Abstract

In recent years the performance of deep learning algorithms has been demonstrated in a variety of application domains. The goal of this paper is to enrich deep learning to be able to predict a set of random variables while taking into account their dependencies. Towards this goal, we propose an efficient algorithm that is able to learn structured models with non-linear functions. We demonstrate the effectiveness of our algorithm in the tasks of predicting words as well as codes from noisy images, and show that by jointly learning multilayer perceptrons and pairwise features, significant gains in performance can be obtained.

1 Introduction

Deep learning algorithms attempt to model high-level abstractions of the data using architectures composed of multiple non-linear transformations. A multiplicity of variants have been proposed \cite{10, 16, 8, 1, 22, 33} and shown to be extremely successful in a wide variety of applications including object detection, speech recognition as well as natural language processing. Recent examples show state-of-the-art results in computer vision, outperforming competitive methods by a large margin \cite{17, 25, 11, 13, 4}. Similarly state-of-the-art results have been demonstrated for natural language processing, e.g., in \cite{26, 5, 6}.

Deep neural networks can, however, be even more powerful when combined with graphical models in order to capture the statistical dependencies between the variables of interest. A few attempts to incorporate structure in deep networks have been proposed, restricted Boltzmann machines (RBMs) \cite{9, 24} and its variants \cite{22} are the most common examples. Their generative nature, however, makes training hard in general since the computation of the partition function is required.

In the context of graphical models, features are typically hand-crafted and the weights of a log-linear model are learned from data. Due to the assumed linearity only simple dependencies can be learned. Common approaches to incorporate non-linearities often rely on a two-step process where a non-linear classifier is trained first and its output is used to generate features for the structured predictor \cite{20, 32}. This, however, is suboptimal as learning the non-linear classifiers ignores the dependencies between the variables of interest.

More recently, several attempts have been presented to extend conditional random fields to learn non-linear functions. Li and Zemel \cite{18} use a hinge loss to learn the unary term defined as a neural net, but keep the pairwise potentials fixed. Conditional Neural Fields \cite{21} extend Conditional Random Fields (CRFs) by augmenting the unary feature using a ‘gate function’ which combines different features in a non-linear manner. However pairwise features remain linear. Domke \cite{3} considers non-linear structured prediction and decomposes the learning problem into a subset of logistic regressors.
In contrast, in this paper we propose an approach which generalizes structured support vector machines and conditional random fields to non-linear functions. Towards this goal, we derive an efficient algorithm which interleaves learning and inference. This generalizes the efficient algorithms of [19, 7] to non-linear functions. We demonstrate the effectiveness of our algorithm in the tasks of predicting words as well as codes from noisy images. Our experiments show that by jointly learning multilayer perceptrons and pairwise features significant gain in performance can be obtained.

2 Non-linear Structured Prediction

In this section we define non-linear structured prediction more formally and derive an efficient message passing algorithm which blends learning and inference. To this end let \( y \in \mathcal{Y} \) be the set of random variables we are interested in predicting, with \( y = (y_1, \ldots, y_N) \). In this work we assume the space of valid configurations to be a product space, i.e., \( \mathcal{Y} = \prod_{i=1}^N \mathcal{Y}_i \), and the domain of each individual variable \( y_i \) to be discrete, i.e., \( \mathcal{Y}_i = \{1, \ldots, |\mathcal{Y}_i|\} \).

Given input data \( x \in \mathcal{X} \) and parameters \( w \in \mathbb{R}^A \) of the scoring function \( F(x, y; w) : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^A \rightarrow \mathbb{R} \), inference amounts to finding the highest scoring configuration \( y^* \):

\[
y^* = \arg \max_y F(x, y; w).
\]

Log-linear models are a special case of this function, with \( F(x, y; w) = w^T \phi(x, y) \). In this setting CRFs [14], structured Support Vector Machines [27, 29] or interpolations of those [7] are typically employed to learn the linear weights \( w \).

In this work, we consider the more general setting in which \( F(x, y; w) \) may be non-linear and non-concave in \( w \). Towards this goal, we extend the structured prediction framework of [7] to non-linear functions. During learning we are interested in finding the parameters \( w \) of our model, given a training set \( \mathcal{D} \) of input-output pairs \( (x, y) \in \mathcal{D} \). We let the probability of an arbitrary configuration \( \hat{y} \) be given by the loss augmented likelihood

\[
p_{x,y}(\hat{y} | w, \epsilon) = \frac{1}{Z_{\epsilon}(x, y, w)} \exp(F(x, \hat{y}; w) + \ell(y, \hat{y}))^{1/\epsilon}.
\]

Hereby \( \ell(y, \hat{y}) : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+ \cup \{0\} \) denotes the loss function which compares a prediction \( \hat{y} \) with a ground truth label \( y \), and \( Z_{\epsilon}(x, y, w) \) refers to the partition function, normalizing the distribution \( p_{x,y} \) to lie within the probability simplex \( \Delta \) via

\[
Z(x, y, w) = \sum_{\hat{y} \in \mathcal{Y}} \exp(F(x, \hat{y}; w) + \ell(y, \hat{y}))^{1/\epsilon}.
\]

The temperature parameter \( \epsilon \geq 0 \) is used to adjust the uniformity of the distribution.

The learning task is defined as minimizing the negative log-likelihood \(-\ln p_{x,y}(y | w, \epsilon)\) of the training data \( \mathcal{D} \) w.r.t. \( w \), i.e.,

\[
\min_w \sum_{(x,y) \in \mathcal{D}} (\epsilon \ln Z_{\epsilon}(x, y, w) - F(x, y; w)).
\]

Subsequently we make use of the relationship between the scaled log-partition function \( \epsilon \ln Z_{\epsilon} \) and the maximum of the negative Gibbs-Helmholtz free energy [30, 12], i.e., expectation plus entropy \( H \), which is summarized in the following lemma (with a proof sketch given for completeness).

**Lemma 1** The log-partition function is the maximum of the negative Gibbs-Helmholtz free energy:

\[
\epsilon \ln Z_{\epsilon}(x, y, w) = \max_{p_{x,y}(\hat{y}) \in \Delta} \sum_{\hat{y} \in \mathcal{Y}} p_{x,y}(\hat{y}) \left( F(x, \hat{y}; w) + \ell(y, \hat{y}) \right) + \epsilon H(p_{x,y}).
\]

**Proof:** Either via conjugate duality or by noting that the minimum of the Kullback-Leibler divergence equals zero, i.e., \( \min_{p_{x,y}(\hat{y}) \in \Delta} D_{\text{KL}}(p_{x,y} | \frac{1}{Z_{\epsilon}} \exp(F(x, \hat{y}; w) + \ell(y, \hat{y})) = 0. \)

For many applications, our model \( F(x, y; w) \) as well as the employed loss \( \ell(y, \hat{y}) \) decompose into a sum of functions, each depending on a local subset of variables \( y_r \), i.e.,

\[
F(x, y; w) = \sum_{r \in \mathcal{R}} f_r(x, y_r; w) \quad \text{and} \quad \ell(y, \hat{y}) = \sum_{r \in \mathcal{R}} \ell(y_r, \hat{y}_r),
\]

2
where \( r \) is a restriction of the variable tuple \( y = (y_1, \ldots, y_N) \) to the subset \( r \subseteq \{1, \ldots, N\} \), more formally \( y_r = (y_i)_{i \in r} \). All subsets \( r \) required to compute the model function \( F \) and the loss \( \ell \) are summarized in the set \( \mathcal{R} \).

Plugging this decomposition into Eq. (1), we obtain the scaled log-partition function

\[
e \ln Z_e(x, y, w) = \max_{p(x,y) \in \Delta} \sum_{r,y_r} p(x,y) f_r^f(x, y_r; w) + \epsilon H(p(x,y)) \tag{3}
\]

where we have introduced marginals \( p(x,y), r(y_r) = \sum_{y|y_r} p(x,y) (y) \) and the loss augmented function \( f_r^f(x, y_r; w) = f_r(x, y_r; w) + \ell_r(y, y_r) \).

Despite the locality of both the scoring function and the loss, the learning task specified in Eq. (1) remains computationally challenging since the entropy \( H(p(x,y)) \) can only be computed exactly for a small set of models of interest. In addition, the marginalization constraints are generally exponential in size. To deal with both issues we approximate the true marginals \( p(x,y), r(y_r) \) with local beliefs \( b_i(x,y), r(y_r) \) that are not required to fulfill marginalization constraints globally, but only locally, i.e., marginals \( b_i(x,y), r(y_r) \) are not required to arise from a common joint distribution \( p(x,y) \) \[30\]. In addition, we approximate the entropy via the fractional entropy \[31\], i.e., \( H(p(x,y)) \approx \sum_r c_r H(b_i(x,y), r) \), which employs counting numbers \( c_r \) to weight the marginal entropy contributions. Putting all this together, our approximation reads as follows:

\[
e \ln Z_e(x, y, w) \approx \max_{b(x,y) \in C(x,y)} \sum_{r,y_r} b(x,y) r(y_r) f_r^f(x, y_r; w) + \sum_r \epsilon c_r H(b_i(x,y), r) \tag{3}
\]

\[
\text{s.t. } b(x,y) \in C(x,y) = \begin{cases} \forall_r y_r, p \in P(r) \sum_{y|y_r} b(x,y,p) = b(x,y,r(y_r)), \\ \forall_r y_r, p \in P(r) b(x,y,p) \approx c_r H(b_i(x,y), r) \end{cases}
\]

where \( P(r) \) refers to the set of parents of region \( r \), i.e., \( P(r) \subseteq \{ p \in \mathcal{R} : r \subseteq p \} \), which subsumes those regions for which we want the marginalization constraint to hold. Conversely, we define the set of children \( C(r) = \{ c \in \mathcal{R} : r \subseteq P(c) \} \).

Given the approximation of the scaled partition function defined in Eq. (3) we learn the parameters of the non-linear model \( w \) by solving the following program:

\[
\min_w \sum_{(x,y) \in \mathcal{D}} \left( \max_{b(x,y) \in C(x,y)} \sum_{r,y_r} b(x,y) r(y_r) f_r^f(x, y_r; w) + \sum_r \epsilon c_r H(b_i(x,y), r) \right) - F(x, y; w). \tag{4}
\]

For general counting numbers we have to solve the maximization w.r.t. beliefs \( b_i(x,y) \) exactly to obtain a subgradient w.r.t. \( w \). We can however derive a more efficient algorithm by blending minimization w.r.t. \( w \) and maximization of the beliefs \( b_i \) when all counting numbers are positive. To interleave both programs we convert maximization of the beliefs into a minimization by dualizing w.r.t. \( b_i \). This can be done since the maximization problem is concave in \( b_i(x,y) \) if \( \forall r, c_r \geq 0 \). Note that we assumed \( \epsilon \geq 0 \) throughout.

**Claim 1** Assume \( c_r \geq 0 \) \( \forall r \) and let \( \bar{F}(w) = \sum_{(x,y) \in \mathcal{D}} F(x, y; w) \) denote the sum of empirical function observations. Let \( \lambda_{(x,y), r \rightarrow p}(\hat{y}_r) \) be the Lagrange multipliers for each marginalization constraint \( \sum_{y|y_r} b(x,y), r(\hat{y}_r) = b(x,y), r(\hat{y}_r) \) within \( C(x,y) \), defined in Eq. (3). Then the approximated general structured prediction task given in Eq. (4) is equivalently given via:

\[
\min_w \sum_{(x,y) \rightarrow} \sum_{r,y_r} \epsilon c_r \ln \sum_{\hat{y}_r} \exp \frac{f_r^f(x, \hat{y}_r; w) + \sum_{c \in C(r)} \lambda_{(x,y), c \rightarrow r}(\hat{y}_c) - \sum_{p \in P(r)} \lambda_{(x,y), r \rightarrow p}(\hat{y}_r)}{\epsilon c_r} - \bar{F}(w). \tag{5}
\]

**Proof:** To obtain the dual of the maximization w.r.t. \( b_i(x,y) \) we utilize its Lagrangian

\[
L(x,y) = \sum_{r,y_r} b(x,y), r(\hat{y}_r) \left( f_r^f(x, \hat{y}_r; w) + \sum_{c \in C(r)} \lambda_{(x,y), c \rightarrow r}(\hat{y}_c) - \sum_{p \in P(r)} \lambda_{(x,y), r \rightarrow p}(\hat{y}_r) \right) + \sum_r \epsilon c_r H(b_i(x,y), r). 
\]
Maximization of the Lagrangian w.r.t. the primal variables $b$ is possible via Lemma 1 and we obtain the dual function being the first term in Eq. (5). For strict convexity, i.e., $\epsilon c_r > 0$ we reconstruct the beliefs to be proportional to the exponentiated, loss-augmented re-parameterization score

$$b_{(x,y),r}(\hat{y}_r|\hat{y}_r) \propto \exp \left( \sum_{c \in C(r)} \lambda_{(x,y),c\rightarrow r}(\hat{y}_c) - \sum_{p \in P(r)} \lambda_{(x,y),r\rightarrow p}(\hat{y}_r) \right) / \epsilon c_r$$

For $\epsilon c_r = 0$ the beliefs correspond to a uniform distribution over the set of maximizers of the loss-augmented re-parameterization score, which concludes the proof. ■

We next derive a message passing algorithm to optimize Eq. (5) using block-coordinate descent to alternate between computing the messages and performing weight vector updates. We emphasize again that this is possible only if $\epsilon c_r \geq 0 \forall r$, as in that case the Gibbs-Helmholtz free energy is convex everywhere. Strictly speaking, we require convexity only within the set of feasible beliefs $C_c^{x,y}$. Hence it is possible to relax the non-negativity restriction of $\epsilon c_r \geq 0 \forall r$ slightly. However, we neglect this extension in the following.

Fig. 1 summarizes our non-linear structured prediction algorithm. Given a fixed $w$ in the first step, we iterate through all regions $r$ and use block-coordinate descent to find the globally optimal value of Eq. (5) w.r.t. $\lambda_{(x,y),r\rightarrow p}(\hat{y}_r) \forall p \in P(r)$. This can be done jointly in closed form and therefore can be computed very efficiently. The second step employs the Armijo rule to find a stepsize along the negative gradient w.r.t. $w$ to ensure sufficient decrease.

**Claim 2** For a chosen region $r$, we let $\mu$ be defined as specified in Eq. (6) of Fig. 1. The block-coordinate descent update of $\lambda_{r\rightarrow p}(\hat{y}_r) \forall p \in P(r), \hat{y}_r$ is then analytically computed via Eq. (4) to minimize the cost function in Eq. (5) w.r.t. the chosen block of variables.

**Proof:** The proof follows. ■

The gradient of the cost function in Eq. (5) w.r.t. $w$ is given by $\sum_{(x,y),r,\hat{y}} b_{(x,y),r}(\hat{y}_r) \nabla_w f_r(x, \hat{y}_r; w) \rightarrow \nabla_w F(w)$ which follows from a straightforward computation. The convergence properties for our algorithm are summarized in the following claim.

**Claim 3** The block-coordinate descent algorithm defined in Fig. 1 is guaranteed to monotonically decrease the cost function.

**Proof:** When optimizing w.r.t. blocks of Lagrange multipliers $\lambda$ we ensure that the cost function decreases since we minimize a convex function analytically. Similarly via the Armijo rule we ensure that a descent step w.r.t. $w$ is taken, which concludes the proof. ■
In practice, Armijo line-search might be prohibitively expensive for larger models with millions of parameters. At the expense of losing guarantees for monotonic decrease, it can be replaced with subgradient steps. For non-smooth functions we found the latter to perform better in practice. Thus we employ subgradient methods for our experiments.

3 Experimental Evaluation

Throughout our experiments, we define graphical models composed of unary and pairwise regions. We experiment with different graphical models, and show that by adding more connections our approach is able to take more dependencies into account and solve the task better.

Non-linear Functions: For all models, we encode unary potentials using multi-layer perceptrons (MLPs) with hidden units defined as

\[ h(x; U) = a(U^T x), \]  

where \( a(x) = \max(0, x) \) is a rectified linear unit (ReLU). For \( k \)-layer perceptrons, our unary potentials are defined as

\[ f_r(x, y_i; w_u) = V_{y_i}^T : h^k(x; U), \]

where \( w_u = \{V, U_1, ..., U_k\} \) are the unary model parameters, with \( V_{y_i} \) the \( y_i \)-th column of \( V \) expressing the fact that the weights are class dependent, and \( h^k(x; U) \) the recursive application of Eq. (8) \( k \) times, e.g., for \( k = 2 \), \( h^2(x; U) = a(U_2^T \cdot a(U_1^T x)) \). Note that the hidden units are not equivalent to latent variables in hidden CRFs or latent structured SVMs. They are deterministic functions that can be computed from the observations \( y \) and the weights \( w \). We believe this is the reason why our approach results in a much easier learning problem than RBMs, and we can learn models with hundred thousands of weights and a large number of hidden units.

We learn the pairwise functions from scratch and thus define the potentials as

\[ f_r(x, y_i, y_j; w_p) = W_{mn} \cdot \mathbb{1}(y_i = m, y_j = n), \]

where \( r = \{i, j\}, w_p = \{W\}, W_{mn} \) is the element of \( W \), and \( \mathbb{1} \) refers to the indicator function.

Experimental protocol: For all experiments, we share all unary weights across the nodes of the graphical model as well as all pairwise weights for all edges. We investigate four strategies to learn the model parameters. ‘Unary only’ denotes training only unary classifiers and ignoring the structure of the graphical model, i.e., pairwise weights are equal to 0. ‘Ours-RandInit’ initializes all weights at random using a zero mean Gaussian with variance 0.001, and trains all weights jointly. ‘Ours-PwTrain’ uses pre-training, by first training the unary MLP potentials and then keeping them fixed when learning the pairwise potentials. ‘Ours-JointTrain’ pre-trains the unaries but jointly optimizes pairwise weights as well as unary weights in the second step. Note that due to the use of ReLU units, the learning process is non-smooth, non-linear and non-convex. Because of the non-smoothness of \( F \) we utilize momentum based sub-gradient descent methods to estimate the weights. In particular, we use a mini-batch size of 100, a step size of 0.01 and a momentum of 0.95. If the unary potential is pre-trained, the initial step size is reduced to 0.001. All the unary classifiers are trained with 100,000 iterations over mini-batches. For all experiments, the validation set is only used to decrease the step size (if the accuracy on the validation set decreases, we reduce the step size by 0.5). Throughout the experiments we use \( \epsilon = 1 \) and set \( c_r = 1 \) for all regions \( r \).
Datasets  We demonstrate the performance of our model on two tasks: word recognition as well as digit code prediction. Our first task consists of word recognition from noisy images. Towards this goal, we created challenging datasets by randomly selecting a fixed number of five character words (20 and 50 in our experiments) from the Webster’s Unabridged dictionary. Thus the graphical model has five nodes (one per character). We then generated writing variations of each word as follows. We took the lower case characters from the Latin alphabet given in the Chars74K dataset [2], and inserted them in random background image patches by alpha matting (i.e., characters have transparency). To increase the difficulty, we perturbed each character image of size $28 \times 28$ by scaling, rotation and translation. As shown in Fig. 2 the task is very challenging, some characters are fairly difficult to recognize even for humans. We denote the resulting datasets as ‘Word20’ and ‘Word50’.

We define the training, validation and test set sizes to be 10,000, 2,000 and 2,000 variations of words, respectively. For digit code recognition, we followed [15] to augment MNIST digits [16] by adding random background image patches. We combined two digits to generate the ‘Code’ dataset, in which the second digit is always larger than or equal to the digit in the first image. The graphical model for this dataset is then a chain with two nodes. The training, validation and test set sizes are 5,000, 1,000 and 25,000 codes respectively.

We begin our evaluation on the word recognition task. We experiment with four graphical models as visualized in Fig. 3 with varying degree of structure. Since we use $28 \times 28$ sized image patches as input throughout our experiments the dimensionality of the features is $A = 28^2 H_1 + H_1 |Y_i| + |Y_i|^2$ for single-layer MLPs and $A = 28^2 H_1 + H_1 H_2 + H_2 |Y_i| + |Y_i|^2$ for two-layer MLPs. Note that the $|Y_i|^2$ contribution originates from the pairwise interactions between output space variables. Our largest model thus has 677,540 weights. We report two metrics, the average character and word accuracy, which correspond to Hamming loss and zero-one loss respectively. Note that Hamming loss is lower bounded by the zero-one loss. Tab. 1 depicts the results for the different models, learning strategies and number of hidden units. We observe the following trends.

Joint training helps:  As expected joint training with pre-trained unary classifiers outperforms all other approaches in most cases. On the simpler Word20 dataset, we also observe that when the unary classifier is strong (e.g., using $H_1 = 1024$ or two-layer perceptrons), the piecewise training scheme can achieve good results. However, as the difficulty of the prediction increases (Word50), it is better to train the model jointly.

Structure helps:  Importantly we highlight that adding structure to the model is key to capture the complex dependencies of the task. We note that more structured models consistently result in improved performance for all ranges of hidden units. Especially, long-range connections improve the performance significantly, compared to using only neighboring connections (i.e., chain).

Deep helps: To further demonstrate that our model is capable of training deep models, we tested our models with two-layer perceptrons with both short-range and long-range connections. The number of hidden units in the first layer is fixed to $H_1 = 512$, and we varied the number of hidden units in the second layer. We observe that the deeper and the more structured the model, the better the performance we achieve. As expected, performance also grows with the number of hidden units.

Efficiency:  Using GPUs, it takes on average 0.176s per iteration for the chain model and 0.257s for the multi layer models is approximately the same. Note that our approach is very efficient, as this is the time per iteration to train 677,540 weights.
Table 1: Word / Character accuracy on the Word dataset generated with 20 and 50 words. Performance improves as (1) joint-training is employed, (2) the model is more structure, and (3) deeper unary classifiers are employed.

(Word50 dataset)

| Model Type   | Method       | $H_1 = 128$ | $H_1 = 256$ | $H_1 = 512$ | $H_1 = 768$ | $H_1 = 1024$ |
|--------------|--------------|-------------|-------------|-------------|-------------|-------------|
| Unary only   | 59.33 / 77.02 | 61.83 / 78.61 | 61.30 / 78.41 | 61.64 / 78.56 | 61.23 / 78.24 |
| Ours-RandInit| 65.39 / 81.96 | 68.24 / 83.44 | 68.41 / 83.66 | 68.63 / 83.64 | 68.05 / 83.36 |
| Ours-PwTrain | 66.46 / 82.70 | 68.29 / 84.22 | 68.52 / 84.14 | 68.57 / 84.02 | 68.24 / 84.14 |
| Ours-JointTrain | 67.22 / 81.03 | 69.46 / 83.49 | 69.91 / 83.16 | 68.84 / 82.91 | 68.55 / 82.01 |
| Weaker Unary  | 44.76 / 66.62 | 54.48 / 73.59 | 55.20 / 74.23 | 56.41 / 75.03 | 54.39 / 73.57 |
| Ours-PwTrain  | 58.36 / 75.55 | 65.74 / 80.11 | 65.77 / 80.08 | 65.74 / 80.13 | 66.21 / 80.41 |
| Ours-JointTrain | 65.58 / 79.94 | 68.47 / 81.88 | 68.17 / 81.70 | 68.25 / 81.70 | 68.30 / 81.76 |

Table 2: Code / Digit accuracy on the Code dataset. One-layer perceptron with chain structure is employed. In the first set of experiments (top four rows), the unary classifiers have been trained with 100,000 iterations, and joint training improvements are moderate. In the second set of experiments (bottom three rows), the improvement of using joint training becomes more significant when weaker unary classifiers are employed (same MLP but trained for 10,000 iterations).

(Word20 dataset)

| Model Type   | Method       | $H_1 = 128$ | $H_1 = 256$ | $H_1 = 512$ | $H_1 = 768$ | $H_1 = 1024$ |
|--------------|--------------|-------------|-------------|-------------|-------------|-------------|
| Unary only   | 7.35 / 60.89 | 10.95 / 65.33 | 12.35 / 66.88 | 12.80 / 66.96 | 12.30 / 67.29 |
| Ours-RandInit| 42.85 / 77.60 | 50.45 / 81.09 | 56.50 / 84.64 | 59.10 / 85.52 | 60.35 / 86.12 |
| Ours-PwTrain | 31.40 / 72.59 | 40.50 / 78.52 | 45.40 / 81.41 | 49.65 / 85.06 | 50.65 / 87.34 |
| Ours-JointTrain | 45.10 / 87.34 | 53.85 / 85.17 | 60.50 / 86.24 | 62.60 / 87.31 | 63.45 / 87.75 |
| Ours-RandInit | 47.95 / 72.74 | 55.50 / 77.54 | 49.00 / 70.78 | 52.15 / 74.59 | 54.25 / 76.68 |
| Ours-PwTrain  | 50.45 / 83.85 | 64.10 / 89.21 | 71.85 / 91.75 | 75.75 / 93.01 | 78.20 / 93.74 |
| Ours-JointTrain | 60.25 / 81.08 | 69.60 / 86.14 | 75.25 / 89.11 | 78.00 / 90.80 | 78.50 / 91.21 |
| Unary only   | 14.30 / 69.15 | 16.55 / 70.96 | 18.50 / 72.11 | 17.35 / 71.70 | 18.80 / 72.24 |
| Ours-RandInit| 64.65 / 88.52 | 68.75 / 89.75 | 71.10 / 90.82 | 70.00 / 90.19 | 69.20 / 90.28 |
| Ours-PwTrain  | 63.60 / 88.99 | 66.40 / 90.02 | 69.95 / 91.02 | 71.00 / 91.43 | 71.60 / 91.92 |
| Ours-JointTrain | 70.30 / 90.57 | 71.25 / 91.13 | 72.95 / 91.77 | 73.10 / 91.69 | 72.25 / 91.22 |
| Ours-RandInit | 90.45 / 96.43 | 89.10 / 96.13 | 90.05 / 96.65 | 89.80 / 96.77 | 90.05 / 96.84 |
| Ours-PwTrain  | 90.30 / 97.20 | 92.35 / 97.82 | 93.70 / 98.22 | 93.75 / 98.25 | 94.45 / 98.49 |
| Ours-JointTrain | 90.95 / 96.43 | 91.30 / 97.18 | 91.65 / 97.28 | 91.15 / 97.30 | 91.30 / 97.45 |

As shown in the left column of Fig. 2, the learned unary filters resemble character strokes. Furthermore, the learned edge weights in Word50 are more complex than those in Word20. On the right-most panels, we also show the training energy vs. the number of iterations. Ours-JointTrain can achieve the lowest energy, while Ours-PwTrain has the highest energy. We refer the reader to the supplementary material for examples of both good predictions and failure cases.

For the Code dataset, we experiment with one-layer perceptrons with chain structure, as the graphical model has only two variables. As shown in Tab. 2, Ours-JointTrain consistently outperforms the
other strategies. However, the improvement over Ours-PwTrain decreases as the unary classifiers get stronger (i.e., when more hidden units are employed). To explore the effect of pre-trained unary classifiers, we combine our models with weaker unary classifiers, which are trained with only 10,000 iterations (instead of 100,000, as the case in all previous experiments). The bottom three rows of Tab. 2 show that with this weaker pre-trained unary classifiers, Ours-PwTrain is outperformed significantly by Ours-JointTrain. This highlights the advantage of jointly fine-tuning both unary and pairwise parameters during training when the pre-trained unary classifiers are weak. Fig. 5 shows the unary and pairwise weights as well as the training energy as a function of the number of iterations. As shown in the middle column, our approach is able to learn the constraints imposed when creating the dataset.

4 Conclusions

We have proposed an efficient algorithm to learned deep models enriched to capture the dependencies between the output variables we are interested in predicting. Our experiments on word and code prediction from noisy images showed that the deeper and the more structured the model, the better the performance we achieve. Furthermore joint learning of all weights outperforms all other strategies. In the future we plan to learn deeper models for object recognition in applications such as holistic semantic scene understanding.
References

[1] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy Layer-Wise Training of Deep Networks. In Proc. NIPS, 2007.
[2] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In VISAPP, 2009.
[3] J. Domke. Structured learning via logistic regression. In Proc. NIPS, 2013.
[4] D. Eigen, J. Rolfe, R. Fergus, and Y. LeCun. Understanding Deep Architectures using a Recursive Convolutional Network. In Proc. ICLR, 2014.
[5] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for improved unconstrained handwriting recognition. PAMI, 2009.
[6] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In Proc. ICASSP, 2013.
[7] T. Hazan and R. Urtasun. A Primal-Dual Message-Passing Algorithm for Approximated Large Scale Structured Prediction. In Proc. NIPS, 2010.
[8] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 2006.
[9] G. E. Hinton and T. J. Sejnowski. Learning and relearning in boltzmann machines. Microstructure of Cognition, 1986.
[10] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley. Boltzmann Machines: Constraint Satisfaction Networks that Learn. Technical report, University of Toronto, 1984.
[11] Y. Jia. Caffe: An Open Source Convolutional Architecture for Fast Feature Embedding. http://caffe.berkeleyvision.org/, 2013.
[12] D. Koller and N. Friedman. Probabilistic Graphical Models: Principles and Techniques. MIT Press, 2009.
[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Proc. NIPS, 2013.
[14] J. Lafferty, A. McCallum, and F. Pereira. Conditional Random Fields: Probabilistic Models for segmenting and labeling sequence data. In Proc. ICML, 2001.
[15] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In Proc. ICML, 2007.
[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In Proc. IEEE, 1998.
[17] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Proc. ICML, 2009.
[18] Y. Li and R. Zemel. High Order Regularization for Semi-Supervised Learning of Structured Output Problems. In Proc. ICML, 2014.
[19] O. Meshi, D. Sontag, T. Jaakkola, and A. Globerson. Learning Efficiently with Approximate inference via Dual Losses. In Proc. ICML, 2010.
[20] S. Nowozin, C. Rother, S. Bagon, T. Sharp, B. Yao, and P. Kohli. Decision tree fields. In Proc. ICCV, 2011.
[21] J. Peng, L. Bo, and J. Xu. Conditional Neural Fields. In Proc. NIPS, 2009.
[22] R. R. Salakhutdinov and G. E. Hinton. An Efficient Learning Procedure for Deep Boltzmann Machines. Neural Computation, 2012.
[23] A. G. Schwing. Inference and Learning Algorithms with Applications to 3D Indoor Scene Understanding. PhD thesis, ETH Zurich, 2013.
[24] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Microstructure of Cognition, 1986.
[25] R. Socher, B. Hsuval, B. Bhat, C. D. Manning, and A. Y. Ng. Convolutional-Recursive Deep Learning for 3D Object Classification. In Proc. NIPS, 2012.
[26] R. Socher, C. C.-Y. Lin, A. Y. Ng, and C. D. Manning. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In Proc. ICML, 2011.
[27] B. Taskar, C. Guestrin, and D. Koller. Max-Margin Markov Networks. In Proc. NIPS, 2003.
[28] P. Tseng and D. P. Bertsekas. Relaxation Methods for Problems with Strictly Convex Separable Costs and Linear Constraints. Mathematical Programming, 1987.
[29] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large Margin Methods for Structured and Interdependent Output Variables. *JMLR*, 2005.

[30] M. J. Wainwright and M. I. Jordan. *Graphical Models, Exponential Families and Variational Inference*. Foundations and Trends in Machine Learning, 2008.

[31] W. Wiegerinck and T. Heskes. Fractional belief propagation. In *Proc. NIPS*, 2003.

[32] J. Yao, S. Fidler, and R. Urtasun. Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation. In *Proc. CVPR*, 2012.

[33] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. *CoRR*, 2013.