Discriminative Hierarchical Rank Pooling for Activity Recognition

Basura Fernando, Peter Anderson, Marcus Hutter, Stephen Gould
The Australian National University
Canberra, Australia
firstname.lastname@anu.edu.au

Abstract

We present hierarchical rank pooling*, a video sequence encoding method for activity recognition. It consists of a network of rank pooling functions which captures the dynamics of rich convolutional neural network features within a video sequence. By stacking non-linear feature functions and rank pooling over one another, we obtain a high capacity dynamic encoding mechanism, which is used for action recognition. We present a method for jointly learning the video representation and activity classifier parameters. Our method obtains state-of-the art results on three important activity recognition benchmarks: 76.7% on Hollywood2, 66.9% on HMDB51 and, 91.4% on UCF101.*

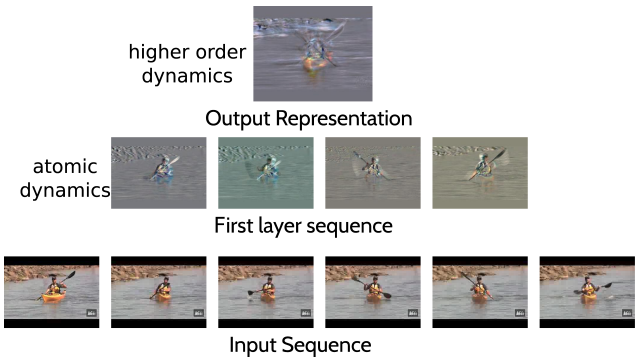


Figure 1: Illustration of hierarchical rank pooling for encoding the temporal dynamics of a video sequence.

1. Introduction

Activity and action recognition in video is important for many real-life applications including human-computer interaction, sports analytic, elderly-care, and healthcare. In action recognition a key challenge is to extract and represent high-level motion patterns, dynamics, and evolution of appearance of videos. Recently, some success has been gained by exploiting temporal information [1, 4, 5, 13, 28, 36]. Some methods use linear ranking machines to capture first order dynamics [4]. Other methods encode temporal information using RNN-LSTMs on video sequences [28, 36, 37]. To further advance activity recognition it is beneficial to exploit temporal information at multiple levels of granularity in a hierarchical manner thereby capturing more complex dynamics of the input sequences [3, 16, 26]. As frame based features improve, *e.g.*, from a convolutional neural network (CNN), it is important to exploit information not only in the spatial domain but also in the temporal domain. Several recent methods have obtained significant improvements in image categorisation and object detection using very deep CNN architectures [25]. Motivated by these deep hierarchies [3, 16, 26, 25], we argue that learning a temporal encoding at a single level is not sufficient to interpret and understand video sequences, and that a temporal hierarchy is needed.

The recently introduced rank pooling method [4, 5] managed to obtain good activity recognition performance using hand-crafted features. Given a sequence of video frames, the rank pooling method returns a vector of parameters encoding the dynamics of that sequence. The vector of parameters is derived from the solution of a linear ranking SVM optimization problem applied to the entire video sequence, *i.e.*, at a single level. As richer features are used to describe the input frames (*e.g.*, CNN features) and more complex video sequences are considered, a single level may no longer be sufficient for good performance. Furthermore, the capacity of linear ranking employed by Fernando *et al*. [4, 5] is limited and the rank pooling representation may not be discriminative for the task at hand.

To overcome these limitations, we propose to encode a video sequence at multiple levels. First, we divide the original video into multiple overlapping video segments. At the lowest level, we encode each video segment using rank pooling. The resulting encoding captures the dynamics of small video segments (see Figure 1). By recursively applying rank pooling on the obtained segment descriptors, we capture higher-order, non-linear, and complex dynamics. The output of the each encoding level is itself another sequence of vectors which captures higher-order dynamics

of previous levels. We repeat this process to obtain more complex and expressive representations at the higher levels of the hierarchy. The final representation of the video is obtained by encoding the top-level dynamic sequence using rank pooling. This strategy allows us to encode more complicated activities thanks to the higher capacity of the model. Moreover, we introduce a non-linear feature transform that we apply at each level before the rank pooling function. We jointly learn the parameters of the non-linear transformation and the activity classifier.

Our hierarchical rank pooling consists of a feed forward network of non-linear operations and rank pooling operations as illustrated in Figure 2. At the final layers we propagate back the errors to learn both video representation and the classifier. We claim that it is possible to capture a large number of temporal dynamics using two or more level of temporal encoding. In short, our main contributions are: (1) a novel temporal encoding method called *hierarchical rank pooling*, and (2) joint learning of the rank pooling based discriminative video representation and classifier. Our proposed method is useful for encoding dynamically evolving frame-based CNN features, and we are able to show significant improvements over other effective temporal encoding methods.

2. Related Work

In the literature, temporal information of video sequences is encoded using different techniques. Fisher encoding [22] of spatial temporal features is commonly used in prior state-of-the-art works [33] while Jain *et al*. [9] used VLAD encoding [10] for action recognition over motion descriptors. Temporal max pooling and sum pooling are used with bag-of-features [32] as well as CNN features [23]. Temporal fusion methods such as late fusion or early fusion are used in [13] in the context of CNN architectures. Temporal information can also be encoded using 3D convolution operators [11, 30]. However, as recently demonstrated in [30], such approaches need to rely on very large video collections to learn meaningful 3D-representations. However, it is not clear how these methods can capture long-term vide dynamics as 3D convolutions are applied only on short video clips. Rank pooling is also used for temporal encoding at representation level [4, 5] or at image level leading to dynamic images [1].

Recently, recurrent neural networks are getting popular and used extensively for sequence encoding, sequence generation and sequence classification [8, 29]. Long-short term memory (LSTM) based approaches may use the hidden state of the encoder as a video representation [28]. A CNN feature based LSTM model for action recognition is presented in [36]. Typically, recurrent neural networks are trained in a probabilistic manner to maximize the likelihood of generating the next element of the sequence. Our hierarchical rank pooling method does not rely on very large number of training samples as in recurrent neural networks. Moreover, our method has a clear objective in capturing dynamics of sequences independent of other sequences and has the capacity to capture complex dynamic signals.

Hierarchical methods have also been used in activity recognition [3, 26]. A CRF-based hierarchical sequence summarization method is presented in [26]; a hierarchical recurrent neural network for skeleton based action recognition is presented in [3]; and a hierarchical action proposal based mid-level representation is presented in [16]. However, our method captures mid-level dynamics as well as dynamics of the entire video using hierarchical rank pooling principle which is suited for rich feature representations such as deep CNN features.

3. Background

In this section we introduce the notation used in this paper and provide background on the *rank pooling* method [4], which our work extends. Let $\mathcal{X} = \{X^1, X^2, \dots, X^n\}$ be a set of n training videos, each belonging to one of K discrete classes. We represent each video by a variable-length sequence of J^i vectors so that $X^i = \langle \boldsymbol{x}_1^i, \boldsymbol{x}_2^i, \dots, \boldsymbol{x}_J^i \rangle$ with order constraints $\boldsymbol{x}_1^i \prec \boldsymbol{x}_2^i \prec \cdots \prec \boldsymbol{x}_t^i \prec \cdots \prec \boldsymbol{x}_J^i$ enforced by the chronology. A compact video representation is needed to classify a variable-length video sequence X into one of the K classes. As such, we define a temporal encoding function $\phi : X \mapsto \boldsymbol{u}$, which maps the video sequence X (or sub-sequence thereof) into a fixed-length feature vector, $\boldsymbol{u} \in \mathbb{R}^D$. Standard supervised classification techniques learned on the set of training videos $\mathcal{X}$ can then be applied to these vectors.

The goal of temporal encoding is to encapsulate valuable dynamic information in X into a single D-dimensional vector $\phi(X)$. In the rank pooling method of Fernando *et al*. [4], the sequence encoder $\phi(\cdot)$ captures time varying information of the entire sequence using a single linear surrogate function f parametrised by $\boldsymbol{u} \in \mathbb{R}^D$. The function f ranks frames of the video $X = \langle \boldsymbol{x}_1, \dots, \boldsymbol{x}_J \rangle$ based on the chronology. Ideally, the ranking function satisfies the constraint,

$$f(\boldsymbol{x}_{t_a}) < f(\boldsymbol{x}_{t_b}) \iff t_a < t_b \tag{1}$$

That is, the ranking function should learn to order frames chronologically. In the linear case this boils down to finding a parameter vector $\boldsymbol{u}$ such that $f(\boldsymbol{x}; \boldsymbol{u}) = \boldsymbol{u}^T \boldsymbol{x}$ satisfies Equation 1. In Fernando *et al*. [4] this is done by training a linear ranking machine such as RankSVM [12] on X. They then use the learned parameters of RankSVM, *i.e*., $\boldsymbol{u}$, as the temporal encoding of the video. Since the ranking function encapsulates ordering information and the parameters lie in the same feature space, the ranking function captures the *evolution of information* in the sequence X.

To include non-linearity in the ranking function, Fernando *et al*. [4] use non-linear feature maps such as signed square root [22] or chi-square feature maps [31]. And to obtain better generalizability, they also smooth the video signal by Time Varying Mean (TVM) vectors. Computing the TVM vectors from the sequence X involves taking the mean of the signal at each time step t and is given by

$$\bar{\boldsymbol{x}}_t = \frac{1}{t}\sum_{\tau=1}^{t} \boldsymbol{x}_\tau \tag{2}$$

Then instead of learning dynamics of the original sequence $\boldsymbol{x}_1 \prec \boldsymbol{x}_2 \prec \cdots \prec \boldsymbol{x}_J$, Fernando *et al*. [4] observed empirically better performance using the TVM sequence $\bar{\boldsymbol{x}}_1 \prec \bar{\boldsymbol{x}}_2 \prec \cdots \prec \bar{\boldsymbol{x}}_J$. We follow their recommendations and apply the same temporal smoothing to the input.

4. Discriminative Hierarchical Rank Pooling

In this section we present our major contributions. First, in Section 4.1 we present our main idea of *hierarchical rank pooling* and formulate our hierarchical rank pooling representation. Next, in Section 4.2 and Section 4.3, we complete the technical details of our hierarchical rank pooling method by describing our temporal encoding machinery and non-linear feature transforms, respectively. Finally, in Section 4.4, we present our second major contribution for learning discriminative dynamics.

4.1. Hierarchical rank pooling

Even with a rich feature representation of each frame in a video sequence, such as derived from a deep CNN model [14], the shallow rank pooling method [4] may not be able to adequately model complex activities over long sequences. As such, we propose a more powerful scheme for encoding the dynamics of CNN features from a video sequence. Motivated by the success of hierarchical encoding of deep neural networks [14, 6], we extend rank pooling to encode dynamics of a sequence at multiple levels in a hierarchical manner. Moreover, at each stage, we apply a non-linear feature transformation (which we later learn) to capture complex dynamical behavior. We call our method *hierarchical rank pooling*.

Our main idea is to perform rank pooling on subsequences of the video. Each invocation of rank pooling provides a fixed-length feature vector that describes the subsequence. Importantly, the feature vectors capture the evolution of frames within each sub-sequence. Now, the subsequences themselves are ordered. As such, we can apply rank pooling over the generated sequence of feature vectors to obtain a higher-level representation. This process is repeated to obtain dynamic representations at multiple levels for a given video sequence until we obtain a final encoding. To make this hierarchical encoding even more powerful, we apply a point-wise non-linear operation on the input to the rank pooling function. An illustration of the approach is shown in Figure 2.

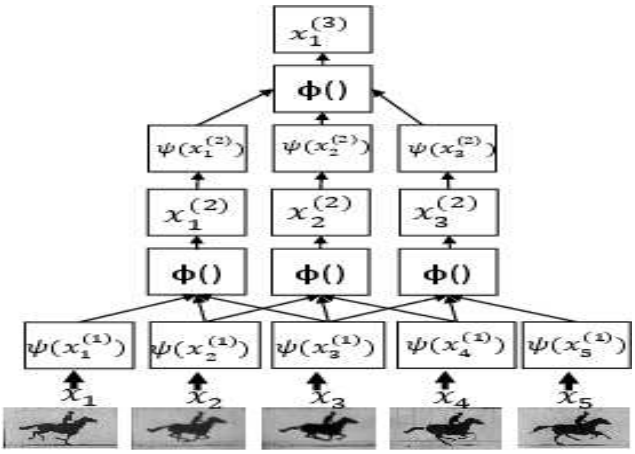


Figure 2: Two layer network of hierarchical rank pooling with window size three ($\mathcal{M}_\ell = 3$) and stride one ($S_\ell = 1$).

Concretely, let $X^{(1)} = \langle \boldsymbol{x}_1, \ldots, \boldsymbol{x}_J \rangle$ be the sequence of CNN feature vectors describing each frame in the input video. We first apply a non-linear transformation ψ to each feature vector to obtain a modified sequence $\widetilde{X}^{(1)} = \langle \psi(\boldsymbol{x}_1), \ldots, \psi(\boldsymbol{x}_J) \rangle$. Then applying temporal encoding ϕ to sub-sequences of $\widetilde{X}^{(1)}$ we obtain a sequence $X^{(2)}$ of feature vectors describing each video sub-sequence. The temporal encoding function ϕ and non-linear transform ψ will be described in detail in the following subsections.

The process of going from $X^{(1)}$ to $X^{(2)}$ constitues the first layer of our hierarchy. We can now extend the process through additional rank pooling layers, which we formalize by the following definition.

Definition 1 (Rank Pooling Layer). *Let* $X^{(\ell)} = \langle \boldsymbol{x}_1^{(\ell)}, \ldots, \boldsymbol{x}_{J_\ell}^{(\ell)} \rangle$ *be a sequence of* J_ℓ *feature vectors. Let* M_ℓ *be the window size and* S_ℓ *be a stride. For* $t \in \{1, S_\ell + 1, 2S_\ell + 1, \ldots\}$ *define transformed sub-sequences* $\widetilde{X}_t^{(\ell)} = \langle \psi(\boldsymbol{x}_t^{(\ell)}), \ldots, \psi(\boldsymbol{x}_{t+M_\ell-1}^{(\ell)}) \rangle$*, where* $\psi(\cdot)$ *is a point-wise non-linear transformation. Then the output of the* ℓ*-th* rank pooling layer *is a sequence* $X^{(\ell+1)} = \langle \ldots, \boldsymbol{x}_t^{(\ell+1)}, \ldots \rangle$ *where* $\boldsymbol{x}_t^{(\ell+1)} = \phi(\widetilde{X}_t^{(\ell)})$ *is a temporal encoding of the transformed sub-sequence* $\widetilde{X}_t^{(\ell)}$.

Each successive layer in our rank pooling hierarchy captures the dynamics of the previous layer. The entire hierarchy can be viewed as applying a stack of non-linear ranking functions on the input video sequence X and shares some conceptual similarities with deep neural networks. A simple illustration of a two-layer hierarchical rank pooling network is shown in Figure 2.

By varying the stride and window size for each layer we

can control the depth of the rank pooling hierarchy. There is no technical reason to limit the number of layers but experimentally we found that a depth of two or three layers is sufficient for activity recognition.

To obtain the final vector representation $\boldsymbol{x}^{(L+1)}$, we construct the sequence for the final layer $X^{(L)}$, and encode the whole sequence $X^{(L)}$ with a single temporal encoding $\phi(\widetilde{X}^{(L)})$. In other words, the last layer in our hierarchy produces a single temporal encoding of last output sequence $\widetilde{X}^{(L)}$. We use this final feature vector $\boldsymbol{x}^{(L+1)}$ of the video as its representation, which can then be classified by standard techniques, *e.g.*, SVM classifier.

4.2. Sequence encoding machinery ϕ

As described above, our hierarchical rank pooling requires a temporal encoding function ϕ. One choice is the point-wise ranking function $f(\cdot;\boldsymbol{u}) : \boldsymbol{x}_t \mapsto t$ parametrized by $\boldsymbol{u}$, which clearly satisfies the order constraints of Equation 1. The idea of rank pooling is to estimate the function $f(\cdot;\boldsymbol{u})$ using sequence data X and to use the parameter $\boldsymbol{u}$ as a sequence representation. Due to its fast implementations, we use Support Vector Regression (SVR) to learn the parameters of the point-wise ranking function [19]. Given a sequence of length J, the SVR parameters are given by

$$\boldsymbol{u}^\star \in \underset{\boldsymbol{u}}{\operatorname{argmin}} \left\{ \frac{1}{2}\|\boldsymbol{u}\|^2 + \frac{C}{2}\sum_{t=1}^{J}\Big[|t - \boldsymbol{u}^T\boldsymbol{v}_t| - \epsilon\Big]^2_{\geq 0} \right\} \quad (3)$$

where $\boldsymbol{v}_t = \psi(\boldsymbol{x}_t)$ is a non-linear feature mapping and $[\cdot]_{\geq 0} = \max\{\cdot, 0\}$ projects onto the positive reals.

The above SVR objective is a relaxation of the mapping $f : \boldsymbol{x}_t \mapsto t$. However, this relaxation is useful in the case of modelling dynamics of videos as the above SVR objective is robust. We use the parameter $\boldsymbol{u}^\star$, returned by SVR, as the temporal encoding vector of sequence X.

4.3. Capturing non-linear dynamics ψ

With the temporal encoding function defined, we now provide details on our non-linear feature mapping, *i.e.*, $\boldsymbol{v}_t = \psi(\boldsymbol{x}_t)$ in Equation 3. Usually, video sequence data contains complex dynamic information that cannot be captured simply using linear methods. To obtain non-linear dynamics, one option is to use non-linear feature maps. Here we transform the input vectors $\boldsymbol{x}_t$ by a non-linear operation $\psi(\boldsymbol{x}_t)$ before applying SVR based rank pooling. The resulting mapping that generates dynamics can be written as $f : \psi(\boldsymbol{x}_t) \mapsto t$. In the literature, Signed Square Root (SSR) and Chi-square feature mappings are used to obtain good results. Neural networks employ sigmoid and hyperbolic tangent functions to model non-linearity. The advantage of SSR is exploited by Fisher vector-based object recognition as well as in activity recognition [4, 33]. Our experience is that it is important to consider positive activations separately from the negative activations when CNN features are used. Typically the rectification applied in CNN architectures keeps only the positive activations, *i.e.*, $\psi(\boldsymbol{x}) = \max\{\boldsymbol{0}, \boldsymbol{x}\}$. However, we argue that negative activations may also contain some useful information and should be considered. Therefore, we propose to use the following non-linear function on the activations of fully connected layers of the CNN architecture. We call this operation the *sign expansion root* (**SER**),

$$\psi(x) = \Big(\sqrt{\max\{0, x\}}, \sqrt{\max\{0, -x\}}\Big) \quad (4)$$

This operation doubles the size of the features space allowing us to capture important non-linear information, one for positives and the other for negatives. The square-root operation takes care of projecting features to a non-linear space induced by the Hellinger kernel [31]. SER has the same property but specifically tailored for CNN activation to exploit negative activations as well.

So far we have described how to represent a video by a fixed-length descriptor using hierarchical rank pooling in an unsupervised manner. These descriptors can be used to learn an SVM classifier for activity recognition. In the next section we describe how to jointly learn the activity classifier *and* video representation using labelled video data. In this way we build a representation that is more discriminative for the activity recognition task.

4.4. Learning discriminative dynamics

Until now we have assumed that the non-linear function ψ used for capturing sequence dynamics is fixed. In this section we extend our framework to allow us to learn sequence dynamics and activity classification jointly. We propose to learn a parametric non-linear function $\psi(W\boldsymbol{x}_t)$ where $W \in \mathbb{R}^{D\times D}$ are shared across sequences.

Recall, the sequence encoding of X is obtained by optimizing Equation 3 to get $\boldsymbol{u}^\star$, now with $\boldsymbol{v}_t = \psi(W\boldsymbol{x}_t)$. Instead of using an SVM we use a soft-max classifier for determining the activity of videos. Let Y be a random variable denoting the activity of the video and let $\{\boldsymbol{w}_c\}_{c=1}^K$ denote the classifier parameters. Then we can write the soft-max probability as

$$p_Y(c) = P(Y = c \mid X; \theta) \propto e^{h_c(X;\boldsymbol{w}_c,W)} \quad (5)$$

where h_c is the scoring function for the c-th class

$$h_c(X;\boldsymbol{w}_c,W) = \boldsymbol{w}_c^T\boldsymbol{u}^\star \quad (6)$$

and $\theta = \{\boldsymbol{w}_c, W \mid c = 1,\ldots,K\}$ are the model parameters. Importantly, $\boldsymbol{u}^\star$ is a function of both the input video X and the non-linearity parameters W, *i.e.*, $\boldsymbol{u}^\star = \phi(X;W)$.

A principled approach to determining the parameters is to find the θ that minimize the regularized negative log-likelihood of the training set, whose objective function can

be written as

$$\mathcal{L}(\theta) = R(\theta) - \sum_{i=1}^{n}\sum_{c=1}^{K} [\![Y^i = c]\!] \log p_{Y^i}(c) \quad (7)$$

where $R(\theta)$ is some regularization function and $[\![\cdot]\!]$ is the indicator function evaluating to one when its argument is true and zero otherwise. As is standard in large-scale machine learning problems we can use stochastic gradient descent to optimize Equation 7.

The main difficulty in differentiating $\mathcal{L}(\theta)$ with respect to (w.r.t.) the model parameters θ is in taking the derivative of $\log p_{Y^i}(c)$,

$$\nabla_\theta \log p_{Y^i}(c) = (1 - p_{Y^i}(c))\, \nabla_\theta h_c(X^i; \boldsymbol{w}_c, W) \quad (8)$$

which in turn requires computing derivatives of the scoring function $h_c(X; \boldsymbol{w}_c, W)$. The partial derivative of h_c w.r.t. the class variables $\boldsymbol{w}_{c'}$ is straightforward,

$$\nabla_{\boldsymbol{w}_{c'}} h_c(X; \boldsymbol{w}_c, W) = \begin{cases} \boldsymbol{u}^\star, & \text{if } c = c' \\ \mathbf{0}, & \text{otherwise.} \end{cases} \quad (9)$$

However, the partial derivative w.r.t. W is more challenging since we have to differentiate *through* the argmin function of Equation 3. Such derivations are proposed before in the context of bi-level optimization [20]. Let e_t be the SVR loss contribution in Equation 3 for each $\boldsymbol{x}_t$, *i.e.*,

$$e_t = \begin{cases} t - \boldsymbol{v}_t^T \boldsymbol{u}^\star - \epsilon, & \text{if } t - \boldsymbol{u}^T \boldsymbol{v}_t \geq \epsilon \\ t - \boldsymbol{v}_t^T \boldsymbol{u}^\star + \epsilon, & \text{if } \boldsymbol{u}^T \boldsymbol{v}_t - t \geq \epsilon \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

Let $\mathbf{1}$ be the all-ones vector of size n. Then let $K_t = \frac{\partial \boldsymbol{v}_t}{\partial W}$ where $(K_t)_{[ij]} \doteq \frac{\partial \boldsymbol{v}_{t[i]}}{\partial W_{ij}}$ where subscript $[i]$ denotes the i-th element of the associated vector. Let $s_t = \widehat{\boldsymbol{w}}_c^T \boldsymbol{v}_t$ be a scalar where the scaled classifier parameters $\widehat{\boldsymbol{w}}_c$ given by

$$\widehat{\boldsymbol{w}}_{c[i]} = \frac{\boldsymbol{w}_{c[i]}}{1 + C \sum_{t:e_t \neq 0} \psi^2_{[i]}(W \boldsymbol{x}_t)} \quad (11)$$

Then we can write the partial derivative of h_c w.r.t. W as

$$\nabla_W h_c = C \sum_{\substack{t=1 \\ e_t \neq 0}}^{J} e_t K_t \odot (\widehat{\boldsymbol{w}}_c \cdot \mathbf{1}^T) - s_t K_t \odot (\boldsymbol{u}^\star \cdot \mathbf{1}^T) \quad (12)$$

where $\odot$ is the Hadamard product. The complete derivation of Equation 12 is given in the supplementary material.

In our experiments, we use this discriminative learning framework for the final layer of the hierarchical rank pooling network. In this case we first construct the sequence for the final layer $X^{(L)}$ and apply SSR feature map. Then we feed forward this sequence through the parameterized non-linear transform $\psi(W\boldsymbol{x}_t^{(L)})$, temporal encoder $\phi(\widetilde{X}^{(L)})$, and apply the classifier to get a classification score. During training we propagate errors back to the parametric non-linear transformation layer $\psi(\cdot)$ and perform a parameter update. Our overall algorithm is summarized below.

Algorithm 1: Hierarchical rank pooling.

for each labeled training video $(X, Y) \in \mathcal{X}$ **do**
  extract CNN features, $X^{(1)} = \langle \boldsymbol{x}_1^i, \boldsymbol{x}_2^i, \ldots, \boldsymbol{x}_J^i \rangle$
  for each rank pooling layer, $\ell = 1 : L-1$ **do**
    generate transformed sub-sequences $\widetilde{X}_t^{(l)}$
    rank pool sub-sequences, $\boldsymbol{x}_t^{(\ell+1)} = \phi(\widetilde{X}_t^{(\ell)})$
    construct $X^{(l+1)}$ as $\langle \ldots, \boldsymbol{x}_t^{(\ell+1)}, \ldots \rangle$
  end for
  get video representation as $\phi(\widetilde{X}^{(L)})$
end for
if learning discriminative dynamics **then**
  train non-linear transform and classifier (§4.4)
else
  train SVM classifier only (§4.3)
end if

5. Experiments

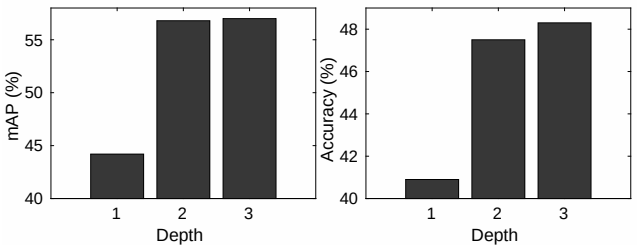
We evaluate our hierarchical rank pooling method on three important activity recognition datasets. We follow exactly the same experimental settings per dataset, using the same training and test splits as described in the literature. The datasets are: HMDB51 [15], which consist of 6,766 video clips divided into 51 action classes with three splits, Hollywood2 [18], which consists of 12 human action classes from 1,707 video clips, and UCF101 [27], which consists of 101 action categories from 13,320 videos over three splits.

Experimental details: Our primary focus is to evaluate activity recognition performance using CNN features and hierarchical rank pooling. We utilize pre-trained CNNs without any fine-tuning. Specifically, for each video we extract activations from the VGG-16 [25] network's first fully connected layer (consisting of 4096 values, only from the central patch). We represent each video frame by this vector before applying ReLU [14]. As a result the frame representation vector contains both positive and negative components of the activations.

Unless otherwise specified, we use a window size M_ℓ of 20, with a stride S_ℓ of one and a hierarchy depth of two in all our experiments. We use a constant $C = 1$ parameter for SVR training (Lib-linear). We test diffenent non-linear SVM classifiers for the final classification always with $C = 1000$ (LibSVM). For rank pooling [4] and trajec-

METHOD	Hollywood2	HMDB51	UCF101
Average pooling	40.9	37.1	69.3
Max pooling	42.4	39.1	72.5
Tempo. pyramid (avg. pool)	46.5	39.1	73.3
Tempo. pyramid (max pool)	48.7	39.8	74.8
LSTM [28]	–	42.8	74.5
LRCN [2]	–	–	68.8
Rank pooling	44.2	40.9	72.2
Recursive rank pooling	52.5	45.8	75.6
Hierarchical rank pooling	**56.8**	**47.5**	**78.8**
Improvement	+8.1	+4.7	+4.0

Table 1: Comparing several temporal pooling methods for activity recognition using VGG-16's fc6 features.



(a) Hollywood2 (b) HMDB-51

Figure 3: Activity recognition performance versus hierarchy depth on Hollywood2 and HMDB-51.

tory extraction [33] (in later experiments) we use the publicly available code from the authors.

5.1. Comparing temporal pooling methods

In this section we compare several temporal pooling methods using VGG-16 CNN features. We compare our hierarchical rank pooling with average-pooling, max-pooling, LSTM [28], two level temporal pyramids with mean pooling, two level temporal pyramids with max pooling, recently proposed LRCN [2], and vanilla rank pooling [4]. To obtain a representation for average pooling, the average CNN feature activation over all frames of a video was computed. The max-pooled vector is obtain by applying the *max* operation over each dimension of the CNN feature vectors from all frames of a given video. We also compare with a variant of hierarchical rank pooling called *recursive rank pooling*, where the next layer's sequence element at time t denoted by $\boldsymbol{x}_t^{(\ell+1)}$ is obtained by encoding *all frames* of the previous layer sequence up to time t, *i.e.* $\boldsymbol{x}_t^{(\ell+1)} = \phi(\langle\psi(\boldsymbol{x}_1^{(\ell)}), \ldots, \psi(\boldsymbol{x}_t^{(\ell)})\rangle)$ for $t = 2, \ldots, J_\ell$.

We compare these base temporal encoding methods on three datasets and report results in Table 1. Results show that the rank pooling method is only slightly better than max pooling or mean pooling. We believe this is due to the limited capacity of rank pooling [4]. Moreover, temporal pyramid seems to outperform rank pooling except for HMDB51 dataset. LRCN results seem disappointing. However, these results in [2] are obtained using a Caffe reference model whereas we use more powerful VGG-16 features (though without any fine tuning). Using a Caffe reference model (also without fine tuning) we obtain 74.6%. Interestingly, when we extend rank pooling to *recursive rank pooling*, we notice a jump in performance from 44.2% to 52.5% for Hollywood2 and 40.9% to 45.8% in HMDB51. We also see a noticeable improvement in UCF101 dataset. Hierarchical rank pooling improves over rank pooling by a significant margin. The results suggest that it is important to exploit dynamic information in a hierarchical manner as it allows complicated sequence dynamics of videos to be expressed. To verify this, we also performed an experiment by varying the depth of the hierarchical rank pooling and reported results for one to three layers. Results are shown in Figure 3.

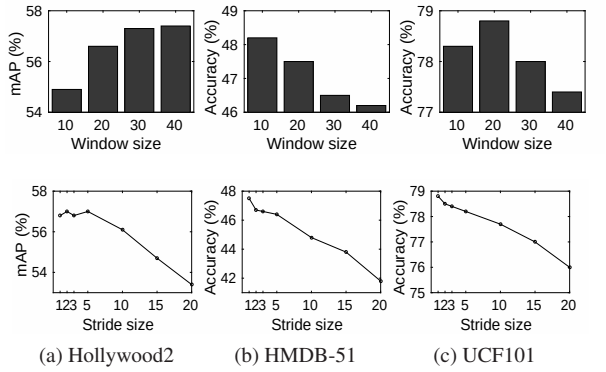
As expected the improvement from depth of one to two is significant. Interestingly, as we increase the depth of the hierarchy to three, the improvement is marginal. Perhaps with only two levels, one can obtain a high capacity dynamic encoding.

5.2. Evaluating other parameters

Hierarchical rank pooling consists of two more hyper-parameters: (1) **window size**, *i.e.*, the size of the video sub-sequences and (2) **stride** of the video sampling. These two parameters control how many sub-sequences can be generated at each layer. In the next experiment we evaluate how performance varies with **window size** and **stride**. Results are reported in Figure 4(top). The window size does not seem to make a big impact on the results (1–2%) for some datasets. However, we experimentally verified that a window size of 20 frames seems to be a reasonable compromise for all activity recognition tasks. The trend in Figure 4(bottom) for the stride is interesting. It shows that the best results are always obtained by using a small stride. Small strides generate more encoded sub-sequences capturing more statistical information.

5.3. The effect of non-linear feature maps

Non-linear feature maps are important for modeling complex dynamics of an input video sequence. In this section we compare our new Sign Expansion Root (SER) feature map introduced in Section 4.3 with the Signed Square Root (SSR) method, which is commonly used in the literature [22]. Results are reported in Table 2. As evident in the table, our new feature map SER is useful not only for hierarchical rank pooling, which gives an improvement of 6.3% over SSR, but also for baseline rank pooling method, which gives an improvement of 6.8%. This seems to suggest that there is valuable information in both positive and negative



(a) Hollywood2 (b) HMDB-51 (c) UCF101

Figure 4: Activity recognition performance versus window size (top) and stride (bottom).

METHOD	Rank pooling	Hierarchical rank pooling
Signed square root (SSR)	44.2	50.5
Sign expansion root (SER)	51.0	**56.8**

Table 2: Effect non-linear feature maps during the training of rank pooling methods using Hollywood2 dataset.

activations of fully connected layers. Furthermore, this experiment suggests that it is important to consider positive and activations separately for activity recognition.

5.4. The effect of non-linear kernel SVM

Generally, it is necessary to utilize SVM classifiers with non-linear kernels to obtain good generalizable results. This is mostly due to the complex nature of image and video data. In this experiment we evaluate several non-linear kernels that exist in literature and compare their effect when used with Hierarchical Rank Pooling method. We compare classification performance using different kernels (1) linear, (2) linear kernel with SSR, (3) Chi-square kernel, (4) Kernelized SER (5) combination of Chi-square kernel with SER. Results are reported in Table 3. On all three datasets we see a common trend. First, the SSR kernel is more effective than not utilizing any kernel or feature map. Interestingly, on deep CNN features, Chi-square Kernel is more effective than SSR. Perhaps this is because the Chi-square kernel utilizes both negative and positive activations in a separate manner to some extent. The SER method seems to be the most effective kernel. Interestingly, applying SER feature map over Chi-square kernel seems to improve results further. We conclude that SER non-linear feature map is effective not only during the training of rank pooling techniques, but also for action classification specially when used with CNN activation features. Next we also evaluate the effect of non-linear kernels on final video representations when used with other pooling methods such as rank pooling, average

KERNEL TYPE	Hollywood2 (mAP %)	HMDB51 (%)	UCF101 (%)
Linear	45.1	40.0	66.7
Signed square root (SSR)	48.6	42.8	72.0
Chi-square kernel	50.6	44.2	73.8
Sign expansion root (SER)	54.0	46.0	76.6
Chi-square + SER	**56.8**	**47.5**	**78.8**

Table 3: Effect of non-linear SVM kernels on action classification with hierarchical rank pooling representation.

KERNEL TYPE	Avg. pool	Max pool	Rank pool	Ours
Linear	38.1	39.6	33.3	45.1
Signed square root (SSR)	38.6	38.4	35.3	48.6
Chi-square kernel	39.9	41.1	40.8	50.6
Sign expansion root (SER)	39.4	41.0	37.4	54.0
Chi-square + SER	**40.9**	**42.4**	**44.2**	**56.8**

Table 4: Effect of non-linear kernels on other pooling methods using Hollywood2 dataset (mAP %).

METHOD	Results
Rank pooling	72.2
Hierarchical rank pooling	78.8
Discriminative hierarchical rank pooling	**81.4**

Table 5: Effect of learning discriminative dynamics for hierarchical rank pooling on the UCF101 dataset.

pooling and max pooling. Results are reported in Table 4 on Hollywood2 dataset. A similar trend as in the previous table can be observed here. We conclude that our kernalized SER is useful not only for our hierarchical rank pooling method, but also for the other considered temporal pooling techniques.

5.5. The effect of discriminative rank pooling

Now we evaluate the effect of discriminative method in Section 4.4, which we implemented with full GPU support. We evaluate the effect of this method only on the largest dataset, UCF101. We first construct the first layer sequence using hierarchical rank pooling. Then we learn the parameters θ using the labelled video data. We initialize the W matrix to the identity and the classifier parameters to those obtained from the linear SVM classifier. Results are reported in Table 5. We improve results by 2.6% over hierarchical rank pooling and a significant improvement of 9.2% over rank pooling. However, due to the lack of training data, we do not fully exploit the advantages of our discriminative rank pooling method using chosen datasets. During test time, we process a video at 120 frames per second.

METHOD	Hollywood2	HMDB51	UCF101
RP. (HOG)	53.4	44.1	72.8
RP. (HOF)	64.0	53.7	78.3
RP. (MBH)	65.8	53.9	82.6
RP. (ALL)	68.5	60.0	86.5
RP. (ALL+CNN)	71.4	63.0	88.1
HRP. (CNN)	56.8	47.5	78.8
RP. (ALL)+ HRP (CNN)	**74.1**	**65.0**	**90.7**

Table 6: Combining CNN-based Hierarchical Rank Pooling (HRP) with improved trajectory features encoded with Fisher vectors and Rank Pooling(RP).

	Hollywood2	HMDB51	UCF101
*our * method*	**76.7**	**66.9**	91.4
Zha et al. *[37]*	–	–	89.6
Yue-Hei-Ng et al. *[36]*	–	–	88.6
Simonyan et al. *[24]*	–	59.4	88.0
Wang et al. *[34]*		65.9	**91.5**
Methods without CNN features			
Lan et al. *[17]*	68.0	65.4	89.1
Fernando et al. *[4]*	73.7	63.7	–
Hoai et al. *[7]*	73.6	60.8	–
Peng et al. *[21]*	–	66.8	–
Wu et al. *[35]*	–	56.4	84.2
Wang et al. *[33]*	64.3	57.2	–

Table 7: Comparison with the state-of-the-art methods.

5.6. Combining with trajectory features

In this experiment we combine CNN features which are encoded using hierarchical rank pooling with trajectory features [33] which are encoded with Fisher vectors [22] and rank pooling [4]. We use Improved Dense Trajectory (MBH, HOG, HOF) [33] encoded with Fisher vectors [22] at frame level. We utilize a Gaussian mixture model of 256 components to create the Fisher vectors. To keep the dimensionality manageable, we halve the size of each descriptor using PCA. This is exactly the same setup used by Fernando *et al*. [4]. Due to the extremely large dimensionality of the Fisher vectors, we do not use hierarchical rank pooling over Fisher vectors. To obtain state-of-the-art results, we use encode trajectory features [33] using rank pooling and combine them with CNN features encoded with hierarchical rank pooling. For each dataset we report results on HOG, HOF and MBH features obtained with the publicly available code of rank pooling [4]. We use average kernel method to combine CNN features with trajectory features. Results are shown in Table 6. Hierarchical rank pooled (CNN) outperforms trajectory based HOG features on all three datasets. Furthermore, on UCF101 dataset, Hierarchical rank pooled (CNN) outperforms rank pooled HOF features. Nevertheless, trajectory based MBH features still dominate the best results for an individual feature. The combination of rank pooled trajectory features (HOG+HOF+MBH) with hierarchically rank pooled CNN features gives a significant improvement. It is interesting to see that the biggest improvement is obtained in Hollywood2 dataset where state-of-the art results are obtained. On HMDB51 dataset just using default parameters, we almost reach previous state-of-the art results. On UCF-101 dataset the combination brings us above the state of the art performance with an improvement of 4.2% over rank pooled trajectory features. We conclude that our hierarchical rank pool features are complimentary to trajectory-based rank pooling.

5.7. Comparing to state-of-the-art

In this section we discuss state-of-the-art performance. We combine rank pooled trajectory features (HOG+HOF+MBH) with CNN features encoded with our method. In addition to that, we choose parameters for hierarchical rank pooling based on the prior experimental results reported in Figures 3 and 4 for each of the dataset, *i.e*., *without* use of any grid search. As in [4, 7] we use data augmentation only for Hollywood2 and HMDB51. We evaluate discriminative rank pooling method only for UCF101 dataset as there is not that much of training data in Hollywood2 and HMDB51 datasets. Results are reported in Table 7. We improves over previous state-of-the-art by 3.0% on Hollywood2 dataset, by 0.1% on HMDB51 dataset and reach state-of-the-art on UCF101 dataset.

6. Conclusion

In this paper we present a novel temporal encoding method called *hierarchical rank pooling* which consists of a network of non-linear operations and rank pooling layers. The obtained video representation has high capacity and capability of capturing informative dynamics of rich frame-based feature representations. We also presented a principled way to learn non-linear dynamics using a stack consisting of parametric non-linear activation layers, rank pooling layers and, a soft-max classifier which we coined *discriminative hierarchical rank pooling*. We demonstrated substantial performance improvement over other temporal encoding and pooling methods such as max pooling, rank pooling, temporal pyramids, and LSTMs. Combining our method with features from the literature, we obtained state-of-the-art results on the Hollywood2, HMDB51 and UCF101 datasets.

Acknowledgements. We thank Sebastian Nowozin for discussions on bi-level optimization. This research was supported by the Australian Research Council Centre of Excellence for Robotic Vision (project number CE140100016).

References

[1] H. Bilen, B. Fernando, E. Gavves, A. Vedaldi, and S. Gould. Dynamic image networks for action recognition. In *CVPR*, 2016.

[2] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[3] Y. Du, W. Wang, and L. Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *CVPR*, 2015.

[4] B. Fernando, E. Gavves, J. Oramas, A. Ghodrati, and T. Tuytelaars. Modeling video evolution for action recognition. In *CVPR*, 2015.

[5] B. Fernando, E. Gavves, J. Oramas, A. Ghodrati, and T. Tuytelaars. Rank pooling for action recognition. *TPAMI*, 2016.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[7] M. Hoai and A. Zisserman. Improving human action recognition using score distribution and ranking. In *ACCV*, 2014.

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] M. Jain, H. Jégou, and P. Bouthemy. Better exploiting motion for better action recognition. In *CVPR*, 2013.

[10] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, pages 3304–3311. IEEE, 2010.

[11] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *PAMI*, 35(1):221–231, 2013.

[12] T. Joachims. Training linear svms in linear time. In *ICKDD*, 2006.

[13] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105. 2012.

[15] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: a large video database for human motion recognition. In *ICCV*, 2011.

[16] T. Lan, Y. Zhu, A. R. Zamir, and S. Savarese. Action recognition by hierarchical mid-level action elements. In *ICCV*, 2015.

[17] Z. Lan, M. Lin, X. Li, A. G. Hauptmann, and B. Raj. Beyond gaussian pyramid: Multi-skip feature stacking for action recognition. In *CVPR*, 2015.

[18] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *CVPR*, 2008.

[19] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[20] P. Ochs, R. Ranftl, T. Brox, and T. Pock. *Scale Space and Variational Methods in Computer Vision (SSVMCV)*, chapter Bilevel Optimization with Nonsmooth Lower Level Problems, pages 654–665. Springer, 2015.

[21] X. Peng, C. Zou, Y. Qiao, and Q. Peng. Action recognition with stacked fisher vectors. In *ECCV*, 2014.

[22] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier. Large-scale image retrieval with compressed fisher vectors. In *CVPR*, 2010.

[23] M. S. Ryoo, B. Rothrock, and L. Matthies. Pooled motion features for first-person videos. In *CVPR*, June 2015.

[24] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, pages 568–576, 2014.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[26] Y. Song, L.-P. Morency, and R. Davis. Action recognition by hierarchical sequence summarization. In *CVPR*, 2013.

[27] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[28] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. *arXiv preprint arXiv:1502.04681*, 2015.

[29] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[30] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015.

[31] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *PAMI*, 34:480–492, 2012.

[32] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Dense trajectories and motion boundary descriptors for action recognition. *IJCV*, 103:60–79, 2013.

[33] H. Wang and C. Schmid. Action recognition with improved trajectories. In *ICCV*, 2013.

[34] L. Wang, Y. Qiao, and X. Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *CVPR*, pages 4305–4314, 2015.

[35] J. Wu, Y. Zhang, and W. Lin. Towards good practices for action video encoding. In *CVPR*, 2014.

[36] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, 2015.

[37] S. Zha, F. Luisier, W. Andrews, N. Srivastava, and R. Salakhutdinov. Exploiting image-trained CNN architectures for unconstrained video classification. In *BMVC*, 2015.

Discriminative Hierarchical Rank Pooling for Activity Recognition: Supplementary Material

In this supplementary note we provide the derivation of Equation (12) in the main paper for updating the parameters $W \in \mathbb{R}^{D\times D}$ of our non-linear function $\psi(W\boldsymbol{x}_t)$. Let us remind the reader that we are given a function of the form

$$\boldsymbol{u}^\star(W) = \underset{\boldsymbol{u}}{\operatorname{argmin}} \left\{ \frac{1}{2}\|\boldsymbol{u}\|^2 + \frac{C}{2}\sum_{t=1}^{J}\Big[|t - \boldsymbol{u}^T\psi(W\boldsymbol{x}_t)| - \epsilon\Big]_{\geq 0}^2 \right\} \tag{1}$$

which we need to differentiate (with respect to W). We start with two lemmas.

Lemma 1.: Let $f : \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ be a conitunous function with first and second derivatives. Let $g(x) = \operatorname{argmin}_y f(x,y)$. Then

$$\frac{dg}{dx} = -\frac{f_{XY}(x,g(x))}{f_{YY}(x,g(x))}$$

where $f_{XY} = \frac{\partial^2 f}{\partial x \partial y}$ and $f_{YY} = \frac{\partial^2 f}{\partial y^2}$.

Proof.

$$\left.\frac{\partial f(x,y)}{\partial y}\right|_{y=g(x)} = 0 \qquad \text{(since } g(x) = \underset{y}{\operatorname{argmin}}\, f(x,y)\text{)} \tag{2}$$

$$\therefore \frac{d}{dx}\frac{\partial f(x,g(x))}{\partial y} = 0 \qquad \text{(differentiating lhs and rhs)} \tag{3}$$

But

$$\frac{d}{dx}\frac{\partial f(x,g(x))}{\partial y} = \frac{\partial^2 f(x,g(x))}{\partial x\partial y} + \frac{\partial^2 f(x,g(x))}{\partial y^2}\frac{dg(x)}{dx} \tag{4}$$

Equating to zero and rearranging gives the desired result

$$\frac{dg(x)}{dx} = -\left(\frac{\partial^2 f(x,g(x))}{\partial y^2}\right)^{-1}\frac{\partial^2 f(x,g(x))}{\partial x\partial y} \tag{5}$$

$$= \frac{f_{XY}(x,g(x))}{f_{YY}(x,g(x))} \tag{6}$$

□

Lemma 2: Let $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ be a conitunous function with first and second derivatives. Let $g(x) = \text{argmin}_{\boldsymbol{y} \in \mathbb{R}^n} f(x, \boldsymbol{y})$. Then

$$\nabla_x g(x) = -f_{YY}(x, g(x))^{-1} f_{XY}(x, g(x)).$$

where $f_{YY} = \nabla^2_{\boldsymbol{yy}} f(x, \boldsymbol{y}) \in \mathbb{R}^{n \times n}$ and $f_{XY} = \frac{d}{dx} \nabla_{\boldsymbol{y}} f(x, \boldsymbol{y}) \in \mathbb{R}^n$.

Proof. Similar to Lemma 1, we have:

$$\nabla_Y f(x, g(x))|_{\boldsymbol{y}=g(x)} = 0 \tag{7}$$
$$\therefore\ f_Y(x, g(x)) = 0 \tag{8}$$
$$\frac{d}{dx} f_Y(x, g(x)) = 0 \tag{9}$$
$$\therefore f_{XY}(x, g(x)) + f_{YY}(x, g(x)) g'(x) = 0 \tag{10}$$
$$\frac{d}{dx} g(x) = -f_{YY}(x, g(x))^{-1} f_{XY}(x, g(x)) \tag{11}$$

□

Now let

$$f(W, \boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{u}\|^2 + \frac{C}{2} \sum_{t=1}^{J} \Big[|t - \boldsymbol{u}^T \psi(W \boldsymbol{x}_t)| - \epsilon \Big]^2_{\geq 0} \tag{12}$$

Let $\boldsymbol{v}_t = \psi(W \boldsymbol{x}_t)$. Then

$$\nabla_{\boldsymbol{u}} \frac{1}{2} \left[|t - \boldsymbol{u}^T \boldsymbol{v}_t| - \epsilon \right]^2_{\geq 0} = \begin{cases} (\boldsymbol{u}^T \boldsymbol{v}_t - t + \epsilon) \boldsymbol{v}_t, & \text{if } t - \boldsymbol{u}^T \boldsymbol{v}_t \geq \epsilon \\ (\boldsymbol{u}^T \boldsymbol{v}_t - t - \epsilon) \boldsymbol{v}_t, & \text{if } \boldsymbol{u}^T \boldsymbol{v}_t - t \geq \epsilon \\ \boldsymbol{0}, & \text{otherwise} \end{cases} \tag{13}$$

$$\nabla^2_{\boldsymbol{uu}} \frac{1}{2} \left[|t - \boldsymbol{u}^T \boldsymbol{v}_t| - \epsilon \right]^2_{\geq 0} = \begin{cases} \boldsymbol{v}_t \boldsymbol{v}_t^T, & \text{if } |t - \boldsymbol{u}^T \boldsymbol{v}_t| \geq \epsilon \\ \boldsymbol{0}, & \text{otherwise} \end{cases} \tag{14}$$

$$\frac{\partial}{\partial W_{ij}} \nabla_{\boldsymbol{u}} \frac{1}{2} \left[|t - \boldsymbol{u}^T \boldsymbol{v}_t| - \epsilon \right]^2_{\geq 0} = \begin{cases} (\boldsymbol{u}^T \frac{\partial \boldsymbol{v}_t}{\partial W_{ij}}) \boldsymbol{v}_t + (\boldsymbol{u}^T \boldsymbol{v}_t - t + \epsilon) \frac{\partial \boldsymbol{v}_t}{\partial W_{ij}}, & \text{if } t - \boldsymbol{u}^T \boldsymbol{v}_t \geq \epsilon \\ (\boldsymbol{u}^T \frac{\partial \boldsymbol{v}_t}{\partial W_{ij}}) \boldsymbol{v}_t + (\boldsymbol{u}^T \boldsymbol{v}_t - t - \epsilon) \frac{\partial \boldsymbol{v}_t}{\partial W_{ij}}, & \text{if } \boldsymbol{u}^T \boldsymbol{v}_t - t \geq \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

Now, let

$$\delta_t = \begin{cases} 1, & \text{if } t - \boldsymbol{u}^T \boldsymbol{v}_t \geq \epsilon \\ -1, & \text{if } \boldsymbol{u}^T \boldsymbol{v}_t - t \geq \epsilon \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

and with slight abuse of notation, let $e_t = \delta_t \cdot (|t - \boldsymbol{v}_t^T \boldsymbol{u}^\star| - \epsilon)$. Then

$$f_{UU}(W, \boldsymbol{u}^\star) = I + C \sum_{t : \delta_t \neq 0} \boldsymbol{v}_t \boldsymbol{v}_t^T \tag{17}$$

$$\frac{\partial}{\partial W_{ij}} f_U(W, \boldsymbol{u}^\star) = -C \sum_{t : \delta_t \neq 0} e_t \frac{\partial \boldsymbol{v}_t}{\partial W_{ij}} - \langle \boldsymbol{u}^\star, \frac{\partial \boldsymbol{v}_t}{\partial W_{ij}} \rangle \boldsymbol{v}_t \tag{18}$$

$$\therefore\ \frac{\partial \boldsymbol{u}^\star(W)}{\partial W_{ij}} = \left(I + C \sum_{t : \delta_t \neq 0} \boldsymbol{v}_t \boldsymbol{v}_t^T \right)^{-1} \left(C \sum_{t : \delta_t \neq 0} e_t \frac{\partial \boldsymbol{v}_t}{\partial W_{ij}} - \langle \boldsymbol{u}^\star, \frac{\partial \boldsymbol{v}_t}{\partial W_{ij}} \rangle \boldsymbol{v}_t \right) \tag{19}$$

Now, $\boldsymbol{v}_t = \psi(W\boldsymbol{x}_t)$ where $\psi(\cdot)$ operates element-wise. Therefore,

$$\left(\frac{\partial \boldsymbol{v}_t}{\partial W_{ij}}\right)_k = \begin{cases} \psi'(W\boldsymbol{x}_t)_{[k]}\boldsymbol{x}_{t[j]}, & \text{if } k = i \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

Moreover, let us approximate Equation 17 by a diagonal matrix. Then,

$$\frac{\partial \boldsymbol{u}_k^\star(W)}{\partial W_{ij}} = \frac{1}{1 + C\sum_{t:\delta_t \neq 0} \psi_k^2(W\boldsymbol{x}_t)} C \sum_{t:\delta_t \neq 0} \left([\![k = i]\!] e_t - \boldsymbol{u}_i^\star \psi_k(W\boldsymbol{x}_t)\right) \psi_i'(W\boldsymbol{x}_t)\boldsymbol{x}_{t[j]} \tag{21}$$

Remember that the score function for class c is given by $h_c = \boldsymbol{w}_c^T \boldsymbol{u}^\star(W)$. Then,

$$\frac{\partial h_c}{\partial W_{ij}} = \frac{\partial h_c}{\partial \boldsymbol{u}^\star} \cdot \frac{\partial \boldsymbol{u}^\star}{\partial W_{ij}} \tag{22}$$

$$= \boldsymbol{w}_c^T \frac{\partial \boldsymbol{u}^\star(W)}{\partial W_{ij}} \tag{23}$$

Now define the scaled classifier parameters $\widehat{\boldsymbol{w}}_c$ by

$$\widehat{\boldsymbol{w}}_{c[i]} = \frac{\boldsymbol{w}_{c[i]}}{1 + C\sum_{t:\delta_t \neq 0} \psi_{[i]}^2(W\boldsymbol{x}_t)} \tag{24}$$

$$\frac{\partial h_c}{\partial W_{ij}} = \widehat{\boldsymbol{w}}_c^T \left(C \sum_{t:\delta_t \neq 0} e_t \frac{\partial \boldsymbol{v}_t}{\partial W_{ij}} - \boldsymbol{u}^T \frac{\partial \boldsymbol{v}_t}{\partial W_{ij}} \boldsymbol{v}_t \right) \tag{25}$$

Let $\mathbf{1}$ be the all-ones vector of size n. Then let $K_t = \frac{\partial \boldsymbol{v}_t}{\partial W}$. Let $s_t = \widehat{\boldsymbol{w}}_c^T \boldsymbol{v}_t$ be a scalar. Then

$$\frac{\partial h_c}{\partial W_{ij}} = C \sum_{t:\delta_t \neq 0} e_t (K_t)_{[ij]} \widehat{\boldsymbol{w}}_{c[i]} - s_t (K_t)_{[ij]} \boldsymbol{u}_{[i]}^\star \tag{26}$$

which gives our result

$$\nabla_W h_c = C \sum_{\substack{t=1 \\ \delta_t \neq 0}}^{J} e_t K_t \odot (\widehat{\boldsymbol{w}}_c \cdot \mathbf{1}^T) - s_t K_t \odot (\boldsymbol{u}^\star \cdot \mathbf{1}^T) \tag{27}$$

where $\odot$ is the Hadamard product. We note that the equation in the main paper is missing the $\delta_t \neq 0$. This is a mistake in transcribing from our notes when writing the paper. Our software implementation and reported results use the correct summation.