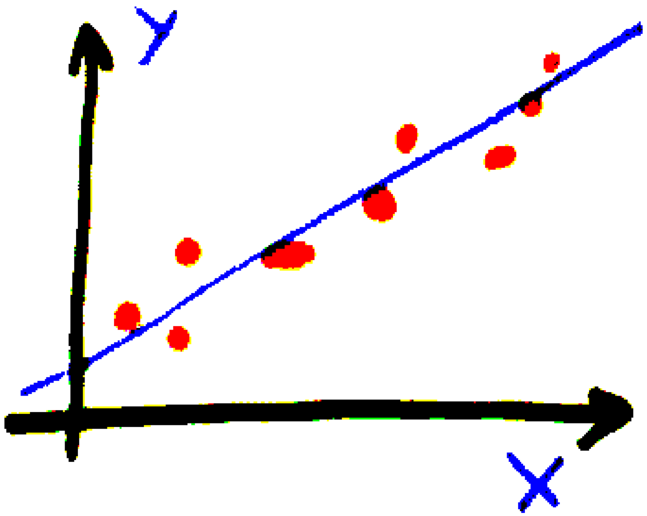
The extremality condition becomes:

$$\frac{1}{P}\sum_{i=1}^{P} -(y^i - W'X^i)X^{i'} = 0$$

Which we can rewrite as:

$$\left[\sum_{i=1}^{P} y^i X^i\right] - \left[\sum_{i=1}^{P} X^i X^{i'}\right] W = 0$$

Regression: Direct Solution



$$\sum_{i=1}^{P} y^i X^i - [\sum_{i=1}^{P} X^i X^{i\prime}]W = 0$$

Can be written as:

$$[\sum_{i=1}^{P} X^i X^{i\prime}]W = \sum_{i=1}^{P} y^i X^i$$

This is a linear system that can be solved with a number of traditional numerical methods (although it may be ill-conditioned or singular).

If the **covariance matrix** $A = \sum_{i=1}^{P} X^i X^{i\prime}$ is non singular, the solution is:

$$W^* = \left[\sum_{i=1}^{P} X^i X^{i\prime}\right]^{-1} \sum_{i=1}^{P} y^i X^i$$

Regression: Iterative Solution

Gradient-based minimization: $W(t+1) = W(t) - \eta \frac{d\mathcal{L}(W)}{dW}$
where η is a well chosen coefficient (often a scalar, sometimes diagonal matrix with positive entries, occasionally a full symmetric positive definite matrix).
The k-th component of the gradient of the quadratic loss $\mathcal{L}(W)$ is:

$$\frac{\partial \mathcal{L}(W)}{\partial w_k} = \sum_{i=1}^{P} -(y^i - W(t)'X^i)x_k^i$$

If η is a scalar or a diagonal matrix, we can write the udpate equation for a single component of W: $w_k(t+1) = w_k(t) + \eta \sum_{i=1}^{P}(y^i - W(t)'X^i)x_k^i$
This update rules converges for well-chosen, small-enough values of η (more on this later).

Regression, Online/Stochastic Gradient

Online gradient descent, aka Stochastic Gradient:

$$W(t+1) = W(t) - \eta \frac{d(W, Y^i, X^i)}{dW}$$

$$w_k(t+1) = w_k(t) + \eta(t)(y^i - W(t)'X^i)x_k^i$$

No sum! The average gradient is replaced by its instantaneous value.
This is called **stochastic gradient descent**. In many practical situation it is **enormously faster** than batch gradient.
But the convergence analysis of this method is very tricky.
One condition for convergence is that $\eta(t)$ must be decreased according to a schedule such that $\sum_t \eta(t)^2$ converges while $\sum_t \eta(t)$ diverges.
One possible such sequence is $\eta(t) = \eta_0/t$.
We can also use second-order methods, but we will keep that for later.

Linear Machines: Regression with Mean Square

Linear Regression, Mean Square Loss:

- decision rule: $y = W'X$
- loss function: $L(W, y^i, X^i) = \frac{1}{2}(y^i - W'X^i)^2$
- gradient of loss: $\frac{\partial L(W,y^i,X^i)}{\partial W}' = -(y^i - W(t)'X^i)X^i$
- update rule: $W(t+1) = W(t) + \eta(t)(y^i - W(t)'X^i)X^i$
- direct solution: solve linear system $[\sum_{i=1}^{P} X^i X^{i\prime}]W = \sum_{i=1}^{P} y^i X^i$

Linear Machines: Perceptron

Perceptron:

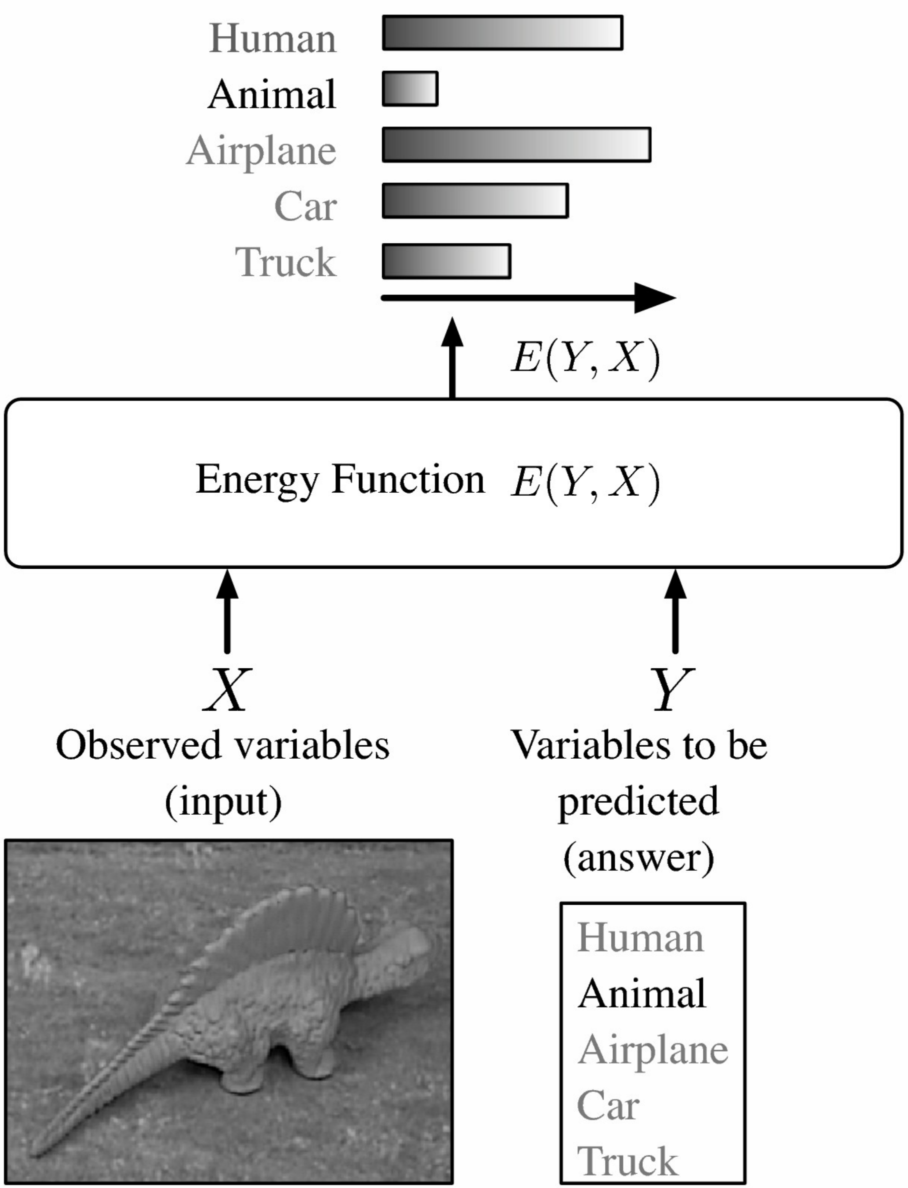
- decision rule: $y = F(W'X)$ (F is the threshold function)
- loss function: $L(W, y^i, X^i) = (F(W'X^i) - y^i)W'X^i$
- gradient of loss: $\frac{\partial L(W, y^i, X^i)}{\partial W}' = -(y^i - F(W(t)'X^i))X^i$
- update rule: $W(t+1) = W(t) + \eta(t)(y^i - F(W(t)'X^i))X^i$
- direct solution: find W such that $-y^i F(W'X^i) < 0 \quad \forall i$

Linear Machines: Logistic Regression

Logistic Regression, Negative Log-Likelihood Loss function:

- decision rule: $y = F(W'X)$, with $F(a) = \tanh(a) = \frac{1-\exp(a)}{1+\exp(a)}$ (sigmoid function).
- loss function: $L(W, y^i, X^i) = 2\log(1 + \exp(-y^i W' X^i))$
- gradient of loss: $\frac{\partial L(W,y^i,X^i)}{\partial W}' = -\left(Y^i - F(W'X)\right) X^i$
- update rule: $W(t+1) = W(t) + \eta(t)(y^i - F(W(t)'X^i))X^i$

Energy-Based Model for Decision-Making

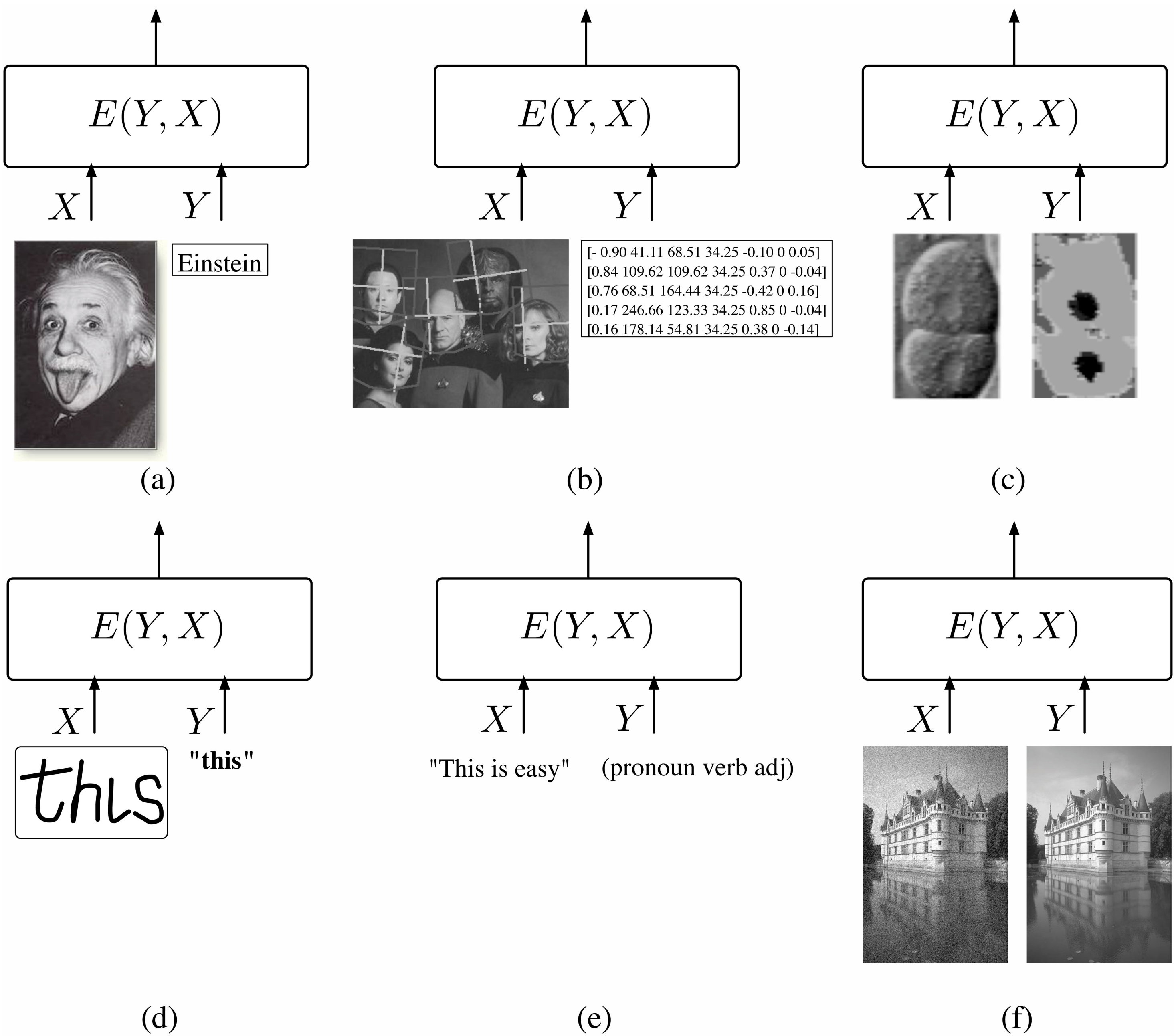


- **Model:** **Measures the compatibility between an observed variable X and a variable to be predicted Y through an energy function E(Y,X).**

$$Y^* = \text{argmin}_{Y \in \mathcal{Y}} E(Y,X).$$

- **Inference:** **Search for the Y that minimizes the energy within a set** $\mathcal{Y}$
- **If the set has low cardinality, we can use exhaustive search.**

Complex Tasks: Inference is non-trivial



- **When the cardinality or dimension of Y is large, exhaustive search is impractical.**
- **We need to use a "smart" inference procedure: min-sum, Viterbi, ...**

What Questions Can a Model Answer?

1. Classification & Decision Making:

- "which value of Y is most compatible with X?"
- Applications: Robot navigation,.....
- Training: give the lowest energy to the correct answer

2. Ranking:

- "Is Y1 or Y2 more compatible with X?"
- Applications: Data-mining....
- Training: produce energies that rank the answers correctly

3. Detection:

- "Is this value of Y compatible with X"?
- Application: face detection....
- Training: energies that increase as the image looks less like a face.

4. Conditional Density Estimation:

- "What is the conditional distribution P(Y|X)?"
- Application: feeding a decision-making system
- Training: differences of energies must be just so.

Decision-Making versus Probabilistic Modeling

Energies are uncalibrated

- The energies of two separately-trained systems cannot be combined
- The energies are uncalibrated (measured in arbitrary untis)

How do we calibrate energies?

- We turn them into probabilities (positive numbers that sum to 1).
- Simplest way: Gibbs distribution
- Other ways can be reduced to Gibbs by a suitable redefinition of the energy.

$$P(Y|X) = \frac{e^{-\beta E(Y,X)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(y,X)}},$$

Partition function

Inverse temperature

Architecture and Loss Function

- **Family of energy functions** $\mathcal{E} = \{E(W, Y, X) : \ W \in \mathcal{W}\}.$
- **Training set** $\mathcal{S} = \{(X^i, Y^i) : i = 1 \dots P\}$
- **Loss functional / Loss function** $\mathcal{L}(E, \mathcal{S})$ $\mathcal{L}(W, \mathcal{S})$
  - Measures the quality of an energy function
- **Training** $W^* = \min_{W \in \mathcal{W}} \mathcal{L}(W, \mathcal{S}).$
- **Form of the loss functional**
  - invariant under permutations and repetitions of the samples

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

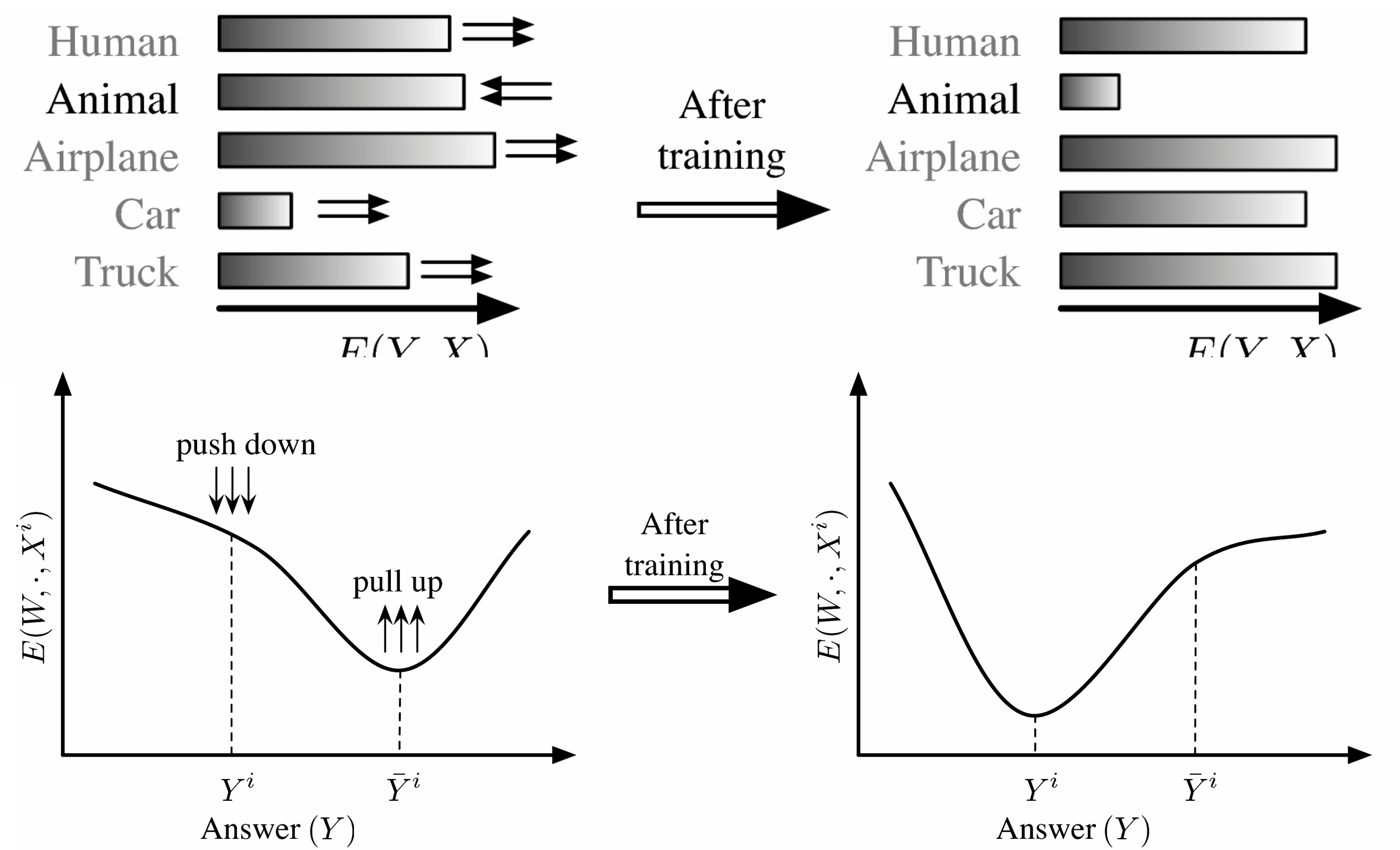
Per-sample loss

Desired answer

Energy surface for a given Xi as Y varies

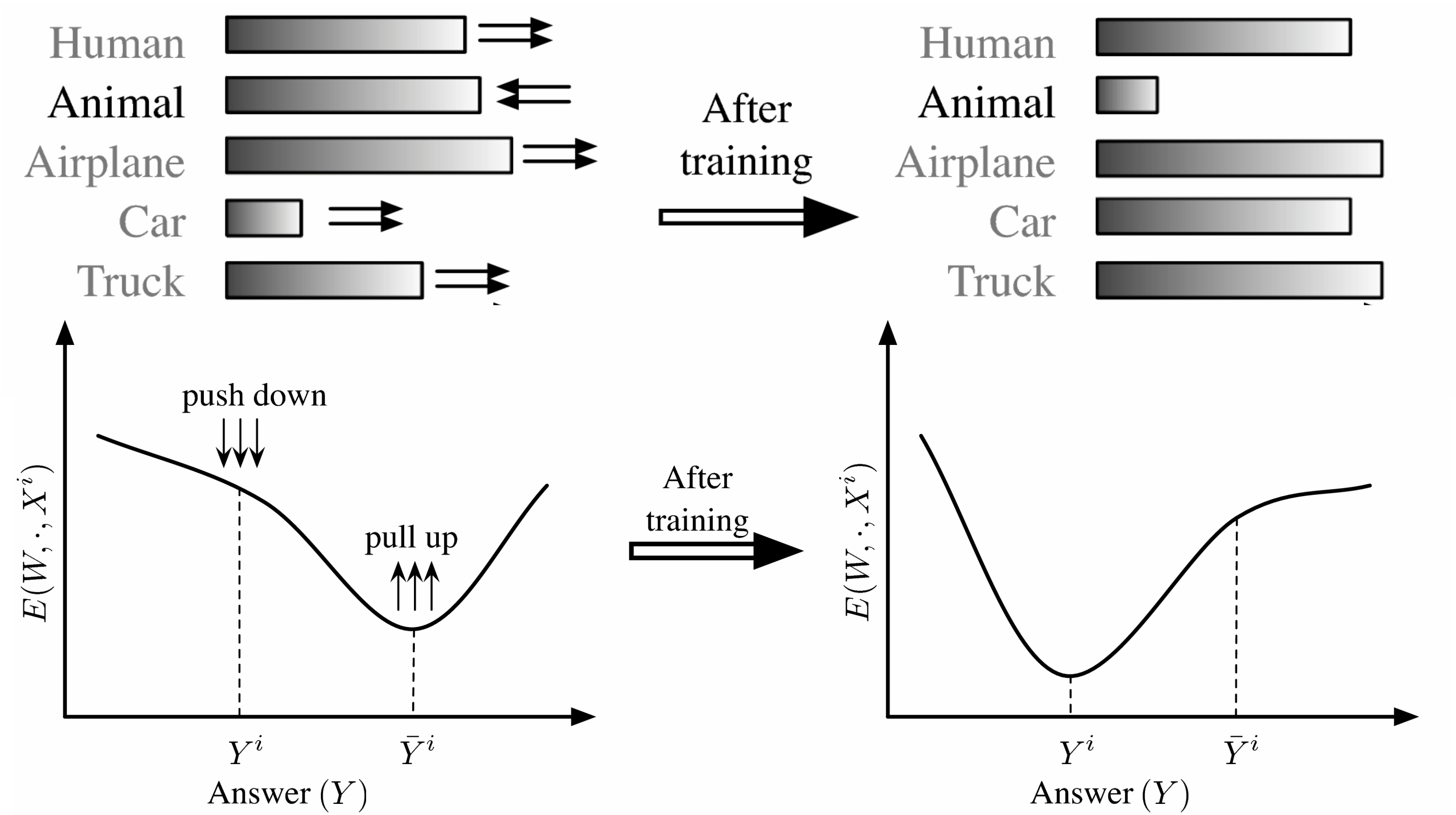
Regularizer

Designing a Loss Functional



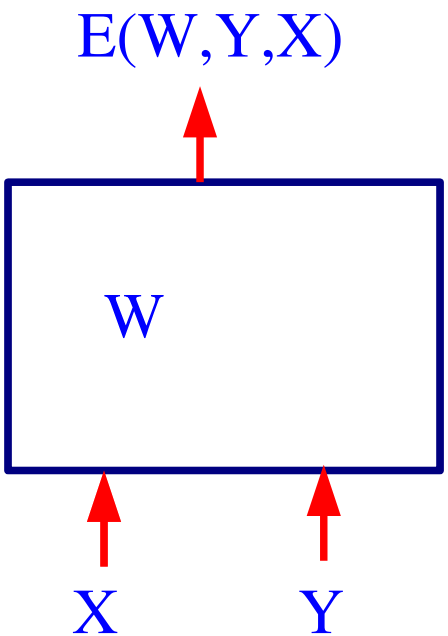
- **Correct answer has the lowest energy -> LOW LOSS**
- **Lowest energy is not for the correct answer -> HIGH LOSS**

Designing a Loss Functional



- **Push down on the energy of the correct answer**
- **Pull up on the energies of the incorrect answers, particularly if they are smaller than the correct one**

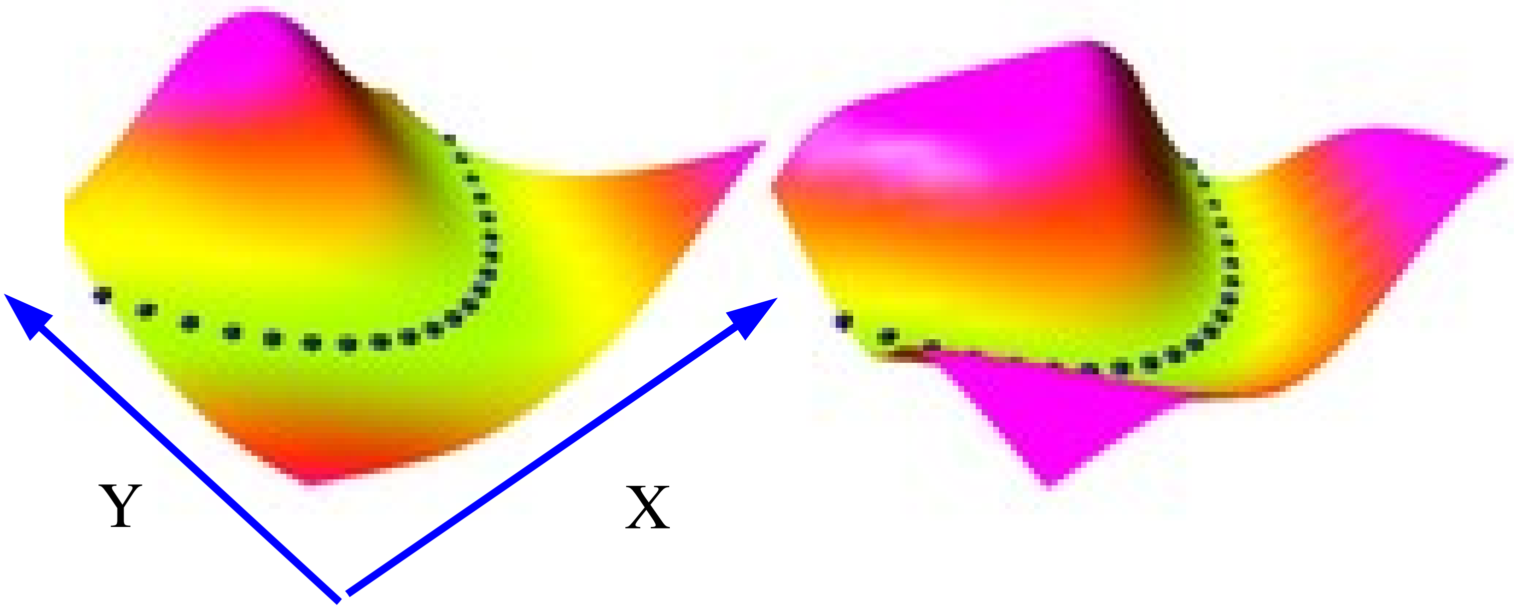
Architecture + Inference Algo + Loss Function = Model



- **1. Design an architecture:** a particular form for E(W,Y,X).
- **2. Pick an inference algorithm for Y:** MAP or conditional distribution, belief prop, min cut, variational methods, gradient descent, MCMC, HMC.....
- **3. Pick a loss function:** in such a way that minimizing it with respect to W over a training set will make the inference algorithm find the correct Y for a given X.
- **4. Pick an optimization method.**

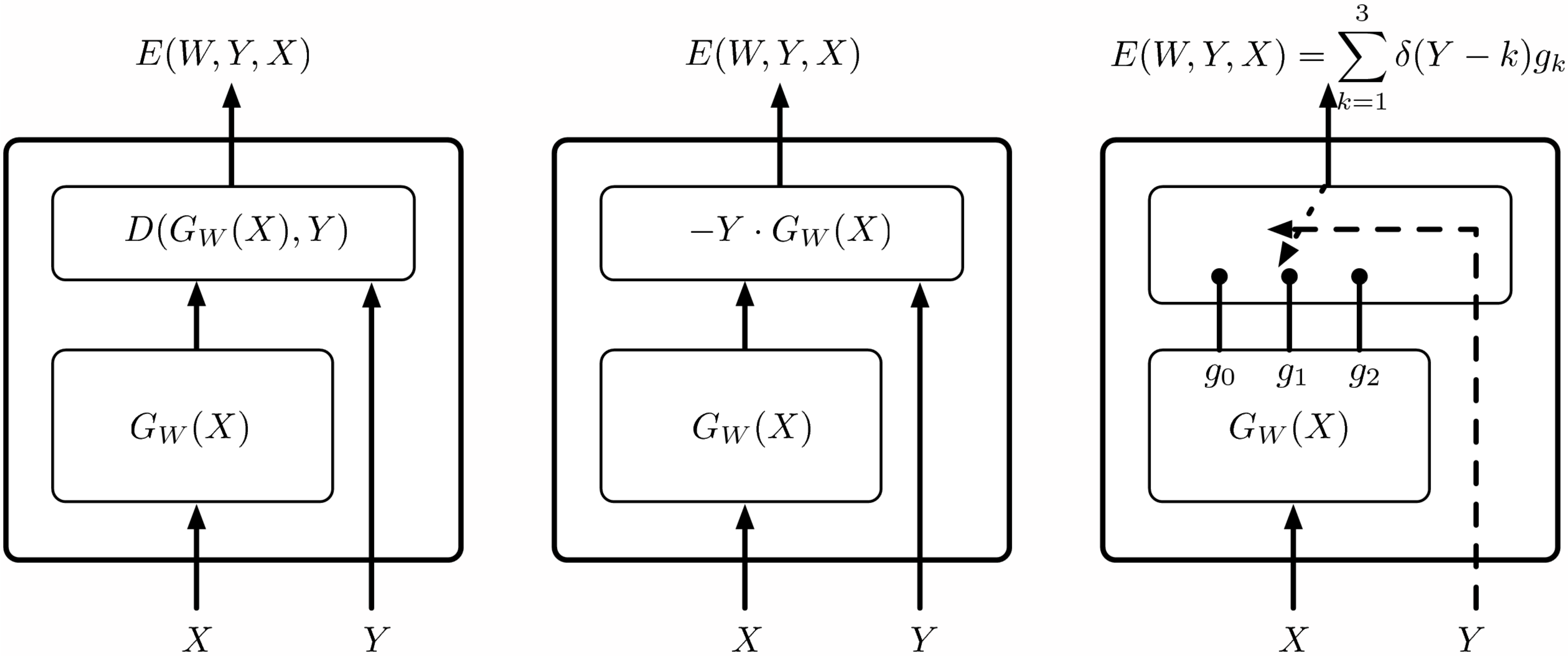
- **PROBLEM: What loss functions will make the machine approach the desired behavior?**

Several Energy Surfaces can give the same answers



- Both surfaces compute Y=X^2
- MINy E(Y,X) = X^2
- Minimum-energy inference gives us the same answer

Simple Architectures



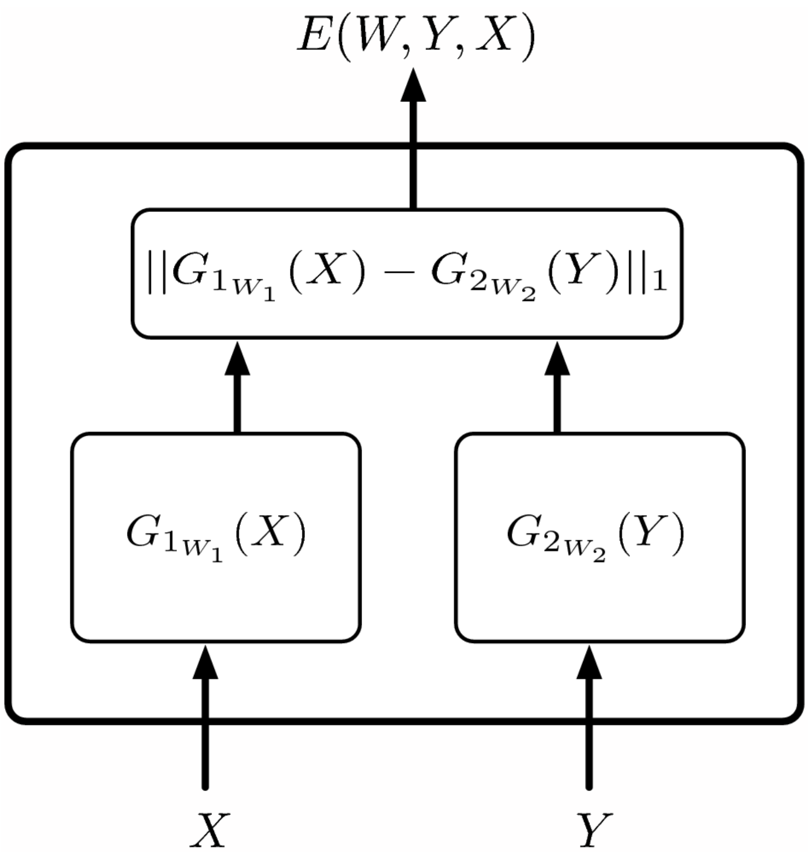
- **Regression**
- **Binary Classification**
- **Multi-class Classification**

$$E(W,Y,X) = \frac{1}{2}||G_W(X) - Y||^2.$$

$$E(W,Y,X) = -Y G_W(X),$$

Simple Architecture: Implicit Regression

$$E(W,X,Y) = ||G1_{W_1}(X) - G2_{W_2}(Y)||_1,$$



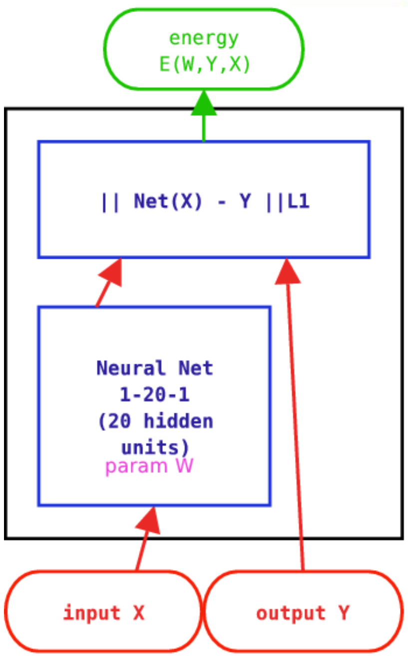
The Implicit Regression architecture

- allows multiple answers to have low energy.
- Encodes a constraint between X and Y rather than an explicit functional relationship
- This is useful for many applications
- Example: sentence completion: "The cat ate the {mouse,bird,homework,...}"
- [Bengio et al. 2003]
- But, inference may be difficult.

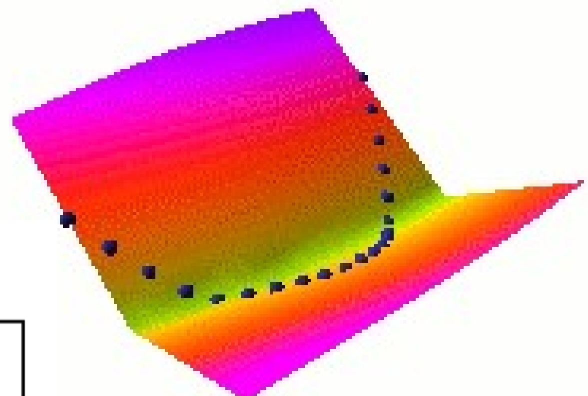
Examples of Loss Functions: Energy Loss

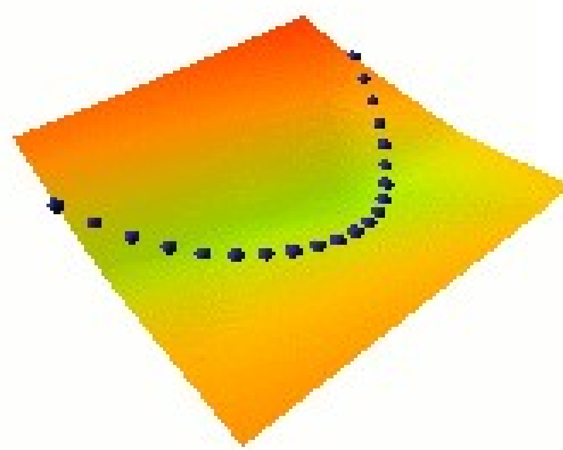
- **Energy Loss** $$L_{energy}(Y^i, E(W, \mathcal{Y}, X^i)) = E(W, Y^i, X^i).$$
  - Simply pushes down on the energy of the correct answer

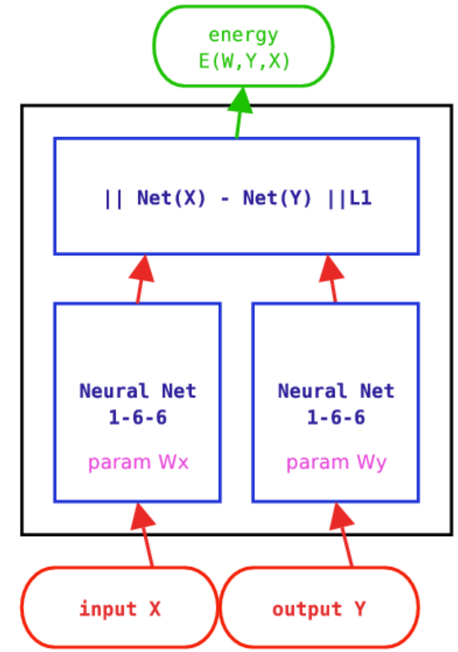
WORKS!!!



(a)







(b)

COLLAPSES!!!

Examples of Loss Functions:Perceptron Loss

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

- **Perceptron Loss [LeCun et al. 1998], [Collins 2002]**
  - Pushes down on the energy of the correct answer
  - Pulls up on the energy of the machine's answer
  - Always positive. Zero when answer is correct
  - No "margin": technically does not prevent the energy surface from being almost flat.
  - Works pretty well in practice, particularly if the energy parameterization does not allow flat surfaces.

Perceptron Loss for Binary Classification

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

- **Energy:** $E(W, Y, X) = -Y G_W(X),$
- **Inference:** $Y^* = \text{argmin}_{Y \in \{-1,1\}} - Y G_W(X) = \text{sign}(G_W(X)).$
- **Loss:** $\mathcal{L}_{\text{perceptron}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} \left(\text{sign}(G_W(X^i)) - Y^i\right) G_W(X^i).$
- **Learning Rule:** $W \leftarrow W + \eta \left(Y^i - \text{sign}(G_W(X^i))\right) \frac{\partial G_W(X^i)}{\partial W},$
- **If Gw(X) is linear in W:** $E(W, Y, X) = -Y W^T \Phi(X)$

$$W \leftarrow W + \eta \left(Y^i - \text{sign}(W^T \Phi(X^i))\right) \Phi(X^i)$$

Linear Machines: Perceptron

Perceptron:

- decision rule: $y = F(W'X)$ (F is the threshold function)
- loss function: $L(W, y^i, X^i) = (F(W'X^i) - y^i)W'X^i$
- gradient of loss: $\frac{\partial L(W, y^i, X^i)}{\partial W}' = -(y^i - F(W(t)'X^i))X^i$
- update rule: $W(t+1) = W(t) + \eta(t)(y^i - F(W(t)'X^i))X^i$
- direct solution: find W such that $-y^i F(W'X^i) < 0 \quad \forall i$

Examples of Loss Functions: Generalized Margin Losses

- **First, we need to define the Most Offending Incorrect Answer**

- **Most Offending Incorrect Answer: discrete case**

Definition 1 *Let Y be a discrete variable. Then for a training sample (X^i, Y^i), the **most offending incorrect answer** $\bar{Y}^i$ is the answer that has the lowest energy among all answers that are incorrect:*

$$\bar{Y}^i = \text{argmin}_{Y \in \mathcal{Y} and Y \neq Y^i} E(W, Y, X^i). \quad (8)$$

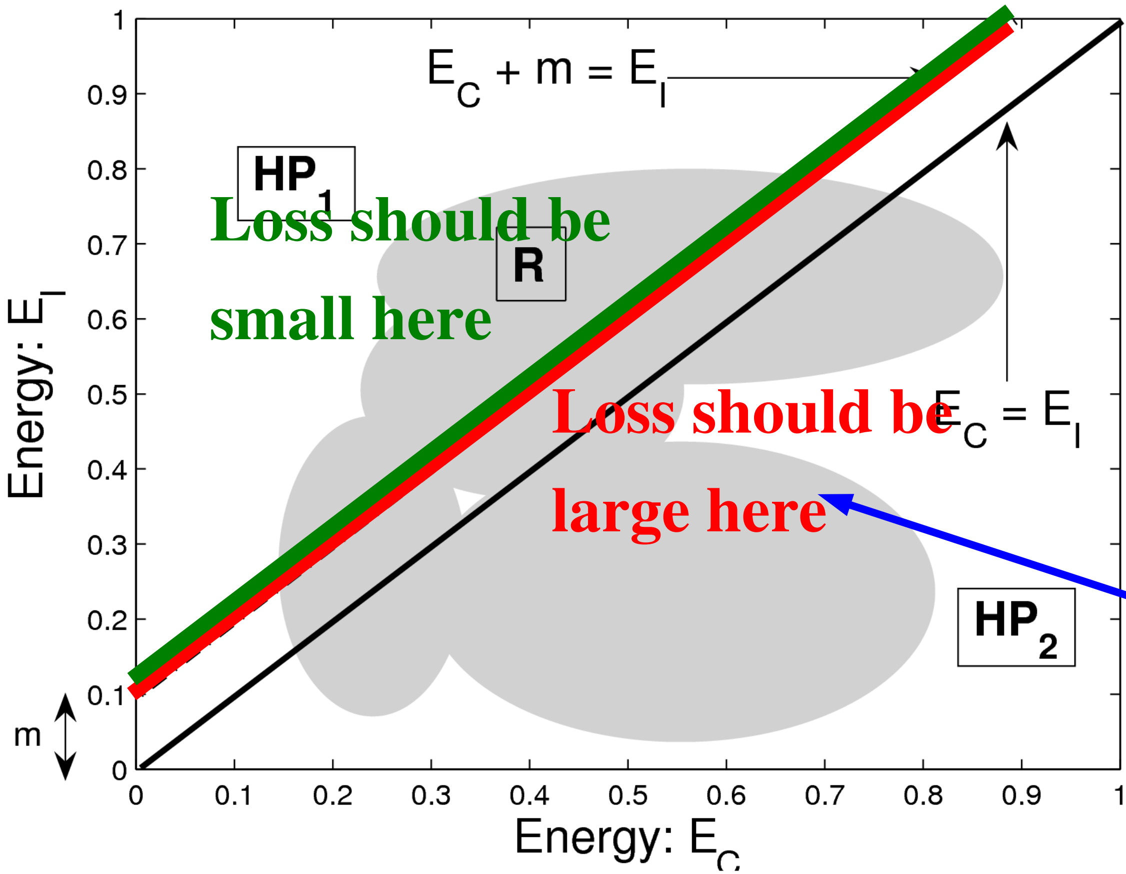
- **Most Offending Incorrect Answer: continuous case**

Definition 2 *Let Y be a continuous variable. Then for a training sample (X^i, Y^i), the **most offending incorrect answer** $\bar{Y}^i$ is the answer that has the lowest energy among all answers that are at least ϵ away from the correct answer:*

$$\bar{Y}^i = \text{argmin}_{Y \in \mathcal{Y}, \|Y - Y^i\| > \epsilon} E(W, Y, X^i). \quad (9)$$

Examples of Loss Functions: Generalized Margin Losses

$$L_{\text{margin}}(W, Y^i, X^i) = Q_m \left(E(W, Y^i, X^i), E(W, \bar{Y}^i, X^i) \right).$$



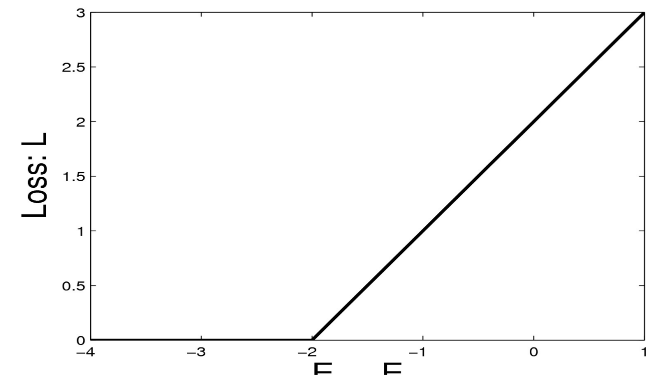
- **Generalized Margin Loss**
  - Qm increases with the energy of the correct answer
  - Qm decreases with the energy of the most offending incorrect answer
  - whenever it is less than the energy of the correct answer plus a margin m.

Examples of Generalized Margin Losses

$$L_{\text{hinge}}(W, Y^i, X^i) = \max\left(0, m + E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right),$$

Hinge Loss

- [Vapnik 1972][Altun et al. 2003], [Taska et al. 2003]
- With the linearly-parameterized binary classifier architecture, we get linear SVM

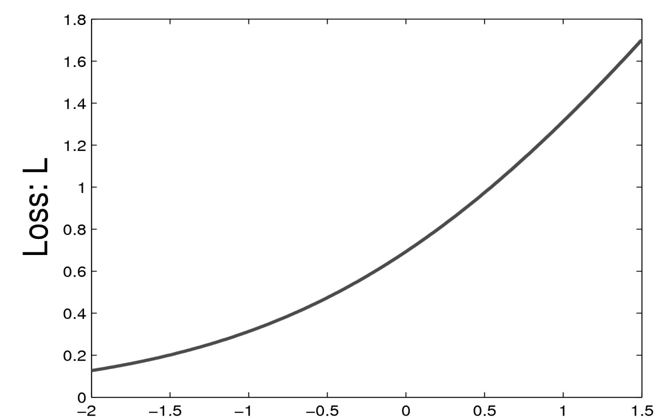


E_correct - E_incorrect

$$L_{\log}(W, Y^i, X^i) = \log\left(1 + e^{E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)}\right).$$

Log Loss

- “soft hinge” loss
- With the linearly-parameterized binary classifier architecture, we get linear Logistic Regression



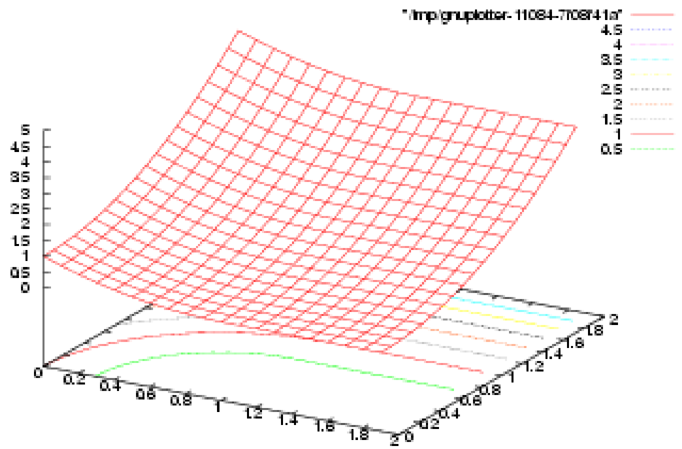
E_correct - E_incorrect

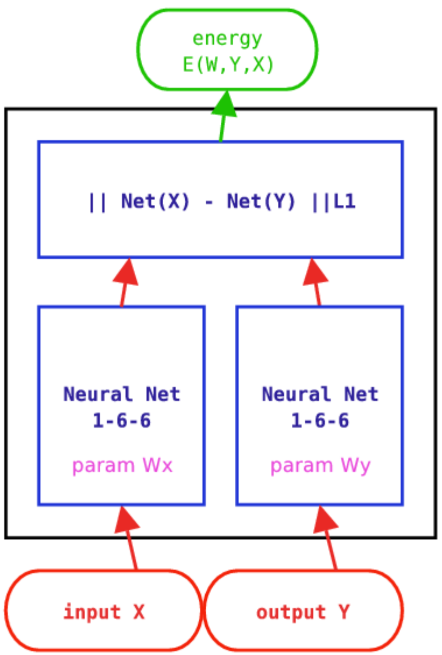
Examples of Margin Losses: Square-Square Loss

$$L_{\text{sq-sq}}(W, Y^i, X^i) = E(W, Y^i, X^i)^2 + \left(\max(0, m - E(W, \bar{Y}^i, X^i))\right)^2.$$

Square-Square Loss

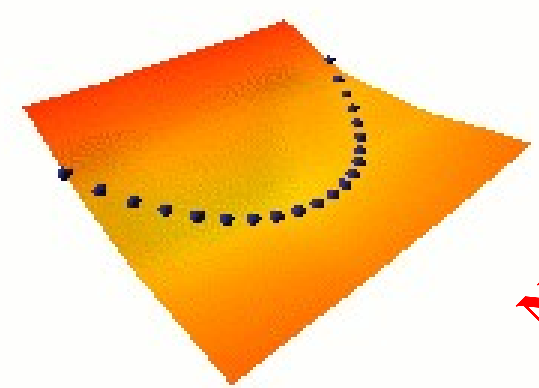
- [LeCun-Huang 2005]
- Appropriate for positive energy functions





(b)

Learning Y = X^2



NO COLLAPSE!!!

Other Margin-Like Losses

- **LVQ2 Loss** [Kohonen, Oja], [Driancourt-Bottou 1991] <- speech recognition

$$L_{\text{lvq2}}(W, Y^i, X^i) = \min\left(1, \max\left(0, \frac{E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)}{\delta E(W, \bar{Y}^i, X^i)}\right)\right),$$

- **Minimum Classification Error Loss** [Juang, Chou, Lee 1997] <- speech r.

$$L_{\text{mce}}(W, Y^i, X^i) = \sigma\left(E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right),$$

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

- **Square-Exponential Loss** [Osadchy, Miller, LeCun 2004] <- face detection

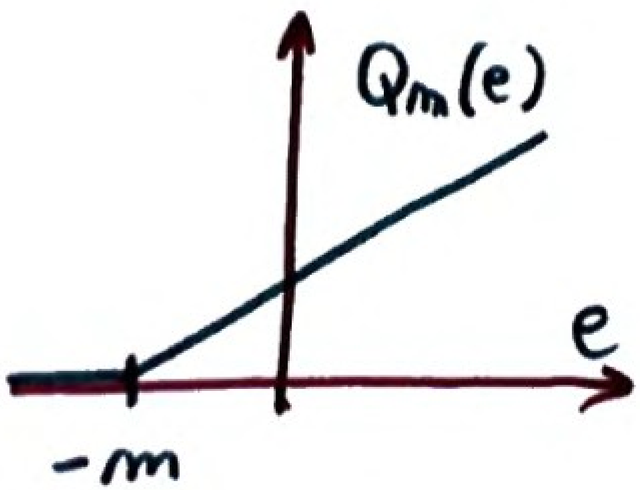
$$L_{\text{sq-exp}}(W, Y^i, X^i) = E(W, Y^i, X^i)^2 + \gamma e^{-E(W, \bar{Y}^i, X^i)},$$

Examples of Loss: Margin Loss

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

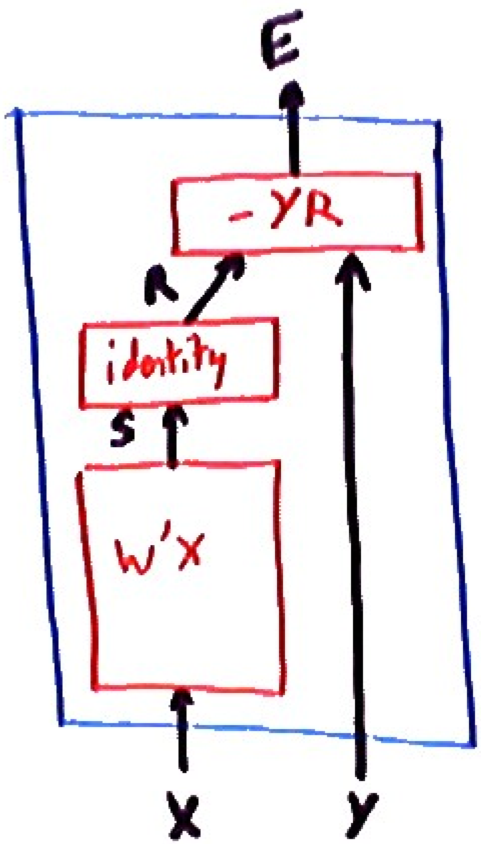
$$L_{\text{margin}}(W, Y^i, X^i) = Q_m \left(E(W, Y^i, X^i) - \min_{Y \in \{Y\}, Y \neq Y^i} E(W, Y, X^i) \right)$$

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

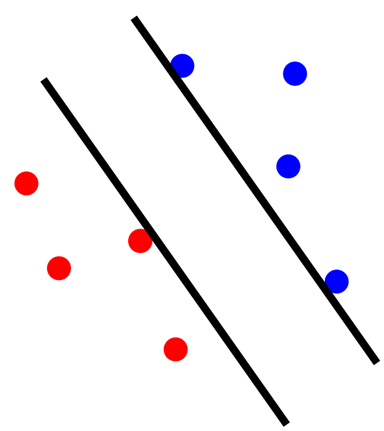


Adjust W so that $E(W, Y^i, X^i)$ gets smaller, while all $E(W, Y, X^i)$ for which $E(W, Y, X^i) - E(W, Y^i, X^i) < m$ get bigger. This guarantees that the energy of the desired Y will be smaller than all other energies by at least m.

Linear Model + Margin Loss + Regularization = SVM



- Minimize the hinge loss: make the energy of all the "good" answers smaller that the energy of any "bad" answer by at least m (the margin).
- Minimize the Regularization term: Make W as short as possible.
- This is equivalent to keeping ||W|| constant, while maximizing m.



Negative Log-Likelihood Loss

- **Conditional probability of the samples (assuming independence)**

$$P(Y^1,\dots,Y^P|X^1,\dots,X^P,W) = \prod_{i=1}^{P} P(Y^i|X^i,W).$$

$$-\log \prod_{i=1}^{P} P(Y^i|X^i,W) = \sum_{i=1}^{P} -\log P(Y^i|X^i,W).$$

- **Gibbs distribution:** $$P(Y|X^i,W) = \frac{e^{-\beta E(W,Y,X^i)}}{\int_{y\in\mathcal{Y}} e^{-\beta E(W,y,X^i)}}.$$

$$-\log \prod_{i=1}^{P} P(Y^i|X^i,W) = \sum_{i=1}^{P} \beta E(W,Y^i,X^i) + \log \int_{y\in\mathcal{Y}} e^{-\beta E(W,y,X^i)}.$$

- **We get the NLL loss by dividing by P and Beta:**

$$\mathcal{L}_{\text{nll}}(W,\mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P}\left(E(W,Y^i,X^i) + \frac{1}{\beta}\log \int_{y\in\mathcal{Y}} e^{-\beta E(W,y,X^i)}\right).$$

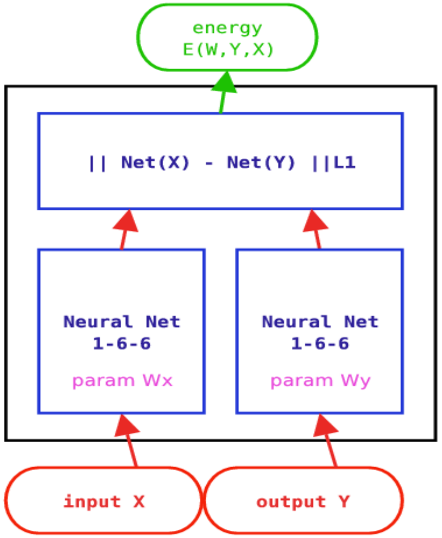
- **Reduces to the perceptron loss when Beta->infinity**

Negative Log-Likelihood Loss

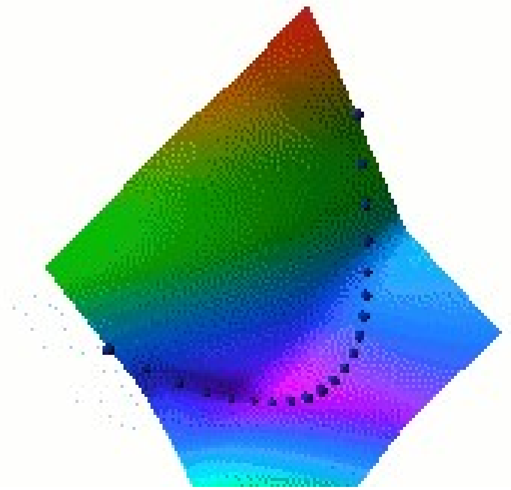
- **Pushes down on the energy of the correct answer**
- **Pulls up on the energies of all answers in proportion to their probability**

$$\mathcal{L}_{\rm nll}(W,\mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P}\left(E(W,Y^i,X^i) + \frac{1}{\beta}\log\int_{y\in\mathcal{Y}} e^{-\beta E(W,y,X^i)}\right).$$

$$\frac{\partial L_{\rm nll}(W,Y^i,X^i)}{\partial W} = \frac{\partial E(W,Y^i,X^i)}{\partial W} - \int_{Y\in\mathcal{Y}} \frac{\partial E(W,Y,X^i)}{\partial W} P(Y|X^i,W),$$



(b)



Negative Log-Likelihood Loss: Binary Classification

- **Binary Classifier Architecture:**

$$\mathcal{L}_{\text{nll}}(W,\mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P}\left[-Y^iG_W(X^i) + \log\left(e^{Y^iG_W(X^i)} + e^{-Y^iG_W(X^i)}\right)\right].$$

$$\mathcal{L}_{\text{nll}}(W,\mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P}\log\left(1 + e^{-2Y^iG_W(X^i)}\right),$$

- **Linear Binary Classifier Architecture:**

$$\mathcal{L}_{\text{nll}}(W,\mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P}\log\left(1 + e^{-2Y^iW^T\Phi(X^i)}\right).$$

- **Learning Rule in the linear case: logistic regression**
- **NLL is used by lots of speech recognition systems (they call it Maximum Mutual Information), lots of handwriting recognition systems (e.g. Bengio, LeCun 94] [LeCun et al. 98]), CRF [Lafferty et al 2001]**

Linear Machines: Logistic Regression

Logistic Regression, Negative Log-Likelihood Loss function:

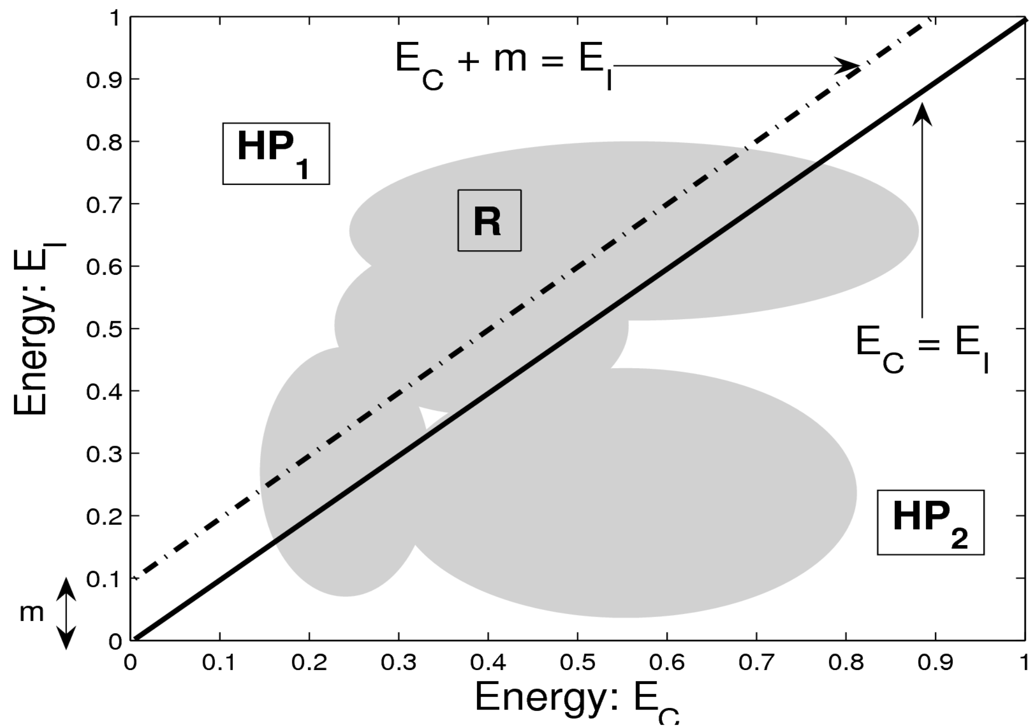
- decision rule: $y = F(W'X)$, with $F(a) = \tanh(a) = \frac{1-\exp(a)}{1+\exp(a)}$ (sigmoid function).
- loss function: $L(W, y^i, X^i) = 2\log(1 + \exp(-y^i W' X^i))$
- gradient of loss: $\frac{\partial L(W,y^i,X^i)}{\partial W}' = -\left(Y^i - F(W'X)\right) X^i$
- update rule: $W(t+1) = W(t) + \eta(t)(y^i - F(W(t)'X^i))X^i$

Negative Log-Likelihood Loss

- **Negative Log Likelihood Loss has been used for a long time in many communities for discriminative learning with structured outputs**
  - Speech recognition: many papers going back to the early 90's [Bengio 92], [Bourlard 94]. They call “Maximum Mutual Information”
  - Handwriting recognition [Bengio LeCun 94], [LeCun et al. 98]
  - Bio-informatics [Haussler]
  - Conditional Random Fields [Lafferty et al. 2001]
  - Lots more......
  - In all the above cases, it was used with non-linearly parameterized energies.

What Makes a "Good" Loss Function

- **Good loss functions make the machine produce the correct answer**
  - Avoid collapses and flat energy surfaces



Sufficient Condition on the Loss

Let (X^i, Y^i) be the i^{th} training example and m be a positive margin. Minimizing the loss function L will cause the machine to satisfy $E(W, Y^i, X^i) < E(W, Y, X^i) - m$ for all $Y \neq Y^i$, if there exists at least one point (e_1, e_2) with $e_1 + m < e_2$ such that for all points (e'_1, e'_2) with $e'_1 + m \geq e'_2$, we have

$$Q_{[E_y]}(e_1, e_2) < Q_{[E_y]}(e'_1, e'_2),$$

where $Q_{[E_y]}$ is given by

$$L(W, Y^i, X^i) = Q_{[E_y]}(E(W, Y^i, X^i), E(W, \bar{Y}^i, X^i)).$$

What Make a "Good" Loss Function

Good and bad loss functions

Loss (equation #)	Formula	Margin
energy loss	$E(W, Y^i, X^i)$	none
perceptron	$E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i)$	0
hinge	$\max\left(0, m + E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)$	m
log	$\log\left(1 + e^{E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)}\right)$	> 0
LVQ2	$\min\left(M, \max(0, E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)$	0
MCE	$\left(1 + e^{-\left(E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)}\right)^{-1}$	> 0
square-square	$E(W, Y^i, X^i)^2 - \left(\max(0, m - E(W, \bar{Y}^i, X^i))\right)^2$	m
square-exp	$E(W, Y^i, X^i)^2 + \beta e^{-E(W, \bar{Y}^i, X^i)}$	> 0
NLL/MMI	$E(W, Y^i, X^i) + \frac{1}{\beta} \log \int_{y \in \mathcal{Y}} e^{-\beta E(W, y, X^i)}$	> 0
MEE	$1 - e^{-\beta E(W, Y^i, X^i)} / \int_{y \in \mathcal{Y}} e^{-\beta E(W, y, X^i)}$	> 0

Advantages/Disadvantages of various losses

- **Loss functions differ in how they pick the point(s) whose energy is pulled up, and how much they pull them up**
- **Losses with a log partition function in the contrastive term pull up all the bad answers simultaneously.**
  - This may be good if the gradient of the contrastive term can be computed efficiently
  - This may be bad if it cannot, in which case we might as well use a loss with a single point in the contrastive term
- **Variational methods pull up many points, but not as many as with the full log partition function.**
- **Efficiency of a loss/architecture: how many energies are pulled up for a given amount of computation?**
  - The theory for this is does not exist. It needs to be developed

Latent Variable Models

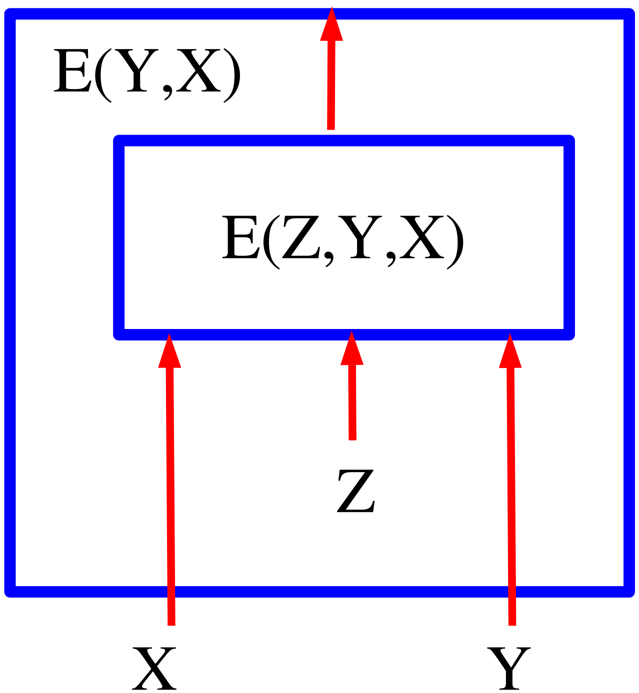
- **The energy includes "hidden" variables Z whose value is never given to us**
  - We can minimize the energy over those latent variables
  - We can also "marginalize" the energy over the latent variables

Minimization over latent variables:

$$E(Y, X) = \min_{Z \in \mathcal{Z}} E(Z, Y, X).$$

Marginalization over latent variables:

$$E(X, Y) = -\frac{1}{\beta} \log \int_{z \in \mathcal{Z}} e^{-\beta E(z, Y, X)}$$



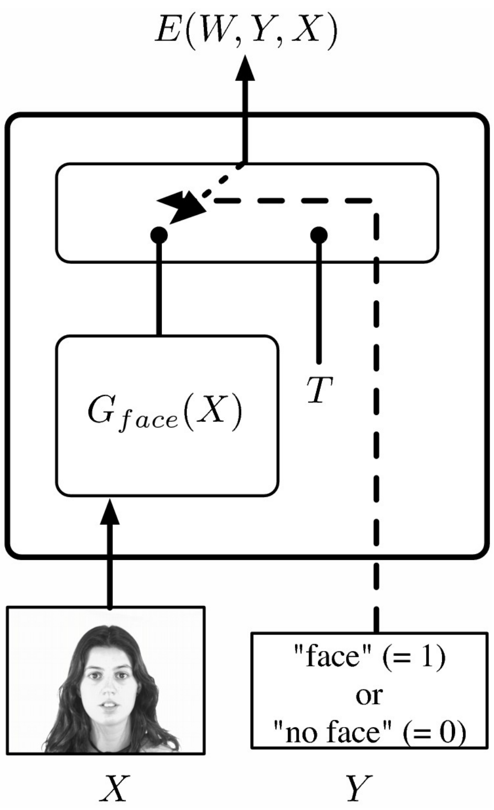
Estimation this integral may require some approximations (sampling, variational methods,....)

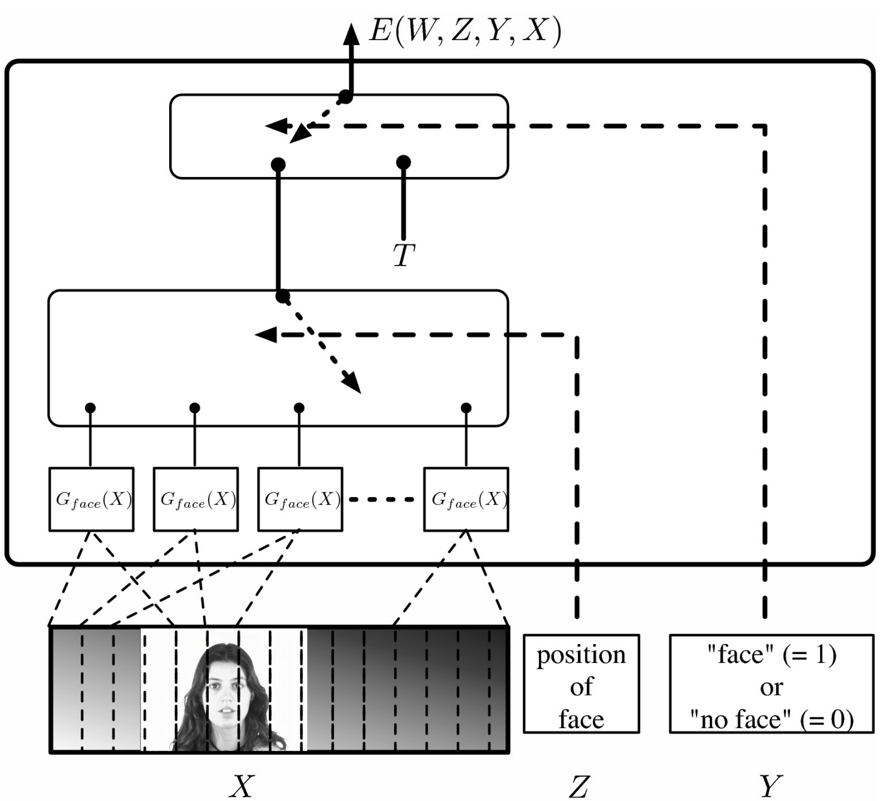
Latent Variable Models

- **The energy includes "hidden" variables Z whose value is never given to us**

$$E(Y,X) = \min_{Z \in \mathcal{Z}} E(Z,Y,X).$$

$$Y^* = \text{argmin}_{Y \in \mathcal{Y}, Z \in \mathcal{Z}} E(Z,Y,X).$$





What can the latent variables represent?

- **Variables that would make the task easier if they were known:**
  - **Face recognition**: the gender of the person, the orientation of the face.
  - **Object recognition**: the pose parameters of the object (location, orientation, scale), the lighting conditions.
  - **Parts of Speech Tagging**: the segmentation of the sentence into syntactic units, the parse tree.
  - **Speech Recognition**: the segmentation of the sentence into phonemes or phones.
  - **Handwriting Recognition**: the segmentation of the line into characters.
- **In general, we will search for the value of the latent variable that allows us to get an answer (Y) of smallest energy.**

Probabilistic Latent Variable Models

- **Marginalizing over latent variables instead of minimizing.**

$$P(Z,Y|X) = \frac{e^{-\beta E(Z,Y,X)}}{\int_{y\in\mathcal{Y},\ z\in\mathcal{Z}} e^{-\beta E(y,z,X)}}.$$

$$P(Y|X) = \frac{\int_{z\in\mathcal{Z}} e^{-\beta E(Z,Y,X)}}{\int_{y\in\mathcal{Y},\ z\in\mathcal{Z}} e^{-\beta E(y,z,X)}}.$$

- **Equivalent to traditional energy-based inference with a redefined energy function:**

$$Y^* = \text{argmin}_{Y\in\mathcal{Y}} - \frac{1}{\beta}\log\int_{z\in\mathcal{Z}} e^{-\beta E(z,Y,X)}.$$

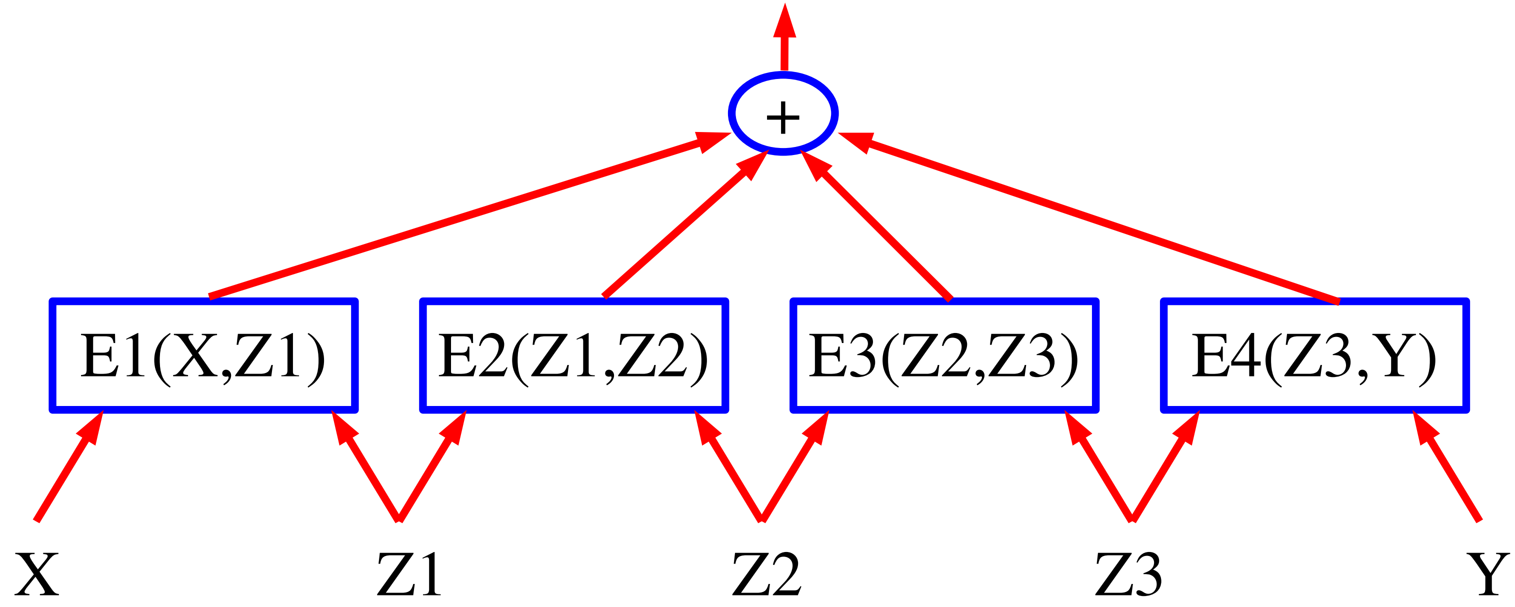
- **Reduces to minimization when Beta->infinity**

Efficient Inference: Energy-Based Factor Graphs

- **Graphical models have given us efficient inference algorithms, such as belief propagation and its numerous variations.**
- **Traditionally, graphical models are viewed as probabilistic models**
- **At first glance, is seems difficult to dissociate graphical models from the probabilistic view (think "Bayesian networks").**
- **Energy-Based Factor Graphs are an extension of graphical models to non-probabilistic settings.**
- **An EBFG is an energy function that can be written as a sum of "factor" functions that take different subsets of variables as inputs.**
- **Basically, most algorithms for probabilistic factor graphs (such as belief prop) have a counterpart for EBFG:**
  - Operations are performed in the log domain
  - The normalization steps are left out.

Energy-Based Factor Graphs

- **When the energy is a sum of partial energy functions (or when the probability is a product of factors):**
  - An EBM can be seen as an unnormalized factor graph in the log domain
  - Our favorite efficient inference algorithms can be used for inference (without the normalization step).
  - Min-sum algorithm (instead of max-product), Viterbi for chain graphs
  - (Log/sum/exp)-sum algorithm (instead of sum-product), Forward algorithm in the log domain for chain graphs

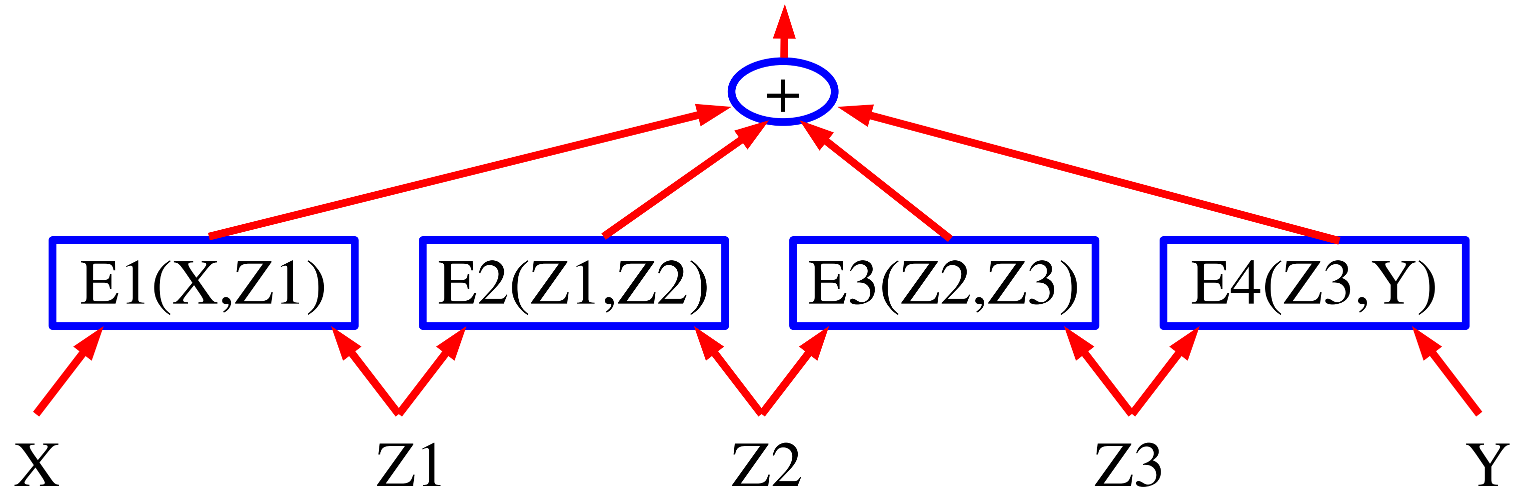


EBFG for Structured Outputs: Sequences, Graphs, Images

- **Structured outputs**
  - When Y is a complex object with components that must satisfy certain constraints.
- **Typically, structured outputs are sequences of symbols that must satisfy "grammatical" constraints**
  - spoken/handwritten word recognition
  - spoken/written sentence recognition
  - DNA sequence analysis
  - Parts of Speech tagging
  - Automatic Machine Translation
- **In General, structured outputs are collections of variables in which subsets of variables must satisfy constraints**
  - Pixels in an image for image restoration
  - Labels of regions for image segmentations
- **We represent the constraints using an Energy-Based Factor Graph.**

Energy-Based Factor Graphs: Three Inference Problems

- **X: input, Y: output, Z: latent variables, Energy: E(Z,Y,X)**
- **Minimization over Y and Z**
  - $$E(Y,X) = \min_{Z \in \mathcal{Z}} E(Z,Y,X). \qquad Y^* = \text{argmin}_{Y \in \mathcal{Y}} E(Y,X).$$
- **Min over Y, marginalization over Z (E(X,Y) is a “free energy”)**
  - $$E(X,Y) = -\frac{1}{\beta} \log \int_{z \in \mathcal{Z}} e^{-\beta E(z,Y,X)} \quad Y^* = \text{argmin}_{Y \in \mathcal{Y}} E(Y,X).$$
- **Marginal Distribution over Y**
  - $$P(Y|X) = \frac{e^{-\beta E(Y,X)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(y,X)}},$$

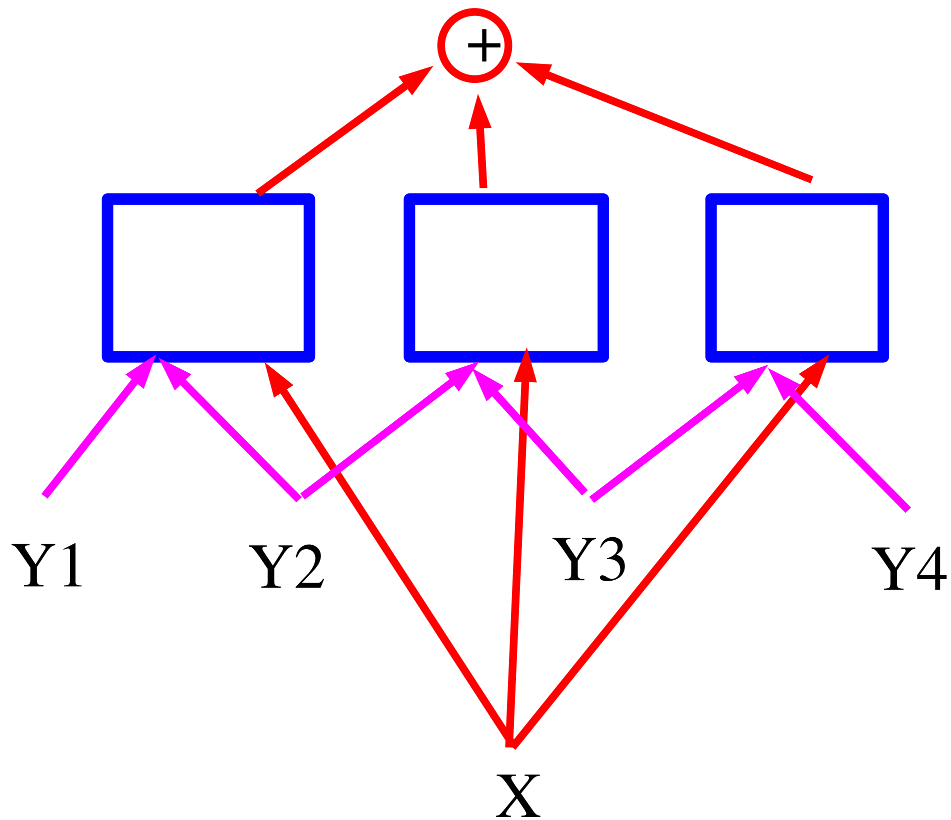


Energy-Based Factor Graphs: simple graphs

Sequence Labeling

$$Y^* = \text{argmin}_{Y \in \mathcal{Y}, Z \in \mathcal{Z}} E(Z, Y, X).$$

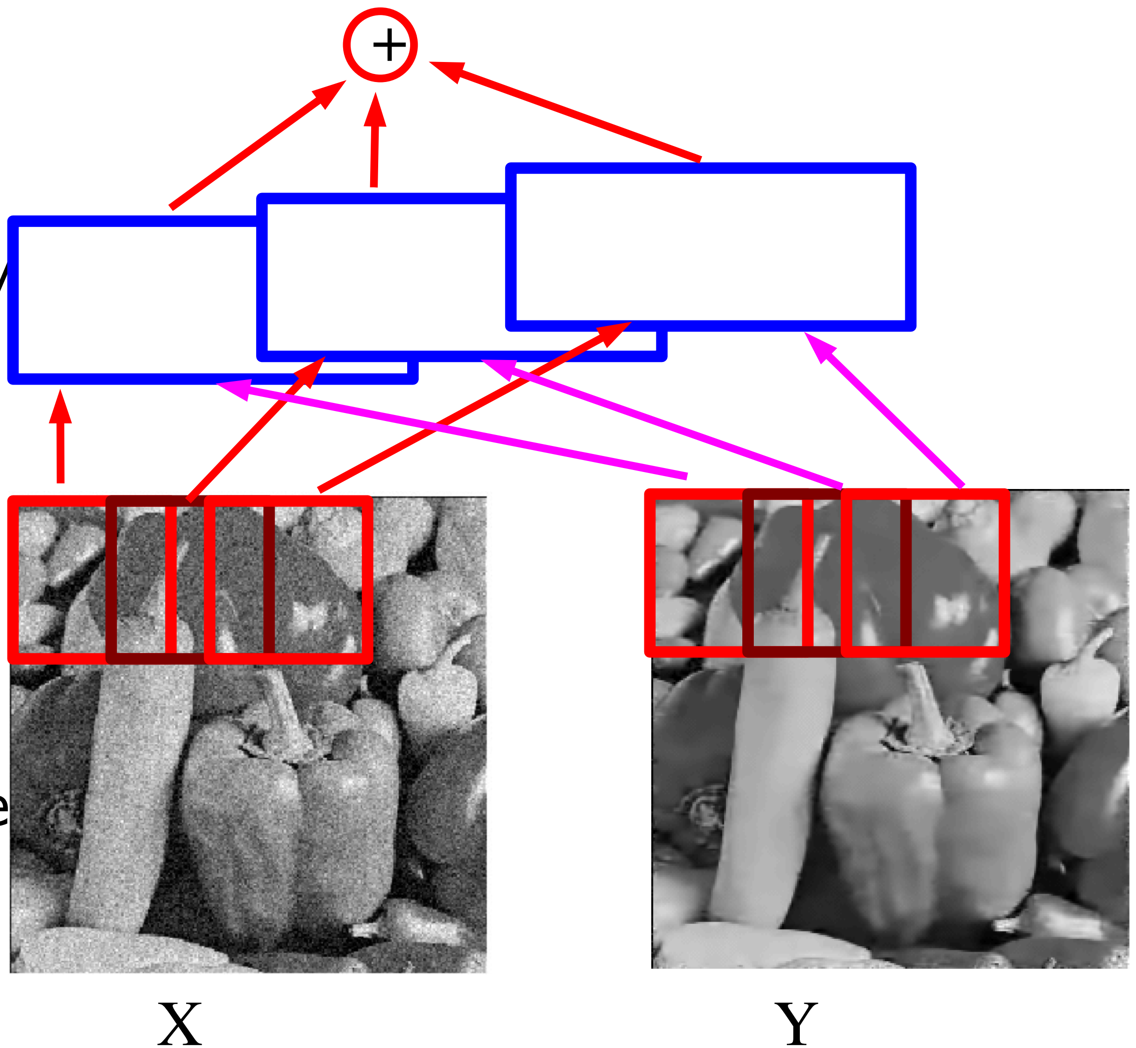
- Output is a sequence Y1,Y2,Y3,Y4......
- NLP parsing, MT, speech/handwriting recognition, biological sequence analysis
- The factors ensure grammatical consistency
- They give low energy to consistent sub-sequences of output symbols
- The graph is generally simple (chain or tree)/
- Inference is easy (dynamic programming)



Energy-Based Factor Graphs: complex/loopy graphs

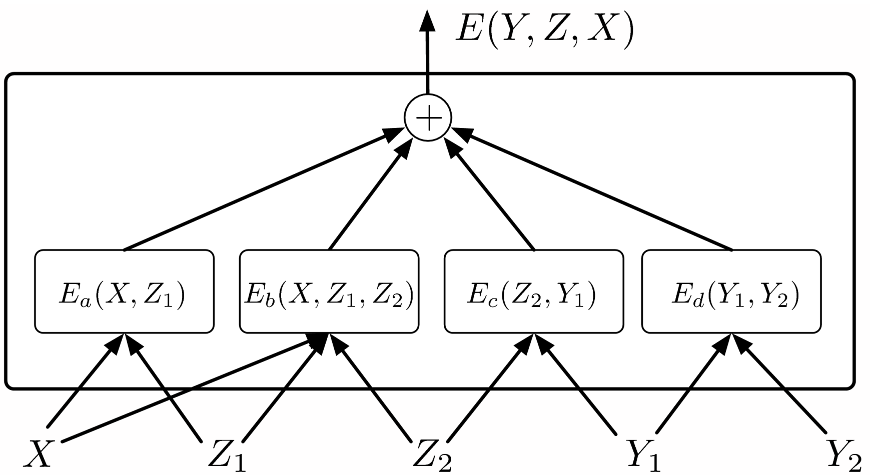
- **Image restoration**
  - The factors ensure local consistency on small overlapping patches
  - They give low energy to "clean" patches, given the noisy versions
  - The graph is loopy when the patches overlap.
  - Inference is difficult, particularly when the patches are large,and when the number of greyscale values is large

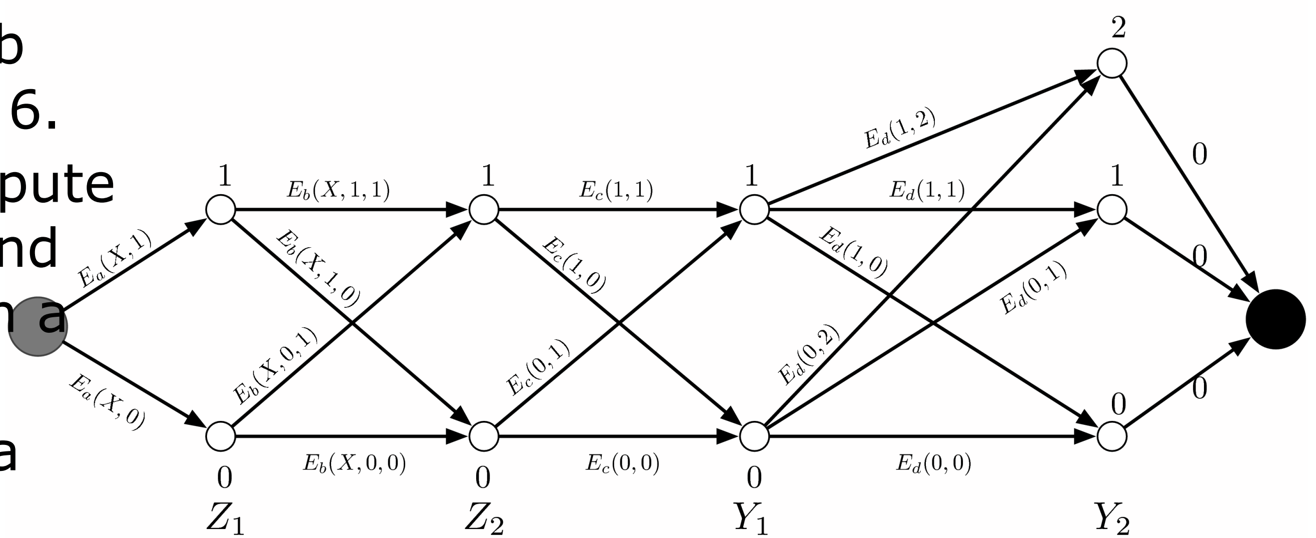
$$Y^* = \text{argmin}_{Y \in \mathcal{Y}} E(Y, X).$$



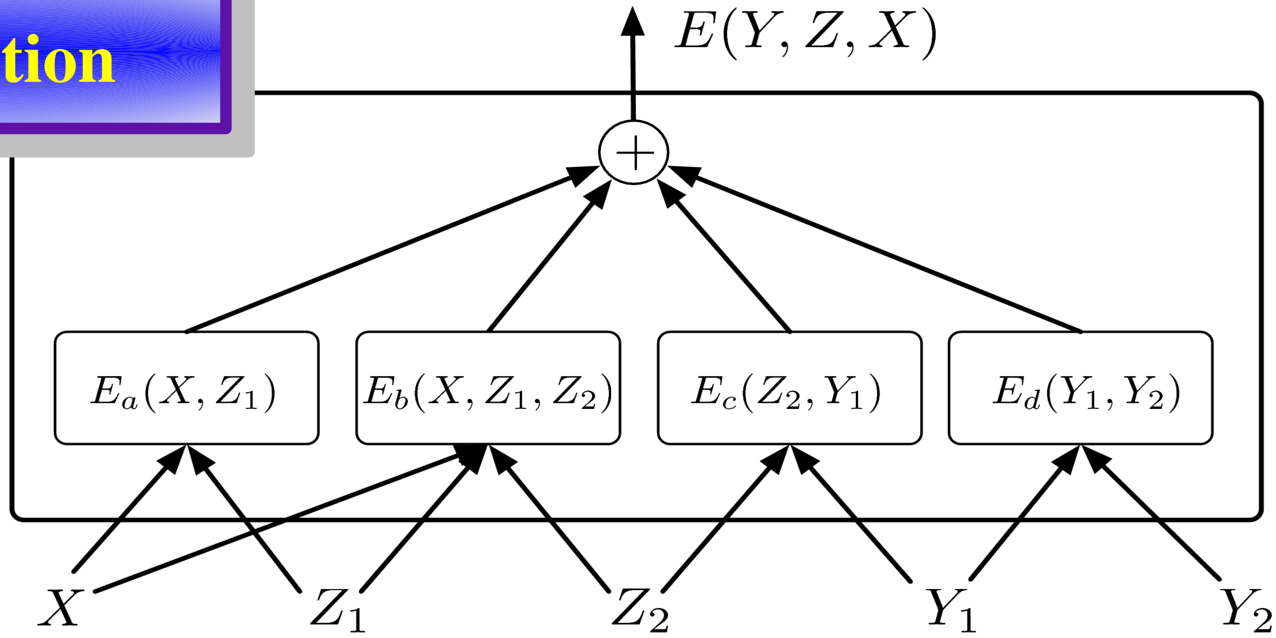
Efficient Inference in simple EBFG

- **The energy is a sum of “factor” functions, the graph is a chain**
- **Example:**
  - Z1, Z2, Y1 are binary
  - Z2 is ternary
  - A naïve exhaustive inference would require 2x2x2x3 energy evaluations (= 96 factor evaluations)
  - BUT: Ea only has 2 possible input configurations, Eb and Ec have 4, and Ed 6.
  - Hence, we can precompute the 16 factor values, and put them on the arcs in a graph.
  - A path in the graph is a config of variable
  - The cost of the path is the



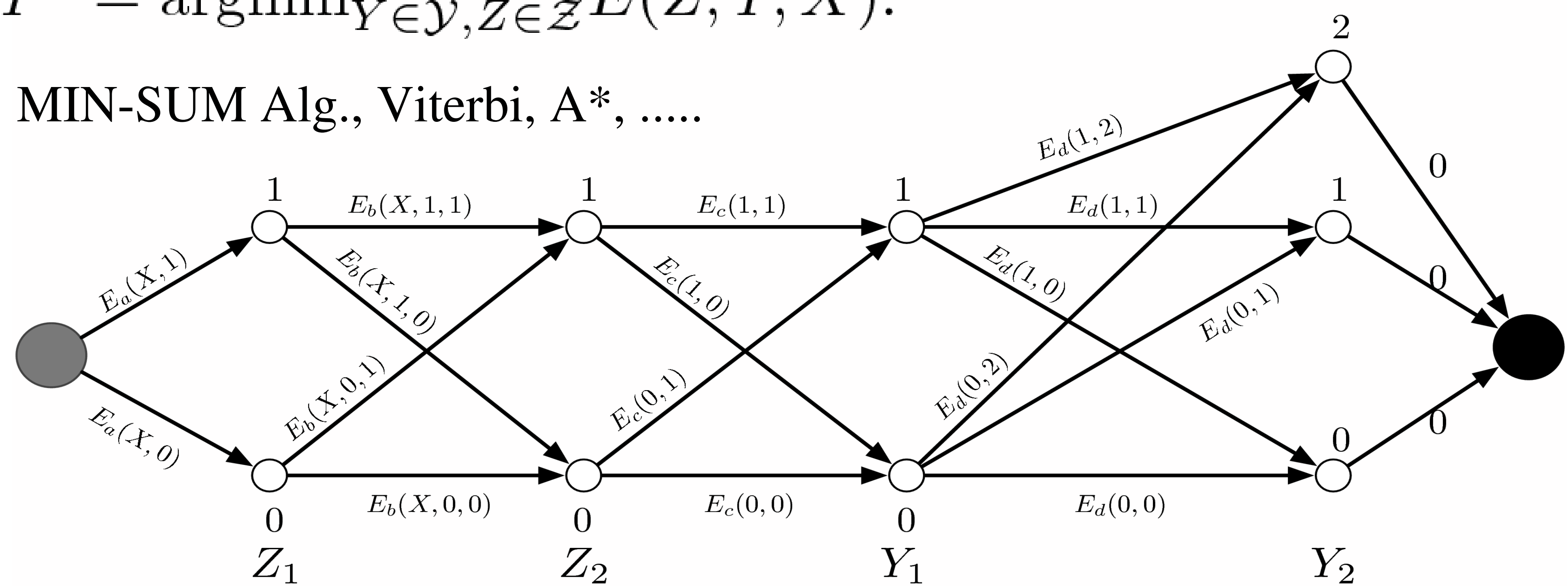


Minimization



$$Y^* = \text{argmin}_{Y \in \mathcal{Y}, Z \in \mathcal{Z}} E(Z, Y, X).$$

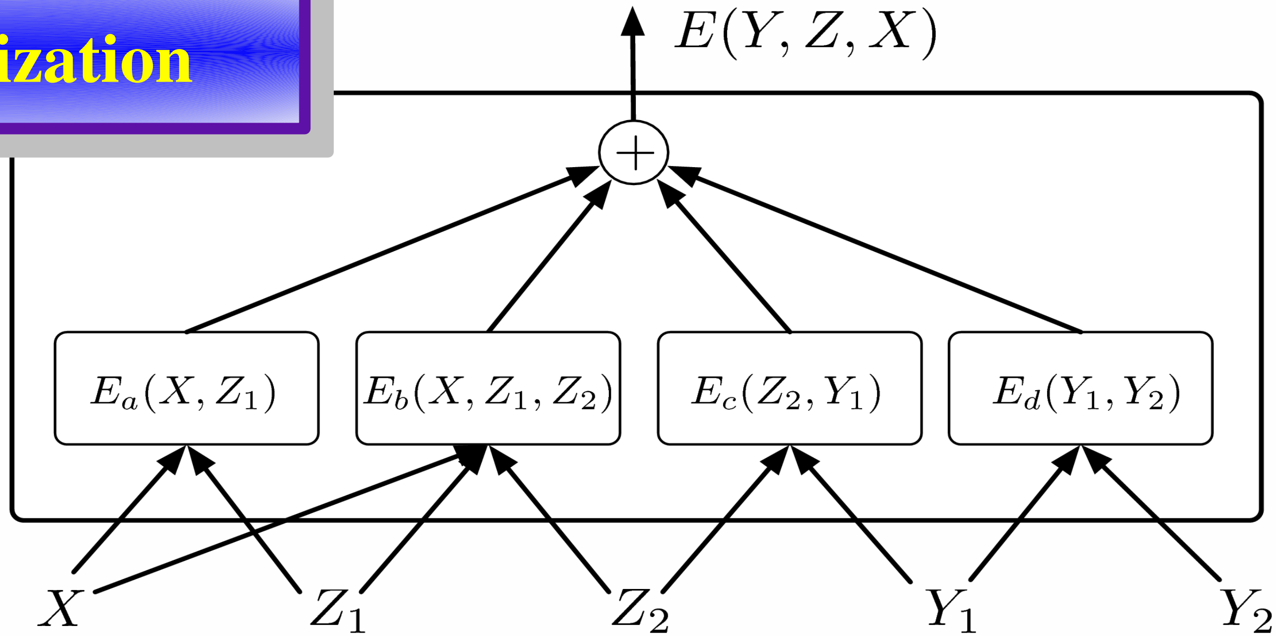
MIN-SUM Alg., Viterbi, A*, ...



Energy-Based Belief Prop: Minimization over Latent Variables

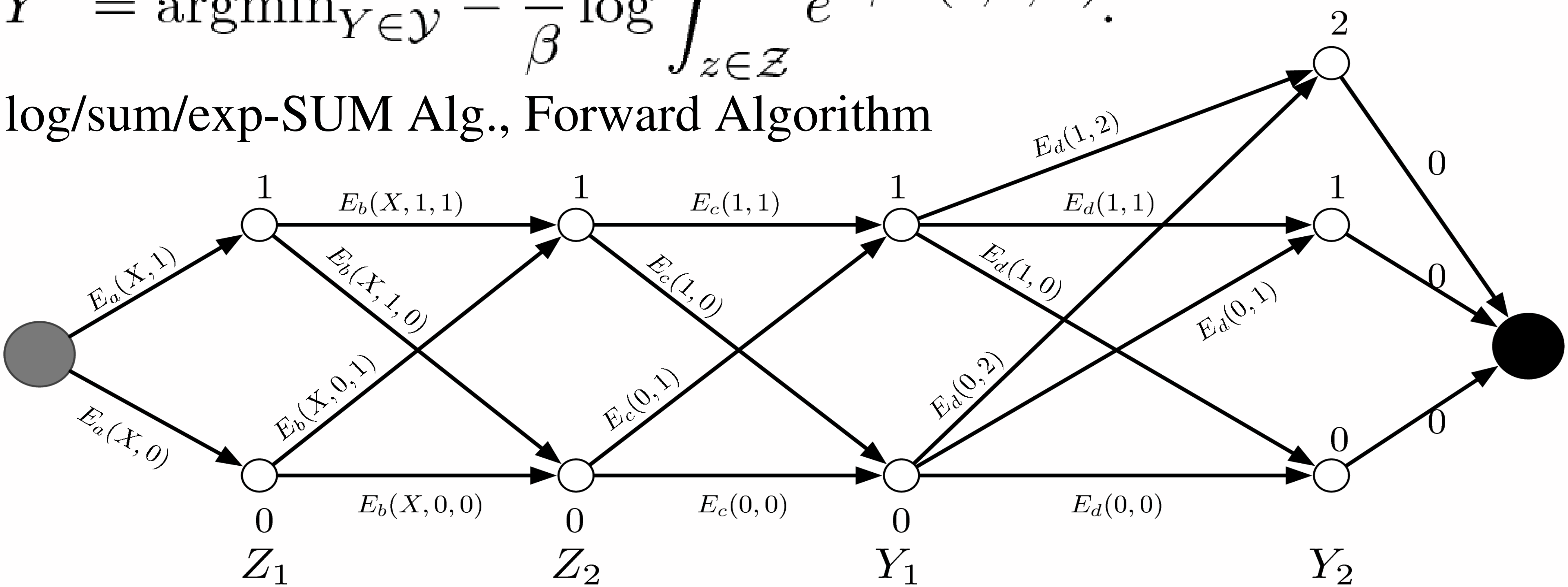
- **The previous picture shows a chain graph of factors with 2 inputs.**
- **The extension of this procedure to trees, with factors that can have more than 2 inputs is the "min-sum" algorithm (a non-probabilistic form of belief propagation)**
- **Basically, it is the sum-product algorithm with a different semi-ring algebra (min instead of sum, sum instead of product), without the normalization step.**
  - [Kschischang, Frey, Loeliger, 2001][McKay's book]

Marginalization



$$Y^* = \text{argmin}_{Y \in \mathcal{Y}} - \frac{1}{\beta} \log \int_{z \in \mathcal{Z}} e^{-\beta E(z, Y, X)}.$$

log/sum/exp-SUM Alg., Forward Algorithm



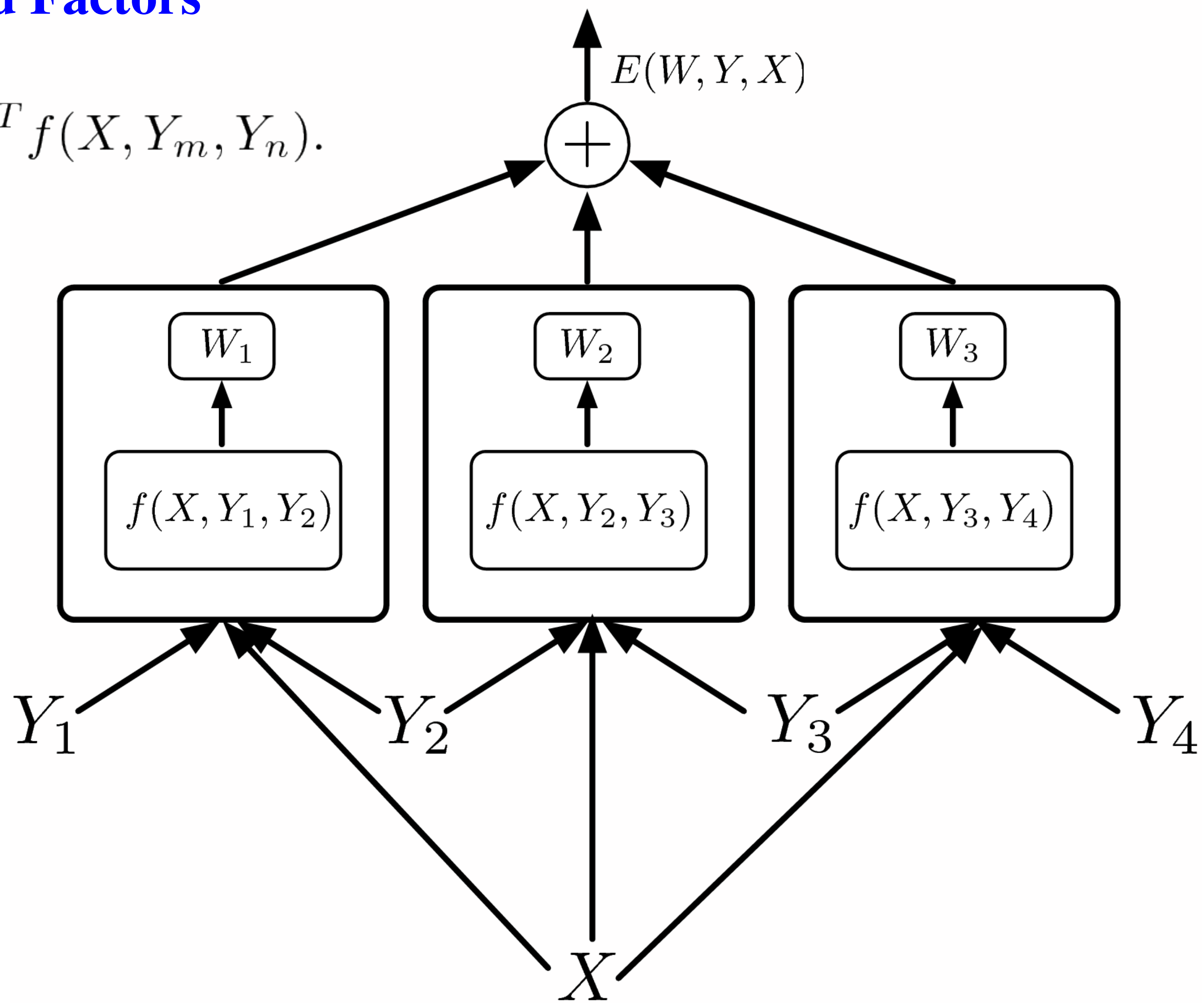
Energy-Based Belief Prop: Marginalization over Latent Variables

- **The previous picture shows a chain graph of factors with 2 inputs.**
  - Going along a path: add up the energies
  - When several paths meet: compute $-\frac{1}{\beta}\log\sum_i e^{-\beta E_{ji}}$
- **The extension of this procedure to trees, with factors that can have more than 2 inputs is the "[log/sum/exp]-sum" algorithm (a non-probabilistic form of belief propagation)**
- **Basically, it is the sum-product algorithm with a different semi-ring algebra (log/sum/exp instead of sum, sum instead of product), and without the normalization step.**
  - [Kschischang, Frey, Loeliger, 2001][McKay's book]

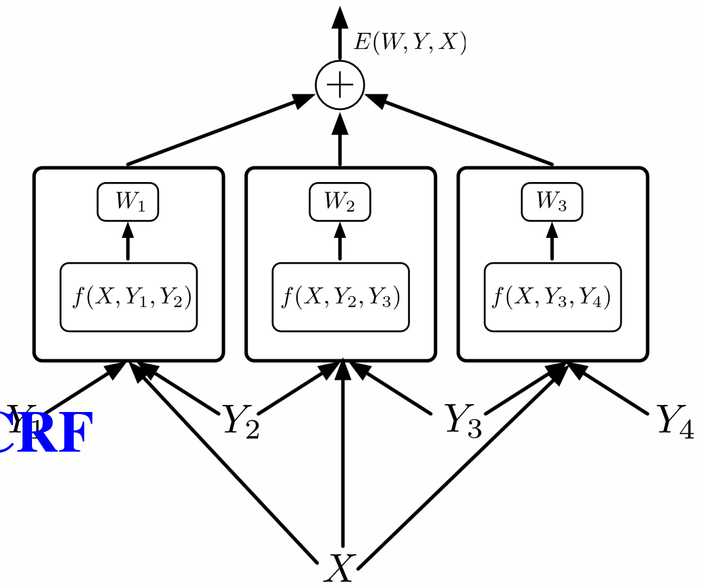
A Simple Case: Linearly Parameterized Factors: CRF, MMMN

- **Linearly Parameterized Factors**

$$E(W,Y,X) = \sum_{(m,n)\in\mathcal{F}} W^T f(X, Y_m, Y_n).$$



Linearly Parameterized Factors + Negative Log Likelihood Loss = Conditional Random Fields



- **Linearly Parameterized Factors + NLL loss = CRF**
  - [Lafferty, McCallum, Pereira, 2001]
- **Non-linear factors = Graph Transformer Networks**
  - [LeCun, Bottou, Bengio, Haffner, 1998]

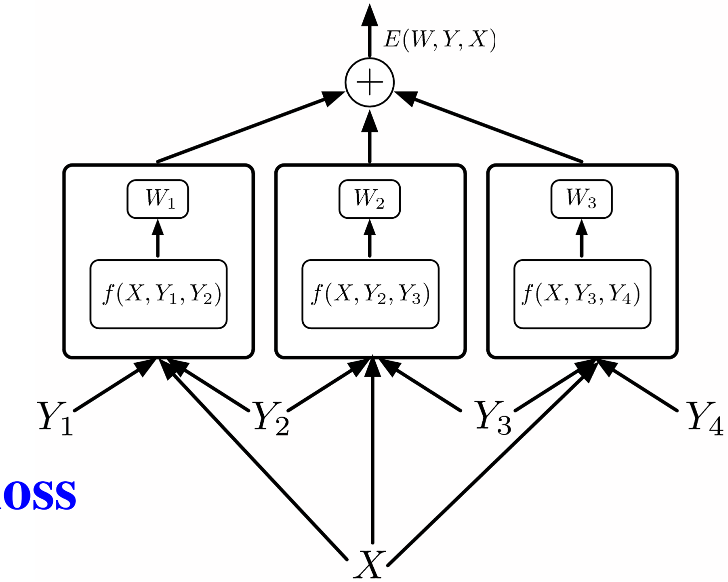
$$\mathcal{L}_{\text{nll}}(W) = \frac{1}{P}\sum_{i=1}^{P} W^T F(X^i, Y^i) + \frac{1}{\beta}\log\sum_{y\in\mathcal{Y}} e^{-\beta W^T F(X^i,y)}.$$

$$\frac{\partial \mathcal{L}_{\text{nll}}(W)}{\partial W} = \frac{1}{P}\sum_{i=1}^{P} F(X^i, Y^i) - \sum_{y\in\mathcal{Y}} F(X^i, y) P(y|X^i, W),$$

$$P(y|X^i, W) = \frac{e^{-\beta W^T F(X^i,y)}}{\sum_{y'\in\mathcal{Y}} e^{-\beta W^T F(X^i,y')}}.$$

simplest/best learning procedure: stochastic gradient

Linearly Parameterized Factors + Perceptron Loss = Sequence Perceptron



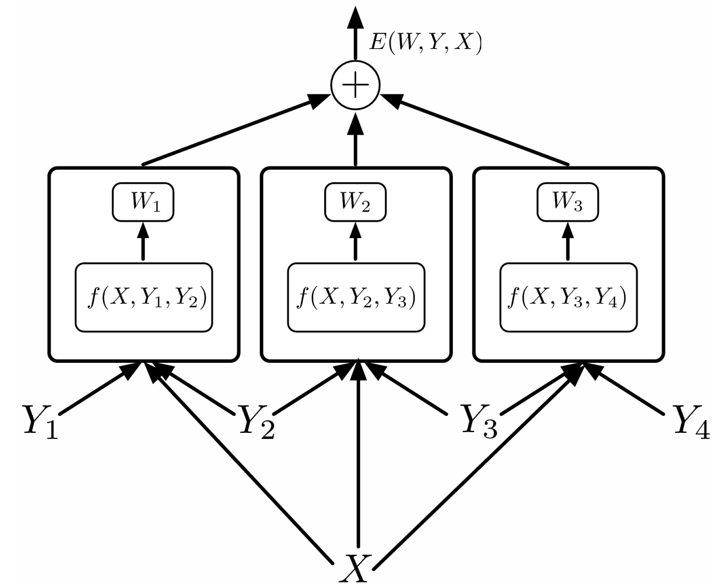
- **Linearly Parameterized Factors + Perceptron loss**
  - [Collins 2000, Collins 2001]
- **Non-linear factors + perceptron loss**
  - [LeCun, Bottou, Bengio, Haffner 1998]

$$\mathcal{L}_{\text{perceptron}}(W) = \frac{1}{P}\sum_{i=1}^{P} E(W,Y^i,X^i) - E(W,Y^{*i},X^i),$$

$$\mathcal{L}_{\text{perceptron}}(W) = \frac{1}{P}\sum_{i=1}^{P} W^T \left(F(X^i,Y^i) - F(X^i,Y^{*i})\right).$$

$$W \leftarrow W - \eta \left(F(X^i,Y^i) - F(X^i,Y^{*i})\right).$$

Linearly Parameterized Factors + Hinge Loss = Max Margin Markov Networks



- **Linearly Parameterized Factor + Hinge loss**
  - [Altun et a. 2003, Taskar et al. 2003]

$$\mathcal{L}_{\text{hinge}}(W) = \frac{1}{P}\sum_{i=1}^{P} \max(0, m + E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)) + \gamma ||W||^2.$$

$$\mathcal{L}_{\text{hinge}}(W) = \frac{1}{P}\sum_{i=1}^{P} \max\left(0, m + W^T \Delta F(X^i, Y^i)\right) + \gamma ||W||^2,$$

$$\Delta F(X^i, Y^i) = F(X^i, Y^i) - F(X^i, \bar{Y}^i)$$

Simple gradient descent rule:

$$\text{If } \Delta F(X^i, Y^i) > -m \text{ then } W \leftarrow W - \eta \Delta F(X^i, Y^i) - 2\gamma W$$

Can be performed in the dual (like an SVM)