Universiteit
Leiden
The Netherlands

Sequence Models for Spatio-Temporal Dependencies in Video Object Detection

Deniz Sen

Thesis advisor: Dr. Prof. Olaf Hellwich - Technical University Berlin
Thesis advisor: Dr. Prof. Fons Verbeek - Leiden Institute of Advanced Computer Science
Thesis advisor: Dr. Prof. Peter Grünwald - Mathematical Institute Leiden University

Defended on Date , Year

MASTER THESIS

STATISTICS AND DATA SCIENCE
UNIVERSITEIT LEIDEN

Contents

i Symbols and Notation

Matrices are capitalized in regular type, vectors are non-capitalized in bold type and scalars are typically non-capitalized in regular type, unless they follow notation conventions as defined below or is explicitly mentioned to be otherwise.

Symbol	Meaning
W	Weight matrix.
N	Dimension of feature space.
C	Number of classes.
D	Dimension of input space X.
X	$D \times n$ matrix of the training inputs $x_{i=1}^{N}$: the design matrix.
T	Length of an input sequence $x_{t=1}^{T}$
t	Time-step.
h	Hidden-state.
c	Cell-state.
h_t	Hidden-state at time-step t.
c_t	Cell-state at time-step t.
F	Force.
m	Mass.
a	Acceleration.
a	Velocity.
$\mathcal{L}$	Loss function or objective function.
$\odot$	Element-wise multiplication.
$\frac{\partial a}{\partial b}$	The partial derivative of a w.r.t. b.
(x)	An arbitrary function f(x).
$\frac{df}{dx}$	Leibniz notation for the first order derivative of an arbitrary function $f(x)$.
$f'(x)$	Shorthand notation for first order derivative of an arbitrary f(x).
$f''(x)$	Shorthand notation for second order derivative of an arbitrary f(x).
$\int_a^b f(x)\,dx$	Integral of an arbitrary function $f(x)$ from a to b.
$w.r.t.$	short hand notation for "with respect to".

ii Acknowledgement

I would like to thank my teachers, supervisors and colleagues Wojtek Kowalczyk, Aske Plaat, Peter Grünwald, Vons Verbeek, Olaf Hellwich and Monika Kwiatkowski, Marc Toussaint, Hongyou Zhou and Bryan Armstrong Gass for their support, time and thoughts.

Abstract

A central goal of video object detection is to utilize the rich spatio-temporal signal that emerges from a sequence of consecutive frames. Although conventional models such as Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Transformers have specialized variants to capture long-range temporal dependencies across multiple frames, image degradation such as motion blur, camera defocus, large pose variation, and occlusions spanning over longer duration in time make this a challenging task. Even though state-of-the-art methods have refined feature selection, proposal, and aggregation modules, designed to capture both short and long-term dependencies, State Space Models (SSMs) have emerged as a promising approach to replace and outperform RNNs and attention-based mechanisms. Within this context, we propose a comparative study between a CNN with no ability to capture long-term dependencies, and variants that utilize RNNs and SSMs to model temporal dependencies to better understand the limitations of these methods when processing the spatio-temporal signal in video object detection.

1 Introduction

Given the precise nature of the human visual system, which is able to localise, classify and predict an objects motion in a dynamic scene with ease, an open problem within the computer vision community has been to perform object detection at a similar pace and accuracy on spatio-temporal data such as videos and webcam-feed. That is, to gain a complete understanding of dynamic scenes and the spatio-temporal relation between consecutive frames, the community has tried to solve localisation and classification of objects both at the current and a future time-step in real-time given a sequence of RGB images. While object detection on still-images has made ample progress and lead to the very first fast real-time object detectors [1; 2], models are trained on images that are not temporally correlated and predictions are made one-single frame at a time. Even-though, it is possible to re-purpose still-image based object detectors for spatio-temporal data such as videos, fast motion and dynamical changes introduce image degradation like motion blur, camera de-focus and large pose variation, unseen in the still-image setting [3]. As previous frames in a sequence provide a rich history of contextual information about a successive frame, efforts to explicitly model such dependencies are a natural choice to mitigate object appearance deteriorations in a frame.

2 Related work

Traditionally, object detectors fall within two general family of frameworks: 1) region proposal-based frameworks and 2) regression-classification-based frameworks. In this section we review members of these two families due to their close relation to the thesis project on both still-image object detection and video object detection.

2.1 Object detection on still-images

Region based frameworks generally consist of a two-step process, where in the first step the model performs a rough scan over the image domain to find proposal regions and then focuses on these proposals [4]. Such an approach is to be preferred over a generic sliding window method, where a square rectangular region slides across the entire image domain. Without proposal regions the window has to slide across each pixel location, making it an exhaustive, inefficient and slow search strategy. Proposal regions thus limit the search space to a smaller number of regions of interests. Among the most notable region-based proposal networks are R-CNN [5], Faster R-CNN [2] and Feature Pyramid Networks [6]. An example of a classical R-CNN can be seen in Fig. 1.

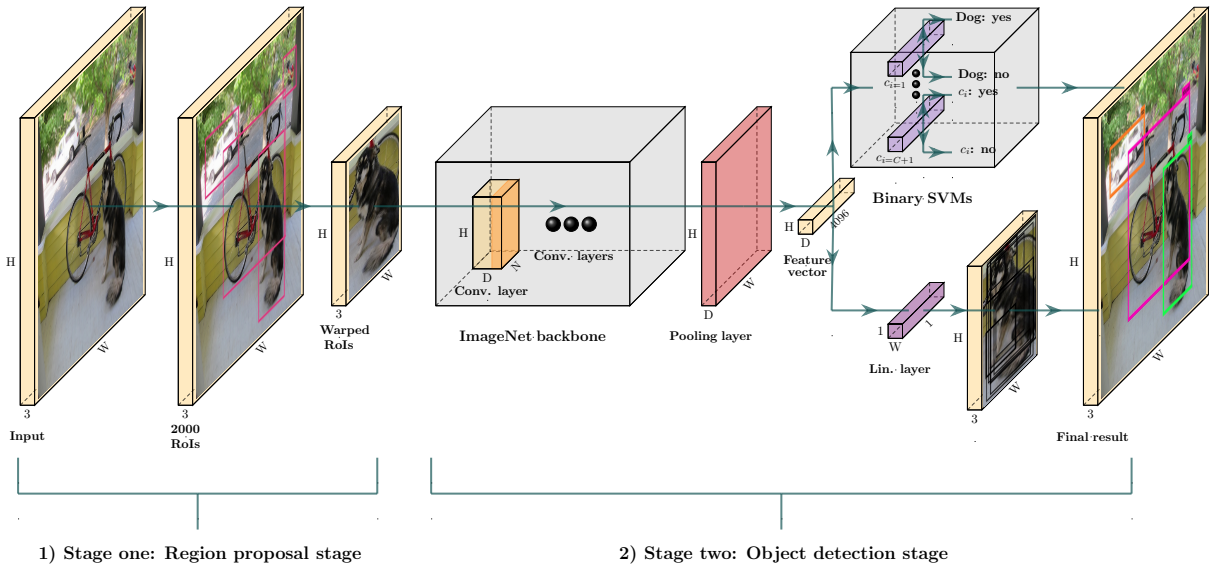
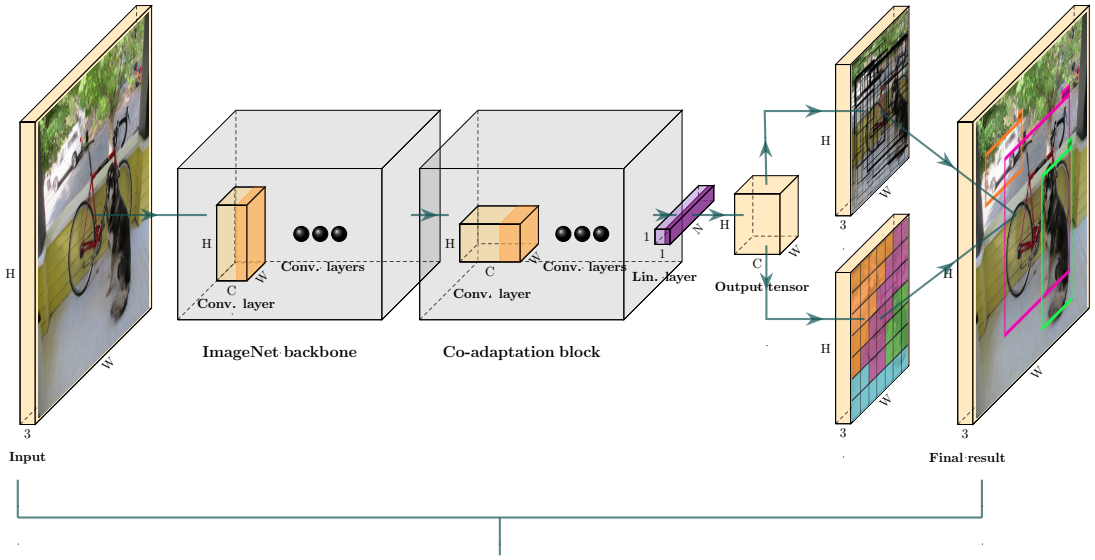


Figure 1: Generalized R-CNN still-image object detection workflow. The system is divided into a 1) a region proposal stage and 2) an object detection stage. The region proposal component generates regions of interest that serve as bounding box candidates, which are warped for

further processing of the object detection component. The warped image is fed to the backbone, pooled and flattened, resulting in a feature vector of size 4096. Features are then used by a binary SVM classifier to compute $C = C + 1$, class plus background probabilities and a linear layer to predict the bounding box adjustment to the bounding box proposal from the region proposal component.

The family of regression-classification based networks treats object detection as a regression problem in which predictions are made directly, using a single convolutional network that has a prediction head for classification and bounding box coordinates, rather than independently processing each potential region. This makes regression based networks extremely speed efficient compared to region based frameworks. Representative works include the initial Yolo model [1] and its different versions [7; 8; 9; 10], Single Shot MultiBox Detection (SSD) [11] and its variants [12; 13]. The generalised workflow of a typical regression based model exemplified using YoloV1 can be see in Fig. 2. Object-detection models trained on still-images tend not to perform competitively on videos due to domain shift factors attributed to dynamic changes in the scenes [14].



1) Single stage system: Input to output is end-to-end.

Figure 2: Generalized YoloV1 still-image object detection workflow. The system divides the image into an $S \times S$ grid. Then for each grid cell it predicts B bounding boxes, confidence for those boxes, and C class probabilities, where these predictions are encoded as an $S \times S(5 \times B + C)$ tensor.

.

2.2 Video object detection

Compared to still-image detectors, which make predictions on a single input image, the main challenge in video object detection lies in how to utilize the rich signal of temporal continuity in multiple video frames to improve performance as well as inference-speed. As such, aggregating spatio-temporal features is a fundamental part in video object detection and can be divided into two categories: 1) pixel-wise [15; 16; 17; 18; 19] and 2) object-wise [20; 21; 22] aggregation strategies. While pixel-level based methods aggregate information over entire feature maps per frame, object-level approaches focus on aggregating bounding-box or object features throughout time by focusing on areas with high probability of containing an object. Early work on video object detection utilized object-level approaches in which still-image detectors are re-purposed for video detection by leveraging temporal information by means of ad-hoc post-processing techniques that propagate or link bounding-boxes across frames [23; 24; 25; 26]. However, these methods perform bounding-box level post-processing built on still-image object detectors, which may be sub-optimal as they are not jointly optimized. As a consequence, they have difficulty dealing with consecutive

failures of the still-image detectors e.g., when the object-of-interest has large occlusion or unusual appearance for a long time. More recent methods [27; 15; 28; 29; 30] have integrated temporal contingency into the model during training for both object-level and pixel level approaches utilizing various aggregation techniques.

Specialised recurrent neural networks [31; 32; 18] and attention-based models [19], which aggregate spatial information through time at a pixel-level across neighbouring frames to refine predictions have also been developed. Similar to pixel level methods, work on attention based mechanism on an object-level are also popular [33; 34; 35; 36; 37].

Recent state-of-the-art methods have explored various ways to improve video object detection. For example, YoloV [30] utilized a pre-trained still-image detector as a backbone and adds a specialised Feature Selection Module (FSM), followed by an attention-based Feature Aggregation Module (FAM) between backbone and the prediction head. In SLTnet [37] box features are first associated and aggregated by linking proposals from the same anchor box in nearby frames. Then, a specialised attention module that aggregates short-term box features for long-term spatio-temporal information is used. Both spatial information from reference frames and the aggregated short and long-term temporal context are then fed into a prediction head to obtain bounding-box coordinates and corresponding object categories. Temporal ROI align [29] utilizes a region of interest (ROI) operator that extracts features from the feature maps of other reference frames to obtain proposals for the current frame at hand by means of feature similarity. The most similar proposal from different reference frames are then aggregated using a temporal attention feature aggregator. All of these methods exploit some form of feature selection or proposal module at a pixel or bounding box level to extract the temporal context, which are then aggregated.

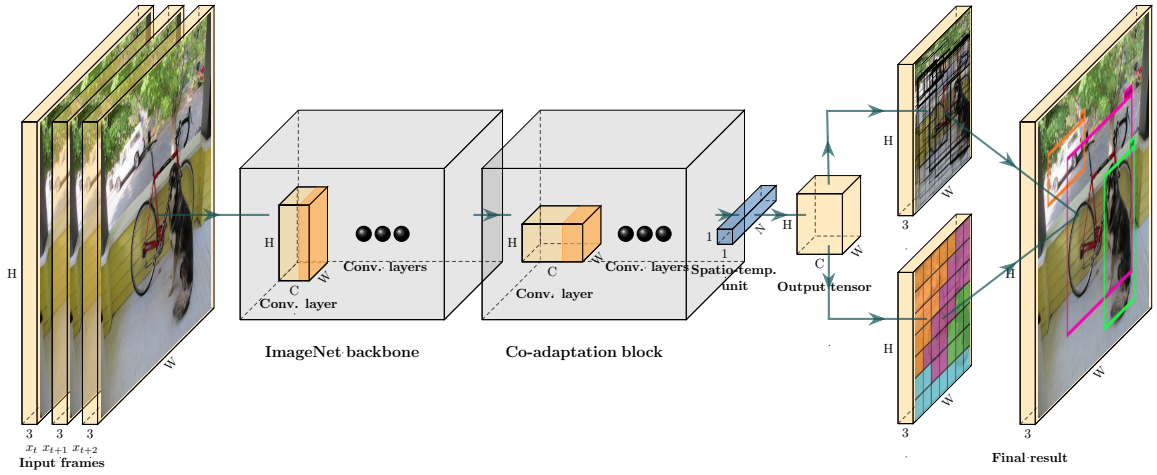


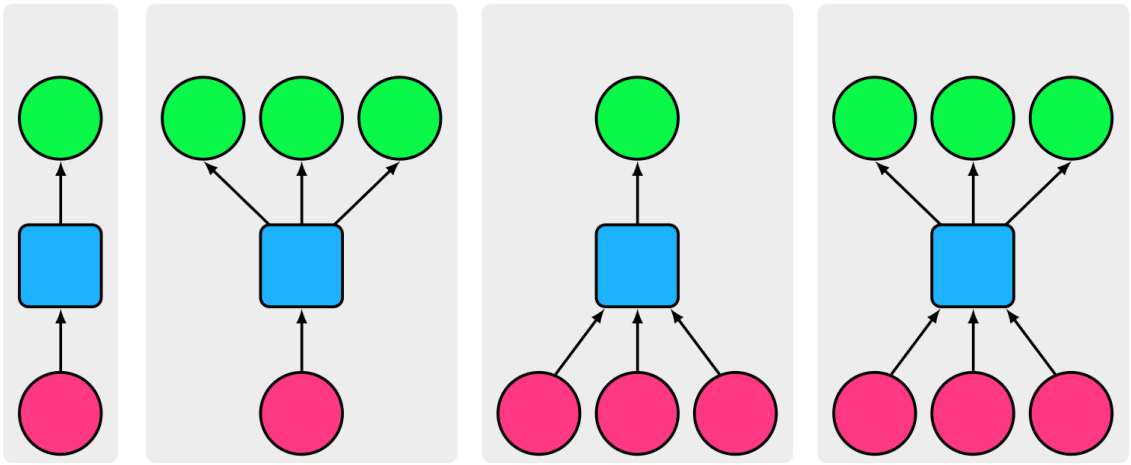
Figure 3: Adapted YoloV1 workflow for video object detection. The linear layer has been replaced by a sequence model i.e., an RNN represented as a blue square.

3 Sequence Models

Different models define different input-to-output representations. In its most simple case, a model that maps a single input to a single output and thus forms a one-to-one input-to-output representation is called a "one-to-one" model (see Fig. 4(a)). Similarly a model that maps a single input to many output, forms a one-to-many relation and is called a "one-to-many" model (see Fig. 4(b)). Models that define an input-to-output relation, in which the input is a single observation are not part of the family of sequence models, as the inputs are processed independently, meaning that potential dependencies between two inputs are not modelled.

Sequence models are a family of models that are designed to handle multiple inputs that are represented as a sequence. Whether they map the input sequence to one output (see Fig. 4(c)) or multiple outputs (see Fig. 4(d)), is task dependent, as long as the model is exposed to multiple

inputs dependently. That way sequence models can capture dependencies between observations, making them particularly suitable for tasks in which changes are a function of time.



(a) One-to-one. (b) One-to-many. (c) Many-to-one. (d) Many-to-many.

Figure 4: Simplified examples of different type of sequence models. Input vectors are highlighted in red, sequence models are blue and output vectors are in green.

3.1 Recurrent Neural Network

Recurrent neural networks (RNNs) [38; 39; 40; 41] are a class of neural networks that belong to the family of sequence models. They process data sequentially and maintain an internal memory or state that enables them to capture long-range dependencies and patterns in sequential data. This makes RNNs well-suited for a variety of computer vision tasks, such as object tracking [42; 43] or gesture recognition [44; 45], in which the previous context (i.e., a frame earlier in the sequence) provides valuable information about the next frame.

Compared to conventional neural networks (i.e., multi-layer perceptrons or convolutional neural networks), a key difference in a RNN is that the output of a neuron is also sent back to itself, describing a recurrent system. Consider a simple recurrent unit or cell as shown in Fig. 5. At each time-step t, the recurrent neuron receives two inputs to compute its hidden-state $\mathbf{h}_t$: 1) the input signal $\mathbf{x}_t$ and 2) its own output from the previous time-step $\mathbf{h}_{t-1}$. This recurrence relation can then be expressed formally as function $f(.)$ of $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ parameterised by the weights W as shown in Eq. (1):

$$\begin{aligned} \mathbf{h}_t \in \mathbb{R}^d &= f_W\left(\mathbf{h}_{t-1}, \mathbf{x}_t\right) \\ &= \phi\left(\mathbf{x}_t W_{ih}^T + \mathbf{b}_{ih} + \mathbf{h}_{t-1} W_{hh}^T + \mathbf{b}_{hh}\right), \end{aligned} \tag{1}$$

where:

- $\mathbf{h}_t \in \mathbb{R}^{n \times p}$ is the hidden-state vector of size $n \times p$ at the time-step t, where n is the batch-size and p is the number of hidden neurons.
- $\mathbf{x}_t \in \mathbb{R}^{n \times i}$ is the input signal at time-step t of size $n \times i$, where n is the batch-size and i is the length of the input vector.
- $W_{ih}^T \in \mathbb{R}^{i \times p}$ is the transposed weight matrix of size $i \times p$, where i is the input length. This weight matrix maps the input to the hidden-state.
- $W_{hh}^T \in \mathbb{R}^{p \times p}$ is the transposed weight matrix of size $p \times p$. This weight matrix maps the hidden-state to the hidden-state.
- $\mathbf{b}_{ih} \in \mathbb{R}^p$ is the vector of bias terms of size p. The index ih denotes that it is the bias for the weights that map from the input i to the hidden-state h.
- $\mathbf{b}_{hh} \in \mathbb{R}^p$ is the vector of bias terms of size p. The index hh denotes that it is the bias for the weights that map from the hidden-state h to the hidden-state h.

- ϕ is the activation function: tanh, sigmoid or ReLU.

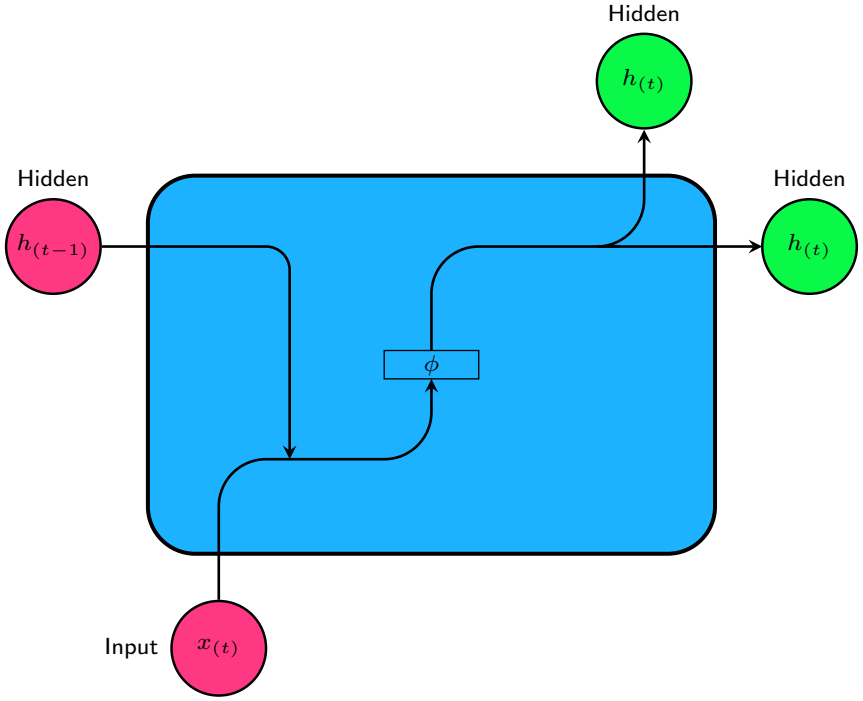


Figure 5: Gating mechanism of a simple RNN cell. The cell takes as input the input $\mathbf{x_t}$ at the current time-step t and the hidden-state at the previous time-step $t-1$, returning the hidden-state at the current time-step $\mathbf{h_t}$. Inputs are colored red, outputs green, and the cell including the gating mechanism are in blue.

3.1.1 Backpropagation Through Time

RNNs are trained in a sequential supervised manner. For each time-step t, the error is given by the difference between the predicted and target value: $(\hat{y}_t - y_t)$, where the overall loss $\mathcal{L}(\hat{y}, y)$ is usually the sum of the time-step specific loss found in the range of interest $[t, T]$ given by:

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{t=1}^{T} \mathcal{L}\left(\hat{\mathbf{y}}_t, \mathbf{y}_t\right) \tag{2}$$

Training of an unfolded recurrent neural network as illustrated in Fig. 6, is done across multiple time-steps using a generalization of the backpropagation algortihmn, known as backpropagation through time (BPTT), where the overall gradient is equal to the sum of the individual error or loss gradients at each time-step [46]. Given the total number of time-steps T, the gradient of the loss with respect to the weights is then given by:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W} &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial W} \\ &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_1} \frac{\partial h_1}{\partial W} \end{aligned} \tag{3}$$

The partial derivative of the hidden-state at time-step t with respect to the first hidden-state, involves the product of Jacobians $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ over multiple time-steps, linking an events at time-step t back to one at a time-step $k = 1$, as:

$$\begin{aligned} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_1} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_{t-2}} \cdots \frac{\partial \mathbf{h}_{1+1}}{\partial \mathbf{h}_1} \\ &= \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \end{aligned} \tag{4}$$

This product of subsequent linking Jacobians in Eq. (4), is defined as terms of $\mathbf{h}_t$ w.r.t. $\mathbf{h}_{t-1}$ i.e., $\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}$, that when evaluated on Eq. (1), yields $\mathbf{W}\phi'(\mathbf{h}_{t-1})]$.
As such:

$$\prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}\phi'(\mathbf{h}_{i-1})] \tag{5}$$

Performing and eigenvalue decomposition on the Jacobian $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ given by $\mathbf{W}\phi'(\mathbf{h}_{t-1})$, the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and corresponding eigenvector $v_1, v_2, \ldots, v_n$ can be obtained, where $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$.

Any change in the hidden-state $\triangle \mathbf{h}_t$ in the direction of the eigenvector $\mathbf{v}_t$ has the effect of multiplying the change with the associated eigenvalue $\mathbf{v}_t \triangle \mathbf{h}_t$. As such the product of the Jacobians in Eq. (5) show that subsequent time-steps will result in scaling the change of the hidden-state by a factor equal to λ_i^t, where λ_i^t represents the ith eigenvalue raised to the power of the current time-step t. As such, the sequence of change $\lambda_i^1 \triangle \mathbf{h}_1, \lambda_i^2 \triangle \mathbf{h}_2, \ldots, \lambda_i^t \triangle \mathbf{h}_t$, the factor λ_i^t will dominate the rate of change between hidden-states $\mathbf{v}_t \triangle \mathbf{h}_t$ as it grows. That is, if the largest eigenvalue $\lambda_1 < 1$ gradients will vanish and if $\lambda_1 > 1$ gradients will explode.

Various solutions have been proposed to mitigate or resolve the gradient problem, such as 1) gradient clipping [47], 2) gradient regularization [47], 3) gating mechanism (i.e., Gate Recurrent Unit [40] or Long Short-Term Memory [39] cells) and 4) orthogonal initialisation methods [48]. To exemplify gating mechanism, the following sections will cover an Long Short-Term Memory cells and will also examine a method that utilizes orthogonality to stabilize gradients.

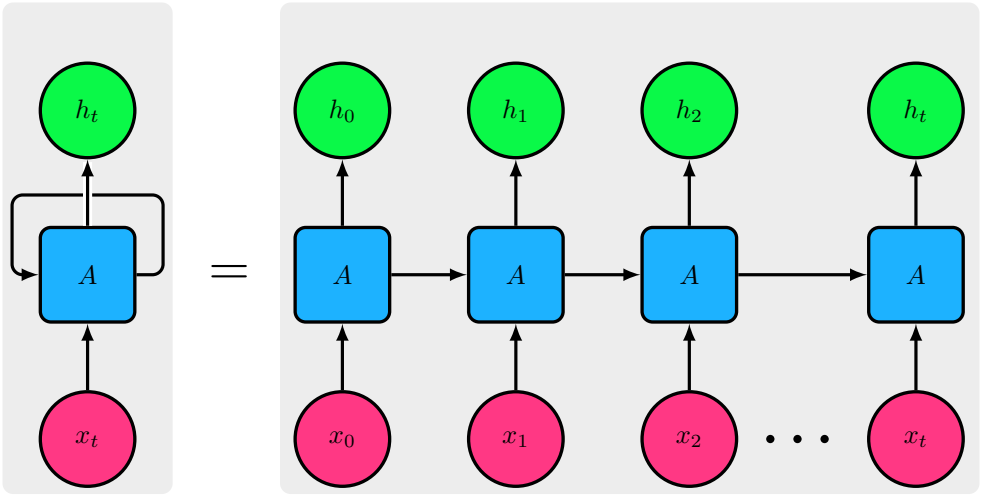


Figure 6: A recurrent neural network that takes input $\mathbf{x}_t$ and the hidden-state from the previous time-step h_{t-1} to produce the current hidden-state $\mathbf{h}_t$ for every time-step t (Left). The same network and its recursion can be unfolded through time (Right). Inputs are colored red, outputs green, and the RNN cell is in blue.

3.1.2 Long Short-Term Memory

A Long Short-Term Memory (LSTM) [39] unit like the Gates Recurrent Unit (GRU) [40], makes use of sophisticated gating mechanisms that control the flow of information to and from the unit. By shutting the gates, these units have the ability to create a loop through which information flows, mitigating the effect of the gradient problem. Consider a simple recurrent LSTM unit or cell as shown in Fig. 7. At each time-step t the LSTM unit receives three inputs to compute a new cell-state $\mathbf{c}_t$ and hidden-state $\mathbf{h}_t$: 1) the input signal $\mathbf{x}_t$ and 2) the hidden-state from the previous time-step $\mathbf{h}_{t-1}$ and 3) the cell-state from the previous time-step $\mathbf{c}_{t-1}$.

Four gates regulate the flow of information: 1) the input-gate i, 2) forget-gate f, 3) the output-gate o and 4) the get-gate g. The input-gate i determines what new information is to be stored in the cell-state $\mathbf{c}_t$. It takes the current input $\mathbf{x}_t$ and the hidden-state from the previous time-step $\mathbf{h}_{t-1}$ and processes them through a series of weighted transformations (W_{ii}, W_{hi}, $\mathbf{b}_{ii}$, $\mathbf{b}_{hi}$), finally

passing them through a sigmoid activation σ, squashing values between 0 and 1 (see Eq. (6)). As such, the output $\mathbf{i}_t$ of input-gate is a vector of values between 0 and 1, where 1 means to completely let the information flow inside and 0 to keep it out. The forget-gate has as similar mechanism and determines how much information from the previous cell-state $\mathbf{c}_{t-1}$ should be discarded, hence forgotten. It takes as input the current input $\mathbf{x}_t$ and the previous hidden-state $\mathbf{h}_{t-1}$, processes them through a series of weighted transformations (W_{if}, W_{hf}, $\mathbf{b}_{if}$, $\mathbf{b}_{hf}$), passing them through a sigmoid activation function σ (see Eq. (7)). As a result, of the sigmoid activation the output is squashed between 0 and 1, where 1 means to completely keep the information and 0 to forget it. The get-gate g creates or "gets" a candidate vector $\mathbf{g}_t$ that can be added to the cell-state. Given the input signal $\mathbf{x}_t$, the previous hidden-state $\mathbf{h}_{t-1}$, it processes the candidate vector through weighted transformations (W_{ig}, W_{hg}, $\mathbf{b}_{ig}$, $\mathbf{b}_{hg}$), lastly processed by a hyperbolic tangent activation function tanh (see Eq. (8)). The output $\mathbf{g}_t$ then represents the new candidate values of new information that can be added to the cell-state. Lastly, the output-gate determines what information from the cell-state should be part of the output at the current time-step. It takes the current input $\mathbf{x}_t$ and the previous hidden-state $\mathbf{h}_{t-1}$, processes them through a series of weighted transformations (W_{io}, W_{ho}, $\mathbf{b}_{io}$, $\mathbf{b}_{ho}$) and passes them through a sigmoid activation function σ (see Eq. (9)). The output vector will consist of values between 0 and 1, where 1 denotes that the information is part of the output and 0 means to leave it in the cell-state.

Updating a the cell-state $\mathbf{c}_t$ then involves combining the previous cell-state $\mathbf{c}_{t-1}$ with the candidate values $\mathbf{g}_t$ scaled by the input-gate values $\mathbf{i}_t$ and forgetting some parts of the previous state using the output of the forget-gate $\mathbf{f}_t$ (see Eq. (10)). To update the hidden-state $\mathbf{h}_t$, the cell-state is passed through a hyperbolic tangent tanh and then scaled by the values of the output gate $\mathbf{o}_t$.

This recurrence relation can then be expressed formally as a function $f(.)$ of $\mathbf{x}_t$, $\mathbf{h}_{t-1}$ and $\mathbf{c}_{t-1}$ parameterised by the weights W as shown in the set of equations Eq. (6) - (11):

$$\begin{aligned}
\mathbf{i}_t &= f_W\left(x_t, h_{t-1}\right) && (6)\\
&= \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right)\\
\mathbf{f}_t &= f_W\left(x_t, h_{t-1}\right) && (7)\\
&= \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\right)\\
\mathbf{g}_t &= f_W\left(x_t, h_{t-1}\right) && (8)\\
&= \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}\right)\\
\mathbf{o}_t &= f_W\left(x_t, h_{t-1}\right) && (9)\\
&= \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}\right)\\
\mathbf{c}_t \in \mathbb{R}^d &= f_t \odot c_{t-1} + i_t \odot g_t && (10)\\
\mathbf{h}_t \in \mathbb{R}^d &= o_t \odot \tanh\left(c_t\right), && (11)
\end{aligned}$$

where:

- $\mathbf{h}_t \in \mathbb{R}^{n\times p}$ is the hidden-state vector of size $n \times p$ at the time-step t, where n is the batch-size and p is the number of hidden neurons.
- $\mathbf{x}_t \in \mathbb{R}^{n\times i}$ is the input signal at time-step t of size $n \times i$, where n is the batch-size and i is the length of the input vector.
- $W_{ii} \in \mathbb{R}^{p\times i}$ is the input-to-input weight matrix of size $p \times i$. This weight matrix maps the input to the hidden-state of the input-gate.
- $W_{hi} \in \mathbb{R}^{p\times p}$ is the hidden-to-input weight matrix of size $p \times p$. This weight matrix maps the hidden-state to the hidden-state of the input-gate.
- $\mathbf{b}_{ii} \in \mathbb{R}^p$ is the bias vector for the input-to-input weights of size p. The index ii denotes that it is the bias for the weights that map from the input x to the input-gate.
- $\mathbf{b}_{hi} \in \mathbb{R}^p$ is the bias vector for the hidden-to-input weights of size p. The index hi denotes that it is the bias for the weights that map from the hidden-state h to the input-gate.
- $W_{if} \in \mathbb{R}^{p\times i}$ is the input-to-forget weight matrix of size $p \times i$. This weight matrix maps the input to the hidden-state of the forget-gate.

- $W_{hf} \in \mathbb{R}^{p \times p}$ is the hidden-to-forget weight matrix of size $p \times p$. This weight matrix maps the hidden-state to the hidden-state of the forget-gate.
- $\mathbf{b}_{if} \in \mathbb{R}^{p}$ is the bias vector for the input-to-forget weights of size p. The index if denotes that it is the bias for the weights that map from the input x to the forget-gate.
- $\mathbf{b}_{hf} \in \mathbb{R}^{p}$ is the bias vector for the hidden-to-forget weights of size p. The index hf denotes that it is the bias for the weights that map from the hidden-state h to the forget-gate.
- $W_{ig} \in \mathbb{R}^{p \times i}$ is the input-to-cell weight matrix of size $p \times i$. This weight matrix maps the input to the hidden-state of the get-gate.
- $W_{hg} \in \mathbb{R}^{p \times p}$ is the hidden-to-cell weight matrix of size $p \times p$. This weight matrix maps the hidden-state to the hidden-state of the get-gate.
- $\mathbf{b}_{ig} \in \mathbb{R}^{p}$ is the bias vector for the input-to-cell weights of size p. The index ig denotes that it is the bias for the weights that map from the input x to the cell gate.
- $\mathbf{b}_{hg} \in \mathbb{R}^{p}$ is the bias vector for the hidden-to-cell weights of size p. The index hg denotes that it is the bias for the weights that map from the hidden-state h to the cell gate.
- $W_{io} \in \mathbb{R}^{p \times i}$ is the input-to-output weight matrix of size $p \times i$. This weight matrix maps the input to the hidden-state of the output gate.
- $W_{ho} \in \mathbb{R}^{p \times p}$ is the hidden-to-output weight matrix of size $p \times p$. This weight matrix maps the hidden-state to the hidden-state of the output gate.
- $\mathbf{b}_{io} \in \mathbb{R}^{p}$ is the bias vector for the input-to-output weights of size p. The index io denotes that it is the bias for the weights that map from the input x to the output gate.
- $\mathbf{b}_{ho} \in \mathbb{R}^{p}$ is the bias vector for the hidden-to-output weights of size p. The index ho denotes that it is the bias for the weights that map from the hidden-state h to the output gate.
- $Tanh$ is an activation function ϕ.
- σ is a sigmoid activation function ϕ.

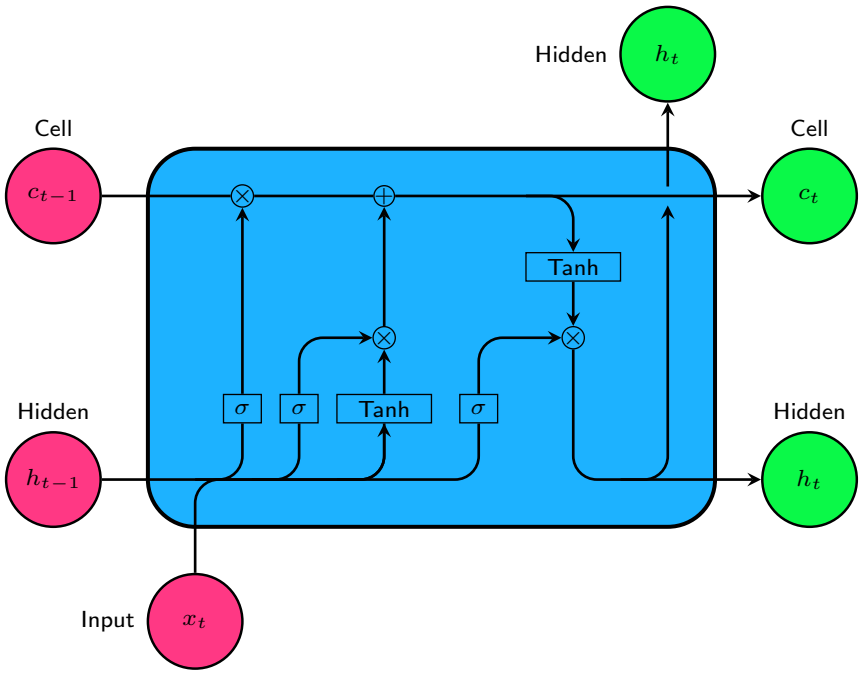


Figure 7: Gating mechanism of an LSTM cell. The cell takes as input the input at the current time-step t, the hidden-state at the previous time-step $t-1$ and cell-state of the previous time-step $t-1$, returning the hidden-state and cell-state of the current time-step.

3.1.3 Cell-State: Backpropagation Through Time

The partial derivative of the cell-state $\mathbf{c}_t$ at time-step t w.r.t the first cell-state $\mathbf{c}_1$, also involves the product of Jacobians $\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}}$ over multiple time-steps, linking an events at time-step t back to one at a time-step $k = 1$, as:

$$\begin{aligned} \frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_1} &= \frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} \frac{\partial \mathbf{c}_{t-1}}{\partial \mathbf{c}_{t-2}} \cdots \frac{\partial \mathbf{c}_{1+1}}{\partial \mathbf{c}_1} \\ &= \prod_{i=k+1}^{t} \frac{\partial \mathbf{c}_i}{\partial \mathbf{c}_{i-1}} \end{aligned} \tag{12}$$

However, examining the exact form of the derivative, including the influence of the gating mechanism on the cell-state $\mathbf{c}_t$, where $\mathbf{u}_t$ is the input to the forget-gate at time-step t:

$$\begin{aligned} \frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} &= \frac{\partial \phi\left(\mathbf{u_t}, \mathbf{c}_{t-1}\right)}{\partial \mathbf{c}_{t-1}} \\ &= \frac{\partial \sigma\left(\mathbf{u}_t\right) \mathbf{c}_{t-1}}{\partial \mathbf{c}_{t-1}} \\ &= \mathbf{c}_{t-1} \underbrace{\frac{\partial \sigma\left(\mathbf{u}_\mathrm{t}\right)}{\partial \mathbf{c}_{\mathrm{t}-1}}}_{=1} + \underbrace{\frac{\partial s_{t-1}}{\partial \mathbf{c}_{\mathrm{t}-1}}}_{=0} \sigma\left(\mathbf{u}_\mathrm{t}\right) \\ &= \sigma\left(\mathbf{u}_\mathrm{t}\right) \end{aligned} \tag{13}$$

Thus:

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \prod_{t=k+1}^{t} \sigma\left(\mathbf{u}_\mathrm{t}\right) \tag{14}$$

Stability between consecutive updates in the cell-state are introduced via the sigmoid activation, binding the term between 0 and 1, preventing the exploding gradient problem [49]. Through multiple products of values below 1, it can still approach zero over long sequences [41]. Examining it as an eigenvalue problem, the sequence of change $\lambda_i^1 \triangle c_1, \lambda_i^2 \triangle c_2, \ldots, \lambda_i^t \triangle c_t$, the factor λ_i^t will still dominate the rate of change between cell-states $v_t \triangle c_t$ as it grows. In this case, due to eigenvalue $\lambda_1 < 1$, the gradients will vanish.

3.1.4 Uniform Refine Long Short-Term Memory

A Uniform Refine Long Short-Term Memory (UR-LSTM) [41] is a refinement of the LSTM gating mechanism in combination with a uniform initialization (U). Standard gate initialisation regimes can prevent learning of long-term temporal dependencies [50] due to saturating gradients. For example, given a simple LSTM cell, with a constant forget-gate value $\mathbf{f}_t$, the influence of the input $\mathbf{x}_t$ will decays exponentially w.r.t. to k time-steps via $\mathbf{f}_t^k$ as shown in Eq. (14) and thus saturate gradients. Initialising the forget-gate, can alleviate the problem of exponential decay, effectively increasing the time-span over which dependencies can be learned [51]. This however, does not alleviate the problem that if gradients of the forget-gate f are past a certain upper or lower threshold, learning stops (See Eq. (7)). To address this, the UR-LSTM adjusts the update of the forget-gate f with an input-dependent additive update $\phi(\mathbf{f}_t, x)$ for some function ϕ, to create an effective gate $g = f + \phi(f, x)$ as shown in Fig. 8 that will be used instead of the original update of the LSTM (see Eq. (11)). As a result, the refine mechanism can be integrated to modulate the gating of an LSTM cell as shown in Fig. 9. Note that the

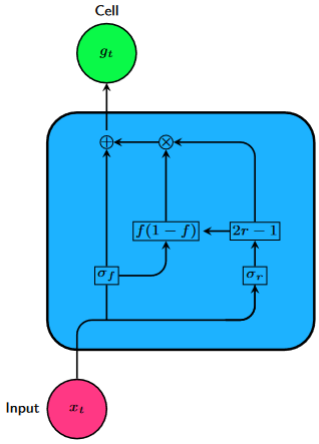


Figure 8: Refine gate used for the UR-LSTM. The mechanism takes as input x_t at time-step t. Note that the σ_f denotes the outputs of the forget-gate f and σ_r the output of the refine-gate r.

adjustment function ϕ is chosen so that after the additive update the activation is bounded between 0 and 1, is symmetric around 0 like a sigmoid and is smooth (i.e., differentiable at any point) for backpropagation. As such, the URLSTM adds two small modifications to the vanilla LSTM. First, it introduces an idealisations strategy on the forget-gate f biases, according to a uniform distribution $\mathcal{U}(0, 1)$:

$$b_f \sim \mathcal{U}(-\frac{1}{\sqrt{\mathrm{p}}}/, \frac{1}{\sqrt{\mathrm{p}}}), \tag{15}$$

p denoting the hidden-size.

Secondly, a refine gate r, which allows for better gradient flow by parameterize the forget-gate f to not saturate as quickly is introduced. Formally, the full mechanism of the refine gate is defined in Eq. (17) - (18). Let u be the get-gate and $\mathbf{u}_t$ its output values from the original LSTM pathway as defined in Eq. (8), then the UR-LSTM gating can be define as:

$$\mathbf{r}_t = \sigma\left(f_{rW}\left(x_t, h_{t-1}\right)\right) \tag{16}$$

$$\mathbf{g}_t = r_t \odot \left(1 - \left(1 - f_t\right)^2\right) + \left(1 - r_t\right) \odot f_t^2 \tag{17}$$

$$\mathbf{c}_t \in \mathbb{R}^d = \mathbf{g}_t \mathbf{c}_{t-1} + \left(1 - \mathbf{g}_t\right) \mathbf{u}_t, \tag{18}$$

where:

- $\mathbf{r}_t \in \mathbb{R}^{n \times p}$ is the output vector of the refinement gate r of size $n \times p$, where n denotes the batch-size and p the hidden size at time-step t.
- f_{rW} is a function parameterized by weights W that produces the refinement gate values before the activation, based on the current input x_t and the previous hidden-state h_{t-1}.
- $\mathbf{g}_t \in \mathbb{R}^{n \times p}$ is the gating vector at time-step t equal to hidden size p,
- $\mathbf{f}_t \in \mathbb{R}^{n \times p}$ is the forget-gate f vector at time-step t equal to hidden size p
- $\mathbf{c}_t \in \mathbb{R}^{n \times p}$ is the cell-state at time-step t equal to hidden size p
- $\mathbf{u}_t \in \mathbb{R}^{n \times p}$ is vector of values from the LSTM get-gate equal to hidden size p
- σ is the sigmoid activation function ϕ.

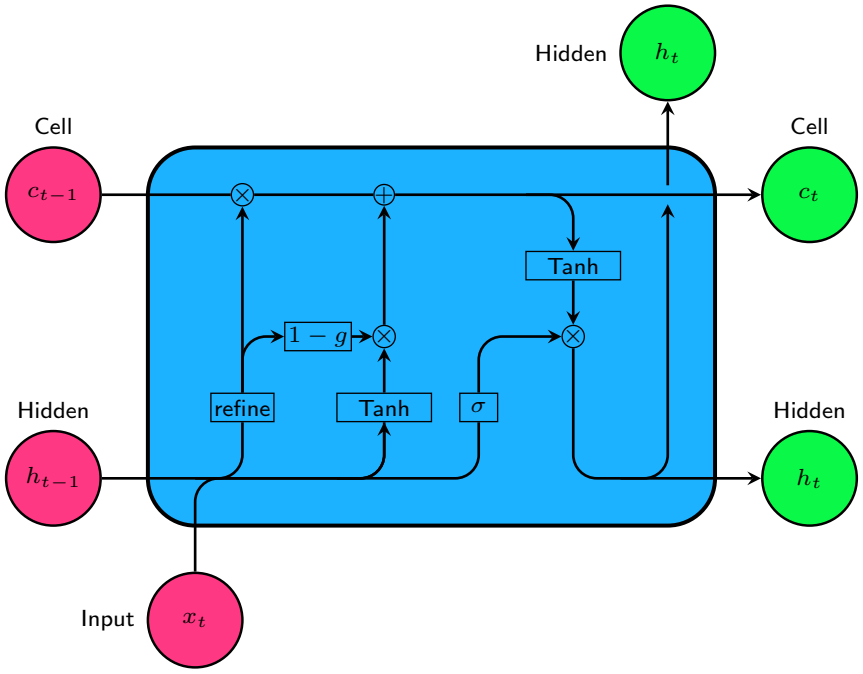


Figure 9: Gating mechanism of a UR-LSTM cell. The cell takes as input the input at the current time-step t, the hidden-state at the previous time-step $t-1$ and cell-state of the previous time-step $t-1$, returning the hidden-state and cell-state of the current time-step.

3.1.5 Refined Cell-State: Backpropagation Through Time

The partial derivative of the refined cell-state $\mathbf{c}_t$ at time-step t w.r.t. the first cell-state $\mathbf{c}_1$ is defined as:

$$\begin{aligned}\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_1} &= \frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} \frac{\partial \mathbf{c}_{t-1}}{\partial \mathbf{c}_{t-2}} \cdots \frac{\partial \mathbf{c}_{1+1}}{\partial \mathbf{c}_1} \\ &= \prod_{i=k+1}^{t} \frac{\partial \mathbf{c}_i}{\partial \mathbf{c}_{i-1}}\end{aligned} \tag{19}$$

Examining the exact form of the derivative including the influence of the gating mechanism on the cell-state $\mathbf{c}_t$, we have:

$$\begin{aligned}\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} &= \frac{\partial \mathbf{g}_t \mathbf{c}_{t-1} + (1 - \mathbf{g}_t)\mathbf{u}_t}{\partial \mathbf{c}_{t-1}} \\ &= \frac{\partial (\mathbf{g}_t \mathbf{c}_{t-1})}{\partial \mathbf{c}_{t-1}} + \frac{\partial ((1 - \mathbf{g}_t)\mathbf{u}_t)}{\partial \mathbf{c}_{t-1}} \\ &= \mathbf{g}_t \underbrace{\frac{\partial \mathbf{c}_{t-1}}{\partial \mathbf{c}_{t-1}}}_{=1} + (1 - \mathbf{g}_t) \underbrace{\frac{\partial \mathbf{u}_t}{\partial \mathbf{c}_{t-1}}}_{=0} \\ &= \mathbf{g}_t \cdot 1 + (1 - \mathbf{g}_t) \cdot 0 \\ &= \mathbf{g}_t + 0 \\ &= \mathbf{g}_t\end{aligned} \tag{20}$$

Thus:

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \prod_{t=k+1}^{t} \mathbf{g}_t \tag{21}$$

Similar to the regular LSTM, the UR-LSTM prevents gradients to explode between consecutive updates in the cell-state through $\mathbf{g}_t$, a function bound between 0 and 1. Gradients can still vanish. As an eigenvalue problem, within the sequence of change $\lambda_i^1 \triangle \mathbf{c}_1, \lambda_i^2 \triangle \mathbf{c}_2, \ldots, \lambda_i^t \triangle \mathbf{c}_t$, the factor λ_i^t can still dominate the rate of change between cell-states $v_t \triangle \mathbf{c}_t$ as it grows. That is, in particular due to eigenvalue $\lambda_1 < 1$, the gradients will still vanish once g_t is below a certain threshold.

3.2 State Space Model

Recently state space models (SSMs) [52; 53; 54] have emerged as an efficient method to model long-range temporal dependencies. While they have been shown to outperform sequence-to-sequence models such as RNNs and Transformers, their application on computer vision tasks is scarce [55]. An attractive feature of SSMs is that compared to transformers [56], which have a quadratic memory and computational cost with respect to the input length, SSMs scale linearly. Furthermore, as they can be formulated as a replacement for the memory mechanism within RNNs using orthogonal polynomial projections, they have shown to address the vanishing and exploding gradient problem [52], which motivates a closer inspection in the following sections. Before a closer examination, preliminaries regarding SSMs, the difference between a continuous-time vs. discrete-time SSM and the general framework used for orthogonal polynomial projections is covered.

3.2.1 Continuous-Time State Space Model

A continuous-time SSM represents a time dynamic system as a set of first order differential equations (ODEs) as shown in Eq. (22). It maps a 1-D input signal $\mathbf{u}(t)$ to an N-D latent space $x'(t)$ before projecting it back to a 1-D signal $y(t)$ as visible in Eq. (23).

$$x'(t) = A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t) \tag{22}$$

$$\mathbf{y}(t) = C(t)\mathbf{x}(t) + D(t)\mathbf{u}(t), \tag{23}$$

where:

- $\mathbf{x}(t) \in \mathbb{R}^n$ is the state vector of size n at time-step t.

- $\mathbf{u}(t) \in \mathbb{R}^m$ is the input vector of size m at time-step t.
- $\mathbf{y}(t) \in \mathbb{R}^p$ is the output vector of size p at time-step t.
- $A(t) \in \mathbb{R}^{n \times n}$ is the system matrix at time-step t, representing how the state variables change with respect to time in response to the system dynamics.
- $B(t) \in \mathbb{R}^{n \times m}$ is the input matrix at time-step t, representing the influence of external forces on the system dynamics.
- $C(t) \in \mathbb{R}^{p \times n}$ is the output matrix at time-step t, relating the state vector to the output vector.
- $D(t) \in \mathbb{R}^{p \times m}$ is the transmission matrix at time-step t, representing the direct influence of the input vector on the output vector.

Consider a mechanical system like a spring-mass system as shown in Fig. 10 to exemplify an SSM. The system's behavior is described using 1) a set of state variables and 2) state equations. The state variables represent the minimal set of information that describes the system at any given time and the state equations express how these state variables change as a function of time using a set of first order ordinary differential equations (ODEs) [57; 58]. As such state space representations describe a dynamic system using state variables and first order ODEs, by expressing the change of the system as a function of its current state. We illustrate this property by examining how to describe the motion of the mass in a spring-mass system, using state space representations. The system has a mass m attached to a spring with a spring constant k and a resulting force F.

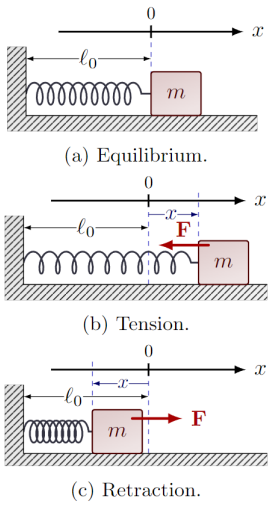


Figure 10: Representation of various dynamic states of a spring-mass system.

To describe the motion of the mass in our spring-mass system using state space representations, we choose two state variables: 1) the displacement $x(t)$ of the mass from its equilibrium position, which is defined as the position of the mass along the x-axis at time t and 2) $v(t)$ the velocity of the mass, which is defined as the rate of change of the position over time [59]. Representing these two variables in a vector we obtain the state vector $\mathbf{x} = [x(t), v(t)]$.

Having defined the minimal set of information that describes the system using the state vector $\mathbf{x}(t)$, we still need to define how the state variables change over time. That is we need to define the state equations. We use use Newton's Second Law:

$$F = ma, \tag{24}$$

where F is the force applied to the mass, m is the mass, and a is the acceleration, along with Hooke's Law for a spring:

$$F = -kx, \tag{25}$$

where F is the force exerted by the spring, k is the spring constant, and x is the displacement from the equilibrium position to derive the state equations.

To form the state equation, first assume that acceleration is defined as the rate of change of velocity v, which can then be defined as a first order ODE: $a = f'(v)$. Secondly, assume that velocity $v(t)$ is defined as the rate of change of the displacement $x(t)$ and can also be defined as a first order ODE: $v(t) = x'(t)$. Since acceleration a is a derivative of a derivative, it can be represented as a second order ODE, therefore $a = x''(t)$. Since both Newtons Second Law and Hooke's Law are equal to F, we can equate the two and substitute a with the second order ODE:

$$ma = -kx(t) \tag{26}$$

$$mx''(t) = -kx(t) \tag{27}$$

$$x''(t) = -\frac{k}{m}x(t), \tag{28}$$

having represented the second order ODE $x''(t)$ as a first order ODE, since $x''(t) = v'(t) = -\frac{k}{m}x$. The state equations are then 1) $x'(t)$ and 2) $v'(t)$, represented in matrix form for completeness:

$$\begin{bmatrix} x'(t) \\ v'(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ v(t) \end{bmatrix}, \tag{29}$$

where A(t), the system matrix is:

$$A(t) = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \tag{30}$$

Adding some external force (or input signal) $\mathbf{u}(t)$ to acts on the system dynamics, regulated through input matrix $B(t)$, we revisit Eq. 26 - 28:

$$ma = -kx(t) + u(t) \tag{31}$$
$$mx''(t) = -kx(t) = u(t) \tag{32}$$
$$x''(t) = -\frac{k}{m}x(t) + \frac{u(t)}{m} \tag{33}$$

Assuming that the external force has no influence (i.e., 0) on the displacement $x(t)$, but has some influences (i.e., 1) on the acceleration $v'(t)$ scaled by mass, we adapt the state equation:

$$\begin{bmatrix} x'(t) \\ v'(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t), \tag{34}$$

where the influence of the external force $\mathbf{u}(t)$ is modulated by input matrix $B(t)$:

$$B(t) = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t), \tag{35}$$

Having defined our state vector $\mathbf{x}(t)$, we can for example predict $\mathbf{y}(t) = [x(t), v(t)]$ as the displacement and velocity of the spring-mass system at any given time. Since the first element in the output vector $\mathbf{y}(t)$ directly corresponds to the displacement $x(t)$ and the second one to the velocity $v(t)$, $C(t)$ is:

$$C(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{36}$$

Assuming that the external input via the transmission matrix $D(t)$ has no influence via the input on the displacement but on the velocity, then:

$$D(t) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{37}$$

3.2.2 Discrete-Time State Space Model

A discrete-time SSM represents a time dynamic system as a set of first order difference equations, which describe the relationship between the values of the system at different points in time [57; 58]. They map a 1-D input signal $\mathbf{u}_t$ to an N-D latent space $\mathbf{x}_t$, which can then also be projected back to a 1-D output signal $\mathbf{y}_t$ as shown in Eq. (38) and Eq. (39).

$$\mathbf{x}_{t+1} = A_t\mathbf{x}_t + B_t\mathbf{u}_t \tag{38}$$
$$\mathbf{y}_t = Cx_t + D_t\mathbf{u}_t \tag{39}$$

where:

- $\mathbf{x}_t \in \mathbb{R}^n$ is the state vector of size n at time-step t.
- $\mathbf{u}_t \in \mathbb{R}^m$ is the input vector of size m at time-step t.

- $\mathbf{y}_t \in \mathbb{R}^p$ is the output vector of size p at time-step t.
- $A_t \in \mathbb{R}^{n \times n}$ is the system matrix at time-step t, representing how the state variables change with respect to time in response to the system dynamics.
- $B_t \in \mathbb{R}^{n \times m}$ is the input matrix at time-step t, representing the influence of external forces on the system dynamics.
- $C_t \in \mathbb{R}^{p \times n}$ is the output matrix at time-step t, relating the state vector to the output vector.
- $D_t \in \mathbb{R}^{p \times m}$ is the transmission matrix at time-step t, representing the direct influence of the input vector on the output vector.

Re-examining the spring-mass system in Fig. 10 to exemplify a discrete-time SSM, the system's behavior is still described using a set of state variables and state equations. The state variables represent the minimal set of information that describes the system at any given time, and the state equations express how these state variables change as a function of time using a set of first order difference equations. To describe the motion of the mass in our spring-mass system using state space representations, the two state variables remain unchanged and are: 1) the displacement $\mathbf{x}(t)$ of the mass from its equilibrium position, defined as the position of the mass along the x-axis at time t, and 2) $v(t)$, the velocity of the mass, defined as the rate of change of the position over time. Representing these two variables in a vector, we obtain the state vector $\mathbf{x} = [x(t), v(t)]$. The state equations for the spring-mass system are then scaled by the discretization rate Δt of the continuous form at time-step t, meaning that the state equation can be formulated in matrix form as:

$$\begin{bmatrix} x_{t+1} \\ v_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ -\frac{k\Delta t}{m} & 1 - \frac{c\Delta t}{m} \end{bmatrix} \begin{bmatrix} x_t \\ v_t \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\Delta t}{m} \end{bmatrix} u_t, \tag{40}$$

where:

- $\mathbf{x}_t \in \mathbb{R}^n$ is the state vector of size n at time-step t.
- $\mathbf{u}_t$ is the external force or the input vector of size m at time-step t.
- $A_t \in \mathbb{R}^{n \times n}$ is the state transition matrix, representing how the state variables change with respect to time in response to the system dynamics.
- $B_t \in \mathbb{R}^{n \times m}$ is the input matrix at time-step t, representing the influence of external forces on the system dynamics.
- Δt is the time-step size over which we discretize a continious signal.
- k and m are the spring constant and mass, respectively.

The matrices A_t, B_t, C_t, and D_t vary with time t in discrete-time SSMs, modelling evolution of the dynamic system over time-steps.

3.2.3 Discrete State Space Models as Polynomials

To see the relationship between SSMs and polynomials, the state equation can be expanded over multiple time-steps, where the system is assumed to be time-invariant (i.e, A and B are constant and usually pre-computed). Consider for simplicity a scaler case with the state equation as followed at every time-step t:

$$x_{t+1} = ax_t + bu_t \tag{41}$$

Expanding Eq. (41) over multiple time-steps:

$$\begin{aligned} x_{t+2} &= ax_{t+1} + bu_{t+1} && (42) \\ &= a(ax_t + bu_t) + bu_{t+1} && (43) \\ &= a^2 x_t + abu_t + bu_{t+1} && (44) \end{aligned}$$

$$x_{t+3} = ax_{t+2} + bu_{t+2} \tag{45}$$
$$= a(a^2x_t + abu_t + bu_{t+1}) + bu_{t+2} \tag{46}$$
$$= a^3x_t + a^2bu_t + abu_{t+1} + bu_{t+2} \tag{47}$$

In general, the state x_t can be represented as a polynomial $P(x)$:

$$P(x) = x_{t+n} = a^n x_k + a^{n-1}bu_t + a^{n-2}bu_{t+1} + \cdots + abu_{t+n-2} + bu_{t+n-1} \tag{48}$$

3.2.4 High-Order Polynomial Projection Operator

High Order Polynomial Projection Operator (HiPPO) utilizes the convenient formulation of an SSM as a polynomial as shown in Eq. 48, to projects a 1-D dimensional input onto a set of orthogonal polynomial basis functions. That is given an input function $f(t) \in \mathbf{R}$ over time t, HiPPO aggregates the time-dynamics or history of $f(t)$ onto an N-dimensional latent-space. Since the latent-space is constructed using polynomial basis function, the parameter N corresponds to the N-th order polynomial. As a result, the latent-space which describes the time-dynamics of a system is represented using the corresponding coefficient $\mathbf{c}(t)$ of the polynomial functions. The general HiPPO framework can then be summarised as followed: 1) for any function f, 2) at every time-step t HiPPO projects $f(t)$ onto a sub-space of orthogonal polynomials, 3) for which the corresponding polynomial coefficients $\mathbf{c}_t \in \mathbf{R}^N$ represent the compressed history of f, 4) leading to a time-dynamic recurrent relationship between consecutive time-steps by formulating the problem as an SMM as shown in Eq. 49:

$$\mathbf{c}_{t+1} = A_t\mathbf{c}_t + B_t\mathbf{u}_t \tag{49}$$

where:

- $\mathbf{c}(t) \in \mathbb{R}^n$ are the coefficients of a N-th degree polynomial for the SSMs latent-space $x'(t)$ in Eq. 38 at time-step t.
- $\mathbf{u}(t) \in \mathbb{R}^n$ is the input vector at time-step t, represented as $u(t)$ in the SMM Eq. 38.
- $A(t) \in \mathbb{R}^{n\times n}$ is the system matrix at time-step t, representing how the state variables change with respect to time in response to the system dynamics.
- $B(t) \in \mathbb{R}^{n\times m}$ is the input matrix at time-step t, representing the influence of external forces or the inputs influence on the system dynamics.

Note that the formulation in Eq. (49) describes a time-variant dynamic model, where A and B change over t, even-though the HiPPO operation can also be formulated as a time-invariant SSM, where A and B are held constant across time. Regardless of its formulation, as HiPPO defines an orthogonal projection, it addresses the exploding and vanishing gradient problem, since the magnitude of the eigenvalue with the biggest influence is 1 (see Appendix VIII.i for proof).

Due to its recurrent formulation and its orthogonal projection, HiPPO can be integrated into an RNN to replace update mechanisms that have been shown to suffer from gradients problem. That is HiPPO can be used as a gating mechanism for any RNN, for example an LSTM as shown in Fig. 11. The HiPPO LSTM cell takes as input the cell-state $\mathbf{c}_{t-1}$ from the previous time-step, the hidden-state $\mathbf{h}_{t-1}$ from the previous time-step and the input $\mathbf{x_t}$ at the current time-step to produce a new hidden-state $\mathbf{h}_t$ and cell-state $\mathbf{c}_t$ by concatenating the input $\mathbf{x}_t$. The hidden-state is updated by using a function τ, which can be any RNN update function i.e., a simply function f_W that is parameterized by weights W and processes the concatenated input $\mathbf{x}_t$ with the cell-state $\mathbf{c}_{t-1}$. The resulting new hidden-state h_t is processed by multi-layer-perceptron to produce a scaler value f_t, representing a single time-point t within a continuous function and passed onto the $hippo_t$ operator to produce a new cell-state that now represent the coefficients for an Nth order orthogonal polynomial. Formally this recurrent relationship can be expressed as:

$$\mathbf{h}_t \in \mathbb{R}^N = \tau(\mathbf{h}_{t-1}, [\mathbf{c}_{t-1}, x_t]) \tag{50}$$
$$\mathbf{f}_t \in \mathbb{R}^1 = f_W(\mathbf{h}_t) \tag{51}$$
$$\mathbf{c}_t \in \mathbb{R}^N = hippo_t(f_t) \tag{52}$$
$$= A_t \mathbf{c}_{t-1} + B_t f_t, \tag{53}$$

where:

- $\mathbf{h}_t \in \mathbb{R}^p$ is the hidden-state of size N, denoting N as the hidden-size due to the N coefficients for the Nth order polynomials of hippo.
- $\mathbf{f}_t \in \mathbb{R}^1$ is a scalar produced by any function f_W parameterised by weights W.
- $\mathbf{c}_t \in \mathbb{R}^N$ is the vector of N coefficients from the $hippo_t$ operator at ever timestep t

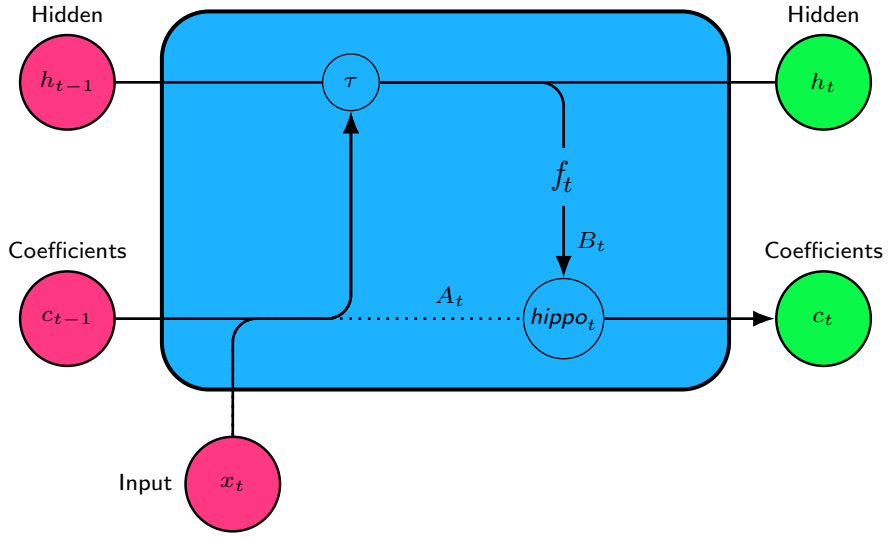


Figure 11: Gating mechanism of a HiPPO LSTM cell. The cell takes as input the input at the current time-step t, the hidden-state at the previous time-step $t-1$ and cell-state of the previous time-step $t-1$, returning the hidden-state and cell-state of the current time-step.

4 Method

We use a YoloV4 framework for both still-image and video object detection. In particular we train YoloV4 on still-image object detection, using the MS COCO2017 [60] dataset. The resulting model is then used to fine-tune three models: 1) YoloV4, 2) YoloV4 + URLSTM (UrYoloV4) and 3) YoloV4 + HippoLSTM (HoloV4) on video object detection, using the ImageNet VID 2015 [61] dataset. To validate the selected sequence models for the video object detection task, they are evaluated on their ability to model temporal information on the sequential MNIST task [62]. For all sequence models that involve the HiPPO SSM, we use Legendre Polynomials, which have been shown to be particularly effective at summarizing long-range dependencies [52]. Like other polynomials, they are orthogonal (see Appendix VIII.ii for proof), ensuring that the magnitude of the biggest eigenvalue is 1. The Legendre SSM recurrence is represented as:

$$\mathbf{c}_{t+1} = -\frac{1}{t} A_t \mathbf{c}_t + \frac{1}{t} B_t \mathbf{u}_t \tag{54}$$
$$A = \begin{cases} \sqrt{(2n+1)(2t+1)} & \text{if } n > t \\ n+1 & \text{if } n = t \\ 0 & \text{if } n < t \end{cases} \tag{55}$$
$$B = \sqrt{2n+1} \tag{56}$$

where the corresponding A and B matrices are defined as above and are scaled with respect to time, resulting in a set of scaled Legendre or LegS polynomials. For more details, we refer to [52].

4.1 YoloV4

YoloV4 like its predecessors belongs to the family of regression-based object detection frameworks. Rather than processing and locating potential regions of interest independently, the idea is to solve a regression problem in which regions of interest and predictions are made at once, using a convolutional network. That is, given an input image, the image is divided into a grid $S \times S$ (i.e., a 19×19 grid), where a fixed number of base bounding box predictions B are assumed to be in each grid cell. Then for each grid cell and each bounding box the model predicts: 1) an offset to the base bounding box to predict the true location of the object and 2) a confidence/classification score of how likely it is that an object of category C appears in the bounding box. The final output tensor is of size $S \times S \times (5 \times B + C)$, where B denotes the number of base bounding boxes, C the classification scores for each classification label and S the number of grids along a single axis. A generalized workflow of YoloV4 can be seen in Figure 12.

4.1.1 Architecture

YoloV4 utilizes a network that is called Darknet-53, which consists of 53 convolutional layers pretrained on ImageNet classification, followed by co-adaptation layers, a neck consisting and a prediction head consisting of 2 convolutional layer making predictions for each one of the three scales: $76 \times 76\times$, $38 \times 38\times$ and 19×19 [9]. A detailed depiction of the network architecture can be seen in Figure 12 with each component being labeled and roughly sketched out. We deviate slightly from the original architecture by using an EfficientNetS [63] backbone. Apart from this slight deviation, our implementation of the YoloV4 architecture stays as close as possible to the original. That is, the neck consists of a spatial pyramid pooling (SSP) [64] component, a Path Aggregation Net (PANet) [65] and a final prediction head. By passing the tensors from the Co-adaptation block into the neck, the spatial pyramid pooling component, pools features from different scales, which helps to capture fine and coarse information (i.e., coarse from the smallest scale 19 and fine from the largest scale 76). The pooled features are then passed into the PANet and aggregated with features from the backbone, effectively creating skip connections. This allows the network to utilize both information from earlier layers and previous layers. The final output from the PANet is then passes onto each of the responsible prediction heads, resulting in three output tensors, one for each of the three scales.

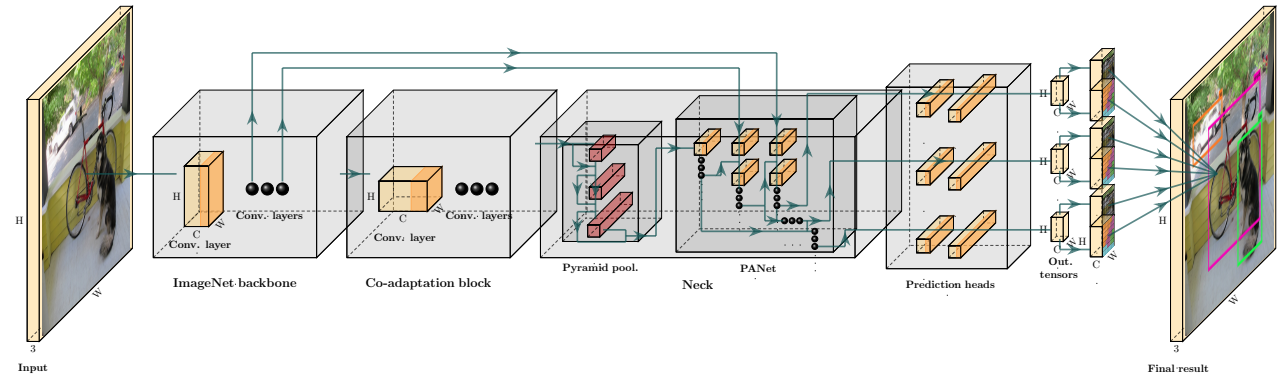


Figure 12: YoloV4 workflow for video object detection. The linear layer from YoloV1 has been replaced by three convolutional prediction heads, each making predictions at different grid sizes or scales. The network consists of four parts: 1) an ImageNet backbone, 2) a co-adaptation block, 3) a neck and 4) a predictions head, making predictions at three scales (76, 38, 19).

4.1.2 Loss

YoloV4 formalises the optimisation landscape through multiple sums of squared errors (SSE) for localisation (i.e., x, y, height and width of bounding boxes). Cross entropy is used for classifying objects to their appropriate class category. Even-though, multiple SEE terms are easy to optimize, they do not perfectly align with the goal of maximizing (mean) average precision, because SEE will weight different localization objectives equally to each other. Additionally, since many of the grids

in the images will not contain any object, confidence score of these cell may be pushed to zero, which can cause gradient instability and early divergence. As a result, gradients from cells that do contain objects can be masked. To remedy this, the loss components are weighted differently, using parameters $\lambda_{class} = 1$, $\lambda_{box} = 8$, $\lambda_{noobj} = 4$ and $\lambda_{obj} = 2$ respectively.

Furthermore, SEE also weights errors in large boxes and small boxes equally, which may be problematic as the loss should reflect that small deviations in large boxes are less important than small deviation in small boxes. That is, a 6-pixel deviation on a 600-pixel wide box should have less of an effect on the loss than a 6-pixel deviation in a 600-pixel wide box. To resolve this issue, the loss applies the square root onto the bounding box width and height predictions. As a result, the square root downscales high values while having less of an effect on smaller width and height values.

A closer examination of the loss in Eq. 58 illustrates that the first two loss component are part of the 1) localisation aspect and the last 3 lines are part of the 2) classification aspect. By using the identity function I also referred to as the object mask or identity, YoloV4 selectively allows certain parts of the loss to be computed based on whether an object is present in a grid cell or not. The identity function I, which will be 1 if an object is in cell i and if bounding box j is responsible for predicting the bounding box. Otherwise it will be zero. Note that the predicted bounding box j is responsible for the ground truth box if it has the highest Intersection over Union (IoU) for any predictors in that grid cell i (see Fig. 13). Consequently, there is only a loss if there is an object in cell i given bounding box j is responsible.

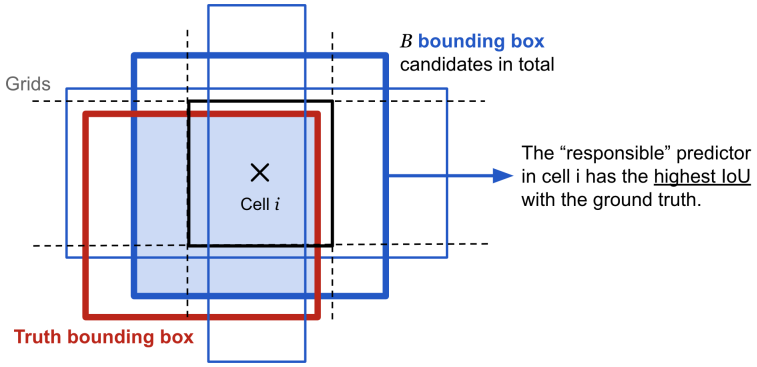


Figure 13: Given cell i shown as a black square, the model proposes B number of bounding box candidates. The bounding box candidate that has the highest intersection over union with the ground truth bounding box is the responsible predictor.

In more detail, in the loss we iterate over each grid from i to S^2 and bounding boxes j to B, determine via the identity I whether an object is in the cell i and if bounding box j is responsible for the prediction and compute the squared difference between the true x, y and predicted $\hat{x}, \hat{y}$ coordinates. This is repeated for the square root of the true w, h and predicted $\hat{w}, \hat{h}$. Yolo also makes predictions about confidence C_{ij}, which reflects how likely it is that an object is in cell i and how accurate the bounding box predictions j are. This is accomplishes by weighting the probability of an object or no object being in the cell i, taking the IoU of the predicted and true bounding box into account. The subsequent loss component takes the squared difference between predicted confidence score $\hat{C}_{ij}$ and true confidence score C_{ij}. Lastly, the loss addresses the classification problem. Given that there is an object as determined by the identity I then being equal to one, we iterate over each class C and take the difference between the true class probability $p_i(c)$ and predicted class probability $\hat{p}_i(c)$.

$$
\begin{aligned}
\underbrace{L(\hat{y}, y)}_{\text{loss}} = \underbrace{\lambda_{\text{box}}}_{\text{constant}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \underbrace{I_{ij}^{obj}}_{\text{identity}} \underbrace{\left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right]}_{\substack{\text{error term} \\ \text{bounding box} \\ \text{midpoint coordinates}}} \\
+ \underbrace{\lambda_{\text{box}}}_{\text{constant}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \underbrace{I_{ij}^{obj}}_{\text{identity}} \underbrace{\left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2\right]}_{\substack{\text{error term} \\ \text{bounding box} \\ \text{height and width}}}
\end{aligned}
\tag{57}
$$

$$
\begin{aligned}
&+ \underbrace{\lambda_{\text{obj}}}_{\text{constant}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \underbrace{I_{ij}^{obj}}_{\text{identity}} \underbrace{\left(C_{ij} - \hat{C}_{ij}\right)^2}_{\substack{\text{error term} \\ \text{confidence scores}}} \\
&+ \underbrace{\lambda_{\text{noobj}}}_{\text{constant}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \underbrace{I_{ij}^{\text{noobj}}}_{\text{identity}} \underbrace{\left(C_{ij} - \hat{C}_{ij}\right)^2}_{\substack{\text{error term} \\ \text{confidence scores}}} \\
&+ \underbrace{\lambda_{\text{class}}}_{\text{constant}} \sum_{i=0}^{S^2} I_i^{obj} \sum_{c \in \text{ classes}} \underbrace{(p_i(c) - \hat{p}_i(c))}_{\substack{\text{error term} \\ \text{conditional class} \\ \text{probabilities}}},
\end{aligned}
$$

where:

- I_i^{obj} is an identity function of whether the cell i contains an object.
- I_{ij}^{obj} is an identity function of whether the j-th bounding box of the cell i is "responsible" for the object prediction.
- C_{ij} is the confidence score of cell i given by P(containing an object) × IoU(pred, truth),
- $\hat{C}_{ij}$ is the predicted confidence score, C denotes the set of all category classes (for MS COCO $C = 80$, for ImageNet VID $C = 30$).
- $p_i(c)$ denotes the conditional probability of whether cell i contains an object of class c.
- and $\hat{p}_i(c)$ denotes the predicted conditional class probability.

4.2 Dataset

4.2.1 Modified National Institute of Standards and Technology Database

The modified National Institute of Standards and Technology database (MNIST) [62] is a large collection of handwritten digits. It has a training set of 60,000 examples, and a test-set of 10,000 examples. The image examples are monochrome (i.e., one color channel) and have a size of 28×28 pixels.

4.2.2 Microsoft Common Objects in Context Dataset

The Microsoft Common Objects in Context (MS COCO) dataset [60] is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of 328K images in total. The object recognition subset from 2017 used for experiments consists of 118,000 training examples and 5,000 examples, annotated with bounding boxes, per-instance segmentation mask and class categories for 80 object categories.

4.2.3 ImageNet Video Dataset

ImageNet Video (VID) dataset [61] is a large-scale public dataset for video object detection contains more than 1 Million frames for training and more than 100,000 frames for validation, annotated with bounding boxes and class categories for 30 object categories. The frames from the ImageNet VID 2015 are extracted from around 3862 videos from the training set and 555 videos from the validation set. Sequence length (i.e., number of frames per video) range from 6 to 5492 frames per video.

5 Experiments

5.1 Sequence Models

Sequence models are validated on the sequential MNIST task [62], a context task in which a model needs to keep an internal memory of previous inputs over a total of $T = 28 \times 28$ consecutive time-steps. The 28×28 pixels of an MNIST images are flattened into a single vector, each index now representing a single pixels or time-step t. The length of the vector then represents the total number of time-steps $T = 28 \times 28$. At each time-step t the model receives a single input x_t until the entire sequence T has been processed. Each model is trained 5 times. A single hidden to hidden layer was used for each model, defining shallow RNNs interns of their depth. The mean and standard

deviation is computed across all 5 repetitions. The two highest performing RNNs in-terms of class accuracy are then selected for video object detection. Training parameters after hyperparameter tuning can be found in Table 1.

Table 1: Sequence model hyperparameters used for training.

Model	hidden size	learning rate	weight decay	momentum	epochs	batch-size	scheduler	optimizer
RNN	512	$1e-5$	0.0	0.9	50	64	expon. decay	Adam
GRU	512	$1e-5$	0.0	0.9	50	64	expon. decay	Adam
LSTM	512	$1e-5$	0.0	0.9	50	64	expon. decay	Adam
UR-LSTM	512	$1e-3$	0.0	0.9	50	64	constant	Adam
HiPPO	512	$1e-3$	0.0	0.9	50	64	constant	Adam
HiPPO-LSTM	512	$1e-3$	0.0	0.9	50	64	constant	Adam

5.1.1 Results

Results for the mean accuracy and standard deviation, computed after 5 repetitions on the sequential MNIST task can be seen in Table 2. Values of the highest performing models are highlighted in boldface. Graphically, these results can be examined in Fig 14-15. The HiPPO-LSTM has the highest accuracy of 0.974 and smallest standard deviation 0.0001. A regular HiPPO SSM has the second highest accuracy and stability with values 0.919 and 0.0003 respectively. The UR-LSTM with an accuracy of 0.637 has the third highest accuracy, but the largest standard deviation 0.430 across all models. Since HiPPO by itself is not an RNN, the UR-LSTM and HiPPO-LSTM are used for subsequent experiments in video object detection.

Table 2: Mean accuracy and standard deviation (SD) across 5 repetitions on the SMNIST task.

Model	Mean	SD
RNN	0.207	0.109
GRU	0.350	0.122
LSTM	0.104	0.008
UR-LSTM	**0.637**	**0.430**
HiPPO	**0.919**	**0.0003**
HiPPO-LSTM	**0.974**	**0.0001**

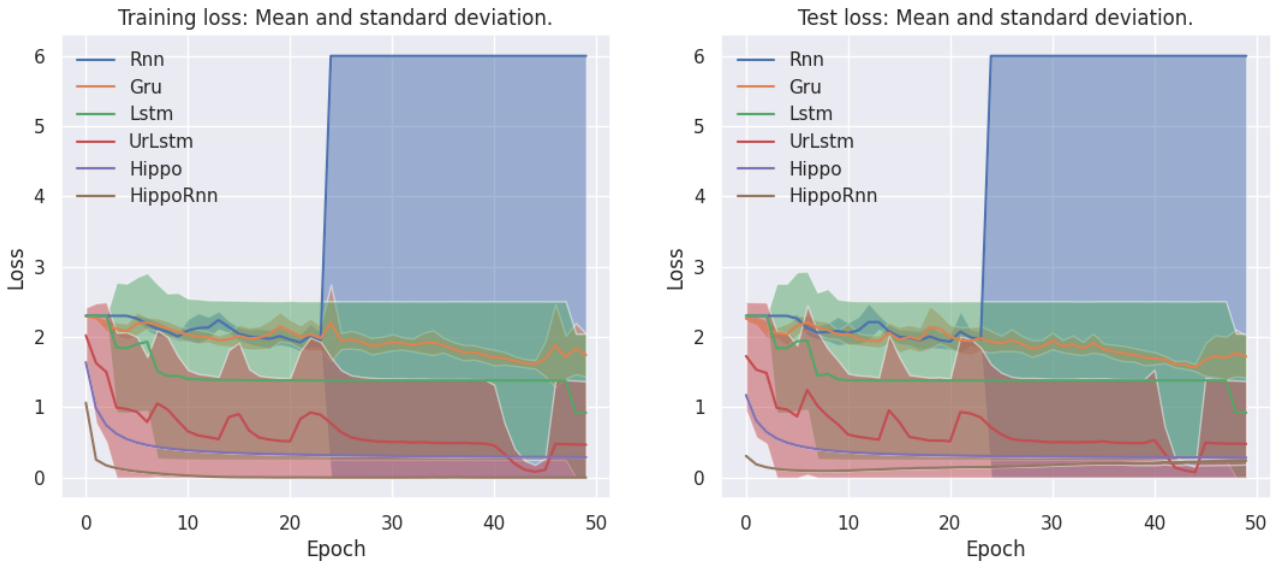


Figure 14: Train vs. test loss as a function of epoch across different recurrent neural networks with $1\times$ standard deviation represented as shaded areas. Loss values are bounded above at 6.0 for graphical illustration.

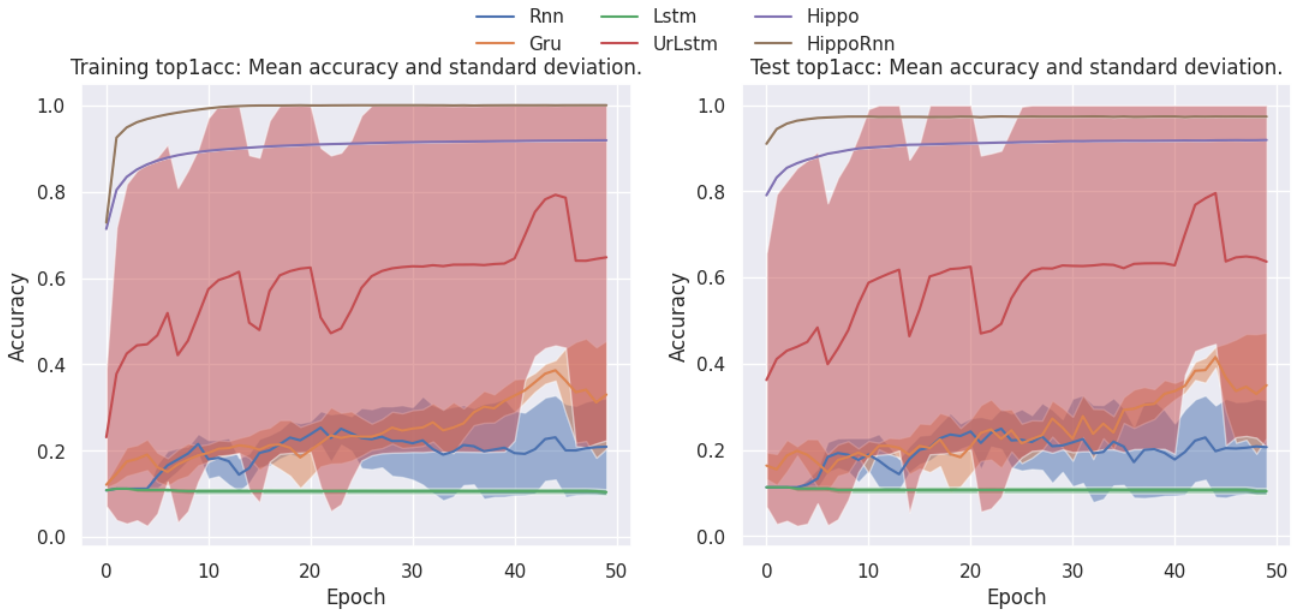


Figure 15: Train vs. test accuracy as a function of epoch across different recurrent neural networks with $1\times$ standard deviation represented as shaded areas.

5.2 Still-Image Object Detection

Since the still-image detector is used to initialize the video-object detector in subsequent experiments, multiple repetition are not necessary. The class accuracy, object accuracy, no object accuracy and the mean average precision (mAP) with an intersection over union threshold $\alpha = 0.5$ (mAP@0.5) are computed every 10th epoch. The mAP was not computed for the training-dataset as bounding box predictions quickly become too large to obtain evaluation results within a reasonable time (i.e., given a dataset of 100K images with multiple objects per image, the total ground truth values and the number of predictions quickly grow to be too large).

The input image is resized to have a square height and width of 608. To increased the dataset size, improve generalization, introduce more variation in object appearance, increase robustness to noise and increase the models spatial in-variance (i.e., rotation in-variance), various image augmentations are applied during training. Bounding boxes are augmented accordingly. Augmentation are applied with a probability of p. The following augmentations with probability p are applied: 1) scaling: $p_s = 1.0$, 2) translation: $p_t = 1.0$, 3) HSV: $p_{hsv} = 1.0$, 4) rotation $p_r = 0.3$, 5) horizontal flipping $p_{hflip} = 0.3$, 6) 4 tiled and 9 tiled mosaic: $p_{mosaic} = 0.3$, 7) mixup: $p_mix = 0.3$, 8) grey-scaling: $p_g = 0.1$, 9) blurring $p_b = 0.1$, 10) CLAHE $p_{clahe} = 0.1$, 11) random channel shuffle: $p_{cs} = 0.1$ and 12) posterize: $p_p = 0.1$.

Gradient accumulation is used to simulate a target batch-size of 128, while the actual batch-size on GPU remained 32 to save memory. Training parameters after hyperparameter tuning can be found in Table 3.

Table 3: YolOV4 hyperparameters used for training.

Model	learning rate	weight decay	momentum	epochs	batch-size	scheduler	optimizer	λ_{class}	λ_{obj}	λ_{noobj}	λ_{box}
YoloV4-608	$1e-3$	0.0005	0.937	300	32	one cycle	Adam	1.0	2.0	4.0	8.0

5.2.1 Results

The results for YoloV4 on the MS COCO 2017 dataset can be seen in Table 4, where class accuracy, object accuracy, no object accuracy, mAP@0.5, recall and precision is depicted. An example of the trained detection model on a test image can be seen in 16. Bounding box predictions that have an IoU of less than 0.5 with the true bounding box and predictions with a confidence score lower than

0.8 were discarded, when evaluation the model using these metrics. On the test-dataset, The YoloV4 model reaches a mAP@0.5 = 0.478, recall= 0.561, precision= 0.587. Class accuracy is 0.785, object accuracy is 0.449 and no object accuracy is 0.997.

Table 4: Results for the YoloV4 object detector after training.

Metric	Train	Test
class accuracy	0.899	0.785
object accuracy	0.501	0.449
no object accuracy	0.996	0.997
mAP@0.5	–	**0.468**
Recall	–	0.561
Precision	–	0.587

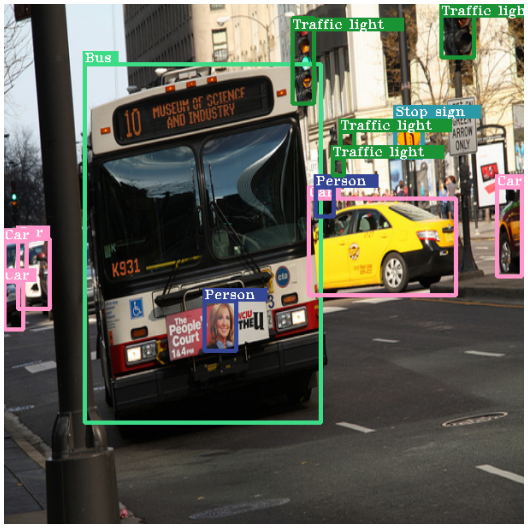


Figure 16: YoloV4 detections on a test still-image from the MS COCO 2017 dataset.

5.3 Video Object Detection

Since YoloV4 makes predictions at 3 scales (i.e., 76, 38 and 19), the networks final feature map rapidly increase in size. At scale 76, the final output tensor has a size of $76 \times 76 \times 105$. At scale 38 the output tensor has a size of $38 \times 38 \times 105$ and at scale 19 $19 \times 19 \times 105$. Due to GPU memory constraints, processing the entire flattened tensor at scale 76 and 38 becomes infeasible. As such we 1) only retain aspects of the output tensor responsible for bounding box predictions (i.e., x, y, height, width coordinates) and 2) eliminate two out of three prediction heads, only retaining the prediction at the smallest scale 19.

To ensure that every $10th$ frame can be sampled from a video without exceeding GPU memory limits, experiments are performed on a subset of the ImageNet Vid 2015 dataset, where videos with a length longer than 80 frames are excluded. Videos that have less than 80 frames and do not allow for an even sampling strategy of every 10th frame using a sequence length of 8, are padded by repeating the parts of the sequence sampling was possible from. For example a video of 40 frames would with a sequence length of 8, result in sampling $40/8 = 5$ frames instead of 10. The missing 5 images are then created by padded the image vector by repeating the sequence of previous frames.

The input image is resized to have a square height and width of 608. The following augmentations with probability p are applied: 1) HSV: $p_{hsv} = 1.0$, 2) grey-scaling: $p_g = 0.1$, 3) blurring $p_b = 0.1$, 4) CLAHE $p_{clahe} = 0.1$, 5) random channel shuffle: $p_{cs} = 0.1$ and 6) posterize: $p_p = 0.1$.

For the Yolo that utilizes an UR-LSTM (UR-Yolo), a deep UR-LSTM with 6 hidden to hidden layers is used. Less lead to performance degradation and more did not yield better results during fine-tuning. For the Yolo variant that utilizes HiPPO (Holo), stacking layers on-top of each other did not result in noticeable performance difference, so a single layer was used.

Gradient accumulation is used to simulate a target batch-size of 128, while the actual batch-size on GPU remained 32 to further save memory. The class accuracy, object accuracy, no object accuracy and the mean average precision (mAP) with an intersection over union threshold $\alpha = 0.5$ (mAP@0.5) are computed every 10th epoch for both the training and test-dataset.

5.3.1 Results

Evaluation metrics for both the training and test-dataset are shown in Table 6 and 7 respectively. The mean and standard deviation is computed across 3 repetitions. The Yolo baseline has the highest mAP@0.5 = 0.376 on the test-data and the smallest standard deviation 0.004. The UrYolo

Table 5: YoloV4 video hyperparameters used for training.

Model	learning rate	weight decay	momentum	epochs	batch-size	scheduler	optimizer	λ_{class}	λ_{obj}	λ_{noobj}	λ_{box}
YoloV4-608	$1e-3$	0.0005	0.937	100	32	one cycle	Adam	1.0	2.0	4.0	8.0
UrYoloV4-608	$1e-3$	0.0005	0.937	100	32	one cycle	Adam	1.0	2.0	4.0	8.0
HoloV4-608	$1e-3$	0.0005	0.937	100	32	one cycle	Adam	1.0	2.0	4.0	8.0

Note: 608 denotes the height and width of the input image.

has a mAP@0.5 = 0.308 and a standard deviation of 0.013. Holo has the same standard deviation 0.013 as the UrYolo variant, but the lowest mAP@0.5 = 0.098.

The mAP@0.5 can be graphical examined as it evolves over the number of epochs in Fig. 18. Similarly for the loss in Fig. 17. Note that since evaluation metrics are computed every 10th epoch for the entire training process of a 100 epochs, x-axis values are from $0-10$, but represent values from $0-100$ at every 10th epoch.

Figure 17: Train vs. test loss as a function of epoch $\times 10$ across different recurrent Yolo networks with $1\times$ standard deviation represented as shaded areas.

Table 6: Metrics on the training-dataset: Mean mAP@0.5 and standard deviation (SD) across 3 repetitions on the ImageNet VID dataset after training. Values are rounded to 3 decimal points.

Model	Mean_{mAP}	SD_{mAP}	Mean_{clsacc}	SD_{clsacc}	Mean_{objacc}	SD_{objacc}	$\text{Mean}_{nobjacc}$	$\text{SD}_{noobjacc}$	Mean_{prec}	SD_{prec}	Mean_{rec}	SD_{rec}
YoloV4	**0.935**	0.004	1.0	0.0	0.993	0.005	0.997	0.0	0.948	0.009	0.941	0.007
Ur-YoloV4	**0.883**	0.024	1.0	0.0	0.945	0.004	0.998	0.0	0.757	0.018	0.926	0.018
HoloV4	**0.372**	0.042	1.0	0.0	0.307	0.043	0.999	0.0	0.853	0.0265	0.397	0.044

Table 7: Metrics on the test-dataset: Mean mAP@0.5 and standard deviation (SD) across 3 repetitions on the ImageNet VID dataset after training. Values are rounded to 3 decimal points.

Model	Mean_{mAP}	SD_{mAP}	Mean_{clsacc}	SD_{clsacc}	Mean_{objacc}	SD_{objacc}	$\text{Mean}_{nobjacc}$	$\text{SD}_{noobjacc}$	Mean_{prec}	SD_{prec}	Mean_{rec}	SD_{rec}
YoloV4	**0.376**	0.004	0.312	0.034	0.342	0.024	0.998	0.0	**0.687**	0.053	**0.391**	0.002
Ur-YoloV4	**0.308**	0.013	0.277	0.033	0.332	0.029	0.998	0.0	**0.455**	0.011	**0.365**	0.016
HoloV4	**0.098**	0.013	0.260	0.038	0.088	0.014	0.999	0.0	**0.448**	0.02	**0.116**	0.024

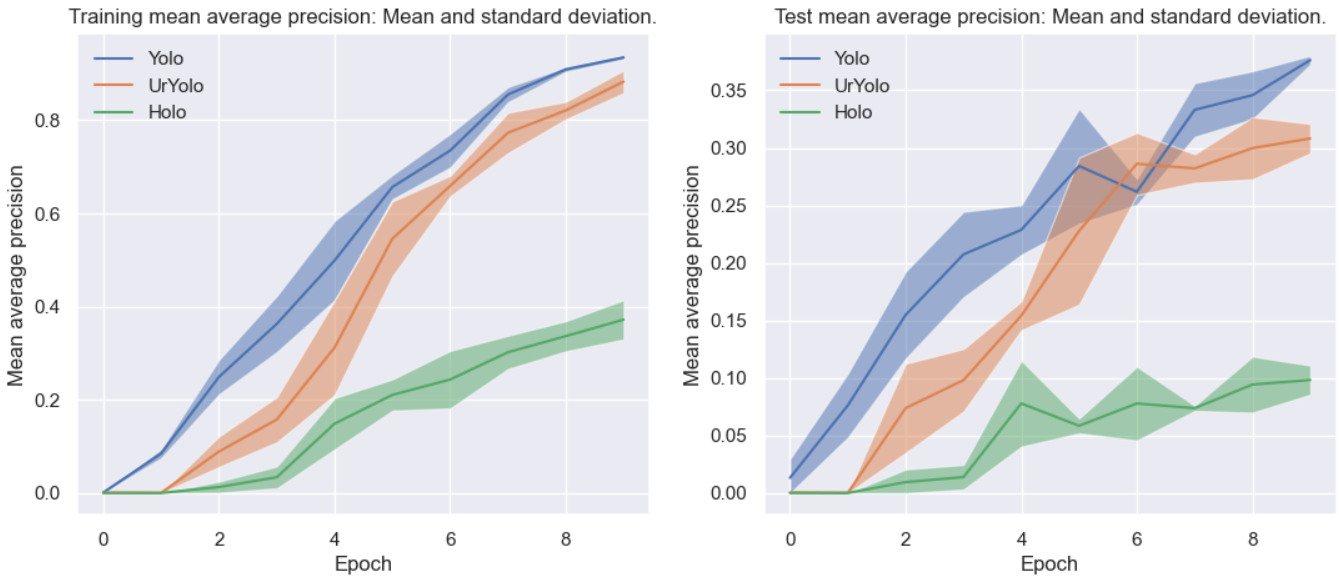


Figure 18: Train vs. test mean average precision as a function of epoch $\times 10$ for different recurrent Yolo networks with $1\times$ standard deviation represented as shaded areas. Values are rounded to 3 decimal points.

6 Discussion

Firstly, in our validation experiments regarding sequence models, a HiPPO-LSTM outperforms all other models with an accuracy of 97.4%, followed by a regular HiPPO approach with an accuracy of 91.9% and a UR-LSTM with an accuracy of 63.7%. All other RNNs like a regular LSTM, GRU or RNN have an accuracy below 40.0%. These results are also reflected in the standard deviations in Table 2, where the HiPPO-LSTM and HiPPO have the smallest standard deviation. HiPPO seems to introduce stability to the solution. Even-though, the UR-LSTM has a more desirable performance as compared to a regular LSTM, solutions are unstable as visible by the large standard deviations. Examining the solution in terms of the loss in Fig. 14 and 15 in more detail, similar conclusion can be drawn. The stability of HiPPO on the optimization landscape is visible. HiPPO models converge and minimize loss faster than other examined competitors. While the UR-LSTM has the largest standard deviation on the mAP as shown in in Table 2, loss as depicted in 14 is still lower than regular RNN variants. That is, even-though the UR-LSTM has the largest standard deviation, its solution with respect to loss is to be preferred. In some cases, as visible by the shaded area for the test accuracy in Fig. 15, the UR-LSTM may reach an accuracy that is larger than HiPPO's. It may however also have an accuracy worse than a regular LSTM in some repetitions. Overall, this set of experiments show the incremental improvement of RNNs in their ability to memorize context.

Secondly, our still-image object detector YoloV4 reaches a mAP of 46.8%, which to our knowledge is slightly better than the other implementation (45.5%) of YoloV4 that utilizes the same backbone. From table Table 4, we observe that for 80 class categories, the model has a 78.8% percent accuracy to make the right classification. The no object accuracy is 99.7% and the object accuracy is at 44.9%. That is the model is better at prediction no objects, where there are no object, but not as good at prediction objects where there are objects. This is to be expected, since the distribution of objects vs. no objects with respect to a cell is uneven. Furthermore, our λ values that weight each loss component differently, has a higher emphasis on λ_{noobj}, limiting the number of false positives (i.e., predicting an objects where there is no object). The recall of 0.561 indicates that the model correctly identifies 56.1% of all actual positive cases. The proportion of true positives, true negatives, false positives and false negatives is reflect in the recall and precision. The recall of 56.1% indicates that the model correctly identifies 56.1% of all actual positive cases (true positives out of the sum of true positives and false negatives). The precision of 58.7% means that 58.7% of the instances predicted as positive by the model are indeed positive (true positives out of the sum of true positives and false positives).

Lastly, we find that giving an object detector access to temporal information through bounding box trajectories using Yolo leads to performance degradation. This is particularly true for HiPPO, which compared to the Yolo baseline, decrease by 27.8% in terms of mAP. Even-though utilizing UrYolo to encode temporal information has a smaller decrease on performance compared to Yolo, the model still performs 6.8% worse. The performance degradation are also reflected in the standard deviation of the mAP as shown in Tables 6 - 7. This is harder to observe graphical in Fig. 18. However at the last epoch, Yolo has the smallest standard deviation (0.004) as compared to UrYolo (0.013) and Holo (0.013). Yolo and UrYolo are closer in performance to each other, both with respect to training loss in Fig. 17 and training mAP in Fig. 17. With respect to test loss and train loss they start to deviate from each other. This would indicate, that UrYolo does not generalize as well as Yolo, even-though training loss and training mAP are close to each other. Holo does not generalize well on the test-data as well and has the lowest performance as visible by loss and map on both the training and test-data.

Recall that, Holo, which utilizes the HiPPO framework maps the information from the bounding box trajectories onto a hidden-state $\mathbf{h}_t$, which is then compressed into a scalar f_t by an MLP f. Using an MLP to compress a vector of length L that has encoded the bounding box trajectory into a scalar, may not be ideal and seems to cause a great loss of information. This may be the cause of the performance degradation. However, adapting the MLP by either stacking multiple layers on-top of each other or compressing bounding box information to a vector of length $L > 1$ did not yield performance benefits. Even-though Holo has the lowest performance, the model does not go down a path of non-sensible predictions i.e., no bounding box predictions at all. That is HiPPO is learning, however converging at a sub-optimal location in the optimization landscape and not scaling. This can be seen by the precision. On the test-data, Holo has a similar precision of 44.8% to UrYolo, which has a precision of 45.5%. That is 44.8% of the instances predicted by Holo as positive are actually positive. While not as good as Yolo with a precision of 68.7%, the model did learn. The same argument can however not be made when examining recall. Holo has the lowest recall of 11.6% among all models. That is Holo only correctly identifies 11.6% of all actual positive cases as positive, missing a lot of detection. To compare, UrYolo identifies 36.5% of positive cases as positive, missing 63.5%. This is closer to the Yolo baseline, which has a recall of 39.1%, missing 61.9% of all possible detections.

7 Conclusion

Firstly, in this study, we replicated findings from [41; 52] and show that both more advanced activation functions and orthogonal projections allow RNNs to learn dependencies across longer sequences. While designing more advances activation functions as exemplified by the UR-LSTM can improve performance it does not alleviate the problem of vanishing gradients since gradients can still decay towards zero during back-propagation through time, causing instability during training. Models that utilize orthogonal projections (i.e., HiPPO), on the contrary, resolve the gradient problem by ensuring that eigenvalues are |1|, thus stabilizing training.

Secondly, we successfully re-implement a YoloV4 [9] object detector, that shows a slight improvement over existing implementation that utilizes the same backbone. However, utilizing RNNs with more sophisticated activation functions or orthogonal projections to fine-tune bounding box trajectories did not result in desirable performance gains for the video object detection task. While RNNs can scale to reach a performance close to a regular object detector, performance degradations emerge. These degradations were especially severe for RNNs that leverage HiPPO as an update mechanism. It should, however be noted that GPU memory constraints limited the scope of investigation and scalability of HiPPO models. Regardless of that, we conclude that while RNNs, as shown in the UrYolo, can result in a fairly decent object detector that can utilizes temporal bounding box information, performance degradations make the usage of these undesirable. That is, using RNNs to extract temporal information and use them to fine-tune predictions on the object level is not favorable.

Even-though transformers [56] may be another natural choice to model temporal dependencies, a clear investigation of transformers on our object detection task may be difficult due to the quadratic instead of linear memory consumption of transformers compared to the models that were investigated in this study. 3D convolutions [66] a type of convolution that are unfolded across time, while maintaining the spatial relationship between frames may yield more desirable results and

have a lower memory footprint.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015. cite arxiv:1506.02640.

[2] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks.," in *NIPS* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 91–99, 2015.

[3] H. Wu, Y. Chen, N. Wang, and Z. Zhang, "Sequence level semantics aggregation for video object detection," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 9216–9224, IEEE, 2019.

[4] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," 2018.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 00, pp. 580–587, June 2014.

[6] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2016.

[7] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," 2016. cite arxiv:1612.08242.

[8] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018. cite arxiv:1804.02767Comment: Tech Report.

[9] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[10] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022.

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector.," in *ECCV (1)* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), vol. 9905 of *Lecture Notes in Computer Science*, pp. 21–37, Springer, 2016.

[12] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "Dssd : Deconvolutional single shot detector.," *CoRR*, vol. abs/1701.06659, 2017.

[13] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2017.

[14] V. Kalogeiton, V. Ferrari, and C. Schmid, "Analysing domain shift factors between videos and images for object detection," *CoRR*, vol. abs/1501.01186, 2015.

[15] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei, "Flow-guided feature aggregation for video object detection," *CoRR*, vol. abs/1703.10025, 2017.

[16] S. Wang, Y. Zhou, J. Yan, and Z. Deng, "Fully motion-aware network for video object detection," in *ECCV*, 2018.

[17] G. Bertasius, L. Torresani, and J. Shi, "Object detection in video with spatiotemporal sampling networks," *CoRR*, vol. abs/1803.05549, 2018.

[18] F. Xiao and Y. J. Lee, "Spatial-temporal memory networks for video object detection," *CoRR*, vol. abs/1712.06317, 2017.

[19] C. Guo, B. Fan, J. Gu, Q. Zhang, S. Xiang, V. Prinet, and C. Pan, "Progressive sparse local attention for video object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[20] M. Shvets, W. Liu, and A. Berg, "Leveraging long-range temporal relationships between proposals for video object detection," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9755–9763, 2019.

[21] Y. Chen, Y. Cao, H. Hu, and L. Wang, "Memory enhanced global-local aggregation for video object detection," *CoRR*, vol. abs/2003.12063, 2020.

[22] D. Cores, M. Mucientes, and V. M. Brea, "Roi feature propagation for video object detection," in *ECAI 2020*, pp. 2680–2687, IOS Press, 2020.

[23] K. Kang, H. Li, J. Yan, X. Zeng, B. Yang, T. Xiao, C. Zhang, Z. Wang, R. Wang, X. Wang, and W. Ouyang, "T-CNN: tubelets with convolutional neural networks for object detection from videos," *CoRR*, vol. abs/1604.02532, 2016.

[24] K. Kang, W. Ouyang, H. Li, and X. Wang, "Object detection from video tubelets with convolutional neural networks," *CoRR*, vol. abs/1604.04053, 2016.

[25] K. Kang, H. Li, T. Xiao, W. Ouyang, J. Yan, X. Liu, and X. Wang, "Object detection in videos with tubelet proposal networks," *CoRR*, vol. abs/1702.06355, 2017.

[26] W. Han, P. Khorrami, T. L. Paine, P. Ramachandran, M. Babaeizadeh, H. Shi, J. Li, S. Yan, and T. S. Huang, "Seq-nms for video object detection," *CoRR*, vol. abs/1602.08465, 2016.

[27] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Detect to track and track to detect," *CoRR*, vol. abs/1710.03958, 2017.

[28] X. Zhu, J. Dai, X. Zhu, Y. Wei, and L. Yuan, "Towards high performance video object detection for mobiles," *CoRR*, vol. abs/1804.05830, 2018.

[29] T. Gong, K. Chen, X. Wang, Q. Chu, F. Zhu, D. Lin, N. Yu, and H. Feng, "Temporal roi align for video object recognition," *CoRR*, vol. abs/2109.03495, 2021.

[30] Y. Shi, N. Wang, and X. Guo, "Yolov: Making still image object detectors great at video object detection," 2022.

[31] S. Tripathi, Z. C. Lipton, S. J. Belongie, and T. Q. Nguyen, "Context matters: Refining object detection in video with recurrent neural networks," *CoRR*, vol. abs/1607.04648, 2016.

[32] Y. Lu, C. Lu, and C.-K. Tang, "Online video object detection using association lstm," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2363–2371, 2017.

[33] H. Deng, Y. Hua, T. Song, Z. Zhang, Z. Xue, R. Ma, N. Robertson, and H. Guan, "Object guided external memory network for video object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[34] J. Deng, Y. Pan, T. Yao, W. Zhou, H. Li, and T. Mei, "Relation distillation networks for video object detection," *CoRR*, vol. abs/1908.09511, 2019.

[35] A. Mhalla, T. Chateau, and N. Essoukri Ben Amara, "Spatio-temporal object detection by deep learning: Video-interlacing to improve multi-object tracking," *Image and Vision Computing*, vol. 88, pp. 120–131, 2019.

[36] Y. Chen, Y. Cao, H. Hu, and L. Wang, "Memory enhanced global-local aggregation for video object detection," *CoRR*, vol. abs/2003.12063, 2020.

[37] D. Cores, V. M. Brea, and M. Mucientes, "Short-term anchor linking and long-term self-guided attention for video object detection," *Image and Vision Computing*, vol. 110, p. 104179, 2021.

[38] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[40] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[41] A. Gu, C. Gulcehre, T. L. Paine, M. Hoffman, and R. Pascanu, "Improving the gating mechanism of recurrent neural networks," 2020.

[42] D. Gordon, A. Farhadi, and D. Fox, "Re3 : Real-time recurrent regression networks for object tracking," *CoRR*, vol. abs/1705.06368, 2017.

[43] S. Yun and S. Kim, "Recurrent yolo and lstm-based ir single pedestrian tracking," in *2019 19th International Conference on Control, Automation and Systems (ICCAS)*, pp. 94–96, 2019.

[44] A. Toro-Ossaba, J. Jaramillo-Tigreros, J. C. Tejada, A. Peña, A. López-González, and R. A. Castanho, "Lstm recurrent neural network for hand gesture recognition using emg signals," *Applied Sciences*, vol. 12, no. 19, 2022.

[45] J. Cifuentes, P. Boulanger, M. T. Pham, F. Prieto, and R. Moreau, "Gesture classification using lstm recurrent neural networks," in *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 6864–6867, 2019.

[46] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[47] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," 2013.

[48] M. Henaff, A. Szlam, and Y. LeCun, "Orthogonal rnns and long-memory tasks," *CoRR*, vol. abs/1602.06662, 2016.

[49] J. Bayer, "Learning sequence representations," 2015.

[50] C. Tallec and Y. Ollivier, "Can recurrent neural networks warp time?," *CoRR*, vol. abs/1804.11188, 2018.

[51] R. Józefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015* (F. R. Bach and D. M. Blei, eds.), vol. 37 of *JMLR Workshop and Conference Proceedings*, pp. 2342–2350, JMLR.org, 2015.

[52] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré, "Hippo: Recurrent memory with optimal polynomial projections," *CoRR*, vol. abs/2008.07669, 2020.

[53] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *CoRR*, vol. abs/2111.00396, 2021.

[54] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," 2024.

[55] Y. Liu, Y. Tian, Y. Zhao, H. Yu, L. Xie, Y. Wang, Q. Ye, and Y. Liu, "Vmamba: Visual state space model," 2024.

[56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[57] K. Ogata, *Modern Control Engineering*. Prentice Hall, 5 ed., 2010.

[58] L. Ljung, *System Identification: Theory for the User*. Prentice Hall, 2 ed., 1999.

[59] K. J. Astrom and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. USA: Princeton University Press, 2008.

[60] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[61] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[62] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," 2016.

[63] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[64] K. He, X. Zhang, S. Ren, and J. Sun, *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*, p. 346–361. Springer International Publishing, 2014.

[65] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8759–8768, 2018.

[66] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "C3D: generic features for video analysis," *CoRR*, vol. abs/1412.0767, 2014.

VIII Appendix

VIII.i Orthogonality of Legendre Polynomials

Theorem 1. *Let $P_n(x)$ and $P_m(x)$ be Legendre polynomials of degree n and m respectively, where $n \neq m$. Then, the Legendre polynomials are orthogonal with respect to the inner product on the interval $[-1, 1]$, if:*

$$\int_{-1}^{1} P_n(x)P_m(x)\,dx = 0 \quad \textit{for} \quad n \neq m.$$

Proof. Using Rodrigues' formulation of the Legendre Polynomial $P_n x$:

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right], \tag{58}$$

and considering it over an integral over the interval $[-1, 1]$:

$$\int_{-1}^{1} f(x)P_n(x)dx, \tag{59}$$

then substituting $P_n(x)$ with Rodrigues' formulation from Eq. 58 into Eq. 59 we have:

$$\begin{aligned}&\int_{-1}^{1} f(x)P_n(x)dx\\ =&\int_{-1}^{1} f(x)\frac{1}{2^n n!}\frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right]dx\end{aligned} \tag{60}$$

Performing integration by parts n times using:

$$\int udv = uv - \int vdu, \tag{61}$$

and defining u and dv as:

$$u = f(x) \tag{62}$$

$$dv = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right]dx, \tag{63}$$

we define $g(x)$ so that:

$$g(x) = \left(x^2-1\right)^n. \tag{64}$$

Plugging $g(x)$ from Eq. (64) into dv in Eq. (63):

$$dv = \frac{1}{2^n n!}\frac{d^n g(x)}{dx^n}dx, \tag{65}$$

and integrating dv w.r.t. x:

$$v = \int \frac{1}{2^n n!}\frac{d^n g(x)}{dx^n}dx, \tag{66}$$

where the anti-derivative of the derivative is:

$$v = \frac{1}{2^n n!}g(x) = \frac{1}{2^n n!}\left(x^2-1\right)^n \tag{67}$$

Finally, applying integration by parts:

$$\int_{-1}^{1} f(x)\frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right]dx \tag{68}$$

$$= \left[f(x)\frac{1}{2^n n!}\left(x^2-1\right)^n\right]_{-1}^{1} - \int_{-1}^{1}\frac{d}{dx}f(x), \tag{69}$$

where the boundary terms evaluate to zero since $(x^2-1)^2$ is zero at $x = \pm 1$. The integral then reduces to:

$$-\int_{-1}^{1} \frac{d}{dx} f(x) \cdot \frac{1}{2^n n!} \left(x^2-1\right)^n dx. \tag{70}$$

Integrating n times, we get:

$$(-1)^n \int_{-1}^{1} f^{(n)}(x) \cdot \frac{1}{2^n n!} \left(x^2-1\right)^n dx, \tag{71}$$

reaching the following relationship, in which we have reduced the integral to a form involving the n-th order derivative of $f(x)$:

$$\int_{-1}^{1} f(x) P_n(x) dx = \frac{(-1)^n}{2^n n!} \int_{-1}^{1} f^{(n)}(x) \left(x^2-1\right)^n dx. \tag{72}$$

If $f(x) = P_m(x)$ is some polynomial of degree m where m is some integer such that $0 \leq m < n$, we have:

$$f^{(n)}(x) = \frac{d^n}{dx^n}[P_m(x)] = 0 \tag{73}$$

■

VIII.ii Eigenvalues of Orthogonal Matrices

Theorem 2. *Let A be an $n \times n$ orthogonal matrix. Then, every eigenvalue λ of A has an absolute value of 1, i.e., $|\lambda| = 1$, if if $A^T A = I$, where A^T is the transpose of A and I is the $n \times n$ identity matrix.*

Proof. Suppose λ is an eigenvalue of A and v is a corresponding eigenvector. Then:

$$Av = \lambda v \tag{74}$$

The orthogonality condition $A^T A = I$ implies that A preserves the Euclidean norm of vectors. Specifically, for any vector v:

$$\|Av\| = \|v\| \tag{75}$$

Consider the norm of Av:

$$\|Av\|^2 = (Av)^T (Av) = (\lambda v)^T (\lambda v) = \lambda^2 v^T v = \lambda^2 \|v\|^2 \tag{76}$$

Since A is orthogonal, we also have:

$$\|Av\|^2 = \|v\|^2 \tag{77}$$

From the above two equations, we equate the norms:

$$\lambda^2 \|v\|^2 = \|v\|^2 \tag{78}$$

Since v is an eigenvector, it is non-zero, so $\|v\|^2 \neq 0$. Therefore, we can divide both sides by $\|v\|^2$:

$$\lambda^2 = 1 \tag{79}$$

Taking the square root of both sides, we find:

$$|\lambda| = 1 \tag{80}$$

As a result, we show that any eigenvalue λ of an orthogonal matrix A satisfies $|\lambda| = 1$.

■

VIII.iii Code Repository and Demonstrations

The corresponding code repository with demonstrations can be found at: yolov4-real-time-obj-detection.