A Tutorial on Energy-Based Learning

Yann LeCun, Sumit Chopra, Raia Hadsell,
Marc'Aurelio Ranzato, and Fu Jie Huang
The Courant Institute of Mathematical Sciences,
New York University
{yann,sumit,raia,ranzato,jhuangfu}@cs.nyu.edu
http://yann.lecun.com
v1.0, August 19, 2006
To appear in "Predicting Structured Data",
G. Bakir, T. Hofman, B. Schölkopf, A. Smola, B. Taskar (eds)
MIT Press, 2006

Abstract

Energy-Based Models (EBMs) capture dependencies between variables by associating a scalar energy to each configuration of the variables. Inference consists in clamping the value of observed variables and finding configurations of the remaining variables that minimize the energy. Learning consists in finding an energy function in which observed configurations of the variables are given lower energies than unobserved ones. The EBM approach provides a common theoretical framework for many learning models, including traditional discriminative and generative approaches, as well as graph-transformer networks, conditional random fields, maximum margin Markov networks, and several manifold learning methods.

Probabilistic models must be properly normalized, which sometimes requires evaluating intractable integrals over the space of all possible variable configurations. Since EBMs have no requirement for proper normalization, this problem is naturally circumvented. EBMs can be viewed as a form of non-probabilistic factor graphs, and they provide considerably more flexibility in the design of architectures and training criteria than probabilistic approaches.

1 Introduction: Energy-Based Models

The main purpose of statistical modeling and machine learning is to encode dependencies between variables. By capturing those dependencies, a model can be used to answer questions about the values of unknown variables given the values of known variables.

Energy-Based Models (EBMs) capture dependencies by associating a scalar *energy* (a measure of compatibility) to each configuration of the variables. *Inference*, i.e., making a prediction or decision, consists in setting the value of observed variables and finding values of the remaining variables that minimize the energy. *Learning* consists in finding an energy function that associates low energies to correct values of the remaining variables, and higher energies to incorrect values. A *loss functional*, minimized during learning, is used to measure the quality of the available energy functions. Within this common inference/learning framework, the wide choice of energy functions and loss functionals allows for the design of many types of statistical models, both probabilistic and non-probabilistic.

Energy-based learning provides a unified framework for many probabilistic and non-probabilistic approaches to learning, particularly for non-probabilistic training of graphical models and other structured models. Energy-based learning can be seen as an alternative to probabilistic estimation for prediction, classification, or decision-making tasks. Because there is no requirement for proper normalization, energy-based approaches avoid the problems associated with estimating the normalization constant in probabilistic models. Furthermore, the absence of the normalization condition allows for much more flexibility in the design of learning machines. Most probabilistic models can be viewed as special types of energy-based models in which the energy function satisfies certain normalizability conditions, and in which the loss function, optimized by learning, has a particular form.

This chapter presents a tutorial on energy-based models, with an emphasis on their use for structured output problems and sequence labeling problems. Section 1 introduces energy-based models and describes deterministic inference through energy minimization. Section 2 introduces energy-based learning and the concept of the loss function. A number of standard and non-standard loss functions are described, including the perceptron loss, several margin-based losses, and the negative log-likelihood loss. The negative log-likelihood loss can be used to train a model to produce conditional probability estimates. Section 3 shows how simple regression and classification models can be formulated in the EBM framework. Section 4 concerns models that contain latent variables. Section 5 analyzes the various loss functions in detail and gives sufficient conditions that a loss function must satisfy so that its minimization will cause the model to approach the desired behavior. A list of "good" and "bad" loss functions is given. Section 6 introduces the concept of non-probabilistic factor graphs and informally discusses efficient inference algorithms. Section 7 focuses on sequence labeling and structured output models. Linear models such as max-margin Markov networks and conditional random fields are re-formulated in the EBM framework. The literature on discriminative learning for speech and handwriting recognition, going back to the late 80's and early 90's, is reviewed. This includes globally trained systems that integrate non-linear discriminant functions, such as neural networks, and sequence alignment methods, such as dynamic time warping and hidden Markov models. Hierarchical models such as the graph transformer network architecture are also reviewed. Finally, the differences, commonalities, and relative advantages of energy-based approaches, probabilistic approaches, and sampling-based approximate methods such as contrastive divergence are discussed in Section 8.

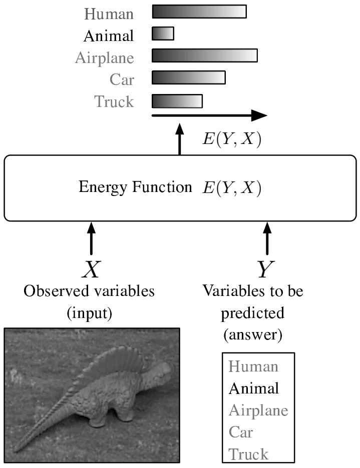


Figure 1: *A model measures the compatibility between observed variables X and variables to be predicted Y using an* energy function $E(Y, X)$. *For example, X could be the pixels of an image, and Y a discrete label describing the object in the image. Given X, the model produces the answer Y that minimizes the energy E.*

1.1 Energy-Based Inference

Let us consider a model with two sets of variables, X and Y, as represented in Figure 1. Variable X could be a vector containing the pixels from an image of an object. Variable Y could be a discrete variable that represents the possible category of the object. For example, Y could take six possible values: animal, human figure, airplane, truck, car, and "none of the above". The model is viewed as an *energy function* which measures the "goodness" (or badness) of each possible configuration of X and Y. The output number can be interpreted as the degree of *compatibility* between the values of X and Y. In the following, we use the convention that small energy values correspond to highly compatible configurations of the variables, while large energy values correspond to highly incompatible configurations of the variables. Functions of this type are given different names in different technical communities; they may be called contrast functions, value functions, or negative log-likelihood functions. In the following, we will use the term *energy function* and denote it $E(Y, X)$. A distinction should be made between the energy function, which is minimized by the inference process, and the loss functional (introduced in Section 2), which is minimized by the learning process.

In the most common use of a model, the input X is given (observed from the world), and the model produces the answer Y that is most compatible with the observed X. More precisely, the model must produce the value Y^*, chosen from a set $\mathcal{Y}$, for which $E(Y, X)$ is the smallest:

$$Y^* = \text{argmin}_{Y \in \mathcal{Y}} E(Y, X). \tag{1}$$

When the size of the set $\mathcal{Y}$ is small, we can simply compute $E(Y, X)$ for all possible values of $Y \in \mathcal{Y}$ and pick the smallest.

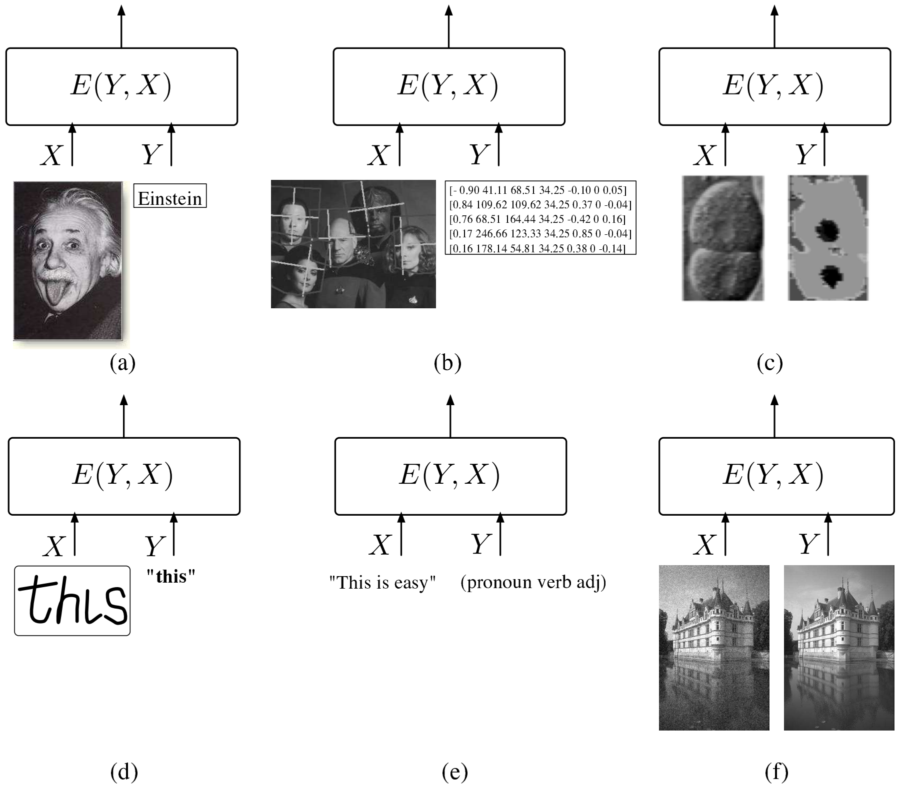


Figure 2: *Several applications of EBMs:* **(a) face recognition:** *Y is a high-cardinality discrete variable;* **(b) face detection and pose estimation:** *Y is a collection of vectors with location and pose of each possible face;* **(c) image segmentation:** *Y is an image in which each pixel is a discrete label;* **(d-e) handwriting recognition and sequence labeling:** *Y is a sequence of symbols from a highly structured but potentially infinite set (the set of English sentences). The situation is similar for many applications in natural language processing and computational biology;* **(f) image restoration:** *Y is a high-dimensional continuous variable (an image).*

In general, however, picking the best Y may not be simple. Figure 2 depicts several situations in which $\mathcal{Y}$ may be too large to make exhaustive search practical. In Figure 2(a), the model is used to recognize a face. In this case, the set $\mathcal{Y}$ is discrete and finite, but its cardinality may be tens of thousands [Chopra et al., 2005]. In Figure 2(b), the model is used to find the faces in an image and estimate their poses. The set $\mathcal{Y}$ contains a binary variable for each location indicating whether a face is present at that location, and a set of continuous variables representing the size and orientation of the face [Osadchy et al., 2005]. In Figure 2(c), the model is used to segment a biological image: each pixel must be classified into one of five categories (cell nucleus, nuclear membrane, cytoplasm, cell membrane, external medium). In this case, $\mathcal{Y}$ contains all the *consistent* label images, i.e. the ones for which the nuclear membranes are encircling the nuclei, the nuclei and cytoplasm are inside the cells walls, etc. The set is discrete, but intractably large. More importantly, members of the set must satisfy complicated consistency constraints [Ning et al., 2005]. In Figure 2(d), the model is used to recognize a handwritten sentence. Here $\mathcal{Y}$ contains all possible sentences of the English language, which is a discrete but infinite set of sequences of symbols [LeCun et al., 1998a]. In Figure 2(f), the model is used to restore an image (by cleaning the noise, enhancing the resolution, or removing scratches). The set $\mathcal{Y}$ contains all possible images (all possible pixel combinations). It is a continuous and high-dimensional set.

For each of the above situations, a specific strategy, called the *inference procedure*, must be employed to find the Y that minimizes $E(Y, X)$. In many real situations, the inference procedure will produce an approximate result, which may or may not be the global minimum of $E(Y, X)$ for a given X. In fact, there may be situations where $E(Y, X)$ has several equivalent minima. The best inference procedure to use often depends on the internal structure of the model. For example, if $\mathcal{Y}$ is continuous and $E(Y, X)$ is smooth and well-behaved with respect to Y, one may use a gradient-based optimization algorithm. If Y is a collection of discrete variables and the energy function can be expressed as a *factor graph*, i.e. a sum of energy functions (factors) that depend on different subsets of variables, efficient inference procedures for factor graphs can be used (see Section 6) [Kschischang et al., 2001, MacKay, 2003]. A popular example of such a procedure is the *min-sum* algorithm. When each element of $\mathcal{Y}$ can be represented as a path in a weighted directed acyclic graph, then the energy for a particular Y is the sum of values on the edges and nodes along a particular path. In this case, the best Y can be found efficiently using dynamic programming (e.g with the Viterbi algorithm or A^*). This situation often occurs in sequence labeling problems such as speech recognition, handwriting recognition, natural language processing, and biological sequence analysis (e.g. gene finding, protein folding prediction, etc). Different situations may call for the use of other optimization procedures, including continuous optimization methods such as linear programming, quadratic programming, non-linear optimization methods, or discrete optimization methods such as simulated annealing, graph cuts, or graph matching. In many cases, exact optimization is impractical, and one must resort to approximate methods, including methods that use surrogate energy functions (such as variational methods).

1.2 What Questions Can a Model Answer?

In the preceding discussion, we have implied that the question to be answered by the model is "What is the Y that is most compatible with this X?", a situation that occurs in *prediction, classification* or *decision-making* tasks. However, a model may be used to answer questions of several types:

1. *Prediction, classification, and decision-making*: "Which value of Y is most compatible with this X?' This situation occurs when the model is used to make hard decisions or to produce an action. For example, if the model is used to drive a robot and avoid obstacles, it must produce a single best decision such as "steer left", "steer right", or "go straight".

2. *Ranking*: "Is Y_1 or Y_2 more compatible with this X?" This is a more complex task than classification because the system must be trained to produce a complete ranking of all the answers, instead of merely producing the best one. This situation occurs in many data mining applications where the model is used to select multiple samples that best satisfy a given criterion.

3. *Detection*: "Is this value of Y compatible with X?" Typically, detection tasks, such as detecting faces in images, are performed by comparing the energy of a *face* label with a threshold. Since the threshold is generally unknown when the system is built, the system must be trained to produce energy values that increase as the image looks less like a face.

4. *Conditional density estimation*: "What is the conditional probability distribution over $\mathcal{Y}$ given X?" This case occurs when the output of the system is not used directly to produce actions, but is given to a human decision maker or is fed to the input of another, separately built system.

We often think of X as a high-dimensional variable (e.g. an image) and Y as a discrete variable (e.g. a label), but the converse case is also common. This occurs when the model is used for such applications as image restoration, computer graphics, speech and language production, etc. The most complex case is when both X and Y are high-dimensional.

1.3 Decision Making versus Probabilistic Modeling

For decision-making tasks, such as steering a robot, it is merely necessary that the system give the lowest energy to the correct answer. The energies of other answers are irrelevant, as long as they are larger. However, the output of a system must sometimes be combined with that of another system, or fed to the input of another system (or to a human decision maker). Because energies are uncalibrated (i.e. measured in arbitrary units), combining two, separately trained energy-based models is not straightforward: there is no *a priori* guarantee that their energy scales are commensurate. Calibrating energies so as to permit such combinations can be done in a number of ways. However, the only *consistent* way involves turning the collection of energies for all possible outputs into a normalized probability distribution. The simplest and most common method for turning a collection of arbitrary energies into a collection of numbers between 0 and 1 whose sum (or integral) is 1 is through the *Gibbs distribution*:

$$P(Y|X) = \frac{e^{-\beta E(Y,X)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(y,X)}}, \tag{2}$$

where β is an arbitrary positive constant akin to an inverse temperature, and the denominator is called the *partition function* (by analogy with similar concepts in statistical physics). The choice of the Gibbs distribution may seem arbitrary, but other probability distributions can be obtained (or approximated) through a suitable re-definition of the energy function. Whether the numbers obtained this way are good probability estimates does not depend on how energies are turned into probabilities, but on how $E(Y, X)$ is estimated from data.

It should be noted that the above transformation of energies into probabilities is only possible if the integral $\int_{y \in \mathcal{Y}} e^{-\beta E(y,X)}$ converges. This somewhat restricts the energy functions and domains $\mathcal{Y}$ that can be used. More importantly, there are many practical situations where computing the partition function is intractable (e.g. when $\mathcal{Y}$ has high cardinality), or outright impossible (e.g. when $\mathcal{Y}$ is a high dimensional variable and the integral has no analytical solution). Hence probabilistic modeling comes with a high price, and should be avoided when the application does not require it.

2 Energy-Based Training: Architecture and Loss Function

Training an EBM consists in finding an energy function that produces the best Y for any X. The search for the best energy function is performed within a family of energy functions $\mathcal{E}$ indexed by a parameter W

$$\mathcal{E} = \{E(W, Y, X) : \ \ W \in \mathcal{W}\}. \tag{3}$$

The *architecture* of the EBM is the internal structure of the parameterized energy function $E(W, Y, X)$. At this point, we put no particular restriction on the nature of X, Y, W, and $\mathcal{E}$. When X and Y are real vectors, $\mathcal{E}$ could be as simple as a linear combination of basis functions (as in the case of kernel methods), or a set of neural net architectures and weight values. Section gives examples of simple architectures for common applications to classification and regression. When X and Y are variable-size images, sequences of symbols or vectors, or more complex structured objects, $\mathcal{E}$ may represent a considerably richer class of functions. Sections 4, 6 and 7 discuss several examples of such architectures. One advantage of the energy-based approach is that it puts very little restrictions on the nature of $\mathcal{E}$.

To train the model for prediction, classification, or decision-making, we are given a set of training samples $\mathcal{S} = \{(X^i, Y^i) : i = 1 \dots P\}$, where X^i is the input for the i-th training sample, and Y^i is the corresponding desired answer. In order to find the best energy function in the family $\mathcal{E}$, we need a way to assess the quality of any particular energy function, based solely on two elements: the training set, and our prior knowledge about the task. This quality measure is called the *loss functional* (i.e. a function of function) and denoted $\mathcal{L}(E, \mathcal{S})$. For simplicity, we often denote it $\mathcal{L}(W, \mathcal{S})$ and simply call it the *loss function*. The learning problem is simply to find the W that minimizes the loss:

$$W^* = \min_{W \in \mathcal{W}} \mathcal{L}(W, \mathcal{S}). \tag{4}$$

For most cases, the loss functional is defined as follows:

$$\mathcal{L}(E, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} L(Y^i, E(W, \mathcal{Y}, X^i)) + R(W). \tag{5}$$

It is an average taken over the training set of a *per-sample loss functional*, denoted $L(Y^i, E(W, \mathcal{Y}, X^i))$, which depends on the desired answer Y^i and on the energies obtained by keeping the input sample fixed and varying the answer Y. Thus, for each sample, we evaluate a "slice" of the energy surface. The term $R(W)$ is the *regularizer*, and can be used to embed our prior knowledge about which energy functions in our family are preferable to others (in the absence of training data). With this definition, the loss is invariant under permutations of the training samples and under multiple repetitions of the training set.

Naturally, the ultimate purpose of learning is to produce a model that will give good answers for new input samples that are not seen during training. We can rely on general results from statistical learning theory which guarantee that, under simple interchangeability conditions on the samples and general conditions on the family of energy functions (finite VC dimension), the deviation between the value of the loss after minimization on the training set, and the loss on a large, separate set of test samples is bounded by a quantity that converges to zero as the size of training set increases [Vapnik, 1995].

2.1 Designing a Loss Functional

Intuitively, the per-sample loss functional should be designed in such a way that it assigns a low loss to *well-behaved* energy functions: energy functions that give the lowest energy to the correct answer and higher energy to all other (incorrect) answers. Conversely, energy functions that do not assign the lowest energy to the correct answers would have a high loss. Characterizing the appropriateness of loss functions (the ones that select the best energy functions) is further discussed in following sections.

Considering only the task of training a model to answer questions of type 1 (prediction, classification and decision-making), the main intuition of the energy-based approach is as follows. Training an EBM consists in shaping the energy function, so that for any given X, the inference algorithm will produce the desired value for Y. Since the inference algorithm selects the Y with the lowest energy, the learning procedure must shape the energy surface so that the desired value of Y has lower energy than all other (undesired) values. Figures 3 and 4 show examples of energy as a function of Y for a given input sample X^i in cases where Y is a discrete variable and a continuous scalar variable. We note three types of answers:

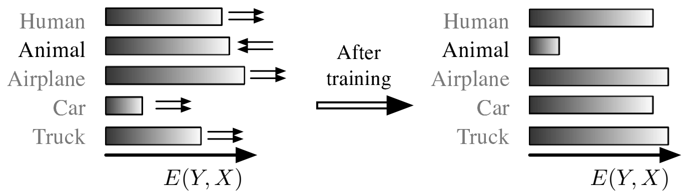


Figure 3: *How training affects the energies of the possible answers in the discrete case: the energy of the correct answer is decreased, and the energies of incorrect answers are increased, particularly if they are lower than that of the correct answer.*

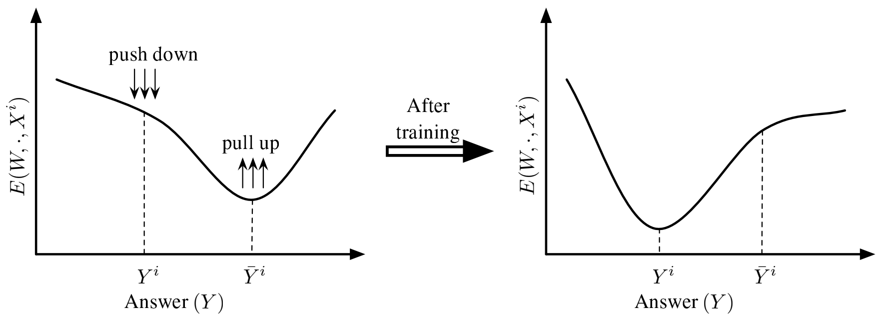


Figure 4: *The effect of training on the energy surface as a function of the answer Y in the continuous case. After training, the energy of the correct answer Y^i is lower than that of incorrect answers.*

- Y^i: the correct answer
- Y^{*i}: the answer produced by the model, i.e. the answer with the lowest energy.
- $\bar{Y}^i$: the *most offending incorrect answer*, i.e. the answer that has the lowest energy among all the incorrect answers. To define this answer in the continuous case, we can simply view all answers within a distance ϵ of Y^i as correct, and all answers beyond that distance as incorrect.

With a properly designed loss function, the learning process should have the effect of "pushing down" on $E(W, Y^i, X^i)$, and "pulling up" on the incorrect energies, particularly on $E(W, \bar{Y}^i, X^i)$. Different loss functions do this in different ways. Section 5 gives sufficient conditions that the loss function must satisfy in order to be guaranteed to shape the energy surface correctly. We show that some widely used loss functions do not satisfy the conditions, while others do.

To summarize: given a training set $\mathcal{S}$, building and training an energy-based model involves designing four components:

1. *The architecture*: the internal structure of $E(W, Y, X)$.
2. *The inference algorithm*: the method for finding a value of Y that minimizes $E(W, Y, X)$ for any given X.
3. *The loss function*: $\mathcal{L}(W, \mathcal{S})$ measures the quality of an energy function using the training set.
4. *The learning algorithm*: the method for finding a W that minimizes the loss functional over the family of energy functions $\mathcal{E}$, given the training set.

Properly designing the architecture and the loss function is critical. Any prior knowledge we may have about the task at hand is embedded into the architecture and into the loss function (particularly the regularizer). Unfortunately, not all combinations of architectures and loss functions are allowed. With some combinations, minimizing the loss will not make the model produce the best answers. Choosing the combinations of architecture and loss functions that can learn effectively and efficiently is critical to the energy-based approach, and thus is a central theme of this tutorial.

2.2 Examples of Loss Functions

We now describe a number of standard loss functions that have been proposed and used in the machine learning literature. We shall discuss them and classify them as "good" or "bad" in an energy-based setting. For the time being, we set aside the regularization term, and concentrate on the data-dependent part of the loss function.

2.2.1 Energy Loss

The simplest and the most straightforward of all the loss functions is the energy loss. For a training sample (X^i, Y^i), the per-sample loss is defined simply as:

$$L_{energy}(Y^i, E(W, \mathcal{Y}, X^i)) = E(W, Y^i, X^i). \tag{6}$$

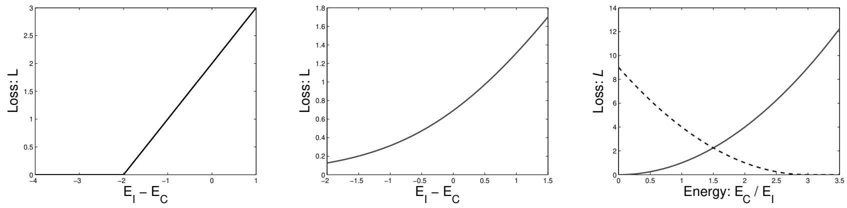


Figure 5: *The hinge loss (left) and log loss (center) penalize $E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)$ linearly and logarithmically, respectively. The square-square loss (right) separately penalizes large values of $E(W, Y^i, X^i)$ (solid line) and small values of $E(W, \bar{Y}^i, X^i)$ (dashed line) quadratically.*

This loss function, although very popular for things like regression and neural network training, cannot be used to train most architectures: while this loss will push down on the energy of the desired answer, it will not pull up on any other energy. With some architectures, this can lead to a *collapsed solution* in which the energy is constant and equal to zero. The energy loss will only work with architectures that are designed in such a way that pushing down on $E(W, Y^i, X^i)$ will automatically make the energies of the other answers larger. A simple example of such an architecture is $E(W, Y^i, X^i) = ||Y^i - G(W, X^i)||^2$, which corresponds to regression with mean-squared error with G being the regression function.

2.2.2 Generalized Perceptron Loss

The generalized perceptron loss for a training sample (X^i, Y^i) is defined as

$$L_{perceptron}(Y^i, E(W, \mathcal{Y}, X^i)) = E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i). \tag{7}$$

This loss is always positive, since the second term is a lower bound on the first term. Minimizing this loss has the effect of pushing down on $E(W, Y^i, X^i)$, while pulling up on the energy of the answer produced by the model.

While the perceptron loss has been widely used in many settings, including for models with structured outputs such as handwriting recognition [LeCun et al., 1998a] and parts of speech tagging [Collins, 2002], it has a major deficiency: there is no mechanism for creating an energy gap between the correct answer and the incorrect ones. Hence, as with the energy loss, the perceptron loss may produce flat (or almost flat) energy surfaces if the architecture allows it. Consequently, a meaningful, uncollapsed result is only guaranteed with this loss if a model is used that cannot produce a flat energy surface. For other models, one cannot guarantee anything.

2.2.3 Generalized Margin Losses

Several loss functions can be described as *margin* losses; the hinge loss, log loss, LVQ2 loss, minimum classification error loss, square-square loss, and square-exponential loss all use some form of margin to create an energy gap between the correct answer and the incorrect answers. Before discussing the generalized margin loss we give the following definitions.

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:*

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

If Y is a continuous variable then the definition of the most offending incorrect answer can be defined in a number of ways. The simplest definition is as follows.

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:*

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

The generalized margin loss is a more robust version of the generalized perceptron loss. It directly uses the energy of the most offending incorrect answer in the contrastive term:

$$L_{\text{margin}}(W, Y^i, X^i) = Q_m \left(E(W, Y^i, X^i), E(W, \bar{Y}^i, X^i) \right). \tag{10}$$

Here m is a positive parameter called the *margin* and $Q_m(e_1, e_2)$ is a convex function whose gradient has a positive dot product with the vector $[1, -1]$ in the region where $E(W, Y^i, X^i) + m > E(W, \bar{Y}^i, X^i)$. In other words, the loss surface is slanted toward low values of $E(W, Y^i, X^i)$ and high values of $E(W, \bar{Y}^i, X^i)$ wherever $E(W, Y^i, X^i)$ is not smaller than $E(W, \bar{Y}^i, X^i)$ by at least m. Two special cases of the generalized margin loss are given below:

Hinge Loss: A particularly popular example of generalized margin loss is the *hinge loss*, which is used in combination with linearly parameterized energies and a quadratic regularizer in support vector machines, support vector Markov models [Altun and Hofmann, 2003], and maximum-margin Markov networks [Taskar et al., 2003]:

$$L_{\text{hinge}}(W, Y^i, X^i) = \max \left(0, m + E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i) \right), \tag{11}$$

where m is the positive margin. The shape of this loss function is given in Figure 5. The difference between the energies of the correct answer and the most offending incorrect answer is penalized linearly when larger than $-m$. The hinge loss only depends on energy differences, hence individual energies are not constrained to take any particular value.

Log Loss: a common variation of the hinge loss is the *log loss*, which can be seen as a "soft" version of the hinge loss with an infinite margin (see Figure 5, center):

$$L_{\log}(W, Y^i, X^i) = \log \left(1 + e^{E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)} \right). \tag{12}$$

LVQ2 Loss: One of the very first proposals for discriminatively training sequence labeling systems (particularly speech recognition systems) is a version of Kohonen's LVQ2 loss. This loss has been advocated by Driancourt and Bottou since the early 90's [Driancourt et al., 1991a, Driancourt and Gallinari, 1992b, Driancourt and Gallinari, 1992a, Driancourt, 1994, McDermott, 1997, McDermott and Katagiri, 1992]:

$$L_{\text{lvq2}}(W, Y^i, X^i) = \min\left(1, \max\left(0, \frac{E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)}{\delta E(W, \bar{Y}^i, X^i)}\right)\right), \quad (13)$$

where δ is a positive parameter. LVQ2 is a zero-margin loss, but it has the peculiarity of saturating the ratio between $E(W, Y^i, X^i)$ and $E(W, \bar{Y}^i, X^i)$ to $1 + \delta$. This mitigates the effect of outliers by making them contribute a nominal cost M to the total loss. This loss function is a continuous approximation of the number of classification errors. Unlike generalized margin losses, the LVQ2 loss is non-convex in $E(W, Y^i, X^i)$ and $E(W, \bar{Y}^i, X^i)$.

MCE Loss: The Minimum Classification Error loss was originally proposed by Juang et al. in the context of discriminative training for speech recognition systems [Juang et al., 1997]. The motivation was to build a loss function that also approximately counts the number of classification errors, while being smooth and differentiable. The number of classification errors can be written as:

$$\theta\left(E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right), \quad (14)$$

where θ is the step function (equal to zero for negative arguments, and 1 for positive arguments). However, this function is not differentiable, and therefore very difficult to optimize. The MCE Loss "softens" it with a sigmoid:

$$L_{\text{mce}}(W, Y^i, X^i) = \sigma\left(E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right), \quad (15)$$

where σ is the logistic function $\sigma(x) = (1 + e^{-x})^{-1}$. As with the LVQ2 loss, the saturation ensures that mistakes contribute a nominal cost to the overall loss. Although the MCE loss does not have an explicit margin, it does create a gap between $E(W, Y^i, X^i)$ and $E(W, \bar{Y}^i, X^i)$. The MCE loss is non-convex.

Square-Square Loss: Unlike the hinge loss, the square-square loss treats the energy of the correct answer and the most offending answer separately [LeCun and Huang, 2005, Hadsell et al., 2006]:

$$L_{\text{sq}-\text{sq}}(W, Y^i, X^i) = E(W, Y^i, X^i)^2 + \left(\max(0, m - E(W, \bar{Y}^i, X^i))\right)^2. \quad (16)$$

Large values of $E(W, Y^i, X^i)$ and small values of $E(W, \bar{Y}^i, X^i)$ below the margin m are both penalized quadratically (see Figure 5). Unlike the margin loss, the square-square loss "pins down" the correct answer energy at zero and "pins down" the incorrect answer energies above m. Therefore, it is only suitable for energy functions that are bounded below by zero, notably in architectures whose output module measures some sort of distance.

Square-Exponential [LeCun and Huang, 2005, Chopra et al., 2005, Osadchy et al., 2005]: The *square-exponential* loss is similar to the *square-square* loss. It only differs in the contrastive term: instead of a quadratic term it has the exponential of the negative energy of the most offending incorrect answer:

$$L_{\text{sq}-\exp}(W, Y^i, X^i) = E(W, Y^i, X^i)^2 + \gamma e^{-E(W, \bar{Y}^i, X^i)}, \quad (17)$$

where γ is a positive constant. Unlike the square-square loss, this loss has an infinite margin and pushes the energy of the incorrect answers to infinity with exponentially decreasing force.

2.2.4 Negative Log-Likelihood Loss

The motivation for the negative log-likelihood loss comes from probabilistic modeling. It is defined as:

$$L_{\text{nll}}(W, Y^i, X^i) = E(W, Y^i, X^i) + \mathcal{F}_\beta(W, \mathcal{Y}, X^i). \tag{18}$$

Where $\mathcal{F}$ is the *free energy* of the ensemble $\{E(W, y, X^i),\ y \in \mathcal{Y}\}$:

$$\mathcal{F}_\beta(W, \mathcal{Y}, X^i) = \frac{1}{\beta} \log \left(\int_{y \in \mathcal{Y}} \exp\left(-\beta E(W, y, X^i)\right) \right). \tag{19}$$

where β is a positive constant akin to an inverse temperature. This loss can only be used if the exponential of the negative energy is integrable over $\mathcal{Y}$, which may not be the case for some choices of energy function or $\mathcal{Y}$.

The form of the negative log-likelihood loss stems from a probabilistic formulation of the learning problem in terms of the maximum conditional probability principle. Given the training set $\mathcal{S}$, we must find the value of the parameter that maximizes the conditional probability of all the answers given all the inputs in the training set. Assuming that the samples are independent, and denoting by $P(Y^i|X^i, W)$ the conditional probability of Y^i given X^i that is produced by our model with parameter W, the conditional probability of the training set under the model is a simple product over samples:

$$P(Y^1, \ldots, Y^P | X^1, \ldots, X^P, W) = \prod_{i=1}^{P} P(Y^i | X^i, W). \tag{20}$$

Applying the maximum likelihood estimation principle, we seek the value of W that maximizes the above product, or the one that minimizes the negative $\log$ of the above product:

$$-\log \prod_{i=1}^{P} P(Y^i | X^i, W) = \sum_{i=1}^{P} -\log P(Y^i | X^i, W). \tag{21}$$

Using the Gibbs distribution (Equation 2), we get:

$$-\log \prod_{i=1}^{P} P(Y^i | X^i, W) = \sum_{i=1}^{P} \beta E(W, Y^i, X^i) + \log \int_{y \in \mathcal{Y}} e^{-\beta E(W, y, X^i)}. \tag{22}$$

The final form of the negative log-likelihood loss is obtained by dividing the above expression by P and β (which has no effect on the position of the minimum):

$$\mathcal{L}_{\text{nll}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} \left(E(W, Y^i, X^i) + \frac{1}{\beta} \log \int_{y \in \mathcal{Y}} e^{-\beta E(W, y, X^i)} \right). \tag{23}$$

While many of the previous loss functions involved only $E(W, \bar{Y}^i, X^i)$ in their contrastive term, the negative log-likelihood loss combines all the energies for all values of Y in its contrastive term $\mathcal{F}_\beta(W, \mathcal{Y}, X^i)$. This term can be interpreted as the Helmholtz free energy (log partition function) of the ensemble of systems with energies $E(W, Y, X^i),\ Y \in \mathcal{Y}$. This contrastive term causes the energies of all the answers to be pulled up. The energy of the correct answer is also pulled up, but not as hard as it is pushed down by the first term. This can be seen in the expression of the gradient for a single sample:

$$\frac{\partial L_{\text{nll}}(W, Y^i, X^i)}{\partial W} = \frac{\partial E(W, Y^i, X^i)}{\partial W} - \int_{Y \in \mathcal{Y}} \frac{\partial E(W, Y, X^i)}{\partial W} P(Y|X^i, W), \quad (24)$$

where $P(Y|X^i, W)$ is obtained through the Gibbs distribution:

$$P(Y|X^i, W) = \frac{e^{-\beta E(W,Y,X^i)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(W,y,X^i)}}. \quad (25)$$

Hence, the contrastive term pulls up on the energy of each answer with a force proportional to the likelihood of that answer under the model. Unfortunately, there are many interesting models for which computing the integral over $\mathcal{Y}$ is intractable. Evaluating this integral is a major topic of research. Considerable efforts have been devoted to approximation methods, including clever organization of the calculations, Monte-Carlo sampling methods, and variational methods. While these methods have been devised as approximate ways of minimizing the NLL loss, they can be viewed in the energy-based framework as different strategies for choosing the Y's whose energies will be pulled up.

Interestingly, the NLL loss reduces to the generalized perceptron loss when $\beta \to \infty$ (zero temperature), and reduces to the log loss (Eq. 12) when $\mathcal{Y}$ has two elements (e.g. binary classification).

The NLL loss has been used extensively by many authors under various names. In the neural network classification literature, it is known as the *cross-entropy loss* [Solla et al., 1988]. It was also used by Bengio et al. to train an energy-based language model [Bengio et al., 2003]. It has been widely used under the name *maximum mutual information estimation* for discriminatively training speech recognition systems since the late 80's, including hidden Markov models with mixtures of Gaussians [Bahl et al., 1986], and HMM-neural net hybrids [Bengio et al., 1990, Bengio et al., 1992, Haffner, 1993, Bengio, 1996]. It has also been used extensively for global discriminative training of handwriting recognition systems that integrate neural nets and hidden Markov models under the names *maximum mutual information* [Bengio et al., 1993, LeCun and Bengio, 1994, Bengio et al., 1995, LeCun et al., 1997, Bottou et al., 1997] and *discriminative forward training* [LeCun et al., 1998a]. Finally, it is the loss function of choice for training other probabilistic discriminative sequence labeling models such as input/output HMM [Bengio and Frasconi, 1996], conditional random fields [Lafferty et al., 2001], and discriminative random fields [Kumar and Hebert, 2004].

Minimum Empirical Error Loss: Some authors have argued that the negative log likelihood loss puts too much emphasis on mistakes: Eq. 20 is a product whose value is dominated by its smallest term. Hence, Ljolje et al. [Ljolje et al., 1990] proposed the *minimum empirical error loss*, which combines the conditional probabilities of the samples additively instead of multiplicatively:

$$L_{\text{mee}}(W, Y^i, X^i) = 1 - P(Y^i|X^i, W). \tag{26}$$

Substituting Equation 2 we get:

$$L_{\text{mee}}(W, Y^i, X^i) = 1 - \frac{e^{-\beta E(W,Y^i,X^i)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(W,y,X^i)}}. \tag{27}$$

As with the MCE loss and the LVQ2 loss, the MEE loss saturates the contribution of any single error. This makes the system more robust to label noise and outliers, which is of particular importance to such applications such as speech recognition, but it makes the loss non-convex. As with the NLL loss, MEE requires evaluating the partition function.

3 Simple Architectures

To substantiate the ideas presented thus far, this section demonstrates how simple models of classification and regression can be formulated as energy-based models. This sets the stage for the discussion of good and bad loss functions, as well as for the discussion of advanced architectures for structured prediction.

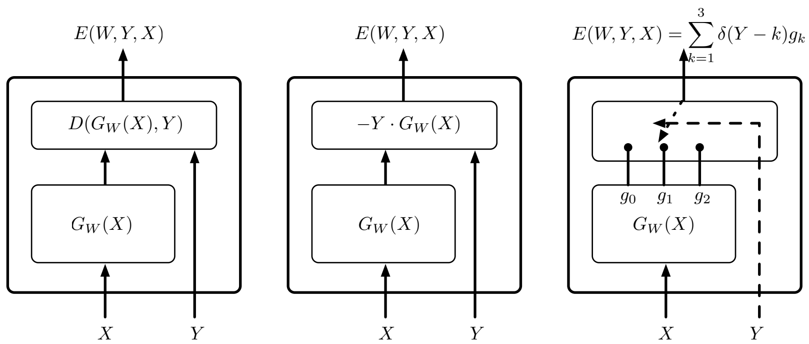


Figure 6: *Simple learning models viewed as EBMs:* **(a) a regressor:** *The energy is the discrepancy between the output of the regression function $G_W(X)$ and the answer Y. The best inference is simply $Y^* = G_W(X)$;* **(b) a simple two-class classifier:** *The set of possible answers is $\{-1, +1\}$. The best inference is $Y^* = \text{sign}(G_W(X))$;* **(c) a multiclass classifier:** *The discriminant function produces one value for each of the three categories. The answer, which can take three values, controls the position of a "switch", which connects one output of the discriminant function to the energy function. The best inference is the index of the smallest output component of $G_W(X)$.*

3.1 Regression

Figure 6(a) shows a simple architecture for regression or function approximation. The energy function is the squared error between the output of a regression function $G_W(X)$ and the variable to be predicted Y, which may be a scalar or a vector:

$$E(W, Y, X) = \frac{1}{2}||G_W(X) - Y||^2. \quad (28)$$

The inference problem is trivial: the value of Y that minimizes E is equal to $G_W(X)$. The minimum energy is always equal to zero. When used with this architecture, the energy loss, perceptron loss, and negative log-likelihood loss are all equivalent because the contrastive term of the perceptron loss is zero, and that of the NLL loss is constant (it is a Gaussian integral with a constant variance):

$$\mathcal{L}_{\text{energy}}(W, \mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P} E(W, Y^i, X^i) = \frac{1}{2P}\sum_{i=1}^{P} ||G_W(X^i) - Y^i||^2. \quad (29)$$

This corresponds to standard regression with mean-squared error.

A popular form of regression occurs when G is a linear function of the parameters:

$$G_W(X) = \sum_{k=1}^{N} w_k \phi_k(X) = W^T \Phi(X). \quad (30)$$

The $\phi_k(X)$ are a set of N *features*, and w_k are the components of an N-dimensional parameter vector W. For concision, we use the vector notation $W^T\Phi(X)$, where W^T denotes the transpose of W, and $\Phi(X)$ denotes the vector formed by each $\phi_k(X)$. With this linear parameterization, training with the energy loss reduces to an easily solvable least-squares minimization problem, which is convex:

$$W^* = \text{argmin}_W \left[\frac{1}{2P}\sum_{i=1}^{P} ||W^T\Phi(X^i) - Y^i||^2\right]. \quad (31)$$

In simple models, the feature functions are hand-crafted by the designer, or separately trained from unlabeled data. In the dual form of kernel methods, they are defined as $\phi_k(X) = K(X, X^k),\ k = 1 \ldots P$, where K is the kernel function. In more complex models such as multilayer neural networks and others, the ϕ's may themselves be parameterized and subject to learning, in which case the regression function is no longer a linear function of the parameters and hence the loss function may not be convex in the parameters.

3.2 Two-Class Classifier

Figure 6(b) shows a simple two-class classifier architecture. The variable to be predicted is binary: $\mathcal{Y} = \{-1, +1\}$. The energy function can be defined as:

$$E(W, Y, X) = -Y G_W(X), \quad (32)$$

where $G_W(X)$ is a scalar-valued *discriminant function* parameterized by W. Inference is trivial:

$$Y^* = \text{argmin}_{Y \in \{-1,1\}} - Y G_W(X) = \text{sign}(G_W(X)). \tag{33}$$

Learning can be done using a number of different loss functions, which include the perceptron loss, hinge loss, and negative log-likelihood loss. Substituting Equations 32 and 33 into the perceptron loss (Eq. 7), we get:

$$\mathcal{L}_{\text{perceptron}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} \left(\text{sign}(G_W(X^i)) - Y^i\right) G_W(X^i). \tag{34}$$

The stochastic gradient descent update rule to minimize this loss is:

$$W \leftarrow W + \eta \left(Y^i - \text{sign}(G_W(X^i)\right) \frac{\partial G_W(X^i)}{\partial W}, \tag{35}$$

where η is a positive step size. If we choose $G_W(X)$ in the family of linear models, the energy function becomes $E(W, Y, X) = -Y W^T \Phi(X)$ and the perceptron loss becomes:

$$\mathcal{L}_{\text{perceptron}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} \left(\text{sign}(W^T \Phi(X^i)) - Y^i\right) W^T \Phi(X^i), \tag{36}$$

and the stochastic gradient descent update rule becomes the familiar perceptron learning rule: $W \leftarrow W + \eta \left(Y^i - \text{sign}(W^T \Phi(X^i))\right) \Phi(X^i)$.

The hinge loss (Eq. 11) with the two-class classifier energy (Eq. 32) yields:

$$\mathcal{L}_{\text{hinge}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} \max(0, m + 2Y^i G_W(X^i)). \tag{37}$$

Using this loss with $G_W(X) = W^T X$ and a regularizer of the form $||W||^2$ gives the familiar linear support vector machine.

The negative log-likelihood loss (Eq. 23) with Equation 32 yields:

$$\mathcal{L}_{\text{nll}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} \left[-Y^i G_W(X^i) + \log\left(e^{Y^i G_W(X^i)} + e^{-Y^i G_W(X^i)}\right)\right]. \tag{38}$$

Using the fact that $\mathcal{Y} = \{-1, +1\}$, we obtain:

$$\mathcal{L}_{\text{nll}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} \log\left(1 + e^{-2Y^i G_W(X^i)}\right), \tag{39}$$

which is equivalent to the log loss (Eq. 12). Using a linear model as described above, the loss function becomes:

$$\mathcal{L}_{\text{nll}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} \log\left(1 + e^{-2Y^i W^T \Phi(X^i)}\right). \tag{40}$$

This particular combination of architecture and loss is the familiar *logistic regression* method.

3.3 Multiclass Classifier

Figure 6(c) shows an example of architecture for multiclass classification for 3 classes. A discriminant function $G_W(X)$ produces an output vector $[g_1, g_2, \ldots, g_C]$ with one component for each of the C categories. Each component g_j can be interpreted as a "penalty" for assigning X to the j^{th} category. A discrete switch module selects which of the components is connected to the output energy. The position of the switch is controlled by the discrete variable $Y \in \{1, 2, \ldots, C\}$, which is interpreted as the category. The output energy is equal to $E(W, Y, X) = \sum_{j=1}^{C} \delta(Y - j) g_j$, where $\delta(Y - j)$ is the Kronecker delta function: $\delta(u) = 1$ for $u = 0$; $\delta(u) = 0$ otherwise. Inference consists in setting Y to the index of the smallest component of $G_W(X)$.

The perceptron loss, hinge loss, and negative log-likelihood loss can be directly translated to the multiclass case.

3.4 Implicit Regression

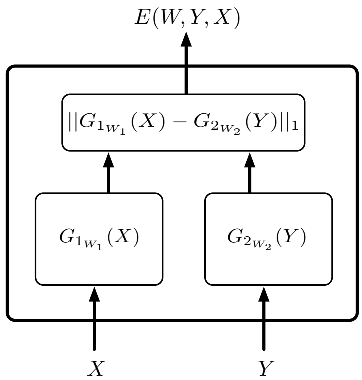


Figure 7: *The implicit regression architecture. X and Y are passed through two functions $G_{1_{W_1}}$ and $G_{2_{W_2}}$. This architecture allows multiple values of Y to have low energies for a given X.*

The architectures described in the previous section are simple functions of Y with a single minimum within the set $\mathcal{Y}$. However, there are tasks for which multiple answers are equally good. Examples include robot navigation, where turning left or right may get around an obstacle equally well, or a language model in which the sentence segment "the cat ate the" can be followed equally well by "mouse" or "bird".

More generally, the dependency between X and Y sometimes cannot be expressed as a function that maps X to Y (e.g., consider the constraint $X^2 + Y^2 = 1$). In this case, which we call *implicit regression*, we model the constraint that X and Y must satisfy and design the energy function such that it measures the violation of the constraint. Both X and Y can be passed through functions, and the energy is a function of their outputs. A simple example is:

$$E(W, Y, X) = \frac{1}{2} ||G_X(W_X, X) - G_Y(W_Y, Y)||^2. \tag{41}$$

For some problems, the function G_X must be different from the function G_Y. In other cases, G_X and G_Y must be instances of the same function G. An interesting example is the *Siamese* architecture [Bromley et al., 1993]: variables X_1 and X_2 are passed through two instances of a function G_W. A binary label Y determines the constraint on $G_W(X_1)$ and $G_W(X_2)$: if $Y = 0$, $G_W(X_1)$ and $G_W(X_2)$ should be equal, and if $Y = 1$, $G_W(X_1)$ and $G_W(X_2)$ should be different. In this way, the regression on X_1 and X_2 is implicitly learned through the constraint Y rather than explicitly learned through supervision. Siamese architectures are used to learn similarity metrics with labeled examples. When two input samples X_1 and X_2 are known to be similar (e.g. two pictures of the same person), $Y = 0$; when they are different, $Y = 1$.

Siamese architectures were originally designed for signature verification [Bromley et al., 1993]. More recently they have been used with the square-exponential loss (Eq. 17) to learn a similarity metric with application to face recognition [Chopra et al., 2005]. They have also been used with the square-square loss (Eq. 16) for unsupervised learning of manifolds [Hadsell et al., 2006].

In other applications, a single non-linear function combines X and Y. An example of such architecture is the trainable language model of Bengio et al [Bengio et al., 2003]. Under this model, the input X is a sequence of a several successive words in a text, and the answer Y is the the next word in the text. Since many different words can follow a particular word sequence, the architecture must allow multiple values of Y to have low energy. The authors used a multilayer neural net as the function $G(W, X, Y)$, and chose to train it with the negative log-likelihood loss. Because of the high cardinality of $\mathcal{Y}$ (equal to the size of the English dictionary), they had to use approximations (importance sampling) and had to train the system on a cluster machine.

The current section often referred to architectures in which the energy was linear or quadratic in W, and the loss function was convex in W, but it is important to keep in mind that much of the discussion applies equally well to more complex architectures, as we will see later.

4 Latent Variable Architectures

Energy minimization is a convenient way to represent the general process of reasoning and inference. In the usual scenario, the energy is minimized with respect to the variables to be predicted Y, given the observed variables X. During training, the correct value of Y is given for each training sample. However there are numerous applications where it is convenient to use energy functions that depend on a set of hidden variables Z whose correct value is never (or rarely) given to us, even during training. For example, we could imagine training the face detection system depicted in Figure 2(b) with data for which the scale and pose information of the faces is not available. For these architectures, the inference process for a given set of variables X and Y involves minimizing over these unseen variables Z:

$$E(Y, X) = \min_{Z \in \mathcal{Z}} E(Z, Y, X). \tag{42}$$

Such hidden variables are called *latent variables*, by analogy with a similar concept in probabilistic modeling. The fact that the evaluation of $E(Y, X)$ involves a minimization over Z does not significantly impact the approach described so far, but the use of latent variables is so ubiquitous that it deserves special treatment.

In particular, some insight can be gained by viewing the inference process in the presence of latent variables as a simultaneous minimization over Y and Z:

$$Y^* = \text{argmin}_{Y \in \mathcal{Y}, Z \in \mathcal{Z}} E(Z, Y, X). \tag{43}$$

Latent variables can be viewed as intermediate results on the way to finding the best output Y. At this point, one could argue that there is no conceptual difference between the Z and Y variables: Z could simply be folded into Y. The distinction arises during training: we are given the correct value of Y for a number of training samples, but we are never given the correct value of Z.

Latent variables are very useful in situations where a hidden characteristic of the process being modeled can be inferred from observations, but cannot be predicted directly. One such example is in recognition problems. For example, in face recognition the gender of a person or the orientation of the face could be a latent variable. Knowing these values would make the recognition task much easier. Likewise in invariant object recognition the pose parameters of the object (location, orientation, scale) or the illumination could be latent variables. They play a crucial role in problems where segmentation of the sequential data must be performed simultaneously with the recognition task. A good example is speech recognition, in which the segmentation of sentences into words and words into phonemes must take place simultaneously with recognition, yet the correct segmentation into phonemes is rarely available during training. Similarly, in handwriting recognition, the segmentation of words into characters should take place simultaneously with the recognition. The use of latent variables in face recognition is discussed in this section, and Section 7.3 describes a latent variable architecture for handwriting recognition.

4.1 An Example of Latent Variable Architecture

To illustrate the concept of latent variables, we consider the task of face detection, beginning with the simple problem of determining whether a face is present or not in a small image. Imagine that we are provided with a face detecting function $G_{\text{face}}(X)$ which takes a small image window as input and produces a scalar output. It outputs a small value when a human face fills the input image, and a large value if no face is present (or if only a piece of a face or a tiny face is present). An energy-based face detector built around this function is shown in Figure 8(a). The variable Y controls the position of a binary switch (1 = "face", 0 = "non-face"). The output energy is equal to $G_{\text{face}}(X)$ when $Y = 1$, and to a fixed threshold value T when $Y = 0$:

$$E(Y, X) = Y G_{\text{face}}(X) + (1 - Y)T.$$

The value of Y that minimizes this energy function is 1 (face) if $G_{\text{face}}(X) < T$ and 0 (non-face) otherwise.

Let us now consider the more complex task of *detecting and locating* a single face in a large image. We can apply our $G_{\text{face}}(X)$ function to multiple windows in the large image, compute which window produces the lowest value of $G_{\text{face}}(X)$, and detect a

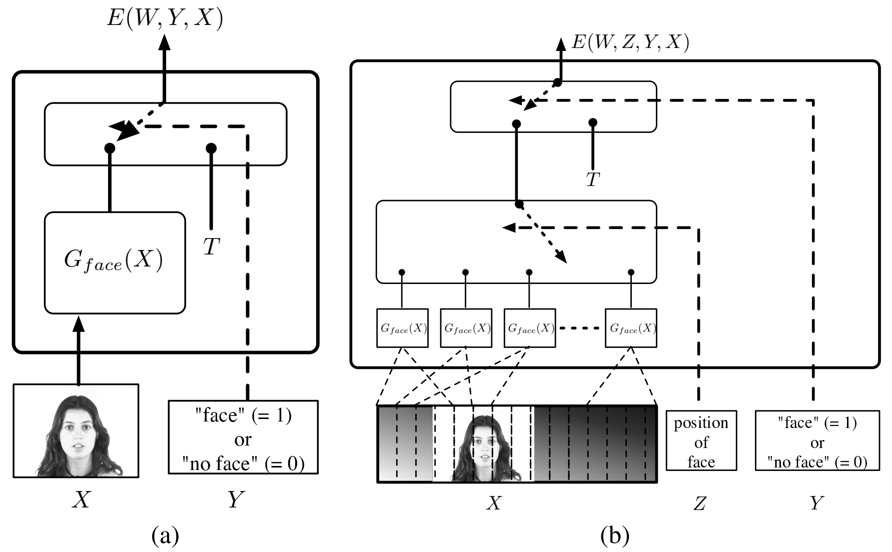


Figure 8: (a): *Architecture of an energy-based face detector. Given an image, it outputs a small value when the image is filled with a human face, and a high value equal to the threshold T when there is no face in the image.* (b): *Architecture of an energy-based face detector that simultaneously locates and detects a face in an input image by using the location of the face as a latent variable.*

face at that location if the value is lower than T. This process is implemented by the energy-based architecture shown in Figure 8(b). The latent "location" variable Z selects which of the K copies of the G_{face} function is routed to the output energy. The energy function can be written as

$$E(Z, Y, X) = Y \left[\sum_{k=1}^{K} \delta(Z - k) G_{\text{face}}(X_k) \right] + (1 - Y)T, \tag{44}$$

where the X_k's are the image windows. Locating the best-scoring location in the image consists in minimizing the energy with respect to Y and Z. The resulting value of Y will indicate whether a face was found, and the resulting value of Z will indicate the location.

4.2 Probabilistic Latent Variables

When the best value of the latent variable for a given X and Y is ambiguous, one may consider combining the contributions of the various possible values by marginalizing over the latent variables instead of minimizing with respect to those variables.

When latent variables are present, the joint conditional distribution over Y and Z

given by the Gibbs distribution is:

$$P(Z, Y|X) = \frac{e^{-\beta E(Z,Y,X)}}{\int_{y \in \mathcal{Y},\ z \in \mathcal{Z}} e^{-\beta E(y,z,X)}}. \tag{45}$$

Marginalizing over Z gives:

$$P(Y|X) = \frac{\int_{z \in \mathcal{Z}} e^{-\beta E(Z,Y,X)}}{\int_{y \in \mathcal{Y},\ z \in \mathcal{Z}} e^{-\beta E(y,z,X)}}. \tag{46}$$

Finding the best Y after marginalizing over Z reduces to:

$$Y^* = \text{argmin}_{Y \in \mathcal{Y}} - \frac{1}{\beta} \log \int_{z \in \mathcal{Z}} e^{-\beta E(z,Y,X)}. \tag{47}$$

This is actually a conventional energy-based inference in which the energy function has merely been redefined from $E(Z, Y, X)$ to $\mathcal{F}(\mathcal{Z}) = -\frac{1}{\beta} \log \int_{z \in \mathcal{Z}} e^{-\beta E(z,Y,X)}$, which is the *free energy* of the ensemble $\{E(z, Y, X),\ z \in \mathcal{Z}\}$. The above inference formula by marginalization reduces to the previous inference formula by minimization when $\beta \to \infty$ (zero temperature).

5 Analysis of Loss Functions for Energy-Based Models

This section discusses the conditions that a loss function must satisfy so that its minimization will result in a model that produces the correct answers. To give an intuition of the problem, we first describe simple experiments in which certain combinations of architectures and loss functions are used to learn a simple dataset, with varying results. A more formal treatment follows in Section 5.2.

5.1 "Good" and "Bad" Loss Functions

Consider the problem of learning a function that computes the square of a number: $Y = f(X)$, where $f(X) = X^2$. Though this is a trivial problem for a learning machine, it is useful for demonstrating the issues involved in the design of an energy function and loss function that work together. For the following experiments, we use a training set of 200 samples (X^i, Y^i) where $Y^i = {X^i}^2$, randomly sampled with a uniform distribution between -1 and $+1$.

First, we use the architecture shown in Figure 9(a). The input X is passed through a parametric function G_W, which produces a scalar output. The output is compared with the desired answer using the absolute value of the difference ($L1$ norm):

$$E(W, Y, X) = ||G_W(X) - Y||_1. \tag{48}$$

Any reasonable parameterized family of functions could be used for G_W. For these experiments, we chose a two-layer neural network with 1 input unit, 20 hidden units (with sigmoids) and 1 output unit. Figure 10(a) shows the initial shape of the energy

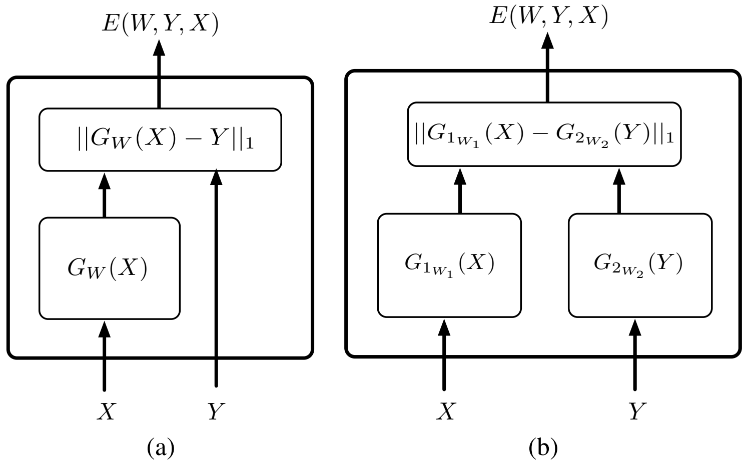


Figure 9: (a): *A simple architecture that can be trained with the* energy *loss.* (b): *An implicit regression architecture where X and Y are passed through functions $G_{1_{W_1}}$ and $G_{2_{W_2}}$ respectively. Training this architecture with the energy loss causes a collapse (a flat energy surface). A loss function with a contrastive term corrects the problem.*

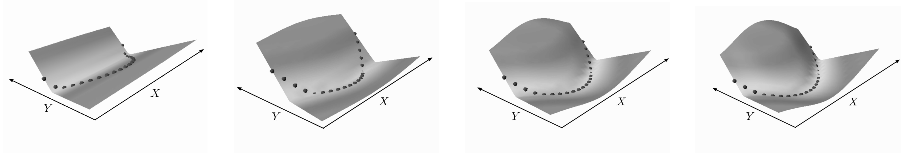
function in the space of the variables X and Y, using a set of random initial parameters W. The dark spheres mark the location of a few training samples.

First, the simple architecture is trained with the energy loss (Eq. 6):

$$\mathcal{L}_{\text{energy}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} E(W, Y^i, X^i) = \frac{1}{P} \sum_{i=1}^{P} ||G_W(X) - Y||_1. \tag{49}$$

This corresponds to a classical form of robust regression. The learning process can be viewed as pulling down on the energy surface at the location of the training samples (the spheres in Figure 10), without considering the rest of the points on the energy surface. The energy surface as a function of Y for any X has the shape of a V with fixed slopes. By changing the function $G_W(X)$, the apex of that V can move around for different X^i. The loss is minimized by placing the apex of the V at the position $Y = X^2$ for any value of X, and this has the effect of making the energies of all other answers larger, because the V has a single minimum. Figure 10 shows the shape of the energy surface at fixed intervals during training with simple stochastic gradient descent. The energy surface takes the proper shape after a few iterations through the training set. Using more sophisticated loss functions such as the NLL loss or the perceptron loss would produce exactly the same result as the energy loss because, with this simple architecture, their contrastive term is constant.

Consider a slightly more complicated architecture, shown in Figure 9(b), to learn the same dataset. In this architecture X is passed through function $G_{1_{W_1}}$ and Y is passed through function $G_{2_{W_2}}$. For the experiment, both functions were two-layer neural networks with 1 input unit, 10 hidden units and 10 output units. The energy is



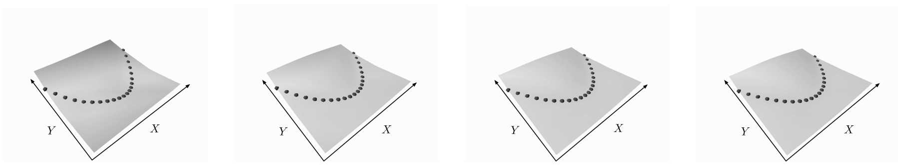
(a) (b) (c) (d)

Figure 10: *The shape of the energy surface at four intervals while training the system in Figure 9(a) with stochastic gradient descent to minimize the* energy loss*. The X axis is the input, and the Y axis the output. The energy surface is shown (a) at the start of training, (b) after 10 epochs through the training set, (c) after 25 epochs, and (d) after 39 epochs. The energy surface has attained the desired shape where the energy around training samples (dark spheres) is low and energy at all other points is high.*

the L_1 norm of the difference between their 10-dimensional outputs:

$$E(W, X, Y) = ||G_{1_{W_1}}(X) - G_{2_{W_2}}(Y)||_1, \tag{50}$$

where $W = [W_1 W_2]$. Training this architecture with the energy loss results in a *collapse* of the energy surface. Figure 11 shows the shape of the energy surface during training; the energy surface becomes essentially flat. What has happened? The shape of the energy as a function of Y for a given X is no longer fixed. With the energy loss, there is no mechanism to prevent G_1 and G_2 from ignoring their inputs and producing identical output values. This results in the collapsed solution: the energy surface is flat and equal to zero everywhere.



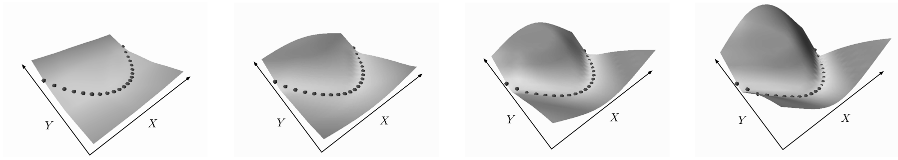
(a) (b) (c) (d)

Figure 11: *The shape of the energy surface at four intervals while training the system in Figure 9(b) using the energy loss. Along the X axis is the input variable and along the Y axis is the answer. The shape of the surface (a) at the start of the training, (b) after 3 epochs through the training set, (c) after 6 epochs, and (d) after 9 epochs. Clearly the energy is collapsing to a flat surface.*

Now consider the same architecture, but trained with the *square-square* loss:

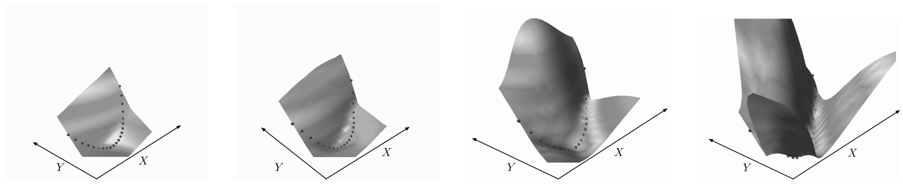
$$L(W, Y^i, X^i) = E(W, Y^i, X^i)^2 - \big(max(0, m - E(W, \bar{Y}^i, X^i))\big)^2. \tag{51}$$

Here m is a positive margin, and $\bar{Y}^i$ is the most offending incorrect answer. The second term in the loss explicitly prevents the collapse of the energy by pushing up on points whose energy threatens to go below that of the desired answer. Figure 12 shows the shape of the energy function during training; the surface successfully attains the desired shape.



(a) (b) (c) (d)

Figure 12: *The shape of the energy surface at four intervals while training the system in Figure 9(b) using* square-square *loss. Along the x-axis is the variable X and along the y-axis is the variable Y. The shape of the surface at (a) the start of the training, (b) after 15 epochs over the training set, (c) after 25 epochs, and (d) after 34 epochs. The energy surface has attained the desired shape: the energies around the training samples are low and energies at all other points are high.*



(a) (b) (c) (d)

Figure 13: *The shape of the energy surface at four intervals while training the system in Figure 9(b) using the negative log-likelihood loss. Along the X axis is the input variable and along the Y axis is the answer. The shape of the surface at (a) the start of training, (b) after 3 epochs over the training set, (c) after 6 epochs, and (d) after 11 epochs. The energy surface has quickly attained the desired shape.*

Another loss function that works well with this architecture is the *negative log-likelihood* loss:

$$L(W, Y^i, X^i) = E(W, Y^i, X^i) + \frac{1}{\beta} \log \left(\int_{y \in \mathcal{Y}} e^{-\beta E(W,y,X^i)} \right). \tag{52}$$

The first term pulls down on the energy of the desired answer, while the second term pushes up on all answers, particularly those that have the lowest energy. Note that the energy corresponding to the desired answer also appears in the second term. The shape of the energy function at various intervals using the negative log-likelihood loss is shown in Figure 13. The learning is much faster than the square-square loss. The minimum is deeper because, unlike with the square-square loss, the energies of the incorrect answers are pushed up to infinity (although with a decreasing force). However, each iteration of negative log-likelihood loss involves considerably more work because pushing up every incorrect answer is computationally expensive when no analytical expression for the derivative of the second term exists. In this experiment, a simple sampling method was used: the integral is approximated by a sum of 20 points regularly spaced between -1 and +1 in the Y direction. Each learning iteration thus requires computing the gradient of the energy at 20 locations, versus 2 locations in the case of the square-square loss. However, the cost of locating the most offending incorrect answer must be taken into account for the square-square loss.

An important aspect of the NLL loss is that it is invariant to global shifts of energy values, and only depends on differences between the energies of the Ys for a given X. Hence, the desired answer may have different energies for different X, and may not be zero. This has an important consequence: *the quality of an answer cannot be measured by the energy of that answer without considering the energies of all other answers.*

In this section we have seen the results of training four combinations of architectures and loss functions. In the first case we used a simple architecture along with a simple energy loss, which was satisfactory. The constraints in the architecture of the system automatically lead to the increase in energy of undesired answers while decreasing the energies of the desired answers. In the second case, a more complicated architecture was used with the simple energy loss and the machine collapsed for lack of a contrastive term in the loss. In the third and the fourth case the same architecture was used as in the second case but with loss functions containing explicit contrastive terms. In these cases the machine performed as expected and did not collapse.

5.2 Sufficient Conditions for Good Loss Functions

In the previous section we offered some intuitions about which loss functions are good and which ones are bad with the help of illustrative experiments. In this section a more formal treatment of the topic is given. First, a set of sufficient conditions are stated. The energy function and the loss function must satisfy these conditions in order to be guaranteed to work in an energy-based setting. Then we discuss the quality of the loss functions introduced previously from the point of view of these conditions.

5.3 Conditions on the Energy

Generally in energy-based learning, the inference method chooses the answer with minimum energy. Thus the condition for the correct inference on a sample (X^i, Y^i) is as follows.

Condition 1 *For sample (X^i, Y^i), the machine will give the correct answer for X^i if*

$$E(W, Y^i, X^i) < E(X, Y, X^i), \ \forall Y \in \mathcal{Y} \ and \ Y \neq Y^i. \tag{53}$$

In other words, the inference algorithm will give the correct answer if the energy of the desired answer Y^i is less than the energies of all the other answers Y.

To ensure that the correct answer is robustly stable, we may choose to impose that its energy be lower than energies of incorrect answers by a positive margin m. If $\bar{Y}^i$ denotes the most offending incorrect answer, then the condition for the answer to be correct by a margin m is as follows.

Condition 2 *For a variable Y and sample (X^i, Y^i) and positive margin m, the inference algorithm will give the correct answer for X^i if*

$$E(W, Y^i, X^i) < E(W, \bar{Y}^i, X^i) - m. \tag{54}$$

5.4 Sufficient Conditions on the Loss Functional

If the system is to produce the correct answers, the loss functional should be designed in such a way that minimizing it will cause $E(W, Y^i, X^i)$ to be lower than $E(W, \bar{Y}^i, X^i)$ by some margin m. Since only the relative values of those two energies matter, we only need to consider the shape of a slice of the loss functional in the 2D space of those two energies. For example, in the case where $\mathcal{Y}$ is the set of integers from 1 to k, the loss functional can be written as:

$$L(W, Y^i, X^i) = L(Y^i, E(W, 1, X^i), \ldots, E(W, k, X^i)). \tag{55}$$

The projection of this loss in the space of $E(W, Y^i, X^i)$ and $E(W, \bar{Y}^i, X^i)$ can be viewed as a function Q parameterized by the other $k - 2$ energies:

$$L(W, Y^i, X^i) = Q_{[E_y]}(E(W, Y^i, X^i), E(W, \bar{Y}^i, X^i)), \tag{56}$$

where the parameter $[E_y]$ contains the vector of energies for all values of Y except Y^i and $\bar{Y}^i$.

We assume the existence of at least one set of parameters W for which condition 2 is satisfied for a single training sample (X^i, Y^i). Clearly, if such a W does not exist, there cannot exist any loss function whose minimization would lead to condition 2. For the purpose of notational simplicity let us denote the energy $E(W, Y^i, X^i)$ associated with the training sample (X^i, Y^i) by E_C (as in "correct energy") and $E(W, \bar{Y}^i, X^i)$ by E_I (as in "incorrect energy"). Consider the plane formed by E_C and E_I. As an illustration, Figure 17(a) shows a 3-dimensional plot of the *square-square* loss function in which the abscissa is E_C and the ordinate is E_I. The third axis gives the value of the loss for the corresponding values of E_C and E_I. In general, the loss function is a family of 2D surfaces in this 3D space, where each surface corresponds to one particular configuration of all the energies except E_C and E_I. The solid red line in the figure corresponds to the points in the 2D plane for which $E_C = E_I$. The dashed blue line correspond to the margin line $E_C + m = E_I$. Let the two half planes $E_C + m < E_I$ and $E_C + m \geq E_I$ be denoted by HP_1 and HP_2 respectively.

Let R be the *feasible region*, defined as the set of values (E_C, E_I) corresponding to all possible values of $W \in \mathcal{W}$. This region may be non-convex, discontinuous, open, or one-dimensional and could lie anywhere in the plane. It is shown shaded in

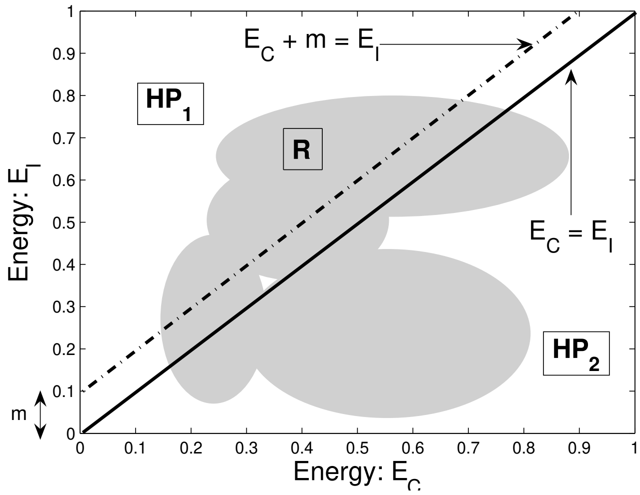


Figure 14: *Figure showing the various regions in the plane of the two energies E_C and E_I. E_C are the (correct answer) energies associated with (X^i, Y^i), and E_I are the (incorrect answer) energies associated with $(X^i, \bar{Y}^i)$.*

Figure 14. As a consequence of our assumption that a solution exists which satisfies conditions 2, R must intersect the half plane HP_1.

Let two points (e_1, e_2) and (e'_1, e'_2) belong to the feasible region R, such that $(e_1, e_2) \in HP_1$ (that is, $e_1 + m < e_2$) and $(e'_1, e'_2) \in HP_2$ (that is, $e'_1 + m \geq e'_2$). We are now ready to present the sufficient conditions on the loss function.

Condition 3 *Let (X^i, Y^i) be the i^{th} training example and m be a positive margin. Minimizing the loss function L will satisfy conditions 1 or 2 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*

$$Q_{[E_y]}(e_1, e_2) < Q_{[E_y]}(e'_1, e'_2), \tag{57}$$

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

$$L(W, Y^i, X^i) = Q_{[E_y]}(E(W, Y^i, X^i), E(W, \bar{Y}^i, X^i)). \tag{58}$$

In other words, the surface of the loss function in the space of E_C and E_I should be such that there exists at least one point in the part of the feasible region R intersecting the half plane HP_1 such that the value of the loss function at this point is less than its value at all other points in the part of R intersecting the half plane HP_2.

Note that this is only a sufficient condition and not a necessary condition. There may be loss functions that do not satisfy this condition but whose minimization still satisfies condition 2.

Table 1: *A list of loss functions, together with the margin which allows them to satisfy condition 3. A margin > 0 indicates that the loss satisfies the condition for any strictly positive margin, and "none" indicates that the loss does not satisfy the condition.*

Loss (equation #)	**Formula**	**Margin**
energy loss (6)	$E(W, Y^i, X^i)$	none
perceptron (7)	$E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i)$	0
hinge (11)	$\max\big(0, m + E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\big)$	m
log (12)	$\log\left(1 + e^{E(W,Y^i,X^i) - E(W,\bar{Y}^i,X^i)}\right)$	> 0
LVQ2 (13)	$\min\big(M, \max(0, E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\big)$	0
MCE (15)	$\left(1 + e^{-\left(E(W,Y^i,X^i) - E(W,\bar{Y}^i,X^i)\right)}\right)^{-1}$	> 0
square-square (16)	$E(W, Y^i, X^i)^2 - \big(\max(0, m - E(W, \bar{Y}^i, X^i))\big)^2$	m
square-exp (17)	$E(W, Y^i, X^i)^2 + \beta e^{-E(W,\bar{Y}^i,X^i)}$	> 0
NLL/MMI (23)	$E(W, Y^i, X^i) + \frac{1}{\beta} \log \int_{y \in \mathcal{Y}} e^{-\beta E(W,y,X^i)}$	> 0
MEE (27)	$1 - e^{-\beta E(W,Y^i,X^i)} / \int_{y \in \mathcal{Y}} e^{-\beta E(W,y,X^i)}$	> 0

5.5 Which Loss Functions are Good or Bad

Table 1 lists several loss functions, together with the value of the margin with which they satisfy condition 3. The energy loss is marked "none" because it does not satisfy condition 3 for a general architecture. The perceptron loss and the LVQ2 loss satisfy it with a margin of zero. All others satisfy condition 3 with a strictly positive value of the margin.

5.5.1 Energy Loss

The energy loss is a bad loss function in general, but there are certain forms of energies for which it is a good loss function. For example consider an energy function of the form

$$E(W, Y^i, X^i) = \sum_{k=1}^{K} \delta(Y^i - k) ||U^k - G_W(X^i)||^2. \tag{59}$$

This energy passes the output of the function G_W through K radial basis functions (one corresponding to each class) whose centers are the vectors U^k. If the centers U^k are fixed and distinct then the energy loss satisfies condition 3 and hence is a good loss function.

To see this, consider the two-class classification case (the reasoning for $K > 2$ follows along the same lines). The architecture of the system is shown in Figure 15.

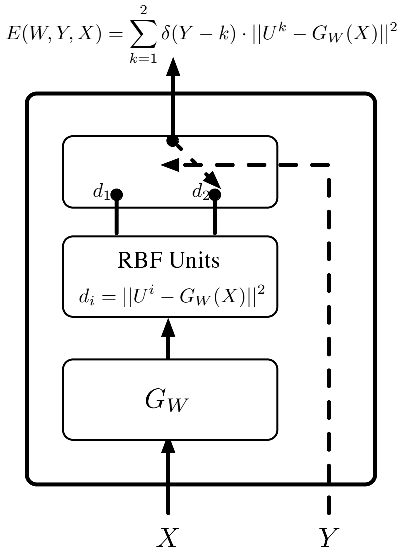


Figure 15: *The architecture of a system where two RBF units with centers U^1 and U^2 are placed on top of the machine G_W, to produce distances d_1 and d_2.*

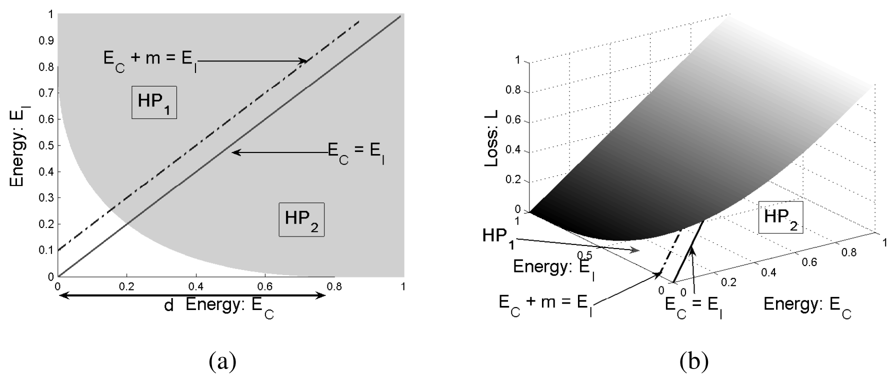


Figure 16: **(a)**: *When using the RBF architecture with fixed and distinct RBF centers, only the shaded region of the (E_C, E_I) plane is allowed. The non-shaded region is unattainable because the energies of the two outputs cannot be small at the same time. The minimum of the energy loss is at the intersection of the shaded region and vertical axis.* **(b)**: *The 3-dimensional plot of the energy loss when using the RBF architecture with fixed and distinct centers. Lighter shades indicate higher loss values and darker shades indicate lower values.*

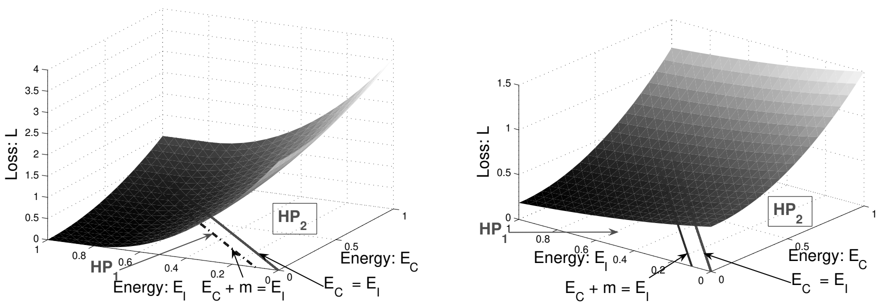
Let $d = ||U^1 - U^2||^2$, $d_1 = ||U^1 - G_W(X^i)||^2$, and $d_2 = ||U^2 - G_W(X^i)||^2$. Since U^1 and U^2 are fixed and distinct, there is a strictly positive lower bound on $d_1 + d_2$ for all G_W. Being only a two-class problem, E_C and E_I correspond directly to the energies of the two classes. In the (E_C, E_I) plane no part of the loss function exists in where $E_C + E_I \leq d$. The region where the loss function is defined is shaded in Figure 16(a). The exact shape of the loss function is shown in Figure 16(b). One can see from the figure that as long as $d \geq m$, the loss function satisfies condition 3. We conclude that this is a good loss function.

However, when the RBF centers U^1 and U^2 are not fixed and are allowed to be learned, then there is no guarantee that $d_1 + d_2 \geq d$. Then the RBF centers could become equal and the energy could become zero for all inputs, resulting in a collapsed energy surface. Such a situation can be avoided by having a contrastive term in the loss function.

5.5.2 Generalized Perceptron Loss

The generalized perceptron loss has a margin of zero. Therefore, it could lead to a collapsed energy surface and is not generally suitable for training energy-based models. However, the absence of a margin is not always fatal [LeCun et al., 1998a, Collins, 2002]. First, the set of collapsed solutions is a small piece of the parameter space. Second, although nothing prevents the system from reaching the collapsed solutions, nothing drives the system toward them either. Thus the probability of hitting a collapsed solution is quite small.

5.5.3 Generalized Margin Loss



(a) (b)

Figure 17: **(a)** *The* square-square *loss in the space of energies E_C and E_I). The value of the loss monotonically decreases as we move from HP_2 into HP_1, indicating that it satisfies condition 3.* **(b)** *The* square-exponential *loss in the space of energies E_C and E_I). The value of the loss monotonically decreases as we move from HP_2 into HP_1, indicating that it satisfies condition 3.*

We now consider the *square-square* and *square-exponential* losses. For the two-class case, the shape of the surface of the losses in the space of E_C and E_I is shown in Figure 17. One can clearly see that there exists at least one point (e_1, e_2) in HP_1 such that

$$Q_{[E_y]}(e_1, e_2) < Q_{[E_y]}(e_1', e_2'), \tag{60}$$

for all points (e_1', e_2') in HP_2. These loss functions satisfy condition 3.

5.5.4 Negative Log-Likelihood Loss

It is not obvious that the negative log-likelihood loss satisfies condition 3. The proof follows.

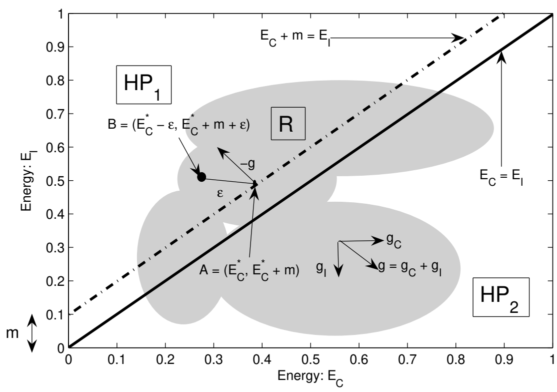


Figure 18: *Figure showing the direction of gradient of the negative log-likelihood loss in the feasible region R in the space defined by the two energies E_C and E_I.*

For any fixed parameter W and a sample (X^i, Y^i) consider the gradient of the loss with respect to the energy E_C of the correct answer Y^i and the energy E_I of the most offending incorrect answer $\bar{Y}^i$. We have

$$g_C = \frac{\partial L(W, Y^i, X^i)}{\partial E_C} = 1 - \frac{e^{-E(W,Y^i,X^i)}}{\sum_{Y \in \mathcal{Y}} e^{-E(W,Y,X^i)}}, \tag{61}$$

and

$$g_I = \frac{\partial L(W, Y^i, X^i)}{\partial E_I} = -\frac{e^{-E(W,\bar{Y}^i,X^i)}}{\sum_{Y \in \mathcal{Y}} e^{-E(W,Y,X^i)}}. \tag{62}$$

Clearly, for any value of the energies, $g_C > 0$ and $g_I < 0$. The overall direction of the gradient at any point in the space of E_C and E_I is shown in Figure 18. One can conclude that when going from HP_2 to HP_1, the loss decreases monotonically.

Now we need to show that there exists at least one point in HP_1 at which the loss is less than at all the points in HP_2. Let $A = (E_C^*, E_C^* + m)$ be a point on the margin line for which the loss is minimum. E_C^* is the value of the correct energy at this point. That is,

$$E_C^* = \operatorname{argmin}\{Q_{[E_y]}(E_C, E_C + m)\}. \tag{63}$$

Since from the above discussion, the negative of the gradient of the loss $Q_{[E_y]}$ at all points (and in particular on the margin line) is in the direction which is inside HP_1, by monotonicity of the loss we can conclude that

$$Q_{[E_y]}(E_C^*, E_C^* + m) \leq Q_{[E_y]}(E_C, E_I), \tag{64}$$

where $E_C + m > E_I$.

Consider a point B at a distance ϵ away from the point $(E_C^*, E_C^* + m)$, and inside HP_1 (see Figure 18). That is the point

$$(E_C^* - \epsilon, E_C^* + m + \epsilon). \tag{65}$$

Using the first order Taylor's expansion on the value of the loss at this point, we get

$$\begin{aligned}
&Q_{[E_y]}(E_C^* - \epsilon, E_C^* + m + \epsilon) \\
&\quad = Q_{[E_y]}(E_C^*, E_C^* + m) - \epsilon \frac{\partial Q_{[E_y]}}{\partial E_C} + \epsilon \frac{\partial Q_{[E_y]}}{\partial E_I} + O(\epsilon^2) \\
&\quad = Q_{[E_y]}(E_C^*, E_C^* + m) + \epsilon \left[\frac{\partial Q_{[E_y]}}{\partial E_C} + \frac{\partial Q_{[E_y]}}{\partial E_I} \right] \begin{bmatrix} -1 \\ 1 \end{bmatrix} + O(\epsilon^2).
\end{aligned} \tag{66}$$

From the previous discussion the second term on the right hand side is negative. So for sufficiently small ϵ we have

$$Q_{[E_y]}(E_C^* - \epsilon, E_C^* + m + \epsilon) < Q_{[E_y]}(E_C^*, E_C^* + m). \tag{67}$$

Thus we conclude that there exists at least one point in HP_1 at which the loss is less than at all points in HP_2.

Note that the energy of the most offending incorrect answer E_I is bounded above by the value of the energy of the next most offending incorrect answer. Thus we only need to consider a finite range of E_I's and the point B cannot be at infinity.

6 Efficient Inference: Non-Probabilistic Factor Graphs

This section addresses the important issue of efficient energy-based inference. Sequence labeling problems and other learning problem with structured outputs can often be modeled using energy functions whose structure can be exploited for efficient inference algorithms.

Learning and inference with EBMs involves a minimization of the energy over the set of answers $\mathcal{Y}$ and latent variables $\mathcal{Z}$. When the cardinality of $\mathcal{Y} \times \mathcal{Z}$ is large, this minimization can become intractable. One approach to the problem is to exploit the structure of the energy function in order to perform the minimization efficiently. One case where the structure can be exploited occurs when the energy can be expressed as a

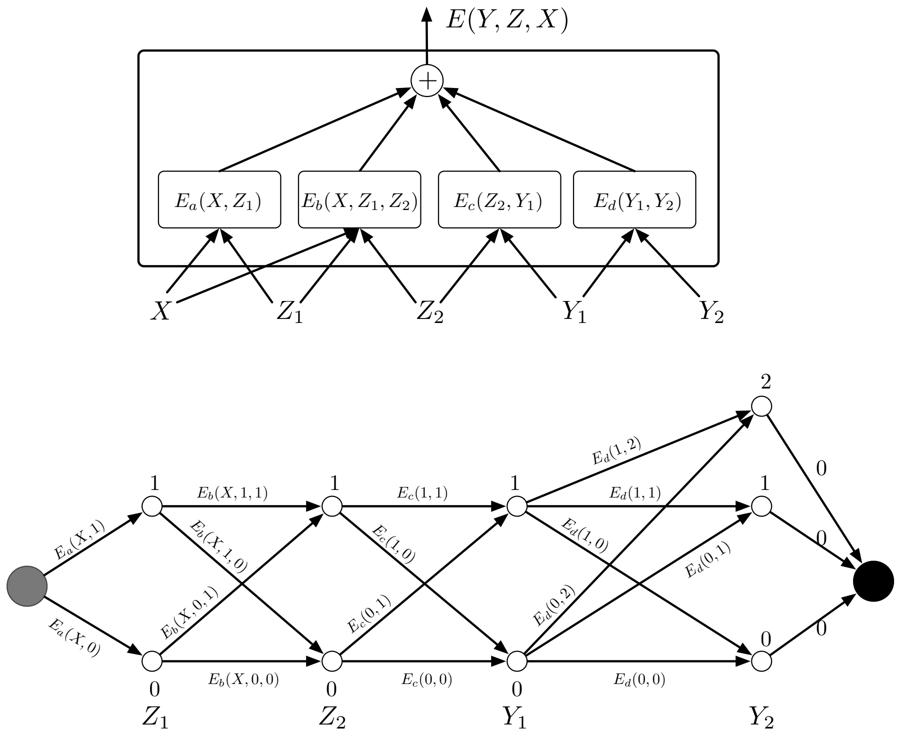


Figure 19: *Top: A log domain factor graph. The energy is a sum of factors that take different subsets of variables as inputs. Bottom: Each possible configuration of Z and Y can be represented by a path in a trellis. Here Z_1, Z_2, and Y_1 are binary variables, while Y_2 is ternary.*

sum of individual functions (called factors) that each depend on different subsets of the variables in $\mathcal{Y}$ and $\mathcal{Z}$. These dependencies are best expressed in the form of a *factor graph* [Kschischang et al., 2001, MacKay, 2003]. Factor graphs are a general form of graphical models, or belief networks.

Graphical models are normally used to represent probability distributions over variables by directly encoding the dependency relationships between variables. At first glance, it is difficult to dissociate graphical models from probabilistic modeling (witness their original name: "Bayesian networks"). However, factor graphs can be studied outside the context of probabilistic modeling, and EBM learning applies to them.

A simple example of a factor graph is shown in Figure 19 (top). The energy function is the sum of four factors:

$$E(Y, Z, X) = E_a(X, Z_1) + E_b(X, Z_1, Z_2) + E_c(Z_2, Y_1) + E_d(Y_1, Y_2), \quad (68)$$

where $Y = [Y_1, Y_2]$ are the output variables and $Z = [Z_1, Z_2]$ are the latent variables. Each factor can be seen as representing soft constraints between the values of its input variables. The inference problem consists in finding:

$$(\bar{Y}, \bar{Z}) = \operatorname{argmin}_{y \in \mathcal{Y},\, z \in \mathcal{Z}} \left(E_a(X, z_1) + E_b(X, z_1, z_2) + E_c(z_2, y_1) + E_d(y_1, y_2) \right). \quad (69)$$

This factor graph represents a *structured output* problem, because the factor E_d encodes dependencies between $Y1$ and $Y2$ (perhaps by forbidding certain combinations of values).

Let's assume that Z_1, Z_2, and Y_1 are discrete binary variables, and Y_2 is a ternary variable. The cardinality of the domain of X is immaterial since X is always observed. The number of possible configurations of Z and Y given X is $2 \times 2 \times 2 \times 3 = 24$. A naive minimization algorithm through exhaustive search would evaluate the entire energy function 24 times (96 single factor evaluations). However, we notice that for a given X, E_a only has two possible input configurations: $Z_1 = 0$ and $Z_1 = 1$. Similarly, E_b and E_c only have 4 possible input configurations, and E_d has 6. Hence, there is no need for more than $2 + 4 + 4 + 6 = 16$ single factor evaluations. The set of possible configurations can be represented by a graph (a trellis) as shown in Figure 19 (bottom). The nodes in each column represent the possible values of a single variable. Each edge is weighted by the output energy of the factor for the corresponding values of its input variables. With this representation, a single path from the start node to the end node represents one possible configuration of all the variables. The sum of the weights along a path is equal to the total energy for the corresponding configuration. Hence, the inference problem can be reduced to searching for the shortest path in this graph. This can be performed using a dynamic programming method such as the Viterbi algorithm, or the A* algorithm. The cost is proportional to the number of edges (16), which is exponentially smaller than the number of paths in general. To compute $E(Y, X) = \min_{z \in \mathcal{Z}} E(Y, z, X)$, we follow the same procedure, but we restrict the graph to the subset of arcs that are compatible with the prescribed value of Y.

The above procedure is sometimes called the *min-sum algorithm*, and it is the log domain version of the traditional max-product for graphical models. The procedure can easily be generalized to factor graphs where the factors take more than two variables as inputs, and to factor graphs that have a tree structure instead of a chain structure. However, it only applies to factor graphs that are bipartite trees (with no loops). When loops are present in the graph, the min-sum algorithm may give an approximate solution when iterated, or may not converge at all. In this case, a descent algorithm such as simulated annealing could be used.

As mentioned in Section 4, variables can be handled through minimization or through marginalization. The computation is identical to the one required for computing the contrastive term of the negative log-likelihood loss (the log partition function), hence we will make no distinctions. The contrastive term in the negative log-likelihood loss function is:

$$-\frac{1}{\beta}\log\int_{Y\in\mathcal{Y},\, z\in\mathcal{Z}} e^{-\beta E(Z,Y,X)}, \tag{70}$$

or simply

$$-\frac{1}{\beta}\log\int_{Y\in\mathcal{Y}} e^{-\beta E(Y,X)}, \tag{71}$$

when no latent variables are present.

At first, this seems intractable, but the computation can be factorized just like with the min-sum algorithm. The result is the so-called *forward algorithm* in the log domain. Values are propagated forward, starting at the start node on the left, and following the arrows in the trellis. Each node k computes a quantity α_k:

$$\alpha_k = -\frac{1}{\beta}\log\sum_j e^{-\beta(E_{kj}+\alpha_j)}, \tag{72}$$

where E_{jk} is the energy attached to the edge linking node j to node k. The final α at the end node is the quantity in Eq. 70. The procedure reduces to the min-sum algorithm for large values of β.

In a more complex factor graph with factors that take more than two variables as input, or that have a tree structure, this procedure generalizes to a non-probabilistic form of belief propagation in the log domain. For loopy graphs, the procedure can be iterated, and may lead to an approximate value for Eq. 70, if it converges at all [Yedidia et al., 2005].

The above procedures are an essential component for constructing models with structures and/or sequential output.

6.1 EBMs versus Internally Normalized Models

It is important to note that at no point in the above discussion did we need to manipulate normalized probability distributions. The only quantities that are manipulated are energies. This is in contrast with hidden Markov models and traditional Bayesian nets. In HMMs, the outgoing transition probabilities of a node must sum to 1, and the emission probabilities must be properly normalized. This ensures that the overall distribution over sequences is normalized. Similarly, in directed Bayesian nets, the rows of the conditional probability tables are normalized.

EBMs manipulate energies, so no normalization is necessary. When energies are transformed into probabilities, the normalization over Y occurs as the very last step in the process. This idea of *late normalization* solves several problems associated with the internal normalization of HMMs and Bayesian nets. The first problem is the so-called *label bias problem*, first pointed out by Bottou [Bottou, 1991]: transitions leaving a given state compete with each other, but not with other transitions in the model. Hence, paths whose states have few outgoing transitions tend to have higher probability than paths whose states have many outgoing transitions. This seems like an artificial constraint. To circumvent this problem, a late normalization scheme was first proposed by Denker and Burges in the context of handwriting and speech recognition [Denker and Burges, 1995]. Another flavor of the label bias problem is the *missing probability mass problem* discussed by LeCun et al. in [LeCun et al., 1998a]. They also make use of a late normalization scheme to solve this problem. Normalized models distribute the probability mass among all the answers that are explicitly modeled by the system. To cope with "junk" or other unforeseen and un-modeled inputs, designers must often add a so-called *background model* that takes some probability mass away from the set of explicitly-modeled answers. This could be construed as a thinly disguised way of removing the normalization constraint. To put it another way, since *every explicit normalization is another opportunity for mishandling unforeseen events*, one should strive to minimize the number of explicit normalizations in a model. A recent demonstration of successful handling of the label bias problem through normalization removal is the comparison between maximum entropy Markov models by McCallum, Freitag and Pereira [McCallum et al., 2000], and conditional random fields by Lafferty, McCallum and Pereira [Lafferty et al., 2001].

The second problem is controlling the relative importance of probability distributions of different natures. In HMMs, emission probabilities are often Gaussian mixtures in high dimensional spaces (typically 10 to 100), while transition probabilities are discrete probabilities over a few transitions. The dynamic range of the former is considerably larger than that of the latter. Hence transition probabilities count for almost nothing in the overall likelihood. Practitioners often raise the transition probabilities to some power in order to increase their influence. This trick is difficult to justify in a probabilistic framework because it breaks the normalization. In the energy-based framework, there is no need to make excuses for breaking the rules. Arbitrary coefficients can be applied to any subset of energies in the model. The normalization can always be performed at the end.

The third problem concerns discriminative learning. Discriminative training often uses iterative gradient-based methods to optimize the loss. It is often complicated, expensive, and inefficient to perform a normalization step after each parameter update by the gradient method. The EBM approach eliminates the problem [LeCun et al., 1998a]. More importantly, the very reason for internally normalizing HMMs and Bayesian nets is somewhat contradictory with the idea of training them discriminatively. The normalization is only necessary for generative models.

7 EBMs for Sequence Labeling and Structured Outputs

The problem of classifying or labeling sequences of symbols or sequences of vectors has long been a topic of great interest in several technical communities. The earliest and most notable example is speech recognition. Discriminative learning methods were proposed to train HMM-based speech recognition systems in the late 1980's [Bahl et al., 1986, Ljolje et al., 1990]. These methods for HMMs brought about a considerable improvement in the accuracy of speech recognition systems, and remains an active topic of research to this day.

With the appearance of multi-layer neural network training procedures, several groups proposed combining neural networks and time alignment methods for speech recognition. The time alignment was implemented either through elastic template matching (Dynamic Time Warping) with a set of reference words, or using a hidden Markov model. One of the main challenges was to design an integrated training method for simultaneously training the neural network and the time alignment module. In the early 1990's, several authors proposed such methods for combining neural nets and dynamic time warping [Driancourt et al., 1991a, Driancourt et al., 1991b, Driancourt and Gallinari, 1992b, Driancourt and Gallinari, 1992a, Driancourt, 1994], as well as for combining neural net and HMM [Bengio et al., 1990, Bourlard and Morgan, 1990, Bottou, 1991, Haffner et al., 1991, Haffner and Waibel, 1991, Bengio et al., 1992, Haffner and Waibel, 1992, Haffner, 1993, Driancourt, 1994, Morgan and Bourlard, 1995, Konig et al., 1996]. Extensive lists of references on the topic are available in [McDermott, 1997, Bengio, 1996]. Most approaches used one dimensional convolutional networks (*time-delay neural networks*) to build robustness to variations of pitch, voice timbre, and speed of speech. Earlier models combined discriminative classifiers with time alignment, but without integrated sequence-level training [Sakoe et al., 1988, McDermott and Katagiri, 1992, Franzini et al., 1990].

Applying similar ideas to handwriting recognition proved more challenging, because the 2D nature of the signal made the segmentation problem considerably more complicated. This task required the integration of image segmentation heuristics in order to generate segmentation hypotheses. To classify the segments with robustness to geometric distortions, 2D convolutional nets were used [Bengio et al., 1993, LeCun and Bengio, 1994, Bengio et al., 1995]. A general formulation of integrated learning of segmentation and recognition with late normalization resulted in the *graph transformer network* architecture [LeCun et al., 1997, LeCun et al., 1998a].

Detailed descriptions of several sequence labeling models in the framework of energy-based models are presented in the next three sections.

7.1 Linear Structured Models: CRF, SVMM, and MMMN

Outside of the discriminative training tradition in speech and handwriting recognition, graphical models have traditionally been seen as probabilistic generative models, and trained as such. However, in recent years, a resurgence of interest for discriminative

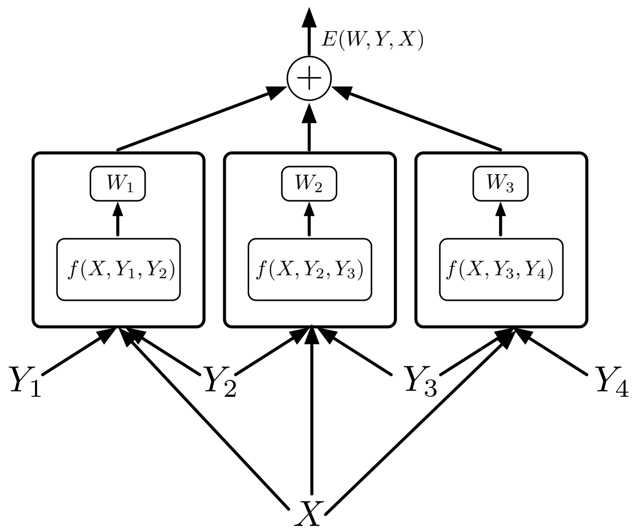


Figure 20: *A log domain factor graph for linear structured models, which include conditional random fields, support vector Markov models, and maximum margin Markov networks.*

training has emerged, largely motivated by sequence labeling problems in natural language processing, notably conditional random fields [Lafferty et al., 2001], perceptron-like models [Collins, 2002], support vector Markov models [Altun et al., 2003], and maximum margin Markov networks [Taskar et al., 2003].

These models can be easily described in an EBM setting. The energy function in these models is assumed to be a linear function of the parameters W:

$$E(W, Y, X) = W^T F(X, Y), \tag{73}$$

where $F(X, Y)$ is a vector of feature functions that depend on X and Y. The answer Y is a sequence of l individual labels $(Y_1, \ldots, Y_l)$, often interpreted as a temporal sequence. The dependencies between individual labels in the sequence is captured by a factor graph, such as the one represented in Figure 20. Each factor is a linear function of the trainable parameters. It depends on the input X and on a pair of individual labels (Y_m, Y_n). In general, each factor could depend on more than two individual labels, but we will limit the discussion to pairwise factors to simplify the notation:

$$E(W, Y, X) = \sum_{(m,n) \in \mathcal{F}} W_{mn}^T f_{mn}(X, Y_m, Y_n). \tag{74}$$

Here $\mathcal{F}$ denotes the set of factors (the set of pairs of individual labels that have a direct inter-dependency), W_{mn} is the parameter vector for factor (m, n), and $f_{mn}(X, Y_m, Y_n)$ is a (fixed) feature vector. The global parameter vector W is the concatenation of all the W_{mn}. It is sometimes assumed that all the factors encode the same kind of interaction between input and label pairs: the model is then called a homogeneous field. The factors share the same parameter vector and features, and the energy can be simplified as:

$$E(W, Y, X) = \sum_{(m,n)\in\mathcal{F}} W^T f(X, Y_m, Y_n). \tag{75}$$

The linear parameterization of the energy ensures that the corresponding probability distribution over W is in the exponential family:

$$P(W|Y, X) = \frac{e^{-W^T F(X,Y)}}{\int_{w'\in\mathcal{W}} e^{-w^T F(X,Y)}}. \tag{76}$$

This model is called the *linear structured model*.

We now describe various versions of linear structured models that use different loss functions. Sections 7.2 and 7.3 will describe non-linear and hierarchical models.

7.1.1 Perceptron Loss

The simplest way to train the linear structured model is with the perceptron loss. LeCun et al. [LeCun et al., 1998a] proposed its use for general, non-linear energy functions in sequence labeling (particularly handwriting recognition), calling it *discriminative Viterbi training*. More recently, Collins [Collins, 2000, Collins, 2002] has advocated its use for linear structured models in the context of NLP:

$$\mathcal{L}_{\text{perceptron}}(W) = \frac{1}{P}\sum_{i=1}^{P} E(W, Y^i, X^i) - E(W, Y^{*i}, X^i), \tag{77}$$

where $Y^{*i} = \text{argmin}_{y\in\mathcal{Y}} E(W, y, X^i)$ is the answer produced by the system. The linear property gives a particularly simple expression for the loss:

$$\mathcal{L}_{\text{perceptron}}(W) = \frac{1}{P}\sum_{i=1}^{P} W^T \left(F(X^i, Y^i) - F(X^i, Y^{*i})\right). \tag{78}$$

Optimizing this loss with stochastic gradient descent leads to a simple form of the perceptron learning rule:

$$W \leftarrow W - \eta \left(F(X^i, Y^i) - F(X^i, Y^{*i})\right). \tag{79}$$

As stated before, the main problem with the perceptron loss is the absence of margin, although this problem is not fatal when the energy is a linear function of the parameters, as in Collins' model. The lack of a margin, which theoretically may lead to stability problems, was overlooked in [LeCun et al., 1998a].

7.1.2 Margin Loss: Max-Margin Markov Networks

The main idea behind margin-based Markov networks [Altun et al., 2003, Altun and Hofmann, 2003, Taskar et al., 2003] is to use a margin loss to train the linearly parameterized factor graph of Figure 20, with the energy function of Equation 73. The loss function is the simple hinge loss with an L_2 regularizer:

$$\mathcal{L}_{\text{hinge}}(W) = \frac{1}{P}\sum_{i=1}^{P}\max(0, m + E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)) + \gamma||W||^2. \quad (80)$$

Because the energy is linear in W, the loss becomes particularly simple:

$$\mathcal{L}_{\text{hinge}}(W) = \frac{1}{P}\sum_{i=1}^{P}\max\left(0, m + W^T \Delta F(X^i, Y^i)\right) + \gamma||W||^2, \quad (81)$$

where $\Delta F(X^i, Y^i) = F(X^i, Y^i) - F(X^i, \bar{Y}^i)$. This loss function can be optimized with a variety of techniques. The simplest method is stochastic gradient descent. However, the hinge loss and linear parameterization allow for the use of a dual formulation as in the case of conventional support vector machines. The question of which optimization method is most suitable is not settled. As with neural net training, it is not clear whether second order methods bring a significant speed improvement over well tuned stochastic gradient methods. To our knowledge, no systematic experimental study of this issue has been published.

Altun, Johnson, and Hofman [Altun et al., 2003] have studied several versions of this model that use other loss functions, such as the exponential margin loss proposed by Collins [Collins, 2000]:

$$\mathcal{L}_{\text{hinge}}(W) = \frac{1}{P}\sum_{i=1}^{P}\exp(E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)) + \gamma||W||^2. \quad (82)$$

This loss function tends to push the energies $E(W, Y^i, X^i)$ and $E(W, \bar{Y}^i, X^i)$ as far apart as possible, an effect which is moderated only by regularization.

7.1.3 Negative Log-Likelihood Loss: Conditional Random Fields

Conditional random fields (CRF) [Lafferty et al., 2001] use the negative log-likelihood loss function to train a linear structured model:

$$\mathcal{L}_{\text{nll}}(W) = \frac{1}{P}\sum_{i=1}^{P} E(W, Y^i, X^i) + \frac{1}{\beta}\log\sum_{y\in\mathcal{Y}} e^{-\beta E(W, y, X^i)}. \quad (83)$$

The linear form of the energy (Eq. 75) gives the following expression:

$$\mathcal{L}_{\text{nll}}(W) = \frac{1}{P}\sum_{i=1}^{P} W^T F(X^i, Y^i) + \frac{1}{\beta}\log\sum_{y\in\mathcal{Y}} e^{-\beta W^T F(X^i, y)}. \quad (84)$$

Following Equation 24, the derivative of this loss with respect to W is:

$$\frac{\partial \mathcal{L}_{\text{nll}}(W)}{\partial W} = \frac{1}{P}\sum_{i=1}^{P} F(X^i, Y^i) - \sum_{y\in\mathcal{Y}} F(X^i, y) P(y|X^i, W), \quad (85)$$

where

$$P(y|X^i, W) = \frac{e^{-\beta W^T F(X^i, y)}}{\sum_{y' \in \mathcal{Y}} e^{-\beta W^T F(X^i, y')}}. \tag{86}$$

The problem with this loss function is the need to sum over all possible label combinations, as there are an exponentially large number of such combinations (2^l for a sequence of l binary labels). However, one of the efficient inference algorithms mentioned in Section 6 can be used.

One of the alleged advantages of CRFs is that the loss function is convex with respect to W. However, the convexity of the loss function, while mathematically satisfying, does not seem to be a significant practical advantage. Although the original optimization algorithm for CRF was based on iterative scaling, recent work indicates that stochastic gradient methods may be more efficient [Vishwanathan et al., 2006].

7.2 Non-Linear Graph Based EBMs

The discriminative learning methods for graphical models developed in the speech and handwriting communities in the 90's allowed for non-linear parameterizations of the factors, mainly mixtures of Gaussians and multi-layer neural nets. Non-linear factors allow the modeling of highly complex dependencies between inputs and labels (such as mapping the pixels of a handwritten word to the corresponding character labels). One particularly important aspect is the use of architectures that are invariant (or robust) to irrelevant transformations of the inputs, such as time dilation or pitch variation in speech, and geometric variations in handwriting. This is best handled by hierarchical, multi-layer architectures that can learn low level features and higher level representations in an integrated fashion. Most authors have used one dimensional convolutional nets (time-delay neural networks) for speech and pen-based handwriting [Bengio et al., 1990, Bottou, 1991, Haffner et al., 1991, Haffner and Waibel, 1991, Driancourt et al., 1991a, Driancourt et al., 1991b, Driancourt and Gallinari, 1992b, Driancourt and Gallinari, 1992a, Bengio et al., 1992, Haffner and Waibel, 1992, Haffner, 1993, Driancourt, 1994, Bengio, 1996], and 2D convolutional nets for image-based handwriting [Bengio et al., 1993, LeCun and Bengio, 1994, Bengio et al., 1995, LeCun et al., 1997, LeCun et al., 1998a].

To some observers, the recent interest in the linear structured model looks like somewhat of a throw-back to the past, and a regression on the complexity scale. One apparent advantage of linearly parameterized energies is that they make the perceptron loss, hinge loss, and NLL loss convex. It is often argued that convex loss functions are inherently better because they allow the use of efficient optimization algorithms with guaranteed convergence to the global minimum. However, several authors have recently argued that convex loss functions are no guarantee for good performance, and that non-convex losses may in fact be easier to optimize than convex ones in practice, even in the absence of theoretical guarantees [Huang and LeCun, 2006, Collobert et al., 2006].

Furthermore, it has been argued that convex loss functions can be efficiently optimized using sophisticated second-order optimization methods. However, it is a well-known but often overlooked fact that a carefully tuned stochastic gradient descent method is often considerably faster in practice than even the most sophisticated second-order optimization methods (which appear better on paper). This is because stochastic gradients can take advantage of the redundancy between the samples by updating the parameters on the basis of a single sample, whereas "batch" optimization methods waste considerable resources to compute exact descent directions, often nullifying the theoretical speed advantage [Becker and LeCun, 1989, LeCun et al., 1998a, LeCun et al., 1998b, Bottou, 2004, Bottou and LeCun, 2004, Vishwanathan et al., 2006].

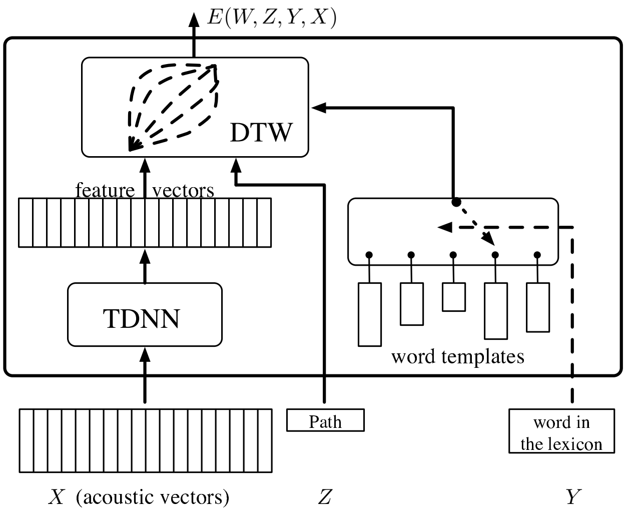


Figure 21: *Figure showing the architecture of a speech recognition system using latent variables. An acoustic signal is passed through a time-delay neural network (TDNN) to produce a high level feature vector. The feature vector is then compared to the word templates. Dynamic time warping (DTW) aligns the feature vector with the word templates so as to reduce the sensitivity of the matching to variations in pronunciation.*

Figure 21 shows an example of speech recognition system that integrates a time-delay neural network (TDNN) and word matching using dynamic time warping (DTW). The raw speech signal is first transformed into a sequence of acoustic vectors (typically 10 to 50 spectral or cepstral coefficients, every 10ms). The acoustic vector sequence is fed to a TDNN that transforms it into a sequence of high level features. Temporal subsampling in the TDNN can be used to reduce the temporal resolution of the feature vectors [Bottou, 1991]. The sequence of feature vectors is then compared to word templates. In order to reduce the sensitivity of the matching to variations in speed of pronunciation, dynamic time warping aligns the feature sequence with the template sequences. Intuitively, DTW consists in finding the best "elastic" warping that maps a sequence of vectors (or symbols) to another. The solution can be found efficiently with dynamic programming (e.g. the Viterbi algorithm or the A* algorithm).

DTW can be reduced to a search for the shortest path in a directed acyclic graph in which the cost of each node is the mismatch between two items in the two input sequences. Hence, the overall system can be seen as a latent variable EBM in which $\mathcal{Y}$ is the set of words in the lexicon, and $\mathcal{Z}$ represents the set of templates for each word, and the set of paths for each alignment graph. The earliest proposal for integrated training of neural nets and time alignment is by Driancourt and Bottou [Driancourt et al., 1991a], who proposed using the LVQ2 loss (Eq. 13) to train this system. It is a simple matter to back-propagate gradients through the DTW module and further back-propagate gradients into the TDNN in order to update the weights. Similarly, gradients can be back-propagated to the word templates in order to update them as well. Excellent results were obtained for isolated word recognition, despite the zero margin of the LVQ2 loss. A similar scheme was later used by McDermott [McDermott, 1997].

A slightly more general method consists in combining neural networks (e.g. TDNN) with hidden Markov models instead of DTW. Several authors proposed integrated training procedures for such combinations during the 90's. The first proposals were by Bengio et al. [Bengio et al., 1991, Bengio et al., 1992, Bengio, 1996] who used the NLL/MMI loss optimized with stochastic gradient descent, and Bottou [Bottou, 1991] who proposed various loss functions. A similar method was subsequently proposed by Haffner et al. in his multi-state TDNN model [Haffner and Waibel, 1992, Haffner, 1993]. Similar training methods were devised for handwriting recognition. Bengio and LeCun described a neural net/HMM hybrid with global training using the NLL/MMI loss optimized with stochastic gradient descent [Bengio et al., 1993, LeCun and Bengio, 1994]. Shortly thereafter, Konig et al. proposed the REMAP method, which applies the expectation maximization algorithm to the HMM in order to produce targets outputs for a neural net based acoustic model [Konig et al., 1996].

The basic architecture of neural net/HMM hybrid systems is similar to the one in Figure 21, except that the word (or language) models are probabilistic finite-state machines instead of sequences. The emission probabilities at each node are generally simple Gaussians operating on the output vector sequences produced by the neural net. The only challenge is to compute the gradient of the loss with respect to the neural net outputs by backpropagating gradients through the HMM trellis. Since the procedure is very similar to the one used in graph transformer networks, we refer to the next section for a description.

It should be noted that many authors had previously proposed methods that combined a separately trained discriminative classifier and an alignment method for speech and handwriting, but they did not use integrated training methods.

7.3 Hierarchical Graph-Based EBMs: Graph Transformer Networks

Sections 7.2 and 7.1 discussed models in which inference and learning involved marginalizing or minimizing over all configurations of variables of a dynamic factor graph. These operations are performed efficiently by building a trellis in which each path corresponds to a particular configuration of those variables. Section 7.2 concentrated on models where the factors are non-linear functions of the parameters, while Section 7.1

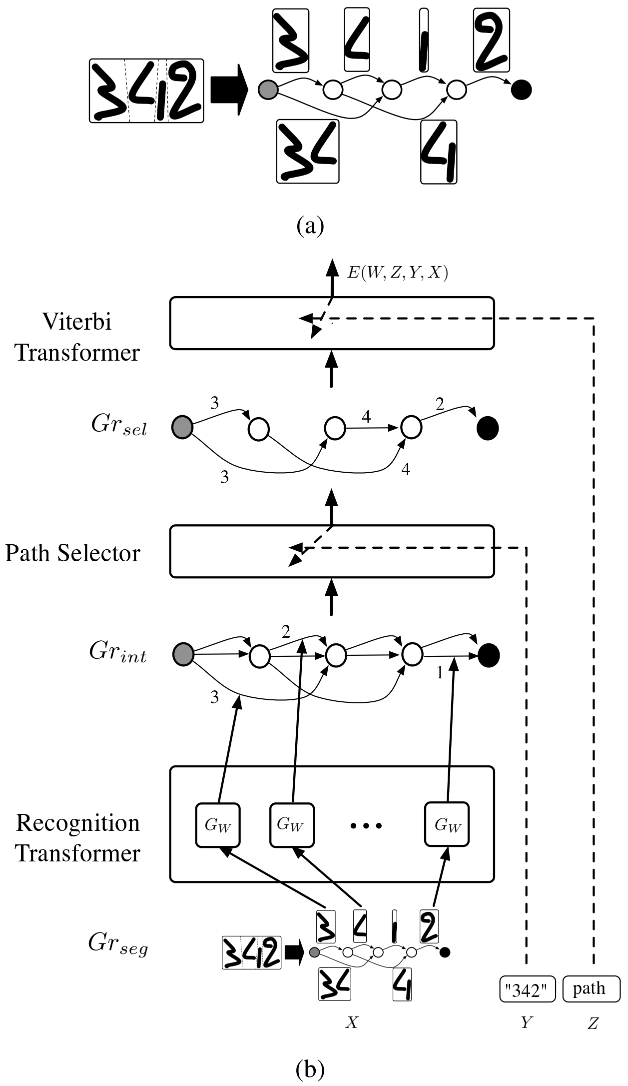


Figure 22: *The architecture of a* graph transformer network *for handwritten word recognition. (a) The segmentation graph Gr_{seg} is generated from the input image, (b) the hierarchical multi-modular architecture takes a set of graphs and outputs another set of graphs.*

focused on simpler models where the factors are linearly parameterized.

The present section discusses a class of models called *graph transformer networks* (GTN) [LeCun et al., 1998a]. GTNs are designed for situations where the sequential structure is so complicated that the corresponding dynamical factor graph cannot be explicitly represented, but must be represented *procedurally*. For example, the factor graph that must be built on-the-fly in order to recognize an entire handwritten sentence in English is extremely large. The corresponding trellis contains a path for every grammatically correct transcription of the sentence, for every possible segmentation of the sentence into characters. Generating this trellis (or its associated factor graph) in its entirety is impractical, hence the trellis must be represented procedurally. Instead of representing the factor graph, the GTN approach views the trellis as the main data structure being manipulated by the machine. A GTN can be seen as a multilayer architecture in which the states are trellises, just as a neural net is a multilayer architecture in which the states are fixed-size vectors. A GTN can be viewed as a network of modules, called *graph transformers*, that take one or more graphs as input and produces another graph as output. The operation of most modules can be expressed as the composition of the input graph with another graph, called a transducer, associated with the module [Mohri, 1997]. The objects attached to the edges of the input graphs, which can be numbers, labels, images, sequences, or any other entity, are fed to trainable functions whose outputs are attached to edge of the output graphs. The resulting architecture can be seen as a *compositional hierarchy* in which low level features and parts are combined into higher level objects through graph composition.

For speech recognition, acoustic vectors are assembled into phones, phones into triphones, triphones into words, and words into sentences. Similarly in handwriting recognition, ink segments are assembled into characters, characters into words, and words into sentences.

Figure 22 shows an example of GTN architecture for simultaneously segmenting and recognizing handwritten words [LeCun et al., 1998a]. The first step involves over-segmenting the image and generating a segmentation graph out of it (see Figure 22(a)). The segmentation graph Gr_{seg} is a directed acyclic graph (DAG) in which each path from the start node to the end node represents a particular way of segmenting the input image into character candidates. Each internal node is associated with a candidate cut produced by the segmentation. Every arc between a source and a destination node is associated with the part of the image that lies between the two cuts. Hence every piece of ink appears once and only once along each path. The next stage passes the segmentation graph Gr_{seg} through the recognition transformer which produces the interpretation graph Gr_{int} with the same number of nodes as Gr_{seg}. The recognition transformer contains as many identical copies of the discriminant functions $G_W(X)$ as there are arcs in the interpretation graph (this number changes for every new input). Each copy of G_W takes the image associated with one arc in the segmentation graph and produces several arcs between the corresponding nodes in the interpretation graph. Each output arc is labeled by a character category, and weighted by the energy of assigning the image to that category. Hence, each path in the interpretation graph represents one possible interpretation of the input for one possible segmentation, with the sum of the weights along the path representing the combined energy of that interpretation. The interpretation graph is then passed through a path selector module that selects only those paths from the interpretation graph that have the same sequence of labels as given by Y (the answer). The output of this module is another graph called Gr_{sel}. Finally a so-called Viterbi transformer selects a single path in Gr_{sel} indexed by the latent variable Z. Each value of Z corresponds to a different path in Gr_{sel}, and can be interpreted as a particular segmentation of the input. The output energy is obtained either by minimizing or by marginalizing over Z. Minimizing over Z is achieved by running a shortest path algorithm on the Gr_{sel} (e.g., the Viterbi algorithm, hence the name Viterbi transformer). The output energy is then the sum of the arc energies along the shortest path. Marginalizing over Z is achieved by running the forward algorithm on Gr_{sel}, as indicated in Section 6, equation 72. The path selector and Viterbi transformer can be seen as particular types of "switch" modules that select paths in their input graph.

In the handwriting recognition systems described in [LeCun et al., 1998a], the discriminant function $G_W(X)$ was a 2D convolutional network. This class of function is designed to learn low level features and high level representations in an integrated manner, and is therefore highly non-linear in W. Hence the loss function is not convex in W. The optimization method proposed is a refined version of stochastic gradient descent.

In [LeCun et al., 1998a], two primary methods for training GTNs are proposed: *discriminative Viterbi training*, which is equivalent to using the generalized perceptron loss (Eq. 7), and *discriminative forward training*, which is equivalent to using the negative log-likelihood loss (Eq. 23). Any of the good losses in Table 1 could also be used.

Training by minimizing the perceptron loss with stochastic gradient descent is performed by applying the following update rule:

$$W \leftarrow W - \eta \left(\frac{\partial E(W, Y^i, X^i)}{\partial W} - \frac{\partial E(W, Y^{*i}, X^i)}{\partial W} \right). \tag{87}$$

How can the gradients of $E(W, Y^i, X^i)$ and $E(W, Y^i, X^i)$ be computed? The answer is simply to back-propagate gradients through the entire structure, all the way back to the discriminant functions $G_W(X)$. The overall energy can be written in the following form:

$$E(W, Y, X) = \sum_{kl} \delta_{kl}(Y) G_{kl}(W, X), \tag{88}$$

where the sum runs over all arcs in Gr_{int}, $G_{kl}(W, X)$ is the l-th component of the k copy of the discriminant function, and $\delta_{kl}(Y)$ is a binary value equal to 1 if the arc containing $G_{kl}(W, X)$ is present in the final graph, and 0 otherwise. Hence, the gradient is simply:

$$\frac{\partial E(W, Y, X)}{\partial W} = \sum_{kl} \delta_{kl}(Y) \frac{\partial G_{kl}(W, X)}{\partial W}. \tag{89}$$

One must simply keep track of the $\delta_{kl}(Y)$.

In Section 5 we concluded that the generalized perceptron loss is not a good loss function. While the zero margin may limit the robustness of the solution, the perceptron loss seems appropriate as a way to refine a system that was pre-trained on segmented characters as suggested in [LeCun et al., 1998a]. Nevertheless, the GTN-based bank check reading system described in [LeCun et al., 1998a] that was deployed commercially was trained with the negative log-likelihood loss.

The second method for training GTNs uses the NLL loss function, with a marginalization over Z using the forward algorithm of Equation 72 over Gr_{sel}, instead of a minimization.

Training by minimizing the NLL loss with stochastic gradient descent is performed by applying the following update rule:

$$W \leftarrow W - \eta \left(\frac{\partial \mathcal{F}_{\mathcal{Z}}(W, Y^i, X^i)}{\partial W} - \frac{\partial \mathcal{F}_{\mathcal{Y},\mathcal{Z}}(W, X^i)}{\partial W} \right), \tag{90}$$

where

$$\mathcal{F}_{\mathcal{Z}}(W, Y^i, X^i) = -\frac{1}{\beta} \log \sum_{z \in \mathcal{Z}} e^{-\beta E(W, Y^i, z, X^i)}, \tag{91}$$

is the free energy obtained by marginalizing over Z, keeping X^i and Y^i fixed, and

$$\mathcal{F}_{\mathcal{Y},\mathcal{Z}}(W, X^i) = -\frac{1}{\beta} \log \sum_{y \in \mathcal{Y},\ z \in \mathcal{Z}} e^{-\beta E(W, y, z, X^i)}, \tag{92}$$

is the free energy obtained by marginalizing over Y and Z, keeping X^i fixed. Computing those gradients is slightly more complicated than in the minimization case. By chain rule the gradients can be expressed as:

$$\frac{\partial \mathcal{F}_{\mathcal{Y},\mathcal{Z}}(W, X^i)}{\partial W} = \sum_{kl} \frac{\partial \mathcal{F}_{\mathcal{Y},\mathcal{Z}}(W, X^i)}{\partial G_{kl}} \frac{\partial G_{kl}(W, X)}{\partial W}, \tag{93}$$

where the sum runs over all edges in the interpretation graph. The first factor is the derivative of the quantity obtained through the forward algorithm (Eq. 72) with respect to one particular edge in the interpretation graph. These quantities can be computed by back-propagating gradients through the trellis, viewed as a feed-forward network with node functions given by Equation 72. We refer to [LeCun et al., 1998a] for details.

Contrary to the claim in [Lafferty et al., 2001], the GTN system trained with the NLL loss as described in [LeCun et al., 1998a] does assign a well-defined probability distribution over possible label sequences. The probability of a particular interpretation is given by Equation 46:

$$P(Y|X) = \frac{\int_{z \in \mathcal{Z}} e^{-\beta E(Z, Y, X)}}{\int_{y \in \mathcal{Y},\ z \in \mathcal{Z}} e^{-\beta E(y, z, X)}}. \tag{94}$$

It would seem natural to train GTNs with one of the generalized margin losses. To our knowledge, this has never been done.

8 Discussion

There are still outstanding questions to be answered about energy-based and probabilistic models. This section offers a relatively philosophical discussion of these questions, including an energy-based discussion of approximate methods for inference and learning. Finally, a summary of the main ideas of this chapter is given.

8.1 EBMs and Probabilistic Models

In Section 1.3, the transformation of energies to probabilities through the Gibbs distribution was introduced:

$$P(Y|X,W) = \frac{e^{-\beta E(W,Y,X)}}{\int_{y\in\mathcal{Y}} e^{-\beta E(W,y,X)}}. \tag{95}$$

Any probability distribution over $\mathcal{Y}$ can be approximated arbitrarily closely by a distribution of that form. With finite energy values, distributions where the probability of some Y is exactly zero can only be approximated. Estimating the parameters of a probabilistic model can be performed in a number of different ways, including maximum likelihood estimation with Bayes inversion, maximum conditional likelihood estimation, and (when possible) Bayesian averaging (possibly with variational approximations). Maximizing the conditional likelihood of the training samples is equivalent to minimizing what we called the negative log-likelihood loss.

Hence, at a high level, discriminative probabilistic models can be seen as a special case of EBMs in which:

- The energy is such that the integral $\int_{y\in\mathcal{Y}} e^{-\beta E(W,y,X)}$ (partition function) converges.
- The model is trained by minimizing the negative log-likelihood loss.

An important question concerns the relative advantages and disadvantages of probabilistic models versus energy-based models. Probabilistic models have two major disadvantages. First, the normalization requirement limits the choice of energy functions we can use. For example, there is no reason to believe that the model in Figure 7 is normalizable over Y. In fact, if the function $G_{W_2}(Y)$ is upper bounded, the integral $\int_{-\infty}^{+\infty} e^{-\beta E(W,y,X)}dy$ does not converge. A common fix is to include an additive term $R_y(Y)$ to the energy, interpreted as a log prior on Y, whose negative exponential is integrable. Second, computing the contrastive term in the negative log-likelihood loss function (or its gradient with respect to W) may be very complicated, expensive, or even intractable. The various types of models can be divided into five rough categories of increasing complexity:

- **Trivial**: When $\mathcal{Y}$ is discrete with a small cardinality, the partition function is a sum with a small number of terms that can be computed simply. Another trivial case is when the partition function does not depend on W, and hence can be ignored for the purpose of learning. For example, this is the case when the energy is a quadratic form in Y with a fixed matrix. These are cases were the energy loss can be used without fear of collapse.
- **Analytical**: When the partition function and its derivative can be computed analytically. For example, when the energy is a quadratic form in Y in which the matrix depends on trainable parameters, the partition function is a Gaussian integral (with variable covariance matrix) and its derivative is an expectation under a Gaussian distribution, both of which have closed-form expressions.

- **Computable**: When the partition function is a sum over an exponential number of terms, but the computation can be factorized in such a way as to make it tractable. The most notable case of this is when the partition function is a sum over configurations of output variables and latent variables of a tree-type graphical model. In this case, belief propagation can be used to compute the partition function. When the graphical model is a simple chain graph (as in the case of HMMs), the set of configurations can be represented by the paths of a weighted trellis. Running the forward algorithm through this trellis yields the partition function. A simple backpropagation-like procedure can be used to compute its gradient (e.g., see [LeCun et al., 1998a] and reference therein).

- **Approachable**: When the partition function cannot be computed exactly, but can be approximated reasonably well using various methods. One notable example is when the partition function is a sum over configurations of a loopy graphical model. The sum cannot be computed exactly, but loopy belief propagation or other variational methods may yield a suitable approximation. With those approximations, the energies of the various answers will still be pulled up, although not as systematically and with the same force as if using the full partition function. In a sense, variational methods could be interpreted in the context of EBM as a way to choose a subset of energies to pull up.

- **Intractable**: When the partition function is truly intractable with no satisfactory variational approximation. In this case, one is reduced to using *sampling methods*. A sampling method is a policy for choosing suitable candidate answers whose energy will be pulled up. The probabilistic approach to this is to sample answers according to their probability under the model, and to pull up their energy. On average, each answer will be pulled up by the appropriate amount according to the partition function.

In this context, using an energy-based loss function other than the negative log-likelihood can be seen as a sampling method with a particular policy for picking the answers whose energy will be pulled up. For example, the hinge loss systematically chooses the most offending incorrect answer as the one whose energy should be pulled up. In the end, using such strategies will produce energy surfaces with which differences of energies cannot be interpreted as likelihood ratios (the same is true with variational methods). We should emphasize again that this is inconsequential if the model is to be used for prediction, classification, or decision-making.

Variational approximation methods can be interpreted in the EBM framework as a particular choice of contrastive term for the loss function. A common approach is to view variational methods and energy-based loss functions as approximations to the probabilistic method. What we propose here is to view the probabilistic approach as a special case of a much larger family of energy-based methods. Energy-based methods are equally well justified as probabilistic methods. They are merely designed for training models to answer a different kind of question than probabilistic models.

An important open question is whether the variational methods that are commonly used (e.g., mean field approximations with popular architectures) actually satisfy condition 3 (see Section 5.2).

8.2 Efficiency in Learning

The most important question that affects the efficiency of learning is: "How many energies of incorrect answers must be explicitly pulled up before the energy surface takes the right shape?". Energy-based loss functions that pull up the most offending incorrect answer only pull up on a single energy at each learning iteration. By contrast, the negative log-likelihood loss pulls up on all incorrect answers at each iteration, including those that are unlikely to produce a lower energy than the correct answer. Hence, unless the NLL computation can be done at very low cost (as in the case of "trivial" and "analytical" models), the energy-based approach is bound to be more efficient.

An important open question is whether alternative loss functions exist whose contrastive term and its derivative are considerably simpler to compute than that of the negative log-likelihood loss, while preserving the nice property that they pull up a large volume of incorrect answers whose energies are "threateningly low". Perhaps, a figure of merit for architectures and loss functions could be defined which would compare the amount of computation required to evaluate the loss and its derivative relative to the volume of incorrect answers whose energy is pulled up as a result.

For models in the "intractable" category, each individual energy that needs to be pulled up or pushed down requires an evaluation of the energy and of its gradient (if a gradient-based optimization method is used). Hence, finding parameterizations of the energy surface that will cause the energy surface to take the right shape with the minimum amount of pushing of pulling is of crucial importance. If $\mathcal{Y}$ is high-dimensional, and the energy surface is infinitely malleable, then the energy surface will have to be pulled up in many places to make it take a suitable shape. Conversely, more "rigid" energy surfaces may take a suitable shape with less pulling, but are less likely to approach the correct shape. There seems to be a bias-variance dilemma similar to the one that influences the generalization performance.

8.3 Learning with Approximate Inference

Very often, the inference algorithm can only give us an approximate answer, or is not guaranteed to give us the global minimum of the energy. Can energy-based learning work in this case? The theory for this does not yet exist, but a few intuitions may shed light on the issue.

There may be certain answers in $\mathcal{Y}$ that our inference algorithm never finds, perhaps because they reside in far-away regions of the space that the algorithm can never reach. Our model may give low energy to wrong answers in these regions, but since the inference algorithm cannot find them, they will never appear in the contrastive term, and their energies will never be pulled up. Fortunately, since those answers are not found by the inference algorithm, we do not need to worry about their energies.

It is an interesting advantage of energy-based learning that only the incorrect answers whose energies are pulled up actually matter. This is in contrast with probabilistic loss functions (e.g. NLL) in which the contrastive term must pull up the energy of every single answer, including the ones that our inference algorithm will never find, which can be wasteful.

8.4 Approximate Contrastive Samples, Contrastive Divergence

Loss functions differ in how the contrastive sample is selected, and how hard its energy is pulled up. One interesting suggestion is to pull up on answers that are always near the correct answer, so as to make the correct answer a local minimum, but not necessarily a global one. This idea is the basis of the *contrastive divergence algorithm* proposed by Hinton [Hinton, 2002, Teh et al., 2003]. Contrastive divergence learning can be seen as an approximation of NLL learning with two shortcuts. First, the contrastive term in Equation 24 is approximated by drawing samples from the distribution $P(Y|X^i, W)$ using a Markov chain Monte Carlo method. Second, the samples are picked by starting the Markov chain at the desired answer, and by running only a few steps of the chain. This produces a sample $\tilde{Y}^i$ that is near the desired answer. Then, a simple gradient update of the parameters is performed:

$$W \leftarrow W - \eta \left(\frac{\partial E(W, Y^i, X^i)}{\partial W} - \frac{\partial E(W, \tilde{Y}^i, X^i)}{\partial W} \right). \tag{96}$$

Since the contrastive sample is always near the desired answer, one can hope that the desired answer will become a local minimum of the energy. Running MCMC for just a few steps limits computational expense. However, there is no guarantee that all incorrect answers with low energy will be pulled up.

8.5 Conclusion

This tutorial was written to introduce and explicate the following major ideas:

- Many existing learning models can be be expressed simply in the framework of energy-based learning.
- Among the many loss functions proposed in the literature, some are good (with a non-zero margin), and some can be bad.
- Probabilistic learning is a special case of energy-based learning where the loss function is the negative log-likelihood, a.k.a. the maximum mutual information criterion.
- Optimizing the loss function with stochastic gradient methods is often more efficient than black box convex optimization methods.
- Stochastic gradient methods can be applied to any loss function including non-convex ones. Local minima are rarely a problem in practice because of the high dimensionality of the space.
- Support vector Markov models, max-margin Markov networks, and conditional random fields are all sequence modeling systems that use linearly parameterized energy factors. Sequence modeling systems with non-linear parameterization for speech and handwriting recognition have been a very active research area since the early 1990's. since the early 90's.

- Graph transformer networks are hierarchical sequence modeling systems in which the objects that are manipulated are trellises containing all the alternative interpretations at a given level. Global training can be performed using stochastic gradient by using a form of back-propagation algorithm to compute the gradients of the loss with respect to all the parameters in the system.

Acknowledgments

The authors wish to thank Geoffrey Hinton, Leon Bottou, Yoshua Bengio, Sebastian Seung, and Brendan Frey for helpful discussions.

This work was supported in part by NSF ITR grant 0325463 "New directions in predictive learning".

References

[Altun and Hofmann, 2003] Altun, Y. and Hofmann, T. (2003). Large margin methods for label sequence learning. In *Proc. of 8th European Conference on Speech Communication and Technology (EuroSpeech).*

[Altun et al., 2003] Altun, Y., Johnson, M., and Hofmann, T. (2003). Loss functions and optimization methods for discriminative learning of label sequences. In *Proc. EMNLP*.

[Bahl et al., 1986] Bahl, L., Brown, P., de Souza, P., and Mercer, R. (1986). Maximum mutual information estimation of hidden markov model parameters for speech recognition. In *Proceedings of Acoustics, Speech, and Signal Processing Conference*, pages 49–52.

[Becker and LeCun, 1989] Becker, S. and LeCun, Y. (1989). Improving the convergence of back-propagation learning with second-order methods. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 29–37, San Mateo. Morgan Kaufman.

[Bengio, 1996] Bengio, Y. (1996). *Neural Networks for Speech and Sequence Recognition*. International Thompson Computer Press, London, UK.

[Bengio et al., 1990] Bengio, Y., Cardin, R., De Mori, R., and Normandin, Y. (1990). A hybrid coder for hidden markov models using a recurrent network. In *Proceeding of ICASSP*, pages 537–540.

[Bengio et al., 1992] Bengio, Y., De Mori, R., Flammia, G., and Kompe, R. (1992). Global optimization of a neural network-hidden Markov model hybrid. *IEEE Transaction on Neural Networks*, 3(2):252–259.

[Bengio et al., 1991] Bengio, Y., DeMori, R., Flammia, G., and Kompe, R. (1991). Global optimization of a neural network - hidden markov model hybrid. In *Proceedings of EuroSpeech'91*.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

[Bengio and Frasconi, 1996] Bengio, Y. and Frasconi, P. (1996). An input/output HMM architecture. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems*, volume 7, pages 427–434. MIT Press, Cambridge, MA.

[Bengio et al., 1993] Bengio, Y., LeCun, Y., and Henderson, D. (1993). Globally trained handwritten word recognizer using spatial representation, space displacement neural networks and hidden markov models. In Cowan, J. and Tesauro, G., editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann.

[Bengio et al., 1995] Bengio, Y., LeCun, Y., Nohl, C., and Burges, C. (1995). Lerec: A nn/hmm hybrid for on-line handwriting recognition. *Neural Computation*, 7(6):1289–1303.

[Bottou, 1991] Bottou, L. (1991). *Une Approche théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole*. PhD thesis, Université de Paris XI, 91405 Orsay cedex, France.

[Bottou, 2004] Bottou, L. (2004). Stochastic learning. In Bousquet, O. and von Luxburg, U., editors, *Advanced Lectures on Machine Learning*, number LNAI 3176 in Lecture Notes in Artificial Intelligence, pages 146–168. Springer Verlag, Berlin.

[Bottou and LeCun, 2004] Bottou, L. and LeCun, Y. (2004). Large-scale on-line learning. In *Advances in Neural Information Processing Systems 15*. MIT Press.

[Bottou et al., 1997] Bottou, L., LeCun, Y., and Bengio, Y. (1997). Global training of document processing systems using graph transformer networks. In *Proc. of Computer Vision and Pattern Recognition*, pages 490–494, Puerto-Rico. IEEE.

[Bourlard and Morgan, 1990] Bourlard, H. and Morgan, N. (1990). A continuous speech recognition system embedding mlp into hmm. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2*, pages 186–193. Morgan Kaufmann.

[Bromley et al., 1993] Bromley, J., Guyon, I., LeCun, Y., Sackinger, E., and Shah, R. (1993). Signature verification using a siamese time delay neural network. In Cowan, J. and Tesauro, G., editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann.

[Chopra et al., 2005] Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *Proc. of Computer Vision and Pattern Recognition Conference*. IEEE Press.

[Collins, 2000] Collins, M. (2000). Discriminative reranking for natural language parsing. In *Proceedings of ICML 2000*.

[Collins, 2002] Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. EMNLP*.

[Collobert et al., 2006] Collobert, R., Weston, J., and Bottou, L. (2006). Trading convexity for scalability. In *Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006)*. IMLS/ICML. ACM Digital Library.

[Denker and Burges, 1995] Denker, J. S. and Burges, C. J. (1995). Image segmentation and recognition. In *The Mathematics of Induction*. Addison Wesley.

[Driancourt, 1994] Driancourt, X. (1994). *Optimisation par descente de gradient stochastique de systèmes modulaires combinant réseaux de neurones et programmation dynamique. Application à la reconnaissance de la parole. (optimization through stochastic gradient of modular systems that combine neural networks and dynamic programming, with applications to speech recognition)*. PhD thesis, Université de Paris XI, 91405 Orsay cedex, France.

[Driancourt et al., 1991a] Driancourt, X., Bottou, L., and Gallinari, P. (1991a). MLP, LVQ and DP: Comparison & cooperation. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 815–819, Seattle.

[Driancourt et al., 1991b] Driancourt, X., Bottou, L., and P., G. (1991b). Comparison and cooperation of several classifiers. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*.

[Driancourt and Gallinari, 1992a] Driancourt, X. and Gallinari, P. (1992a). Empirical risk optimisation: neural networks and dynamic programming. In *Proceedings of Neural Networks for Signal Processing (NNSP)*.

[Driancourt and Gallinari, 1992b] Driancourt, X. and Gallinari, P. (1992b). A speech recognizer optimaly combining learning vector quantization, dynamic programming and multi-layer perceptron. In *Proceedings of ICASSP*.

[Franzini et al., 1990] Franzini, M., Lee, K. F., and Waibel, A. (1990). Connectionnist viterbi training: A new hybrid method for continuous speech recognition. In *Proceedings of ICASSP*, page 245.

[Hadsell et al., 2006] Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'06)*. IEEE Press.

[Haffner, 1993] Haffner, P. (1993). Connectionist speech recognition with a global MMI algorithm. In *EUROSPEECH'93, 3rd European Conference on Speech Communication and Technology*, Berlin.

[Haffner et al., 1991] Haffner, P., Franzini, M., and Waibel, A. H. (1991). Integrating time-alignment and neural networks for high performance continuous speech recognition. In *Proceeding of ICASSP*, pages 105–108. IEEE.

[Haffner and Waibel, 1991] Haffner, P. and Waibel, A. H. (1991). Time-delay neural networks embedding time alignment: a performance analysis. In *EUROSPEECH'91, 2nd European Conference on Speech Communication and Technology*, Genova, Italy.

[Haffner and Waibel, 1992] Haffner, P. and Waibel, A. H. (1992). Multi-state time-delay neural networks for continuous speech recognition. In *Advances in Neural Information Processing Systems*, volume 4, pages 579–588. Morgan Kaufmann, San Mateo.

[Hinton, 2002] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800.

[Huang and LeCun, 2006] Huang, F.-J. and LeCun, Y. (2006). Large-scale learning with svm and convolutional nets for generic object categorization. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'06)*. IEEE Press.

[Juang et al., 1997] Juang, B.-H., Chou, W., and Lee, C.-H. (1997). Minimum classification error rate methods for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 5(3):257–265.

[Konig et al., 1996] Konig, Y., Bourlard, H., and Morgan, N. (1996). REMAP: Recursive estimation and maximization of A posteriori probabilities — application to transition-based connectionist speech recognition. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 388–394. The MIT Press.

[Kschischang et al., 2001] Kschischang, F., Frey, B., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Trans. Information Theory*, 47(2):498–519.

[Kumar and Hebert, 2004] Kumar, S. and Hebert, M. (2004). Discriminative fields for modeling spatial dependencies in natural images. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA.

[Lafferty et al., 2001] Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. International Conference on Machine Learning (ICML)*.

[LeCun and Bengio, 1994] LeCun, Y. and Bengio, Y. (1994). word-level training of a handwritten word recognizer based on convolutional neural networks. In IAPR, editor, *Proc. of the International Conference on Pattern Recognition*, volume II, pages 88–92, Jerusalem. IEEE.

[LeCun et al., 1997] LeCun, Y., Bottou, L., and Bengio, Y. (1997). Reading checks with graph transformer networks. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 151–154, Munich. IEEE.

[LeCun et al., 1998a] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[LeCun et al., 1998b] LeCun, Y., Bottou, L., Orr, G., and Muller, K. (1998b). Efficient backprop. In Orr, G. and K., M., editors, *Neural Networks: Tricks of the trade*. Springer.

[LeCun and Huang, 2005] LeCun, Y. and Huang, F. (2005). Loss functions for discriminative training of energy-based models. In *Proc. of the 10-th International Workshop on Artificial Intelligence and Statistics (AIStats'05)*.

[Ljolje et al., 1990] Ljolje, A., Ephraim, Y., and Rabiner, L. R. (1990). Estimation of hidden markov model parameters by minimizing empirical error rate. In *Proc. of International Conference on Acoustics, Speech, and Signal Processing*, pages 709–712.

[MacKay, 2003] MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press. Available from `http://www.inference.phy.cam.ac.uk/mackay/itila/`.

[McCallum et al., 2000] McCallum, A., Freitag, D., and Pereira, F. (2000). Maximum entropy markov models for information extraction and segmetnation. In *Proc. International Conference on Machine Learning (ICML)*, pages 591–598.

[McDermott, 1997] McDermott, E. (1997). *Discriminative Training for Speech Recognition*. PhD thesis, Waseda University.

[McDermott and Katagiri, 1992] McDermott, E. and Katagiri, S. (1992). Prototype-based discriminative training for various speech units. In *Proceedings of ICASSP-92, San Francisco, CA, USA*, pages 417–420.

[Mohri, 1997] Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

[Morgan and Bourlard, 1995] Morgan, N. and Bourlard, H. (1995). Continuous speech recognition: An introduction to the hybrid hmm/connectionist approach. *IEEE Signal Processing Magazine*, 12(3):25–42.

[Ning et al., 2005] Ning, F., Delhomme, D., LeCun, Y., Piano, F., Bottou, L., and Barbano, P. (2005). Toward automatic phenotyping of developing embryos from videos. *IEEE Transactions on Image Processing*, 14(9):1360–1371. Special issue on Molecular and Cellular Bioimaging, to appear.

[Osadchy et al., 2005] Osadchy, R., Miller, M., and LeCun, Y. (2005). Synergistic face detection and pose estimation with energy-based model. In *Advances in Neural Information Processing Systems (NIPS 2004)*. MIT Press.

[Sakoe et al., 1988] Sakoe, H., Isotani, R., Yoshida, K., Iso, K., and Watanabe, T. (1988). Speaker-independant word recognition using dynamic programming neural networks. In *Proceedings of ICASSP-88, New York*, pages 107–110.

[Solla et al., 1988] Solla, S., Levin, E., and Fleisher, M. (1988). Accelerated learning in layered neural networks. *Complex Systems*, 2(6):625–639.

[Taskar et al., 2003] Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin markov networks. In *Proc. NIPS*.

[Teh et al., 2003] Teh, Y. W., Welling, M., Osindero, S., and E., H. G. (2003). Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4:1235–1260.

[Vapnik, 1995] Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer Verlag.

[Vishwanathan et al., 2006] Vishwanathan, S. V. N., Schraudolph, N. N., Schmidt, M. W., and Murphy, K. P. (2006). Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006)*. IMLS/ICML.

[Yedidia et al., 2005] Yedidia, J., Freeman, W., and Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312.