Spatial Inference Machines

Roman Shapovalov Dmitry Vetrov
Lomonosov Moscow State University
http://bayesgroup.ru

Pushmeet Kohli
Microsoft Research Cambridge
http://research.microsoft.com/~pkohli/

Abstract

This paper addresses the problem of semantic segmentation of 3D point clouds. We extend the inference machines framework of Ross et al. by adding spatial factors that model mid-range and long-range dependencies inherent in the data. The new model is able to account for semantic spatial context. During training, our method automatically isolates and retains factors modelling spatial dependencies between variables that are relevant for achieving higher prediction accuracy. We evaluate the proposed method by using it to predict 17-category semantic segmentations on sets of stitched Kinect scans. Experimental results show that the spatial dependencies learned by our method significantly improve the accuracy of segmentation. They also show that our method outperforms the existing segmentation technique of Koppula et al.

1. Introduction

Probabilistic graphical models are a powerful tool for modeling interactions between random variables. They are used to solve a wide range of labelling problems encountered in computer vision (semantic segmentation [5, 7, 11], dense stereo estimation [18], denoising [20, 13]) and beyond (part of speech tagging in natural language processing, gene finding in bioinformatics). Prediction using these models typically comprises two key steps: learning, which involves estimation of the dependencies between variables from training data, and inference, which involves estimation of the most probable values of the variables of interest under the model.

Most methods for learning graphical model parameters are inspired by statistical learning theory and follow the principle of empirical risk minimization. Inference of the maximum a posteriori (MAP) solution in graphical models, too, is a well studied problem. Although it is NP-hard to find the exact MAP solution of a general model, a number of methods (like graph cuts or message-passing algorithms such as belief propagation) have been proposed to find exact or approximate solution in certain families of models.

Pairwise Markov random field (MRF) is a widely-used variant of graphical models that incorporates dependencies only between pairs of random variables. The dependencies in pairwise random fields are typically sparse (with few exceptions, e.g. [9]). In other words, most pairs of variables are assumed to be conditionally independent given the rest of the variables. This enables efficient inference of the MAP solution under the model, but low expressive power of such sparse MRFs limits their prediction accuracy. Although recent work has tried to overcome this limitation using higher order models, such methods suffer from increased computational cost [7].

Labelling via sequential classification. Sequential classification is an alternative prediction mechanism that can also handle dependencies between variables. Starting from an initial estimate, it works by repeatedly obtaining refined estimates of the variables using the inferred values of variables from the previous iteration. Initially, it was applied in natural language processing for part of speech tagging [2]; subsequently, it spread to computer vision applications. The *auto-context* algorithm [19] is a typical example. It uses a sequence of classifiers to infer pixel labels. Each classifier takes as argument the image labelling from the previous iteration. Specifically, the classifier uses the previous labelling of pixels at certain displacements with respect to the pixel of interest to estimate the label of that pixel. *Semantic texton forest* (STF) [17] is a two-stage sequential classification algorithm that works in a similar manner. In the first stage, STF computes semantic textons and region priors using local appearance of pixels. In the second stage, it classifies pixels according to the output response of the first stage pooled in rectangles around the point. *Entanglement forest model* [10] generalizes both auto-context and STF. It comprises of a collection of decision trees. Each node of each tree computes features based on the predictions made by the nodes in the upper (previous) levels of the tree.

Munoz et al. [12] introduced the *stacked hierarchical labeling* framework for semantic segmentation that was later applied to 3D point cloud data by Xiong et al. [21]. In that model, sequential classification is performed on consecutive layers of a hierarchical image segmentation, from coarse to fine. For each region, the probabilistic classifier computes the posterior distribution over labels and passes it to the classifiers for regions of the lower layer of the segmentation hierarchy as features. Ross et al. [15] inter-

pret this idea as performing generalized message-passing inference in graphical models and develop the *inference machines* framework to explain it. Under this framework, the solution to the labelling problem is found by performing inference in a graphical model. However, instead of learning the parameters of potential functions and then performing inference using a message passing algorithm, they propose to learn the messages passed by the inference algorithm directly. Their method is efficient and achieves results that are similar in accuracy to a computationally expensive conventional non-linear MRF training method (namely, functional gradient boosting [11]). However, it does suffer from a limitation: it only captures local interactions via small higher-order factors, and does not take the *scene context* into account.

The importance of accounting for scene context is widely acknowledged in the computer vision community [14, 19, 6, 3]. *Semantic context*—the dependency of labels on the labels of different parts of the scene [17]—is particularly difficult to account for in models based on local dependencies. While in theory inference machines can employ factors that span different parts of the scene, it remains unclear how to define them. Moreover, message update rules would become more complicated and would require more expressive message predictors, which would lead to overfitting.

Incorporating context in inference machines. We build on the model of Ross et al. [15] by incorporating contextual dependencies. Our method is different from the conventional inference machine based method proposed in [15] in the following ways:

- we learn the message prediction functions using a two-stage procedure: first, we train probabilistic output classifiers that predict individual factor messages using the aggregated messages from the previous message passing step as the argument; second, we learn weights that will be used for aggregating the individual messages;

- we change the notion of a factor that generates the factor-to-node message: in our formulation it is an ordered pair of *source* and *destination* variable sets. The source (group of variables) of the factor affects the belief about the label of the destination (individual variable) by passing a message in this direction;

- we incorporate prior knowledge about different kinds of contextual dependencies by means of *factor types*. A separate prediction function is learned for each factor type, which helps to keep them simple. The aim of factor type design is to model mid-range (scene context) dependencies that can be highly anisotropic unlike to close-range (low-level) dependencies. We show how to use these factors to account for spatial semantic context in 3D point cloud segmentation.

We evaluate our method on 3D point clouds provided by Koppula et al. [8] that were generated by stitching Kinect depth frames. Experimental results show that our method outperforms the semantic segmentation technique proposed by Koppula et al. [8] in terms of both speed and accuracy.

2. Graphical models and inference machines

Notation and background. Prior to describing the spatial inference machines framework, we review the formulation of graphical models in terms of factor graphs to settle the notation. Our task is to predict the value of a vector of discrete random variables $\mathbf{y} \in L^N$, where L is the set of states, and N is the number of variables. In semantic segmentation, each variable state y_v may denote the label assigned to the corresponding super-pixel—a group of co-located similar image pixels or 3D points.

Graphical models express relations between variables in form of *factors*. We define a factor $f \in F$ as a group of variables along with the corresponding *potential function* $\phi_f(\cdot)$, which scores label assignments to the variables. The probability of any label configuration $\mathbf{y}$ is defined as

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{f \in F} \phi_f\Big(\bigcup_{v \in f} \{y_v\} \Big), \qquad (1)$$

where Z is the normalizing constant that makes it a proper probability distribution.

Pairwise random field is a special case of this formulation where each factor is a set of exactly two variables. For segmentation of image pixels or 3D point clouds, those pairwise factors typically correspond to the pairs of super-pixels that are spatially close to each other and/or have similar appearance. The potential function of any given factor is typically modeled as a parametric function defined over the variables from the the factor scope and features of local scene appearance. During training, parameters of these functions are estimated by minimizing some loss function (e.g. some approximation of log-likelihood loss [4] or regularized hinge loss [8, 16]) on a training set. Learning algorithms typically proceed by iteratively calling an inference-based oracle, which is time-consuming in practice.

Message-passing algorithms are widely used for estimating marginal distributions over variables in the above-mentioned models. These methods work by iteratively passing messages between variables and factors. The messages sent from factors to variables in the n-th iteration of the algorithm are computed as

$$\mu^n_{f \to v}(y_v) = \sum_{\mathbf{y}' : y'_v = y_v} \phi(\mathbf{y}') \prod_{v' \in f \setminus \{v\}} \mu^n_{v' \to f}(y'_v), \qquad (2)$$

where $\mu^n_{v\to f}$ are the messages from variables to factors:

$$\mu^n_{v\to f}(y_v) = \prod_{f'\colon v\in f', f'\neq f} \mu^{n-1}_{f'\to v}(y_v). \tag{3}$$

Messages are typically initialized to uniform distributions over labels.

To obtain the final unnormalized estimates of marginal probabilities for each variable, messages from the corresponding factors are multiplied: $p(y_v) \propto \prod_{f\colon v\in f} \mu^n_{f\to v}(y_v)$. The order in which messages are computed for different variables or factors (so-called scheduling policy) may vary. For graphical models with cycles this loopy belief propagation method may not converge, though running it for a restricted number of iterations often works well in practice, especially for the graphs without many tight loops.

Message-passing inference machines. Learning parameters of the potential functions in a graphical model is typically achieved by minimizing hinge loss or (some proxy for) maximum likelihood loss. In both cases, optimization procedure is iterative and requires MAP inference in each step, which makes training computationally demanding.

Note that the message-passing formalism allows us to directly deal with messages rather than factors forming the graphical model. So instead of learning the factors, one may try to directly learn how to update messages that are circulating during loopy belief propagation. Ross et al. [15] suggest a message-passing approach to *sequential classification*. In their method, messages to factors are treated as arbitrary functions of the previous iteration messages to factors:

$$\mu^n_{v\to f}(y_v) = \bar{g}_n\Bigg(\mathbf{x}_f, \bigoplus_{\substack{f'\colon v\in f',\\ f'\neq f}} \mathbf{x}_{f'}, \bigoplus_{\substack{v', f'\colon v'\in f',\\ v\in f', f'\neq f}} \mu^{n-1}_{v'\to f'}(y_{v'})\Bigg). \tag{4}$$

Here, the operation $\oplus$ is domain-specific and can imply, for instance, feature averaging or stacking (concatenation), and $\mathbf{x}_f$ is a vector of appearance features for the factor scope (e.g. color or texture features of the corresponding superpixels). Similar to belief propagation, inference machines recompute messages iteratively, applying (4). In the last iteration, the classifier applies a function with a slightly different signature: for each variable it takes as argument all the messages from the neighbouring factors and returns the predicted label of the variable.

Since message computation in (4) avoids using potential functions, they don't need to be modelled explicitly. Training an inference machine involves directly learning the functions $\bar{g}_n$ that can be different in each iteration. The function $\bar{g}_n$ depends on the factor f only by means of its features $\mathbf{x}_f$. If factors have different impact on the label assigned to a variable depending on the local features, one needs to train a quite complex function to reflect this fact.

The prediction function $\bar{g}_n$ can take form of any probabilistic output classifier. During training, the previous iteration messages are estimated on the hold-out set. The rest of the data are used for fitting the prediction functions, where ground truth values of y_v serve as values of the target variables.

3. Spatial inference machines

This section explains how we extend the inference machine model by employing the new notion of *factor types*. We first define *d-factor* (short for directed factor) using a pair $p = (d_f, S_f)$ composed of a destination variable d_f and a set of source variables S_f. Each d-factor f belongs to one of the factor types $t(f) \in T$, which encodes how source variables' labels and corresponding features impact the label of destination. Factor types allow us to include prior knowledge about the interactions between source and destination variables. This is important for modelling mid-range dependencies that can be highly anisotropic. Such explicit coding leads to simpler message prediction functions and reduces the risk of overfitting. We compute the messages from the source to the destination variables using learned factor type specific functions:

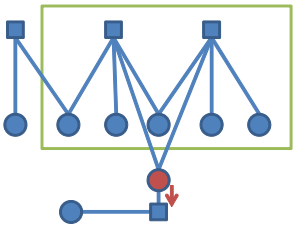
$$\mu^n_{S_f\to d_f}(y_{d_f}) = \\ g_{n,t(f)}\big(p_{n-1}(y_{d_f}), \mathbf{x}_{d_f}, \mathbf{x}_f, \mathbf{x}_{S_f}, \mathbb{E}_{v\sim S_f}(p_{n-1}(y_v))\big). \tag{5}$$

Here $g_{n,t(f)}(\cdot)$ is the prediction function used in the n-th iteration for the factor type $t(f)$. We use random forest [1] for prediction, but any other probabilistic output classifier can also be used. The d-factor features are combined with destination features $\mathbf{x}_{d_f}$, the previous iteration beliefs about the label $p_{n-1}(y_{d_f})$, d-factor features $\mathbf{x}_f$, aggregated source features $\mathbf{x}_{S_f}$, and the averaged source set beliefs from previous iteration $\mathbb{E}_{v\sim S_f}(p_{n-1}(y_v))$. In the n-th iteration of the algorithm, the *belief* (probability of each label y_v) is defined as the weighted product of d-factor *messages* from S_f to d_f:

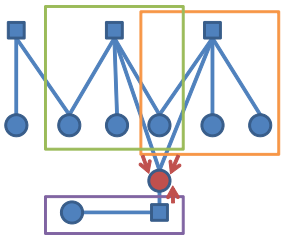
$$p_n(y_v) \propto \prod_{\{f:d_f=v\}} (\mu^n_{S_f\to v}(y_v))^{\alpha^n_{t(f)}}, \tag{6}$$

where $\alpha_{t(f)}$ is the weight of the response of the factor type $t(f)$.

The last argument in (5) averages the beliefs associated with different class labels from the previous iteration. This causes the loss of information about the layout of the labels within the source region, but prevents overfitting and makes inference computationally tractable and independent of point density. Source regions should be chosen to be small enough to prevent excessive averaging, but large enough to be robust.



(a) Ross et al. [15]



(b) Our method

Figure 1: Different ways to apply sequential classification. Variables are shown as circles, factors as squares. To compute the message that a variable (red circle) sends to a factor, Ross et al. [15] (a) use the previous iteration messages that have been sent to all variables that share a factor with the marked variable. Messages from that variable to the two other factors are computed in a similar way. In our method (b), three d-factors (sources are sets of variables within the corresponding coloured boxes) send messages to the common destination (red circle), which are multiplied to get the next iteration (or final) label

Note that in contrast to (4), our message prediction function does not depend on previous iteration messages directly; instead it takes their weighted products, i.e. iteration beliefs. Also, the scope of our predictors is smaller: they take the predicted labels for only the variables involved in a single factor, while the prediction functions used by Ross et al. [15] take concatenated messages from all the variables that share a factor with v, excluding the target factor f. Fig. 1 illustrates the difference. We combine the results of the small-scope predictors explicitly using (6) to get the beliefs for the next iteration or final marginal probabilities. Weights α^n could be just set to ones or learned by a different procedure, as described later in this section.

Preventing overfitting of the model. Training of our model involves learning the parameters of the prediction function (5) and (optionally) the weights in (6). Since message prediction functions depend on the beliefs from the previous iteration, usage of the same training set for estimation of previous iteration beliefs might lead to a biased model. To prevent overfitting, we use k-fold cross-validation in each iteration to get beliefs, motivated by Munoz et al. [12]. For a certain fold, the predictors for each factor type are trained on all the other folds, and then used to get the fold labels for the next iteration. To train the final predictors to be used at inference stage, all folds are used. We summarize the training process in Algorithm 1.

Spatial d-factors. *Spatial* factors are important special types of d-factors that model dependencies between variables that represent points in some coordinate space. For example, factor types could be parameterized by a pair: $(\delta\mathbf{v}, r)$. This means that each 2D pixel $\mathbf{v} = (x, y)$ or 3D point $\mathbf{v} = (x, y, z)$ induces one d-factor of each spatial type, where $\mathbf{v}$ is the destination, and all the pixels/points within the distance r from the point $\mathbf{v} + \delta\mathbf{v}$ form the d-factor source. This formulation allows us to model arbitrary mid- and long-range dependencies. Pairwise dependencies between neighboring points may form a special factor type too. These *structural* d-factors are responsible for close-range interactions. They are not associated with any particular displacements, since they typically express isotropic interactions.

Algorithm 1 Inference machine training

Input: labeled instance $(\mathbf{x}, \mathbf{y})$, set of factors F divided on folds $\mathbf{f}$, set of factor types T, number of iter's N.
Output: set of prediction functions $\{g_{n,t}(\cdot)\}_{t\in T, n\in[1,N]}$
set uniform initial beliefs $p_0(y_v), \forall y_v \in \mathbf{y}$
for $n = 1$ to N **do**
  for all $\mathbf{f} \in F$ **do**
    for all $t \in T$ **do**
      fit temporary prediction function $g_{tmp,t}(\cdot)$ from (5) using d-factors $\left\{ f \in \bigcup_{\mathbf{f}'\in F\setminus\{\mathbf{f}\}} \mathbf{f}' \;\middle|\; t(f) = t \right\}$
    end for
    $\mu^n_{S_f \to d_f}(y_{d_f}) \leftarrow g_{tmp,t(f)}(\langle \text{features of } f \rangle), \forall f \in \mathbf{f}$
  end for
  for all $t \in T$ **do**
    fit this iteration prediction function $g_{n,t}(\cdot)$ using d-factors $\left\{ f \in \bigcup_{\mathbf{f}'\in F} \mathbf{f}' \;\middle|\; t(f) = t \right\}$
  end for
  specify factor type weights α^n, e.g. by setting $\alpha^n = \mathbf{1}$, or by maximizing (7)
  if $n < N$ **then**
    for all $y_v \in \mathbf{y}$ **do**
      obtain this iteration beliefs $p_n(y_v)$ estimated on the hold-out sets using (6)
    end for
  end if
end for

Learning factor type weights. Using a large number of factor types may lead to overfitting and degradation of final predictive performance. To prevent this, we introduce weights α^n_t (6) that modulate the contribution of different factors. The parameters α^n are learned at each iteration n by maximizing the regularized sum of probabilistic estimates for correct labels:

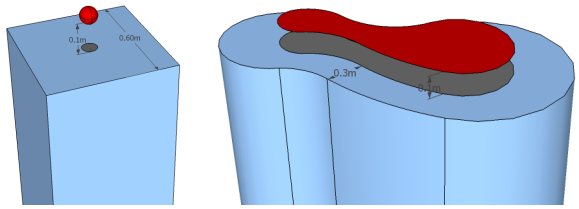
$$\max_{\alpha\geq 0} \sum_v \left(\frac{\prod_{\{f:d_f=v\}} (\mu^n_{S_f\to v}(\bar{y}_v))^{\alpha_{t(f)}}}{\sum_{l\in L} \prod_{\{f:d_f=v\}} (\mu^n_{S_f\to v}(l))^{\alpha_{t(f)}}} + C \sum_{t\in T} \alpha_t \right), \tag{7}$$

where $\bar{y}_v$ is the ground truth label of v, and l sums over all

Table 1: Factor types used in our model for point cloud segmentation. The rows contain names of factor types and their acronyms, followed by relative coordinates of corresponding support regions

#	name	ac	b. box $((x_0, y_0, z_0), (x_1, y_1, z_1))$, m
0	structural	S	n/a
1	Local	Lo	$((-0.1, -0.1, -0.1), (0.1, 0.1, 0.1))$
2	To-Down	Td	$((0.1, -0.3, -\infty), (\infty, 0.3, -0.1))$
3	From-Up	Fu	$((-\infty, -0.3, 0.1), (-0.1, 0.3, \infty))$
4	Down	D	$((-0.3, -0.3, -\infty), (0.3, 0.3, -0.1))$
5	Left-Restr	Lr	$((-0.3, -1.0, -0.3), (0.3, -0.1, 0.3))$
6	Right-Restr	Rr	$((-0.3, 0.1, -0.3), (0.3, 1.0, 0.3))$
7	From	F	$((-\infty, -0.3, -0.3), (-0.1, 0.3, 0.3))$
8	To	T	$((0.1, -0.3, -0.3), (\infty, 0.3, 0.3))$
9	Up	U	$((-0.3, -0.3, 0.1), (0.3, 0.3, \infty))$

the class labels L. L_1 regularization of weights is used to remove weak factor types and leads to a sparse representation. In particular, if the data lacks mid-range dependencies, the corresponding factor types' weights are set to zero due to regularization (see Section 5). C is the per-variable regularization parameter. Maximization is performed using a first-order optimization method. Note that this factor type selection step is conceptually similar to the one in the *things and stuff* model [6], although the authors solve a different problem, i.e. object detection. Their spatial dependencies of the form "the detection i is about 100 pixels away from the region j" are similar to our spatial factors. They generate a large set of such dependencies and select the relevant ones using the structured expectation-maximization algorithm, while we use L_1 regularization.



(a) Point src region (b) Superpixel source region

Figure 2: Regions used to collect source points when the destination element is represented by point (a) and superpixel (b) for *Down* factor type. The red sphere and plain segment denote the destination point and superpixel in the point cloud, respectively. The statistics that form feature vectors ($\mathbf{x}_{S_f}$ and $\mathbb{E}_{v \sim S_f}(p_{n-1}(y_v))$) are estimated using the points within the blue regions

4. Implementation details

We now demonstrate how the model can be applied to semantic segmentation of colored 3D point clouds (Fig. 3a).

Model structure. The ground truth labelling generated by Koppula et al. [8] was done at the super-pixel level and resulted in all points within each super-pixel taking the same label. Given this, we found it was reasonable to classify superpixels rather than individual points, particularly, for efficiency reasons. We also keep the structure of interactions of Koppula et al. [8]: all the superpixels that have minimum distance less than 0.6 m are connected by *structural* links. A structural link which connects any superpixels v and u induces two pairwise d-factors $(v, \{u\})$ and $(u, \{v\})$.

To define spatial d-factors, we introduce a coordinate system associated with each spatial d-factor destination point. Because the point clouds capture indoor scenes, we can define those coordinate systems such that no degrees of freedom are left. The vertical direction is defined unambiguously. For every point, the position of camera that filmed it is known, and most objects are situated close enough to walls, so there is another dedicated direction: towards camera, orthogonally to the closest wall.

We used an heuristic algorithm for finding walls (robust vertical plane fitting), which was able to find almost all walls. Note that this is not our final result for wall detection: we just use it to compute the direction orthogonal to the closest wall, which is our X axis; Z is directed upwards; and the Y axes (horizontally, along the wall) is estimated as the orthogonal direction to them both. We define spatial d-factors for this problem using bounding boxes (possibly open) in this relative coordinate system. See Table 1 for the list of spatial factor types we use. For example, *Down* factor type (line 4) assumes that spatial d-factor source includes all the points lower than the destination with 10 cm gap in the 60 cm $\times$ 60 cm corridor (Fig. 2).

Since our elementary unit is a superpixel, we define its source as the union of all sets that would be sources for individual points of the superpixel. Thus, for a tabletop segment, *Down* factor type collects all the points below it with 10 cm gap and including a 30 cm border (Fig. 2b). Note that this definition of spatial d-factors is similar to the dependencies Desai et al. [3] used for object detection in images where the pairwise potentials indicated one of the *far, near, above, below, next-to* and *on-top* relative locations of candidate detections.

Features. We use the unary and pairwise features from Koppula et al. [8]. Unary features describe appearance of the superpixel, e.g. its planarness, orientation, and color gradient histogram. Edge features describe the relation between superpixels, e.g. angle between normals or vertical displacement between centroids. See Table 2 and the orig-

Table 2: Unary and pairwise features derived from Koppula et al. [8]. λ_i refers to the vector of eigenvectors of the superpixel covariance matrix, sorted in the order of decreasing eigenvalues

Unary features for superpixel *i*	cnt
Visual appearance	**48**
Histogram of HSV color values	14
Average HSV color value	3
Average of HOG features of the blocks in image spanned by the points of a superpixel	31
Local shape and geometry	**8**
Linearness $(\lambda_{i0} - \lambda_{i1})$, planarness $(\lambda_{i1} - \lambda_{i2})$, scatter λ_{i0}	3
Vertical component of the normal n_{iz}	1
Vertical position of the centroid c_{iz}	1
Vertical and horizontal extent of the bounding box	2
Distance from the scene boundary	1
Pairwise features for (*i*,*j*)	cnt
Visual appearance	**3**
Difference of average HSV color values	3
Local shape and geometry	**2**
Coplanarity and convexity	2
Geometric context	**6**
Horizontal distance between centroids	1
Vertical displacement between the centroids $(c_{iz} - c_{jz})$	1
Dot product of the normals $\mathbf{n_i} \cdot \mathbf{n_j}$	1
Difference in the angles between the normals and the vertical vector $(\cos^{-1} n_{iz} - \cos^{-1} n_{jz})$	1
Distance between the closest points	1
Relative position from the camera (in front of / behind)	1

inal paper for more details. The basic classifiers (5) use concatenation of local feature vectors and previous iteration labels, which may depend on the factor type. For spatial d-factors we concatenate local features of destination, mean previous iteration labels of source and destination, totally $56 + 2|L|$ values, where $|L|$ is the number of class labels. For structural links we also add factor features as defined by Koppula et al. [8], totally $56 + 11 + 2|L|$ values. Note that our spatial factors do not include any source or edge features.

Form of predictors. We use random forest [1] of 100 trees as a prediction function for inference machines. To determine a split function at a node, splits on d randomly selected features are tested according to Gini index; d is chosen as the square root of the number of features.

5. Experiments

Dataset. We evaluate our method on a recent dataset collected by Koppula et al. [8]. The authors of this dataset used Kinect to collect depth maps and RGB images of office and living room interiors. Scans corresponding to common scenes were stitched automatically to get colored 3D point clouds. Each of the 24 office and 28 home scenes was reconstructed from 8–9 scans. The point clouds were segmented into 17 categories by labelling superpixels. The point clouds corresponding to office scenes were given the labels: (*wall, floor, tableTop, tableDrawer, tableLeg, chairBackRest, chairBase, chairBack, monitor, printerFront, printerSide, keyboard, cpuTop, cpuFront, cpuSide, book, paper*), while those corresponding to home scenes were labelled with (*wall, floor, tableTop, tableDrawer, tableLeg, chairBackRest, chairBase, sofaBase, sofaArm, sofaBackRest, bed, bedSide, quilt, pillow, shelfRack, laptop, book*).

Protocol. We try to repeat the protocol of Koppula et al. [8] exactly. We perform 4-fold cross-validation for both home and office scenes such that no scene can be shared by two or more folds. The structural links used by our model correspond to the pairwise factors used in [8] which only operate on points that labelled with one of the 17 classes, i.e. points corresponding to the background are discarded from both training and test. This results in 690 super-pixels for the office scenes and 800 for the home scenes. Most super-pixels belong to the wall class in both sets.

To aggregate precision over classes, both micro- and macro-averaging are used. Both measures are important, because micro-precision (accuracy) p tends to underestimate mislabeling of under-represented classes, while macro-precision P and recall R treat all classes equally regardless of their size:

$$p = \frac{\sum_{i=1}^{|L|} TP_i}{\sum_{i=1}^{|L|} TP_i + FP_i} = \frac{\sum_{i=1}^{|L|} TP_i}{\sum_{i=1}^{|L|} TP_i + FN_i} = r, \quad (8)$$

$$P = \frac{1}{|L|} \sum_{i=1}^{|L|} \frac{TP_i}{TP_i + FP_i}, R = \frac{1}{|L|} \sum_{i=1}^{|L|} \frac{TP_i}{TP_i + FN_i}, \quad (9)$$

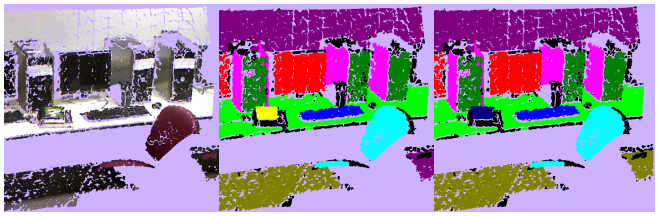
where L is the set of class labels, TP_i, FP_i, TN_i, FN_i are the true positive, false positive, true negative, and false negatives rates for the i-th class, respectively. The results are summarized in Table 3.

Segmentation quality. The model with only structural dependencies (STR) works better than the linear CRF [8], although the same structure and features are used. This probably happens because of the non-linear prediction function used in our method.

It turns out that adding spatial factor types without learning weights (STR+SPAT), in spite of being well-motivated from a probabilistic viewpoint, performs worse than just using structural factors. Learning of weights (STR+SPAT_C) works better than the previous approaches in theory, because it is more general: when all weights are set to 1, it corresponds to combination of structural and spatial factor types, the latter could be switched off by setting their

Table 3: Results on *Office* and *Home* scenes. Micro and macro precision and recall after 5 iterations of training on cross-validation. STR: only structural factors are used. STR+SPAT: structural and all spatial factor types are combined, all weights are assumed to be 1. STR+SPAT_C: 10 factor types, learn coefficients with regularization, $C = 0.03$

Method	Office scenes			Home scenes		
	micro	macro		micro	macro	
	P/R	Prec	Rec	P/R	Prec	Rec
chance	0.262	0.058	0.058	0.293	0.058	0.058
SVM_CRF [8]	0.840	0.805	0.726	0.722	0.568	0.548
STR	0.889	0.872	0.825	0.777	0.690	0.609
STR+SPAT	0.866	0.811	0.794	0.711	0.578	0.527
STR+SPAT_C	**0.902**	**0.882**	**0.844**	**0.783**	**0.716**	**0.620**



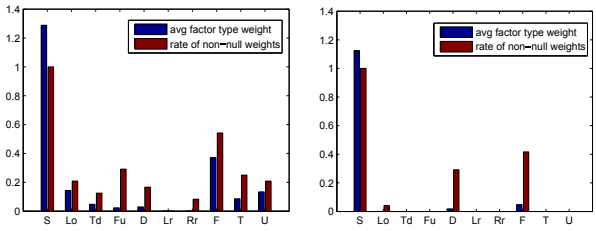
(a) Source colored point cloud (b) Only structural factor types (c) With spatial factor types

Figure 3: Scene example where spatial factor types improve segmentation quality. The model with only structural factor types (b) misclassifies the book superpixel (on the left) and the floor superpixel (on the right), while the model with structural and spatial factor types (c) classifies the whole scene correctly. Color map: ***wall***, ***floor***, ***tableTop***, ***chair***, ***monitor***, ***keyboard***, ***cpuTop***, ***cpuFront***, ***cpuSide***, ***book***. *Better viewed in color*

weights to 0. Regularization is used to prevent overfitting. In practice, large regularization coefficients lead to the spatial factor types being discarded (they are assigned zero weights).

In office scenes learned spatial factor types gain 1–1.5% improvement. Their poorer performance on home scenes can be explained by the idiosyncracies of the data. While a single wall is present in office data, corners are usual in home data. Near the corners, the relative angle the to wall is ambiguous, so "horizontal" spatial factor types are not reliable. While "vertical" factor types are still reliable in this case, they are often overpowered by structural factors, which connect the superpixels within 0.6 m distance and thus almost surely connect superpixels that are close by horizontal position.

Please note that we fixed the value of the hyperparameter $C = 0.03$; task-specific learning of this parameter can further improve results, but we refrained from doing this because the datasets were too small to obtain a separate



(a) *Office* weights (b) *Home* weights

Figure 4: Factor type weights averaged across iterations and folds, and rate of non-null d-factors of each type for *Office* and *Home* data [8]. It can be seen that structural (S) factor type weights are never null, while left (Lr) and right (Rr) factor types are almost useless. This may mean that there is no particular order of objects on tables

validation set. Spatial factor types (Table 1) were defined suboptimally, which also provides room for improvement. The parametrization of spatial factor types with continuous variables potentially allows for efficient search of the best-performing factor types using gradient-based or sampling techniques, which is a promising direction for future work.

The model with only structural factors classifies *book* on the left in Fig. 3b as *cpuTop* due to neighboring *cpuFront* and *cpuSide* superpixels. Spatial features of structural dependencies are not expressive enough to forbid *cpuTop* anywhere except on the top of *cpuFront* and *cpuSide*. Spatial d-factors account for that explicitly, the book is classified correctly (Fig. 3c). Also, structural factors are restricted to the length of 0.6 m, so they cannot encode the dependency between the *tableTop* and *floor* superpixels on the right. Increasing the distance threshold for structural dependencies would lead to capturing spurious dependencies given the limited training data [8]. The model with spatial factor types classifies the floor superpixel correctly.

Computation time. An important advantage of inference machines is fast inference. For the model with the structural and 9 spatial factor types, average inference time for an office scene is 0.7 second for 5 iterations on 8-core CPU. Note that this does not include pre-processing time, where source indices and features of structural and spatial d-factors are computed. Extracting source indices might take up to several minutes per scene if they are estimated for the whole superpixel (as shown in Fig. 2b), not just for only the centroid of the superpixel. In contrast, MAP inference in CRF using mixed-integer programming takes 18 minutes per scene; solving the linear programming relaxation using quadratic pseudo-boolean optimization is faster (10 seconds), but 2–3% less accurate [8]. Thus, our inference is either thousands times faster and 6% more accurate, or ten times faster and at least 8% more accurate.

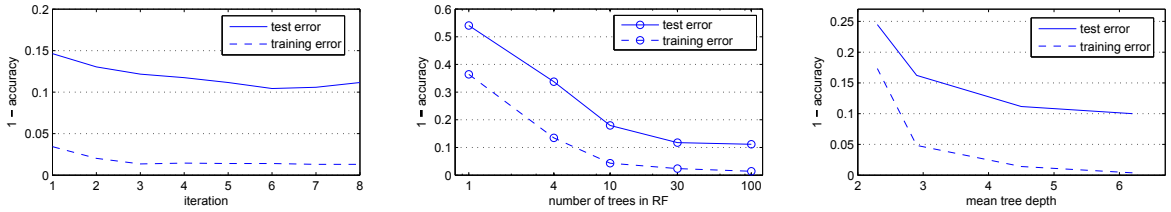


Figure 5: Left: evolution of test set and training set error during training on *Office* scenes for 100 trees in random forest. Training error is decreasing, while test error tends to stabilize or even increase after the iteration 5–6 due to overfitting. Center: error after 5 iterations depending on the number of trees used in random forest. Right: error after 5 iterations when depth of trees is restricted during training

Relevance of spatial factor types. Because of L_1 regularization in the objective function (7), the learned weights are sparse. The null weights mean irrelevant factor types, so the weights can give us a clue on which subset of factor types is sufficient for modeling the relations (see Fig. 4). The degree of sparsity depends on the number of iteration and regularization coefficient C. In our experiments, we observed that in the first iteration only the weights corresponding to structural factors were assigned non-zero weights for both the home and office datasets. This hints to the fact that structural factors define strong relationships, which can be later improved by spatial d-factors. This also reflects the fact that close-range dependencies are generally more informative than mid-range ones. Another explanation is that there are typically several structural factors for each destination (that have a single weight for all) and one d-factor of each spatial type (with separate weights), so the regularization penalizes the latter more.

Number of iterations. We discovered that 5 iterations of training are typically enough. After that, accuracy stabilizes and sometimes even slightly degrades because of overfitting (Fig. 5). During the entire training process, accuracy is uniformly higher in the model with spatial factor types than without them.

6. Conclusions

In this paper we introduce a method for 3D point cloud segmentation that builds on the inference machines framework [15] by explicitly accounting for spatial semantic context. The resulting method outperforms max-margin conditional random field learning [8] both in terms of speed and accuracy. Note that such a technique can be applied to other segmentation problems where mid-range context dependencies are expected, e.g. to image segmentation.

Acknowledgments. This work was supported by Microsoft Research Connections programs in Russia and by RFBR grants 12-01-00938, 12-01-33085.

References

[1] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 3, 6

[2] E. Brill. A simple rule-based part of speech tagger. In *ACL*, pages 112–116, Trento, IT, 1992. 1

[3] C. Desai, D. Ramanan, and C. Fowlkes. Discriminative models for multi-class object layout. In *ICCV*, pages 229–236, 2009. 2, 5

[4] J. Domke. Parameter Learning with Truncated Message-Passing. In *CVPR*, number x, pages 2937–2944, Colorado Springs, CO, 2011. 2

[5] B. Fulkerson, A. Vedaldi, and S. Soatto. Class segmentation and object localization with superpixel neighborhoods. In *ICCV*, pages 670–677, 2009. 1

[6] G. Heitz and D. Koller. Learning spatial context: Using stuff to find things. In *ECCV*, pages 30–43, Marseille, France, 2008. 2, 5

[7] P. Kohli, L. Ladický, and P. H. S. Torr. Robust Higher Order Potentials for Enforcing Label Consistency. *IJCV*, 82(3):302–324, Jan. 2009. 1

[8] H. S. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic Labeling of 3D Point Clouds for Indoor Scenes. In *NIPS*, Granada, ES, 2011. 1, 2, 5, 6, 7, 8

[9] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *NIPS*, 2011. 1

[10] A. Montillo, J. Shotton, J. M. Winn, J. E. Iglesias, D. N. Metaxas, and A. Criminisi. Entangled decision forests and their application for semantic segmentation of ct images. In *IPMI*, 2011. 1

[11] D. Munoz, J. A. Bagnell, and M. Hebert. Stacked hierarchical labeling. In *ECCV*, Heraklion, Grece, 2010. 1, 4

[12] D. Munoz, J. A. Bagnell, N. Vandapel, and M. Hebert. Contextual classification with functional Max-Margin Markov Networks. In *CVPR*, pages 975–982, Miami, FL, June 2009. 1, 2

[13] S. Nowozin and C. H. Lampert. Global connectivity potentials for random field models. In *CVPR*, pages 818–825, June 2009. 1

[14] A. Rabinovich, A. Vedaldi, C. Galleguillos, E. Wiewiora, and S. Belongie. Objects in Context. In *ICCV*, 2007. 2

[15] S. Ross, D. Munoz, M. Hebert, and J. A. Bagnell. Learning Message-Passing Inference Machines for Structured Prediction. In *CVPR*, pages 2737–2744, Colorado Springs, CO, 2011. 1, 2, 3, 4, 8

[16] R. Shapovalov and A. Velizhev. Cutting-Plane Training of Non-associative Markov Network for 3D Point Cloud Segmentation. In *3DIMPVT*, pages 1–8, Hangzhou, China, 2011. 2

[17] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *CVPR*, June 2008. 1, 2

[18] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for Markov random fields. *LNCS*, 3952(6):16–29, 2006. 1

[19] Z. Tu. Auto-context and its application to high-level vision tasks. In *CVPR*, Anchorage, AL, June 2008. 1, 2

[20] O. J. Woodford, C. Rother, and V. Kolmogorov. A global perspective on MAP inference for low-level vision. In *ICCV*, number Iccv, pages 2319–2326. ICCV, 2009. 1

[21] X. Xiong, D. Munoz, J. A. Bagnell, and M. Hebert. 3-D Scene Analysis via Sequenced Predictions over Points and Regions. In *ICRA*, Shanghai, China, 2011. 1