Object Detection Grammars

Pedro F. Felzenszwalb and David McAllester

February 11, 2010

1 Introduction

We formulate a general grammar model motivated by the problem of object detection in computer vision. We focus on four aspects of modeling objects for the purpose of object detection.

First, we are interested in modeling objects as having parts which are themselves (recursively) objects. For example a person can be represented as being composed of a face, a trunk, arms, and legs where a face is composed of eyes, a nose and a mouth.

Second, we are interested modeling object (and part) categories as being composed of subcategories or subtypes. For example we might distinguish sitting people from standing people and smiling faces from frowning faces.

Third, we are interested in modeling the relative positions of the parts that make up an object. For example, in a person, the position of the hand is related to the position of the lower arm which is related to the position of the upper arm which is related to the position of the torso.

Fourth, we are interested in modeling the appearance of objects so that we can find them in images. For example, a pattern of edges in a particular location of an image might give evidence for, or against, the presence of a part at that location.

These four aspects of models — parts, subtypes, positions, and appearance — can be represented in a single grammar formalism that we call an object detection grammar.

2 The Formal Model

We formally define object detection grammars in three stages roughly corresponding to parts and subtypes (stage one), positions (stage two), and appearance (stage three). After introducing the

general model we define restrictions that simplify inference and learning.

2.1 Modeling Parts and Subtypes

We first focus on representing parts and subtypes. Each object (part) is represented by a nonterminal symbol in a grammar formalism. A description of an object in terms of parts and subtypes is represented by grammar productions.

We define models using the notion of a *bag grammar*. A bag grammar is similar to a context free grammar except that it generates bags (multisets) of terminals instead of strings. Intuitively this reflects the fact that there is no sequential order among the parts that make up an object.

We define a (scored) bag grammar to consist of a set of nonterminal symbols N, a set of terminal symbols T and a set of scored productions R of the form

$$X \xrightarrow{\alpha} \{Y_1, \ldots, Y_n\},$$

with $X \in N$, $Y_i \in N \cup T$, $\alpha \in \mathbb{R}$. The right hand side of each production is a bag (multiset) of elements of $N \cup T$. Here α is the score of the production.

A bag grammar defines a nondeterministic process for replacing nonterminals by bags of terminals and nonterminals. Each production from X represents a possible choice when replacing X, where the score of the production represents its "desirability". This process can be continued until only terminal symbols remain. Hence a bag grammar defines a nondeterministic process for deriving a bag of terminals from a nonterminal.

As in the case of a context free grammar we can define derivation trees. However, while with a context free grammar the children of each node in a derivation tree are ordered from left to right, in the case of a bag grammar the children are unordered. The leafs of a derivation tree represent the bag of terminals generated by the derivation. The score of a derivation using a bag grammar is the sum of the scores of the productions used in the derivation (counting multiplicities).

More formally we define the *derives* relation $\rightsquigarrow$ to be the least relation such that for each terminal symbol $A \in T$

$$A \overset{0}{\rightsquigarrow} \{A\}$$

and for each production $X \xrightarrow{\alpha} \{Y_1, \ldots, Y_n\} \in R$

$$\text{if } Y_i \overset{\alpha_i}{\rightsquigarrow} B_i \text{ then } X \overset{\beta}{\rightsquigarrow} \bigcup_i B_i \text{ with } \beta = \alpha + \sum_{i=1}^{n} \alpha_i.$$

If $X \overset{\alpha}{\rightsquigarrow} \{A_1, \ldots, A_k\}$ we say X derives $\{A_1, \ldots, A_k\}$ with score α. Here X is an arbitrary symbol and $\{A_1, \ldots, A_k\}$ is a bag of terminals.

In an object detection model the nonterminals of a bag grammar represent objects and the terminals represent appearance models as described in Section 2.3. A decomposition of an object X into parts $Y_1, \ldots, Y_n$ is represented by a production $X \overset{\alpha}{\rightarrow} \{Y_1, \ldots, Y_n\}$. A decomposition of an object X into subtypes $X_1, \ldots, X_m$ is represented by productions $X \overset{\alpha_i}{\rightarrow} \{X_i\}$ for $1 \leq i \leq m$. Here α_i will reflect to the prevalence of the i-th subtype.

A grammar will be called cyclic (or recursive) if a nonterminal X can generate itself. Otherwise the grammar is called acyclic.

For acyclic grammars a maximum scoring derivation from each nonterminal can be computed by dynamic programming. Section 2.3 describes a grammar for which the maximum scoring derivation problem is analogous to the problem of "matching" a deformable part-based model to an image. Section 3 describes the dynamic programming algorithm for the non-recursive case in more detail.

We can handle cyclic grammars if we work with non-negative weights rather than scores and seek a minimum weight derivation. In that case a generalization of Dijkstra's shortest paths algorithm can be used [9, 5]. Alternatively, an A* algorithm [5] can be used to maximize score if one can give a monotone heuristic function. Here however, we focus on the acyclic grammars in which simple dynamic programming can be used.

Note that the maximum scoring derivation problem does not specify a fixed bag of terminals to be derived. Each terminal symbol may be used an arbitrary number of times. Another related problem involves specifying a bag of terminals and ask for the maximum scoring derivation of exactly that bag. This bag parsing problem is analogous to string parsing with context-free grammars. One can give an exponential time dynamic programming algorithm based on computing a maximum scoring derivation of each sub-bag of the input bag from each nonterminal. This is similar to the CKY algorithm for parsing strings, but bags differ from strings in that there are exponentially many sub-bags of a bag. It can be shown that bag parsing is NP-hard even for acyclic grammars (there is a simple reduction from 3D matching). Bag parsing is equivalent to string parsing under arbitrary permutations of the input. This problem was considered for machine translation in the past, and shown to be NP-hard in [1].

2.2 Modeling Placement

We now consider a set Ω of *positions* for the symbols in a bag grammar. For example, in object detection we often construct an image pyramid $(I_1, \ldots, I_L)$, where I_1 is the input image and $I_{\ell+1}$ is a lower resolution version of I_ℓ. In this case we can take the elements of Ω to be triples (ℓ, x, y) where ℓ specifies a level in the pyramid and x and y are coordinates within I_ℓ. In this case ℓ specifies the scale or size for a symbol while x and y specify a location. We could also take the elements of Ω to be four-tuples (ℓ, x, y, θ) where θ defines an orientation for a symbol. In general Ω is an arbitrary set defining instantiation parameters for the symbols of a grammar.

A *placed symbol*, written as $Y(\omega)$, is a pair of a symbol $Y \in N \cup T$ and a position $\omega \in \Omega$.

A *placed production* is a production of the form

$$X(\omega_0) \stackrel{\alpha}{\rightarrow} \{Y_1(\omega_1), \ldots, Y_n(\omega_n)\}.$$

A *placed production schema* is a parameterized placed production of the following form.

$$\forall z \in \mathcal{D} : X(\omega_0(z)) \stackrel{\alpha(z)}{\rightarrow} \{Y_1(\omega_1(z)), \ldots, Y_n(\omega_n(z))\} \tag{1}$$

This schema represents a (possibly infinite) set of placed productions. Here $\mathcal{D}$ is a set parameterizing the placed productions defined by the schema. Each value $z \in \mathcal{D}$ defines a placed production, $\omega_i(z)$ are functions specifying positions in Ω as a function of z, and $\alpha(z)$ is a function specifying the score of the placed production defined by z.

For example we might write a schema for faces as follows

$$\forall(\omega_0, \omega_1, \omega_2, \omega_3) \in \Omega^4 : \text{FACE}(\omega_0) \stackrel{\alpha(\omega_0,\omega_1,\omega_2,\omega_3)}{\rightarrow} \{\text{LEFT.EYE}(\omega_1), \text{RIGHT.EYE}(\omega_2), \text{MOUTH}(\omega_3)\}.$$

Here $\alpha(\omega_0, \omega_1, \omega_2, \omega_3)$ is a function assigning a score for each placement of the parts that make up a face at a particular position. This score might be high if the eyes and mouth are where they should be in a typical face, and low otherwise.

A set of placed production schemas defines a bag grammar whose terminals and nonterminals are placed symbols. The definition of the relation $\rightsquigarrow$ given in Section 2.1 then applies to define expansions of placed symbols into bags of the following form where each A_i is a terminal symbol

$$X(\omega_0) \stackrel{\alpha}{\rightsquigarrow} \{A_1(\omega_1), \ldots, A_k(\omega_k)\}.$$

The right hand side of this terminal expansion will be called a *placed terminal bag.*

2.3 Modeling Appearance

Let $\mathcal{I}$ be an *input space* (such as the space of images). We now assume that each terminal symbol A has an associated *appearance evaluation function* $f_A : \Omega \times \mathcal{I} \to \mathbb{R}$. Now $f_A(\omega, I)$ specifies a score for placing A in position ω within I.

In practice we can use filters to define appearance evaluation functions. For example, suppose we have an image pyramid $(I_1, \ldots, I_L)$. Each image I_ℓ can be divided into $k \times k$ cells and a d-dimensional feature vector (such as a HOG descriptor [2]) can be computed in each cell. The result is an array H_ℓ whose entries are d-dimensional feature vectors. This leads to a feature map pyramid $(H_1, \ldots, H_L)$. Now suppose the elements of Ω are triples (ℓ, x, y) specifying positions in the feature map pyramid. We can associate each terminal symbol A with a template F_A, defined by a $w \times h$ array of d-dimensional weight vectors. Now we can define the appearance score of $A(\omega)$ to be the response of F_A at position ω in the feature map pyramid

$$f_A((\ell, x, y), I) = \sum_{i=1}^{w} \sum_{j=1}^{h} F_A[i, j] \cdot H_\ell[x + i, y + j].$$

In general we take an *object detection grammar* over an input space $\mathcal{I}$ to consist of a set of nonterminal symbols N, a set of terminal symbols T, a set of positions Ω, a set of placed production schemas S, and an appearance evaluation function $f_A : \Omega \times \mathcal{I} \to \mathbb{R}$ for each terminal $A \in T$.

For a grammar model G over the input space $\mathcal{I}$ we extend the appearance evaluation functions for the terminals to all symbols. This leads to an evaluation function $f_G : (N \cup T) \times \Omega \times \mathcal{I} \to \mathbb{R}$. Now $f_G(Y, \omega, I)$ specifies a score for placing symbol Y in position ω within I. This score is the maximum over all expansions of $Y(\omega)$ into a placed terminal bag, of the score of the expansion plus the score of placing the terminals in their respective positions

$$f_G(Y, \omega, I) = \max_{Y(\omega) \overset{\alpha}{\rightsquigarrow} \{A_1(\omega_1), \ldots, A_k(\omega_k)\}} \alpha + \sum_{i=1}^{k} f_{A_i}(\omega_i, I).$$

Under these definitions we can think of the terminals of a grammar as the basic building blocks that can be found in an image. The nonterminals define abstract objects whose appearance are defined in terms of expansions into terminals.

2.4 Object Detection

Suppose we want to detect instances of an object Y. Given an image I we can search for values ω where $f_G(Y, \omega, I)$ is above some threshold. This is an abstract formulation of the sliding window approach. A placement ω is analogous to a placement of a window. We can return high scoring placements ω as the "detections" within I. Of course in practice one should do non-maximum suppression to avoid multiple detections of the same object.

For example, suppose G is a grammar for faces with a nonterminal FACE that expands to terminals corresponding to eyes and mouth. Then $f_G(\text{FACE}, \omega, I)$ will define a score for placing a face in position ω that takes into account the placement of the parts in some ideal location with respect to the location of the face. If $\text{FACE}(\omega)$ expands to subtypes $\text{SMILING}(\omega)$ and $\text{FROWNING}(\omega)$ (where the score for the expansion is zero) then $f_G(\text{FACE}, \omega, I)$ will be the larger of $f_G(\text{SMILING}, \omega, I)$ and $f_G(\text{FROWNING}, \omega, I)$.

2.5 Isolated Deformation Grammars

We now consider a restriction on object detection grammars which limits there expressive power but which simplifies computing scores $f_G(Y, \omega, x)$ and optimal derivations.

Let Δ be a set of *displacements* for positions in Ω and $\oplus$ be an operation $\oplus : \Omega \times \Delta \to \Omega$. For example, if the positions in Ω are triples (ℓ, x, y) we might have Δ be tuples (dx, dy) and $(\ell, x, y) \oplus (dx, dy) = (\ell, x + dx, y + dy)$. We use δ to specify an element of Δ.

An *isolated deformation grammar* is defined to be one where every placed production schema has one of the following two forms

$$\forall \omega : X(\omega) \stackrel{\alpha}{\to} \{Y_1(a_1(\omega)), \ldots, Y_n(a_n(\omega))\},$$
$$\forall \omega, \delta : X(\omega) \stackrel{\alpha(\delta)}{\to} \{Y(\omega \oplus \delta)\}.$$

We will call production schemas of the first form *structural rules* and production schemas of the second form *deformation rules*. Structural rules can express both decompositions of an object into parts and of an object type into subtypes. The functions $a_i : \Omega \to \Omega$ in a structural rule specify "anchor" positions for the symbols Y_i in terms of a position for X. The function $\alpha : \Delta \to \mathbb{R}$ in a deformation rule specifies a score for every possible displacement of Y relative to X. Note that to simplify notation we omit the domain $\mathcal{D} = \Omega$ of the parameter ω in a structural rule and the

domain $\mathcal{D} = \Omega \times \Delta$ of the parameter (ω, δ) in a deformation rule.

An isolated deformation grammar defines a "factored" model such that the position of a symbol in a derivation tree is only related to the position of its parent.

For example, we can define an isolated deformation grammar for faces with

$$N = \{\text{FACE}, \text{EYE}, \text{EYE}', \text{MOUTH}, \text{MOUTH}'\},$$
$$T = \{\text{FACE.FILTER}, \text{EYE.FILTER}, \text{SMILE.FILTER}, \text{FROWN.FILTER}\}.$$

We have a structural rule for representing a face in terms of a global template and parts

$$\forall \omega : \text{FACE}(\omega) \xrightarrow{0} \{\text{FACE.FILTER}(\omega), \text{EYE}'(\omega \oplus \delta_l), \text{EYE}'(\omega \oplus \delta_r), \text{MOUTH}'(\omega \oplus \delta_m)\}.$$

Here $\delta_l, \delta_r, \delta_m$ are constants that specify the ideal displacement between each part and the face. Note that EYE$'$ appears twice in the right hand side under different ideal displacements, to account for the two eyes in a face. We can move the parts that make up a face from their ideal locations using deformation rules

$$\forall \omega, \delta : \text{EYE}'(\omega) \xrightarrow{||\delta||^2} \{\text{EYE}(\omega \oplus \delta)\},$$
$$\forall \omega, \delta : \text{MOUTH}'(\omega) \xrightarrow{||\delta||^2} \{\text{MOUTH}(\omega \oplus \delta)\}.$$

Finally we associate templates with the part nonterminals using structural rules

$$\forall \omega : \text{EYE}(\omega) \xrightarrow{0} \{\text{EYE.FILTER}(\omega)\},$$
$$\forall \omega : \text{MOUTH}(\omega) \xrightarrow{s} \{\text{SMILE.FILTER}(\omega)\},$$
$$\forall \omega : \text{MOUTH}(\omega) \xrightarrow{f} \{\text{FROWN.FILTER}(\omega)\}.$$

The last two rules specify two different templates that can be used for the mouth at different scores that reflect the prevalence of smiling and frowning faces.

2.6 Labeled Derivation Trees

Let G be an object detection grammar. We define a *labeled derivation tree* to be a rooted tree such that: (1) each leaf v has an associated placed terminal $A(\omega)$; (2) each internal node v has an associated placed nonterminal $X(\omega)$, a placed production schema, and a value z for the schema leading to a placed production with $X(\omega)$ in the left hand side and the placed symbols associated

with the children of v in the right hand side. If Z is a labeled derivation tree and $Y(\omega)$ is associated with the root of the tree we say Z is rooted at $Y(\omega)$.

The evaluation function f_G can also be defined in terms of labeled derivation trees.

Let Z be a labeled derivation tree and I be an image. We define $s(Z, I)$ to be the *score of a labeled derivation tree within an image*. If Z is rooted at a placed terminal $A(\omega)$ we define

$$s(Z, I) = f_A(\omega, I). \tag{2}$$

Otherwise, if v is the root of the tree and $Z_1, \ldots, Z_n$ are the subtrees below v we define

$$s(Z, I) = \alpha(z) + \sum_{i=1}^{n} s(Z_i, I). \tag{3}$$

Here α is the score function of the production schema associated with v and z is the parameter for the schema associated with v.

Let $\mathcal{Z}_{Y(\omega)}$ be the set of labeled derivation trees rooted at $Y(\omega)$. Now we have

$$f_G(Y, \omega, I) = \max_{Z \in \mathcal{Z}_{Y(\omega)}} s(Z, I). \tag{4}$$

We also define a maximum score labeled derivation tree rooted at $Y(\omega)$ as

$$Z^*_G(Y, \omega, I) = \operatorname*{argmax}_{Z \in \mathcal{Z}_{Y(\omega)}} s(Z, I). \tag{5}$$

3 Inference

For inference we are interested in computing maximum scoring derivations from given nonterminals placed at different positions within an image. This problem is defined more concretely as computing $f_G(Y, \omega, I)$ and $Z^*_G(Y, \omega, I)$ as defined by (4) and (5). Here we consider only the case of acyclic (non-recursive) isolated deformation grammars in which these quantities can be computed by efficient dynamic programming. We consider computing f_G, the computation of Z^*_G is analogous.

The dynamic programming algorithm computes $f_G(Y, \omega, I)$ for every symbol Y and position ω. We will store $f_G(Y, \omega, I)$ in an array $F[Y, \omega]$.

Since the grammar is acyclic we can order the nonterminals symbols such that for every production schema with X in the left hand side, the nonterminals in the right hand side come after X in the order. Let $(X_1, \ldots, X_{|N|})$ be such an order. Note that this is exactly a topological ordering

of a graph where the nodes correspond to nonterminal symbols, and there is a directed edge from X to Y if there is a rule with X in the left hand side and Y in the right hand side.

For a terminal symbol $F[A, \omega] = f_A(\omega, I)$. The algorithm starts by computing these values for every terminal. The runtime of this step depends on the complexity of the appearance models, but it is typically $O(|T||\Omega|)$.

For the nonterminals we compute $F[X_i, \omega]$ in order of decreasing i. Let $R_1, \ldots, R_k$ be the set of rules with X_i in the left hand side. We compute k arrays $R_j[\omega]$ for $1 \leq j \leq k$ corresponding to the score of a maximum derivation rooted at $X_i(\omega)$ using R_j to expand $X_i(\omega)$. Then we have $F[X_i, \omega] = \max_j R_j[\omega]$.

If R_j is a structure rule $\forall \omega : X(\omega) \stackrel{\alpha}{\rightarrow} \{Y_1(a_1(\omega)), \ldots, Y_n(a_n(\omega))\}$ then

$$R_j[\omega] = \alpha + F[Y_1, a_1(\omega)] + \cdots + F[Y_n, a_n(\omega)].$$

The values in this array can be computed in $O(|\Omega|)$ time if n is bounded by a constant.

If R_j is a deformation rule $\forall \omega, \delta : X(\omega) \stackrel{\alpha(\delta)}{\rightarrow} \{Y(\omega \oplus \delta)\}$ then

$$R_j[\omega] = \max_{\delta}(\alpha(\delta) + F[Y, \omega \oplus \delta]).$$

The values in this array can be computed in $O(|\Omega||\Delta|)$ time by brute force search.

The dynamic programming method will compute all of F in $O(k_1|\Omega||\Delta| + k_2|\Omega|)$ time after we evaluate the appearance models at each position. Here k_1 is the total number of deformation rules in the grammar and k_2 is the total number of structure rules. The method can also be used to compute optimal labeled derivation trees rooted at a placed symbol. We note that in some important cases the dependency on $|\Delta|$ can be removed by using fast generalized distance transform algorithms [3]. In those cases the total runtime is $O(k|\Omega|)$ where k is the number of production schemas in the grammar.

4 LSVM Learning

Now we consider the problem of learning parameters of an object detection grammar. We first describe the notion of a linear grammar and then describe how to apply LSVM training [6] to learn these models from weakly labeled data.

4.1 Linear Object Detection Grammars

We say that an object detection grammar is *linear* if there exists a parameter vector $\theta \in \mathbb{R}^d$ such that the score function associated with each production schema r has the form

$$\alpha_r(z) = \theta \cdot \phi_r(z),$$

and the appearance model associated with each terminal A has the form

$$f_A(\omega, I) = \theta \cdot \phi_A(\omega, I).$$

Here $\phi_r(z)$ and $\phi_A(\omega, I)$ are functions associated with the schema r and terminal A respectively. These functions return feature vectors in $\mathbb{R}^d$.

In practice the vector θ might be divided into blocks, with one block per schema r and one block per terminal symbol A. In this case the feature vectors $\phi_r(z)$ and $\phi_A(\omega, I)$ will be sparse, $\phi_r(z)$ has zeros outside the block for r and $\phi_A(\omega, I)$ has zeros outside the block for A. For example, suppose we have an isolated deformation grammar where the positions in Ω are triples (ℓ, x, y) specifying a position within a feature map pyramid and the displacements in Δ are tuples (dx, dy) specifying a shift within a level of the pyramid. In a deformation rule we have $z = (\omega, \delta)$ and we can define $\phi_r(z)$ to be (dx^2, dy^2, dx, dy) in the block of parameters for r, and zeros everywhere else. Then $\theta \cdot \phi_r(z)$ is a quadratic function of the displacement specified by z, with coefficients defined by a block of entries in θ. For the appearance models suppose $(H_1, \ldots, H_L)$ is a feature map pyramid computed from I. We can define $\phi_A((\ell, x, y), I)$ to be a subarray of H_ℓ with the top-left corner at (x, y) in the block for A, and zeros everywhere else. Then $\theta \cdot \phi_A(\omega, I)$ is the response of a linear filter defined by a block of entries in θ at position (ℓ, x, y) of the feature map pyramid.

Now suppose we have a linear grammar parameterized by θ. Given a labeled derivation tree Z and an input I we can define a cumulative feature vector $\Phi(Z, I)$ as follows

$$\Phi(Z, I) = \sum_{v \in \text{int}(Z)} \phi_{r(v)}(z(v)) + \sum_{v \in \text{leaf}(Z)} \phi_{A(v)}(\omega(v), I). \tag{6}$$

The first sum is over internal nodes of the derivation tree and the second sum is over leaf nodes. Recall that internal nodes of Z are labeled by a production schema $r(v)$ and value $z(v)$ for the schema, while leafs are labeled by placed terminals $A(v)(\omega(v))$.

With this definition we have

$$s(Z, I) = \theta \cdot \Phi(Z, I).$$

That is, the score of Z in I is a linear function of θ. In particular, if we had "positive examples" (Z, I) for which we want $s(Z, I)$ to be high (above some threshold), and "negative examples" (Z, I) for which we want $s(Z, I)$ to be low (below the threshold) we could train θ using a linear SVM formulation. Each example (Z, I) would lead to a feature vector $\Phi(Z, I)$ for SVM training.

In practice we would like to learn the parameters of a model from weakly labeled examples. In this case examples may specify the locations of objects but not their full derivation.

Note that for a linear grammar we can rewrite (4) as

$$f_G(Y, \omega, I) = \max_{Z \in \mathcal{Z}_{Y(\omega)}} \theta \cdot \Phi(Z, I). \tag{7}$$

Because a maximum of linear functions is convex, we have that $f_G(Y, \omega, I)$ is a convex function of the parameter vector θ.

4.2 LSVM Training

LSVMs (Latent Support Vector Machines) [6] are a generalization of SVMs to incorporate latent information. Like an SVM, a latent SVM is trained as a classifier. Formally we assume training data of the form $(x_1, y_1), \ldots, (x_n, y_n)$ where x_i is an input from some abstract input space and $y_i \in \{-1, +1\}$. Let $L(x)$ denote a set of possible latent values for x. For $z \in L(X)$ let $\Phi(x, z)$ be a feature vector associated with the pair (x, z). The latent SVM framework trains binary classifiers that threshold a score of the form

$$f_\theta(x) = \max_{z \in L(x)} \theta \cdot \Phi(x, z). \tag{8}$$

Suppose we have training data $(x_1, y_1), \ldots, (x_n, y_n)$. In analogy to standard SVM training, LSVM training is defined by the following optimization problem

$$\theta^* = \operatorname*{argmin}_\theta \sum_{i=1}^{n} \max(0, 1 - y_i f_\theta(x_i)) + \lambda ||\theta||^2. \tag{9}$$

The hinge loss pushes $f_\theta(x)$ to be above $+1$ for a positive example, and below -1 for a negative one. This optimization problem is not convex but we can use the methods in [6] to find a good model. An alternative involves the Concave-Convex produre as used in [10].

As discussed in Section 2.4 we can reduce object detection to classification using an abstract formulation of the sliding window approach. To detect instances of an object Y in an image I we

look for placements ω such that $f_G(Y, \omega, I)$ is above some threshold. Now we reduce the problem of learning the parameters θ of a linear grammar from weakly labeled data to LSVM training.

Let $x = (Y, \omega, I)$. For visual object detection we have that I is an image, Y is an object such as "person", and ω is a specification of the position of the object (it may include scale and other pose information). For a linear grammar equation (7) defines a score function $f_\theta(x) = f_G(Y, \omega, I)$ of the form in (8). Now a latent value $z \in L(x)$ is a derivation rooted at $Y(\omega)$ and $\Phi(x, z)$ is the cumulative feature vector (6) associated with the derivation.

A training example of the form $((Y, \omega, I), +1)$ states that there is an instance of Y at position ω in the image I. Note that the example is weakly labeled because it does not specify a derivation. A training example of the form $((Y, \omega, I), -1)$ states that there is not an instance of Y at position ω in the image I. When training an object detector negative instances can often be represented implicitly by a set of pairs of the form $((Y, I), -1)$. Each such pair indicates that there is no instance of Y in the image I and is treated as a compact representation of the set of all negative training instances of the form $((Y, \omega, I), -1)$. In practice "data mining" is used to convert negative training instances of the form $((Y, I_1), -1)$, $\ldots$, $((Y, I_N), -1)$ into a set of "hard negatives" of the form $((Y, \omega_i, I_j), -1)$ [6].

5 Neural Grammars

For a linear acyclic grammar, the function $f_G(Y, \omega, I)$ can be defined using the following recursive equations (these can be used to compute $f_G(Y, \omega, I)$ by dynamic programming)

$$f_G(X, \omega, I) = \max_{X(\omega) \xrightarrow{\phi} Y_1(\omega_1), \ldots, Y_n(\omega_n)} \theta \cdot \phi + \sum_{i=1}^{n} f_G(Y_i, \omega_i, I) \tag{10}$$

$$f_G(A, \omega, I) = \theta \cdot \phi_A(\omega, I) \tag{11}$$

In (10) the max is taken over productions from $X(\omega)$ rather than full derivations. Recall that each production from $X(\omega)$ corresponds to an instance of a production schema r defined by a parameter z. Above we have $\phi = \phi_r(z)$.

We now consider a method of computing scores in an acyclic grammar involving a nonlinear sigmoidal function as in neural networks. To construct a neural grammar we define $f_G(Y, \omega, I)$ with

the following equations where σ is a sigmoid function

$$f_G(X,\omega,I) = \max_{X(\omega) \overset{\phi,a}{\rightarrow} Y_1(\omega_1),\ldots,Y_n(\omega_n)} \sigma\left(\theta \cdot \phi + a_0 + \sum_{i=1}^{n} a_i f_G(Y_i,\omega_i,I)\right) \quad (12)$$

$$f_G(A,\omega,I) = \sigma\left(a_A + \theta \cdot \phi_A(\omega,I)\right) \quad (13)$$

In (12) we have that $a = (a_0,\ldots,a_n)$ are a bias term and multiplicative weights associated with the production schema from which the production is drawn. In (13) we have that a_A is a bias term associated with the nonterminal A. We take σ to be a zero-centered sigmoid function $\sigma(x) = 2/(1+e^{-x}) - 1$. Note that $\sigma(x) \in (-1,1)$ with $\sigma(0) = 0$. This will be important for learning.

Equations (12) and (13) define a kind of neural network whose nodes are the placed symbols and productions of a grammar. This network alternates (nonlinear) summation with maximization.

The network defined by (12) and (13) can be trained with a form of back-propagation. The network value is defined by the parameter vector θ and a parameter vector α where α assigns bias terms and multiplicative weights to each production schema and terminal symbol. The objective of training is to set the parameter vectors θ and α. As described in section 4.2, we assume that the training data is given as classification data of the form (x_1,y_1), ..., (x_n,y_n) with $y_i \in \{-1,1\}$ and where each x_i is a triple (Y_i,ω_i,I_i). We will write $f_{\theta,\alpha}(x_i)$ for $f_G(Y_i,\omega_i,I_i)$ to emphasize the dependence on the parameter vectors θ and α. Although we have no empirical experience with Neural grammars, the following optimization seems like a natural generalization of LSVM training where $\gamma \in (0,1)$ is a margin requirement. We cannot use margin 1 because $f_{\theta,\alpha}(x) \in (-1,1)$ and hence margin requirement of 1 can never be satisfied.

$$\theta^*,\alpha^* = \operatorname*{argmin}_{\theta,\alpha} \sum_{i=1}^{n} \max(0, \gamma - y_i\ f_{\theta,\alpha}(x_i)) + \lambda_1||\theta||^2 + \lambda_2||\alpha||^2$$

This objective function can be optimized by gradient descent leading to a form of back-propagation on (12) and (13).

6 Discussion

Object detection grammars are closely related to pictorial structure models [7, 4]. Tree-structured pictorial structure models are a special case of isolated deformation grammars. In a pictorial

structure model the structure of the object is fixed, whereas in an object detection grammar we can model objects with variable structure, both by using subtypes and multiple possible decompositions of a symbol into sets of parts. Object detection grammars also allow for the same part to be re-used within an object model, such as having a single eye model appearing twice in a face. The pictorial structure matching problem involves finding the "best" match of a model to an image. For a given (tree-structured) pictorial structure model we can define an object detection grammar by picking an arbitrary root part. If Y is the nonterminal corresponding to that part we have that $f_G(Y, \omega, I)$ is the score of the best configuration of the parts constrained to placing the root at position ω.

Other grammar formalisms have been described in the literature (e.g. [8, 12, 11]). The work in [8] gives a generative model for images based on a hierarchy of reusable parts and compositional grouping rules. The model described here is closely related but the matching problem we consider, computing $f_G(Y, \omega, I)$, does not lead to a globally coherent interpretation of an image. In particular an optimal derivation tree will generally explain only a part of an image, and some terminal symbols may overlap leading to potential overcounting of evidence. We have concentrated on the problem of computing optimal derivations in this way because, as discussed in Section 2.1, bag parsing is NP-hard even for "Markov models". The problem we solve is an alternative that allows for efficient computation, but does not lead to a complete interpretation of an image by itself.

With respect to learning we have only discussed the problem of learning parameters for a grammar model with fixed structure. Being able to learn the structure of a grammar from weakly labeled data is a challenging problem that remains to be solved.

References

[1] C. Brew. Letting the cat out of the bag: generation for shake-and-bake MT. In *International Conference on Computational Linguistics*, 1992.

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[3] P. Felzenszwalb and D. Huttenlocher. Distance transforms of sampled functions. Technical Report 2004-1963, Cornell University, 2004.

[4] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1), 2005.

[5] P. Felzenszwalb and D. McAllester. The generalized A* architecture. *Journal of Artificial Intelligence Research*, 29:153–190, 2007.

[6] P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[7] M.A. Fischler and R.A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computer*, 22(1), 1973.

[8] Y. Jin and S. Geman. Context and hierarchy in a probabilistic image model. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[9] D. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5, February 1977.

[10] C.N.J. Yu and T. Joachims. Learning structural SVMs with latent variables. In *International Conference on Machine Learning*, 2009.

[11] L.L. Zhu, Y. Chen, and A. Yuille. Unsupervised learning of probabilistic grammar-markov models for object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(1):114–128, 2009.

[12] S.C. Zhu and D. Mumford. A stochastic grammar of images. *Foundations and Trends in Computer Graphics and Vision*, 2(4):259–362, 2007.