FACTORIZED CRF WITH BATCH NORMALIZATION BASED ON THE ENTIRE TRAINING DATA

Eran Goldman *Jacob Goldberger*

Faculty of Engineering, Bar-Ilan University, Ramat-Gan, Israel

ABSTRACT

Batch normalization (BN) is a key component of most neural network architectures. A major weakness of Batch Normalization is its critical dependence on having a reasonably large batch size, due to the inherent approximation of estimating the mean and variance with a single batch of data. Another weakness is the difficulty of applying BN in autoregressive or structured models. In this study we show that it is feasible to calculate the mean and variance using the entire training dataset instead of standard BN for any network node obtained as a linear function of the input features. We dub this method Full Batch Normalization (FBN). Our main focus is on a factorized autoregressive CRF model where we show that FBN is applicable, and allows for the integration of BN into the linear-chain CRF likelihood. The improved performance of FBN is illustrated on the huge publicly available SKU dataset that contains images of retail store product displays.

Index Terms— CRF, Batch normalization, FBN

1. INTRODUCTION

Batch normalization (BN) [1] is a technique for improving the speed, performance and stability of artificial neural networks. BN also makes it possible to use significantly higher learning rates, and reduces the sensitivity to initialization. It has become a part of the standard toolkit for training deep networks and has been widely used in various areas such as machine vision [1, 2], speech [3] and natural language processing [4]. BN stabilizes the distributions of layer inputs by introducing additional network layers that control the first two moments (mean and variance) of these distributions. It can be inserted extensively into a network, either between a linear mapping and a nonlinearity or after the non-linear activation [2, 1, 5]. Normalization techniques typically make neural networks more amenable to optimization, by allowing the training of very deep networks without the use of careful initialization schemes [6, 7]. While BN's original motivation was to reduce internal covariate shift during training [1], recent work has proposed instead that its effectiveness stems from making the optimization landscape smoother [8].

A major weakness of BN is its critical dependence on having a reasonably large batch size, due to the inherent approximation of estimating the mean and variance with a single batch of data. When trained with small batch sizes, BN exhibits a significant degradation in performance [9, 10, 11]. Several approaches have addressed the issues encountered by BN, by not relying on the stochastic minibatch altogether (e.g. layer [12], instance [13], weight [14] and group normalization [11]). Instead, they collect various types of statistics from the current instance alone, i.e., from each individual element in the minibatch separately. These methods have a much smaller computational cost but typically achieve inferior results compared to BN [15].

Conditional Random Field (CRF) [16] is a popular probabilistic graphical model that was originally applied to language processing tasks [16]. The CRF pairwise potential matrix can be forced to be low-rank as a form of regularization [17, 18, 19]. This causes the training objective function to be non-convex and therefore more difficult to optimize. The samples of linear-chain CRF are sequences of inputs and their corresponding sequences of outputs. The likelihood of the model is an autoregressive function of a sequence, where the model weights are shared across multiple steps. Hence, BN cannot be applied directly to learn minibatch statistics. A recent study [20] proposed an approximated CRF likelihood which allows for integrating BN in a simple and effective way that shows significant improvement in performance.

In this study we present a variant of batch normalization that can be combined into the *exact* likelihood function of low-rank linear-chain CRF models. We can efficiently and accurately compute BN statistics from the whole training data, while still using minibatches for training. This method is directly applicable to any layer whose output is a linear function of the model input.

We show here that in the case of linear-chain CRF with a low-rank pairwise potential matrix, it is feasible to compute the batch-normalization statistics over all the training data even though we use minibatches to compute the gradients. Furthermore, when BN is computed this way, it can in fact be integrated in the original CRF likelihood. The improved performance of the proposed method is demonstrated on a huge publicly available dataset that contains images of retail store product displays, taken in varying settings and viewpoints [20]. We show that we achieve better classification results when applying our method on the approximated

CRF likelihood, and further improve the performance by a wide margin when training the exact CRF likelihood.

2. CRF WITH FACTORIZED PAIRWISE POTENTIAL MATRIX

In this section we describe the factorized CRF model and explain the challenges we face when trying to use BN in the training phase. We are given a sequence of observations and the goal is to classify each item in the sequence to one of k predefined classes (see example in Fig. 1). The Conditional Random Field (CRF) [16] model has conditional distribution $p(\mathbf{y}|\mathbf{x})$ that obeys a conditional Markov property. The probability distribution of a linear-chain CRF is:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_{t=1}^{n}\exp(y_{t-1}^{\top}Py_t + x_t^{\top}Uy_t + b^{\top}y_t) \quad (1)$$

where $\mathbf{x} = (x_1, ..., x_n)$ is the input sequence, $\mathbf{y} = (y_1, ..., y_n)$ is the corresponding sequence of the target labels and Z is the partition function defined as the global probability normalization over all possible sequence label-assignments of length n. The CRF model parameters are P, U and b where P is a $k \times k$ pairwise potential matrix that models the relationship between labels of consecutive items, U is a unary potential matrix modeling the information in the input features and the vector b is the label bias. Note that we use a one-hot encoding for the labels.

In case the number of classes is large, the pairwise potential matrix P has many parameters. In order to properly learn and generalize the massive variety of possible neighboring patterns, we can enforce a low-dimensional structure on the large and sparse matrix P. We can thus assume that $P = R^{\top}Q$ where the dimensions of R and Q are $d \times k$ s.t. $d \ll k$. Substituting $P = R^{\top}Q$ in the CRF function (1) we get a factorized CRF model:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_{t=1}^{n}\exp((Ry_{t-1})^{\top}Qy_t + x_t^{\top}Uy_t + b^{\top}y_t). \quad (2)$$

The columns of Q can be viewed as low-dimensional embeddings of the right-side classes y_t, and the columns of R are embeddings of the classes of the left-side object y_{t-1}. As a byproduct of the matrix factorization, the computation complexity of the dynamic programming inference algorithms (either Viterbi or forward-backward [21]) is reduced from $\mathcal{O}(nk^2)$ to $\mathcal{O}(nkd)$.

We can easily normalize the coordinates of input vector x_t in a pre-processing step to have zero mean and unit variance. However, the pairwise component (Ry_{t-1}) depends on the model parameters and hence its normalization cannot be done before training. It is not trivial to normalize this term during training since the parameters are shared across all the instances in the sequences. The CRF gradient is computed by a dynamic programming procedure [21] which cannot be carried out if BN simultaneously uses all the time steps. A similar situation occurs in LSTM where it was suggested to learn the BN statistics in each time step independently [22]. Another solution is the approximated CRF likelihood [20] which learns the CRF parameters by optimizing a pairwise surrogate likelihood:

Fig. 1: Examples of an input sequence $\mathbf{x}$ from the SKU retail dataset [20].

$$p(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{n} p(y_t|x_t, y_{t-1}) \quad (3)$$

where the normalization is done for each time-step separately.

Minibatches are composed here of adjacent pairs $\langle x_t, y_{t-1}\rangle$ randomly taken from the training sequences. The minibatch instances in this case are ordered pairs of x_t the input instance at step t and its neighboring label y_{t-1}. The likelihood takes the form of a regular instance classifier and standard BN statistics can easily be collected from the instances in the minibatch. It was shown that optimizing (3) with BN yields better performance than directly optimizing the exact CRF without BN (2) [20].

3. NORMALIZATION BASED ON THE WHOLE TRAINING DATA

In this section we show that in case of factorized CRF, where BN is applied to a linear function of the input Ry_{t-1} (2), it is feasible to compute the BN statistics based on the entire training dataset even though the optimization is carried out in minibatches.

Given a particular node in the deep network, producing a scalar value z for each input example, the minibatch values of the node are $z_1, ..., z_m$ where m is the minibatch size. The BN is defined as $\hat{z}_i \leftarrow \frac{z_i - \mu}{\sigma}$ where μ and σ^2, the empirical mean and variance of z, are calculated using the current minibatch. Assume now that z is a neuron in the first hidden layer and BN is applied between the linear and the non-linear transformations, i.e. $\varphi(BN(z)) = \varphi(BN(w^{\top}x))$ s.t. x is the input feature vector. The first two moments of the input, $\mu = \mathrm{E}(x)$ and $C = \mathrm{Var}(x)$, can be computed in a pre-processing step using the entire training dataset. Even unlabeled data can be used here since these statistics do not depend on the label values. Applying BN z based on the whole data we obtain:

$$\mathrm{FBN}(z) = \frac{z - \mathrm{E}z}{\sqrt{\mathrm{Var}(z)}} = \frac{w^{\top}(x - \mu)}{\sqrt{w^{\top}Cw}}. \quad (4)$$

We denote the BN procedure based on statistics collected from the whole training set *Full Batch Normalization* (FBN). Although the FBN is based on the whole training set, the computation of an FBN layer only requires the current input sample x. As a byproduct, it makes the computational graph implementation of the back-propagation much simpler than standard BN, and reduces time and memory consumption. There are BN alternatives based on a single instance [13, 12, 11]. However, they are typically inferior to standard BN if the batch size is large enough [15]. Here we take the largest batch size we can have – the whole training data. This makes the collected statistics much more accurate and stable. In the test phase, instead of using a moving average of minibatch means and variances, we can use the training data mean and variance. We thus avoid the train-test discrepancy that exists in standard batch normalization [9]. In addition, FBN decouples the batch-size parameter used for stochastic gradient optimization from the batch-size parameter used for normalization, a constraint in BN which may cause sub-optimal convergence [23]. Moreover, FBN can overcome the difficulties many tasks encounter when they cannot increase their batch size due to memory constraints (e.g. high-resolution image analysis or manipulation) and therefore cannot effectively use standard BN [11, 24].

We next apply FBN to the task of learning a factorized CRF. Let $\mathbf{p} = (p_1, ..., p_k)^\top$ be the train-set object class statistics. Let y be a one-hot representation of a label of a single object. The mean and variance of y are $\mathbf{p}$ and $\mathrm{diag}(p_1, ..., p_k) - \mathbf{p}\mathbf{p}^\top$ respectively where $\mathrm{diag}()$ is a $k \times k$ diagonal matrix. Let $R_i = (r_{i1}, ..., r_{ik})$ be the i-th row of the factorized CRF parameter R. The FBN operation of $R_i y$, where the normalization is computed on the entire training set is:

$$\mathrm{FBN}(R_i y) = \frac{R_i y - \mathrm{E}(R_i y)}{\sqrt{\mathrm{Var}(R_i y)}} = \frac{R_i(y - \mathbf{p})}{\sqrt{\sum_{j=1}^{k}(r_{ij}^2 p_j) - (R_i \mathbf{p})^2}}. \tag{5}$$

The label statistics $\mathbf{p}$ can be computed from the training data in a pre-processing step and is fixed during the training procedure. Hence, although the statistics of the FBN operation (5) are computed on the entire training set, during the optimization they are only a function of the current instance label y (and the model parameter R). We can thus replace minibatch based statistics in (3) by the statistics of the entire training set. In addition to faster convergence and improved performance the optimization is much simpler since FBN computation only requires the current instance. Furthermore, we can now easily integrate FBN into the of the exact factorized CRF objective:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_{t=1}^{n} \exp(\mathrm{FBN}(Ry_{t-1})^\top Q y_t + x_t^\top U y_t + b^\top y_t). \tag{6}$$

Table 1: FBN for CRF with normalized deep class embedding algorithm

Training data: A data sequence $\mathbf{x}$ with corresponding label sequences $\mathbf{y}$.

Optimize the likelihood function:

$$\mathcal{L}(R, Q, U, b) = p(\mathbf{y}|\mathbf{x}) =$$

$$\frac{1}{Z}\prod_{t=1}^{n} \exp(\mathrm{FBN}(Ry_{t-1})^\top Q y_t + x_t^\top U y_t + b^\top y_t)$$

$$\text{s.t. } \mathrm{FBN}(R_i y) = \frac{R_i(y - \mathbf{p})}{\sqrt{\sum_{j=1}^{k}(r_{ij}^2 p_j) - (R_i \mathbf{p})^2}}.$$

and $\mathbf{p} = (p_1, ..., p_k)^\top$ is the train-set object class statistics.

4. EXPERIMENTAL RESULTS

In this section we describe the principal experiments conducted in this study that enable full batch normalization in CRF training. We applied FBN on the large-scale SKU dataset and obtained substantial improvement in performance. Due to paper length restrictions, full implementation details and additional experiments will be published separately.

4.1. SKU Dataset

The SKU dataset was published by [20] and contains sequences of fixed-size image patches, originated from photos of retail store displays taken on-site. The objects are the inventory items positioned at the front of the displays, and the classes are their stock-keeping-unit (SKU) unique identifiers. Each object was originally annotated by its class label, while the object bounding-box coordinates were either manually annotated or extracted by a suitable detector [25]. The image patches were cropped and reshaped into single-object images of size 150×450 pixels, and grouped into shelves; i.e., sequences of horizontal layouts, arranged from left to right. Each image-patch x_t was fed into a pre-trained ResNet50 CNN [2] to acquire the feature vector $h_t = h(x_t)$ – the activations of the last hidden Global Average Pooling (GAP) layer [26, 27], and was later standardized element-wise.

The benchmark contains 76,081 sequences of 460,121 single-object images, originated from 24,024 photos of store displays. Each object is labeled as one of 972 different classes. Sequence lengths vary from 2 to 32, and are typically between 4-12. The average sequence length is 6 with a SD of 2.4.

The model was trained using a single Tesla V100 GPU. The time to train a single batch was 0.15 seconds, and the training typically converged after 15-20 epochs (˜90 minutes). At test time we ran the forward-backward algorithm [21] to

Table 2: Classification results on SKU dataset [20].

Training Method	Accuracy	AP	$R^{P=0.75}$	$R^{P=0.95}$
Log-linear CRF	0.856	0.904	0.898	0.644
Factorized CRF [20]	0.883	0.938	0.935	0.780
Approximate CRF + BN [20]	0.878	0.931	0.937	0.756
Approximated CRF + FBN (ours)	0.889	0.938	0.943	0.783
Factorized CRF + LN [12]	0.899	0.944	0.949	0.833
Factorized CRF + FBN (ours)	**0.904**	**0.953**	**0.954**	**0.852**

extract object marginal probabilities, which took less than 0.1 seconds to classify all the objects in a single image.

4.2. Compared methods

First, we reproduced the following experiments described in [20]: *(a) Log-linear CRF*: learning the log-linear parameters of the linear-chain CRF (1). *(b) Factorized CRF*: learning a linear-chain CRF with the factorized parameters of the pairwise weight matrix as defined in Eq. (2). *(c) Approximate Factorized CRF + BN*: the previous state-of-the-art model proposed by [20] where the CRF pairwise weight matrix was factorized, and the network was trained by the surrogate likelihood (3), with batch-normalization for the embedding features in the matrix factor R.

Next, we performed the following experiments: *(d) Approximate Factorized CRF + FBN*: This time we only replaced the BN with FBN whereas the remainder of the model parameters remained identical to the previous experiment. This change alone improved performance by a considerable margin. *(e) Factorized CRF + FBN*: BN cannot be applied directly to CRF because the minibatch statistics are shared across various nodes in the sample, and are also part of the complex sequence-level normalization factor. However, FBN does not require computing any minibatch statistics and hence can easily be incorporated into the exact CRF formula. We trained the likelihood defined in Eq. (6).

As described earlier, BN is mostly preferred when applicable and works better with large batches. FBN provides an accurate batch normalization of the largest batch size possible. However, it is noteworthy to consider other popular normalization methods. Instance Normalization (IN) [13] and Group Normalization (GN) [11]) are designed for multi-channel CNNs and are therefore irrelevant for our case of normalizing the class embedding matrix. *Layer Normalization (LN)* [12, 28] normalizes the inputs across the features rather than the batch, and is often used in recurrent models. Given the current embedding vector $z \equiv Ry_{t-1}$ the normalization formula is $LN(z) = \frac{z-\mu}{\sigma}$, where μ and σ^2 are the empirical mean and variance of z instance-level features. In our case, layer normalization in fact helped in avoiding numerical instability issues. Unsurprisingly, however, it was unable to perform as well as FBN since the LN formulation is very different from the desired BN operation.

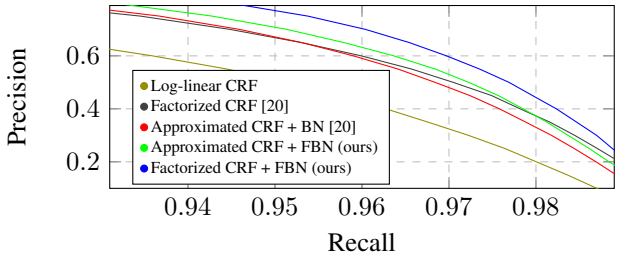

Fig. 2: Precision-Recall curves for different training methods.

4.3. Quantitative results

Table 2 lists the test results in terms of Accuracy, Average Precision (AP), and the recall values attained for precision of 0.75 ($R^{P=0.75}$) and 0.95 ($R^{P=0.95}$). Figure 2 depicts the Precision-Recall curve. FBN, when applied to the approximate method proposed by [20] instead of standard BN (Approximated CRF + FBN), improves the accuracy by more than 1%. This improvement is even more evident for the major objective for this dataset of maximizing recall while preserving high precision. The method that used FBN obtained 3% higher recall while preserving 95% precision.

In our *full method (Factorized CRF + FBN)*, we trained the exact CRF likelihood with FBN (6). The accuracy over the former state-of-the-art method [20] increased by more than 2%. and the recall attained for 95% precision increased by almost **10%** compared to [20] and more than **20%** compared to the standard CRF. It can be seen that our full method is particularly useful for very high recall values by contrast to the approximate alternatives (Figure 2). Our full method also outperforms the other BN alternatives by a considerable margin. Note that the test set is very large and thus the results are statistically significant as every minor improvement in accuracy rate corresponds to a large number of additional correctly classified objects. For instance, we increased the number of correct classifications by more than 9,200 objects over the former SOTA [20] while attaining 95% precision.

To conclude, BN is motivated by the well-known fact that whitening inputs stabilizes the training procedure. Standard BN implementations extract the BN statistics from the current minibatch. In this work, we proposed a BN procedure in which the data statistics can be computed using all the data available in the training phase. We thus sever the ties between normalization and minibatch. The proposed FBN algorithm yields a more accurate normalization than BN at a reduced complexity of the computational graph since only the current instance is explicitly involved in the FBN operation. The improved performance of FBN was demonstrated on learning the parameters of a factorized CRF model. A possible future research direction would involve extending the applicability of FBN to intermediate hidden layers as well.

5. REFERENCES

[1] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[3] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al., "Deep speech 2: End-to-end speech recognition in english and mandarin," in *ICML*, 2016.

[4] Cesar Laurent, Gabriel Pereyra, Philemon Brakel, Ying Zhang, and Yoshua Bengio, "Batch normalized recurrent neural networks," in *ICASSP*, 2016.

[5] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017.

[6] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[7] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma, "Fixup initialization: Residual learning without normalization," in *ICLR*, 2019.

[8] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry, "How does batch normalization help optimization?," in *NeurIPS*, 2018.

[9] Sergey Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," in *NeurIPS*, 2017.

[10] Saurabh Singh and Abhinav Shrivastava, "EvalNorm: Estimating batch normalization statistics for evaluation," in *ICCV*, 2019.

[11] Yuxin Wu and Kaiming He, "Group normalization," in *CVPR*, 2018.

[12] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[13] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[14] Tim Salimans and Diederik P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *NeurIPS*, 2016.

[15] Igor Gitman and Boris Ginsburg, "Comparison of batch normalization and weight normalization algorithms for the large-scale image classification," *arXiv preprint arXiv:1709.08145*, 2017.

[16] John Lafferty, Andrew McCallum, and Fernando CN Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *ICML*, 2001.

[17] Siddhartha Chandra, Nicolas Usunier, and Iasonas Kokkinos, "Dense and low-rank gaussian CRFs using deep embeddings," in *ICCV*, 2017.

[18] Peng Wang, Chunhua Shen, and Anton van den Hengel, "Efficient SDP inference for fully-connected CRFs based on low-rank decomposition," in *CVPR*, 2015.

[19] Greg Durrett and Dan Klein, "Neural CRF parsing," in *ACL*, 2015.

[20] Eran Goldman and Jacob Goldberger, "CRF with deep class embedding for large scale classification," *CVIU*, vol. 191, pp. 102865, 2020.

[21] Charles Sutton and Andrew McCallum, "An introduction to conditional random fields for relational learning," *Introduction to statistical relational learning*, pp. 93–128, 2006.

[22] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville, "Recurrent batch normalization," in *ICLR*, 2017.

[23] Xiangru Lian and Ji Liu, "Revisit batch normalization: New understanding and refinement via composition optimization," in *AISTATS*, 2019.

[24] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur, "A learned representation for artistic style," *ICLR*, 2017.

[25] Eran Goldman, Roei Herzig, Aviv Eisenschtat, Jacob Goldberger, and Tal Hassner, "Precise detection in densely packed scenes," in *CVPR*, 2019.

[26] Yoshua Bengio, Aaron Courville, and Pascal Vincent, "Representation learning: A review and new perspectives," *TPAMI*, vol. 35, no. 8, pp. 1798–1828, 2013.

[27] Min Lin, Qiang Chen, and Shuicheng Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[28] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin, "Understanding and improving layer normalization," in *NeurIPS*, 2019.